SIEMENS

SIMATIC NET

PROFINET
IO-Base 用户编程接口

编程手册

10/2015

C79000-G8952-C195-18

法律资讯

警告提示系统

为了您的人身安全以及避免财产损失，必须注意本手册中的提示。人身安全的提示用一个警告三角表示，仅与财产损失有关的提示不带警告三角。警告提示根据危险等级由高到低如下表示。

⚠危险
表示如果不采取相应的小心措施，**将会**导致死亡或者严重的人身伤害。

⚠警告
表示如果不采取相应的小心措施，**可能**导致死亡或者严重的人身伤害。

⚠小心
表示如果不采取相应的小心措施，可能导致轻微的人身伤害。

注意
表示如果不采取相应的小心措施，可能导致财产损失。

当出现多个危险等级的情况下，每次总是使用最高等级的警告提示。如果在某个警告提示中带有警告可能导致人身伤害的警告三角，则可能在该警告提示中另外还附带有可能导致财产损失的警告。

合格的专业人员

本文件所属的产品/系统只允许由符合各项工作要求的**合格人员**进行操作。其操作必须遵照各自附带的文件说明，特别是其中的安全及警告提示。
由于具备相关培训及经验，合格人员可以察觉本产品/系统的风险，并避免可能的危险。

按规定使用Siemens 产品

请注意下列说明：

⚠警告
Siemens 产品只允许用于目录和相关技术文件中规定的使用情况。如果要使用其他公司的产品和组件，必须得到 Siemens 推荐和允许。正确的运输、储存、组装、装配、安装、调试、操作和维护是产品安全、正常运行的前提。必须保证允许的环境条件。必须注意相关文件中的提示。

商标

所有带有标记符号 ® 的都是西门子股份有限公司的注册商标。本印刷品中的其他符号可能是一些其他商标。若第三方出于自身目的使用这些商标，将侵害其所有者的权利。

责任免除

我们已对印刷品中所述内容与硬件和软件的一致性作过检查。然而不排除存在偏差的可能性，因此我们不保证印刷品中所述内容与硬件和软件完全一致。印刷品中的数据都按规定经过检测，必要的修正值包含在下一版本中。

Siemens AG
Division Process Industries and Drives
Postfach 48 48
90026 NÜRNBERG
德国

文件订购号: C79000-G8952-C195
℗ 10/2015 本公司保留更改的权利

前言

SIMATIC NET – IO-Base 用户编程接口

本手册将为您使用 C/C++ 编程语言编写用户程序奠定坚实的基础。

您是初学者？利用本手册可系统性地熟悉相关内容。可首先阅读 IO-Base 用户编程接口概述。您将从中了解有关该接口的设计原理和系列功能的所有必知信息。

您是专业的编程人员？利用本手册可立即开始工作。借助快速入门中的步骤序列和全面的参考信息，您将了解创建 IO-Base 用户程序的最佳途径。

发现示例很实用？可灵活利用所提供的示例程序将您自己的想法付诸实践。

本手册的用途

若要通过 IO-Base 用户编程接口实现一个 IO 控制器或一个 IO 设备，请使用本手册。

较之先前版本的更改

以太网接口数据类型与函数的说明

文档概述

我们建议您先熟悉以下文档，然后再创建 IO-Base 用户程序：

文档名称	阅读原因
手册 PROFINET 系统描述	提供 PROFINET IO 相关主题的基本知识： 网络组件、数据交换和通信、PROFINET IO、基于组件的自动化、PROFINET IO 和基于组件的自动化的应用示例。
使用入门 PROFINET IO	使用入门通过具体示例逐步引导您完成调试过程直至功能性应用。
手册 从 PROFIBUS DP 到 PROFINET IO	如果已安装 PROFIBUS 系统并要将其转换为 PROFINET 系统，请阅读此文档。

文档名称	阅读原因
自述文件 • 包含在“SIMATIC NET PC 软件”DVD 中 • 包含在“DK-16xx PN IO”CD 中	您将在其中了解到有关 SIMATIC NET PC 软件产品的最新信息。
安装手册 包含在“SIMATIC NET PC 软件”DVD 中	该手册将逐步引导您在 PC 上安装 SIMATIC NET 产品（仅限 SOFTNET IE PN IO）。
手册 调试 PC 站	本手册提供有关将 PC 调试并组态为 PROFINET IO 控制器的所需信息。
手册 IO-Base 用户编程接口	创建 IO-Base 控制器或 IO-Base 设备用户程序时的参考手册。
手册 PG/PC 的工业通信	该手册为您介绍工业通信，并描述可用的通信协议。
手册 SIMATIC NET - 双绞线和光纤网络	借助此文档组态并设置工业以太网。

所需的基本经验

我们建议用户程序编写人员具有以下经验：

- C/C++ 编程经验
- 编程技术：
  - 多线程技术
  - 回调例程
- 英文技术术语知识
- PROFINET IO 系统知识
- 自动化工程领域的基本经验
- STEP 7 Professional (TIA Portal) 或 NCM PC 组态工具的基本经验

手册适用范围

手册介绍了作为多款产品组成部分的 IO-Base 用户编程接口：

- CP 1616
- CP 1604
- SOFTNET PN IO
- DK-16xx PN IO
- PN 驱动程序

认证

本文档所列产品和系统采用了符合 DIN ISO 9001
质量管理系统进行生产和销售，并且均通过 DQS 认证。在所有 IQNet
国家/地区都承认该 DQS 证书（证书登记号 2613）。

参见

(http://support.automation.siemens.com/)

商标

下文的一些名称以及可能的其它名称不带注册商标符号 ®，它们均为 Siemens AG 的注册商标：

SIMATIC NET, HARDNET, SOFTNET, CP 1612, CP 1613, CP 5611, CP 5612, CP 5613, CP 5614, CP 5622

服务与支持

除了产品文档外，Siemens
自动化客户支持还具有丰富全面的在线信息平台，从而为您提供随时随地的支持。
可在以下 Internet 地址找到服务与支持页面：

(http://support.automation.siemens.com/WW/llisapi.dll?func=cslib.csinfo2&aktprim=99&lang=en)

除新闻外，您还将找到以下信息：

- 产品信息、产品支持、应用程序和工具
- 技术论坛
- 技术支持 - 咨询 Siemens 专家
- 我们的服务提供：
  - 技术咨询、工程支持
  - 现场服务
  - 备件和维修
  - 维护、优化、现代化等

有关联系数据，请访问以下 Internet 网址：
(http://www.automation.siemens.com/partner/guiwelcome.asp?lang=en)

参见

(https://support.automation.siemens.com/WW/llisapi.dll?func=cslib.csinfo&lang=en&objid=38718979&caller=view)

SITRAIN - Siemens 自动化和工业解决方案培训

通过 300 多套不同的课程，SITRAIN 涵盖了 Siemens
在自动化和驱动技术领域内的整个产品和系统范围。
除了基本课程外，我们还提供专为满足个人需求而量身定制的培训以及一系列不同的教学媒体和方案，例如 CD-ROM 或 Internet 上的自学课程。

有关培训课程以及如何联系客户顾问的详细信息，请访问以下 Internet 网址：

(www.siemens.com/sitrain)

安全提示：

西门子为其产品及解决方案提供工业安全功能，以支持工厂、解决方案、机器、设备和/或网络的安全运行。 这些功能是整个工业安全机制的重要组成部分。
有鉴于此，西门子不断对产品和解决方案进行开发和完善。
西门子强烈建议您定期了解产品更新和升级信息。

此外，要确保西门子产品和解决方案的安全操作，还须采取适当的预防措施（例如：设备单元保护机制），并将每个组件纳入先进且全面的工业安全保护机制中。

可能使用的所有第三方产品须一并考虑。 更多有关工业安全的信息，请访问 http://www.siemens.com/industrialsecurity。

要及时了解有关产品的更新和升级信息，请订阅相关产品的时事通讯。
更多相关信息，请访问 http://support.automation.siemens.com。

SIMATIC NET 词汇表

在 SIMATIC NET 词汇表部分针对本文档中所用的专业术语进行了解释。

用户可在以下位置找到 SIMATIC NET 词汇表：

- SIMATIC NET 手册集

  该 DVD 随一些 SIMATIC NET 产品一起提供。

- Internet 上的以下条目 ID：

  50305045 (http://support.automation.siemens.com/WW/view/zh/50305045)

目录

IO 控制器功能快速入门

1

本章介绍了用 C/C++ 编程语言基于 IO-Base
用户编程接口创建用户程序的推荐的、详细操作过程。

首先要熟悉有关 PROFINET IO 的基本知识。 逐渐完成各个步骤，并最终编写您自己的 IO-Base 控制器用户程序。

1.1 步骤

说明

按照以下步骤，可快速高效地创建 IO-Base 控制器用户程序：

步骤	说明
1	了解有关 PROFINET 的基本知识。 可通过阅读《PROFINET 系统说明》手册来进行了解。
2	熟悉 IO-Base 用户编程接口的基本特性。 可通过阅读本手册的“IO 控制器的 IO-Base 用户编程接口概述 (页 23)”部分来进行熟悉。
3	熟悉以下文件，了解已经为以下步骤中提供了哪些支持，以及如何利用这些支持。 这些文件如下： • 包含附加信息和最新修改说明的自述文件 • IO-Base 用户编程接口的 C 头文件： – 包含函数和数据结构的“pniousrx.h”。 – 包含错误代码与返回代码的“pnioerrx.h” • 用于连接 IO-Base 控制器用户程序的“pniousrx.lib”或“libpniousr”库。 • 示例程序和相应的组态。
4	熟悉“..\Examples\easy”或“../examples/easy”子文件夹下示例程序“PnioEasy”的源文本，并在章节IO 控制器数据类型与函数的说明 (页 41)中了解相关函数和数据结构。

步骤	说明
5	根据系统组态，了解需要发送和接收哪些数据（IO 数据、数据记录、报警），以及涉及哪些 IO 设备。 阅读所用 IO 设备的用户文档。
6	完成 STEP 7 或 NCM PC 组态并将其下载到相关设备中。
7	修改提供的示例程序“PnioEasy”，使其可在您的系统中运行。 通过这种方式，您可以了解相关的重要技术，而无需再自行开发。 编译并加入修改后的示例程序，在您准备好的系统上对其进行测试。
8	现在创建您自己的包含全部功能的 IO-Base 控制器用户程序。

1.2 产品功能概述

说明

下表概要介绍了可在各 SIMATIC NET 产品中应用的 IO 控制器功能。

函数组和名称	产品		
	SOFTNET PN IO (RT)	CP 1616/CP 1604 (RT + IRT) 带有版本 V2.0 及更高版本的 DK-16xx	PN Driver
管理函数			
PNIO_controller_open()	X	X	X
PNIO_register_cbf()	X	X	X
PNIO_controller_close()	X	X	X
PNIO_interface_open()	X	X	X
PNIO_interface_register_cbf()	X	X	X
PNIO_interface_close()	X	X	X
PNIO_interface_set_ip_and_nos()	X	X	X
与模式相关的函数			
PNIO_set_mode()	X	X	X

函数组和名称	产品		
回调事件 PNIO_CBE_MODE_IND	X	X	X
PNIO_device_activate()	X	X	X
回调事件 PNIO_CBE_DEV_ACT_CONF	X	X	X
用于读取 IO 数据的函数			
PNIO_data_read()	X	X	X
PNIO_output_data_read()	–	X	-
PNIO_data_read_cache_refresh()	–	X	-
PNIO_data_read_cache()	–	X	-
用于写入 IO 数据的函数			
PNIO_data_write()	X	X	X
PNIO_data_write_cache()	–	X	-
PNIO_data_write_cache_flush()	–	X	-
用于读取数据记录的接口			
PNIO_rec_read_req()	X	X	X
回调事件 PNIO_CBE_REC_READ_CONF	X	X	X
PNIO_interface_rec_read_req()	X	X	X
PNIO_CBE_IFC_REC_READ_CONF	X	X	X
用于写入数据记录的接口			
PNIO_rec_write_req( )	X	X	X
回调事件 PNIO_CBE_REC_WRITE_CONF	X	X	X
报警接口			
回调事件 PNIO_CBE_ALARM_IND	X	X	X
PNIO_CBE_IFC_ALARM_IND	X	X	X
诊断接口			
PNIO_ctrl_diag_req( )	–	X	X
回调事件 PNIO_CBE_CTRL_DIAG_CONF	–	X	-
用 LED 发出指示的站			
回调事件 PNIO_CBE_START_LED_FLASH_IND	–	X	-
回调事件 PNIO_CBE_STOP_LED_FLASH_IND	–	X	-

函数组和名称	产品		
常规			
回调事件 PNIO_CBE_CP_STOP_REQ	–	X	-
等时实时模式 (IRT) 接口			
PNIO_CP_register_cbf()	–	X	-
回调事件 PNIO_CP_CBE_STARTOP_IND	–	X	-
回调事件 PNIO_CP_CBE_OPFAULT_IND	–	X	-
PNIO_CP_set_opdone()	–	X	-
PROFIenergy (PE) 接口			
PNIO_register_pe_cbf()	X	X	-
PNIO_pe_cmd_req()	X	X	-
产品特有的用于 PN 驱动程序的函数			
SERV_CP_init	–	–	X
SERV_CP_undo_init	–	–	X
SERV_CP_get_network_adapters	–	–	X
SERV_CP_startup	–	–	X
SERV_CP_shutdown	–	–	X
SERV_CP_set_trace_level	–	–	X

说明

只有使用通信处理器 CP 1604 和 CP 1616 才支持对模块和子模块进行部分访问。

实时模式概述

2

下文概要介绍了各种实时模式以及相关的数据通信。

IO-Base 用户编程接口支持以下模式：

- RT（Real Time，实时）
- IRT（等时实时）

2.1 实时和等时实时模式

概述

PROFINET 标准区分了实时 (RT) 和等时实时 (IRT) 模式。

RT 模式

在实时模式 (RT) 下，实时帧（RT 数据）按高优先级传送。

IRT 模式

在等时模式（IRT
模式）下，在固定的时间段内以高优先级周期性传送（确定的间隔）实时帧。

IRT

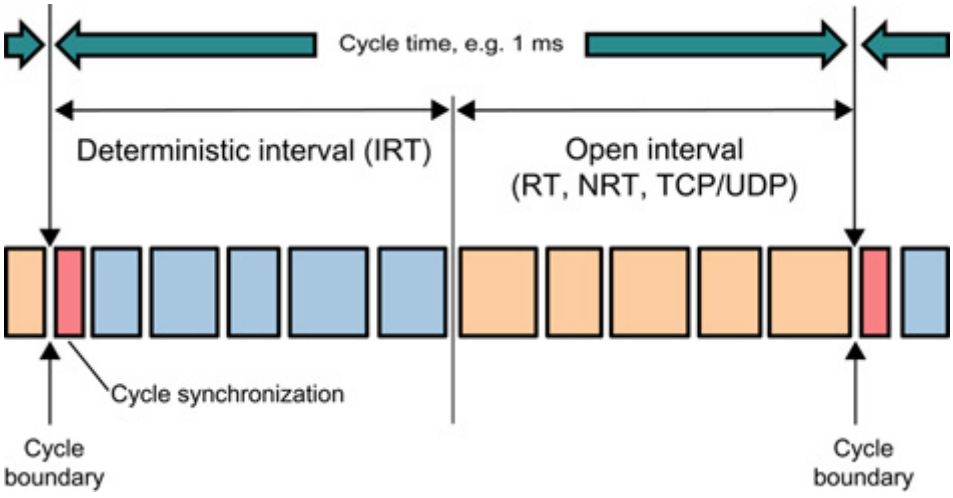
传送间隔结束信息将通知给 IO-Base 用户程序。IO-Base
用户程序借助该通知信号与总线时钟同步。在传送间隔内，将传送 RT、NRT（Non Real Time，非实时）和 TCP/UDP 帧。

如果 IO-Base 用户程序仅在确定的间隔外访问 IRT 数据，即称为等时实时模式。

如果可随时访问 IRT 数据，说明 IO-Base 用户程序处于非等时模式。

IO-Base
用户编程接口提供需要在初始化、运行和终止阶段调用的其它函数。在组态中指定时间和间隔，并将此信息传递给 IO 控制器、IO 设备和交换机。

具体拓扑必须另行规划。


Cycle time, e.g. 1 ms
Deterministic interval (IRT)
Open interval
(RT, NRT, TCP/UDP)
Cycle synchronization
Cycle
boundary
Cycle
boundary

IO 控制器的 IO-Base 用户编程接口概述

3

本章介绍了 IO-Base 控制器用户编程接口的基本特性，可为您创建自己的 IO-Base 用户程序奠定基础。

有关函数调用和数据访问在“IO 控制器数据类型与函数的说明 (页 41)”部分详细介绍。

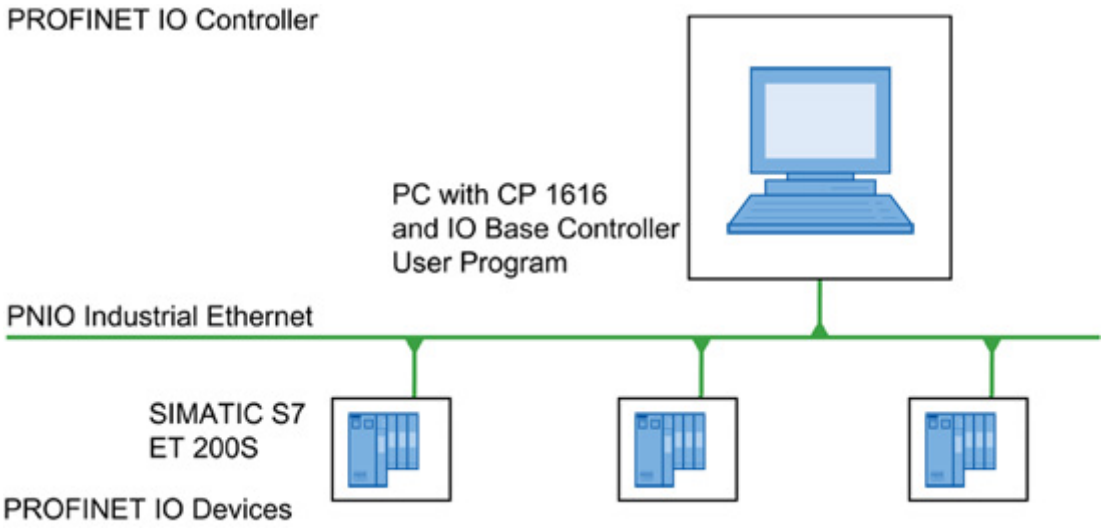
3.1 IO 控制器 IO-Base 用户编程接口的典型应用

应用示例说明

下图说明了一个典型应用：
一台 PC 通过工业以太网与多台 PROFINET IO 设备通信。

IO-Base 控制器用户程序和 SIMATIC NET 产品的各 IO-Base 函数均在 PC 上运行。通过工业以太网上的一个 PROFINET 通信处理器或一个标准以太网适配器与多个 PROFINET IO 设备（例如 ET 200S）进行数据交换。

此组态将在提供的示例程序中再次提及。

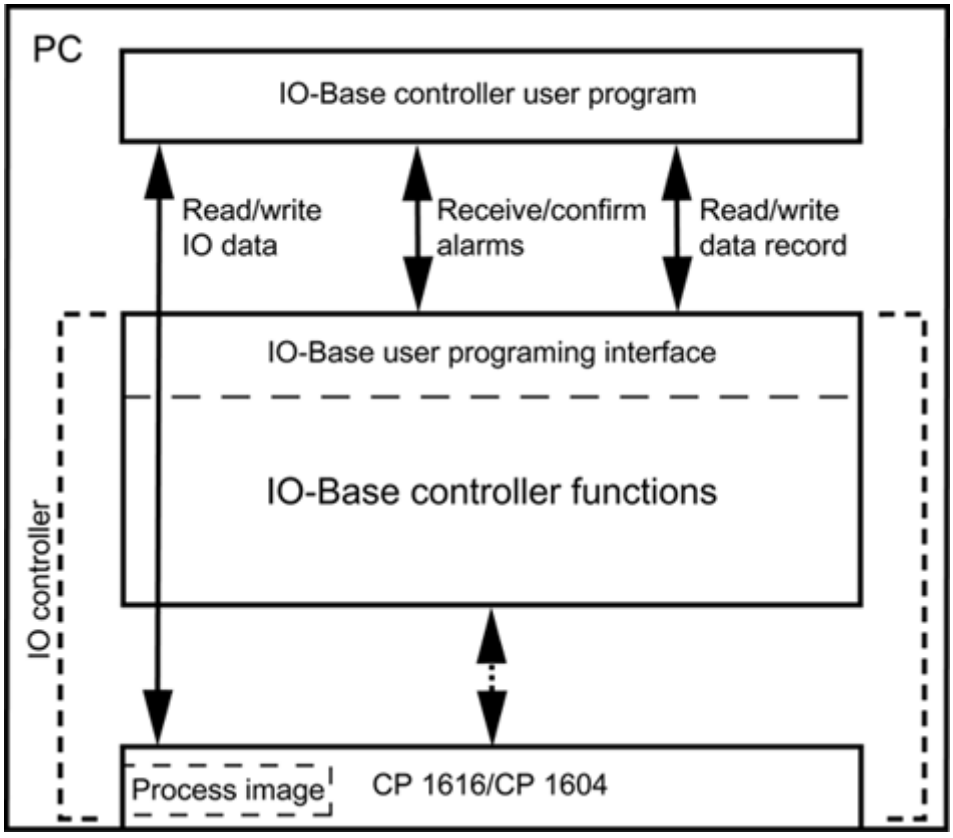


3.2 PC 内的软件架构

包含 CP 1616/CP 1604 的 PROFINET I/O 架构说明

PROFINET IO 借助 IO-Base 用户编程接口提供 C 用户程序与 PROFINET IO 设备通信所需的所有函数。

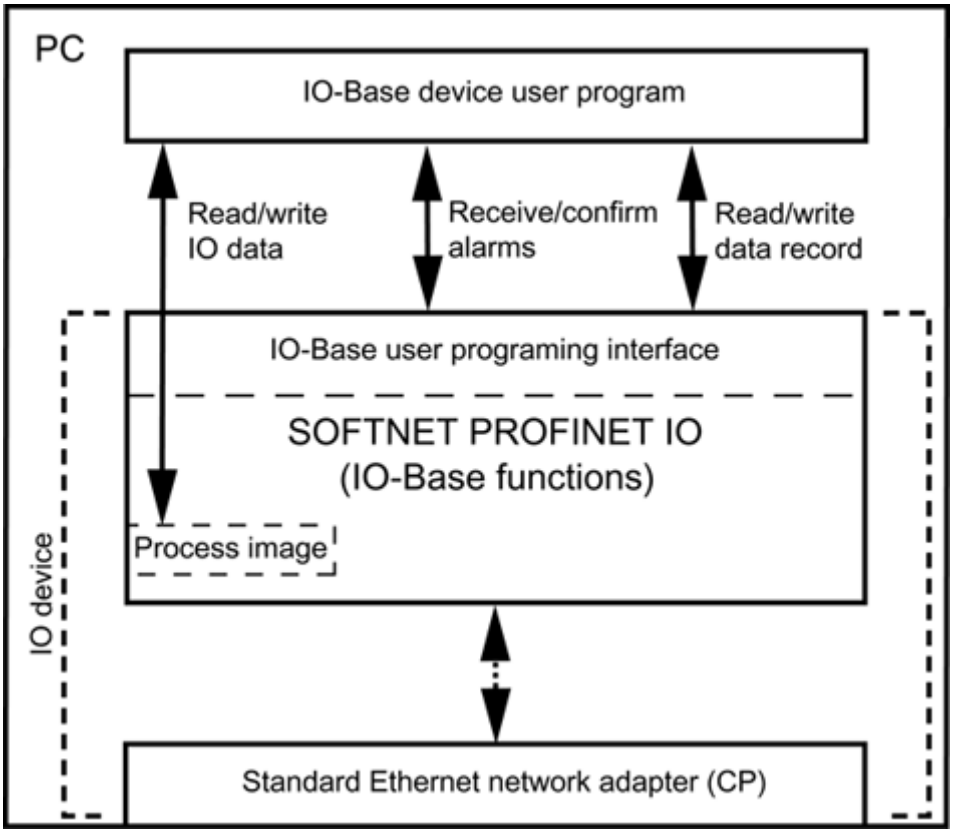
其中包括用于读/写 IO 数据、发送报警与接收报警确认以及读/写数据记录的 IO-Base 函数。 PROFINET IO 软件通过 PROFINET IO 通信处理器连接到过程。 通过 STEP 7 或 NCM PC 组态与 IO 控制器的通信。



SOFTNET PROFINET IO 架构说明

SOFTNET PROFINET IO 借助 IO-Base 用户编程接口提供 C 用户程序与 PROFINET IO 设备通信所需的所有函数。

其中包括用于读/写 IO 数据、接收/确认报警以及读/写数据记录的 IO-Base 函数。在过程方向上，SOFTNET PROFINET IO 软件通过标准以太网适配器（CP - 通信处理器）连接到工业以太网。 通过 STEP 7 或 NCM PC 软件组态与 I/O 设备的连接。



DLL、头文件、导入库和示例程序

IO-Base 编程接口是由一个文件构成的 C 编程接口。 在程序安装过程中安装该接口。

要使用 IO-Base 编程接口，需要以下文件：

文件类型	文件名	目的
头文件	pniobase.h	包含 IO-Base 用户编程接口的全局结构与定义。
头文件	pniousrx.h	包含 IO-Base 用户编程接口用于实现 IO 控制器功能的数据类型、常量和函数声明。

文件类型	文件名	目的
头文件	pnioerrx.h	包含错误代码
库	pniousrx.lib 或 libpniousr	用于连接 IO-Base 控制器用户程序的 IO-Base 编程接口。

3.3 IO 控制器中 IO-Base 控制器用户程序的典型顺序

概述

一个典型的 IO-Base 控制器用户程序可划分为 3 个阶段。

- 初始化阶段
- 生产运行阶段
- 完成阶段

下面将对这几个阶段进行详细介绍。

3.3.1 初始化阶段

说明

初始化阶段划分为五个步骤：

步骤	操作	目的
1	PNIO_controller_open( )	• 在 IO-Base 用户编程接口中注册 IO 控制器。 • 注册用于数据记录和报警的回调函数。
2	PNIO_register_cbf( ) 采用参数 PNIO_CBE_MODE_IND	注册模式更改回调函数。

步骤	操作	目的
3	PNIO_CP_register_cbf( )	如需支持 IRT 组态，因此支持等时实时模式，则必须注册回调函数 PNIO_CP_CBE_OPFAULT_IND( ) 和 PNIO_CP_CBE_STARTOP_IND( )。之后便会通知这些回调函数。
4	PNIO_set_mode( ) 采用参数 PNIO_MODE_OPERATE	将控制器设置为 OPERATE 模式。
5	等待回调事件 PNIO_CBE_MODE_IND 第二次发生。	实现的模式应为 OPERATE，否则便是有错误存在。

3.3.2 生产运行阶段

概述

在生产运行阶段与 IO 设备交换数据。 具体数据如下：

- 读/写 IO 数据
- 接收和确认报警
- 读/写数据记录

数据交换将在下文详细介绍。

读/写 IO 数据

通过以下函数可读取和写入过程数据：

- PNIO_data_read( )
- PNIO_output_data_read( )
- PNIO_data_read_cache_refresh( )
- PNIO_data_read_cache( )
- PNIO_data_write( )

- PNIO_data_write_cache( )
- PNIO_data_write_cache_flush( )

访问 PC 上 PROFINET IO 的 IO 数据。

IO-Base 接口按组态中指定的方式与 IO 设备进行周期性通信。

有关更多信息，请参见“非等时访问周期性 IO 数据 (RT) (页 30)”部分。

接收和确认报警

接收到报警（事件 PNIO_CBE_ALARM_IND）将引起对相应回调函数的调用。

有关更多信息，请参见“回调机制 (页 38)”部分。

读/写数据记录

使用以下函数可通过访问 IO 设备读取或写入数据记录，例如参数分配：

- PNIO_rec_read_req( )
- PNIO_rec_write_req( )

请求的数据记录或对已发送数据的确认将通过回调机制提供。

有关更多信息，请参见“回调机制 (页 38)”部分。

读数据记录作业的顺序

步骤	操作	目的
1	PNIO_rec_read_req( )	发送读数据记录作业。
2	回调函数（事件 PNIO_CBE_REC_READ_CONF）	指示是否已确认读数据记录作业和已收到数据记录。 数据记录指针将附加在此回调函数上，因而可供用户程序使用。

写数据记录作业的顺序

步骤	操作	目的
1	PNIO_rec_write_req( )	发送写数据记录作业。
2	回调函数（事件 PNIO_CBE_REC_WRITE_CONF）	评估确认情况。

3.3.3 完成阶段

说明

IO 控制器的完成阶段包含三个步骤：

步骤	操作	目的
1	PNIO_set_mode( ) (OFFLINE)	将 IO 控制器设置为 OFFLINE 模式。
2	等到变为 OFFLINE 模式。	等到回调函数（事件 PNIO_CBE_MODE_IND）通知 PNIO_MODE_OFFLINE 模式。 此后，将停止与 IO 设备通信。
3	PNIO_controller_close( )	在 IO-Base 用户编程接口中取消注册 IO 控制器。

3.4 IO-Base 函数的基本数据交换

说明

IO-Base 函数具有三种基本的数据交换方式：

- 周期性非等时实时 IO 数据通信 (RT)
  - 写入 RT IO 数据
  - 读取 RT IO 数据

  IO 数据交换也包含状态信息。

  此特殊功能将在下一节介绍。
- 周期性等时实时 IO 数据通信 (IRT)
  这些访问函数与上文介绍的访问函数相同。 该访问必须仅与 IRT 数据相关，且必须在 PNIO_CP_CBE_STARTOP_IND 回调事件与 PNIO_CP_set_opdone( ) 函数调用之间进行。
- 非周期性 IO 数据交换
  - 写入和读取数据记录
  - 接收报警

  有关更多信息，请参见“回调机制 (页 38)”部分。

3.5 非等时访问周期性 IO 数据 (RT)

非等时访问周期性 IO 数据 (RT)

对于周期性数据通信，共有两种访问方式：

- 无缓冲直接访问过程映像
- 有缓冲快速访问过程映像

无缓冲直接访问过程映像

采用无缓冲直接访问过程映像方式时，总是将 IO 数据直接写入 CP 的过程映像存储器中。

无缓冲直接访问过程映像通过以下函数实现：

函数	说明
PNIO_data_read( )	立即从过程映像中读取输入数据
PNIO_output_data_read( )	立即从过程映像中读取输出数据
PNIO_data_write( )	立即向过程映像中写入输出数据

只要其中一个函数被调用，数据就会立即传送到过程映像。

说明

在一个周期内仅有少量输出数据有变化时，应始终采用其中一种访问类型。

有缓冲快速访问过程映像

采用有缓冲快速访问过程映像方式时，将始终通过主计算机上的缓冲进行访问。 在使用 refresh 和 flush 函数时，始终会传送整个过程映像。
由于是完全访问，因此要采用传送机制来提高速度。

有缓冲快速访问过程映像通过以下函数实现：

对于读取

读操作涉及两个步骤：

步骤	函数	说明
1	PNIO_data_read_cache_refresh( )	将输出数据的远程状态和所有输入数据从过程映像传递到主计算机的读取缓存。
2	PNIO_data_read_cache( )	从主计算机的读取缓存读取每个子模块的各个输入数据。

对于写入

写操作涉及两个步骤：

步骤	函数	说明
1	PNIO_data_write_cache( )	将每个子模块的各输出数据写入到主计算机的写入缓存。
2	PNIO_data_write_cache_flush( )	将输入数据的本地状态和所有输出数据从主计算机的写入缓存传递到过程映像。

说明

SOFTNET PN IO 不支持缓冲访问。

说明

为了在使用缓存时实现最佳的速度效果，请注意以下说明：

- 在对设备进行项目工程设计时，首先安排带有输入数据的子模块，然后再安排带有输出数据的子模块。
- 组态数据长度大的少数子模块，而非数据量很小的许多子模块。
- 避免子模块同时包含输入和输出数据。
- 首先读取输入数据，然后再写入输出数据。

到 IO 设备的数据传送

在执行写入和读取时，IO-Base 函数都将访问 PROFINET IO 的过程映像，而非 IO 设备本身。

过程映像与 IO 设备之间的数据交换将通过底层 IO-Base 函数周期性自动进行处理。此数据交换的详细内容在组态中指定。

说明

IO-Base 控制器用户程序不需要周期性地写入或读取 RT IO 数据。

说明

让 IO-Base 控制器用户程序比组态的周期更频繁地访问过程映像没有什么实际意义。

IO 数据和数据状态

IO 数据的质量通过数据状态的 GOOD 或 BAD 这两个值来描述。

无论是写入还是读取时，都将交换以下两个数据状态：

- 本地状态（IO-Base 控制器用户程序的状态）
- 远程状态（通信伙伴的状态）

3.5.1 状态的周期性写入

PNIO_data_write( ) 和 PNIO_data_write_cache( ) 函数的顺序

借助 PNIO_data_write( ) 和 PNIO_data_write_cache( )
函数，不但可写入通信伙伴的输出数据，还可写入这些数据相应的本地状态。
同时还会从通信伙伴读取此输出数据的远程状态。
因此在周期性写入期间共涉及两个状态。

通信方向	值
到通信伙伴	• 输出数据 • 本地状态
来自通信伙伴	远程状态

本地状态

通常情况下，IO-Base 控制器用户程序会将本地状态设置为 GOOD。

如果输出数据不正确或无效，IO-Base 控制器用户程序则将本地状态设置为 BAD。

在这种情况下，通信伙伴便可输出所组态的替换值。

说明

如果存在设备故障，则 IO-Base 库将自动将数据状态设置为“BAD”。
用户在收到恢复报警时，必须将 IO 数据初始化为“GOOD”状态。
对于没有数据的插槽（例如前端模块，以及 IRT 端口），IO-Base 库将自动将其状态设置为“GOOD”。

通信伙伴的远程状态

通信伙伴使用远程状态传递其是否能够成功处理过程数据或者是否存在问题的信息。

此状态与在上一周期中从同一模块传送的输出数据相关，而不是与刚刚启动的写入作业相关。

3.5.2 状态的周期性读取

PNIO_data_read( ) 和 PNIO_data_read_cache( ) 函数的顺序

PNIO_data_read( ) 和 PNIO_data_read_cache( ) 函数不但会从通信伙伴读取输入数据，还将读取相应的远程数据状态。

输入数据的本地状态也将发送给通信伙伴。

因此在周期性读取期间共涉及两个状态。

通信方向	值
来自通信伙伴	• 输入数据 • 远程状态
到通信伙伴	本地状态

通信伙伴的远程状态

通信伙伴利用远程状态报告输入数据的质量（GOOD 或 BAD）。

如果通信伙伴报告 BAD，IO-Base 控制器用户程序可使用替换值继续处理。

说明

如果存在设备故障，则 IO-Base 库将自动将数据状态设置为“BAD”。
用户在收到恢复报警时，必须将 IO 数据初始化为“GOOD”状态。
对于没有数据的插槽（例如前端模块，以及 IRT 端口），IO-Base 库将自动将其状态设置为“GOOD”。

本地状态

通常情况下，IO-Base 控制器用户程序会将本地状态设置为 GOOD。

但是，如果 IO-Base 控制器用户程序无法处理返回的数据，则可能会将读作业的本地状态设置为 BAD。

通信伙伴在接到此状态后，便可识别出其发送的数据是否已被成功处理。

3.6 等时实时和非等时实时访问周期性 IO 数据 (IRT)

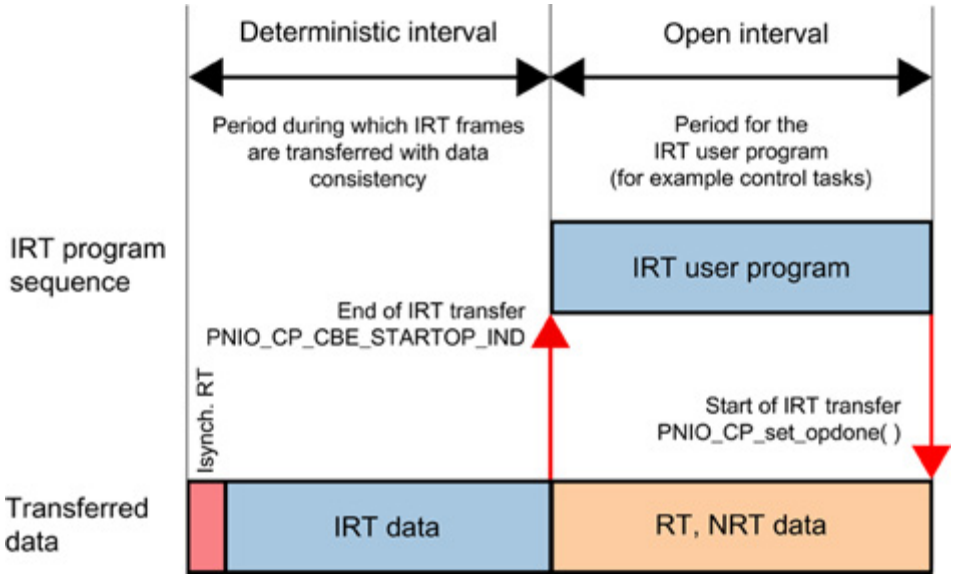
说明

在等时实时模式下，IRT 用户程序在两个周期性重复的 IRT 时间点之间处理和写入 IRT 数据。

- IRT 传送结束
- IRT 传送开始

在到达 IRT 传送结束时，DMA 将 IRT 输入数据从过程映像传送到主机存储器中，并且 PNIO_CP_CBE_STARTOP_IND 回调事件将此情况通知 IRT 用户程序。 此后，所有 IRT 数据便处于主机存储器中，可开始数据处理过程。 在 PNIO_CP_register_cbf( ) 函数调用后，将报告首批 PNIO_CP_CBE_STARTOP_IND 回调事件的情况。

在 IRT 用户程序完成数据处理后，它会调用 PNIO_CP_set_opdone( ) 函数。
该函数会使用 DMA 将 IRT 输出数据复制到过程映像存储器中，并通知通信处理器数据是否有效，以便可在下一个总线循环传送这些数据。



违背等时实时模式

如果 IRT 数据处理完毕的信号传递得过晚，则 DMA 传送无法及时完成，并且 IRT 输出数据无法标记为有效。

从而在下一个总线循环中，通信处理器会将 IRT 数据作为无效数据传送。
用户程序将通过 PNIO_CP_CBE_OPFAULT_IND 回调事件获知此情况。

等时实时 IO 数据（IRT 数据）的一致性

为了给用户程序更多的数据处理时间，通过 DMA
在主机存储器和过程映像存储器之间传送 IRT 数据。 IRT 数据因此仅在 IRT 传送结束和 IRT 传送开始之间的时段内一致。

这意味着用户程序只能在 PNIO_CP_CBE_STARTOP_IND 回调事件和 PNIO_CP_set_opdone( ) 之间访问一致的 IRT 数据。

在 PNIO_CP_CBE_OPFAULT_IND 回调事件到达时，数据将不再一致。
此情况将持续到下一个 PNIO_CP_CBE_STARTOP_IND 回调事件为止。

3.6.1 访问 IRT 数据

等时实时访问 IRT 数据

如果在 PNIO_CP_CBE_STARTOP_IND 回调事件和 PNIO_CP_set_opdone( ) 之间访问 IRT 数据，则称为等时实时访问 IRT 数据。 访问 IRT 数据的函数与访问非等时实时周期性 RT 数据的函数相同。

非等时实时访问 IRT 数据

如果在 PNIO_CP_CBE_STARTOP_IND 回调事件和 PNIO_CP_set_opdone( ) 以外的时间访问 IRT 数据，则称为异步访问 IRT 数据。 该访问将返回不一致的数据，并且返回 PNIO_WARN_IRT_INCONSISTENT 警告来指示此情况。

确定哪些数据为 IRT 数据

为使 IO 控制器用户程序能够识别引用 IRT 数据的逻辑地址，可配合使用 PNIO_ctrl_diag_req( ) 函数与 PNIO_CTRL_DIAG_CONFIG_SUBMODULE_LIST 诊断服务。

等时实时数据与经速度优化的访问函数

如果 IO 控制器用户程序使用经速度优化的访问函数来访问 IRT 数据，则这些访问将会被转发到无缓存的函数。 这使得经速度优化的 RT 数据访问函数对 IRT 数据透明。

从而您可在等时实时处理例程中使用经速度优化的访问函数来同时访问 IRT 数据和 RT 数据，无需再进行区分。 为保持性能不下降，在等时实时处理例程中应仅访问 IRT 数据。

PNIO_data_read_cache_refresh( ) 和 PNIO_data_write_cache_flush( ) 函数仅影响 RT 数据，而不影响 IRT 数据。

3.7 PROFINET IO 设备寻址

地址空间

PROFINET IO
系统有一个用于读取访问和一个用于写入访问的地址空间，这些空间供所有设备共同使用。 并且均在组态中指定。

请参见与 IO-Base
控制器用户程序通信的设备和模块（模块或子模块）的插槽地址组态情况。

3.8 回调机制

工作原理

在 IO-Base 控制器用户程序中编写回调函数。

回调函数可任意命名。

回调事件是由 IO-Base 接口调用的异步事件。 其可中断 IO-Base 控制器用户程序的执行，并可在单独的线程上启动回调函数。
这意味着必须具备同步技术。

每个回调事件都伴随一个回调类型。 回调事件类型定义了回调事件。

IO 控制器的回调函数

下表列出了 IO 控制器上的各回调事件和事件类型。
该表还说明了要使用什么来注册回调函数以及哪些情况会触发回调事件：

回调事件（异步）	回调事件类型	注册手段	触发条件
报警到达	PNIO_CBE_ALARM_IND	PNIO_controller_open( )	... IO 设备
读数据记录作业的结果到达	PNIO_CBE_ REC_READ_CONF	PNIO_controller_open( )	... PNIO_rec_read_req( )
写数据记录作业的结果到达	PNIO_CBE_REC_WRITE_CONF	PNIO_controller_open( )	... PNIO_rec_write_req( )

回调事件（异步）	回调事件类型	注册手段	触发条件
本地模式发生变化。	PNIO_CBE_MODE_IND	PNIO_register_cbf( )	... PNIO_set_mode( )
IO 设备连接的状态已改变。	PNIO_CBE_ DEV_ACT_CONF	PNIO_register_cbf( )	... PNIO_device_activate( )
激活 LED 的闪烁模式	PNIO_CBE_START_LED_FLASH_IND	PNIO_register_cbf( )	... 标识请求抵达。
取消激活 LED 的闪烁模式	PNIO_CBE_STOP_LED_FLASH_IND	PNIO_register_cbf( )	... 结束标识的请求抵达。
报告远程下载请求（固件更新或重新组态）	PNIO_CBE_CP_STOP_REQ	PNIO_register_cbf( )	... 固件更新请求或重新组态请求抵达
等时实时数据处理开始	PNIO_CP_CBE_ STARTOP_IND	PNIO_CP_register_cbf( )	... IRT 传送阶段结束；IRT 数据位于主机存储器中。
IRT 周期超出	PNIO_CP_ OPFAULT_IND	PNIO_CP_register_cbf( )	... PNIO_CP_set_opdone( )；用户程序调用 PNIO_CP_set_opdone( ) 过晚。

说明

在编译用户程序时包含多线程标准库。

说明

在回调函数内，只能调用访问 IO 数据的函数。
这些函数包括：

- PNIO_data_read( )
- PNIO_output_data_read( )
- PNIO_data_read_cache( )
- PNIO_data_read_cache_refresh( )
- PNIO_data_write( )
- PNIO_data_write_cache( )
- PNIO_data_write_cache_flush( )
- PNIO_CP_set_opdone( )

协调回调的顺序

回调函数可随时中断 IO-Base 控制器用户程序。 不同事件的回调函数也可互相中断。
由于可能从不同的线程调用回调函数，因此必须将回调函数设计成能够同时多次执行（可重入）。 具体而言，这表示共享变量的写入和读取必须由同步机制提供保护。

避免回调函数中的等待，尤其是当进入关键部分时。
请限制同一回调事件再次调用此回调函数。 应当尽可能应使用单独的数据库。

对于每个回调事件，都可注册一个单独的回调函数。
另一方面，也可将多个回调事件组合到一个回调函数中。

IO 控制器数据类型与函数的说明

4

本章详细介绍了 IO 控制器 IO-Base 用户编程接口的具体函数及其使用的数据类型。

本章主要希望为您编写 IO-Base 控制器用户程序提供参考。

4.1 管理函数

概述

IO 控制器可识别以下管理函数：

- PNIO_controller_open( )
- PNIO_register_cbf( )
- PNIO_controller_close( )

IRT 模式下可用的其它管理函数在“等时实时模式 (IRT) 接口 (页 85)”部分介绍。

4.1.1 PNIO_controller_open( )（将设备注册为 IO 控制器）

说明

IO-Base 控制器用户程序使用此函数向 IO-Base 函数注册 IO 控制器。

该函数还可注册处理报警事件和读/写数据记录事件的回调函数，或者使用 PNIO_CEP_SLICE_ACCESS ExtPar 标记启用对子模块的部分访问（周期性数据）。

以下说明仅适用于 SOFTNET PN IO：

在 PC 启动后，仅在 SIMATIC NET 组态阶段完成后允许使用 PNIO_controller_open( ) 函数。

如果该阶段未完成，将返回错误代码 PNIO_ERR_INTERNAL 并且必须重进行该调用。

相反，您还可评估“SimaticNetPcStationUpEvent”事件。 有关详细说明，请参见《调试 PC 站》手册。

语法

```
PNIO_UINT32  PNIO_controller_open(
    PNIO_UINT32      CpIndex,                    //in
    PNIO_UINT32      ExtPar,                     //in
    PNIO_CBF         cbf_rec_read_conf,          //in
    PNIO_CBF         cbf_rec_write_conf,         //in
    PNIO_CBF         cbf_alarm_ind,              //in
    PNIO_UINT32     *Handle                      //out
);
```

参数

名称	说明
CpIndex	模块索引 - 用于唯一标识通信模块。 请参见此参数的组态（CP 的“模块索引”）。
ExtPar	此参数的 32 个位均可用于进行参数分配。 有关各个位的意义，请参见“ExtPar（扩展参数）(页 116)”部分。 未定义的位将“保留”且不可被置位。
cbf_rec_read_conf	回调函数的地址，该回调函数在回调事件类型为 PNIO_CBE_REC_READ_CONF 的读数据记录作业抵达后启动。
	注意 函数指针不允许为空。
cbf_rec_write_conf	回调函数的地址，该回调函数在回调事件类型为 PNIO_CBE_REC_WRITE_CONF 的写数据记录作业抵达后启动。
	注意 函数指针不允许为空。
cbf_alarm_ind	回调函数的地址，该函数在回调事件类型为 PNIO_CBE_ALARM_IND 的报警抵达后启动。
	注意 函数指针可以为空。
Handle	分配给所注册 IO 控制器的句柄，在执行成功时返回给 IO-Base 控制器用户程序。 必须在后续所有函数调用中包含该句柄。

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_CONFIG_IN_UPDATE
- PNIO_ERR_CREATE_INSTANCE
- PNIO_ERR_INTERNAL
- PNIO_ERR_MAX_REACHED
- PNIO_ERR_NO_CONFIG
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_NO_LIC_SERVER（仅限 SOFTNET PNIO）
- PNIO_ERR_NO_RESOURCE
- PNIO_ERR_PRM_CALLBACK
- PNIO_ERR_PRM_CP_ID
- PNIO_ERR_PRM_EXT_PAR
- PNIO_ERR_PRM_HND

4.1.2 PNIO_register_cbf( )（回调函数注册）

说明

该函数注册一个回调函数。

说明
回调函数只能在离线模式下注册。

语法

```
PNIO_UINT32  PNIO_register_cbf(
    PNIO_UINT32        Handle,                    //in
    PNIO_CBE_TYPE    CbeType,                   //in
    PNIO_CBF         Cbf                        //in
);
```

参数

名称	说明
Handle	PNIO_controller_open( ) 的句柄
CbeType	与回调函数“Cbf”注册对应的回调事件类型；请参见PNIO_CBE_TYPE（回调事件类型） (页 114)部分。
Cbf	回调函数的地址，该回调函数在回调事件“CbeType”抵达后启动。
	注意 函数指针不允许为空。
	注意 如果已经向一个回调事件类型注册了回调函数，则只能在 PNIO_controller_close( ) 之后注册其它回调函数。

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_ALLREADY_DONE
- PNIO_ERR_INTERNAL
- PNIO_ERR_MODE_VALUE
- PNIO_ERR_PRM_CALLBACK
- PNIO_ERR_PRM_TYPE
- PNIO_ERR_WRONG_HND

编程示例

以下示例说明一个回调函数还可用于多个回调事件类型。

```
PNIO_CBF  callback1( );     // 适用于报警和 MODE 变化
PNIO_CBF  callback2( );     // 适用于数据记录确认
PNIO_controller_open(
    CpIndex,
    ExtPar,
    callback2,              // 适用于 READ 事件
    callback2,              // 也适用于 WRITE 事件
    callback1,              // 适用于 ALARM 事件
    &Handle
);
PNIO_register_cbf(
    Handle,
    PNIO_CBE_MODE_IND,
    callback1              // 适用于 MODE 变化
);
```

4.1.3 PNIO_controller_close( )（取消注册）

说明

IO-Base 控制器用户程序使用此函数取消注册 IO 控制器。

所有已注册的回调函数都将被取消注册（包括通过 PNIO_register_cbf( ) 注册的回调函数）。 在 PNIO_controller_close( ) 函数执行完毕后，再不会为已取消注册的 IO 控制器调用其它回调函数。

说明

在 PNIO_controller_close( ) 函数执行时，回调函数也可执行。

说明

在成功取消注册后，CP 上的句柄不再有效，并不再可供 IO-Base 控制器用户程序使用。

说明

在退出 IO-Base 控制器用户程序前，必须调用 PNIO_controller_close( ) 函数。

说明

如果 CP 在同一用户程序中同时充当 IO 控制器和 IO 设备，则在取消注册该 IO 控制器/IO 设备前，还必须停止写入和读取该 IO 控制器/IO 设备的 IO 数据。

语法

```
PNIO_UINT32  PNIO_controller_close(
    PNIO_UINT32     Handle                      //in
);
```

参数

名称	说明
Handle	PNIO_controller_open( ) 的句柄

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_WRONG_HND

4.1.4 PNIO_iosystem_reconfig

说明

此函数支持对 IO 系统进行组态控制。产品 CD 中包含实施此函数的示例。

语法

```
PNIO_UINT32 PNIO_CODE_ATTR PNIO_iosystem_reconfig(

 PNIO_UINT32 Handle,

 PNIO_IOS_RECONFIG_MODE Mode,

 PNIO_UINT32 DeviceCnt,

 PNIO_ADDR *DeviceList,

 PNIO_UINT32 PortInterconnectionCnt

 PNIO_ADDR *PortInterconnectionList,
```

```
);
```

元素

名称	说明
Handle	“PNIO_controller_open()”的句柄
Mode	指定要执行的模式。 • PNIO_IOS_RECONFIG_MODE_DEACT：取消激活所有 IO 设备，以避免出现诊断消息 • PNIO_IOS_RECONFIG_MODE_TAILOR：开始利用“DeviceList”和“PortInterconnectionList”中指定的参数进行适当“调整”，然后激活必不可少或可选的以及“DeviceList”中包含的所有 IO 设备。
DeviceCnt	“DeviceList”中包含的条目数
DeviceList	存在于实际设置中的可选 IO 设备的逻辑地址列表。
PortInterconnectionCnt	“PortInterconnectionList”中包含的条目数
PortInterconnectionList	端口互连的逻辑地址列表。 各端口互连是由一对逻辑地址构成的（逻辑地址设备 1/端口 2、逻辑地址设备 2/端口 1）

返回值

如果成功，则返回“PNIO_OK”。

如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_MODE_VALUE
- PNIO_ERR_TAILOR_INV_STATION
- PNIO_ERR_TAILOR_INV_PORT
- PNIO_ERR_NOT_POSSIBLE
- PNIO_ERR_WRONG_HND

4.2 与模式相关的函数

概述

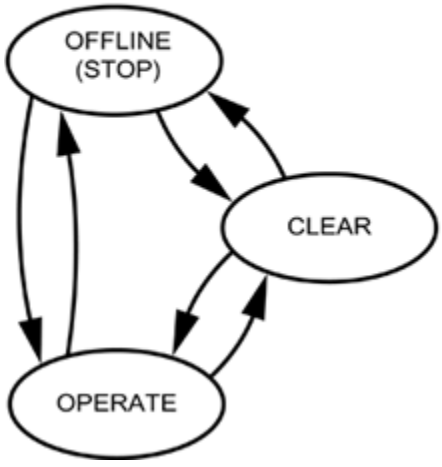
IO 控制器处于以下模式之一：

- OFFLINE (STOP)
- CLEAR
- OPERATE

可按照箭头所指示的方向改变模式。

通过调用 PNIO_set_mode( ) 函数来更改。

Operating mode changes of the IO controller



OFFLINE (STOP)

OFFLINE 模式是调用 PNIO_controller_open( ) 后的 IO 控制器状态。（OFFLINE 模式对应于 STOP 模式。）

在此模式下无法传送 IO 数据、数据记录或报警。

说明

回调函数只能在 OFFLINE 模式下注册。

说明

如果 IO 控制器处于 OFFLINE 模式，则不可激活或取消激活 IO 设备。

说明

如果 IO 控制器处于 OFFLINE 模式，所有由 PROFINET IO 写入到过程映像的 IO 数据以及数据的本地状态（“IOlocState”）都将丢失。
因此，在切换到 OPERATE 模式后，所有 IO 设备的全部 IO 数据都必须重新写入。

CLEAR

只有某些特定的 PROFINET IO 设备支持 IO 控制器的此状态，例如 SIMATIC NET IE/PB Link PN IO。 有关此功能的详细信息，请参见制造商信息。

OPERATE（常规模式）

在 OPERATE 模式（正常运行）下，将传送当前 IO 数据。

如果相应的回调函数已注册，则可以写入和读取数据记录，也可以处理报警。

4.2.1 PNIO_set_mode( )（设置工作模式）

说明

可使用此函数切换到新的模式。

该函数将立即返回。 如果函数返回 PNIO_OK，则通过 PNIO_CBE_MODE_IND 事件报告实际状态变化情况。 用户只有在已通过 PNIO_CBE_MODE_IND( ) 确认前一个 PNIO_set_mode( ) 作业后，才可再次调用 PNIO_set_mode( )。

语法

```
PNIO_UINT32 PNIO_set_mode(
 PNIO_UINT32 Handle,               //in
 PNIO_MODE_TYPE Mode               //in
) ;
```

参数

名称	说明
Handle	PNIO_controller_open( ) 的句柄
Mode	要设置的新模式；请参见“PNIO_MODE_TYPE（运行模式类型） (页 112)”部分。

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_MODE_VALUE
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_SET_MODE_NOT_ALLOWED
- PNIO_ERR_WRONG_HND
- PNIO_ERR_SEQUENCE

4.2.2 回调事件 PNIO_CBE_MODE_IND（报告自身工作模式变化）

说明

PNIO_CBE_MODE_IND 回调事件报告 IO
控制器的模式变化；请参见“PNIO_CBE_TYPE（回调事件类型） (页 114)”部分。

必须使用 PNIO_register_cbf( ) 函数注册了该回调事件。

说明

此回调事件必须在 PNIO_set_mode 调用前注册。

说明

在 PNIO_controller_open( ) 执行完毕后，IO 控制器将处于 OFFLINE 模式。
但是，由于此时还未注册回调函数，因此报告该模式变化情况。

语法

以下语法显示此事件的具体参数，将其作为 PNIO_CBE_PRM 结构中“联合体”的一部分；请参见“PNIO_CBE_PRM（回调事件参数） (页 115)”部分。

```
typedef struct
{
    PNIO_MODE_TYPE           Mode;                          //in
} ATTR_PACKED PNIO_CBE_PRM_MODE_IND;
```

参数

名称	说明
Mode	新模式；请参见“PNIO_MODE_TYPE（运行模式类型） (页 112)”部分和头文件 PNIOUSRX.H。

4.2.3 PNIO_device_activate( )（激活/取消激活 IO 设备）

激活/取消激活的含义

IO 设备激活后即存在连接，如果取消激活 IO 设备，则不存在连接。

在 IO 控制器的工作模式切换到“OPERATE”模式后，之前激活过的已组态 I/O 设备若是可访问，则会通过站恢复报警告知该设备已激活。

函数说明

借助此函数，IO 控制器可激活或取消激活 IO 设备。

该函数将立即返回。如果该函数已返回 PNIO_OK，则使用“PNIO_CBE_DEV_ACT_CONF”事件报告该 IO 设备是否可收到激活或取消激活的信息。

除此之外，IO 设备实际激活或取消激活的情况也将通过站恢复报警 (PNIO_ALARM_DEV_RETURN) 或站故障报警 (PNIO_ALARM_DEV_FAILURE) 报告。

事件“PNIO_CBE_DEV_ACT_CONF”和站故障报警或站恢复报警的时间顺序不可预测。

说明

DN 驱动程序可并行执行作业

最多可以同时执行 8 个作业。

说明

请注意，已取消激活的 IO 设备只有在 PNIO_ALARM_DEV_FAILURE 回调和 PNIO_CBE_DEV_ACT_CONF 回调抵达后才可断开或关闭。

语法

```
PNIO_UINT32 PNIO_device_activate(
 PNIO_UINT32 Handle, //in
 PNIO_ADDR *pAddr, //in
 PNIO_DEF_ACT_TYPE DeviceMode //in
);
```

参数

名称	说明
Handle	“PNIO_controller_open()”的句柄
pAddr	接收发送作业的 IO 设备的任意地址。 **解释** 每个 IO 设备可以有多个模块，因而也可以有多个地址。 要激活或取消激活整个 IO 设备，只需要访问这些 IO 设备地址中的一个即可。
DeviceMode	“DeviceMode”参数可接受两种状态： • PNIO_DA_FALSE 取消激活 IO 设备。 • PNIO_DA_TRUE 激活 IO 设备。

返回值

如果成功，则返回“PNIO_OK”。

如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_DEV_ACT_NOT_ALLOWED
- PNIO_ERR_INTERNAL
- PNIO_ERR_MODE_VALUE
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_PRM_ADD
- PNIO_ERR_PRM_DEV_STATE
- PNIO_ERR_WRONG_HND

4.2.4 回调事件 PNIO_CBE_DEV_ACT_CONF（将连接状态通知给 IO 设备）

说明

回调事件 PNIO_CBE_DEV_ACT_CONF 将 IO
设备的连接状态（已激活或已取消激活）通知给 IO-Base 控制器用户程序。

必须使用 PNIO_register_cbf( ) 函数注册了该回调事件。

语法

以下语法显示此事件的具体参数，将其作为 PNIO_CBE_PRM
结构中“联合体”的一部分；请参见“PNIO_CBE_PRM（回调事件参数） (页 115)”部分。

```
typedef struct
{
    PNIO_ADDR              *pAddr;
    PNIO_DEV_ACT_TYPE   Mode;
    PNIO_UINT32             Result;
} ATTR_PACKED PNIO_CBE_PRM_DEV_ACT_CONF;
```

参数

名称	说明
pAddr	接收发送作业的 IO 设备的地址指针。
Mode	设备连接的当前状态
Result	（错误代码） “Result”参数通过 PNIO_OK 指示已成功激活或取消激活。

4.3 用于读取 IO 数据的函数

概述

以下函数可用于读取 IO 数据：

- 无缓冲直接访问过程映像
  - PNIO_data_read( )
  - PNIO_output_data_read( )
- 有缓冲快速访问过程映像
  - PNIO_data_read_cache_refresh( )
  - PNIO_data_read_cache( )

4.3.1 PNIO_data_read( )（读取输入数据）

说明

此函数从过程映像读取输入数据。此函数随后立即返回。过程映像会通过 IO 控制器周期性地保持最新状态，而与此读取作业无关。

此基本机制将在“非等时访问周期性 IO 数据 (RT) (页 30)”部分进行介绍。

说明

PNIO_data_read( ) 函数读取单个子模块的数据。因此，仅在等时实时模式中访问 IRT 数据时才能实现不受子模块限制的数据一致性。

说明

该函数不可重入。确保仅通过线程上下文或使用锁定策略进行函数访问。

语法

```
PNIO_UINT32 PNIO_data_read(
 PNIO_UINT32 Handle,
 PNIO_ADDR *pAddr,
 PNIO_UINT32 BufLen,
 PNIO_UINT32 *pDataLen,
 PNIO_UINT8 *pBuffer,
 PNIO_IOXS IOlocState,
 PNIO_IOXS *pIOremState
);
```

参数

名称	说明
Handle	PNIO_controller_open( ) 的句柄
pAddr	子模块的逻辑地址。对于部分访问，此地址可为子模块的任何地址。 示例： 4 字节输入。地址 100 至 103 有效地址：100、101、102、103
	注意 在 IO 控制器上，“pAddr”是从中读取数据的远程子模块的地址。
BufLen	可用数据缓冲区的长度（字节）。 此参数指定将读取多少字节。可读取的字节数少于可用字节数（即，从指定地址至模块末端的字节数）。

名称	说明
pDataLen	读取数据缓冲区的长度（字节） 如果 IO 设备数据的长度小于或等于“BufLen”，则“*pDataLen”包含已读取的 IO 数据长度。 如果 IO 设备数据的长度大于可用数据缓冲区，则尽可能多地将数据读取至数据缓冲区。 但是，“*pDataLen”应包含读取所有 IO 数据所需的数据缓冲区的长度。
pBuffer	指向读取 IO 数据的数据缓冲区的指针。
IOlocState	读取 IO 数据的本地状态（GOOD 或 BAD）。
	注意 IO-Base 控制器用户程序使用“IOlocState”指定本地状态。 在下一个周期，通过底层 IO-Base 函数将“IOlocState”发送至远程设备。 BAD 状态会通知通信伙伴，IO-Base 控制器用户程序无法处理收到的 IO 数据。 部分访问： 对于每个部分访问，所传输的状态（“IOlocState”）都将适用于整个子模块。
pIOremState	读取 IO 数据的远程状态（GOOD 或 BAD）。 指示读取数据的质量/有效性。

示例：PNIO_data_read( ) 部分访问

例如，我们假定以下两个输入模块：

- 2 字节，I 地址：0 ... 1
- 4 字节，I 地址：2 ... 5

I 地址	BufLen	*pDataLen OUTPUT	返回值
2	8	4	OK
2	2	4	OK
3	2	3	OK

I 地址	BufLen	*pDataLen OUTPUT	返回值
5	1	1	OK
5	8	1	OK
1	2	1	OK
1	1	1	OK

返回值

如果成功，则返回
PNIO_OK。如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_CONNECTION
- PNIO_ERR_PRM_ADD
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_IO_TYPE
- PNIO_ERR_PRM_LEN
- PNIO_ERR_PRM_LOC_STATE
- PNIO_ERR_PRM_RSTATE
- PNIO_ERR_UNKNOWN_ADDR
- PNIO_ERR_WRONG_HND
- PNIO_WARN_IRT_INCONSISTENT

4.3.2 PNIO_output_data_read( )（读取输出数据）

说明

此函数从过程映像读取输出数据。 此函数随后立即返回。 过程映像会通过 IO 控制器周期性地保持最新状态，而与此读取作业无关。

此基本机制将在“非等时访问周期性 IO 数据 (RT) (页 30)”部分进行介绍。

说明

PNIO_output_data_read( ) 函数读取单个子模块的数据。
因此，仅在等时实时模式中访问 IRT 数据时才能实现不受子模块限制的数据一致性。

语法

```
PNIO_UINT32  PNIO_output_data_read(
    PNIO_UINT32        Handle,                          //in
    PNIO_ADDR         *pAddr,                           //in
    PNIO_UINT32        BufLen,                          //in
    PNIO_UINT32       *pDataLen,                        //out
    PNIO_UINT8        *pBuffer,                         //out
    PNIO_IOXS         *pIOlocState,                     //out
    PNIO_IOXS         *pIOremState                      //out
);
```

参数

名称	说明
Handle	PNIO_controller_open( ) 的句柄
pAddr	子模块的逻辑地址。 对于部分访问，此地址可为子模块的任何地址。 示例： 4 字节输出。 地址 100 至 103 有效地址：100、101、102、103
	注意 在 IO 控制器上，“pAddr”是从中读取数据的远程子模块的地址。
BufLen	可用数据缓冲区的长度（字节）。 此参数指定将读取多少字节。 可读取的字节数少于可用字节数（即，从指定地址至模块末端的字节数）。

名称	说明
pDataLen	读取数据缓冲区的长度（字节） 如果 IO 设备数据的长度小于或等于“BufLen”，则“*pDataLen”包含已读取的 IO 数据长度。 如果 IO 设备数据的长度大于可用数据缓冲区，则尽可能多地将数据读取至数据缓冲区。 但是，“*pDataLen”应包含读取所有 IO 数据所需的数据缓冲区的长度。
pBuffer	指向读取 IO 数据的数据缓冲区的指针。
pIOlocState	读取 IO 数据的本地状态（GOOD 或 BAD）。
	注意 IO-Base 控制器用户程序使用“IOlocState”指定本地状态。 在下一个周期，通过底层 IO-Base 函数将“IOlocState”发送至远程设备。 BAD 状态会通知通信伙伴，IO-Base 控制器用户程序无法处理收到的 IO 数据。 部分访问： 对于每个部分访问，所传输的状态（“IO-locState”）都将适用于整个子模块。
pIOremState	读取 IO 数据的远程状态（GOOD 或 BAD）。 指示读取数据的质量/有效性。

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_CONNECTION
- PNIO_ERR_PRM_ADD
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_IO_TYPE
- PNIO_ERR_PRM_LEN

- PNIO_ERR_PRM_LOC_STATE
- PNIO_ERR_PRM_RSTATE
- PNIO_ERR_UNKNOWN_ADDR
- PNIO_ERR_WRONG_HND

4.3.3 PNIO_data_read_cache_refresh( )（将 IO 数据传输至读取缓存）

说明

IO-Base 控制器用户程序使用此函数启动将 CP
上过程映像中的以下数据传输至读取缓存：

- RT IO 输入数据及其远程状态
- RT IO 输出数据的远程状态

此基本机制将在“非等时访问周期性 IO 数据 (RT) (页 30)”部分进行介绍。

此函数不会将任何 IRT IO
数据传输至读取缓存；另请参见“等时实时和非等时实时访问周期性 IO 数据 (IRT) (页 35)”部分。

语法

```
PNIO_UINT32  PNIO_data_read_cache_refresh(
    PNIO_UINT32     Handle                      //in
);
```

参数

名称	说明
Handle	PNIO_controller_open( ) 的句柄

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_WRONG_HND

4.3.4 PNIO_data_read_cache( )（从读取缓存中读取 IO 数据）

说明

此函数从读取缓存中读取 RT 输入数据，并将 RT 输出数据的本地状态写至写入缓存。 此函数随后立即返回。 过程映像会通过 IO
控制器周期性地保持最新状态，而与此读取作业无关。

此基本机制将在“非等时访问周期性 IO 数据 (RT) (页 30)”部分进行介绍。

此函数不会将任何 IRT IO
数据传输至读取缓存；另请参见“等时实时和非等时实时访问周期性 IO 数据 (IRT) (页 35)”部分。

说明

PNIO_data_read_cache( ) 读取单个子模块的数据。 因此，仅在等时实时模式中访问 IRT 数据时才能实现不受子模块限制的数据一致性。

语法

```
PNIO_UINT32  PNIO_data_read_cache(
    PNIO_UINT32        Handle,                  //in
    PNIO_ADDR         *pAddr,                   //in
    PNIO_UINT32        BufLen,                  //in
    PNIO_UINT32       *pDataLen,                //out
    PNIO_UINT8        *pBuffer,                 //out
    PNIO_IOXS          IOlocState,              //in
    PNIO_IOXS         *pIOremState              //out
);
```

参数

名称	说明
Handle	PNIO_controller_open( ) 的句柄
pAddr	子模块的逻辑地址。 对于部分访问，此地址可为子模块的任何地址。 示例： 4 字节输入。 地址 100 至 103 有效地址：100、101、102、103

名称	说明
	注意 在 IO 控制器上，“pAddr”是从中读取数据的远程子模块的地址。
BufLen	可用数据缓冲区的长度（字节）。 此参数指定将读取多少字节。 可读取的字节数少于可用字节数（即，从指定地址至模块末端的字节数）。
pDataLen	读取数据缓冲区的长度（字节） 如果 IO 设备数据的长度小于或等于“BufLen”，则“*pDataLen”包含已读取的 IO 数据长度。 如果 IO 设备数据的长度大于可用数据缓冲区，则尽可能多地将数据读取至数据缓冲区。 但是，“*pDataLen”应包含读取所有 IO 数据所需的数据缓冲区的长度。
pBuffer	指向读取 IO 数据的数据缓冲区的指针
IOlocState	读取 IO 数据的本地状态（GOOD 或 BAD）
	注意 IO-Base 控制器用户程序使用“IOlocState”指定本地状态。 在下一个周期，通过底层 IO-Base 函数将“IOlocState”发送至远程设备。 BAD 状态会通知通信伙伴，IO-Base 控制器用户程序无法处理收到的 IO 数据。 部分访问： 对于每个部分访问，所传输的状态（“IO-locState”）都将适用于整个子模块。
pIOremState	读取 IO 数据的远程状态（GOOD 或 BAD）。 指示读取数据的质量/有效性。

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_NO_CONNECTION
- PNIO_ERR_PRM_ADD
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_IO_TYPE
- PNIO_ERR_PRM_LEN
- PNIO_ERR_PRM_LOC_STATE
- PNIO_ERR_PRM_RSTATE
- PNIO_ERR_UNKNOWN_ADDR
- PNIO_ERR_VALUE_LEN
- PNIO_ERR_WRONG_HND

4.4 用于写入 IO 数据的函数

概述

以下函数用于写入 IO 数据：

- 无缓冲直接访问过程映像
  - PNIO_data_write( )
- 有缓冲快速访问过程映像
  - PNIO_data_write_cache( )
  - PNIO_data_write_cache_flush( )

4.4.1 PNIO_data_write( )（写入 IO 数据）

说明

O-Base 控制器用户程序使用此函数将 IO
数据写入过程映像。此函数随后立即返回。在下一个周期，IO 控制器将 IO 数据传输至 IO 设备。

此基本机制将在“非等时访问周期性 IO 数据 (RT) (页 30)”部分进行介绍。

说明

PNIO_data_write( ) 函数写入单个子模块的数据。因此，仅在等时实时模式中访问 IRT 数据时才能实现不受子模块限制的数据一致性。

说明

该函数不可重入。确保仅通过线程上下文或使用锁定策略进行函数访问。

语法

```
PNIO_UINT32 PNIO_data_write(
 PNIO_UINT32 Handle,
 PNIO_ADDR *pAddr,
 PNIO_UINT32 BufLen,
 PNIO_UINT8 *pBuffer,
 PNIO_IOXS IOlocState,
 PNIO_IOXS *pIOremState
);
```

参数

名称	说明
Handle	PNIO_controller_open( ) 的句柄
pAddr	子模块的逻辑地址。
	注意 在 IO 控制器上，“pAddr”是向其写入数据的远程子模块的地址。 对于部分访问，此地址可为子模块的任何逻辑地址。必须相应地调整数据长度 (BufLen)。调整范围最大至模块末端（请参见“ExtPar（扩展参数） (页 116)”部分）。
BufLen	数据缓冲区的长度（字节）。 此参数指定要写入的字节数。可写入的字节数少于从指定地址至模块末端的可用字节数。
	注意 部分访问未启用： “BufLen”必须与寻址模块的输出数据长度准确匹配，否则在返回代码中报告错误。 部分访问启用： “BufLen”可能会小于或等于寻址模块的输出数据长度。 最大数据长度为从指定地址至模块末端的可用字节数。
pBuffer	指向要写入 IO 数据的数据缓冲区的指针。
IOlocState	要写入 IO 数据的本地状态（GOOD 或 BAD）。 部分访问启用： 最后的传输状态（“IOlocState”）适用于整个子模块。 换句话说，如果部分状态具有 BAD 状态，则模块的所有数据都具有 BAD 状态。
pIOremState	远程状态，写入 IO 数据的返回状态（GOOD 或 BAD）。
	注意 此状态与早期写入的 IO 数据有关，与刚刚发送的写入作业无关。
	注意 在 BAD 状态下，远程设备会发送信号至 IO-Base 控制器用户程序，告知其不能处理之前收到的 IO 数据。

示例：PNIO_data_write( ) 部分访问

例如，我们假定以下三个输出模块：

- 2 字节，Q 地址：0 ... 1
- 4 字节，Q 地址：2 ... 5
- 4 字节，Q 地址：10 ... 13

Q 地址	BufLen	返回值
2	8	PNIO_ERR_VALUE_LEN
2	2	PNIO_OK
1	2	PNIO_ERR_VALUE_LEN
1	1	PNIO_OK
3	3	PNIO_OK
3	1	PNIO_OK
3	4	PNIO_ERR_VALUE_LEN
6	4	PNIO_ERR_UNKNOWN_ADDR
10	1	PNIO_OK
11	1	PNIO_OK
9	1	PNIO_ERR_UNKNOWN_ADDR

返回值

如果成功，则返回
PNIO_OK。如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_NO_CONNECTION
- PNIO_ERR_PRM_ADD
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_IO_TYPE
- PNIO_ERR_PRM_LEN
- PNIO_ERR_PRM_LOC_STATE
- PNIO_ERR_PRM_RSTATE

- PNIO_ERR_UNKNOWN_ADDR
- PNIO_ERR_VALUE_LEN
- PNIO_ERR_WRONG_HND
- PNIO_WARN_IRT_INCONSISTENT

4.4.2 PNIO_data_write_cache( )（将 IO 数据写至写入缓存）

说明

IO-Base 控制器用户程序使用此函数将 RT 输出数据写至写入缓存，并从读取缓存读取 RT 输入数据的远程状态。 数据将保留在写入缓存中，直到下一个 PNIO_data_write_cache_flush( ) 函数调用为止。

此基本机制将在“非等时访问周期性 IO 数据 (RT) (页 30)”部分进行介绍。

此函数对于 IRT IO 数据是透明的；另请参见“等时实时和非等时实时访问周期性 IO 数据 (IRT) (页 35)”部分。

说明

PNIO_data_write_cache( ) 函数总是与一个子模块的数据相关。
因此，仅在等时实时模式中访问 IRT 数据时才能实现不受子模块限制的数据一致性。

语法

```
PNIO_UINT32  PNIO_data_write_cache(
    PNIO_UINT32     Handle,                          //in
    PNIO_ADDR      *pAddr,                           //in
    PNIO_UINT32     BufLen,                          //in
    PNIO_UINT8     *pBuffer,                         //in
    PNIO_IOXS       IOlocState,                      //in
    PNIO_IOXS      *pIOremState                     //out
);
```

参数

名称	说明
Handle	PNIO_controller_open( ) 的句柄
pAddr	子模块的逻辑地址。

名称	说明
	注意 在 IO 控制器上，“pAddr”是向其写入数据的远程子模块的地址。 对于部分访问，此地址可为子模块的任何逻辑地址。 必须相应地调整数据长度 (BufLen)。 调整范围最大至模块末端（请参见“ExtPar（扩展参数） (页 116)”部分）。
BufLen	数据缓冲区的长度（字节）。 此参数指定要写入的字节数。 可写入的字节数少于从指定地址至模块末端的可用字节数。
	注意 部分访问未启用： “BufLen”必须与寻址模块的输出数据长度准确匹配，否则在返回代码中报告错误。 部分访问启用： “BufLen”可能会小于或等于寻址模块的输出数据长度。 最大数据长度为从指定地址至模块末端的可用字节数。
pBuffer	指向要写入 IO 数据的数据缓冲区的指针。
IOlocState	要写入 IO 数据的本地状态（GOOD 或 BAD）。 部分访问启用： 最后的传输状态（“IOlocState”）适用于整个子模块。 换句话说，如果部分状态具有 BAD 状态，则模块的所有数据都具有 BAD 状态。
pIOremState	远程状态，写入 IO 数据的返回状态（GOOD 或 BAD）。
	注意 此状态与早期写入的 IO 数据有关，与刚刚发送的写入作业无关。
	注意 在 BAD 状态下，远程设备会发送信号至 IO-Base 控制器用户程序，告知其不能处理之前收到的 IO 数据。
	注意 仅在 PNIO_data_read_cache_refresh( ) 调用后传送此状态。

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_NO_CONNECTION
- PNIO_ERR_PRM_ADD
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_IO_TYPE
- PNIO_ERR_PRM_LEN
- PNIO_ERR_PRM_LOC_STATE
- PNIO_ERR_PRM_RSTATE
- PNIO_ERR_UNKNOWN_ADDR
- PNIO_ERR_VALUE_LEN
- PNIO_ERR_WRONG_HND

4.4.3 PNIO_data_write_cache_flush( )（将 IO 数据从写入缓存写至过程映像）

说明

IO-Base 控制器用户程序使用此函数启动将写入缓存中的以下数据传输至 CP 上的过程映像：

- RT IO 输出数据及其本地状态。
- RT IO 输入数据的本地状态。

此基本机制将在“非等时访问周期性 IO 数据 (RT) (页 30)”部分进行介绍。

此函数不会传输写入缓存中的任何 IRT IO 数据；另请参见“等时实时和非等时实时访问周期性 IO 数据 (IRT) (页 35)”部分。

语法

```
PNIO_UINT32  PNIO_data_write_cache_flush(
    PNIO_UINT32     Handle,                          //in
);
```

参数

名称	说明
Handle	PNIO_controller_open( ) 的句柄

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_WRONG_HND

4.5 用于读取数据记录的接口

说明

IO 控制器通过 PNIO_rec_read_req( ) 函数发送读取数据记录作业。
读取数据记录作业被发送至 IO 设备。

IO 设备通过将数据记录发送至 IO 控制器来响应此读取数据记录作业。

通过回调事件 PNIO_CBE_REC_READ_CONF 指示此数据记录到达 IO 控制器；将启动注册的回调函数。

此基本机制将在“非等时访问周期性 IO 数据 (RT) (页 30)”部分进行介绍。

IO 设备支持的诊断数据记录

下表列出 IO 设备支持、因而能够由 IO 控制器读取的诊断数据记录的数据记录号：

数据记录号（记录索引）	内容和含义
0x800A	子模块插槽的通道诊断
0x800B	子模块插槽的通道诊断和供应商特定诊断信息
0x800C	子模块插槽的通道诊断和供应商特定诊断信息
0xC00A	插槽的通道诊断
0xC00B	插槽的通道诊断和供应商特定诊断信息
0xC00C	插槽的通道诊断和供应商特定诊断信息

数据记录号（记录索引）	内容和含义
0xE002	分配给 IO 控制器的 IO 设备的预期组态和实际组态偏差。
0xE00A	AR 的通道诊断
0xE00B	AR 的通道诊断和供应商特定诊断信息
0xE00C	AR 的通道诊断和供应商特定诊断信息
0xF00A	API 的通道诊断
0xF00B	API 的通道诊断和供应商特定诊断信息
0xF00C	API 的通道诊断和供应商特定诊断信息

可以在相关 IO 设备的文档中找到所支持的其它数据记录。

4.5.1 PNIO_rec_read_req( )（发送读取数据记录作业）

说明

IO
控制器使用此函数发送读取数据记录作业。如果函数返回“PNIO_OK”，则通过“PNIO_CBE_REC_READ_CONF”回调事件来指示作业的结果。

还可以使用此函数读取存储在数据记录中的 IO 设备的诊断数据。请阅读相关 IO 设备的说明。

说明

DN 驱动程序可并行执行作业
最多可以同时执行 16 个作业。

语法

```
PNIO_UINT32 PNIO_rec_read_req(
 PNIO_UINT32 Handle, //in
 PNIO_ADDR *pAddr, //in
 PNIO_UINT32 RecordIndex, //in
 PNIO_REF ReqRef, //in
```

```
 PNIO_UINT32 Length //in
);
```

参数

名称	说明
Handle	“PNIO_controller_open()”的句柄
pAddr	读取数据记录作业所对应的 IO 设备模块的地址。
	注意 如果地址 (pAddr) 为诊断地址，则必须将参数“IODataType”设置为输入。
	注意 “IODataType”参数将在“PNIO_ADDR（地址结构） (页 114)”部分介绍。
RecordIndex	数据记录号
ReqRef	由 IO-Base 控制器用户程序分配的引用。
	注意 IO-Base 控制器用户程序通过“ReqRef”引用能够区别不同的作业。此参数可由 IO-Base 控制器用户程序分配任何值，且通过回调事件“PNIO_CBE_REC_READ_CONF”返回。
Length	4096 字节的数据记录长度。通过回调事件“PNIO_CBE_REC_READ_CONF”返回指向数据记录及其实际长度的指针。

返回值

如果成功，则返回“PNIO_OK”。

如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_PRM_ADD

- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_REC_INDEX
- PNIO_ERR_VALUE_LEN
- PNIO_ERR_WRONG_HND

4.5.2 回调事件 PNIO_CBE_REC_READ_CONF（指示读取数据记录作业的结果）

说明

PNIO_CBE_REC_READ_CONF 回调事件报告通过 PNIO_rec_read_req 调用先前发送的读取数据记录作业的结果；请参见“PNIO_CBF（PNIO 回调函数） (页 116)”部分。

必须使用 PNIO_controller_open( ) 函数注册了该回调事件。

语法

以下语法显示此事件的具体参数，将其作为 PNIO_CBE_PRM 结构中“联合体”的一部分；请参见“PNIO_CBE_PRM（回调事件参数） (页 115)”部分。

```
typedef struct
{
    PNIO_UINT32           Length;                  //in
    const PNIO_UINT8     *pBuffer;                 //in
    PNIO_ADDR            *pAddr;                   //in
    PNIO_UINT32           RecordIndex;             //in
    PNIO_REF              ReqRef;                  //in
    PNIO_ERR_STAT         Err;                     //in
} ATTR_PACKED PNIO_CBE_PRM_REC_READ_CONF;
```

参数

名称	说明
Length	“pBuffer”指向的传送的数据记录的长度（字节）。
pBuffer	指向数据缓冲区的指针。在正常的情况下，数据缓冲区将包含请求的数据记录。
	注意 数据缓冲区仅在回调函数内有效，函数完成后即失效！

名称	说明
	注意 不得在处理回调函数的过程中更改数据缓冲区。
pAddr	响应读取数据记录作业的 IO 设备模块的地址。
RecordIndex	数据记录号
ReqRef	由 IO-Base 控制器用户程序分配的引用。
	注意 此处，返回通过调用 PNIO_rec_read_req( ) 指定的引用。 IO-Base 控制器用户程序通过此引用能够区别不同的作业。
Err	执行作业有关的错误代码

4.6 用于写入数据记录的接口

说明

IO 控制器通过 PNIO_rec_write_req( ) 函数发送写入数据记录作业。
写入数据记录作业与数据记录一起被发送至 IO 设备。

IO 设备接收并确认此写入数据记录作业。

通过回调事件 PNIO_CBE_REC_WRITE_CONF 指示写入数据记录确认到达 IO 控制器；将启动已注册的回调函数。

4.6.1 PNIO_rec_write_req( )（发送写入数据记录作业）

说明

IO 控制器使用此函数发送写入数据记录作业。如果函数返回 PNIO_OK，则通过 PNIO_CBE_REC_WRITE_CONF 回调事件来指示作业的结果。

说明

DN 驱动程序可并行执行作业

最多可以同时执行 16 个作业。

语法

```
PNIO_UINT32 PNIO_rec_write_req(
 PNIO_UINT32 Handle, //in
 PNIO_ADDR *pAddr, //in
 PNIO_UINT32 RecordIndex, //in
 PNIO_REF ReqRef //in
 PNIO_UINT32 Length, //in
 PNIO_UINT8 *pBuffer //in
);
```

参数

名称	说明
Handle	“PNIO_controller_open()”的句柄
pAddr	读取数据记录作业所对应的 IO 设备模块的地址。
	注意 如果地址 (pAddr) 为诊断地址，则必须将参数“IODataType”设置为输入。
	注意 “IODataType”参数将在“PNIO_ADDR（地址结构） (页 114)”部分介绍。
RecordIndex	数据记录号
ReqRef	由 IO-Base 控制器用户程序分配的引用。

名称	说明
	注意 IO-Base 控制器用户程序通过“ReqRef”引用能够区别不同的作业。此参数可由 IO-Base 控制器用户程序分配任何值，并在回调事件“PNIO_CBE_REC_WRITE_CONF”中返回。
Length	数据缓冲区的长度（字节） 最大长度取决于所使用的版本和产品。可在“readme.txt”文件中查看此信息。
pBuffer	包含数据记录的数据缓冲区。
	注意 “PNIO_rec_write_req()”函数的调用方必须使包含要写入数据记录的“pBuffer”数据缓冲区可用。

返回值

如果成功，则返回“PNIO_OK”。

如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_PRM_ADD
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_REC_INDEX
- PNIO_ERR_VALUE_LEN
- PNIO_ERR_WRONG_HND

4.6.2 回调事件 PNIO_CBE_REC_WRITE_CONF（指示写入数据记录作业的结果）

说明

PNIO_CBE_REC_WRITE_CONF 回调事件报告通过 PNIO_rec_write_req 调用先前发送的写入数据记录作业的结果；请参见“PNIO_CBF（PNIO 回调函数） (页 116)”部分。

必须使用 PNIO_controller_open( ) 函数注册了该回调事件。

语法

以下语法显示此事件的具体参数，将其作为 PNIO_CBE_PRM 结构中“联合体”的一部分；请参见“PNIO_CBE_PRM（回调事件参数） (页 115)”部分。

```
typedef struct
{
    PNIO_ADDR            *pAddr;                  //in
    PNIO_UINT32           RecordIndex;            //in
    PNIO_REF              ReqRef;                 //in
    PNIO_ERR_STAT         Err;                    //in
} ATTR_PACKED PNIO_CBE_PRM_REC_WRITE_CONF;
```

参数

名称	说明
pAddr	读取数据记录作业所对应的 IO 设备模块的地址。
RecordIndex	数据记录号
ReqRef	由 IO-Base 控制器用户程序分配的引用。
	注意 返回通过 PNIO_rec_write_req( ) 指定的“ReqRef”引用。 IO-Base 控制器用户程序通过此引用能够区别不同的作业。
Err	执行作业有关的错误代码。
	注意 “Err”是由 IO 设备传送的 PROFINET IO 错误代码，这里的错误是指在 IO 设备上执行此读取数据记录作业时发生的错误。

4.7 报警接口

说明

报警自动生成并使用回调事件 PNIO_CBE_ALARM_IND 进行指示。

4.7.1 回调事件 PNIO_CBE_ALARM_IND（指示报警）

说明

PNIO_CBE_ALARM_IND 回调事件将 IO 设备中的进入报警报告给 IO 控制器；请参见“PNIO_CBF（PNIO 回调函数） (页 116)”部分。

如果想要在 IO-Base 控制器用户程序中使用此回调事件，其必须通过 PNIO_controller_open( ) 函数进行注册。

说明

通过 PNIO_controller_open( ) 注册此回调事件是可选的。
如果未注册此回调事件，报警到达后即会在内部对其进行确认。

说明

此回调函数退出后，将在内部确认 IO 设备的报警。

说明

在属于此回调事件的回调函数退出前，不会再将报警信号发送到此 IO 控制器的 IO-Base 控制器用户程序。 这些报警信号将在回调函数退出后发出。

说明

设备返回报警 (PNIO_ALARM_DEV_RETURN) 或插入报警 (PNIO_ALARM_PLUG) 到达后，会将 BAD 状态的无效 IO 数据传送至 IO 设备。 因此，IO-Base 控制器用户程序必须在返回或插入报警后，以 GOOD 状态和有效值写入所有输出数据。 这与输出数据的初始化相对应。

语法

以下语法显示此事件的具体参数，将其作为 PNIO_CBE_PRM 结构中联合体的一部分；请参见“PNIO_CBE_PRM（回调事件参数） (页 115)”部分。

```
typedef struct
{
    PNIO_ADDR            *pAddr;                    //in
    PNIO_REF              ReqRef;                   //in
    const PNIO_CTRL_ALARM_DATA     *pAlarmData;       //in
} ATTR_PACKED PNIO_CBE_PRM_ALARM_IND;
```

参数

名称	说明	
pAddr	生成报警的设备的地址。 例如，将传送以下地址：	
	…的故障	**“pAddr”的含义**
	IO 设备（站故障报警）	IO 设备的基址（诊断地址）
	模块（拔出报警）	模块地址
IndRef	内部分配的引用（保留）	
pAlarmData	包含有关报警信息的结构。	
	注意 “pAlarmData”指向在 PNIO_CTRL_ALARM_DATA 结构中的数据。 此对象仅在回调函数内有效，函数完成后即失效！ 不得在处理回调函数的过程中更改数据。	
	注意 PNIO_CTRL_ALARM_DATA 结构在 pniousrx.h 头文件中定义，且其中还包括“报警类型”和“报警说明符”参数。 有关报警的详细信息，请参见以下文档： • STEP 7 文档，例如 STEP 7 在线帮助中有关 SFB 54 的内容。 • “SIMATIC NET，PC 软件”CD 上 SIMATIC NET 文档“Alarm ASE.pdf”中 PROFINET IO 标准的摘要内容。 • IO 设备的文档	

返回值

无返回值

4.8 读出组态

概述

读出组态的诊断接口为 IO 控制器用户程序提供诊断请求服务。

要请求诊断服务，诊断接口要使 PNIO_ctrl_diag_req( ) 函数和 PNIO_CBE_CTRL_DIAG_CONF 回调事件可用：

以下诊断请求服务可用：

诊断服务	说明
PNIO_CTRL_DIAG_CONFIG_SUBMODULE_LIST	所有已组态子模块的列表

4.8.1 PNIO_ctrl_diag_req( )（触发诊断请求）

说明

此函数触发 IO 控制器的诊断请求。如果函数返回 PNIO_OK，则通过 PNIO_CBE_CTRL_DIAG_CONF 回调事件来指示作业的结果。

如果 IO-Base 控制器用户程序想要使用此函数，必须首先使用 PNIO_register_cbf( ) 函数注册 PNIO_CBE_CTRL_DIAG_CONF 回调事件。

语法

```
PNIO_UINT32 PNIO_ctrl_diag_req(

 PNIO_UINT32 Handle,

 PNIO_CTRL_DIAG *pDiagReq

);
```

参数

名称	说明
Handle	PNIO_controller_open( ) 的句柄
pDiagReq	诊断请求 - 指向包含诊断请求作业的已填充 PNIO_CTRL_DIAG 结构的指针；有关诊断请求的结构，请参见“PNIO_CTRL_DIAG（诊断请求） (页 119)”部分。
	注意 IO-Base 接口会一直占用“pDiagReq”指向的存储器，直到收到确认为止。通过 PNIO_CBE_CTRL_DIAG_CONF( ) 回调事件发出确认信号。 通过 PNIO_CBE_CTRL_DIAG_CONF 回调事件指示诊断请求结果后，存储器会立即返回至 IO-Base 控制器用户程序。

返回值

如果成功，则返回
PNIO_OK。如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_WRONG_HND
- PNIO_ERR_PRM_POINTER
- PNIO_ERR_PRM_INVALIDARG
- PNIO_ERR_PRM_ADD

4.8.2 回调事件 PNIO_CBE_CTRL_DIAG_CONF（指示诊断请求的结果）

说明

PNIO_CBE_CTRL_DIAG_CONF 回调事件将诊断请求的结果报告给 IO 控制器；另请参见“PNIO_ctrl_diag_req( )（触发诊断请求） (页 80)”部分。

如果想要在 IO-Base 用户程序中使用此回调事件，其必须通过 PNIO_register_cbf( ) 函数进行注册。

语法

以下语法显示此事件的具体参数，将其作为 PNIO_CBE_PRM 结构中“联合体”的一部分；请参见“PNIO_CBE_PRM（回调事件参数） (页 115)”部分。

```
typedef struct
{
    PNIO_CTRL_DIAG      *pDiagData;                    //in
    PNIO_UINT32          DiagDataBufferLen;            //in
    PNIO_UINT8          *pDiagDataBuffer;              //in
    PNIO_UINT32          ErrorCode;                    //in
} ATTR_PACKED PNIO_CBE_CTRL_DIAG_CONF;
```

参数

名称	说明
pDiagData	指向诊断请求的指针 - 指向传送至 PNIO_ctrl_diag_req( ) 函数的诊断请求；有关诊断请求的结构，请参见“PNIO_CTRL_DIAG（诊断请求） (页 119)”部分。
DiagDataBufferLen	“pDiagDataBuffer”中诊断回复的长度
pDiagDataBuffer	指向诊断响应缓冲区的指针 - 根据诊断服务 PNIO_CTRL_DIAG_ENUM 解释诊断响应；请参见“PNIO_CTRL_DIAG_ENUM（诊断服务） (页 120)”部分。
	注意 此缓冲区仅在回调中有效。 此缓冲区的结构取决于诊断请求。
ErrorCode	执行诊断服务时出现的错误代码。

返回值

无返回值。

4.9 用 LED 发出指示的站

概述

该接口由以下回调事件组成：

- PNIO_CBE_START_LED_FLASH_IND
- PNIO_CBE_STOP_LED_FLASH_IND

4.9.1 回调事件 PNIO_CBE_START_LED_FLASH_IND（激活 LED 的闪烁模式）

说明

PNIO_CBE_START_LED_FLASH_IND 回调事件通知 IO-Base 控制器用户程序已收到标识请求。

随后，IO-Base 控制器用户程序采取适当措施引起操作员的注意，例如点亮二极管。必须使用 PNIO_register_cbf( ) 函数注册了该回调事件。

以下语法显示此事件的具体参数，将其作为 PNIO_CBE_PRM 结构中“联合体”的一部分；请参见“PNIO_CBE_PRM（回调事件参数） (页 115)”部分。

说明

只要需要模块本身引起注意，PNIO_CBE_START_LED_FLASH_IND 和 PNIO_CBE_STOP_LED_FLASH_IND 回调就会每三秒钟交替出现。

当 PNIO_CBE_START_LED_FLASH_IND( ) 到达时，LED 应以“频率”参数指定的频率闪烁。

语法

```
typedef struct
{
    PNIO_UINT32       Frequency;                    //in
} ATTR_PACKED PNIO_CBE_PRM_START_LED_FLASH_IND;
```

参数

名称	说明
频率	指定的信号发送频率（赫兹）

4.9.2 回调事件 PNIO_CBE_STOP_LED_FLASH_IND（禁用 LED 的闪烁模式）

说明

PNIO_CBE_STOP_LED_FLASH_IND 回调事件通知 IO-Base
控制器用户程序已收到停止标识请求。

必须使用 PNIO_register_cbf( ) 函数注册了该回调事件。

以下语法显示此事件的具体参数，将其作为 PNIO_CBE_PRM
结构中“联合体”的一部分；请参见“PNIO_CBE_PRM（回调事件参数） (页 115)”部分。

语法

此事件类型无其它具体参数作为 PNIO_CBE_PRM
结构中“联合体”的一部分；请参见“PNIO_CBE_PRM（回调事件参数） (页 115)”部分。

4.10 回调事件 PNIO_CBE_CP_STOP_REQ（指示远程下载请求）

说明

PNIO_CBE_CP_STOP_REQ 回调事件通知 IO-Base
控制器用户程序已通过网络收到远程下载请求（固件更新或重新组态）。 必须使用 PNIO_register_cbf( ) 函数注册了该回调事件。

接收该回调后，用户程序必须发送 PNIO_controller_close( )。
之后，用户程序可再次调用 PNIO_controller_open( )。

但是，函数 PNIO_controller_close( ) 及 PNIO_controller_open( ) 不得在属于回调事件 PNIO_CBE_CP_STOP_REQ 的函数内执行。

下载期间，PNIO_controller_open( )
函数返回错误代码“PNIO_ERR_CONFIG_IN_UPDATE”或“PNIO_ERR_NO_FW_COMMUNICATION”。

成功下载后，PNIO_controller_open( ) 函数返回“PNIO_OK”错误代码。

说明

如果用户程序不注册该回调，则只要用户程序处于活动状态，就不能进行远程下载请求或重新组态。

语法

此事件类型无其它具体参数作为 PNIO_CBE_PRM
结构中“联合体”的一部分；请参见“PNIO_CBE_PRM（回调事件参数） (页 115)”部分。

4.11 等时实时模式 (IRT) 接口

概述

该接口由以下函数和回调事件组成：

- PNIO_CP_register_cbf( ) 函数
- 回调事件 PNIO_CP_CBE_STARTOP_IND
- 回调事件 PNIO_CP_CBE_OPFAULT_IND
- PNIO_CP_set_opdone( ) 函数

4.11.1 PNIO_CP_register_cbf( )（注册回调）

说明

该函数注册一个回调函数。

说明

回调函数仅可由 IO 控制器用户程序在 OFFLINE 模式下注册。

说明

PNIO_CP_CBE_OPFAULT_IND 回调事件类型必须在 PNIO_CP_CBE_STARTOP_IND 回调事件类型之前注册。

语法

```
PNIO_UINT32  PNIO_CP_register_cbf(
    PNIO_UINT32         CpHandle,          //in
    PNIO_CP_CBE_TYPE    CbeType,           //in
    PNIO_CP_CBF         Cbf                //in
);
```

参数

名称	说明
CpHandle	PNIO_controller_open( ) 或 PNIO_device_open( ) 的句柄
CbeType	要注册回调函数“Cbf”的回调事件类型；请参见“PNIO_CP_CBE_TYPE（回调事件类型） (页 131)”部分。
Cbf	要在回调事件“CbeType”到达后启动的回调函数。
	注意 函数指针不允许为空。

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_ALLREADY_DONE
- PNIO_ERR_INTERNAL
- PNIO_ERR_INVALID_CONFIG
- PNIO_ERR_PRM_CALLBACK
- PNIO_ERR_PRM_TYPE
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND

4.11.2 回调事件 PNIO_CP_CBE_STARTOP_IND（开始等时实时数据处理）

说明

PNIO_CP_CBE_STARTOP_IND 回调事件通知 IO-Base 用户程序 IRT 数据传送阶段结束，并且 DMA 将 IRT 输入数据从过程映像传送到主机存储器。

此后，所有的 IRT 输入数据都在主机存储器中。 这样，用户程序可以访问一致的 IRT 数据。

以下语法显示此事件的具体参数，将其作为 PNIO_CP_CBE_PRM 结构中“联合体”的一部分；请参见“PNIO_CP_CBE_PRM（回调事件参数） (页 131)”部分。

语法

```
typedef struct
{
    PNIO_UINT32          CpHandle;             //in
    PNIO_CYCLE_INFO      CycleInfo;            //in
} ATTR_PACKED PNIO_CP_CBE_PRM_STARTOP_IND;
```

参数

名称	说明
CpHandle	PNIO_controller_open( ) 或 PNIO_device_open( ) 的句柄
CycleInfo	有关当前周期的信息

注意
在 PNIO_CP_CBE_STARTOP_IND 事件期间，不得调用可能危及 IO-Base 用户程序实时功能的任何函数。 这里的函数指的是需要较长处理时间（如文件操作或屏幕显示）的函数。 请参见操作系统说明，找出在此情况中可能涉及的函数。 为了访问 IO-Base 用户接口的 IRT 数据，我们通常建议您限制使用这些函数。

4.11.3 PNIO_CP_set_opdone( )（等时实时数据处理结束）

说明

通过调用该函数，IO-Base 用户程序将已完成 IRT IO 数据的等时实时处理通知给 IO-Base 接口。 之后，IO-Base 接口启动使用 DMA 将 IRT
输出数据从主机存储器传送到过程映像的过程。

语法

```
PNIO_UINT32  PNIO_CP_set_opdone(
    PNIO_UINT32          CpHandle          //in
    PNIO_CYCLE_INFO     *CycleInfo;        //out
);
```

参数

名称	说明
CpHandle	PNIO_controller_open( ) 或 PNIO_device_open( ) 的句柄
CycleInfo	执行该调用的当前周期的相关信息。 如果指针为 0，则不返回任何信息。

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_WRONG_HND

4.11.4 回调事件 PNIO_CP_CBE_OPFAULT_IND（违反等时实时模式）

说明

PNIO_CP_CBE_OPFAULT_IND 回调事件将违反等时实时模式通知给 IO-Base 用户程序。 这意味着用户程序在收到 PNIO_CP_CBE_STARTOP_IND 回调事件后调用 PNIO_CP_set_opdone( ) 过晚。

以下语法显示此事件的具体参数，将其作为 PNIO_CP_CBE_PRM 结构中“联合体”的一部分；请参见“PNIO_CP_CBE_PRM（回调事件参数） (页 131)”部分。

语法

```
typedef struct
{
    PNIO_UINT32           CpHandle;           //in
    PNIO_CYCLE_INFO       CycleInfo;          //in
} ATTR_PACKED PNIO_CP_CBE_PRM_OPFAULT_IND;
```

参数

名称	说明
CpHandle	PNIO_controller_open( ) 或 PNIO_device_open( ) 的句柄
CycleInfo	有关当前周期的信息

4.12 用于通知新总线周期开始的接口

说明

该接口由以下回调事件组成：

PNIO_CP_CBE_NEWCYCLE_IND

4.12.1 回调事件 PNIO_CP_CBE_NEWCYCLE_IND（新总线周期）

说明

PNIO_CP_CBE_NEWCYCLE_IND 回调事件通知 IO-Base 用户程序新总线周期开始。只有使用 PNIO_CP_register_cbf( ) 注册了该事件后，才会通知该事件。

以下语法显示此事件的具体参数，将其作为 PNIO_CP_CBE_PRM 结构中“联合体”的一部分；请参见“PNIO_CP_CBE_PRM（回调事件参数） (页 131)”部分。

语法

```
typedef struct
{
    PNIO_UINT32          CpHandle;             //in
    PNIO_CYCLE_INFO      CycleInfo;            //in
} ATTR_PACKED PNIO_CP_CBE_PRM_NEWCYCLE_IND;
```

参数

名称	说明
CpHandle	PNIO_controller_open( ) 或 PNIO_device_open( ) 的句柄
CycleInfo	有关当前周期的信息

4.13 PROFIenergy 编程接口

4.13.1 PROFIenergy 编程接口概览

说明

在以下文本中，“PROFIenergy”缩写为“PE”。

如果在操作中有较长的中断时间，可以通过 IO 控制器将支持 PE 函数的 IO 设备转换为节能模式，并在中断结束后再次返回至正常的操作状态。
对于大量消耗电流的 IO 设备，这一操作可大幅度节省能源。 相关函数如下：

- PNIO_register_pe_cbf()

  此函数负责注册回调函数。 用于处理响应的回调函数必须由用户执行。

- PNIO_pe_cmd_req()
  用于 PE 作业的函数。 PE 作业基于作业 ID 进行选择。

- cbf_pe_cmd_conf()
  PE 作业结果由公共回调函数通知给用户。

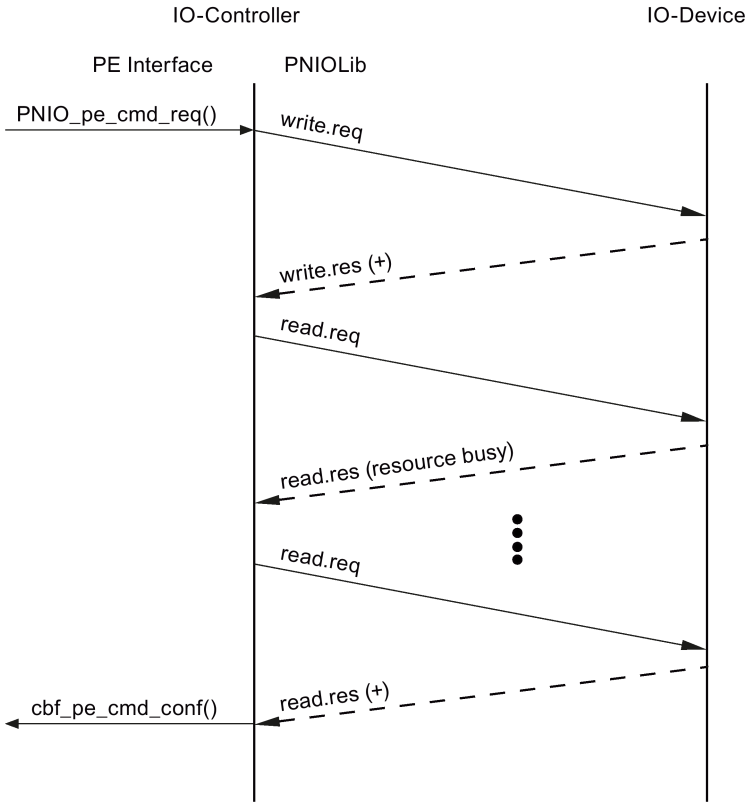
由 PNIOLib 将这些函数提供给用户程序。 PNIOLib 通过与 IO 设备进行写入记录和读取记录通信来处理 PE 作业。 每个 PE 作业由请求/响应对组成。
对于写入记录，会将请求发送至 IO 设备。 对于写入记录，将收到响应。
由于响应不会立即收到，必须重复（轮询）读取记录直到收到响应为止。
最大等待时间由常数 PNIO_PE_SERVICE_REQ_LIFETIME_DEFAULT 定义且为 10 秒。
PNIOLib 再次将 PE 作业结果提供给用户程序。

说明

ET200S PE 组态

使用 PE 函数前，需要相应地组态 ET200S PE。 请参见“PROFIenergy 设备（CP 1604/CP 1616） (页 110)”部分。

下图显示了函数的顺序：



上图说明： write.req 是由 PNIOLib 发送至 IO 设备的数据记录作业。 PE 作业嵌入数据记录中。write.res(+) 是来自 IO 设备的肯定响应，通知已收到数据记录作业。 使用数据记录作业 read.req 从 IO 设备获取 PE 作业的结果。如果结果不可用，IO 设备以“资源忙”的错误进行响应。重复读取记录作业 read.req，直到结果可用为止。（最多 10 秒）。

4.13.2 PROFIenergy 函数的描述

4.13.2.1 PNIO_register_pe_cbf()

说明

该函数注册一个 PE 确认回调函数。

说明

此回调函数可在任何 IO 控制器操作模式中注册。 但是，必须在 PNIO_controller_open( ) 后以及第一个 PNIO_pe_cmd_req() 函数调用前注册回调函数。

如果未进行注册，PNIO_pe_cmd_req() 函数返回错误： PNIO_ERR_SEQUENCE

语法

```
PNIO_UINT32 PNIO_register_pe_cbf(
 PNIO_UINT32 Handle,
 PNIO_PE_CBF cbf_pe_cmd_conf,
);
```

参数

名称	说明
Handle	PNIO_controller_open( ) 的句柄
cbf_pe_cmd_conf	回调函数的地址，该回调函数在来自 PNIOLib 的 PE 确认到达后启动。 **注意** 函数指针不允许为空。

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_WRONG_HND
- PNIO_ERR_PRM_CALLBACK
- PNIO_ERR_ALREADY_DONE

4.13.2.2 PNIO_pe_cmd_req()

说明

此函数用于将 PE 作业发送至 IO 设备。 它是控制器应用中所有 PE 请求的组函数。 由 PE 组回调函数 cbf_pe_cmd_conf 通知 PE 作业的结果（确认）。

如果 IO-Base 控制器用户程序想要使用此函数（即 PROFIenergy 功能），它必须已使用 PNIO_register_pe_cbf() 调用（PNIO_register_pe_cbf() (页 93)）注册回调函数。

可在表格 4-1 所有可能的 PE 作业的列表 (页 100)中找到所有可能的 PE 作业的概述。

语法

```
PNIO_UINT32 PNIO_pe_cmd_req(
 PNIO_UINT32 Handle,
 PNIO_ADDR *pAddr,
 PNIO_REF ReqRef,
 PNIO_PE_REQ_PRM *pPeReqPrm,
);
```

参数

名称	说明
Handle	PNIO_controller_open( ) 的句柄
pAddr	作业所对应的 IO 设备前端模块的逻辑地址。

名称	说明
ReqRef	由 IO-Base 控制器用户程序分配的作业引用。 **注意** IO-Base 控制器用户程序通过“ReqRef”引用能够区别不同的 PE 作业。 此参数可由 IO-Base 控制器用户程序分配任何值，并在回调事件中返回。
pPeReqPrm	指向“PNIO_PE_REQ_PRM”参数结构（与“PNIO_PE_PRM_xxx_REQ”参数结构相对应）的指针。 对于不需要任何参数的作业，仅评估 CmdId 和 CmdModifier。 请参见“PNIO_PE_REQ_PRM”结构的描述（PE 作业参数）；PNIO_PE_REQ_PRM (页 102) **注意** 将复制“pPeReqPrm”指向的存储器内容，也就是说从函数返回后，存储器可以再次使用。

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_WRONG_HND
- PNIO_ERR_SEQUENCE

4.13.2.3 回调事件 PNIO_PE_CBF（PE 作业结果）

说明

PNIO_PE_CBF 回调事件报告通过 PNIO_pe_cmd_req 调用先前发送的 PE 作业的结果。 回调函数必须由用户执行且通过 PNIO_register_pe_cbf() 调用注册。 请参见 5.13.2.3 部分

用户可为回调函数指定任何名称。 例如，在本手册中，使用名称“cbf_pe_cmd_conf”。

语法

原型（在 pniousrx.h 中定义）：

```
typedef void (* PNIO_PE_CBF)(
 PNIO_PE_CBE_PRM * pPeCbfPrm
);
```

定义（用于在应用程序中执行）：

```
void cbf_pe_cmd_conf(PNIO_PE_CBE_PRM *pPeCbfPrm)
{
 //PE confirmation processing …
 …
}
```

参数

名称	说明
pPeCbfPrm	指向 PE 确认参数的指针 各种回调事件具有相同的数据类型 PNIO_PE_CBE_PRM（请参见 PNIO_PE_CBE_PRM（回调事件参数） (页 97)），该数据类型使用“联合体”对各回调事件的各种参数进行分组。 **注意** 退出此回调函数后，“pPeCbfPrm”参数失效。 如有必要，必须事先保存这些参数。

返回值

无返回值。

4.13.3 PROFIenergy 数据类型描述

4.13.3.1 PNIO_PE_CBE_PRM（回调事件参数）

说明

各种 PE 回调事件具有相同的数据类型
PNIO_CBE_PRM，该数据类型使用“联合体”对各个 PE 作业的各种响应参数进行分组。

语法

```
typedef struct {
 PE_CBE_HDR pe_hdr;
 union {
 PNIO_PE_PRM_GENERIC_CONF_NEG RespNegative;
 PNIO_PE_PRM_START_PAUSE_CONF StartPause;
 PNIO_PE_PRM_END_PAUSE_CONF EndPause;
 PNIO_PE_PRM_PE_IDENTIFY_CONF PeIdentify;
 PNIO_PE_PRM_PEM_STATUS_CONF PemStatus;
 PNIO_PE_PRM_Q_MODE_LIST_ALL_CONF ModeList;
 PNIO_PE_PRM_Q_MODE_GET_MODE_CONF ModeGet;
 PNIO_PE_PRM_Q_MSRMT_GET_LIST_CONF MeasurementList;
 PNIO_PE_PRM_Q_MSRMT_GET_VAL_CONF MeasurementValue;
 } pe_prm;
} ATTR_PACKED PNIO_PE_CBE_PRM;
```

参数

名称	说明
pe_hdr	pe_hdr 结构包含由所有 PE 回调事件共享的参数。 具体结构参见“PE_CBE_HDR（PROFIenergy 回调事件标头） (页 99)”部分
pe_prm	作业特定的参数结构的“联合体”。 正确参数结构的选择基于 pe_hdr 结构中的元素 CmdId 和 struct_id。

下表给出了 CmdId 和 struct_id 参数结构的依存关系：

pe_prm 参数结构	CmdId struct_id
PNIO_PE_PRM_GENERIC_CONF_NEG	CmdId 无效 struct_id 无效
PNIO_PE_PRM_START_PAUSE_CONF	PNIO_PE_CMD_START_PAUSE PE_CNF_STRUCT_ID_UNIQUE
PNIO_PE_PRM_END_PAUSE_CONF	PNIO_PE_CMD_END_PAUSE PE_CNF_STRUCT_ID_UNIQUE
PNIO_PE_PRM_PE_IDENTIFY_CONF	PNIO_PE_CMD_PE_IDENTIFY PE_CNF_STRUCT_ID_UNIQUE
PNIO_PE_PRM_PEM_STATUS_CONF	PNIO_PE_CMD_PEM_STATUS PE_CNF_STRUCT_ID_UNIQUE
PNIO_PE_PRM_Q_MODE_LIST_ALL_CONF	PNIO_PE_CMD_Q_MODES PE_CNF_STRUCT_ID_QMODE_LIST_ALL
PNIO_PE_PRM_Q_MODE_GET_MODE_CONF	PNIO_PE_CMD_Q_MODES PE_CNF_STRUCT_ID_QMODE_GET_MODE
PNIO_PE_PRM_Q_MSRMT_GET_LIST_CONF	PNIO_PE_CMD_Q_MSRMT PE_CNF_STRUCT_ID_QMSRMT_LIST_ALL
PNIO_PE_PRM_Q_MSRMT_GET_VAL_CONF	PNIO_PE_CMD_Q_MSRMT PE_CNF_STRUCT_ID_QMSRMT_GET_VAL

4.13.3.2 PE_CBE_HDR（PROFIenergy 回调事件标头）

说明

此结构描述由所有 PE 回调事件共享的“cbf_pe_cmd_conf”回调函数的参数。

语法

```
typedef struct {
 PNIO_PE_CMD_ENUM CmdId;
 PNIO_UNIT32 Handle;
 PNIO_ADDR Addr;
 PNIO_REF Ref;
 PNIO_ERR_STAT Err;
 PNIO_UINT16 state;
 PNIO_UINT16 struct_id;
 PNIO_UINT32 length;
} ATTR_PACKED PE_CBE_HDR;
```

参数

名称	说明
CmdId	PE 作业 ID（PNIO_PE_CMD_ENUM (页 100)）
Handle	“Handle”，通过 PNIO_pe_cmd_req() 传送
Addr	作业的前端模块的地址
参考	PE 作业的返回的 ReqRef
Err	执行作业有关的错误代码 **注意** “Err”是由 IO 设备传送的 PROFINET IO 错误代码，这里的错误是指在 IO 设备上执行此读取数据记录作业时发生的错误。
state	作业结果： 1= ok（作业成功） 2=err（无法执行作业）

名称	说明
struct_id	作业结果的结构 ID： 根据 CmdId，struct_id 精确定义响应参数的类型。 这种区分对于 PNIO_PE_CMD_Q_MODES 之类的“组作业”是必要的。 对于这些作业，CmdId 不唯一。 可能的组合在 PNIO_PE_CBE_PRM PNIO_PE_CBE_PRM（回调事件参数） (页 97)部分中介绍
length	后面的参数的长度（字节）

4.13.3.3 PNIO_PE_CMD_ENUM

说明

此枚举类型列出了 PE 作业。 这意味着请求和确认。

语法

```
typedef enum {
 PNIO_PE_CMD_RESERVED =0,
 PNIO_PE_CMD_START_PAUSE =1,
 PNIO_PE_CMD_END_PAUSE =2,
 PNIO_PE_CMD_Q_MODES =3,
 PNIO_PE_CMD_PEM_STATUS =4,
 PNIO_PE_CMD_PE_IDENTIFY =5,
 PNIO_PE_CMD_Q_MSRMT =16
} PNIO_PE_CMD_ENUM;
```

参数

表格 4- 1　所有可能的 PE 作业的列表

PE 作业	说明
PNIO_PE_CMD_START_PAUSE	启动节能模式
PNIO_PE_CMD_END_PAUSE	结束节能模式

PE 作业	说明
PNIO_PE_CMD_Q_MODES	根据 CmdModifier 返回节能模式的信息
PNIO_PE_CMD_PEM_STATUS	返回当前节能模式
PNIO_PE_CMD_PE_IDENTIFY	返回所有支持的 PE 作业的列表
PNIO_PE_CMD_Q_MSRMT	根据 CmdModifier 返回能耗信息

4.13.3.4 PNIO_PE_CMD_MODIFIER_ENUM

说明

此枚举类型列出了定义作业类型为“组作业”的精确作业所需的 PE 命令修饰符。 PE 命令修饰符与请求一起使用。 PE 作业修饰符是 PNIO_PE_REQ_PRM 结构的元素。
请参见“PNIO_PE_REQ_PRM (页 102)”部分

对于确认，通过数据结构标识符 (struc_id) 处理该函数。
请参见“PNIO_PE_CBE_PRM（回调事件参数） (页 97)”部分

说明

组作业是 PNIO_PE_CMD_Q_MODES（查询模式）和 PNIO_PE_CMD_Q_MSRMT（查询测量）作业。

语法

```
typedef enum {
 PE_CMD_MODIF_NOT_USED = 0,
 PE_CMD_MODIF_Q_MODE_LIST_ALL = 1,
 PE_CMD_MODIF_Q_MODE_GET_MODE = 2,
 PE_CMD_MODIF_Q_MSRMT_LIST_ALL = 1,
 PE_CMD_MODIF_Q_MSRMT_GET_VAL = 2,
} PNIO_PE_CMD_MODIFIER_ENUM;
```

参数

PE 作业修饰符	说明
PE_CMD_MODIF_NOT_USED	用于所有唯一作业。 这些作业不需要修饰符。
PE_CMD_MODIF_Q_MODE_LIST_ALL	用于 PNIO_PE_CMD_Q_MODES 作业。 返回所有可用节能模式的列表。
PE_CMD_MODIF_Q_MODE_GET_MODE	用于 PNIO_PE_CMD_Q_MODES 作业。 返回节能模式的数据。
PE_CMD_MODIF_Q_MSRMT_LIST_ALL	用于 PNIO_PE_CMD_Q_MSRMT 作业。 返回所有可用测量值的列表（测量值 ID 的列表）
PE_CMD_MODIF_Q_MSRMT_GET_VAL	用于 PNIO_PE_CMD_Q_MSRMT 作业。 返回测量值的数据。

4.13.3.5 PNIO_PE_REQ_PRM

说明

单独 PE 作业的 PE 参数以适当的参数结构传送。 所有这些参数结构在 PNIO_PE_REQ_PRM 结构中组合成一个“联合体”(union)。
下面几个部分中详细介绍了各个结构。

语法

```
typedef struct {
 PNIO_PE_CMD_ENUM CmdId;
 PNIO_PE_CMD_MODIFIER_ENUM CmdModifier;
 union {
```

各种 PE 作业参数结构的“联合体”（详细信息见相关部分）

```
 } rq;
} ATTR_PACKED PNIO_PE_REQ_PRM;
```

参数

名称	说明
CmdId	PE 作业 ID
CmdModifier	PE 作业修饰符
rq	所有 PE 作业参数结构的“联合体”

4.13.3.6 PNIO_PE_PRM_START_PAUSE_REQ

说明

用于触发运行中断开始的 PNIO_PE_CMD_START_PAUSE PE 作业的参数。

语法

```
typedef struct {
 PNIO_UINT32 time_ms;
} ATTR_PACKED PNIO_PE_PRM_START_PAUSE_REQ;
```

参数

名称	说明
time_ms	指定预期的中断持续时间（以毫秒为单位）。 如果此时间对于某个 IO 设备过短，则该 IO 设备不会暂停运行。

4.13.3.7 PNIO_PE_CMD_START_PAUSE_CONF

说明

用于确认运行中断开始的回调事件的参数。

语法

```
typedef struct {
 PNIO_UINT32 pe_mode_id;
} ATTR_PACKED PNIO_PE_PRM_START_PAUSE_CONF;
```

参数

名称	说明	
pe_mode_id	实现的模式	
	0x00	节能模式
	0x01...0xFE	供应商特定
	0xFF	功能就绪

4.13.3.8 PNIO_PE_PRM_END_PAUSE_CONF

说明

用于确认运行中断结束的回调事件的参数。

语法

```
typedef struct {
 PNIO_UINT32 time_ms;
} ATTR_PACKED PNIO_PE_PRM_END_PAUSE_CONF;
```

参数

名称	说明
time_ms	达到运行就绪状态所需的时间（以毫秒为单位）。

4.13.3.9 PNIO_PE_PRM_PE_IDENTIFY_CONF

说明

回调事件的参数，该事件返回所有支持的 PE 服务的列表。

语法

```
typedef struct {
 PNIO_UINT8 numServices;
 PNIO_UINT8 serviceId[254];
} ATTR_PACKED PNIO_PE_PRM_PE_IDENTIFY_CONF;
```

参数

名称	说明
numServices	后面的 serviceId 字段中的有效参数数目
serviceId[254]	支持的 PE 作业 ID 的字段

4.13.3.10 PNIO_PE_PRM_PEM_STATUS_CONF

说明

描述当前节能模式的回调事件的参数。

语法

```
typedef struct {
 PNIO_UINT8 PE_Mode_ID_Source;
 PNIO_UINT8 PE_Mode_ID_Destination;
 PNIO_UINT32 Time_to_operate;
 PNIO_UINT32 Remaining_time_to_destination;
 float Mode_Power_Consumption;
 float Energy_Consumption_to_Destination;
 float Energy_Consumption_to_operate;
} ATTR_PACKED PNIO_PE_CMD_PEM_STATUS_CONF;
```

参数

名称	说明	
PE_Mode_ID_Source	原始节能模式	
	0x00	节能模式（断电）
	0x01...0xFE	供应商特定
	0xFF	运行就绪
PE_Mode_ID_Destination	要达到的节能模式	
	0x00	节能模式
	0x01...0xFE	供应商特定
	0xFF	运行就绪
Time_to_operate	达到运行就绪状态前的最大时间	
Remaining_time_to_destination	达到目标状态前的剩余时间	
Mode_Power_Consumption	当前节能模式下的能耗 [kW] 如果未知： = 0.0（32 位浮点）	
Energy_Consumption_to_Destination	当前状态变化的能耗 [kWh]	
Energy_Consumption_to_operate	从当前节能状态到达到运行就绪状态的能耗 [kWh]	

4.13.3.11 PNIO_PE_PRM_Q_MODE_LIST_ALL_CONF

说明

回调事件的参数，该事件返回所有支持的节能模式的列表。

语法

```
typedef struct {
 PNIO_UINT8 numModeIds;
 PNIO_UINT8 peModeId[254];
} ATTR_PACKED PNIO_PE_PRM_Q_MODE_LIST_ALL_CONF;
```

参数

名称	说明	
numModeIds	后面的 peModeId[254] 字段中的有效参数数目	
peModeId[254]	带有所有支持的节能模式 ID 的字段 节能模式 ID：	
	0	节能模式（断电）
	1-0xFE	供应商特定
	0xFF	运行就绪

4.13.3.12 PNIO_PE_PRM_Q_MODE_GET_MODE_REQ

说明

发出某个节能模式所有详细信息的信号的请求的参数。

语法

```
typedef struct {
 PNIO_UINT8 peModeId;
 PNIO_UINT8 reserved;
} ATTR_PACKED PNIO_PE_PRM_Q_MODE_GET_MODE_REQ;
```

参数

名称	说明
peModeId;	节能模式 ID
reserved;	保留

4.13.3.13 PNIO_PE_PRM_Q_MODE_GET_MODE_CONF

说明

回调事件的参数，该事件返回特定节能模式的详细信息。

语法

```
typedef struct {
 PNIO_UINT8 peModeId;
 PNIO_UINT8 PE_Mode_Attributes;
 PNIO_UINT32 Time_min_Pause;
 PNIO_UINT32 Time_to_Pause;
 PNIO_UINT32 Time_to_operate;
 PNIO_UINT32 Time_min_length_of_stay;
 PNIO_UINT32 Time_max_length_of_stay;
 float Mode_Power_Consumption;
 float Energy_Consumption_to_pause;
 float Energy_Consumption_to_operate;
} ATTR_PACKED PNIO_PE_PRM_Q_MODE_GET_MODE_CONF;
```

参数

名称	说明
peModeId	节能状态 ID
PE_Mode_Attributes	位 0；可能值： 0 = 时间和性能/能源信息是静态的 1 = 时间和性能/能源信息是动态的 位 1 到 7：保留
Time_min_Pause	运行中断的最小持续时间。 以下三个参数之和（以毫秒为单位）： • Time_to_Pause • Time_to_operate • Time_min_length_of_stay
Time_to_Pause	达到节能模式前预期的持续时间。 只能是静态的。
Time_to_operate	达到运行就绪状态前预期的持续时间。 可以是静态或动态的。
Time_min_length_of_stay	节能模式的最小持续时间。
Time_max_length_of_stay	节能模式的最大持续时间。
Mode_Power_Consumption	当前节能模式下的能耗 [kW] 如果未定义： = 0.0（32 位浮点）
Energy_Consumption_to_pause	切换到节能模式的能耗 [kWh]
Energy_Consumption_to_operate	从当前节能状态到达到运行就绪状态的能耗 [kWh]

4.13.4 组态 PROFIenergy 设备

4.13.4.1 PROFIenergy 设备（CP 1604/CP 1616）

CP 1604/CP1616 作为 PROFIenergy 设备

自固件版本 2.6 起，通信处理器 CP 1604 和 CP 1616 支持能源管理配置文件“PROFIenergy”。 这样便可以使用 PROFIenergy 命令“暂停开始”关闭安装了这些通信处理器的 PC 主机，或使用 PROFIenergy 命令“暂停结束”重新打开 PC 主机。 为此，CP 1616 需要一个外部电源并且其硬件版本必须为 10。CP 1604 也需要一个外部电源并且其硬件版本必须为 8。

为此，用户程序中必须考虑以下内容：

如果要使用 PROFIenergy 功能，则在调用“PNIO_device_open()”函数时，还需要在“ExtPar”功能参数（第二个功能参数）中设置“PNIO_CEP_PE_MODE_ENABLE”标志。 此标志在头文件“pniolib/iobase/inc/pniobase.h”中定义，如下所示： #define PNIO_CEP_PE_MODE_ENABLE 0x00000020

通过将数据记录写入 IO 设备发送 PROFIenergy 命令。 通过读取数据记录获取设备的响应。 使用 IO 设备应用程序时将调用的回调函数为“cbf_rec_write”（IO 控制器发送 PROFIenergy 命令时）和“cbf_rec_read”（IO 控制器查询 IO 设备对命令的响应时）。 IO 控制器必须始终查询 IO 设备的响应。 这意味着每个 PROFIenergy 命令都是通过写入和读取数据记录进行处理。

有关 PROFIenergy 机制的详细信息，请参见示例程序“dev_easy_pe”。

ET200S PE 的 PE 组态数据记录

说明

PE 组态数据记录的结构是供应商特定的。

使用 PE 函数前，IO 设备需要相应进行组态。 组态基于 PE 组态数据记录和“PNIO_rec_write_req( )”请求（写入记录作业）。

说明

ET200S PE 设备的电源模块以如下所述的数据记录进行组态。

在“RecordIndex”= 3 时通过写入记录函数“PNIO_rec_write_req( )”将组态参数传送到 ET200S PE。

语法

```
#define PNIO_PE_CFG_DEV_NUM_SLOTS_MAX 8

typedef struct {

 PNIO_UINT8 slot_num;

 PNIO_UINT8 power_mode;

} ATTR_PACKED PNIO_PE_CFG_SLOT;


typedef struct {

 PNIO_UINT8 reserved1;

 PNIO_UINT8 num_cfg_slots;

 PNIO_PE_CFG_SLOT slot[PNIO_PE_CFG_DEV_NUM_SLOTS_MAX];

} ATTR_PACKED PNIO_PE_PRM_CONFIG_DEVICE_REQ;
```

参数

名称	说明
slot_num	要关闭的电源模块插槽，1 到 62 0: 此字段中的条目不相关
power_mode	0: 继续工作（不关闭模块） 1: 关闭
reserved1	未使用
num_cfg_slots	插槽-组态块的数目
slot [num_cfg_slots]	长度为“num_cfg_slots”的“插槽-组态块”(PNIO_PE_CFG_SLOT)的字段。 最大值： PNIO_PE_CFG_DEV_NUM_SLOTS_MAX 8

4.14 数据类型

下面介绍的数据类型用于 IO-Base 用户编程接口：

- 基本数据类型
- IO-Base 用户编程接口的特殊数据类型

4.14.1 基本数据类型

使用下列基本数据类型：

文件类型	定义
PNIO_UINT8	无符号字符型（8 位）
PNIO_UINT16	无符号短整型（16 位）
PNIO_UINT32	无符号长整型（32 位）
PNIO_REF	无符号长整型（32 位）

4.14.2 PNIO_MODE_TYPE（运行模式类型）

说明

PNIO_MODE_TYPE 数据类型包含所支持模式的 ID。

语法

```
typedef enum
{
    PNIO_MODE_OFFLINE,
    PNIO_MODE_CLEAR,
    PNIO_MODE_OPERATE
} PNIO_MODE_TYPE;
```

元素

名称	说明
PNIO_MODE_OFFLINE	OFFLINE 模式（“停止”）
PNIO_MODE_CLEAR	CLEAR 模式
PNIO_MODE_OPERATE	OPERATE 模式

4.14.3 PNIO_IO_TYPE（方向类型）

说明

地址数据方向的标识符。

语法

```
typedef enum
{
    PNIO_IO_IN,
    PNIO_IO_OUT,
    PNIO_IO_INOUT
} PNIO_IO_TYPE;
```

元素

名称	说明
PNIO_IO_IN	输入数据方向的标识符
PNIO_IO_OUT	输出数据方向的标识符
PNIO_IO_INOUT	双向数据方向的标识符

4.14.4 PNIO_ADDR（地址结构）

说明

PNIO_ADDR 地址结构用于使用此处介绍的所有函数进行寻址。

语法

```
typedef struct
{
    PNIO_ADDR_TYPE      AddrType;
    PNIO_IO_TYPE        IODataType;
    union
    {
        PNIO_UINT32     Addr;
        PNIO_UINT32     Reserved[5];
    } u;
} ATTR_PACKED PNIO_ADDR;
```

参数

名称	说明
AddrType	对于 IO 控制器，值必须总是为 PNIO_ADDR_LOG。
IODataType	输入或输出
Addr	地址
Reserved	为将来的扩展保留

4.14.5 PNIO_CBE_TYPE（回调事件类型）

说明

IO-Base 用户编程接口可识别以下回调事件类型：

回调事件类型	回调事件
PNIO_CBE_ALARM_IND	报警到达。*
PNIO_CBE_REC_READ_CONF	读数据记录作业的结果到达。*
PNIO_CBE_REC_WRITE_CONF	写数据记录作业的结果到达。*

回调事件类型	回调事件
PNIO_CBE_MODE_IND	本地模式发生变化。
PNIO_CBE_DEV_ACT_CONF	激活/取消激活 IO 设备的结果到达。
PNIO_CBE_CTRL_DIAG_CONF	表示诊断请求的结果
PNIO_CBE_CP_STOP_REQ	表示远程固件下载或重新组态请求
PNIO_CBE_START_LED_FLASH	激活 LED 的闪烁模式
PNIO_CBE_STOP_LED_FLASH	取消激活 LED 的闪烁模式
PNIO_CBE_REMA_READ_CONF	???
PNIO_CBE_IOSYSTEM_RECONFIG	???
PNIO_CBE_IFC_SET_ADDR_CONF	???
PNIO_CBE_IFC_REC_READ_CONF	???
PNIO_CBE_IFC_ALARM_IND	???

*

*此回调事件类型使用“PNIO_controller_open()”函数进行注册，无法在“PNIO_register_cbf()”函数中注册。

4.14.6 PNIO_CBE_PRM（回调事件参数）

说明

各种回调事件具有相同的数据类型
PNIO_CBE_PRM，该数据类型使用“联合体”对各回调事件的各种参数进行分组。

语法

```
typedef struct
{
    PNIO_CBE_TYPE     CbeType;                //in
    PNIO_UINT32       Handle;                 //in
    union
    {
        ...          /*Union over the various
        ...            callback event parameters
                       (details in relevant sections)*/
    };
} ATTR_PACKED PNIO_CBE_PRM;
```

参数

名称	说明
CbeType	回调事件
Handle	来自 PNIO_controller_open( ) 或 PNIO_device_open( ) 的句柄。

4.14.7 PNIO_CBF（PNIO 回调函数）

说明

使用回调事件参数 PNIO_CBE_PRM 来声明类型 PNIO_CBF 的常规 PNIO 回调函数。

语法

```
typedef void (* PNIO_CBF)(
    PNIO_CBE_PRM       *pCbfPrm
);
```

4.14.8 ExtPar（扩展参数）

说明

“ExtPar”的位用于参数分配。

在 32 个可能的位中，只有 PNIO_CEP_MODE_CTRL 和 PNIO_CEP_SLICE_ACCESS 位当前已定义。

当调用 PNIO_controller_open( ) 函数时，这些位将置位。

参数

标识符	说明
PNIO_CEP_MODE_CTRL	置位此位的 IO-Base 用户程序将被授权更改自己的模式。 该位必须始终置位。
PNIO_CEP_SLICE_ACCESS	该位激活部分访问，也就是说，可以写入或读取子模块的任何子区域。 并非所有产品都支持该 ID。 有关详细信息，请参见相关产品的自述。

示例 1

以下示例置位 PNIO_CEP_MODE_CTRL 位：

```
PNIO_controller_open(
    ..., PNIO_CEP_MODE_CTRL, ...
);
```

示例 2

以下示例置位 PNIO_CEP_MODE_CTRL 和 PNIO_CEP_SLICE_ACCESS 位：

```
PNIO_controller_open(
    ..., PNIO_CEP_MODE_CTRL | PNIO_CEP_SLICE_ACCESS, ...
);
```

部分访问示例

一个 4 字节输出模块的起始地址为 2。从地址 3 开始写入 2 个字节。

4.14.9 PNIO_DEV_ACT_TYPE（用于激活和取消激活 IO 设备的类型）

说明

用于取消激活和激活 IO 设备的标识符。

语法

```
typedef enum
{
    PNIO_DA_FALSE,
    PNIO_DA_TRUE
} PNIO_DEV_ACT_TYPE;
```

元素

名称	说明
PNIO_DA_FALSE	禁用
PNIO_DA_TRUE	激活

4.14.10 PNIO_IOXS（IO 数据的状态）

说明

用于描述 IO 数据状态的标识符。

语法

```
typedef enum
{
    PNIO_S_GOOD,
    PNIO_S_BAD
} PNIO_IOXS;
```

元素

名称	说明
PNIO_S_GOOD	IO 数据有效。
PNIO_S_BAD	IO 数据无效。

4.14.11 PNIO_CTRL_DIAG（诊断请求）

说明

此结构用于设置诊断请求。 它与 PNIO_ctrl_diag_req( ) 函数和 PNIO_CBE_CTRL_DIAG_CONF 回调事件配合使用。

语法

```
typedef struct
{
    PNIO_CTRL_DIAG_ENUM         DiagService;
    union
    {
        PNIO_UINT32             Reserved1[8];
        PNIO_ADDR               Addr;
    }u;
    PNIO_REF                    ReqRef;
    PNIO_UINT32                 Reserved2;
} ATTR_PACKED PNIO_CTRL_DIAG;
```

元素

名称	说明
DiagService	解释诊断应答 - 依据 PNIO_CTRL_DIAG_ENUM 诊断服务的诊断服务；请参见“PNIO_CTRL_DIAG_ENUM（诊断服务） (页 120)”部分。
Reserved1	为将来的扩展保留
Addr	用于对 IO 设备状态进行诊断查询的子模块的地址；适用于 DiagService = PNIO_CTRL_DIAG_DEVICE_STATE。
ReqRef	用户为该诊断请求提供的唯一引用。
RESERVED2	为将来的扩展保留

4.14.12 PNIO_CTRL_DIAG_ENUM（诊断服务）

说明

此枚举类型列出了诊断服务。对支持的服务进行描述。

语法

```
typedef enum{
 PNIO_CTRL_DIAG_RESERVED = 0,
 PNIO_CTRL_DIAG_CONFIG_SUBMODULE_LIST = 1,
 PNIO_CTRL_DIAG_DEVICE_STATE = 2,
 PNIO_CTRL_DIAG_CONFIG_IOROUTER_PRESENT = 3,
 PNIO_CTRL_DIAG_CONFIG_OUTPUT_SLICE_LIST= 4,
 #ifndef PNIO_SOFTNET /* not supported for PNIO SOFTNET */
 PNIO_CTRL_DIAG_CONFIG_NAME_ADDR_INFO = 5 /* FW V2.5.2.0 and higher */
 #endif /* PNIO_SOFTNET */
 PNIO_CTRL_DIAG_GET_COMM_COUNTER_DATA = 6,
 PNIO_CTRL_DIAG_RESET_COMM_COUNTERS = 7 /* FW V2.5.2.0 and greater */
 PNIO_CTRL_DIAG_DEVICE_DIAGNOSTIC = 8
} PNIO_CTRL_DIAG_ENUM;
```

元素

诊断服务	说明
PNIO_CTRL_DIAG_CONFIG_SUBMODULE_LIST	提供所有已组态子模块的列表 - PNIO_CBE_CTRL_DIAG_CONF 回调事件中的诊断应答缓冲区“pDataBuffer”包含类型为 PNIO_CTRL_DIAG_CONFIG_ SUBMODULE 的元素；请参见“PNIO_CTRL_DIAG_CONFIG_SUBMODULE（子模块组态信息） (页 122)”部分。
PNIO_CTRL_DIAG_DEVICE_STATE	诊断查询提供有关 IO 设备状态的信息。PNIO_CBE_CTRL_DIAG_CONF 回调事件中的诊断应答缓冲区“pDataBuffer”包含类型为 PNIO_CTRL_DIAG_DEVICE 的元素；请参见“PNIO_CTRL_DIAG_DEVICE_STATE (页 134)”部分。 PN 驱动程序不支持此服务。
PNIO_CTRL_DIAG_CONFIG_ IOROUTER_PRESENT	诊断请求提供有关是否为 IO 路由预留输出位片的信息；请参见“PNIO_CTRL_DIAG_CONFIG_IOROUTER_PRESENT（关于 IO 路由器的诊断请求） (页 124)”部分。 PN 驱动程序不支持此服务。
PNIO_CTRL_DIAG_CONFIG_ OUTPUT_SLICE_LIST	诊断请求提供有关为 IO 路由保留的输出位片的大小的信息；请参见“PNIO_CTRL_DIAG_CONFIG_OUTPUT_SLICE_LIST（关于 IO 路由器的诊断查询） (页 125)”部分。 PN 驱动程序不支持此服务。
PNIO_CTRL_DIAG_CONFIG_NAME_ADDR_INFO	诊断查询将返回已组态的设备名称、设备类型和 IP 地址参数（IP 地址、掩码和默认路由器）。结果将以“PNIO_CTRL_DIAG_CONFIG_NAME_ADDR_INFO_DATA”结构返回；请参见“PNIO_CTRL_DIAG_CONFIG_NAME_ADDR_INFO_DATA (页 126)”部分。

诊断服务	说明
PNIO_CTRL_DIAG_GET_COMM_COUNTER_DATA	此诊断查询以以太网接口的统计数据形式返回有关数据传输质量的信息。请参见“PNIO_CTRL_COMM_PORT_COUNTER_DATA 和 PNIO_CTRL_COMM_COUNTER_DATA (页 128)”结构的说明。 PN 驱动程序不支持此服务。
PNIO_CTRL_DIAG_RESET_COMM_COUNTERS	此诊断查询以以太网接口的统计数据形式返回有关数据传输质量的信息。请参见“PNIO_CTRL_COMM_PORT_COUNTER_DATA 和 PNIO_CTRL_COMM_COUNTER_DATA (页 128)”结构的说明。 此外，统计数据的计数器将在读出数据后复位为 0。 PN 驱动程序不支持此服务。
PNIO_CTRL_DIAG_DEVICE_DIAGNOSTIC	诊断请求将返回关于 AR 中止的详细信息。该结果将通过“PNIO_CTRL_DIAG_DEVICE_DIAGNOSTIC”结构返回；请参见“PNIO_CTRL_DIAG_DEVICE_DIAGNOSTIC (页 127)”部分。

4.14.13 PNIO_CTRL_DIAG_CONFIG_SUBMODULE（子模块组态信息）

说明

该结构提供有关已组态子模块的信息。 针对诊断请求 PNIO_CTRL_DIAG_CONFIG_SUBMODUL_LIST 返回。

语法

```
typedef struct
{
    PNIO_ADDR           Address;
    PNIO_UINT32         DataLength;
    PNIO_DATA_TYPE      DataType;
    PNIO_COM_TYPE       ComType;
    PNIO_UINT32         Api;
    PNIO_UINT32         ReductionFactor;
    PNIO_UINT32         Phase;
    PNIO_UINT32         CycleTime;
    PNIO_UINT32         Reserved1[2];
    PNIO_UINT32         StatNo;
    PNIO_UINT32         Slot;
    PNIO_UINT32         Subslot;
    PNIO_UINT32         Reserved2[2];
} ATTR_PACKED PNIO_CTRL_DIAG_CONFIG_SUBMODULE;
```

元素

名称	说明
地址	子模块的逻辑地址
DataLength	子模块的数据长度
	注意 输入长度为 0 的子模块的逻辑地址是诊断地址。 有关诊断地址的含义，请参见 STEP 7 的文档。
DataType	指定 IO 数据的类型。
ComType	指定是否对子模块使用直接数据交换。
Api	指定子模块所属的 Api。
ReductionFactor	“ReductionFactor”指定发送数据的频率。 例如，ReductionFactor 为 2 时，每 2 * CycleTime 发送一次数据。
Phase	如果“ReductionFactor”> 1，这指定一个周期内的发送时间； 值范围： 0 到 ReductionFactor – 1
CycleTime	发送时钟（31.25 µs 的倍数）
StatNo	插入子模块的站的编号。
Slot	从设备角度看的模块编号

名称	说明
Subslot	从设备角度看的子模块编号
Reserved1	为将来的扩展保留
Reserved2	为将来的扩展保留

4.14.14 PNIO_CTRL_DIAG_CONFIG_IOROUTER_PRESENT（关于 IO 路由器的诊断请求）

说明

PNIO_CTRL_DIAG_CONFIG_IOROUTER_PRESENT
诊断请求提供有关是否将子模块输出位分配给外部 IO 控制器的信息。

通知来自于在 PNIO_register_cbf( ) 中注册的回调事件类型 PNIO_CBE_TYPE = PNIO_CBE_CTRL_DIAG_CONF 的回调函数（另请参见 PNIO_CTRL_DIAG 和 PNIO_CTRL_DIAG_ENUM）。

回调事件结构 PNIO_CBE_CTRL_DIAG_CONF
中的参数“DiagDataBuffer”和“DiagDataBufferLen”（请参见“回调事件 PNIO_CBE_CTRL_DIAG_CONF（指示诊断请求的结果）
(页 81)”部分）返回请求的结果。

参数

DiagDataBufferLen	说明
0	IO 路由器未组态，输出位没有限制。
不等于 0	以参数 `pDiagReq -> DiagService = PNIO_CTRL_DIAG_CONFIG_OUTPUT_SLICE_LIST` 调用 PNIO_ctrl_diag_req( ) 函数，将可用于本地 IO 控制器用户程序的输出位片通知给用户。

4.14.15 PNIO_CTRL_DIAG_CONFIG_OUTPUT_SLICE_LIST（关于 IO 路由器的诊断查询）

说明

如果由于 IO 路由将输出位片分配给外部 IO 控制器，则本地 IO 控制器用户程序无法再写入这些输出位片。

PNIO_CTRL_DIAG_CONFIG_OUTPUT_SLICE_LIST 诊断查询提供有关本地 IO 控制器用户程序可以写入的输出位的信息。
“PNIO_CTRL_DIAG_CONFIG_IOROUTER_PRESENT（关于 IO 路由器的诊断请求）(页 124)”部分介绍了 PNIO_CTRL_DIAG_CONFIG_IOROUTER_PRESENT 诊断请求的扩展。

通知来自于在 PNIO_register_cbf( ) 中注册的回调事件类型 PNIO_CBE_TYPE = PNIO_CBE_CTRL_DIAG_CONF 的回调函数（另请参见 PNIO_CTRL_DIAG 和 PNIO_CTRL_DIAG_ENUM）。

可用于本地 IO 控制器用户程序的输出位片以下面介绍的 PNIO_CTRL_DIAG_CONFIG_OUTPUT_SLICE 数据结构的形式进行报告。多个输出位片的数据结构依次排列。

回调事件结构 PNIO_CBE_CTRL_DIAG_CONF 中的参数“DiagDataBuffer”和“DiagDataBufferLen”（请参见“回调事件 PNIO_CBE_CTRL_DIAG_CONF（指示诊断请求的结果）(页 81)”部分）返回请求的结果。

说明

对比只能分配到单个 IO 控制器的输出位，输入位总是可以由两个 IO 控制器读取。因此不需要信息函数。

语法

```
typedef struct
{
    PNIO_ADDR        Address;
    PNIO_UINT16      BitOffset;
    PNIO_UINT16      BitLength;
} ATTR_PACKED PNIO_CTRL_DIAG_CONFIG_OUTPUT_SLICE;
```

元素

名称	说明
地址	子模块的逻辑地址
BitOffset	指定为位数
BitLength	指定为位数

示例 1

IO 路由将使长度为 8 位的子模块的输出位片 2 到 5 可供外部 IO 控制器进行输出。

对于两个剩余的输出位片（位 0、1 和位
6、7），诊断请求返回两次具有相同子模块逻辑地址和以下内容的上述结构：

剩余 输出位片	BitOffset	BitLength
1	0	2
2	6	2

示例 2

长度为 8 位的非路由子模块完全可供本地 IO 控制器用户程序使用。
诊断请求返回“BitOffset”= 0 和“BitLength”= 8。

4.14.16 PNIO_CTRL_DIAG_CONFIG_NAME_ADDR_INFO_DATA

说明

该结构包括控制器的 PROFINET 网络参数，如设备名称、设备类型和
地址参数（IP 地址、掩码和默认路由器）。 通过 STEP 7
进行组态，将这些参数传输到控制器中。

语法

```
typedef struct {
 PNIO_UINT8 name[256];
 PNIO_UINT8 TypeOfStation[256];
 PNIO_UINT32 ip_addr;
 PNIO_UINT32 ip_mask;
 PNIO_UINT32 default_router;
} ATTR_PACKED PNIO_CTRL_DIAG_CONFIG_NAME_ADDR_INFO_DATA;

 PNIO_UINT32 PNIO_CODE_ATTR SERV_CP_set_type_of_station (PNIO_UINT32 CpIndex, char*
TypeName);
```

参数

参数	说明
name[256]	模块的 PROFINET 站名称
TypeOfStation[256]	PROFINET 设备类型
ip_addr	模块的 IP 地址
ip_mask	模块的 IP 子网掩码
default_router	模块的 IP 默认网关

4.14.17 PNIO_CTRL_DIAG_DEVICE_DIAGNOSTIC

说明

“PNIO_CTRL_DIAG_DEVICE_DIAGNOSTIC”结构返回 IO 设备的诊断信息和状态。

要执行请求，必须通过调用 PNIO_ctrl_diag_req() 函数，在“u.Addr”参数中指定 IO 设备子模块的逻辑地址。

调用“PNIO_CBE_CTRL_DIAG_CONF”回调事件时，将返回请求结果。参数“DiagDataBuffer”包含结构“PNIO_CTRL_DIAG_DEVICE_DIAGNOSTIC”。通过“DiagDataBufferLen”指定长度。

语法

```
typedef struct {
```

```
 PNIO_DEV_ACT_TYPE Mode;

 PNIO_UINT16 ErrorCause;

 PNIO_UINT8 ReasonCode;

 PNIO_UINT8 AdditionalInfo[10];

} ATTR_PACKED PNIO_CTRL_DIAG_DEVICE_DIAGNOSTIC_DATA;
```

元素

名称	说明
Mode	IO 设备 PNIO_DEV_ACT_TYPE 的当前状态；请参见“PNIO_DEV_ACT_TYPE（用于激活和取消激活 IO 设备的类型） (页 118)”部分。
ErrorCause	IO 设备的详细诊断信息
ReasonCode	IO 设备的详细诊断信息
AdditionalInfo[10]	IO 设备的详细诊断信息

4.14.18 PNIO_CTRL_COMM_PORT_COUNTER_DATA 和 PNIO_CTRL_COMM_COUNTER_DATA

说明

这些结构以以太网接口的统计数据形式返回有关数据传输质量的信息。

语法

```
typedef struct {
 PNIO_UINT32 SndNoError;
 PNIO_UINT32 SndCollision;
 PNIO_UINT32 SndOtherError;
 PNIO_UINT32 RcvNoError;
 PNIO_UINT32 RcvResError;
 PNIO_UINT32 RcvRejected;
} ATTR_PACKED PNIO_CTRL_COMM_PORT_COUNTER_DATA;

typedef struct {
 PNIO_CTRL_COMM_PORT_COUNTER_DATA Port[4];
 PNIO_UINT16 NumberOfPorts;
} ATTR_PACKED PNIO_CTRL_COMM_COUNTER_DATA;
```

说明

如果 NumberOfPorts = 3，则 Port[3] 的内容不相关。

参数 (PNIO_CTRL_COMM_PORT_COUNTER_DATA)

参数	说明
SndNoError	帧已无错误发送
SndCollision	尝试发送失败（冲突）
SndOtherError	尝试发送失败
RcvNoError	帧已无错误接收
RcvResError	接收的帧不正确
RcvRejected	已拒接帧

参数 (PNIO_CTRL_COMM_COUNTER_DATA)

参数	说明
Port[4]	端口 1/-4 的统计数据
NumberOfPorts	以太网接口的数量（可以是 3（工业 PC）或 4 (CP 1616/CP 1604)）

4.14.19 PNIO_DATA_TYPE（IO 数据类型）

说明

此枚举类型指定了子模块的数据类型。 它用于 PNIO_CTRL_DIAG_CONFIG_SUBMODULE 结构。

语法

```
typedef enum
{
    PNIO_DATA_RT        = 0;
    PNIO_DATA_IRT       = 1
}PNIO_DATA_TYPE
```

元素

名称	说明
PNIO_DATA_RT	RT 数据
PNIO_DATA_IRT	IRT 数据

4.14.20 PNIO_COM_TYPE（传送类型）

说明

此枚举类型指定了子模块的通信类型。 它用于 PNIO_CTRL_DIAG_CONFIG_SUBMODULE 结构。

语法

```
typedef enum
{
    PNIO_COM_UNICAST              = 0x0,
    PNIO_COM_DIRECT_DATA_EX       = 0x1
}PNIO_COM_TYPE
```

元素

名称	说明
PNIO_COM_UNICAST	在 IO 设备和 IO控制器间传送
PNIO_COM_DIRECT_DATA_EX	直接数据交换传送

4.14.21 PNIO_CP_CBE_TYPE（回调事件类型）

说明

IO-Base 用户编程接口可识别以下回调事件类型：

回调事件类型	表示的信息
PNIO_CP_CBE_STARTOP_IND	等时实时数据处理开始
PNIO_CP_CBE_OPFAULT_IND	违背等时实时模式
PNIO_CP_CBE_NEWCYCLE_IND	新总线周期开始

这些回调事件只能由 IO 控制器或 IO 设备注册。

如果要同时实现 IO 控制器和 IO 设备，则需要在一个 IO-Base 用户程序中同时寻址两个设备。

4.14.22 PNIO_CP_CBE_PRM（回调事件参数）

说明

各种回调事件具有相同的数据类型 PNIO_CP_CBE_PRM，该数据类型使用“联合体”对各回调事件的各种参数进行分组。

语法

```
typedef struct
{
    PNIO_CP_CBE_TYPE    CbeType;
    PNIO_UINT32         CpIndex;
    union
    {
        ...           /*Union over the various
        ...             callback event parameters
                        (details in relevant sections)*/
    }u;
} ATTR_PACKED PNIO_CP_CBE_PRM;
```

参数

名称	说明
CbeType	回调事件
CpIndex	来自 PNIO_controller_open( ) 或 PNIO_device_open( ) 的 CpIndex

4.14.23 PNIO_CP_CBF（PNIO 回调函数）

说明

使用回调事件参数 PNIO_CP_CBE_PRM 来声明类型 PNIO_CP_CBF 的常规 PNIO 回调函数。

语法

```
typedef void (* PNIO_CP_CBF)(
    PNIO_CP_CBE_PRM       *pCbfPrm
);
```

4.14.24 PNIO_CYCLE_INFO（有关当前周期的信息）

说明

该结构包含有关当前周期的信息，并允许标识以下内容：

- 分辨率为 10 ns 的高精度计数器（“ClockCount”）
- 中断丢失（“CycleCount”值在每个周期的变化不同）
- “CountSinceCycleStart”返回周期开始与调用函数（返回 PNIO_CYCLE_INFO 结构）之间的时间。
- PNIO_CP_CBE_STARTOP_IND 回调事件与 PNIO_CP_set_opdone( ) 函数调用之间的时间（PNIO_CP_CBE_STARTOP_IND 回调事件的输入值“CountSinceCycleStart”与 PNIO_CP_set_opdone( ) 函数返回值之间的差值）

它用作回调事件的输入参数：

- PNIO_CP_CBE_STARTOP_IND
- PNIO_CP_CBE_OPFAULT_IND
- PNIO_CP_CBE_NEWCYCLE_IND

该结构用作函数 PNIO_CP_set_opdone( ) 的返回值。

语法

```
typedef struct
{
    PNIO_UINT32     CycleCount;
    PNIO_UINT32     ClockCount;
    PNIO_UINT32     CountSinceCycleStart;
} ATTR_PACKED PNIO_CYCLE_INFO;
```

参数

名称	说明
CycleCount	“CycleCount”值在每个周期以对应于周期持续时间的值（以 31.25 µs 为基数）为增量递增。 值范围为 0 到 216-1；数据类型 UINT32 用于将来的扩展。 **示例** 周期为 1 ms 时，值每次增加 32。
ClockCount	CP 上自由运行的硬件计数器，分辨率为 10 ns（32 位宽）。
CountSinceCycleStart	“CountSinceCycleStart”值指定从上一个周期开始以来的时间（以 10 ns 为基数）。 该值为 32 位宽。

4.14.25 PNIO_CTRL_DIAG_DEVICE_STATE

说明

PNIO_CTRL_DIAG_DEVICE_STATE 结构提供有关 IO 设备状态的信息。

要开始请求，必须通过在“pDiagReq”指向的 PNIO_CTRL_DIAG 结构的“u.Addr”元素中调用 PNIO_ctrl_diag_req( ) 函数来传送 IO 设备子模块的逻辑地址。

PNIO_CBE_CTRL_DIAG_CONF 回调事件中的“DiagDataBuffer”和“DiagDataBufferLen”参数（请参见“回调事件 PNIO_CBE_CTRL_DIAG_CONF（指示诊断请求的结果） (页 81)”部分）返回请求的结果 - PNIO_CTRL_DIAG_DEVICE 结构。

语法

```
typedef struct{

 PNIO_DEV_ACT_TYPE Mode;

 PNIO_UINT32 DiagState;

 PNIO_UINT32 Reason;

 PNIO_UINT32 Reserved1[12];

} ATTR_PACKED PNIO_CTRL_DIAG_DEVICE;
```

元素

名称	说明
Mode	IO 设备“PNIO_DEV_ACT_TYPE”的当前状态；请参见“PNIO_DEV_ACT_TYPE（用于激活和取消激活 IO 设备的类型） (页 118)”部分。
DiagState	与此 IO 设备建立的连接的当前状态 - 可能值如下： • PNIO_CTRL_DEV_DIAG_OFFLINE • PNIO_CTRL_DEV_DIAG_ADDR_INFO_OK • PNIO_CTRL_DEV_DIAG_AR_CONNECTING • PNIO_CTRL_DEV_DIAG_AR_IN_DATA 头文件“pniousrx.h”中说明了各个值的含义。
Reason	如果连接建立中出错，在此处输入错误原因。在“pniousrx.h”头文件的“PNIO_CTRL_REASON_...”常量中可找到可能值。

IO 设备功能快速入门

5

本章推荐了用 C/C++ 编程语言基于 IO-Base
设备用户编程接口创建用户程序的分布过程。

首先要熟悉有关 PROFINET IO 的基本知识。 逐渐完成各个步骤，并最终编写您自己的 IO-Base 设备用户程序。

5.1 步骤

说明

按照以下步骤，可快速高效地创建 IO-Base 设备用户程序：

步骤	说明
1	了解有关 PROFINET 的基本知识。 可通过阅读《PROFINET 系统说明》手册来进行了解。
2	熟悉 IO 设备的 IO-Base 设备用户编程接口的基本属性。 可通过阅读本手册的“IO 设备的 IO-Base 用户编程接口概述 (页 141)”部分来进行熟悉。
3	熟悉以下文件，了解已经为以下步骤中提供了哪些支持，以及如何利用这些支持。 这些文件如下： • 包含附加信息和最新修改说明的自述文件。 • IO-Base 设备用户编程接口的 C 头文件： – 包含函数和数据结构的“pniousrx.h”。 – 包含错误代码与返回代码的“pnioerrx.h” – IO-Base 的全局定义“pniobase.h” • 用于连接至 IO-Base 设备用户程序的“libpniousr”库。 • 示例程序和相应的数据库。
4	熟悉“../Examples/deveasy”子目录中示例程序“DevEasy”的源文本，并查看“IO 设备的 IO-Base 用户编程接口概述 (页 141)”部分中的函数和数据结构。

步骤	说明
5	根据系统组态和 GSDML 文件，了解需要发送和接收哪些数据（IO 数据、数据记录、报警），以及涉及哪些 IO 设备和 IO 控制器。
6	完成 STEP 7 或 NCM 组态并将其下载到相关设备中。
7	修改提供的示例程序“DevEasy”，使其可在您的系统中运行。 通过这种方式，您可以了解相关的重要技术，而无需再自行开发。 编译并加入修改后的示例程序，在您准备好的系统上对其进行测试。
8	现在创建您自己的包含全部功能的 IO-Base 设备用户程序。

5.2 产品功能概述

说明

下表概要介绍了可在各 SIMATIC NET 产品中应用的 IO 设备功能。

函数组和名称	SIMATIC NET 产品	
	SOFTNET PN IO (RT)	CP 1616/CP 1604 (RT + IRT) 带有版本 V2.0 及更高版本的 DK-16xx
管理函数		
PNIO_device_open( )	–	x
PNIO_device_close( )	–	x
PNIO_device_start( )	–	x
PNIO_device_stop( )	–	x
PNIO_CBF_DEVICE_STOPPED( ) 回调函数	–	x
IO 设备组态的接口		
PNIO_api_add( )	–	x
PNIO_api_remove( )	–	x
PNIO_CBF_PULL_PLUG_CONF( ) 回调函数	–	x
PNIO_mod_plug( )	–	x

函数组和名称	SIMATIC NET 产品	
PNIO_mod_pull( )	–	x
PNIO_sub_plug( )	–	x
PNIO_sub_pull( )	–	x
PNIO_sub_plug_ext( )	–	x
PNIO_sub_plug_ext_IM( )	–	x
用于在 IO 设备上写入 IO 数据的接口		
PNIO_initiate_data_write( )	–	x
PNIO_CBF_DATA_WRITE( ) 回调函数	–	x
PNIO_initiate_data_write( )	–	x
用于在 IO 设备上读取 IO 数据的接口		
PNIO_initiate_data_read( )	–	x
PNIO_initiate_data_read_ext( )	–	x
PNIO_CBF_DATA_READ( ) 回调函数	–	x
用于在 IO 设备上读取/写入数据记录的接口		
PNIO_CBF_REC_READ( ) 回调函数	–	x
PNIO_CBF_REC_WRITE( ) 回调函数	–	x
用于在 IO 设备上管理诊断数据的接口		
PNIO_build_channel_properties( )	–	x
PNIO_diag_channel_add( )	–	x
PNIO_diag_channel_remove( )	–	x
PNIO_diag_generic_add( )	–	x
PNIO_diag_generic_remove( )	–	x
IO 设备的报警接口		
PNIO_process_alarm_send( )	–	x
PNIO_diag_alarm_send( )	–	x
PNIO_ret_of_sub_alarm_send( )	–	x
PNIO_CBF_REQ_DONE( ) 回调函数	–	x
用于建立和终止 IO 设备连接的接口		
PNIO_device_ar_abort( )	–	x

函数组和名称	SIMATIC NET 产品	
PNIO_set_appl_ state_ready( )	–	x
PNIO_CBF_CHECK_IND( ) 回调函数	–	x
PNIO_CBF_AR_CHECK_IND( ) 回调函数	–	x
PNIO_CBF_AR_INFO_IND( ) 回调函数	–	x
PNIO_CBF_AR_INDATA_IND( ) 回调函数	–	x
PNIO_CBF_AR_ABORT_IND( ) 回调函数	–	x
PNIO_CBF_AR_OFFLINE_IND( ) 回调函数	–	x
PNIO_CBF_APDU_STATUS_IND( ) 回调函数	–	x
PNIO_CBF_CBF_PRM_END_IND( ) 回调函数	–	x
用 LED 发出指示的站		
回调函数 PNIO_CBF_START_LED_FLASH( )	–	x
回调函数 PNIO_CBF_STOP_LED_FLASH( )	–	x
常规		
回调函数 PNIO_CBF_CP_STOP_REQ( )	–	x
等时实时模式 (IRT) 接口		
PNIO_CP_register_cbf( )	–	x
PNIO_CP_CBE_STARTOP_IND( ) 回调事件	–	x
PNIO_CP_CBE_OPFAULT_IND( ) 回调事件	–	x
PNIO_CP_set_opdone( )	–	x
特定于 CP 1616/CP 1604 的函数		
PNIO_CP_set_appl_watchdog( )	–	x
PNIO_CP_trigger_appl_watchdog( )	–	x
回调函数 PNIO_CBF_APPL_WATCHDOG( )	–	x

IO 设备的 IO-Base 用户编程接口概述

6

本章介绍了 IO-Base 设备用户编程接口的基本特性，可为您创建自己的 IO-Base 设备用户程序奠定基础。

有关函数调用和数据访问在“IO 设备数据类型与函数的说明 (页 171)”部分详细介绍。

说明

IO 数据方向始终从 IO 控制器的角度进行描述。 因此，IO 设备写入输入数据（数据到 IO 控制器）及读取输出数据（数据来自 IO 控制器）。
IO 设备的 IO-Base 用户编程接口的整个描述中均使用此惯例。

说明

IO 设备功能只能与 CP 1616/CP 1604 通信处理器配合使用。
“DK ERTEC 400/200”产品的接口与 ERTEC 200 和 400 的接口相同。

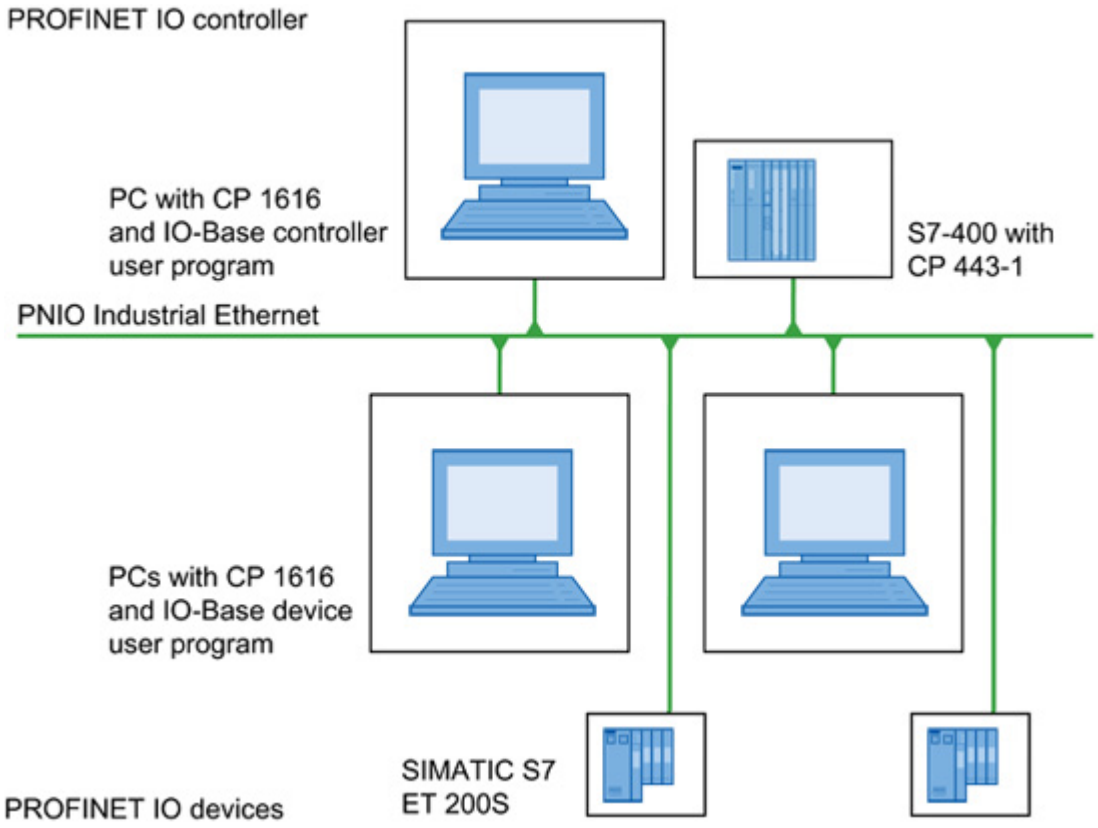
6.1 IO-Base 设备用户编程接口的典型应用

说明

下图说明了一个典型应用： 一台 PC 通过工业以太网与多台 PROFINET IO 控制器通信。

IO-Base 设备用户程序和 SIMATIC NET PC 软件产品的 IO-Base 设备函数均在 PC 上运行。 数据通信通过 PROFINET IO 通信处理器与 SIMATIC S7 PROFINET CP (PLC) 进行处理，或由 PC 与 IO-Base 控制器用户程序通过工业以太网进行处理。

此组态将在提供的示例程序中再次提及。

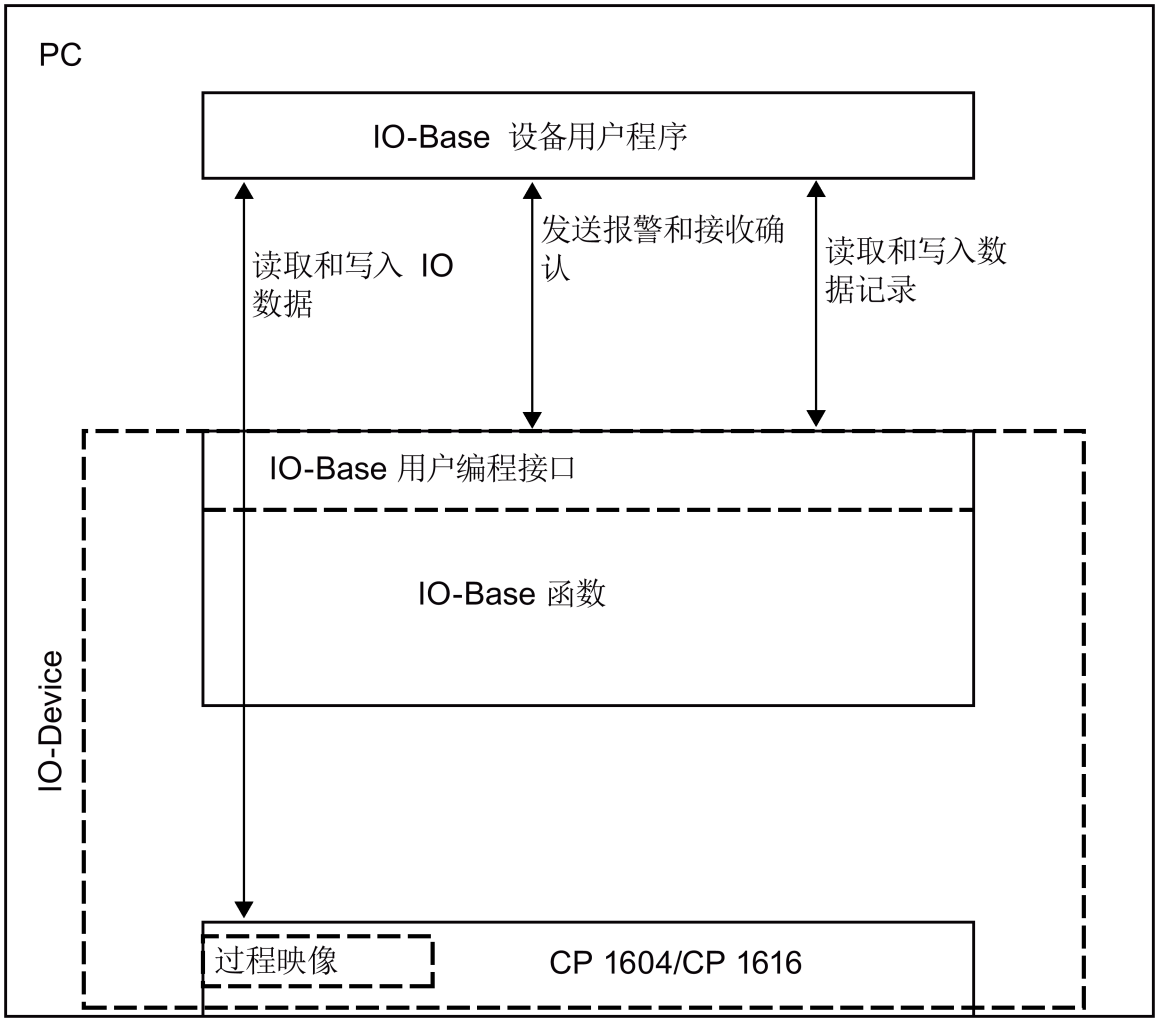


6.2 具有 PROFINET IO 的 PC 上的软件架构

说明

PROFINET IO 借助 IO-Base 用户编程接口提供 C 用户程序与 PROFINET IO 设备通信所需的所有函数。

其中包括用于读/写 IO 数据、发送报警与接收报警确认以及读/写数据记录的 IO-Base 函数。PROFINET IO 软件通过 PROFINET IO 通信处理器连接到过程。通过 STEP 7 或 NCM PC 组态与 IO 控制器的通信。



头文件、库和示例程序

IO 设备支持需要下列文件：

文件类型	文件名	目的
头文件	pniobase.h	包含 IO 控制器和 IO 设备的全局结构和定义。
头文件	pniousrd.h	包含数据类型、常量和函数声明。
头文件	pnioerrx.h	包含错误代码。
库	libpniousr	链接到您的 IO-Base 设备用户程序。

6.3 IO-Base 设备用户程序的典型顺序

概述

IO-Base 设备用户程序的典型顺序可划分为 3 个阶段。

- 初始化阶段
- 生产运行阶段
- 完成阶段

下面将对这几个阶段进行详细介绍。

6.3.1 初始化阶段

说明

初始化阶段划分为多个步骤。 对由 IO-Base 设备用户程序激活的函数调用和由 IO-Base 接口激活的回调函数调用必须加以区分。

说明

除非另行指定，否则唯一可以使用的回调函数为下表“目的”列中指定的 IO-Base 函数。

步骤	操作	目的
1	PNIO_device_open( )	• 向 CP 注册。 • 注册用于过程数据、数据记录和报警的回调函数。
2	PNIO_api_add( )	注册 API（Application Process Identifier，应用程序进程标识符）。

步骤	操作	目的
3	PNIO_mod_plug( )	• 如果固件版本 >= V2.6 将前端模块 (DAP) 插入插槽 1 将相应子模块插入插槽 1 的子插槽 1 • 如果固件版本 < V2.6 将前端模块 (DAP) 插入插槽 0 将相应子模块插入插槽 0 的子插槽 1 如果 PNIO_CBF_PULL_PLUG_CONF( ) 已注册，则等待其调用。
4	PNIO_sub_plug_ext_IM( )	插入相关子模块 0（对应于前端站）。 如果 PNIO_CBF_PULL_PLUG_CONF( ) 已注册，则等待其调用。
5	PNIO_mod_plug( )	插入其它模块。 如果 PNIO_CBF_PULL_PLUG_CONF( ) 已注册，则等待其调用。
6	PNIO_sub_plug_ext_IM( )	插入相关子模块。 如果 PNIO_CBF_PULL_PLUG_CONF( ) 已注册，则等待其调用。
7	PNIO_CP_register_cbf( )	如需支持 IRT 组态，因此支持等时实时模式，则必须注册回调函数 PNIO_CP_CBE_OPFAULT_IND 和 PNIO_CP_CBE_STARTOP_IND。 不然，最终才允许使用 PNIO_set_appl_state_ready( ) 函数（请参见步骤 17）建立连接。 之后便会通知这些回调函数。
8	PNIO_device_start( )	激活 IO 设备，IO 控制器此后便能识别该 IO 设备。

步骤	操作	目的
9	等待 PNIO_CBF_AR_CHECK_IND( ) 回调函数调用。	IO 控制器建立到 IO-Base 设备用户程序的连接以及传送其预期用于 IO-Base 设备用户程序的组态后，IO-Base 接口立即调用此回调函数。 通过调用此回调函数，将应用关系全局参数传送到 IO-Base 设备用户程序中检查。 如果应用关系全局参数存在错误，IO-Base 设备用户程序可中止应用关系的建立；另请参见 PNIO_device_ar_abort( )。 最终允许使用 PNIO_set_appl_state_ready( ) 函数（请参见步骤 17）建立连接。
10	响应任何 PNIO_CBF_CHECK_IND( ) 回调函数调用。	在 IO 设备的子模块的组态与 IO 控制器的组态不匹配时，将为每个这样的子模块调用该回调函数。 用户程序因此能更改子模块的组态或将相关子模块标记为兼容或不良。 如果组态匹配，则不调用该回调函数。
11	等待 PNIO_CBF_AR_INFO_IND( ) 回调函数调用。	在与 IO 控制器建立了应用关系建立后，IO-Base 立即调用该回调函数。 通过调用该回调函数，将在该应用关系中运行的模块及子模块通知给 IO-Base 设备用户程序。
12	响应 NIO_CBF_REC_WRITE( ) 回调函数调用。	如果 IO 控制器为子模块传送参数分配数据记录，则由 IO-Base 接口调用该回调函数。 通过调用该回调，将每个子模块的全部参数分配数据传送到 IO-Base 设备用户程序。
13	等待 PNIO_CBF_PRM_END_IND( ) 回调函数调用。	IO 控制器报告参数分配阶段结束后，IO-Base 接口立即调用该回调函数。
14	PNIO_initiate_data_write( )	通过该调用，IO-Base 设备用户程序启动对 PNIO_CBF_DATA_WRITE( ) 回调函数的调用，以便 IO-Base 设备用户程序能够初始化有效子模块的输入数据并将本地状态设置为“GOOD”。 必须将所有无效子模块的本地状态设置为“BAD”。

步骤	操作	目的
		注意 PROFINET IO 标准要求将全部有效子模块的所有输出数据设置为有效值，以及在发送应用程序准备就绪信息（步骤 17）前将本地状态设置为 GOOD。
15	PNIO_initiate_data_read( )	通过该调用，IO-Base 设备用户程序启动对 PNIO_CBF_DATA_READ( ) 回调函数的调用，以便它能够将有效子模块的所有输出数据的本地状态设置为“GOOD”。 必须将所有无效子模块的本地状态设置为“BAD”。
		注意 PROFINET IO 标准要求在发送应用程序准备就绪信息（步骤 17）前，将所有功能子模块的本地状态设置为 GOOD。
16	等待 PNIO_CP_CBE_STARTOP_IND 事件到达	若要支持 IRT 组态，程序必须等待 PNIO_CP_CBE_STARTOP_IND 回调事件。 该事件通知 IO-Base 设备用户程序现在可处理 IRT 数据（开始等时实时数据处理）。 **等时实时模式** 通过调用函数 PNIO_initiate_data_read_ext( ) 及 PNIO_initiate_data_write_ext( )，IO-Base 设备用户程序可命令 IO-Base 编程接口调用 PNIO_CBF_DATA_READ( ) 或 PNIO_CBF_DATA_WRITE( ) 回调函数，以使用户程序能够初始化具有“GOOD”状态的 IRT 子模块。 **非等时模式** 在等时实时模式下，可在该回调事件之外执行以上列出的函数。 必须仅在如下所述的步骤 18 之前执行这些函数。 **步骤 17 完成** 必须在启动该回调事件前调用 PNIO_CP_set_opdone( )。

步骤	操作	目的
17	PNIO_set_appl_ state_ready( )	通过该调用，IO-Base 设备用户程序将无效子模块的列表报告给 IO 控制器，并发告知对方可以开始数据交换。
18	等待 PNIO_CBF_AR_ INDATA_IND( ) 回调函数调用。	IO 控制器第一次传送 IO 数据后，IO-Base 接口立即调用该回调函数。 通知周期性数据交换开始。

集成周期性运行中的 IO 设备（最低要求）

要集成周期性运行中的 IO 设备，必须至少采取以下步骤：

步骤	操作	目的
1	PNIO_device_open( )	• 向 CP 注册。 • 注册用于过程数据、数据记录和报警的回调函数。
2	PNIO_api_add( )	注册 API（应用程序进程标识符）- 必须为每个组态的配置文件成功执行具有相应 API 的 PNIO_api_add( )。 请注意，若是前端站，必须是 API=0 的 PNIO_api_add( )。

6.3.2 生产运行阶段

概述

在生产运行阶段与 IO 控制器交换数据。 具体数据如下：

- 读/写 IO 数据
- 发送报警及接收报警确认
- 读/写数据记录

数据交换将在下文详细介绍。

读取 RT IO 数据

读 IO 数据（从 IO 控制器角度来论的输出数据）分两步进行：

步骤	操作	目的
1	PNIO_initiate_data_read( ) 或 PNIO_initiate_data_read_ext( )	将读取请求发送到 IO-Base 接口。 这促使 IO-Base 接口执行步骤 2。仅在处理完步骤 2 后返回该调用。
2	PNIO_CBF_DATA_READ( )	IO-Base 接口为具有输出数据的每个子模块调用该回调函数，另外还会将从 IO 控制器接收输出数据的数据缓冲区的指针发送给该回调函数。

写入 RT IO 数据

写 RT IO 数据（从 IO 控制器角度来论的输入数据）分两步进行：

步骤	操作	目的
1	PNIO_initiate_data_write( ) 或 PNIO_initiate_data_write_ext( )	将写入请求发送到 IO-Base 接口。 这促使 IO-Base 接口执行步骤 2。仅在处理完步骤 2 后返回该调用。
2	PNIO_CBF_DATA_WRITE( )	IO-Base 接口为具有输入数据的每个子模块调用该回调函数，并且还会将数据缓冲区指针发送给该回调函数，要发送到 IO 控制器的输入数据将被复制到相应数据缓冲区。

读取及写入 IRT IO 数据

读取及写入 IRT IO 数据涉及六个步骤：

步骤	操作	目的
1	等待 PNIO_CP_CBE_STARTOP_IND 事件。	通过该事件，IO-Base 接口将开始等时实时数据处理通知给 IO 设备用户程序。
2	PNIO_initiate_data_read_ext(PNIO_ACCESS_IRT_WITHOUT_LOCK)	将读取请求发送到 IO-Base 接口。 这促使 IO-Base 接口执行步骤 3。仅在处理完步骤 3 后返回该调用。
3	PNIO_CBF_DATA_READ( )	IO-Base 接口为每个具有输出数据的 IRT 子模块调用该回调函数。 数据缓冲区指针将传送给该回调函数，且数据缓冲区包含从 IO 控制器接收的输出数据。
4	PNIO_initiate_data_write_ext(PNIO_ACCESS_IRT_WITHOUT_LOCK)	将写入请求发送到 IO-Base 接口。 这促使 IO-Base 接口执行步骤 5。仅在处理完步骤 5 后返回该调用。
5	PNIO_CBF_DATA_WRITE( )	IO-Base 接口为每个具有输入数据的 IRT 子模块调用该回调函数。 将数据缓冲区指针传送给该回调函数。 必须通过 IO-Base 用户程序将要发送到 IO 控制器的输入数据复制到该缓冲区。
6	PNIO_CP_set_opdone( )	IO 设备用户程序使用该调用通知 IO-Base 接口等时实时数据处理结束。

在非等时模式下，没有总线周期耦合。即，步骤 6 紧接步骤 1 执行；步骤 2 到 5 与 PNIO_CP_CBE_STARTOP_IND 事件异步，即，它们可在任意时间点执行。

读数据记录作业的响应

读数据记录作业的响应由一个步骤组成：

步骤	操作	目的
1	等到调用 PNIO_CBF_REC_READ 回调函数。	IO-Base 接口接收到 IO 控制器的读数据记录作业后，立即调用由 IO-Base 设备用户程序注册的 PNIO_CBF_REC_READ 回调函数。

写数据记录作业的响应

写数据记录作业的响应由一个步骤组成：

步骤	操作	目的
1	等到调用 PNIO_CBF_REC_WRITE 回调函数。	IO-Base 接口接收到 IO 控制器的写数据记录作业后，立即调用由 IO-Base 设备用户程序注册的 PNIO_CBF_REC_WRITE 回调。

发送报警及接收报警确认

按以下所述的两个步骤处理报警：

步骤	操作	目的
1	调用以下函数之一： PNIO_process_alarm_send( ) PNIO_diag_alarm_send( ) PNIO_ret_of_sub_alarm_send ( )	发送报警
2	等到调用 PNIO_CBF_REQ_DONE 回调函数。	接收确认

建立和终止 IO 设备连接时的回调事件

建立连接时，IO-Base 接口使用相应回调函数提供信息。

下表列出了“提供的信息”以及相应的回调函数名称。

回调函数名称	表示的信息
PNIO_CBF_AR_CHECK_IND( )	应用关系信息
PNIO_CBF_CHECK_IND( )	检查指示 - 实际模块组态与项目工程中的组态不匹配。
PNIO_CBF_AR_INFO_IND( )	期望的 IO 设备组态
PNIO_CBF_PRM_END_IND( )	IO 控制器分配参数结束
PNIO_CP_CBE_STARTOP_ IND( )	等时实时数据处理开始

回调函数名称	表示的信息
PNIO_CP_ OPFAULT_IND( )	违背等时实时模式；用户程序调用 PNIO_CP_set_opdone( ) 过晚。
PNIO_CBF_AR_INDATA_IND( )	应用关系 InData - 完成与 IO 控制器的第一次数据交换。
PNIO_CBF_AR_ABORT_IND( )	中止事件 - 完成交换数据前连接中止。
PNIO_CBF_AR_OFFLINE_IND( )	离线事件 - 完成交换数据后连接中止。
PNIO_CBF_APDU_STATUS_IND( )	IO 控制器的状态

说明

这些服务包括被动功能。 PROFINET 库调用所有这些回调函数以响应 PROFINET IO 控制器的动作。

6.3.3 完成阶段

说明

IO 设备的完成阶段包含四个步骤：

步骤	操作	目的
1	PNIO_device_stop( )	IO 设备取消激活，因此不能再为 IO 控制器所用。 PNIO_device_stop() 确认总是伴随着 PNIO_CBF_DEVICE_STOPPED( ) 回调。 如果 IO 控制器与 IO 设备之间存在数据交换（接口调用了 PNIO_CBF_AR_INDATA_IND( ) 回调函数），IO-Base 接口将调用 PNIO_CBF_AR_OFFLINE_IND( )。 如果 IO 控制器与 IO 设备之间还不存在数据交换（接口未调用 PNIO_CBF_AR_INDATA_IND( ) 回调函数），IO-Base 接口将调用 PNIO_CBF_AR_ABORT_IND( ) 和 PNIO_CBF_DEVICE_STOPPED( ) 回调函数。
2	PNIO_mod_pull( )	从 IO-Base 接口移除所有模块及其子模块。 如果 PNIO_CBF_PULL_PLUG_CONF( ) 已注册，则等待其调用。
3	PNIO_api_remove( )	删除应用程序进程标识符。
4	PNIO_device_close( )	从通信处理器上取消注册。

6.4 IO-Base 设备函数的基本数据交换

说明

IO-Base 设备函数使用以下基本方法交换数据：

- 非等时访问周期性 IO 数据 (RT)
  - 写入 IO 数据
  - 读取 IO 数据

  IO 数据交换也包含状态信息。

  此特殊功能将在下一节介绍。
- 等时实时和非等时实时访问周期性 IO 数据 (IRT)

  访问函数与上文介绍的函数相同。 必须仅访问 IRT 数据。

  在 PNIO_CP_CBE_STARTOP_IND 事件与 PNIO_CP_set_opdone( ) 函数调用之间访问 IRT 数据被称为等时实时访问。

  在 PNIO_CP_CBE_STARTOP_IND 事件与 PNIO_CP_set_opdone( ) 函数调用之外访问 IRT 数据则称为非等时实时访问。

  有关更多信息，请参见“等时实时和非等时实时访问周期性 IO 数据 (IRT) (页 157)”部分。
- 访问非周期性数据
  - 写入和读取数据记录
  - 发送报警及接收报警确认

  有关更多信息，请参见“回调机制 (页 167)”部分。

6.5 非等时访问周期性 IO 数据 (RT)

理论上的工作原理

在 IO-Base 设备用户程序写入和读取 IO 数据时，都是对 PC 上的本地过程映像执行读写操作。 没有任何数据是通过网络发送的。

底层 IO-Base 设备函数或硬件周期性地独立处理本地过程映像与 IO 控制器之间的数据交换。

此数据交换的详细内容在组态中指定。

说明

IO-Base 设备用户程序不必在每个总线循环都写入或读取 IO 数据。

说明

IO-Base 设备用户程序不必比组态的周期更为频繁地访问过程映像。

IO 数据和数据状态

IO 数据的质量通过数据状态的 GOOD 或 BAD 这两个值来描述。

无论是写入还是读取时，都将交换以下两个数据状态：

- 本地状态（IO-Base 设备用户程序的状态）
- 远程状态（通信伙伴的状态）

6.5.1 状态的周期性读取

PNIO_CBF_DATA_READ( ) 函数的顺序

IO-Base 设备用户程序通过调用 PNIO_initiate_data_read( ) 或 PNIO_initiate_data_read_ext( ) 函数启动读取操作。 之后，针对由 IO 控制器运行的每个子模块，IO-Base 接口将调用 PNIO_CBF_DATA_READ( ) 回调函数。 通过调用该回调函数，通信伙伴可从本地过程映像读出输出数据及相应的远程数据状态。

此外，还会将该输出数据的本地状态写入通信伙伴的本地过程映像。

因此在周期性读取期间共涉及两个状态。

通信方向	值
来自 IO 控制器	• IO 控制器的输出数据 • 远程状态
到 IO 控制器	本地状态

通信伙伴的远程状态

通信伙伴利用远程状态报告输出数据的质量（GOOD 或 BAD）。

如果通信伙伴报告 BAD，IO-Base 控制器用户程序可使用替换值继续处理。

本地状态

通常情况下，IO-Base 设备用户程序都会将本地状态设置为 GOOD。

但是，如果 IO-Base
设备用户程序无法进一步处理所提供的输出数据，则会在执行读作业期间将本地状态设置为 BAD。

通信伙伴在接到此状态后，便可识别出其发送的输出数据是否已被成功处理。

6.5.2 状态的周期性写入

写入过程映像的顺序

IO-Base 设备用户程序通过调用 PNIO_initiate_data_write( ) 或
PNIO_initiate_data_write_ext( ) 函数启动写入操作。 之后，针对由 IO
控制器运行的每个子模块，IO-Base 接口将调用 PNIO_CBF_DATA_WRITE( )
回调函数。 通过调用 PNIO_CBF_DATA_WRITE( )
回调函数，将输入数据及其相应的本地数据状态写入通信伙伴的本地过程映像。

通信伙伴还将从本地过程映像中读取该输入数据的远程状态。

因此在周期性写入期间共涉及两个状态。

通信方向	值
来自 IO 控制器	• IO 控制器的输入数据 • 本地状态
到 IO 控制器	远程状态

本地状态

通常情况下，IO-Base 设备用户程序都会将本地状态设置为 GOOD。

如果输入数据不正确或无效，IO-Base 设备用户程序应将本地状态设置为 BAD。

在这种情况下，通信伙伴便可输出所组态的替换值。

通信伙伴的远程状态

通信伙伴使用远程状态传递其是否能够成功处理过程输入数据或者是否存在问题的信息。

此状态与之前同一子模块传送的输入数据相关，而不是与刚刚启动的写入作业相关。

6.6 等时实时和非等时实时访问周期性 IO 数据 (IRT)

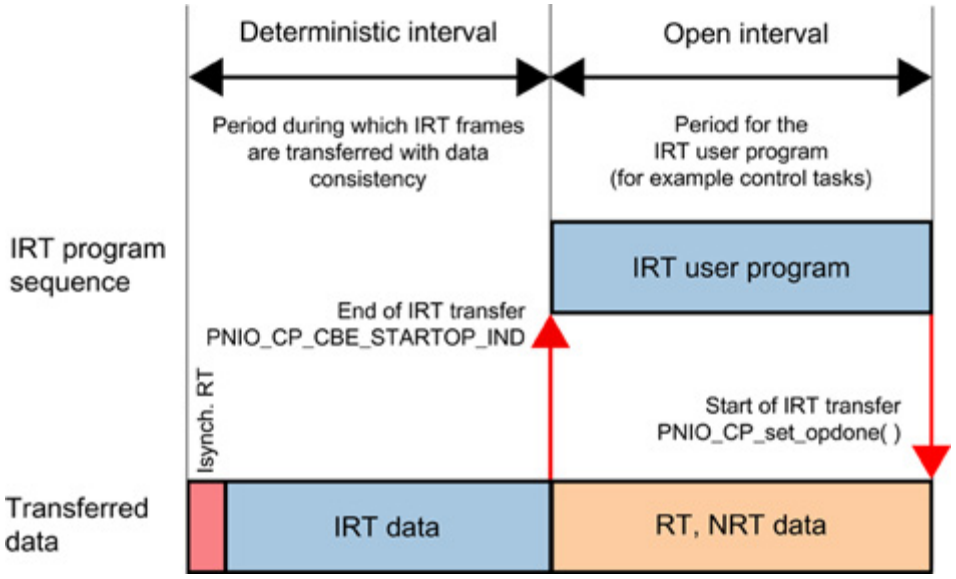
说明

在等时实时模式下，用户程序在两个周期性重复的 IRT 时间点之间处理和写入 IRT 数据。

- IRT 传送结束
- IRT 传送开始

在到达 IRT 传送结束时，DMA 将 IRT 数据从过程映像传送到主机存储器中，并且 PNIO_CP_CBE_STARTOP_IND 回调事件将此情况通知 IRT 用户程序。 此后，所有 IRT 数据便处于主机存储器中，可开始数据处理过程。 在 PNIO_CP_register_cbf( ) 函数调用后，将报告首批 PNIO_CP_CBE_STARTOP_IND 回调事件的情况。

在 IRT 用户程序完成数据处理后，它会调用 PNIO_CP_set_opdone( ) 函数报告此情况。 如此将使用 DMA 将 IRT 输入数据（从 IO 控制器的角度来看）复制到过程映像存储器中，并通知通信处理器数据是否有效，以便可在下一个总线循环传送这些数据。



违背等时实时模式

如果 IRT 数据处理完毕的信号传递得过晚，则 DMA 传送无法及时完成，并且 IRT 输入数据（从 IO 控制器的角度来看）无法标记为有效。

从而在下一个总线循环中，通信处理器会将 IRT 数据作为无效数据传送。
用户程序将通过 PNIO_CP_CBE_OPFAULT_IND 回调事件获知此情况。

等时实时 IO 数据（IRT 数据）的一致性

为了给用户程序更多的数据处理时间，通过 DMA
在主机存储器和过程映像存储器之间传送 IRT 数据。 IRT 数据因此仅在 IRT 传送结束和 IRT 传送开始之间的时段内一致。

这意味着用户程序只能在 PNIO_CP_CBE_STARTOP_IND 回调事件和 PNIO_CP_set_opdone( ) 之间访问一致的 IRT 数据。

在 PNIO_CP_CBE_OPFAULT_IND 回调事件到达时，数据将不再一致。
此情况将持续到下一个 PNIO_CP_CBE_STARTOP_IND 回调事件为止。

6.6.1 访问 IRT 数据

等时实时访问 IRT 数据

使用以下函数及相应的回调函数访问 IRT 数据：

- 读
  - PNIO_initiate_data_read_ ext(PNIO_ACCESS_IRT_ WITHOUT_LOCK)
  - PNIO_CBF_DATA_READ( )
- 写
  - PNIO_initiate_data_write_ ext(PNIO_ACCESS_IRT_ WITHOUT_LOCK)
  - PNIO_CBF_DATA_WRITE( )

如果在 PNIO_CP_CBE_STARTOP_IND 回调事件和 PNIO_CP_set_opdone( ) 之间访问 IRT 数据，则称为等时实时访问 IRT 数据。

非等时实时访问 IRT 数据

如果在 PNIO_CP_CBE_STARTOP_IND 回调事件和 PNIO_CP_set_opdone( ) 以外的时间访问 IRT 数据，则称为异步访问 IRT 数据。

这些访问不被允许且不返回 PNIO_WARN_IRT_INCONSISTENT 错误代码。

确定哪些子模块包含 IRT 数据

IO-Base 设备用户程序使用 PNIO_CBF_AR_INFO_IND( ) 回调函数的传送参数“pAR”识别包含 IRT 数据并因此必须在等时实时模式中进行处理的子模块。

或者，IO-Base 设备用户程序可在第一次调用 PNIO_CP_CBE_STARTOP_IND 回调函数时调用 PNIO_initiate_data_read_ext(PNIO_ACCESS_IRT_WITHOUT_LOCK) 及 PNIO_initiate_data_write_ext(PNIO_ACCESS_IRT_WITHOUT_LOCK) 函数，并在执行 PNIO_CBF_DATA_READ( ) 和 PNIO_CBF_DATA_WRITE( ) 回调函数期间建立 IO 设备列表。

6.7 访问非周期性数据

理论上的工作原理

非周期性数据交换包括对读数据记录请求及写数据记录请求进行响应，以及发送报警和确认接收。

6.7.1 处理数据记录作业

说明

IO 设备用户程序必须通过 PNIO_device_open( ) 注册以下两个回调函数：

- PNIO_CBF_REC_READ( )
- PNIO_CBF_REC_WRITE( )

在收到 IO 控制器的读数据记录或写数据记录请求后，IO-Base 接口总是调用这些回调函数。 要使 IO 设备用户程序能够处理读数据记录或写数据记录作业，这些回调函数必须包含数据记录号及子模块号作为参数。 请参见 IEC 61158 了解哪些记录已标准化以及哪些可根据需要自由使用。

6.7.2 发送报警及接收报警确认

说明

IO-Base 接口为 IO 设备用户程序提供了用于将报警发送到 IO 控制器或向其报告意外事件的函数。

对于 IO 设备发送的每个报警，通过 IO-Base 接口调用 PNIO_CBF_REQ_DONE( ) 回调函数，IO-Base 设备用户程序都会收到一个报警确认。

通过用户句柄分配报警确认。 IO-Base 设备用户程序在发送报警时分配该用户句柄。

PNIO_CBF_REQ_DONE( ) 回调函数包括用户句柄，并将其作为传送参数。

说明

仅在调用 calling PNIO_set_appl_ state_ready( ) 后，IO-Base
设备用户程序才能发送报警。

PROFINET IO 的会话密钥

会话密钥是一个在每次建立应用关系时都会递增的值。
每个连接的会话密钥必须单独存放（“ArNumber”）且在后续的所有函数调用中与“ArNumber”参数一起使用。

通过调用 PNIO_CBF_AR_CHECK_IND( ) 回调函数，在连接建立期间，在 `pAr -> SessionKey` 参数中将会话密钥传送给应用关系。

每个报警都必须包括会话密钥。 IO
控制器然后才能丢弃过期报警以及读数据记录和写数据记录作业的确认。

示例

在 PROFINET IO 中，报警与 IO 控制器及 IO
设备之间的应用关系重建之间实际上存在竞争。 以下示例很好地说明了这一点：

假设以下前提成立：

- 在 IO 控制器与 IO 设备之间存在应用关系。
- 子模块因接触不良导致暂时出现故障，因而 IO 设备要将诊断报警发送到 IO 控制器。
- 但是，在 IO 设备通过网络发送该报警之前，IO
  控制器终止了现有的应用关系（例如，由于重启所致），并想重建应用关系。
- 但是，重建 IO 控制器与 IO 设备的应用关系后，在子模块上已不存接触不良问题。

如果之前存在的应用关系的报警此时到达 IO 控制器，则 IO
控制器会错误地认为子模块上仍然存在接触不良问题。

PROFINET IO 通过上述会话密钥解决这类问题。

6.8 管理诊断数据

说明

PROFINET IO 中有两种不同的诊断过程。

- 通道诊断数据
- 制造商特定诊断数据

说明

在 IEC 61158-6 PROFINET IO“应用层协议规范”版本 1.0 中，强烈建议使用“通道诊断数据”作为诊断过程。

6.8.1 通道诊断数据

在 PROFINET IO 中，诊断数据编码为“通道诊断数据”。
“通道属性”是“通道诊断数据”的一部分。 这些属性可由 IO-Base 设备用户程序使用 PNIO_build_channel_properties( ) 函数生成。 关于“通道诊断数据”的确切定义，请参见 PROFINET IO 标准。

在 PROFINET IO 中，一个子模块可由包含多个通道。
每个通道可能有许多通道诊断数据。 可通过 IO-Base 设备用户程序使用 PNIO_diag_channel_add( ) 函数将这些数据存储在子模块上。

如果通道诊断数据不再有效，IO-Base 设备用户程序必须通过 PNIO_diag_channel_remove( ) 函数将其从子模块中删除。

以下函数可用于管理通道诊断数据：

函数	目的
PNIO_build_channel_properties( )	生成通道属性
PNIO_diag_channel_add( )	将通道诊断数据存储在子插槽中
PNIO_diag_channel_remove( )	从子插槽中移除通道诊断数据

设置通道诊断数据

设置通道诊断数据可分为四个步骤：

步骤	操作	目的
1	PNIO_build_channel_properties( )	通过“进入状态”参数生成通道属性
2	PNIO_diag_channel_add( )	将通道诊断数据存储在子插槽中。
3	PNIO_diag_alarm_send( )	将通道出错进入状态作为诊断报警通知给 IO 控制器。
4	PNIO_CBF_REQ_DONE( )	评估确认情况。

移除通道诊断数据

移除通道诊断数据可分为四个步骤：

步骤	操作	目的
1	PNIO_build_channel_properties( )	通过“退出状态”参数生成通道属性。
2	PNIO_diag_channel_add( )	从子模块中移除通道诊断数据。
3	PNIO_diag_alarm_send( )	将通道出错退出状态作为诊断报警发送给 IO 控制器。
4	PNIO_CBF_REQ_DONE( )	评估确认情况。

6.8.2 制造商特定诊断数据

制造商特定诊断数据允许 IO-Base 设备用户程序存储子模块的制造商特定诊断数据。 没有为制造商特定诊断数据指定固定结构。

也必须为制造商特定诊断数据指定通道属性。 有关确切细节信息，请参见 PROFINET IO 标准。

以下函数可用于管理制造商特定诊断数据：

函数	目的
PNIO_build_channel_properties( )	生成通道属性
PNIO_diag_generic_add( )	将制造商特定诊断数据存储在子插槽中
PNIO_diag_generic_remove( )	从子插槽中移除制造商特定诊断数据存储

设置制造商特定诊断数据

设置制造商特定诊断数据可分为四个步骤：

步骤	操作	目的
1	PNIO_build_channel_properties( )	通过“进入状态”参数生成通道属性
2	PNIO_diag_generic_add( )	将供应商特定诊断数据存储在子插槽中。
3	PNIO_diag_alarm_send( )	将通道出错进入状态作为诊断报警通知给 IO 控制器。
4	PNIO_CBF_REQ_DONE( )	评估确认情况。

移除制造商特定诊断数据

移除制造商特定诊断数据可分为四个步骤：

步骤	操作	目的
1	PNIO_build_channel_properties( )	通过“退出状态”参数生成通道属性
2	PNIO_diag_generic_remove( )	从子模块中移除通道诊断数据。
3	PNIO_diag_alarm_send( )	将通道出错退出状态作为诊断报警通知给 IO 控制器
4	PNIO_CBF_REQ_DONE( )	评估确认情况。

6.9 在生产运行阶段插拔模块的注意事项

移除模块/子模块时的拔出报警

在 IO-Base 设备用户程序调用以下函数拔出模块或子模块后，IO-Base 接口会立即生成 PROFINET IO 拔出报警：

- PNIO_mod_pull( )
- PNIO_sub_plug_ext_IM

注意
如果拔出一个模块，将自动移除其包含的所有子模块。

注意
PROFINET 标准规定，必须使用 BAD 状态首先写入要拔出的子模块的 IO 数据。 只有在连接建立期间注册了子模块时才可能做到这一点。

说明

在 PNIO_CBF_AR_CHECK_IND( ) 回调函数及 PNIO_CBF_PRM_END_IND( ) 之间不允许插/拔模块。

说明

只有通过 PNIO_device_open( ) 注册了 PNIO_CBF_PULL_PLUG_CONF( ) 回调函数后，才支持生产运行期间插拔模块。

说明

如果未注册 PNIO_CBF_PULL_PLUG_CONF( ) 回调函数，但要更改模块组态，则必须使用 PNIO_device_stop( ) 调用停止 IO 设备，以在插/拔模块或子模块前终止应用关系。

在插入模块和子模块后，通过 PNIO_device_start( ) 再次启动 IO 设备。之后重复从“初始化阶段 (页 144)”部分中的步骤 8 开始的步骤。

插入模块/子模块时的插入报警

在 IO-Base 设备用户程序调用以下函数插入模块或子模块后，IO-Base 接口会立即生成 PROFINET IO 插入报警。

- PNIO_mod_plug( )
- PNIO_sub_plug_ext_IM

插入后重新分配参数

继每个 PNIO_sub_plug_ext_IM 之后，IO 控制器会为相应的子模块分配新参数。
这意味着 IO-Base 接口为 IO 控制器传送的每个参数分配数据记录调用 PNIO_CBF_REC_WRITE( ) 回调函数。

IO-Base 接口通过调用 PNIO_CBF_PRM_END_IND( ) 回调函数指示参数分配完成。

紧接参数分配，IO-Base
设备用户程序检查插入的子模块是否能够使用传送的参数设置运行。

- 如果答案是“能”，则 IO-Base
  设备用户程序将输入数据设置为待发送，并将该子模块输入输出的本地状态设置为 GOOD。 之后 IO-Base 设备用户程序通过一个空的子模块列表调用 PNIO_set_appl_state_ready( ) 函数。

  这样就完成了插入过程。

- 如果答案是“不能”，则 IO-Base
  设备用户程序必须使该子模块输入输出的本地状态设置保持为 BAD（在拔出前设置为 BAD；请参见上文注意事项）。 在这种情况下，IO-Base 设备用户程序必须调用 PNIO_set_appl_state_ready( ) 函数指出子模块列表中的相应子模块。

  插入操作随即因出错而结束。

6.9.1 “子模块恢复”的注意事项

说明

如果插入的模块存在功能问题，则 IO-Base
设备用户程序会将输入输出数据的本地状态设置为 BAD。 对于已分配的 IO 控制器，这意味着输入输出数据对 IO 控制器的用户程序不再有效。

当子模块功能恢复时，IO-Base 设备用户程序必须将输入输出数据的本地状态设置为 GOOD。 之后，IO-Base 设备用户程序必须通过调用 PNIO_ret_of_sub_alarm_send( ) 函数将 BAD 到 GOOD 的变化通知给 IO 控制器。 由于子模块恢复报警，IO 控制器不会将新的参数设置发送给子模块。

相应子模块随后再次恢复正常。

6.10 回调机制

工作原理

在 IO-Base 设备用户程序中编写回调函数。 回调函数可任意命名。

回调事件是由 IO-Base 接口启动的异步事件。 其可中断 IO-Base 控制器用户程序的执行，并可在单独的线程上启动回调函数。
这意味着必须具备同步技术。

IO 设备的回调函数

下表列出了 IO 设备上的各回调事件和事件类型。
该表还说明了要使用什么来注册回调函数以及哪些情况会触发回调事件：

回调事件（异步）	回调事件类型	注册手段	触发条件
读数据	PNIO_CBF_ DATA_READ	PNIO_device_open( )	... 包含如下调用的用户程序： • PNIO_initiate_data_ read( ) • PNIO_initiate_data_read_ ext( )
写数据	PNIO_CBF_ DATA_WRITE	PNIO_device_open( )	... 包含如下调用的用户程序： • PNIO_initiate_data_ write( ) • PNIO_initiate_data_ write_ext( )
读数据记录	PNIO_CBF_REC_READ	PNIO_device_open( )	... IO 控制器
写数据记录	PNIO_CBF_REC_WRITE	PNIO_device_open( )	... IO 控制器

回调事件 （异步）	回调事件类型	注册手段	触发条件
发送报警作业的确认	PNIO_CBF_REQ_DONE	PNIO_device_open( )	... 包含如下调用的用户程序： • PNIO_process_alarm_send( ) • PNIO_diag_alarm_send( ) • PNIO_ret_of_sub_alarm_send( )
比较期望组态	PNIO_CBF_CHECK_IND	PNIO_device_open( )	IO-Base 接口
连接建立	PNIO_CBF_AR_INFO_IND	PNIO_device_open( )	IO 控制器
开始数据交换	PNIO_CBF_AR_INDATA_IND	PNIO_device_open( )	IO 控制器
之前没进行数据交换即连接终止	PNIO_CBF_AR_ABORT_IND	PNIO_device_open( )	IO 控制器、用户程序
之前进行数据交换后连接终止	PNIO_CBF_AR_OFFLINE_IND	PNIO_device_open( )	IO 控制器、用户程序
IO 控制器的状态变化	PNIO_CBF_APDU_STATUS_IND	PNIO_device_open( )	IO 控制器
固件下载， 更新或组态下载	PNIO_CBF_CP_STOP_REQ	PNIO_device_open( )	组态站
激活 LED 的闪烁模式	PNIO_CBF_START_LED_FLASH	PNIO_device_open( )	IO 控制器
取消激活 LED 的闪烁模式	PNIO_CBF_STOP_LED_FLASH	PNIO_device_open( )	IO 控制器
确认插拔	PNIO_CBF_PULL_PLUG_CONF	PNIO_device_open( )	用户程序，通过调用： • PNIO_mod_plug( ) • PNIO_sub_plug_ext_IM • PNIO_mod_pull( ) • PNIO_sub_pull( )

回调事件 （异步）	回调事件类型	注册手段	触发条件
等时实时数据处理开始	PNIO_CP_CBE_STARTOP_ IND	PNIO_CP_register_cbf( )	IRT 传送阶段结束，数据存储在主机存储器中
IRT 周期超出	PNIO_CP_CBE_OPFAULT_ IND	PNIO_CP_register_cbf( )	PNIO_CP_set_opdone( ) 被应用程序调用得过晚。

说明

在编译用户程序时包含多线程标准库。

说明

在回调函数内，只能调用访问 IO 数据的函数。 这些函数包括：

- PNIO_initiate_data_read( )
- PNIO_initiate_data_read_ext( )
- PNIO_initiate_data_write( )
- PNIO_initiate_data_write( )
- PNIO_CP_set_opdone( )

协调回调的顺序

回调函数可随时中断 IO-Base 设备用户程序。 不同事件的回调函数也可互相中断。
由于可能从不同的线程调用回调函数，因此必须将回调函数设计成能够同时多次执行（可重入）。

具体而言，这表示共享变量的写入和读取必须由同步机制提供保护。

避免回调函数中的等待时间，尤其是当进入关键部分时。
否则，将限制对其及其它回调函数的新调用。 应当尽可能应使用单独的数据库。

IO 设备数据类型与函数的说明

7

本章详细介绍了 IO 设备 IO-Base 设备用户编程接口的具体函数及其使用的数据类型。

本章主要希望为您编写 IO-Base 设备用户程序提供参考。

7.1 IO 设备的管理函数

概述

IO 设备可识别以下管理函数：

- PNIO_device_open( )
- PNIO_device_close( )
- PNIO_device_start( )
- PNIO_device_stop( )

这些函数将在以下部分详细介绍。

IRT 模式下可用的其它管理函数在“等时实时模式 (IRT) 接口 (页 232)”部分介绍。

7.1.1 PNIO_device_open( )（将设备注册为 IO 设备）

说明

IO-Base 设备用户程序使用此函数向 IO-Base 接口注册 IO 设备。

将使用函数表注册要使用的回调函数。

说明

当显示返回值“PNIO_ERR_CONFIG_IN_UPDATE”时，表示 PNIO 控制器尚未给 IO 设备分配 IP
地址。在此情况下，应用程序需要调用“PNIO_device_open”，直到不再显示该返回值。

语法

```
PNIO_UINT32  PNIO_device_open(
    PNIO_UINT32          CpIndex,          //in
    PNIO_UINT32          ExtPar,           //in
    PNIO_UINT16          VendorId,         //in
    PNIO_UINT16          DeviceId,         //in
    PNIO_UINT16          InstanceId,       //in
    PNIO_UINT32          MaxAr,            //in
    PNIO_ANNOTATION     *pDevAnnotation,   //in
    PNIO_CFB_FUNCTIONS  *pCbf,             //in
    PNIO_UINT32         *pDevHndl          //out
```

参数

名称	说明
CpIndex	模块索引 - 用于唯一标识通信模块；必须是 1！
ExtPar	始终必须传送 PNIO_CEP_MODE_CTRL！
VendorId	IO 设备的供应商 ID - 由 PROFINET 用户组织分配。这对应于 GSDML 文件的“VendorID”。
DeviceId	IO 设备 ID - 在制造商的 PROFINET IO 产品范围内必须唯一。这对应于 GSDML 文件的“DeviceID”。
InstanceId	“实例 ID”是 IO 设备的预留标识符，其值 = 1。
MaxAR	取决于所需模式， • 如果仅需要支持实时模式，“MaxAR”必须始终为 1。 • 如果需要支持实时及等时实时模式，“MaxAR”必须为 2。 • 如果需要支持实时模式、等时实时模式以及管理器连接，“Max AR”必须为 3。
pDevAnnotation	指向模块版本标识结构的指针；请参见“PNIO_ANNOTATION（版本 ID 结构） (页 242)”部分。
pCbf	指向回调函数函数表的指针 - 如果为回调函数输入了空指针，则不支持相应的服务。 请参见头文件“pniousrd.h”中对 PNIO_CFB_FUNCTIONS 结构的定义，了解哪些回调函数需要注册。
pDevHandle	指向分配给 IO 设备的句柄的指针。 必须在后续所有函数调用中包含该句柄。

“ExtPar”参数的附加信息

“PNIO_device_open( )”管理函数的“ExtPar”参数的标志名称。	说明
PNIO_CEP_SET_MRP_ROLE_OFF	通过“PNIO_CEP_SET_MRP_ROLE_OFF”标志，将 MRP 角色设置为值“MRP 角色关闭”。这意味着可通过 IO 基站接口关闭 MRP 角色。此标志可与其它标志链接。
PNIO_CEP_PE_MODE_ENABLE	如果设置此标志，则可通过 PNIO 控制器的作业关闭和启动主机 PC 以节省能源。

返回值

如果成功，则返回
PNIO_OK。如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_CONFIG_IN_UPDATE
- PNIO_ERR_CREATE_INSTANCE
- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_OS_RES
- PNIO_ERR_PRM_CALLBACK
- PNIO_ERR_PRM_CP_ID
- PNIO_ERR_PRM_DEV_ANNOTATION
- PNIO_ERR_PRM_EXT_PAR
- PNIO_ERR_PRM_HND
- PNIO_ERR_PRM_MAX_AR_VALUE
- PNIO_ERR_PRM_PCBF
- PNIO_ERR_SEQUENCE

7.1.2 PNIO_device_close( )（取消将设备注册为 IO 设备）

说明

IO-Base 设备用户程序使用此函数从之前通过“PNIO_device_open”注册的 IO-Base 接口取消注册 IO 设备。

说明

只有在 IO 设备通过异步回调函数确认所有作业后，才能调用 PNIO_device_close( ) 函数。

如果没有遵守此规则，将返回 PNIO_ERR_SEQUENCE，且需要再次执行 PNIO_device_close( ) 作业。

说明

只有在接收到 PNIO_device_stop( ) 的成功确认后，才可调用 PNIO_device_close( ) 函数。

说明

之前注册的所有回调函数都将取消注册。 在 PNIO_device_close( ) 函数执行完毕后，将不会为取消注册的 IO 设备调用其它回调函数。

说明

在成功取消注册后，句柄将不再有效，并不再可供 IO-Base 设备用户程序使用。

语法

```
PNIO_UINT32  PNIO_device_close(
    PNIO_UINT32        DevHndl            //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND

7.1.3 PNIO_device_start( )（启动 IO 设备）

说明

只有在调用该函数后，IO 控制器才能对 IO 设备进行寻址。

语法

```
PNIO_UINT32  PNIO_device_start(
    PNIO_UINT32          DevHndl             //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND

7.1.4 PNIO_device_stop( )（停止 IO 设备）

说明

调用该函数后，不能再通过网络对 IO 设备进行寻址。 IO 设备取消激活，因此不能再为 IO 控制器所用。

如果 IO 控制器与 IO 设备之间存在数据交换（接口调用了 PNIO_CBF_AR_INDATA_IND( ) 回调函数），IO-Base 接口将调用 PNIO_CBF_AR_OFFLINE_IND( )。

如果 IO 控制器与 IO 设备之间没有数据交换（接口没有调用 PNIO_CBF_AR_INDATA_IND( ) 回调函数），IO-Base 接口将调用 PNIO_CBF_AR_ABORT_IND( ) 回调函数。

PNIO_device_stop() 确认通过 PNIO_CBF_DEVICE_STOPPED( ) 回调来实现。

语法

```
PNIO_UINT32  PNIO_device_stop(
    PNIO_UINT32          DevHndl             //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND

7.1.5 回调函数 PNIO_CBF_DEVICE_STOPPED( )（报告设备停止）

说明

PNIO_CBF_DEVICE_STOPPED( ) 回调函数会在 PNIO_device_stop( ) 函数被调用后由 IO-Base 接口调用。 它通知 IO-Base 设备用户程序 IO 设备不能再通过网络进行寻址。

语法

```
typedef void (* PNIO_CBF_DEVICE_STOPPED)(
    PNIO_UINT32     DevHndl,                //in
    PNIO_UINT32     ErrorCode               //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
ErrorCode	错误消息

返回值

无返回值。

7.2 IO 设备组态的接口

概述

I/O 设备利用 PNIO_CBF_PULL_PLUG_CONF( )
函数发送应用程序进程标识符和状态/状态变化。

7.2.1 回调函数 PNIO_CBF_PULL_PLUG_CONF( )（确认插/拔）

说明

IO-Base 接口调用 PNIO_CBF_PULL_PLUG_CONF( )
回调函数以确认模块/子模块的插/拔操作。

语法

```
typedef PNIO_IOXS (*PNIO_CBF_PULL_PLUG_CONF)(
    PNIO_UINT32        DevHndl,               //in
    PNIO_UINT32        API,                   //in
    PNIO_UINT32        Action,                //in
    PNIO_DEV_ADDR     *pAddr,                 //in
    PNIO_UINT32        ErrorCode              //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
API	应用程序进程标识符
操作	指定是否有对模块/子模块插拔的确认。 可能的值如下： • PNIO_MOD_PULL • PNIO_SUB_PULL • PNIO_MOD_PLUG • PNIO_SUB_PLUG
pAddr	指向模块/子模块地址的指针
ErrorCode	有关插/拔的错误消息

返回值

无返回值。

7.3 插拔函数

概述

IO-Base 设备用户程序调用插拔函数以将当前组态通知给 IO-Base 接口。可使用下列函数：

函数	说明
PNIO_sub_plug_ext_IM( )	插入子模块；甚至是目前还未组态的子模块。
PNIO_sub_pull	拔出子模块。

因兼容性原因而存在

通过 PNIO_sub_plug_ext_IM( ) 函数进一步扩展了 IO-Base user 用户编程接口。该函数用于替换以下两个之前因兼容性原因而仍然存在的函数。

之前的函数	当前函数
PNIO_sub_plug( )	PNIO_sub_plug_ext_IM( )
PNIO_sub_plug_ext( )	PNIO_sub_plug_ext_IM( )

说明

不再使用之前的函数。

7.3.1 PNIO_sub_plug_ext_IM( )（扩展的插入子模块）

说明

该函数必须由 IO-Base 设备用户程序在初始化阶段调用，请参见“初始化阶段 (页 144)”部分。 它向 IO-Base 接口报告当前组态。

如果当前组态发生变化（例如发生故障或移除的子模块再次起作用），也可以在操作期间调用该函数。 在这种情况下，IO-Base 接口会自动向 IO 控制器发送插入报警。

该函数可以通知 IO 控制器新插入了与组态不相符的子模块。 在这种情况下，将向 IO 控制器发送“插入错误的子模块报警”。

PNIO_sub_plug_ext_IM( ) 是异步函数。 返回值提供有关接口是否接受作业的信息。 PNIO_CBF_PULL_PLUG_CONF( ) 回调函数提供插入确认。 与之前的函数 PNIO_sub_plug() 及 PNIO_sub_plug_ext() 相反，组态子模块是为了读取和写入 I&M 数据记录。

说明

如果通过“插入报警”或“插入错误的子模块报警”将 IO 设备的变化通知给 IO 控制器，则 IO 控制器不会触发任何回调函数。

说明

在循环数据交换开始之前不会向 IO 控制器发送“插入报警”，该数据交换通过 PNIO_CBF_AR_INDATA_IND( ) 回调函数向 IO-Base 设备用户程序发信号。

说明

只要模块中插入子模块，就会向 IO 控制器发送“插入报警”。
如果插入模块，则不会触发报警。

说明

不管是使用哪个函数（PNIO_sub_plug_ext_IM( )、PNIO_sub_plug( ) 或 PNIO_sub_plug_ext( )）插入了子模块，工程师站对 I&M 数据记录的写入/读取查询总会报告给 IO-Base 设备用户程序。

语法

```
PNIO_UINT32  PNIO_CODE_ATTR(
    PNIO_UINT32             DevHndl,              //in
    PNIO_UINT32             API,                  //in
    PNIO_DEV_ADDR          *pAddr,                //in
    PNIO_UINT32             SubIdent              //in
    PNIO_UINT32             AlarmType             //in
    PNIO_PLUG_IM0_BITS      IM0_bits              //in
);
```

参数

名称	说明	
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄	
API	模块的应用程序进程标识符 - 必须与通过 PNIO_api_add( ) 函数调用来传送的 API 参数之一相对应。	
pAddr	指向插入了子模块的子插槽的地址的指针。	
	注意 在 IO 设备地址中只能为子插槽编号指定 1 到“MaxSubslots”的值。 通过调用 PNIO_api_add( ) 来指定“MaxSubslots”的值。	
SubIdent	相关 GSDML 文件中的子模块 ID。	
AlarmType	“AlarmType”参数决定在插入新子模块后向 IO 控制器发送“插入报警”还是“插入错误的子模块报警”。 如果新子模块的子模块 ID 错误，而且与项目工程中的子模块不兼容，则必须按照标准发出“插入错误的子模块报警”。	
	如果要发送... ,	**将“AlarmType”参数设置为...**
	插入报警	PNIO_ALARM_TYPE_PLUG
	插入错误的子模块报警	PNIO_ALARM_TYPE_PLUG_WRONG
IM0_bits	具有参数“IM0_bits”设置的子模块列于“I&M0FilterData”数据记录中。 传送值在以下的段落“IM0_bits”中介绍。	

IM0_bits

“IM0_bits”参数的传送值具有以下含义：

名称	说明
PNIO_PLUG_IM0_BITS_NOTHING	该子模块不包含任何 I&M 数据。
PNIO_PLUG_IM0_BITS_ SUBMODULE_REP_MOD_DEV	该子模块包含替代模块及 IO 设备的 I&M 数据。 只有前端模块（即插槽 0 及子插槽 1 中的子模块）包含 I&M 数据时才选择该值。

名称	说明
PNIO_PLUG_IM0_BITS_ SUBMODULE_REP_MODULE	该子模块包含替代模块的 I&M 数据。 在每个插槽只插入了一个包含 I&M 数据的子模块时，选择该值。 在一个插槽插入了多个具有 I&M 数据的子模块时，为第一个子模块选择该值。
PNIO_PLUG_IM0_BITS_SUBMOD ULE	该子模块包含 I&M 数据。 在一个插槽插入了多个具有 I&M 数据的子模块时，为第二个及所有其它子模块选择该值。

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_PRM_ADD
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND

7.3.2 PNIO_sub_pull( )（拔出子模块）

说明

当插入的子模块发生故障或被移除时，IO-Base 设备用户程序调用该函数。
在这种情况下，IO-Base 接口会自动向 IO 控制器发送拔出报警。

PNIO_sub_pull( ) 是异步函数。 返回值提供有关接口是否接受作业的信息。
PNIO_CBF_PULL_PLUG_CONF( ) 回调函数提供移除确认。

说明

如果通过“插入报警”或“插入错误的子模块报警”将 IO 设备的变化通知给 IO 控制器，则具有 CP 16xx 的 IO 控制器不会触发任何回调函数。

说明

在循环数据交换开始之前不会向 IO 控制器发送“插入报警”，该数据交换通过 PNIO_CBF_AR_INDATA_IND( ) 回调函数向 IO-Base 设备用户程序发信号。

说明

只要移除子模块，就会生成“拔出报警”。

如果移除没有子模块的模块，则不会生成报警。

语法

```
PNIO_UINT32 PNIO_sub_pull(
    PNIO_UINT32          DevHndl,          //in
    PNIO_UINT32          API,              //in
    PNIO_DEV_ADDR       *pAddr             //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
API	模块的应用程序进程标识符 - 必须与通过 PNIO_api_add( ) 函数调用来传送的 API 参数之一相对应。
pAddr	指向移除了子模块的子插槽的地址的指针。

返回值

如果成功，则返回 PNIO_OK。

如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIOERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_PRM_ADD
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND

7.3.3 PNIO_sub_plug( )（插入子模块，不用于新开发）

说明

说明

此函数已废弃。 不应再使用。
请改用函数 PNIO_sub_plug_ext_IM( )。

该函数必须由 IO-Base 设备用户程序在初始化阶段调用。 它向 IO-Base 接口报告当前组态。

如果当前组态发生变化（例如发生故障或移除的子模块再次起作用），也可以在操作期间调用该函数。 在这种情况下，IO-Base 接口会自动向 IO 控制器发送插入报警。

PNIO_sub_plug( ) 是异步函数。 返回值提供有关接口是否接受作业的信息。
PNIO_CBF_PULL_PLUG_CONF( ) 回调函数提供插入确认。

说明

如果通过“插入报警”或“插入错误的子模块报警”将 IO 设备的变化通知给 IO 控制器，则 IO 控制器不会触发任何回调函数。

说明

在循环数据交换开始之前不会向 IO 控制器发送“插入报警”，该数据交换通过 PNIO_CBF_AR_INDATA_IND( ) 回调函数向 IO-Base 设备用户程序发信号。

说明

只要模块中插入子模块，就会向 IO 控制器发送“插入报警”。
如果插入模块，则不会触发报警。

语法

```
PNIO_UINT32 PNIO_sub_plug(
    PNIO_UINT32            DevHndl,          //in
    PNIO_UINT32          API,              //in
    PNIO_DEV_ADDR         *pAddr,           //in
    PNIO_UINT32            SubIdent          //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
API	子模块的应用程序进程标识符 - 必须与通过 PNIO_api_add( ) 函数调用来传送的 API 参数之一相对应。
pAddr	指向插入了子模块的子插槽的地址的指针。
	注意 在 IO 设备地址中只能为子插槽编号指定 1 到“MaxSubslots”的值。 通过调用 PNIO_api_add( ) 来指定“MaxSubslots”的值。
SubIdent	子模块 ID - 对应于 GSDML 文件的“SubmoduleIdentNumber”。

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_PRM_ADD
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND

7.3.4 PNIO_sub_plug_ext( )（扩展插入子模块，不用于新开发）

说明

说明
此函数已废弃。 不应再使用。
请改用函数 PNIO_sub_plug_ext_IM( )。

该函数由 IO-Base 设备用户程序在启动时调用。 它向 IO-Base 接口报告当前组态。

如果当前组态发生变化（例如发生故障或移除的子模块再次起作用），也可以在操作期间调用该函数。 在这种情况下，IO-Base 接口会自动向 IO 控制器发送插入报警。

PNIO_sub_plug_ext( ) 可以通知 IO 控制器插入了与组态不相符的子模块。
这意味着，新的子模块 ID 与旧的子模块 ID 既不相符也不兼容。 在这种情况下，将向 IO 控制器发送“插入错误的子模块报警”。

PNIO_sub_plug_ext( ) 是异步函数。 返回值提供有关接口是否接受作业的信息。
PNIO_CBF_PULL_PLUG_CONF( ) 回调函数提供插入确认。

说明

如果通过“插入报警”或“插入错误的子模块报警”将 IO 设备的变化通知给 IO 控制器，则不会通过带有 CP 16xx 的 IO 控制器触发任何回调函数。

说明

在循环数据交换开始之前不会向 IO 控制器发送“插入报警”，该数据交换通过 PNIO_CBF_AR_INDATA_IND( ) 回调函数向 IO-Base 设备用户程序发信号。

说明

只要模块中插入子模块，就会向 IO 控制器发送“插入报警”。
如果插入模块，则不会触发报警。

语法

```
PNIO_UINT32 PNIO_sub_plug_ext(
    PNIO_UINT32             DevHndl,             //in
    PNIO_UINT32             API,                 //in
    PNIO_DEV_ADDR         *pAddr,            //in
    PNIO_UINT32            SubIdent          //in
    PNIO_UINT32             AlarmType            //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
API	模块的应用程序进程标识符 - 必须与通过 PNIO_api_add( ) 函数调用来传送的 API 参数之一相对应。

<table>
<tr><th>名称</th><th colspan="2">说明</th></tr>
<tr><td>pAddr</td><td colspan="2">指向插入了子模块的子插槽的地址的指针。</td></tr>
<tr><td></td><td colspan="2">注意
在 IO 设备地址中只能为子插槽编号指定 1 到“MaxSubslots”的值。
通过调用 PNIO_api_add( ) 来指定“MaxSubslots”的值。</td></tr>
<tr><td>SubIdent</td><td colspan="2">相关 GSDML 文件中的子模块 ID。</td></tr>
<tr><td>报警类型</td><td colspan="2">“AlarmType”参数决定在插入新子模块后向 IO 控制器发送“插入报警”还是“插入错误的子模块报警”。
如果新子模块的子模块 ID 错误，而且与项目工程中的子模块不兼容，则必须按照标准发出“插入错误的子模块报警”。</td></tr>
<tr><td></td><td>如果要发送... ,</td><td>将“AlarmType”参数设置为...</td></tr>
<tr><td></td><td>插入报警</td><td>PNIO_ALARM_TYPE_PLUG</td></tr>
<tr><td></td><td>插入错误的子模块报警</td><td>PNIO_ALARM_TYPE_PLUG_WRONG</td></tr>
</table>

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_PRM_ADD
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND

7.4 用于在 IO 设备上写入 IO 数据的接口

在 IO 设备上，通过以下函数启动“写入 IO 数据”从 IO-Base 设备用户程序到 IO-Base 接口的数据交换：

- PNIO_initiate_data_write( )
- PNIO_initiate_data_write( )

对于每个具有输入数据并满足选择条件的子插槽，IO-Base 接口都会调用 PNIO_CBF_DATA_WRITE( ) 回调函数。

说明

PNIO_initiate_data_write( ) 和 PNIO_initiate_data_write_ext( ) 函数是同步的，换言之，仅当所有 PNIO_CBF_DATA_WRITE( ) 回调完成后，这两个函数才返回。

说明

如果在 PNIO_CBF_PRM_END_IND() 函数中调用 PNIO_initiate_data_write( ) 和 PNIO_initiate_data_read( ) 函数，它们会返回错误代码 PNIO_WARN_NO_SUBMODULES 和 PNIO_WARN_IRT_INCONSISTENT。可以忽略这些返回值。

7.4.1 PNIO_initiate_data_write( )（启动写入 RT 数据）

说明

该函数触发向 IO-Base 接口写入 IO 数据的操作。

之后，针对具有输入数据且与 IO 控制器存在 I/O 应用关系的子模块，IO-Base 接口将调用 PNIO_CBF_DATA_WRITE( ) 回调函数。 这要求 IO-Base 设备用户程序将相关 IO 数据写入过程映像。

语法

```
PNIO_UINT32  PNIO_initiate_data_write(
    PNIO_UINT32          DevHndl,              //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND

7.4.2 PNIO_initiate_data_write_ext( )（触发写入选择性数据）

说明

该函数启动向 IO-Base 接口选择性写入一次 IO 数据的操作。

之后，对于每个具有输入数据且符合传送条件的子模块，IO-Base 接口都会调用 PNIO_CBF_DATA_WRITE( ) 回调函数。 仅当与 IO 控制器存在 I/O 应用关系时才能实现此操作。

PNIO_CBF_DATA_WRITE( ) 回调函数要求 IO-Base 设备用户程序将相应的 IO 数据写入过程映像。

与 PNIO_initiate_data_write( ) 相比，PNIO_initiate_data_write_ext( ) 具有以下优势：

- 可以写入 RT 和 IRT 数据。
- 可以限制 IO-Base 接口调用 PNIO_CBF_DATA_WRITE( ) 时所针对的子模块。
- 可以自由选择 RT 模块的访问模式。

语法

```
PNIO_UINT32  PNIO_initiate_data_write_ext(
    PNIO_UINT32          DevHndl,          //in
    PNIO_DEV_ADDR       *pAddr,            //in
    PNIO_ACCESS_ENUM     AccessType        //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
pAddr	调用 PNIO_CBF_DATA_WRITE( ) 所针对的模块/子模块的地址。 如果通过“pAddr”传送 NULL，将为 IO 设备的每个子模块调用 PNIO_CBF_DATA_WRITE( )。 如果通过“pAddr”传送指向某个模块的地址，换句话说 pAddr -> Subslot 等于 0，将为指定模块的每个子模块调用 PNIO_CBF_DATA_WRITE( )。 如果通过“pAddr”指定插槽和子插槽不等于 0 的地址，则只为指定的子模块调用 PNIO_CBF_DATA_WRITE( )。
AccessType	选择 IO 数据类型和访问类型，另请参见“PNIO_APPL_READY_LIST_TYPE（应用程序就绪） (页 244)”部分。

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_PRM_ACCESS_TYPE
- PNIO_ERR_PRM_ADD
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND
- PNIO_WARN_IRT_INCONSISTENT
- PNIO_WARN_NO_SUBMODULES

7.4.3 回调函数 PNIO_CBF_DATA_WRITE( )（写入数据）

说明

IO-Base 设备用户程序发出“写入数据”后，IO-Base 接口调用 PNIO_CBF_DATA_WRITE( ) 回调函数。

IO-Base 设备用户程序必须读取已寻址子模块的物理输入，并将其复制到 IO-Base 接口指定的数据缓冲区。

本地和远程状态也会交换。

语法

```
typedef PNIO_IOXS (* PNIO_CBF_DATA_WRITE)(
    PNIO_UINT32        DevHndl,             //in
    PNIO_DEV_ADDR     *pAddr,               //in
    PNIO_UINT32     BufLen,                           //in
    PNIO_UINT8        *pBuffer,             //out
    PNIO_IOXS          IOremState           //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
pAddr	指向子模块地址的指针
BufLen	可用的数据缓冲区的长度（字节）
	注意 任何情况下都不得超出“BufLen”指定的长度。 这是用户的责任。
pBuffer	指向要写入 IO 数据的数据缓冲区的指针。
IOremState	远程状态

返回值

本地状态： PNIO_S_GOOD 或 PNIO_S_BAD

7.5 用于在 IO 设备上读取 IO 数据的接口

说明

在 IO 设备上，通过以下函数启动“读取 IO 数据”从 IO-Base 设备用户程序到 IO-Base 接口的数据交换：

- PNIO_initiate_data_read( )
- PNIO_initiate_data_read_ext( )

对于每个具有输出数据并满足选择条件的子插槽，IO-Base 接口都会调用 PNIO_CBF_DATA_READ( ) 回调函数。

说明

PNIO_initiate_data_read( ) 和 PNIO_initiate_data_read_ext( ) 函数是同步的，换言之，仅当所有 PNIO_CBF_DATA_READ( ) 回调完成后，这两个函数才返回。

如果在 PNIO_CBF_PRM_END_IND() 函数中调用 PNIO_initiate_data_write( ) 和 PNIO_initiate_data_read( ) 函数，它们会返回错误代码 PNIO_WARN_NO_SUBMODULES 和 PNIO_WARN_IRT_INCONSISTENT。

可以忽略这些返回值。

7.5.1 PNIO_initiate_data_read( )（启动读取 RT 数据）

说明

该函数将 RT IO 输出数据从 IO-Base 接口读取到 IO-Base 设备用户程序一次（数据传输方向为 IO 控制器到 IO 设备）。

之后，针对具有 RT 输出数据且与 IO 控制器存在 I/O 应用关系的子模块，IO-Base 接口将调用 PNIO_CBF_DATA_READ( ) 回调函数。 这要求 IO-Base 设备用户程序从过程映像读取相关 RT IO 数据。

语法

```
PNIO_UINT32  PNIO_initiate_data_read(
    PNIO_UINT32          DevHndl,              //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND

7.5.2 PNIO_initiate_data_read_ext( )（启动读取选择性数据）

说明

该函数启动在 IO-Base 接口上选择性读取一次 IO 数据的操作。

之后，对于每个具有输出数据且符合传送条件的子模块，IO-Base 接口都会调用 PNIO_CBF_DATA_READ( ) 回调函数。 仅当与 IO 控制器存在 I/O 应用关系时才能实现此操作。

The PNIO_CBF_DATA_READ( ) 回调函数要求 IO-Base 设备用户程序从过程映像读取相应的 IO 数据。

与 PNIO_initiate_data_read( ) 相比，PNIO_initiate_data_read_ext( ) 具有以下优势：

- 可以读取 RT 和 IRT 数据。
- 可以限制 IO-Base 接口调用 PNIO_CBF_DATA_READ( ) 时所针对的子模块。
- 可以自由选择 RT 模块的访问模式。

语法

```
PNIO_UINT32 PNIO_initiate_data_read_ext(
    PNIO_UINT32          DevHndl,          //in
    PNIO_DEV_ADDR       *pAddr,            //in
    PNIO_ACCESS_ENUM     AccessType        //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
pAddr	调用 PNIO_CBF_DATA_READ( ) 所针对的模块/子模块的地址。 如果通过“pAddr”传送 NULL，将为 IO 设备的每个子模块调用 PNIO_CBF_DATA_READ( )。 如果通过“pAddr”传送指向某个模块的地址，换句话说 pAddr -> Subslot 等于 0，将为指定模块的每个子模块调用 PNIO_CBF_DATA_READ( )。 如果通过“pAddr”指定插槽和子插槽不等于 0 的地址，则只为指定的子模块调用 PNIO_CBF_DATA_READ( )。
AccessType	选择 IO 数据类型和访问类型，另请参见“PNIO_APPL_READY_LIST_TYPE（应用程序就绪） (页 244)”部分。

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_PRM_ACCESS_TYPE
- PNIO_ERR_PRM_ADD
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND
- PNIO_WARN_IRT_INCONSISTENT
- PNIO_WARN_NO_SUBMODULES

7.5.3 回调函数 PNIO_CBF_DATA_READ( )（读取数据）

说明

IO-Base 设备用户程序通过 PNIO_initiate_data_read( ) 或 PNIO_initiate_data_read_ext( ) 发出“读取数据”后，IO-Base 接口调用该回调函数。

IO-Base 设备用户程序必须读取存储在数据缓冲区中的 IO 数据，并将其写入已寻址子模块的物理输出。

远程和本地状态也会交换。

语法

```
typedef PNIO_IOXS (* PNIO_CBF_DATA_READ)(
    PNIO_UINT32        DevHndl,              //in
    PNIO_DEV_ADDR     *pAddr,                //in
    PNIO_UINT32     BufLen,                             //in
    PNIO_UINT8     *pBuffer,                            //in
    PNIO_IOXS          IOremState            //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
pAddr	指向子模块地址的指针
BufLen	可用的数据缓冲区的长度（字节）
pBuffer	指向要读取 IO 数据的数据缓冲区的指针。
IOremState	远程状态

返回值

本地状态： PNIO_S_GOOD 或 PNIO_S_BAD

7.6 用于在 IO 设备上读取/写入数据记录的接口

概述

以下回调函数可用于 IO 设备读取数据记录和写入数据记录：

- PNIO_CBF_REC_READ( )
- PNIO_CBF_REC_WRITE( )

7.6.1 回调函数 PNIO_CBF_REC_READ( )（处理读取数据记录作业）

说明

当控制器向 IO 设备发送读取数据记录请求时，IO-Base 接口将调用 PNIO_CBF_REC_READ( ) 回调函数。

通过子模块地址和数据记录编号寻址数据记录。 IO-Base 设备用户程序从子插槽读取请求的数据，并将其写入“pBuffer”指定的地址区域。

语法

```
typedef void (* PNIO_CBF_REC_READ)(
    PNIO_UINT32           DevHndl,              //in
    PNIO_UINT32           API,                  //in
    PNIO_UINT16     ArNumber,             //in
    PNIO_UINT16     SessionKey,           //in
    PNIO_UINT32           SequenceNum,          //in
    PNIO_DEV_ADDR        *pAddr,                //in
    PNIO_UINT32     RecordIndex,                    //in
    PNIO_UINT32          *pBufLen,              //in,out
    PNIO_UINT8           *pBuffer,              //out
    PNIO_ERR_STAT        *pPnioState            //out
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
API	应用程序进程标识符
ArNumber	应用关系编号
SessionKey	当前应用关系的会话密钥
SequenceNum	数据记录的连续编号
pAddr	指向此作业的本地模块地址的指针。
	注意 还有未分配给子模块的读取数据记录作业（请参见 IEC 61158），例如数据记录编号 0xEC00 到 0xEFFF。 在这种情况下，pAddr 指向未插入模块的模块地址！
RecordIndex	数据记录号

名称	说明
pBufLen	输入： 允许的最大数据记录长度（字节） 输出： 实际写入的数据记录长度（字节）
	注意 任何情况下都不得超出“pBufLen”指定的长度。 这是用户的责任。
pBuffer	指向 IO-Base 设备用户程序将读取数据复制到的数据缓冲区的指针。
pPnioState	执行作业有关的错误代码

返回值

无返回值。

7.6.2 回调函数 PNIO_CBF_REC_WRITE( )（处理写入数据记录作业）

说明

当控制器向 IO 设备发送写入数据记录请求时，IO-Base 接口将调用 PNIO_CBF_REC_WRITE( ) 回调函数。

通过本地插槽/子插槽地址和数据记录编号寻址数据记录。 IO-Base 设备用户程序接受从“pBuffer”指定的地址开始的数据记录数据，并将其写入子插槽。

语法

```
typedef void (* PNIO_CBF_REC_WRITE)(
    PNIO_UINT32           DevHndl,             //in
    PNIO_UINT32           API,                 //in
    PNIO_UINT16     ArNumber,            //in
    PNIO_UINT16     SessionKey,          //in
    PNIO_UINT32           SequenceNum,         //in
    PNIO_DEV_ADDR        *pAddr,               //in
    PNIO_UINT32     RecordIndex,                  //in
    PNIO_UINT32          *pBufLen,             //in
    PNIO_UINT8     *pBuffer,                        //in
    PNIO_ERR_STAT        *pPnioState           //out
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
API	应用程序进程标识符
ArNumber	应用关系编号
SessionKey	当前应用关系的会话密钥
SequenceNum	数据记录的连续编号
pAddr	**注意** 还有未分配给子模块的写入数据记录作业（请参见 IEC 61158），例如数据记录编号 0xEC00 到 0xEFFF。 在这种情况下，“pAddr”指向未插入模块的模块地址！
RecordIndex	数据记录号
pBufLen	数据量（字节）
pBuffer	指向记录数据的指针
pPnioState	执行作业有关的错误代码

返回值

无返回值。

7.7 用于在 IO 设备上管理诊断数据的接口

概述

以下函数可用于诊断 IO 设备：

- PNIO_build_channel_properties( )
- PNIO_diag_channel_add( )
- PNIO_diag_ext_channel_add( )
- PNIO_diag_channel_remove( )
- PNIO_diag_ext_channel_remove( )

- PNIO_diag_generic_add( )
- PNIO_diag_generic_remove( )

7.7.1 PNIO_build_channel_properties( )（生成通道属性）

说明

使用此函数，IO-Base 设备用户程序可以生成用于通道诊断数据记录的压缩位结构。可通过 PNIO_diag_channel_add( ) 函数将该位结构下载到子插槽。

语法

```
PNIO_UINT16 PNIO_build_channel_properties(
    PNIO_UINT32    DevHndl,                //in
    PNIO_UINT16    Type,                   //in
    PNIO_UINT16    Spec,                   //in
    PNIO_UINT16    Dir                     //in
);
```

参数

名称	说明	
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄	
Type	通道诊断属性的压缩大小 可能的值如下： • PNIO_DIAG_CHANPROP_TYPE_SUBMOD • PNIO_DIAG_CHANPROP_TYPE_1BIT • PNIO_DIAG_CHANPROP_TYPE_2BIT • PNIO_DIAG_CHANPROP_TYPE_4BIT • PNIO_DIAG_CHANPROP_TYPE_BYTE • PNIO_DIAG_CHANPROP_TYPE_WORD • PNIO_DIAG_CHANPROP_TYPE_DWORD • PNIO_DIAG_CHANPROP_TYPE_LWORD	
Spec	通道诊断属性的含义：	
	值	说明

名称	说明	
	PNIO_DIAG_CHAN_SPEC_ERROR_APPEARS	错误的进入状态。
	PNIO_DIAG_CHAN_SPEC_ERROR_DISAPP_FREE	错误的退出状态，没有其它问题。
	PNIO_DIAG_CHAN_SPEC_ERROR_DISAPP_REMAIN	错误的退出状态，存在其它问题。
Dir	将为其创建诊断数据的模块的数据方向。	
	值	**说明**
	PNIO_DIAG_CHAN_DIRECTION_MANUFACTURE	供应商特定
	PNIO_DIAG_CHAN_DIRECTION_INPUT	输入
	PNIO_DIAG_CHAN_DIRECTION_OUTPUT	输出
	PNIO_DIAG_CHAN_DIRECTION_BIDIRECT	输入/输出

返回值

通道属性的压缩位结构。

7.7.2 PNIO_diag_channel_add( )（将通道诊断数据存储在子插槽中）

说明

该函数用于将通道诊断数据记录下载到子插槽中。

说明

可以将多条诊断数据记录下载到子插槽中。 这些记录由 DiagTag 引用。

语法

```
PNIO_UINT32  PNIO_diag_channel_add(
    PNIO_UINT32          DevHndl,            //in
    PNIO_UINT32          API,                //in
    PNIO_DEV_ADDR       *pAddr,              //in
    PNIO_UINT16          ChannelNum,         //in
    PNIO_UINT16          ChannelProp,        //in
    PNIO_UINT32          ChannelErrType,     //in
    PNIO_UINT16          DiagTag             //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
API	应用程序进程标识符 - 必须与子模块所属的 API 参数相对应。
pAddr	指向本地子插槽地址的指针，诊断数据记录将下载到该地址。
ChannelNum	通道编号。 值范围 0x8000（请参见 IEC 61158）
ChannelProp	“ChannelProp”参数由 PNIO_build_channel_properties( ) 函数生成。 PNIO_build_channel_properties( ) 函数必须使用以下参数调用： • Type = PNIO_DIAG_CHANPROP_TYPE_SUBMOD • Dir = PNIO_DIAG_CHAN_DIRECTION_MANUFACTURE（供应商特定） • Spec = PNIO_DIAG_CHAN_SPEC_ERROR_APPEARS（错误的进入状态） 诊断数据 - PNIO_build_channel_properties( ) 函数的返回值
ChannelErrType	通道错误类型 - 可在头文件中找到的带“PNIO_DIAG_CHAN_ERROR_”前缀的可能值。
DiagTag	用户指定的对诊断数据记录的唯一引用 (!=0)

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_PRM_ADD
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND

7.7.3 PNIO_diag_ext_channel_add( )（将扩展通道诊断数据存储在子插槽中）

说明

该函数用于将扩展通道诊断数据记录下载到子插槽中。

说明

可以将多条诊断数据记录下载到子插槽中。 这些记录由“DiagTag”引用。

语法

```
PNIO_UINT32  PNIO_diag_ext_channel_add(
    PNIO_UINT32          DevHndl,            //in
    PNIO_UINT32          API,                //in
    PNIO_DEV_ADDR       *pAddr,              //in
    PNIO_UINT16          ChannelNum,          //in
    PNIO_UINT16          ChannelProp,         //in
    PNIO_UINT16          ChannelErrType,     //in
    PNIO_UINT16          ExChannelErrType,   //in
    PNIO_UINT32          ExChannelAddValue,  //in
    PNIO_UINT16          DiagTag              //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
API	应用程序进程标识符 - 必须与子模块所属的 API 参数相对应。
pAddr	指向本地子插槽地址的指针，诊断数据记录将下载到该地址。
ChannelNum	通道编号 - 值范围 0x8000（请参见 IEC 61158）
ChannelProp	“ChannelProp”参数由 PNIO_build_channel_properties( ) 函数生成。 PNIO_build_channel_properties( ) 函数必须使用以下参数调用： • Type = PNIO_DIAG_CHANPROP_TYPE_SUBMOD • Dir = PNIO_DIAG_CHAN_DIRECTION_MANUFACTURE（供应商特定） • Spec = PNIO_DIAG_CHAN_SPEC_ERROR_APPEARS（错误的进入状态） 诊断数据 - PNIO_build_channel_properties( ) 函数的返回值
ChannelErrType	通道错误类型 - 可在头文件中找到的带“PNIO_DIAG_CHAN_ERROR_”前缀的可能值。
ExtChannelErrType	“ExtChannelErrorType”参数可用于传送有关通道错误的其它信息；有关可能值，请参见 PROFINET IO 标准 IEC 61158。
ExtChannelAddValue	“ExtChannelErrorType”参数可用于传送除通道错误类型之外的其它信息；有关可能值，请参见 PROFINET IO 标准 IEC 61158。
DiagTag	用户指定的对诊断数据记录的唯一引用 (!=0)

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION

- PNIO_ERR_PRM_ADD
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND

7.7.4 PNIO_diag_channel_remove( )（从子插槽中删除诊断数据）

说明

使用该函数，可以删除子插槽中加载的通道诊断数据记录。

语法

```
PNIO_UINT32  PNIO_diag_channel_remove(
    PNIO_UINT32             DevHndl,          //in
    PNIO_UINT32             API,              //in
    PNIO_DEV_ADDR          *pAddr,            //in
    PNIO_UINT16          DiagTag              //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
API	应用程序进程标识符 - 必须与子模块所属的 API 参数相对应。
pAddr	指向本地子插槽地址的指针，将从该地址删除诊断数据记录。
DiagTag	• DiagTag ≠ 0 - 用户指定的对要删除的诊断数据记录的唯一引用。 • DiagTag = 0 - 删除子模块的所有通道诊断数据记录。
	注意 如果使用无效的“DiagTag”调用 PNIO_diag_channel_remove( ) 函数，则不会产生任何错误消息。

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_PRM_ADD
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND

7.7.5 PNIO_diag_ext_channel_remove( )（从子插槽中删除扩展通道诊断数据）

说明

使用该函数，可以删除子插槽中加载的扩展通道诊断数据记录。

语法

```
PNIO_UINT32  PNIO_diag_channel_remove(
    PNIO_UINT32              DevHndl,           //in
    PNIO_UINT32              API,               //in
    PNIO_DEV_ADDR           *pAddr,             //in
    PNIO_UINT16           DiagTag              //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
API	应用程序进程标识符 - 必须与子模块所属的 API 参数相对应。

名称	说明
pAddr	指向本地子插槽地址的指针，将从该地址删除诊断数据记录。
DiagTag	• DiagTag ≠ 0 - 用户指定的对要删除的诊断数据记录的唯一引用。 • DiagTag = 0 - 删除子模块的所有扩展通道诊断数据记录。
	注意 如果使用无效的“DiagTag”调用 PNIO_diag_ext_channel_remove( ) 函数，则不会产生任何错误消息。

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_PRM_ADD
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND

7.7.6 PNIO_diag_generic_add( )（将供应商特定诊断数据存储在子插槽中）

说明

该函数用于将制造商特定的诊断数据记录下载到子插槽中。

说明
可以将多条制造商特定的诊断数据记录下载到子模块中。 这些记录由“DiagTag”引用。

语法

```
PNIO_UINT32  PNIO_diag_generic_add(
    PNIO_UINT32        DevHndl,            //in
    PNIO_UINT32        API,                //in
    PNIO_DEV_ADDR     *pAddr,              //in
    PNIO_UINT16          ChannelNum,          //in
    PNIO_UINT16          ChannelProp,         //in
    PNIO_UINT16        DiagTag,            //in
    PNIO_UINT16        UserStructIdent,    //in
    PNIO_UINT8        *pInfoData,          //in
    PNIO_UINT32        InfoDataLen         //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
API	应用程序进程标识符 - 必须与子模块所属的 API 参数相对应。
pAddr	指向本地子插槽地址的指针，诊断数据记录将下载到该地址。
ChannelNum	通道编号 - 按照 PROFINET IO 标准，供应商特定诊断的“ChannelNumber”总是 0x0 到 0x7FFF（请参见 IEC 61158）！
ChannelProp	诊断数据 - PNIO_build_channel_properties( ) 函数的返回值
DiagTag	用户指定的对诊断数据记录的唯一引用 (!=0)
UserStructIdent	诊断数据的结构；请参见 IEC 61158， 值范围为 0x0 到 0x7FFF 的用户结构标识符。
pInfoData	指向网络格式（大端）诊断数据的指针
InfoDataLen	诊断数据的长度（字节），最大值 1404。

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_PRM_ADD

- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_LEN
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND

7.7.7 PNIO_diag_generic_remove( )（从子模块中删除供应商特定诊断数据）

说明

使用该函数，可以删除子模块中加载的供应商特定诊断数据记录。

语法

```
PNIO_UINT32  PNIO_diag_generic_remove(
    PNIO_UINT32             DevHndl,          //in
    PNIO_UINT32             API,              //in
    PNIO_DEV_ADDR          *pAddr,            //in
    PNIO_UINT16          DiagTag              //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
API	应用程序进程标识符 - 必须与子模块所属的 API 参数相对应。
pAddr	指向本地子模块地址的指针，将从该地址删除诊断数据记录。
DiagTag	• DiagTag ≠ 0 - 用户指定的对要删除的诊断数据记录的唯一引用。 • DiagTag = 0 - 删除子模块的所有供应商特定通道诊断数据记录。
	注意 如果使用无效的“DiagTag”调用 PNIO_diag_generic_remove( ) 函数，则不会产生任何错误消息。

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_PRM_ADD
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND

7.8 IO 设备的报警接口

概述

以下函数可用于处理 IO 设备和 IO 控制器之间的报警：

- PNIO_process_alarm_send( )
- PNIO_diag_alarm_send( )
- PNIO_ret_of_sub_alarm_send( )
- PNIO_CBF_REQ_DONE( )

7.8.1 PNIO_process_alarm_send( )（发送过程报警）

说明

该函数可将子模块特定过程报警从 IO 设备发送到 IO 控制器。 通过 PNIO_CBF_REQ_DONE( ) 回调函数表示异步作业的结果。

报警数据不储存在子模块上。 函数的用户也可以在 pData 中添加用户特定的报警数据。

语法

```
PNIO_UINT32  PNIO_process_alarm_send(
    PNIO_UINT32       DevHndl,              //in
    PNIO_UINT32       API,                  //in
    PNIO_UINT16     ArNumber,             //in
    PNIO_UINT16     SessionKey,           //in
    PNIO_DEV_ADDR    *pAddr,                //in
    PNIO_UINT8       *pData,                //in
    PNIO_UINT32       DataLen,              //in
    PNIO_UINT16        UserStructIdent,    //in
    PNIO_UINT32       UserHndl              //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
API	应用程序进程标识符 - 必须与子模块所属的 API 参数相对应。
ArNumber	PNIO_CBF_AR_CHECK_IND( ) 中的应用关系编号 - 有关回调的说明，请参见“回调函数 PNIO_CBF_AR_CHECK_IND( )（报告应用关系检查指示） (页 222)”部分。
SessionKey	PNIO_CBF_AR_CHECK_IND( ) 中的会话密钥 - 有关回调的说明，请参见“回调函数 PNIO_CBF_AR_CHECK_IND( )（报告应用关系检查指示） (页 222)”部分。
pAddr	指向生成报警的本地子模块地址的指针。
pData	包含网络格式（大端）报警数据的数据缓冲区。
DataLen	数据长度（字节）- 根据控制器的能力，IO 设备应用必定会拒绝“DataLen”为 172 字节或更多字节的服务。 如果 IO 控制器是 CP 1616 或 CP 1604，“DataLen”的最大值为 172 个字节。
UserStructIdent	有关诊断数据的结构，请参见 IEC 61158/用户结构标识符。
UserHndl	IO-Base 设备用户程序分配的句柄，用于在通过 PNIO_CBF_REQ_DONE( ) 收到确认时识别报警。

返回值

如果成功，则异步返回 PNIO_OK。
如果发生错误，可能出现以下同步或异步返回值（有关值的含义，请参见头文件“pnioerrx .h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_PRM_ADD
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_LEN
- PNIO_ERR_SEQUENCE
- PNIO_ERR_SESSION
- PNIO_ERR_UNKNOWN_ADDR
- PNIO_ERR_VALUE_LEN
- PNIO_ERR_WRONG_HND

7.8.2 PNIO_diag_alarm_send( )（发送诊断报警）

说明

该函数可将子模块特定诊断报警从 IO 设备发送到 IO 控制器。 通过 PNIO_CBF_REQ_DONE( ) 回调函数表示异步作业的结果。

诊断报警可以是进入或退出状态。 在发出进入状态报警前，必须通过 PNIO_diag_channel_add( )、PNIO_diag_ext_channel_add( ) 或 PNIO_diag_generic_add( ) 将相关的诊断数据报告传送到 IO-Base 接口，以便在以后读出报警数据。

如果报警原因不再存在，必须先通过 PNIO_diag_channel_remove( )、PNIO_diag_ext_channel_remove( ) 或 PNIO_diag_generic_remove( ) 删除数据记录，然后触发相应的退出状态报警。

语法

```
PNIO_UINT32  PNIO_diag_alarm_send(
    PNIO_UINT32          DevHndl,            //in
    PNIO_UINT32          API,                //in
    PNIO_UINT16     ArNumber,            //in
    PNIO_UINT16     SessionKey,          //in
    PNIO_UINT32          AlarmState,         //in
    PNIO_DEV_ADDR       *pAddr,              //in
    PNIO_UINT8          *pData,              //in
    PNIO_UINT32       DataLen,              //in
    PNIO_UINT16        UserStructIdent,    //in
    PNIO_UINT32       UserHndl              //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
API	应用程序进程标识符 - 必须与子模块所属的 API 参数相对应。
ArNumber	PNIO_CBF_AR_CHECK_IND( ) 中的应用关系编号 - 有关回调的说明，请参见“回调函数 PNIO_CBF_AR_CHECK_IND( )（报告应用关系检查指示） (页 222)”部分。
SessionKey	PNIO_CBF_AR_CHECK_IND( ) 中的会话密钥 - 有关回调的说明，请参见“回调函数 PNIO_CBF_AR_CHECK_IND( )（报告应用关系检查指示） (页 222)”部分。
AlarmState	指示诊断报警是进入还是退出状态，可能值有： • PNIO_STATE_ALARM_APPEARS • PNIO_STATE_ALARM_DISAPPEARS
pAddr	指向生成报警的本地子模块地址的指针。
	注意 如果错误地指定“插槽”或“子插槽”结构元素，该函数的返回值中不会有任何错误消息。

名称	说明
pData	指向网络格式（大端）报警数据的指针 - 必须通过“pData”指针传送诊断报警的诊断数据。 该诊断数据必须与之前通过 PNIO_diag_generic_add( ) 或 PNIO_diag_channel_add( ) 函数传送的诊断数据相对应。 如果使用了 PNIO_diag_generic_add( ) 函数传送诊断数据，则诊断数据与“pInfoData”指向的数据相对应。 如果使用了 PNIO_diag_channel_add( ) 函数传送诊断数据，则诊断数据必须以包含以下值的字段的形式传送： • ChannelNum（大端） • ChannelProp（大端） • ChannelErrorType（大端） 如果传送“pData = NULL”，则表示所有之前设置的报警均退出报警状态。
	注意 “pData”指针必须指向“用户结构标识符”数据元素之后的报警数据。 “Alarm ASE.pdf”文件中介绍了报警数据。 您可以在产品 CD“DK-16xx PN IO”的“doc”文件夹中找到该文档。
DataLen	使用“pData”传送的数据的长度（字节）。
	注意 当所有报警都在退出状态时，传送 DataLen = 0。 根据 IO 控制器的能力，IO 设备应用必定会拒绝“DataLen”为 172 字节或更多字节的服务。如果 IO 控制器是 CP 1616 或 CP 1604，“DataLen”的最大值为 172 个字节。 （参数“DataLen”的最大值受“AlarmNotification”PDU 最大长度的限制。 根据 PROFINET IO 标准，网络上 PDU 的最大长度取决于 IO 控制器，其值在 200 到 1432 字节之间（包括 PDU 报头）。 每次发送的 PDU 的报头长度为 28 字节，也就是说，IO 设备应用必定会拒绝“DataLen”值为 172 或更大值的服务。

名称	说明	
UserStructIdent	用户结构标识符 - 提供有关“pData”指向的数据的信息。	
	0x0000 到 0x7FFF	“pData”中的供应商特定数据，另请参见“PNIO_diag_generic_add( )（将供应商特定诊断数据存储在子插槽中）(页 206)”函数。
	0x8000	“pData”中的通道诊断数据
		注意 当使用 PNIO_build_channel_properties 创建通道属性时，必须为“Type”传送值 PNIO_DIAG_CHANPROP_TYPE_SUBMOD，为“Dir”传送值 PNIO_DIAG_CHAN_DIRECTION_MANUFACTURE。
	0x8001	不得设置此值。
		注意 在一个报警通知 PDU 中，不能向 IO 控制器发送多条诊断报警。 这意味着，“UserStructIdent”参数不得设置为值 0x8001。
	0x8002	扩展通道诊断数据
UserHndl	IO-Base 设备用户程序分配的句柄，用于在通过 PNIO_CBF_REQ_DONE( ) 收到确认时识别报警。	

返回值

如果成功，则异步返回 PNIO_OK。
如果发生错误，可能出现以下同步或异步返回值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_PRM_ADD
- PNIO_ERR_PRM_BUF
- PNIO_ERR_PRM_LEN

- PNIO_ERR_SEQUENCE
- PNIO_ERR_SESSION
- PNIO_ERR_UNKNOWN_ADDR
- PNIO_ERR_VALUE_LEN
- PNIO_ERR_WRONG_HND

7.8.3 PNIO_ret_of_sub_alarm_send( )（发送子模块返回报警）

说明

如果随用户数据 IOPS/IOCS发送的值的状态从 BAD 变为 GOOD，IO 设备将使用此函数向 IO 控制器发送子模块特定的子模块返回报警。 IO 控制器的反应与出现模块插入报警时相似，但在这种情况下，不会为子模块分配新的参数设置。

通过 PNIO_CBF_REQ_DONE( ) 回调函数表示异步作业的结果。

语法

```
PNIO_UINT32  PNIO_ret_of_sub_alarm_send(
    PNIO_UINT32        DevHndl,           //in
    PNIO_UINT32        API,               //in
    PNIO_UINT16     ArNumber,            //in
    PNIO_UINT16     SessionKey,          //in
    PNIO_DEV_ADDR     *pAddr,            //in
    PNIO_UINT32       UserHndl             //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
API	应用程序进程标识符 - 必须与子模块所属的 API 参数相对应。
ArNumber	PNIO_CBF_AR_CHECK_IND( ) 中的应用关系编号 - 有关回调的说明，请参见“回调函数 PNIO_CBF_AR_CHECK_IND( )（报告应用关系检查指示） (页 222)”部分。

名称	说明
SessionKey	PNIO_CBF_AR_CHECK_IND( ) 中的会话密钥 - 有关回调的说明，请参见“回调函数 PNIO_CBF_AR_CHECK_IND( )（报告应用关系检查指示）(页 222)”部分。
pAddr	指向生成报警的本地子模块地址的指针。
	注意 如果错误地指定“插槽”或“子插槽”结构元素，该函数的返回值中不会有任何错误消息。
UserHndl	IO-Base 设备用户程序分配的句柄，用于在通过 PNIO_CBF_REQ_DONE( ) 收到确认时识别报警。

返回值

如果成功，则异步返回 PNIO_OK。
如果发生错误，可能出现以下同步或异步返回值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_PRM_ADD
- PNIO_ERR_SEQUENCE
- PNIO_ERR_SESSION
- PNIO_ERR_UNKNOWN_ADDR
- PNIO_ERR_WRONG_HND

7.8.4 回调函数 PNIO_CBF_REQ_DONE( )（信号报警确认）

说明

IO 控制器通过 PNIO_CBF_REQ_DONE( ) 回调函数确认由 IO 设备发出的报警。

语法

```
typedef void (* PNIO_CBF_REQ_DONE)(
    PNIO_UINT32         DevHndl,            //in
    PNIO_UINT32         UserHndl,           //in
    PNIO_UINT32         Status,             //in
    PNIO_ERR_STAT      *pPnioState          //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
UserHndl	调用报警时 IO-Base 设备用户程序分配的句柄。
Status	如果成功，则返回 PNIO_OK。 如果发生错误，请参见头文件“pnioerrx.h”。
pPnioState	PROFINET IO 错误代码随报警确认一起传送。

返回值

无返回值。

7.9 用于建立和终止 IO 设备连接的接口

概述

以下函数可用于建立连接建立和终止，以及表示 IO 设备的操作状态：

- PNIO_device_ar_abort( )
- PNIO_set_appl_state_ready( )
- PNIO_CBF_CHECK_IND( )
- PNIO_CBF_AR_INFO_IND( )
- PNIO_CBF_AR_INDATA_IND( )
- PNIO_CBF_AR_ABORT_IND( )
- PNIO_CBF_AR_OFFLINE_IND( )

- PNIO_CBF_APDU_STATUS_IND( )
- PNIO_CBF_PRM_END_IND( )

7.9.1 PNIO_device_ar_abort( )（强制连接中止）

说明

该函数用于中止应用关系。 结果是 IO 控制器与 IO 设备之间的连接终止。

如果 IO 控制器与 IO 设备之间存在数据交换（接口调用了 PNIO_CBF_AR_INDATA_IND( ) 回调函数），IO-Base 接口将调用 PNIO_CBF_AR_OFFLINE_IND( )。

如果 IO 控制器与 IO 设备之间没有数据交换（接口没有调用 PNIO_CBF_AR_INDATA_IND( ) 回调函数），IO-Base 接口将调用 PNIO_CBF_AR_ABORT_IND( ) 回调函数。

语法

```
PNIO_UINT32  PNIO_device_ar_abort(
    PNIO_UINT32          DevHndl,                //in
    PNIO_UINT16     ArNumber,            //in
    PNIO_UINT16     SessionKey               //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
ArNumber	PNIO_CBF_AR_CHECK_IND( ) 中的应用关系编号 - 有关回调的说明，请参见“回调函数 PNIO_CBF_AR_CHECK_IND( )（报告应用关系检查指示） (页 222)”部分。
SessionKey	PNIO_CBF_AR_CHECK_IND( ) 中的会话密钥 - 有关回调的说明，请参见“回调函数 PNIO_CBF_AR_CHECK_IND( )（报告应用关系检查指示） (页 222)”部分。

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_SEQUENCE
- PNIO_ERR_SESSION
- PNIO_ERR_WRONG_HND

7.9.2 PNIO_set_appl_state_ready( )（数据交换就绪信号）

说明

该函数向 IO 控制器传送“应用程序就绪”和所有无功能子模块。 用户程序通知 IO-Base 接口已准备好交换数据。

语法

```
PNIO_UINT32 PNIO_set_appl_state_ready(
    PNIO_UINT32                    DevHndl,          //in
    PNIO_UINT16     ArNumber,             //in
    PNIO_UINT16     SessionKey,           //in
    PNIO_APPL_READY_LIST_TYPE    *pList             //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
ArNumber	PNIO_CBF_AR_CHECK_IND( ) 中的应用关系编号 - 有关回调的说明，请参见“回调函数 PNIO_CBF_AR_CHECK_IND( )（报告应用关系检查指示） (页 222)”部分。

名称	说明
SessionKey	PNIO_CBF_AR_CHECK_IND( ) 中的会话密钥 - 有关回调的说明，请参见“回调函数 PNIO_CBF_AR_CHECK_IND( )（报告应用关系检查指示） (页 222)”部分。
pList	无功能子模块的链表，请参见“PNIO_APPL_READY_LIST_TYPE （应用程序就绪） (页 244)”部分。

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND

7.9.3 回调函数 PNIO_CBF_CHECK_IND( )（信号检查指示）

说明

如果在 IO 设备上检测到预期组态和实际组态之间有差别，IO-Base 接口将调用 PNIO_CBF_CHECK_IND( ) 回调函数。 在这种情况下，IO-Base 设备用户程序必须提供有关实际插入的模块的模块/子模块 ID 和模块/子模块状态的特定信息。

不会为缺失的模块和子模块调用 PNIO_CBF_CHECK_IND( )。
这些模块和子模块会自动报告为缺失。 这意味着 PNIO_CBF_CHECK_IND( ) 不需要值 PNIO_MOD_STATE_NO_MODULE 和 PNIO_SUB_CHECK_NO_SUBMODULE。

以下值作为模块/子模块值返回：

值	模块/子模块状态
PNIO_MOD_STATE_PROPER_MODULE	插入已组态模块时。
PNIO_MOD_STATE_SUBSTITUTED_MODULE	插入替代模块时。
PNIO_MOD_STATE_WRONG_MODULE	插入错误模块时。
PNIO_SUB_CHECK_PROPER_MODULE	插入已组态子模块时。
PNIO_SUB_CHECK_SUBSTITUTED_MODULE	插入替代子模块时。
PNIO_SUB_CHECK_WRONG_MODULE	插入错误子模块时。

语法

```
typedef void (* PNIO_CBF_CHECK_IND)(
    PNIO_UINT32         DevHndl,            //in
    PNIO_UINT32         API,                //in
    PNIO_UINT16     ArNumber,            //in
    PNIO_UINT16     SessionKey,          //in
    PNIO_DEV_ADDR     *pAddr,               //in
    PNIO_UINT32       *pModIdent,          //out
    PNIO_UINT16       *pModState,          //out
    PNIO_UINT32       *pSubIdent,          //out
    PNIO_UINT16       *pSubState           //out
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
API	应用程序进程标识符
ArNumber	应用关系编号
SessionKey	当前应用关系的会话密钥
pAddr	指向未插入合适子模块的子插槽的地址的指针。

名称	说明
pModIdent	指向缓冲区的指针，IO-Base 设备用户程序必须将实际插入的模块的模块标识写入该缓冲区。 对应于 GSDML 文件的“ModuIIdentNumber”的模块标识。
pModState	指向缓冲区的指针，IO-Base 设备用户程序必须将模块状态写入该缓冲区。可能的值包括： • PNIO_MOD_STATE_WRONG_MODULE • PNIO_MOD_STATE_SUBSTITUTED_MODULE • PNIO_MOD_STATE_PROPER_MODULE
pSubIdent	指向缓冲区的指针，IO-Base 设备用户程序将实际插入的子模块的模块标识写入该缓冲区。
pSubState	指向缓冲区的指针，IO-Base 设备用户程序将子模块状态写入该缓冲区。可能的值包括： • PNIO_SUB_CHECK_WRONG_SUBMODULE • PNIO_SUB_CHECK_SUBSTITUTED_SUBMODULE • PNIO_SUB_CHECK_PROPER_SUBMODULE

返回值

无返回值。

7.9.4 回调函数 PNIO_CBF_AR_CHECK_IND( )（报告应用关系检查指示）

说明

IO-Base 接口通过调用该回调函数将应用关系属性通知给 IO-Base 设备用户程序。

该回调函数会将参数 `pAr -> ArNumber` 和 `pAr -> SessionKey` 的新值传送到用户应用程序，并且必须由应用程序在诊断和报警功能中使用。

每个连接的会话密钥必须单独存储（“ArNumber”）且在后续的所有函数调用中与“ArNumber”参数一起使用。

说明

“pAr”中存在的模块列表以及其它值始终为空！

语法

```
typedef void (* PNIO_CBF_AR_CHECK_IND)(
    PNIO_UINT32       DevHndl,                  //in
    PNIO_UINT32       HostIp,                   //in
    PNIO_UINT16       ArType,                   //in
    PNIO_UUID_TYPE    ArUUID,                   //in
    PNIO_UINT32       ArProperties,             //in
    PNIO_UUID_TYPE    CmiObjUUID,               //in
    PNIO_UINT16       CmiStationNameLength,     //in
    PNIO_UINT8       *pCmiStationName,          //in
    PNIO_AR_TYPE     *pAr                       //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
HostIp	IO 控制器的 IP 地址
ArType	连接建立的类型
ArUUID	应用关系 UUID
ArProperties	应用关系的属性
CmiObjUUID	连接建立启动器（例如 IO 控制器）的 UUID。
CmiStationName Length	下一个域所指向的站名称的长度（字节）。
pCmiStationName	连接建立启动器（例如 IO 控制器）的站名称。
pAr	指向 PNIO_AR_TYPE 结构的指针 - 应用程序必须存储参数 `pAr -> ArNumber` 和 `pAr -> SessionKey`。

返回值

无返回值。

7.9.5 回调函数 PNIO_CBF_AR_INFO_IND( )（报告应用关系信息）

说明

IO-Base 接口使用回调函数 PNIO_CBF_AR_INFO_IND( )
将在该应用关系中运行的模块及子模块通知给 IO-Base 设备用户程序。
使用传送参数“pAR”，可传送指向连锁结构的指针。
该结构包含所有的重要信息，例如，已建立的应用关系包含哪些模块及子模块。

说明

IO-Base 设备用户程序必须期望建立多个 AR。 例如，工程师站可通过另一个管理器 AR 在 IO 设备中查询 I&M 数据记录。

语法

```
typedef void (* PNIO_CBF_AR_INFO_IND)(
    PNIO_UINT32     DevHndl,                //in
    PNIO_UINT16     ArNumber,               //in
    PNIO_UINT16     SessionKey,             //in
    PNIO_AR_TYPE   *pAR                     //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
ArNumber	应用关系编号
SessionKey	当前应用关系的会话密钥
pAr	指向连锁应用关系结构的指针；请参见“PNIO_AR_TYPE（应用关系） (页 249)”部分。

返回值

无返回值。

7.9.6 回调函数 PNIO_CBF_AR_INDATA_IND( )（报告应用关系 InData）

说明

IO-Base 接口使用回调函数 PNIO_CBF_AR_INDATA_IND( ) 通知 IO-Base 设备用户程序已启动周期性数据交换。

说明

在 PNIO_set_appl_ready( ) 函数返回结果前，可由 IO-Base 编程接口调用回调函数 PNIO_CBF_AR_INDATA_IND( )。

语法

```
typedef void (* PNIO_CBF_AR_INDATA_IND)(
    PNIO_UINT32     DevHndl,                //in
    PNIO_UINT16     ArNumber,               //in
    PNIO_UINT16     SessionKey              //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
ArNumber	应用关系编号
SessionKey	当前应用关系的会话密钥

返回值

无返回值。

7.9.7 回调函数 PNIO_CBF_AR_ABORT_IND( )（报告中止事件）

说明

在报告应用关系 InData 前，IO-Base 接口使用回调函数 PNIO_CBF_AR_ABORT_IND( ) 将 IO 控制器中断或终止现有连接的时间通知给 IO-Base 设备用户程序。

语法

```
typedef void (* PNIO_CBF_AR_ABORT_IND)(
    PNIO_UINT32     DevHndl,              //in
    PNIO_UINT16     ArNumber,             //in
    PNIO_UINT16     SessionKey,           //in
    PNIO_AR_REASON  ReasonCode            //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
ArNumber	应用关系编号
SessionKey	当前应用关系的会话密钥
ReasonCode	连接中止的原因，请参见 PNIO_APPL_READY_LIST_TYPE（应用程序就绪） (页 244)部分。

返回值

无返回值。

7.9.8 回调函数 PNIO_CBF_AR_OFFLINE_IND( )（报告离线事件）

说明

在报告应用关系 InData 后，IO-Base 接口使用回调函数 PNIO_CBF_AR_OFFLINE_IND( ) 将 IO 控制器中断或终止现有连接的时间作为离线事件通知给 IO-Base 设备用户程序。

语法

```
typedef void (*PNIO_CBF_AR_OFFLINE_IND)(
    PNIO_UINT32     DevHndl,              //in
    PNIO_UINT16     ArNumber,             //in
    PNIO_UINT16     SessionKey,           //in
    PNIO_AR_REASON  ReasonCode            //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
ArNumber	应用关系编号
SessionKey	当前应用关系的会话密钥
ReasonCode	连接中止的原因，请参见 PNIO_APPL_READY_LIST_TYPE（应用程序就绪） (页 244)部分。

返回值

无返回值。

7.9.9 回调函数 PNIO_CBF_APDU_STATUS_IND( )（报告 IO 控制器状态）

说明

IO-Base 接口使用回调函数 PNIO_CBF_APDU_STATUS_IND( ) 将 IO 控制器状态已更改这一情况通知给 IO-Base 设备用户程序。

语法

```
typedef void (* PNIO_CBF_APDU_STATUS_IND)(
    PNIO_UINT32             DevHndl,            //in
    PNIO_UINT16             ArNumber,           //in
    PNIO_UINT16             SessionKey,         //in
    PNIO_APDU_STATUS_IND    ApduStatus          //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
ArNumber	应用关系编号

名称	说明
SessionKey	当前应用关系的会话密钥
ApduStatus	IO 控制器的新状态，请参见“PNIO_APDU_STATUS_IND（IO 控制器状态） (页 253)”部分

返回值

无返回值。

7.9.10 回调函数 PNIO_CBF_PRM_END_IND( )（报告 IO 控制器分配参数结束）

说明

IO-Base 接口使用回调函数 PNIO_CBF_PRM_END_IND( ) 将 IO 控制器已完成参数分配通知给 IO-Base 设备用户程序。

语法

```
typedef void (* PNIO_CBF_PRM_END_IND)(
    PNIO_UINT32     DevHndl,            //in
    PNIO_UINT16     ArNumber,           //in
    PNIO_UINT16     SessionKey,         //in
    PNIO_UINT32     API,                //in
    PNIO_UINT32     SlotNum,            //in
    PNIO_UINT32     SubslotNum          //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
ArNumber	应用关系编号
SessionKey	当前应用关系的会话密钥
API	应用程序进程标识符 - 仅当 SubslotNr 不等于 0 时才有效。

名称	说明	
SlotNum	插槽号（仅当 SubslotNr 不等于 0 时才有效。）	
SubslotNum	子插槽号	
	0	所有子模块的参数分配结束，例如，在建立连接时。
	否则	该子模块的参数分配结束，例如，稍后插入子模块时。

返回值

无返回值。

7.10 用 LED 发出指示的站

概述

该接口由以下回调函数组成：

- PNIO_CBF_START_LED_FLASH( )
- PNIO_CBF_STOP_LED_FLASH( )

7.10.1 PNIO_CBF_START_LED_FLASH( )（激活 LED 的闪烁模式）

说明

回调函数 PNIO_CBF_START_LED_FLASH( ) 将标识请求到达通知给 IO-Base 设备用户程序。

随后，IO-Base 设备用户程序可以采取适当的措施来引起操作员注意，例如点亮二极管。该回调函数必须已注册到 PNIO_device_open( ) 函数中。

说明

只要模块需要自我识别，回调函数 PNIO_CBF_START_LED_FLASH( ) 和 PNIO_CBF_STOP_LED_FLASH( ) 就会每 3 秒钟循环交替一次。

当 PNIO_CBF_START_LED_FLASH( ) 到达时，LED 应当以在参数“Frequency”中指定的频率闪烁。

语法

```
typedef void (* PNIO_CBF_START_LED_FLASH)(
    PNIO_UINT32    DevHndl,                  //in
    PNIO_UINT32    Frequency                 //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄
频率	指定的信号发送频率（赫兹）

返回值

无返回值。

7.10.2 PNIO_CBF_STOP_LED_FLASH( )（取消激活 LED 的闪烁模式）

说明

由 PNIO_CBF_START_LED_FLASH( ) 进行的 IO 设备识别通过该回调函数复位。

语法

```
typedef void (* PNIO_CBF_STOP_LED_FLASH)(
    PNIO_UINT32   DevHndl                   //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄

返回值

无返回值。

7.11 常规回调函数 PNIO_CBF_CP_STOP_REQ( )（远程下载和重新组态请求）

说明

只要通过网络收到固件更新或重新组态请求，IO-Base 用户编程接口便会调用该回调函数。

接收该回调函数时，用户程序必须发送 PNIO_device_close( )。
之后，用户程序可以再次调用 PNIO_device_open( )。

但是，不得在属于回调事件 PNIO_CBE_CP_STOP_REQ 的函数内执行函数 PNIO_device_close( ) 和 PNIO_device_open( )。

下载期间，PNIO_device_open( ) 函数会返回错误代码“PNIO_ERR_CONFIG_IN_UPDATE”或“PNIO_ERR_NO_FW_COMMUNICATION”。 下载成功后，PNIO_device_open( ) 函数会返回错误代码“PNIO_OK”。

说明

如果用户程序未注册该回调函数，则只要用户程序处于活动状态，就无法重新组态。

语法

```
typedef void (*PNIO_CBF_CP_STOP_REQ)(
    PNIO_UINT32   DevHndl                  //in
);
```

参数

名称	说明
DevHndl	来自 PNIO_device_open( ) 的 IO 设备句柄

返回值

无返回值。

7.12 等时实时模式 (IRT) 接口

概述

该接口由以下函数和回调事件组成：

- PNIO_CP_register_cbf( )
- 回调事件 PNIO_CP_CBE_STARTOP_IND
- 回调事件 PNIO_CP_CBE_OPFAULT_IND
- PNIO_CP_set_opdone( )

7.12.1 PNIO_CP_register_cbf( )（注册回调）

说明

该函数注册一个回调函数。

说明

回调函数只能由 IO 设备用户程序在 PNIO_device_start( ) 之前注册。

说明

PNIO_CP_CBE_OPFAULT_IND 回调事件类型必须在 PNIO_CP_CBE_STARTOP_IND 回调事件类型之前注册。

语法

```
PNIO_UINT32  PNIO_CP_register_cbf(
    PNIO_UINT32         CpHandle,         //in
    PNIO_CP_CBE_TYPE    CbeType,          //in
    PNIO_CP_CBF         Cbf               //in
);
```

参数

名称	说明
CpHandle	来自 PNIO_device_open( ) 的 IO 设备句柄
CbeType	要注册回调函数“Cbf”的回调事件类型；请参见“PNIO_CP_CBE_TYPE（回调事件类型） (页 259)”部分。
Cbf	要在回调事件“CbeType”到达后启动的回调函数。
	注意 函数指针不允许为空。

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_ALLREADY_DONE
- PNIO_ERR_INTERNAL
- PNIO_ERR_INVALID_CONFIG
- PNIO_ERR_PRM_CALLBACK
- PNIO_ERR_PRM_TYPE
- PNIO_ERR_SEQUENCE
- PNIO_ERR_WRONG_HND

7.12.2 回调事件 PNIO_CP_CBE_STARTOP_IND（开始等时实时数据处理）

说明

PNIO_CP_CBE_STARTOP_IND 回调事件通知 IO-Base 用户程序 IRT 数据传送阶段结束并且 IRT 输出数据（从 IO 控制器的角度来看）使用 DMA 从过程映像传输到主机存储器。 此后，所有 IRT 输出数据（从 IO 控制器的角度来看）都保存在主机存储器中。 这样，用户程序可以访问一致的 IRT 数据。

以下语法显示此事件的具体参数，将其作为 PNIO_CP_CBE_PRM 结构中“联合体”的一部分；请参见“PNIO_CP_CBE_PRM（回调事件参数） (页 260)”部分。

说明

在每个 IRT 数据传送阶段结束时，系统都会调用 PNIO_CP_CBE_STARTOP_IND 回调事件。 即使未与 IO 控制器建立连接，因而 IO 设备中没有 IRT 数据，也是如此。

如果因 PNIO_CP_CBE_STARTOP_IND 回调事件而调用 IO 数据访问函数，则即使没有 IRT 数据，也会出现警告返回值 PNIO_WARN_NO_SUBMODULES，但在这种情况下可以将该值忽略。

语法

```
typedef struct
{
    PNIO_UINT32           CpHandle;           //in
    PNIO_CYCLE_INFO       CycleInfo;          //in
} ATTR_PACKED PNIO_CP_CBE_PRM_STARTOP_IND;
```

参数

名称	说明
CpHandle	PNIO_controller_open( ) 或 PNIO_device_open( ) 的句柄
CycleInfo	有关当前周期的信息

注意
在 PNIO_CP_CBE_STARTOP_IND 事件发生期间，不允许调用任何可能损害用户程序实时功能的函数。 这里的函数指的是需要较长处理时间（如文件操作或屏幕显示）的函数。 请参见操作系统说明，以了解在您所处的情况下可能会涉及哪些函数。 为了访问 IO-Base 用户接口的 IRT 数据，我们通常建议您限制使用这些函数。

7.12.3 回调事件 PNIO_CP_CBE_OPFAULT_IND（违反等时实时模式）

说明

PNIO_CP_CBE_OPFAULT_IND 回调事件将违反等时实时模式通知给 IO-Base 用户程序。 这意味着用户程序在收到 PNIO_CP_CBE_STARTOP_IND 回调事件后调用 PNIO_CP_set_opdone( ) 过晚。

以下语法显示此事件的具体参数，将其作为 PNIO_CP_CBE_PRM 结构中“联合体”的一部分；请参见“PNIO_CP_CBE_PRM（回调事件参数） (页 260)”部分。

语法

```
typedef struct
{
    PNIO_UINT32           CpHandle;              //in
    PNIO_CYCLE_INFO       CycleInfo;             //in
} ATTR_PACKED PNIO_CP_CBE_PRM_OPFAULT_IND;
```

参数

名称	说明
CpHandle	PNIO_controller_open( ) 或 PNIO_device_open( ) 的句柄
CycleInfo	有关当前周期的信息

7.12.4 PNIO_CP_set_opdone( )（等时实时数据处理结束）

说明

通过调用该函数，IO-Base 用户程序将已完成 IRT IO 数据的等时实时处理通知给 IO-Base 接口。 之后，IO-Base 接口启动使用 DMA 将 IRT
输入数据（从控制器的角度来看）从主机存储器传送到过程映像的过程。

语法

```
PNIO_UINT32  PNIO_CP_set_opdone(
    PNIO_UINT32          CpHandle          //in
    PNIO_CYCLE_INFO     *CycleInfo;        //out
);
```

参数

名称	说明
CpHandle	PNIO_controller_open( ) 或 PNIO_device_open( ) 的句柄
CycleInfo	执行该调用的当前周期的相关信息。 如果指针为 0，则不返回任何信息。

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_WRONG_HND

7.13 用于通知新总线周期开始的接口

说明

该接口由以下回调事件组成：

PNIO_CP_CBE_NEWCYCLE_IND

7.13.1 回调事件 PNIO_CP_CBE_NEWCYCLE_IND（新总线周期）

说明

PNIO_CP_CBE_NEWCYCLE_IND 回调事件通知 IO-Base 用户程序新总线周期开始。只有使用 PNIO_CP_register_cbf( ) 注册了该事件后，才会通知该事件。

以下语法显示此事件的具体参数，将其作为 PNIO_CP_CBE_PRM 结构中“联合体”的一部分；请参见“PNIO_CP_CBE_PRM（回调事件参数） (页 260)”部分。

语法

```
typedef struct
{
    PNIO_UINT32          CpHandle;             //in
    PNIO_CYCLE_INFO      CycleInfo;            //in
} ATTR_PACKED PNIO_CP_CBE_PRM_NEWCYCLE_IND;
```

参数

名称	说明
CpHandle	PNIO_controller_open( ) 或 PNIO_device_open( ) 的句柄
CycleInfo	有关当前周期的信息

7.14 I&M 数据记录（标识和维护）

7.14.1 概述

说明

例如，如果 STEP 7 查询 IO 设备或 IO 模块的 I&M 数据，则调用 IO-Base 设备用户程序的读取数据记录回调函数。
传送的数据记录号必须按照“分配数据记录”部分中的表进行解释。

读写数据记录

存在 I&M 数据记录 I&M0 到 I&M4。

这些数据记录按照 PROFINET IO 标准分配有以下数据记录号：

数据记录号	类型	相关说明所在的部分
0xAFF0	PNIO_IM0_TYPE	"PNIO_IM0_TYPE（I&M0 数据记录） (页 263)"
0xAFF1	PNIO_IM1_TYPE	"PNIO_IM1_TYPE（I&M1 数据记录） (页 265)"
0xAFF2	PNIO_IM2_TYPE	"PNIO_IM2_TYPE（I&M2 数据记录） (页 266)"
0xAFF3	PNIO_IM3_TYPE	"PNIO_IM3_TYPE（I&M3 数据记录） (页 267)"
0xAFF4	PNIO_IM4_TYPE	"PNIO_IM4_TYPE（I&M4 数据记录） (页 268)"

IO 设备用户程序将数据写入 PNIO_CBF_REC_READ
回调函数的“pBuffer”，从而传送相应的 I&M 数据记录。

IO 设备用户程序将数据复制到相应的 I&M 记录，从而在 PNIO_CBF_REC_WRITE
回调函数的“pBuffer”中获取 I&M 数据记录。

I&M 数据记录的更多相关信息

更多相关信息，请参见 PROFIBUS 用户组织 e. V. (http://www.profibus.com)
提供的文档：

- 配置文件指南第 1 部分： 标识和维护功能。 版本 1.1.1。2005 年 3 月。订货号：3.502
- I&M 功能规范，版本 1.0.0，2005 年 6 月。
- PROFINET IO 的 GSDML 规范，版本 2.10，2006 年 8 月，订货号：2.352
- IEC 61158

7.14.2 I&M 写入说明

说明

在 IO 控制器用户程序中，可按以下步骤对其中一条 I&M 数据记录执行写入数据记录作业：

步骤	说明
1	将“pBuffer”参数从 PNIO_CBF_REC_READ( ) 回调函数转换为“概述 (页 237)”部分“转换数据记录”表中的一种数据类型。
2	在文件中保持性存储以下元素： • IM_Tag_Function （“PNIO_IM1_TYPE（I&M1 数据记录） (页 265)”部分） • IM_Tag_Location （“PNIO_IM1_TYPE（I&M1 数据记录） (页 265)”部分） • IM_Date （“PNIO_IM2_TYPE（I&M2 数据记录） (页 266)”部分） • IM_Descriptor （“PNIO_IM3_TYPE（I&M3 数据记录） (页 267)”部分） • IM_Signature （“PNIO_IM4_TYPE（I&M4 数据记录） (页 268)”部分）
3	将“IM_Revision_Counter”元素递增。 溢出后，“IM_Revision_Counter”不从 0x0000 开始计数，而是从 0x0001 开始。

说明

始终拒绝数据记录“I&M0”的写入数据记录作业。
“I&M0”数据记录不包含任何可写入的元素。

错误代码

下表显示的是始终由 PNIO_CBF_REC_READ( ) 回调函数返回的错误代码元素。

如果不发生错误，则值为“0”，如果发生错误，则结果如下表所述：

名称	发生错误时的值
ErrorCode	= 0xDF (IODWriteRes)
ErrorDecode	= 0x80 (PNIORW)
ErrorCode1	按照 PROFINET IO 标准中的表“使用 ErrorDecode PNIORW 进行 ErrorCode1 编码”填充。
ErrorCode2	按照 PROFINET IO 标准以用户特定的值填充。

7.14.3 I&M 读取说明

说明

读取数据记录作业的处理步骤如下：

步骤	说明
1	将“pBuffer”参数从 PNIO_CBF_REC_READ( ) 回调函数转换为“概述 (页 237)”部分“转换数据记录”表中的一种数据类型。
2	按照“PNIO_IM0_TYPE（I&M0 数据记录） (页 263)”到“PNIO_IM1_TYPE（I&M1 数据记录） (页 265)”部分的说明在数据缓冲区中设置 I&M 参数。 从保持性存储器中读取以下元素： • IM_Tag_Function • IM_Tag_Location • IM_Date • IM_Descriptor • IM_Signature 默认情况下，这些元素的值为零，且在工程师站上设置。

说明

I&M 数据记录中数据长度为两个或四个字节的所有元素都具有大端格式。

说明

每个支持可写入 I&M 数据记录的子模块的 GSDML 文件中必须包括“Writeable_IM_Records”条目。

说明

用于读取数据记录号为 0xF840 的信息数据记录“I&M0FilterData”的工程师站作业由 PROFINET IO 函数在内部进行处理，并且不传递到 PROFINET IO 设备用户程序。

错误代码元素

下表描述了错误代码元素及其在 PNIO_CBF_REC_WRITE( ) 回调函数（另请参见“回调函数 PNIO_CBF_REC_WRITE( )（处理写入数据记录作业） (页 197)”部分）的“pPnioState”参数出错和不出错时的可能值：

名称	不发生错误时的值	发生错误时的值
ErrorCode	= 0	= 0xDF (IODWriteRes)
ErrorDecode	= 0	= 0x80 (PNIORW)
ErrorCode1	= 0	按照 PROFINET IO 标准中的表“使用 ErrorDecode PNIORW 进行 ErrorCode1 编码”填充。
ErrorCode2	= 0	按照 PROFINET IO 标准以用户特定的值填充。

7.15 数据类型

说明

以下数据类型用于 IO-Base 用户编程接口：

- PNIO_ANNOTATION（版本 ID 结构） (页 242)
- PNIO_APPL_READY_LIST_TYPE（应用程序就绪） (页 244)
- PNIO_AR_REASON（连接中止的原因） (页 246)
- PNIO_AR_TYPE（应用关系） (页 249)
- PNIO_CFB_FUNCTIONS（回调函数注册结构） (页 250)
- PNIO_DEV_ADDR（IO 设备地址类型） (页 252)

- PNIO_APDU_STATUS_IND（IO 控制器状态） (页 253)
- PNIO_IOCR_TYPE（应用关系） (页 253)
- PNIO_IOCS_TYPE（应用关系） (页 255)
- PNIO_MODULE_TYPE（应用关系） (页 256)
- PNIO_SUBMOD_TYPE（应用关系） (页 257)
- PNIO_ACCESS_ENUM（访问类型） (页 258)
- PNIO_CP_CBE_TYPE（回调事件类型） (页 259)
- PNIO_CP_CBE_PRM（回调事件参数） (页 260)
- PNIO_CP_CBF（常规 PNIO 回调函数） (页 260)

7.15.1 PNIO_ANNOTATION（版本 ID 结构）

说明

版本 ID 必须通过 PNIO_device_open( ) 函数调用进行传送。 这包含 IO 设备类型、订货号、硬件及软件版本。

PNIO_ANNOTATION 结构的参数在“HW Config”工具的在线诊断中显示，并且可以让用户（维护人员）识别模块类型和设备版本（或者是在设备上运行的控制应用程序的版本）。

订购更换设备时，订货号（“orderId”）很有用。

用户应用程序应当使用有用的值来初始化该参数，因为该信息对于工厂维护和设备标识非常重要。

语法

```
#define MAX_DEVICE_TYPE_LENGTH            25
#define MAX_ORDER_ID_LENGTH               20

typedef struct
{
    PNIO_UINT8        deviceType[MAX_DEVICE_TYPE_LENGTH+1];
    PNIO_UINT8           orderId[MAX_ORDER_ID_LENGTH+1];
    PNIO_UINT16          hwRevision;
    PNIO_UINT8           swRevisionPrefix;
    PNIO_UINT16          swRevision1;
    PNIO_UINT16          swRevision2;
    PNIO_UINT16          swRevision3;
```

```
} ATTR_PACKED PNIO_ANNOTATION;
```

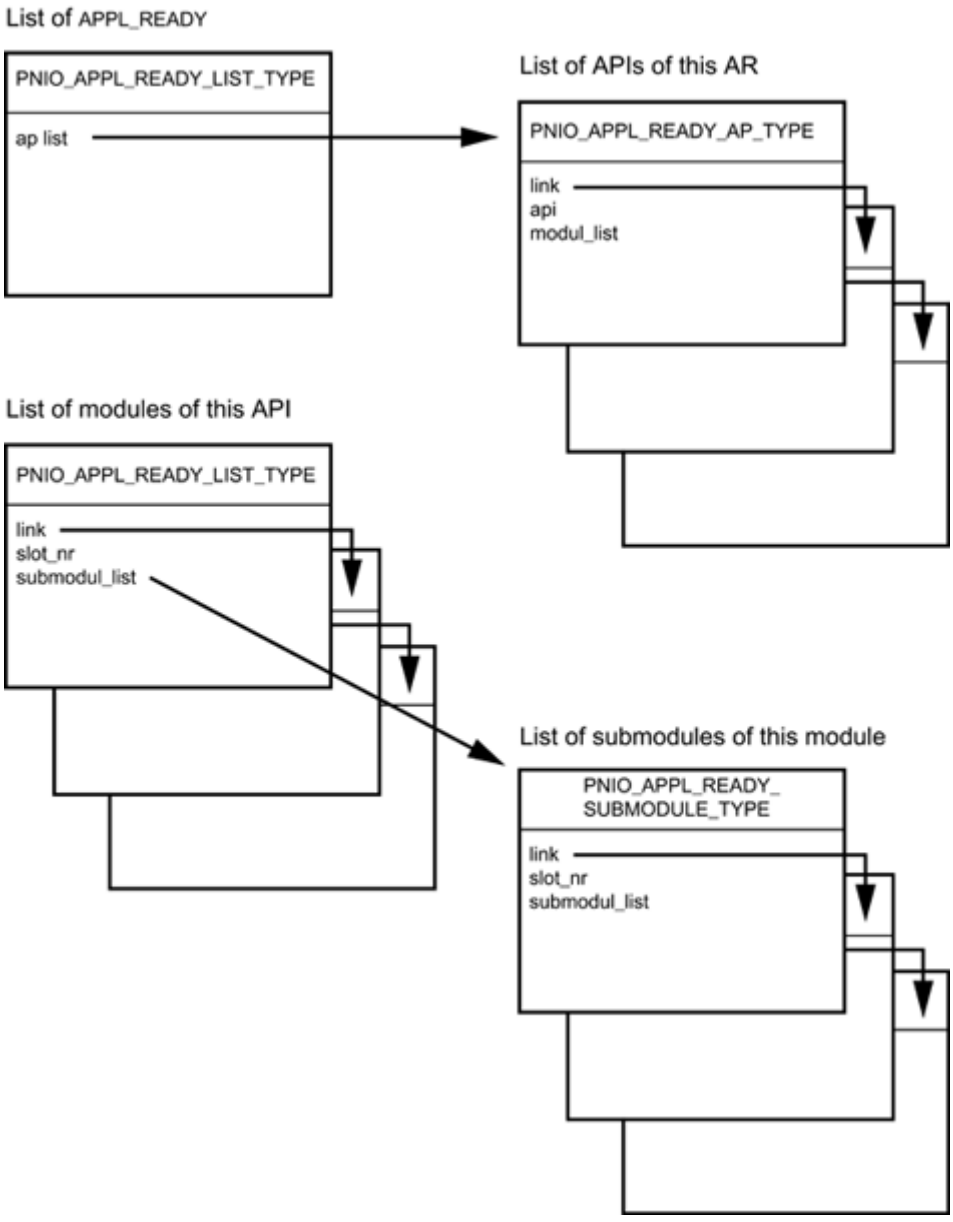
元素

名称	说明
deviceType	IO 设备标识符 - 对应于 GSDML 文件中的“ProductFamily”。
orderId	IO 设备订货号 - 对应于 GSDML 文件中的“OrderNumber Value”。
hwRevision	硬件版本号 -“hwRevision”对应于 GSDML 文件中的“HardwareRelease Value”。
SwRevisionPrefix	软件版本前缀 -“swRevisionPrefix”对应于 GSDML 文件中“SoftwareRelease Value”的前缀字符。 **示例** 从 GSDML 文件中的行 `<SoftwareRelease Value="Z1.2.3" />` 可知，“swRevisionPrefix”是 Z。
SwRevision1	软件版本的第一个数字值 -“swRevision1”对应于 GSDML 文件中“SoftwareRelease Value”的第一个数字值。 **示例** 从 GSDML 文件中的行 `<SoftwareRelease Value="Z1.2.3" />` 可知，“swRevision1”是 1。
SwRevision2	软件版本的第二个数字值 -“swRevision2”对应于 GSDML 文件中“SoftwareRelease Value”的第二个数字值。 **示例** 从 GSDML 文件中的行 `<SoftwareRelease Value="Z1.2.3" />` 可知，“swRevision2”是 2。
SwRevision3	软件版本的第三个数字值 -“swRevision3”对应于 GSDML 文件中“SoftwareRelease Value”的第三个数字值。 **示例** 从 GSDML 文件中的行 `<SoftwareRelease Value="Z1.2.3" />` 可知，“swRevision3”是 3。

7.15.2 PNIO_APPL_READY_LIST_TYPE（应用程序就绪）

说明

子模块的层级结构列表（AP/模块/子模块，不重复）。 在该列表中，IO-Base 设备用户程序必须输入未达到应用程序就绪状态的所有模块/子模块。
下图给出了该列表的层级结构。



语法

```
typedef struct pnio_appl_ready_tag
{
   PNIO_LIST_ENTRY_TYPE    ap_list;
} PNIO_APPL_READY_LIST_TYPE;

typedef struct pnio_appl_ready_ap_tag
{
   PNIO_LIST_ENTRY_TYPE    link;
   PNIO_UINT32             api;

/* with list from type PNIO_APPL_READY_MODULE_TYPE  */
   PNIO_LIST_ENTRY_TYPE    module_list;
} PNIO_APPL_READY_AP_TYPE;

typedef struct pnio_appl_ready_module_tag
{
   PNIO_LIST_ENTRY_TYPE    link;
   PNIO_UINT16             slot_nr;

/*with list from type PNIO_APPL_READY_SUBMODULE_TYPE*/
   PNIO_LIST_ENTRY_TYPE    submodule_list;
} PNIO_APPL_READY_MODULE_TYPE;

typedef struct pnio_appl_ready_submodule_tag
{
   PNIO_LIST_ENTRY_TYPE    link;
   PNIO_UINT16             subslot_nr;
} PNIO_APPL_READY_SUBMODULE_TYPE;

/* device control */
typedef struct pnio_list_entry_tag
{
   struct pnio_list_entry_tag   *Flink; /* forward link */
   struct pnio_list_entry_tag   *Blink; /* backward link */
} PNIO_LIST_ENTRY_TYPE;
```

元素

名称	说明
ap_list	该应用关系所支持的 API 列表
Api	该列表条目所支持的 API
module_list	该 API 的所有模块列表
slot_nr	该列表条目的模块编号
submodul_list	该模块所有子模块的列表

名称	说明
submodul_nr	该列表条目的子模块编号
link	每个列表条目都有一个引用下一个列表元素的链接元素。 下一个列表元素的引用为空时，表明到达列表结尾。

7.15.3 PNIO_AR_REASON（连接中止的原因）

说明

使用 PNIO_CBF_AR_ABORT_IND( ) 回调函数传送到 IO-Base 设备用户程序的可能的连接中止条件列表。

语法

```
typedef enum
{
    PNIO_AR_REASON_NONE,
    PNIO_AR_REASON_RESERVED1,
    PNIO_AR_REASON_RESERVED2,
    PNIO_AR_REASON_MEM,
    PNIO_AR_REASON_FRAME,
    PNIO_AR_REASON_MISS,
    PNIO_AR_REASON_TIMER,
    PNIO_AR_REASON_ALARM,
    PNIO_AR_REASON_ALSND,
    PNIO_AR_REASON_ALACK,
    PNIO_AR_REASON_ALLEN,
    PNIO_AR_REASON_ASRT,
    PNIO_AR_REASON_RPC,
    PNIO_AR_REASON_ABORT,
    PNIO_AR_REASON_RERUN,
    PNIO_AR_REASON_REL,
    PNIO_AR_REASON_PAS,
    PNIO_AR_REASON_RMV,
    PNIO_AR_REASON_PROT,
    PNIO_AR_REASON_NARE,
    PNIO_AR_REASON_BIND,
    PNIO_AR_REASON_CONNECT,
    PNIO_AR_REASON_READ,
    PNIO_AR_REASON_WRITE,
    PNIO_AR_REASON_CONTROL,
    PNIO_AR_REASON_PULLPLUG,
    PNIO_AR_REASON_AP_RMV,
    PNIO_AR_REASON_LNK_DWN,
    PNIO_AR_REASON_MMAC,
    PNIO_AR_REASON_SYNC,
    PNIO_AR_REASON_TOPO,
    PNIO_AR_REASON_DCP_NAME,
    PNIO_AR_REASON_DCP_RESET,
    PNIO_AR_REASON_PRM,
    PNIO_AR_REASON_IRDATA,
    PNIO_AR_REASON_MAX,
}PNIO_AR_REASON;
```

元素

名称	PROFINET IO 标准中的说明
PNIO_AR_REASON_NONE	无
PNIO_AR_REASON_RESERVED1	内部使用

名称	PROFINET IO 标准中的说明
PNIO_AR_REASON_RESERVED2	内部使用
PNIO_AR_REASON_MEM	存储器空间不足
PNIO_AR_REASON_FRAME	添加提供者或消费者失败
PNIO_AR_REASON_MISS	消费者缺失
PNIO_AR_REASON_TIMER	cmi 超时
PNIO_AR_REASON_ALARM	打开报警失败
PNIO_AR_REASON_ALSND	发送报警
PNIO_AR_REASON_ALACK	发送报警确认
PNIO_AR_REASON_ALLEN	报警数据过长
PNIO_AR_REASON_ASRT	报警指示错误
PNIO_AR_REASON_RPC	rpc 客户端调用
PNIO_AR_REASON_ABORT	ar 中止
PNIO_AR_REASON_RERUN	重新运行现有中止
PNIO_AR_REASON_REL	获取版本
PNIO_AR_REASON_PAS	设备已钝化
PNIO_AR_REASON_RMV	设备/ar 已移除
PNIO_AR_REASON_PROT	违反协议
PNIO_AR_REASON_NARE	NARE 错误
PNIO_AR_REASON_BIND	rpc 绑定错误
PNIO_AR_REASON_CONNECT	rpc 连接错误
PNIO_AR_REASON_READ	rpc 读取错误
PNIO_AR_REASON_WRITE	rpc 写入错误
PNIO_AR_REASON_CONTROL	rpc 控制错误
PNIO_AR_REASON_PULLPLUG	在 check.rsp 之后、in-data.ind 之前禁止插拔
PNIO_AR_REASON_AP_RMV	已弃用！ AP 已移除
PNIO_AR_REASON_LNK_DWN	链接“停止”
PNIO_AR_REASON_MMAC	无法注册多播 mac
PNIO_AR_REASON_SYNC	未同步 （不能启动伙伴 ar）

名称	PROFINET IO 标准中的说明
PNIO_AR_REASON_TOPO	拓扑错误 （不能启动伙伴 ar）
PNIO_AR_REASON_DCP_NAME	dcp，站名称已更改
PNIO_AR_REASON_DCP_RESET	dcp，复位为出厂设置
PNIO_AR_REASON_PRM	已弃用！不能启动伙伴 AR，因为 0x8ipp 子模块在第一个 AR 中 • 应用程序就绪处于未决状态（参数化错误） • 已锁定 （无参数化） • 错误或已拔出 （无参数化）
PNIO_AR_REASON_IRDATA	还没有 irdata 记录
PNIO_AR_REASON_MAX	最大值

7.15.4 PNIO_AR_TYPE（应用关系）

说明

该结构通过 PNIO_CBF_AR_INFO_IND( ) 回调函数进行发送时，可通知用户程序 IO 控制器要控制哪些模块和子模块。

该结构通过 PNIO_CBF_AR_CHECK_IND( ) 回调函数发送时，可将应用关系属性通知给用户程序（指针“pModList”始终为空，且“NumOfMod=0”！这意味着未传送任何模块列表）。

语法

```
typedef struct
{
    void                *pReserved;
    PNIO_UINT16          ArNumber;
    PNIO_UINT16          SessionKey;
    PNIO_UINT32          NumOfIocr;
    PNIO_IOCR_TYPE      *pIoCrList;
    PNIO_UINT32          NumOfMod;
    PNIO_MODULE_TYPE    *pModList;
} ATTR_PACKED PNIO_AR_TYPE;
```

元素

名称	说明
ArNumber	应用关系编号
SessionKey	当前应用关系的会话密钥
NumOflocr	“pIoCrList”中的 IO-CR 条目数
pIoCrList	指向 IO-CR 列表的指针
NumOfMod	“pModList”中模块数
pModList	指向模块列表的指针
pReserved	为将来的扩展保留

7.15.5 PNIO_CFB_FUNCTIONS（回调函数注册结构）

说明

该结构与 PNIO_device_open( ) 一起用于注册回调函数。

语法

```
typedef struct
{
    PNIO_UINT32    size          ;/* Size of this structure */
    PNIO_CBF_DATA_WRITE           cbf_data_write;
    PNIO_CBF_DATA_READ            cbf_data_read;
    PNIO_CBF_REC_READ             cbf_rec_read;
    PNIO_CBF_REC_WRITE            cbf_rec_write;
    PNIO_CBF_REQ_DONE             cbf_alarm_done;
    PNIO_CBF_CHECK_IND            cbf_check_ind;
    PNIO_CBF_AR_CHECK_IND         cbf_ar_check_ind;
    PNIO_CBF_AR_INFO_IND          cbf_ar_info_ind;
    PNIO_CBF_AR_INDATA_IND        cbf_ar_indata_ind;
    PNIO_CBF_AR_ABORT_IND         cbf_ar_abort_ind;
    PNIO_CBF_AR_OFFLINE_IND       cbf_ar_offline_ind;
    PNIO_CBF_APDU_STATUS_IND      cbf_apdu_status_ind;
    PNIO_CBF_PRM_END_IND          cbf_prm_end_ind;
    PNIO_CBF_CP_STOP_REQ          pnio_cbf_cp_stop_req;
    PNIO_CBF_DEVICE_STOPPED       pnio_cbf_device_stopped;
    PNIO_CBF_START_LED_FLASH      cbf_start_led_flash;
    PNIO_CBF_STOP_LED_FLASH       cbf_stop_led_flash;
    PNIO_CBF_PULL_PLUG_CONF       cbf_pull_plug_conf;
} ATTR_PACKED PNIO_CFB_FUNCTIONS;
```

元素

名称	说明
size	结构大小 - 始终“=sizeof(PNIO_CFB_FUNCTIONS)”。 用于确定该结构的版本。
其余字段	指向相应回调函数的指针 所有回调函数必须在 PNIO_CFB_FUNCTIONS 结构中指定，但以下函数除外： • PNIO_CBF_START_LED_FLASH( ) • PNIO_CBF_STOP_LED_FLASH( ) • PNIO_CBF_PULL_PLUG_CONF( ) 如果想要支持在生产运行期间拆除和插入模块/子模块，则必须注册 PNIO_CBF_PULL_PLUG_CNF( )。

7.15.6 PNIO_DEV_ADDR（IO 设备地址类型）

说明

PNIO_DEV_ADDR 地址结构用于使用此处描述的所有 IO-Base 接口函数进行寻址。

语法

```
typedef struct
{
    PNIO_ADDR_TYPE            AddrType;
    PNIO_IO_TYPE              IODataType;
    union
    {
        PNIO_UINT32     RESERVED;
        struct
        {
            PNIO_UINT32 RESERVED1[2];
            PNIO_UINT32 Slot;
            PNIO_UINT32 Subslot;
            PNIO_UINT32 RESERVED2[1];
        } Geo;
    }u;
} ATTR_PACKED PNIO_DEV_ADDR;
```

元素

名称	说明
AddrType	对于 IO 设备，必须始终为 PNIO_ADDR_GEO！
IODataType	指定数据类型为输入类型还是输出类型。 可能的值： • PNIO_IO_IN，表示输入数据 • PNIO_IO_OUT，表示输出数据
addr.Geo.Slot	插槽号（地理地址）
addr.Geo.Subslot	子插槽号（地理地址）
Reserved	为将来的扩展保留
Geo. RESERVED1 [2]	为将来的扩展保留
Geo. RESERVED2	为将来的扩展保留

7.15.7 PNIO_APDU_STATUS_IND（IO 控制器状态）

说明

APDU 中包含的 IO 控制器状态的说明。

语法

```
typedef enum
{
    PNIO_EVENT_APDU_STATUS_STOP,
    PNIO_EVENT_APDU_STATUS_RUN,
    PNIO_EVENT_APDU_STATUS_STATION_OK,
    PNIO_EVENT_APDU_STATUS_STATION_PROBLEM,
    PNIO_EVENT_APDU_STATUS_PRIMARY,
    PNIO_EVENT_APDU_STATUS_BACKUP
} PNIO_APDU_STATUS_IND;
```

元素

名称	说明
PNIO_EVENT_APDU_STATUS_STOP	STOP 模式下的 IO 控制器
PNIO_EVENT_APDU_STATUS_RUN	RUN 模式下的 IO 控制器
PNIO_EVENT_APDU_STATUS_STATION_OK	工作站正常
PNIO_EVENT_APDU_STATUS_STATION_PROBLEM	工作站不正常
PNIO_EVENT_APDU_STATUS_PRIMARY	主要状态
PNIO_EVENT_APDU_STATUS_BACKUP	备用状态

7.15.8 PNIO_IOCR_TYPE（应用关系）

说明

该结构由 PNIO_AR_TYPE 使用，并且提供输入/输出数据类型的相关信息

语法

```
typedef struct
{
    PNIO_UINT32          CycleTime;
    PNIO_UINT32          Direction;
    PNIO_UINT32          IOCRProperties;
    PNIO_UINT32          SendClock;
    PNIO_UINT32          ReductioFactor;
    PNIO_UINT32          Phase;
    PNIO_UINT32          NumOfIoCs;
    PNIO_IOCS_TYPE      *pIoCsList;
    PNIO_UINT32          reserved[3];
} ATTR_PACKED PNIO_IOCR_TYPE;
```

元素

名称	说明
CycleTime	IO 连接关系的周期时间（毫秒）- 根据标准，该时间的计算公式为： SendClock x 31.25 µs x ReductioFactor
Direction	请参见 PNIO_IOCR_TYPE_ENUM
IOCRProperties	请参见 PNIO_IOCR_PROP_ENUM
SendClock	发送周期持续时间除以时间基准 (31.25 µs)。
ReductioFactor	缩减系数
Phase	为该 IOCR 传送子模块数据所处的阶段。
NumOfIoCs	“pIOCsList”中的条目数
pIoCsList	属于该 IOCR 的子模块列表。
Reserved	为将来的扩展保留。

PNIO_IOCR_TYPE_ENUM 元素

名称	说明
PNIO_IOCR_TYPE_RESERVED_0	保留
PNIO_IOCR_TYPE_INPUT	输入帧（从 IO 控制器的角度来看）
PNIO_IOCR_TYPE_OUTPUT	输出帧（从 IO 控制器的角度来看）

名称	说明
PNIO_IOCR_TYPE_MULTICAST_PROVIDER	该子模块的输出数据作为多播提供者连接关系进行分布（输入帧）。
PNIO_IOCR_TYPE_MULTICAST_CONSUMER	该子模块的输入数据作为多播消费者连接关系进行接收（输出帧）。

PNIO_IOCR_PROP_ENUM 元素

名称	说明
PNIO_IOCR_PROP_RT_CLASS_MASK	掩码元素
PNIO_IOCR_PROP_RT_CLASS_1	实时帧
PNIO_IOCR_PROP_RT_CLASS_2	实时帧
PNIO_IOCR_PROP_RT_CLASS_3	等时实时帧
PNIO_IOCR_PROP_RT_CLASS_1_UDP	UDP 上的 RT

注意
在用户程序中，确保调用 PNIO_device_ar_abort( ) 时，未知代码会导致连接中止。

7.15.9 PNIO_IOCS_TYPE（应用关系）

说明

该结构由 PNIO_IOCR_TYPE 使用，并且提供 IOCR 包含哪些子模块的相关信息。

语法

```
typedef struct
{
    PNIO_UINT32          SlotNum;
    PNIO_UINT32          SubslotNum;
    PNIO_UINT32          reserved;
} ATTR_PACKED PNIO_IOCS_TYPE;
```

元素

名称	说明
SlotNum	模块编号
SubslotNum	子模块编号
Reserved	为将来的扩展保留

7.15.10 PNIO_MODULE_TYPE（应用关系）

说明

该结构由 PNIO_AR_TYPE 使用。 描述模块组态。

语法

```
typedef struct
{
    PNIO_UINT32          API;
    PNIO_UINT32          SlotNum;
    PNIO_UINT32          NumOfSubmod;
    PNIO_UINT32          ModProperties;
    PNIO_UINT32          ModIdent;
    PNIO_SUBMOD_TYPE    *pSubList;
} ATTR_PACKED PNIO_MODULE_TYPE;
```

元素

名称	说明
API	指定模块的应用程序进程标识符
SlotNum	模块的插槽号
NumOfSubmod	“pSubList”中的子模块数
ModProperties	预留 - 必须始终为 0！
ModIdent	指定模块的模块标识 - 对应于 GSDML 文件中的“ModulIdentNumber”。
pSubList	指向子模块列表的指针

7.15.11 PNIO_SUBMOD_TYPE（应用关系）

说明

该结构由 PNIO_MODULE_TYPE 使用。 描述子模块组态。

语法

```
typedef struct
{
    PNIO_UINT32         SlotNum;
    PNIO_UINT32         SubSlotNum;
    PNIO_UINT32         SubMProperties;
    PNIO_UINT32         SubMIdent;
    PNIO_UINT32         InDatLen;
    PNIO_UINT32         InIopsLen;
    PNIO_UINT32         InIocsLen;
    PNIO_UINT32         OutDatLen;
    PNIO_UINT32         OutIopsLen;
    PNIO_UINT32         OutIocsLen;
    PNIO_UINT32         reserved[8];
} ATTR_PACKED PNIO_SUBMOD_TYPE;
```

元素

名称	说明
SlotNum	子模块所属的模块编号。
SubSlotNum	子模块的子插槽号
SubMProperties	子模块属性 - 请参见 PNIO_SUB_PROPERTIES_ENUM。
SubMIdent	GSDML 文件中的子模块标识
InDatLen	该子模块的 IO 控制器输入数据的长度
InIopsLen	输入数据本地状态的长度
InIocsLen	输入数据远程状态的长度
OutDatLen	该子模块的 IO 控制器输出数据的长度
OutIopsLen	输出数据（从 IO 控制器的角度来看）本地状态的长度
OutIocsLen	输出数据（从 IO 控制器的角度来看）远程状态的长度
Reserved	为以后的扩展保留

PNIO_SUB_PROPERTIES_ENUM 元素

名称	说明
PNIO_SUB_PROP_TYPE_MASK	用于选择以下 IO 数据类型的掩码。
	注意 该掩码需要与“SubMProperties”字段进行 AND 运算，才能与以下常量进行比较。 此操作很有必要，因为按照“SubMProperties”中的标准，其余位会标记为预留。
PNIO_SUB_PROP_TYPE_NO_DATA	不含数据的模块
PNIO_SUB_PROP_TYPE_INPUT	具有输入数据的模块
PNIO_SUB_PROP_TYPE_OUTPUT	具有输出数据的模块
PNIO_SUB_PROP_TYPE_INPUT_OUTPUT	具有输入和输出数据的模块

注意
在用户程序中，确保调用 PNIO_device_ar_abort( ) 回调函数时，未知常量会导致连接中止。

7.15.12 PNIO_ACCESS_ENUM（访问类型）

说明

该枚举类型列出可能的访问类型。

这些值以 PNIO_initiate_data_read_ext( ) 和 PNIO_initiate_data_write_ext( ) 函数的参数形式提供。 它们限制了调用 PNIO_DATA_READ_CBF( ) 和 PNIO_DATA_WRITE_CBF( ) 回调函数时可用的子模块和访问类型。

语法

```
typedef enum
{
   PNIO_ACCESS_ALL_WITHOUT_LOCK,
   PNIO_ACCESS_RT_WITH_LOCK,
   PNIO_ACCESS_RT_WITHOUT_LOCK,
   PNIO_ACCESS_IRT_WITHOUT_LOCK
} PNIO_ACCESS_ENUM;
```

元素

名称	说明
PNIO_ACCESS_ALL_WITHOUT_LOCK	RT 和 IRT 数据都将被访问。 不能保证 RT 数据的一致性。 只有在等时实时访问时，才能保证 IRT 数据的一致性。
PNIO_ACCESS_RT_WITH_LOCK	仅访问 RT 数据。 可以保证数据的一致性。
PNIO_ACCESS_RT_WITHOUT_LOCK	仅访问 RT 数据。 不能保证数据的一致性。
PNIO_ACCESS_IRT_WITHOUT_LOCK	仅访问 IRT 数据。 只有在等时实时访问时，才能保证 IRT 数据的一致性。

7.15.13 PNIO_CP_CBE_TYPE（回调事件类型）

说明

IO-Base 用户编程接口可识别以下回调事件类型：

回调事件类型	回调事件
PNIO_CP_CBE_STARTOP_IND	等时实时数据处理开始
PNIO_CP_CBE_OPFAULT_IND	违背等时实时模式
PNIO_CP_CBE_NEWCYCLE_IND	新总线周期开始

这些回调事件只能由 IO 控制器或 IO 设备注册。

如果要同时实现 IO 控制器和 IO 设备，则需要在一个 IO-Base 用户程序中同时实现两个设备。

7.15.14 PNIO_CP_CBE_PRM（回调事件参数）

说明

各种回调事件具有相同的数据类型 PNIO_CP_CBE_PRM，该数据类型使用“联合体”对各回调事件的各种参数进行分组。

语法

```
typedef struct
{
    PNIO_CP_CBE_TYPE    CbeType;            //in
    PNIO_UINT32         Handle;             //in
    union
    {
        ...        /*Union over the various
        ...          callback event parameters
                     (details in relevant sections)*/
    };
} ATTR_PACKED PNIO_CP_CBE_PRM;
```

参数

名称	说明
CbeType	回调事件类型
Handle	来自 PNIO_controller_open( ) 或 PNIO_device_open( ) 的句柄

7.15.15 PNIO_CP_CBF（常规 PNIO 回调函数）

说明

使用回调事件参数 PNIO_CP_CBE_PRM 来声明类型 PNIO_CP_CBF 的常规 PNIO 回调函数。

语法

```
typedef void (* PNIO_CP_CBF)(
    PNIO_CP_CBE_PRM       *pCbfPrm
);
```

7.15.16 PNIO_CYCLE_INFO（有关当前周期的信息）

说明

该结构包含有关当前周期的信息，并允许标识以下内容：

- 分辨率为 10 ns 的高精度计数器（“ClockCount”）
- 中断丢失（“CycleCount”值在每个周期的变化不同）
- “CountSinceCycleStart”返回周期开始与调用函数（返回 PNIO_CYCLE_INFO 结构）之间的时间。
- PNIO_CP_CBE_STARTOP_IND 回调事件与 PNIO_CP_set_opdone( ) 函数调用之间的时间（PNIO_CP_CBE_STARTOP_IND 回调事件的输入值“CountSinceCycleStart”与 PNIO_CP_set_opdone( ) 函数返回值之间的差值）

它用作回调事件的输入参数：

- PNIO_CP_CBE_STARTOP_IND
- PNIO_CP_CBE_OPFAULT_IND
- PNIO_CP_CBE_NEWCYCLE_IND

该结构用作函数 PNIO_CP_set_opdone( ) 的返回值。

语法

```
typedef struct
{
    PNIO_UINT32     ClockCount;
    PNIO_UINT32     CycleCount;
    PNIO_UINT32     CountSinceCycleStart;
} ATTR_PACKED PNIO_CYCLE_INFO;
```

参数

名称	说明
ClockCount	CP 上自由运行的硬件计数器，分辨率为 10 ns（32 位宽）。
CycleCount	“CycleCount”值在每个周期以对应于周期持续时间的值（以 31.25 µs 为基数）为增量递增。 该值为 16 位宽。 **示例** 周期为 1 ms 时，值每次增加 32。
CountSinceCycleStart	“CountSinceCycleStart”值指定从上一个周期开始以来的时间（以 10 ns 为基数）。 该值为 32 位宽。

7.15.17 PNIO_BLOCK_HEADER

说明

PNIO_BLOCK_HEADER 结构是如下描述的 I&M 数据记录结构的一部分：

- PNIO_IM0_Type
- PNIO_IM1_TYPE
- PNIO_IM2_TYPE
- PNIO_IM3_TYPE
- PNIO_IM4_TYPE

语法

```
typedef struct
{
    PNIO_UINT16     BlockType;
    PNIO_UINT16     BlockLength;
    PNIO_UINT8      BlockVersionHigh;
    PNIO_UINT8      BlockVersionLow;
} ATTR_PACKED PNIO_BLOCK_HEADER;
```

元素

PNIO_BLOCK_HEADER 元素的值取决于 I&M 数据记录的结构。 可以在如下 I&M 数据记录描述（PNIO_IM0 到 PNIO_IM4）中找到这些值。

7.15.18 PNIO_IM0_TYPE（I&M0 数据记录）

说明

PNIO_IM0_TYPE 结构用于通知 IO 控制器有关模块或设备的常规信息。

语法

```
typedef struct
{
    PNIO_BLOCK_HEADER   BlockHeader;
    PNIO_UINT8          VendorIDHigh;
    PNIO_UINT8          VendorIDLow;
    PNIO_UINT8          OrderID[20];
    PNIO_UINT8          IM_Serial_Number[16];
    PNIO_UINT16         IM_Hardware_Revision (Big Endian);
    PNIO_UINT8          IM_Software_Revision[4];
    PNIO_UINT16         IM_Revision_Counter (Big Endian);
    PNIO_UINT16         IM_Profile_ID (Big Endian);
    PNIO_UINT16         IM_Profile_Specific_Type (Big Endian);
    PNIO_UINT8          IM_Version_Major;
    PNIO_UINT8          IM_Version_Minor;
    PNIO_UINT16         IM_Supported (Big Endian);
} ATTR_PACKED PNIO_IM0_TYPE;
```

“BlockHeader”的参数

名称	值
BlockType	0x0020
BlockLength	0x0038
BlockVersionHigh	0x01
BlockVersionLow	0x00

PNIO_BLOCK_HEADER 结构在“PNIO_BLOCK_HEADER (页 262)”部分介绍。

其余元素

名称	说明	值
VendorIDHigh	供应商 ID 的第一个字节	**示例** 0x00 (Siemens AG)
VendorIDLow	供应商 ID 的第二个字节	**示例** 0x2A (Siemens AG)
OrderId	空白序列号（将由供应商分配）。	**示例** 6GK1 161-6AA00
IM_Serial_Number	空白序列号（将由供应商分配）。	**示例** 000-000-000-0001
IM_Hardware_Revision	设备硬件修订版本（将由供应商分配）。	**示例** 0x01
IM_Software_Revision	软件修订版本（将由供应商分配） - 其中包括： • SwRevisionPrefix • IM_SWRevision_ Functional_Enhancement • IM_SWRevision_Bug_Fix • IM_SWRevision_Internal_Change	**示例** 'V', 1, 0, 0
IM_Revision_Counter	修订计数器 - 每次硬件变化或重新分配 IO 设备参数后递增此值。	• 1: 出厂设置 • 2 到 0xFFFF： 重新分配参数后
IM_Profile_ID	配置文件 ID - 与在: (http://www.profibus.com/IM/Profile_ID_Table.xml)中的定义对应	**示例** 常见 PROFINET IO 设备的示例： 0xF600
IM_Profile_Specific_Type	配置文件特定的设备类型 - 根据: (http://www.profibus.com/IM/Profile_specific_type_table_6102.xml)	**示例** 对于通信模块： 0x0004
IM_Version_Major	I&M 主要版本	0x01

名称	说明	值
IM_Version_Minor	I&M 次要版本	0x01
IM_Supported	位 0 始终为值 0。 始终支持 IM0。 位 1 到 4 指定支持哪条 I&M 数据记录 位 1 = 1 - 支持 IM1。 位 2 = 1 - 支持 IM2。 位 3 = 1 - 支持 IM3。 位 4 = 1 - 支持 IM4。	**示例 1** 0x0002 表示： 支持 IM0 和 IM1。 **示例 2** 0x0006 表示： 支持 IM0、IM1 和 IM2。

7.15.19 PNIO_IM1_TYPE（I&M1 数据记录）

说明

PNIO_IM1_TYPE 结构用于通知 IO 控制器有关模块或设备的功能和位置。

语法

```
typedef struct
{
    PNIO_BLOCK_HEADER   BlockHeader;    /* BlockType: 0x0021 */
    PNIO_UINT8          IM_Tag_Function[32];
    PNIO_UINT8          IM_Tag_Location[22];
} ATTR_PACKED PNIO_IM1_TYPE;
```

“BlockHeader”的参数

名称	值
BlockType	0x0021
BlockLength	0x0038
BlockVersionHigh	0x01
BlockVersionLow	0x00

PNIO_BLOCK_HEADER 结构在“PNIO_BLOCK_HEADER (页 262)”部分介绍。

其余元素

名称	说明	值
IM_Tag_Function	模块或设备功能或用途介绍。	**示例** 1) “压力传感器 7”
IM_Tag_Location	位置说明。	**示例** 1) “生产车间 3”

1) 字符串将被填补到最大长度。

7.15.20 PNIO_IM2_TYPE（I&M2 数据记录）

说明

PNIO_IM2_TYPE 结构用于通知 IO 控制器有关模块或设备的安装日期。

语法

```
typedef struct
{
    PNIO_BLOCK_HEADER   BlockHeader;
    PNIO_UINT8          IM_Date[16];
} ATTR_PACKED PNIO_IM2_TYPE;
```

“BlockHeader”的参数

名称	值
BlockType	0x0022
BlockLength	0x0038
BlockVersionHigh	0x01
BlockVersionLow	0x00

PNIO_BLOCK_HEADER 结构在“PNIO_BLOCK_HEADER (页 262)”部分介绍。

“IM_Date”元素

说明	值
安装日期的格式： YYYY-MM-DD hh:mm	**示例** "2008-05-04 10:15" 字符串将被填补到最大长度。

7.15.21 PNIO_IM3_TYPE（I&M3 数据记录）

说明

PNIO_IM3_TYPE 结构用于通知 IO 控制器有关模块或设备的其它具体信息。

语法

```
typedef struct
{
    PNIO_BLOCK_HEADER   BlockHeader;    /* BlockType: 0x0023 */
    PNIO_UINT8          IM_Descriptor[54];
} ATTR_PACKED PNIO_IM3_TYPE;
```

“BlockHeader”的参数

名称	值
BlockType	0x0023
BlockLength	0x0010
BlockVersionHigh	0x01
BlockVersionLow	0x00

PNIO_BLOCK_HEADER 结构在“PNIO_BLOCK_HEADER (页 262)”部分介绍。

“IM_Signature”元素

说明	值
有关模块或设备的其它具体信息	**示例** “2008-05-04 10:15 更换” 字符串将被填补到最大长度。

7.15.22 PNIO_IM4_TYPE（I&M4 数据记录）

说明

PNIO_IM4_TYPE 结构用于通知 IO 控制器有关模块或设备的安全代码。

语法

```
typedef struct
{
    PNIO_BLOCK_HEADER   BlockHeader;    /* BlockType: 0x0024*/
    PNIO_UINT8          IM_Signature[54];
} ATTR_PACKED PNIO_IM4_TYPE;
```

“BlockHeader”的参数

名称	值
BlockType	0x0024
BlockLength	0x0010
BlockVersionHigh	0x01
BlockVersionLow	0x00

PNIO_BLOCK_HEADER 结构在“PNIO_BLOCK_HEADER (页 262)”部分介绍。

“IM_Descriptor”元素

包含安全代码。

7.15.23 PNIO_diag_alarm_send( ) 函数的“pData”参数

说明

这些结构不在头文件中定义。 这些结构仅用于说明如何根据 IEC 61158-6 PROFINET IO 构造“pData”参数。

说明

“pData”是指向网络格式（大端法）报警数据的指针。

报警数据是之前在 IO 设备上以网络格式（大端法）存储为压缩结构的诊断数据。

通过下表可了解根据所使用的函数如何来构造“pData”：

函数	结构
PNIO_diag_generic_add( )	结构 1
PNIO_diag_channel_add( )	结构 2
PNIO_diag_ext_channel_add( )	结构 3

说明

当警报退出状态时，除了“ChannelProp”的值，将在“pData”中传送与进入状态报警相同的传送值。

PNIO_build_channel_properties( ) 函数返回值“ChannelProp”。

要构成“Spec”参数的返回值，在没有其它问题时使用值 2，在存在其它问题时使用值 3。

“ChannelProp”值是由 PNIO_build_channel_properties( ) 函数形成的。 请按如下说明对 PNIO_build_channel_properties( ) 函数使用“Spec”参数：

2 - 退出状态时出现问题，没有其它问题。

3 - 退出状态时出现问题，存在其它问题。

说明

如果所有警报都退出状态，则传输空指针到“pData”。

结构 1

该诊断数据与 PNIO_diag_generic_add( ) 一起传输；“pData”指向压缩结构：

```
Struct
{
   PNIO_UINT16 ChannelNum;              /* ChannelNum parameter as for
                                        PNIO_diag_generic_add( )in big
                                        endian */
   PNIO_UINT16 ChannelProp;             /* ChannelProp parameter from
                                        PNIO_build_channel_properties
                                        ( ) in big endian */
   PNIO_UINT8 InfoData[InfoDataLen];    /* Diagnostics data to which the
                                        parameter pInfoData from
                                        PNIO_diag_generic_add( )
                                        points*/
};
```

结构 2

该诊断数据与 PNIO_diag_channel_add( ) 一起传输；“pData”指向压缩结构：

```
Struct
{
   PNIO_UINT16 ChannelNum;              /* ChannelNum parameter as for
                                        PNIO_diag_ channel_add( ) */
   PNIO_UINT16 ChannelProp;             /* ChannelProp parameter from
                                        PNIO_build_channel_properties
                                        ( ) */
   PNIO_UINT16 ChannelErrorType;        /* ChannelErrorType parameter
                                         as for PNIO_diag_channel_add
                                        ( ) */
};
```

结构 3

该诊断数据与 PNIO_diag_ext_channel_add( ) 一起传输： “pData”指向压缩结构：

```
Struct
{
   PNIO_UINT16 ChannelNum;       /* ChannelNum parameter as for
                                 PNIO_diag_ext_channel_add( ) */
   PNIO_UINT16 ChannelProp;      /* ChannelProp parameter from
                                 PNIO_diagbuild_channel_properties( ).*/
   PNIO_UINT16 ChannelErrorType;       /* ChannelErrorType parameter as
                                        for PNIO_diag_ext_channel_add( )
                                       */
   PNIO_UINT16 ExtChannelErrorType;    /* ExtChannelErrorType parameter
                                       as for PNIO_diag_ext_channel_
                                       add( ) */
   PNIO_UINT32 ExtChannelAddValue;     /* ExtChannelAddValue parameter
                                       as for PNIO_diag_ext_channel_
                                       add( ) */
};
```

特定于 CP 1616/CP 1604 的函数

8

概述

针对充当 IO 控制器或 IO 设备的 CP 1616/1604，可使用下列附加函数：

- PNIO_CP_set_appl_watchdog( )
- PNIO_CP_trigger_watchdog( )
- PNIO_CBF_APPL_WATCHDOG( )
- SERV_CP_set_type_of_station
- SERV_CP_get_fw_info
- SERV_CP_download( )
- SERV_CP_reset( )
- SERV_CP_info 编程接口
- SERV_CP_info_open( )
- SERV_CP_info_close( )
- SERV_CP_INFO_PRM
- SERV_CP_info_register_cbf( )
- SERV_CP_INFO_TYPE
- SERV_CP_INFO_STOP_REQ
- SERV_CP_INFO_PDEV_DATA
- SERV_CP_INFO_PDEV_INIT_DATA
- SERV_CP_INFO_LED_STATE

8.1 PNIO_CP_set_appl_watchdog( )（用户程序看门狗）

说明

用户程序使用此函数注册时间监视（看门狗）功能。
用户程序指定“wdTimeOutInMs”中的超时间隔。
激活看门狗后，用户程序在该超时间隔内调用 PNIO_CP_trigger_watchdog( ) 函数至少一次，以通知 IO-Base 接口用户程序仍在运行。
如果用户程序在此间隔内没有调用 PNIO_CP_trigger_watchdog( )，则监视停止，调用注册的回调函数，并且 CP 不再发送 IO 数据。 其后，除了 PNIO_contoller_close( ) 和 PNIO_device_close( )，不允许其它函数调用。 为了允许 CP 再次发送 IO 数据，用户程序必须先调用 PNIO_controller_close( ) 或 PNIO_device_close( )，然后再次调用 PNIO_contoller_open( ) 或 PNIO_controller_open( )。

说明

只允许一个用户程序注册用户程序超时监视功能。

说明

只有 PNIO_CP_trigger_watchdog 函数至少已被调用一次后，才会启动看门狗。

说明

如果超过用户程序的超时间隔，则会停止其它用户程序通过同一 CP 进行 PROFINET IO 通信。

说明

为了在超时间隔过后重新启动看门狗，必须取消注册之前的看门狗，然后再次注册。

说明

如果要反复注册与取消注册看门狗，则必须保证各函数调用间的等待时间。
该时间必须至少为 20 ms。

语法

```
PNIO_UINT32  PNIO_CP_set_appl_watchdog(
    PNIO_UINT32    CPIndex,                      //in
    PNIO_UINT32    wdTimeOutInMs,                //in
    PNIO_CBF_APPL_WATCHDOG  pnio_appl_wd_cbf     //in
);
```

参数

名称	说明
CpIndex	模块索引 - 用于唯一标识通信模块。 请参见此参数的组态（CP 的“模块索引”）。
wdTimeOutInMs	以 ms 指定的超时间隔。 要取消注册用户程序看门狗，该值必须设置为 0。
pnio_appl_wd_cbf	在超出超时间隔后调用的回调函数。 要取消注册用户程序看门狗，该值必须设置为 0。

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_ALLREADY_DONE
- PNIO_ERR_CREATE_INSTANCE
- PNIO_ERR_INTERNAL
- PNIO_ERR_MAX_REACHED
- PNIO_ERR_PRM_CP_ID

8.2 PNIO_CP_trigger_watchdog( )（用户程序看门狗）

说明

此函数用于重新触发用户程序看门狗。

语法

```
PNIO_UINT32  PNIO_CP_trigger_watchdog(
    PNIO_UINT32     CPIndex               //in
);
```

参数

名称	说明
CpIndex	CP 索引，将为此 CP 重新触发用户程序看门狗。

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_PRM_CP_ID

8.3 PNIO_CBF_APPL_WATCHDOG( )（用户程序看门狗）

说明

只要用户程序看门狗响应，就通过 IO-Base 接口调用此回调函数。
调用此回调函数后，已调用 PNIO_controller_open( ) 的所有用户程序必须调用 PNIO_controller_close( )，已调用 PNIO_device_open( ) 的所有用户程序必须发送 PNIO_device_close( )。

语法

```
typedef void (*PNIO_CBF_APPL_WATCHDOG)(
    PNIO_UINT32     CPIndex               //in
);
```

参数

名称	说明
CpIndex	CP 索引，为此 CP 注册了用户程序看门狗。

返回值

无返回值。

8.4 SERV_CP_set_type_of_station

说明

此函数会在固件中设置一个新的设备类型。
必须先调用此函数，然后再调用“PNIO_controller_open()/PNIO_device_open”。

只有在复位 CP
后，更改设备类型才会生效。这意味着调用“SERV_CP_set_type_of_station()”后，需要进行固件复位。 此后，需要重新启动 PC。

如果您“复位为出厂设置”，器件类型将复位为默认值“S7 PC”。
此后，需要在“复位”生效之前复位（重启）固件。

说明

命名设备类型

在 STEP 7 和 TIA Portal 中，仅能识别 19 个字符。
因此，建议设备类型避免使用较长的名称以及元音变音（ä、ö 等）或特殊字符。

语法

```
PNIO_UINT32 SERV_CP_set_type_of_station(

 PNIO_UINT32 CpIndex, // in

 char *TypeName // in

);
```

参数

参数	说明
CpIndex	模块索引 - 用于唯一标识通信模块。请参见此参数的组态（CP 的“模块索引”）。
TypeName	字符串。 要设置的设备类型。 注意： 最大长度为 255 个字符。 超出此长度的字符串将被截断。

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_xxx

8.5 SERV_CP_get_fw_info

说明

此函数将返回各个固件组态参数。 有关返回参数的详细描述，请参见输出结构的说明。

语法

```
PNIO_UINT32 PNIO_CODE_ATTR SERV_CP_get_fw_info (
 PNIO_UINT32 CpIndex,
 SERV_CP_FW_INFO_TYPE* p_info);
```

参数

参数	说明
CpIndex	输入参数，用于标识从中查询信息的 CP。 这些值在 <1, n> 范围内（n = MAX_CP16XX_DEVICES）。 PC 中只支持 CP 1616。 “MAX_CP16XX_DEVICES”在“host_linwin\driver\cp16xx_base.h”中定义。
SERV_CP_FW_INFO_TYPE* p_info	指向 CP 组态参数由固件返回的输出结构。 该结构由调用者（用户）提供（分配）。 数据通过“Servlib”复制到该结构中。

返回值

DPR_UINT32 PNIO_OK： 成功执行，输出结构中的数据有效。
如果产生错误，可能出现以下值：
（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：
PNIO_ERR_NO_FW_COMMUNICATION
如果产生错误，输出结构中的值无效。

8.5.1 结构 SERV_FW_VERS_TYPE 的说明

语法

```
typedef struct serv_fw_vers_s {
 PNIO_UINT16 v1;
 PNIO_UINT16 v2;
 PNIO_UINT16 v3;
 PNIO_UINT16 v4;
 PNIO_UINT16 v5;
 PNIO_UINT16 reserved;
} ATTR_PACKED SERV_FW_VERS_TYPE;
```

名称	说明
V1	1. 插入版本标识符（如，1 用 V1.0.0.0.0 来表示）
V2	2. 插入版本标识符
V3	3. 插入版本标识符
V4	4. 插入版本标识符
V5	5. 插入版本标识符
保留	-

8.5.2 结构 SERV_CP_FW_INFO_TYPE 的说明

语法

```
typedef struct serv_cp_fw_info_s {
 SERV_FW_VERS_TYPE fw_version;
 PNIO_UNIT8 cp_ser_nr[16];
 PNIO_UNIT8 mlfb[32];
 PNIO_UINT16 hw_version;
 PNIO_UNIT8 mac[6];
 PNIO_UINT32 ip_addr;
 PNIO_UINT32 ip_mask;
 PNIO_UINT32 default_router;
 PNIO_UINT8 name[256];
 PNIO_UINT8 TypeOfStation[256];

} ATTR_PACKED SERV_CP_FW_INFO_TYPE;
```

名称	
fw_version	固件版本，请参见“SERV_FW_VERS_TYPE”结构
cp_ser_nr	模块的序列号（以空值终止的 ASCII 字符串）
mlfb	模块的订货号（以空值终止的 ASCII 字符串）
hw_version	模块的硬件版本

名称	
mac	模块的第一个以太网地址（每个模块有多个以太网地址）。 第一个以太网地址用于主网络驱动程序的 PNIO 堆栈，第二个以太网地址用于 PROFINET 协议。 模块的每个 LAN 插口都有一个不同的以太网地址。
ip_addr	模块的 IP 地址（IE/PNIO 参数）
ip_mask	模块的 IP 子网掩码
default_router	模块的 IP 默认网关
名称	模块的 PROFINET 站名称
TypeOfStation	PROFINET 设备类型

8.6 SERV_CP_download( )（下载固件或组态）

说明

此函数用于将固件或组态下载到 CP 1616。仅在没有注册任何 IO 控制器或 IO 设备应用程序时可调用该函数。 下载固件后，CP 自动重启。
下载的固件或组态将保存在 CP 1616 上并具有保持性。

说明

此函数需要独占访问模块。 如果注册了其它应用程序（PROFINET IO 控制器以及 PROFINET IO 设备等），将终止此函数，并返回错误代码 PNIO_ERR_MAX_REACHED。

语法

```
PNIO_UINT32  SERV_CP_download(
    PNIO_UINT32                CpIndex,         // in
    PNIO_CP_DLD_TYPE           DataType,        // in
    PNIO_UINT8                *pData,           // in
    PNIO_UINT32                DataLen          // in
);
```

参数

名称	说明	
CpIndex	CP 索引，为此 CP 注册了用户程序看门狗。	
DataType	PNIO_CP_DLD_CONFIG	下载 XDB 格式的 STEP 7 组态
	PNIO_CP_DLD_FIRMWARE	下载 FWL 格式的固件
pData	指向包含 STEP 7 组态或固件的存储区的指针。	
DataLen	包含 STEP 7 组态或固件的存储区的长度。	

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_CORRUPTED_DATA
- PNIO_ERR_CREATE_INSTANCE
- PNIO_ERR_DRIVER_IOCTL_FAILED
- PNIO_ERR_FLASH_ACCESS
- PNIO_ERR_INTERNAL
- PNIO_ERR_MAX_REACHED
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_NO_RESOURCE
- PNIO_ERR_OS_RES
- PNIO_ERR_PRM_CP_ID
- PNIO_ERR_PRM_TYPE

8.7 SERV_CP_reset( )（模块重启）

说明

此函数用于重启 CP 1616 模块。

语法

```
PNIO_UINT32  SERV_CP_reset(
    PNIO_UINT32              CpIndex,         // in
    PNIO_CP_RESET_TYPE       ResetType        // in
);
```

参数

名称	说明	
CpIndex	模块索引 - 用于唯一标识通信模块	
ResetType	PNIO_CP_RESET_SAFE	只有未注册应用任何程序时可重启。
	PNIO_CP_RESET_FORCE	注册了应用程序时也可重启。

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_CREATE_INSTANCE
- PNIO_ERR_DRIVER_IOCTL_FAILED
- PNIO_ERR_INTERNAL
- PNIO_ERR_MAX_REACHED
- PNIO_ERR_NO_RESOURCE
- PNIO_ERR_PRM_CP_ID

8.8 SERV_CP_info 编程接口

说明

SERV_CP_info 编程接口包含 CP 1616/CP 1604
支持的常见服务函数。例如，用户应用程序使用此接口可以获取端口事件或模块 LED 状态的信息。

8.9 SERV_CP_info_open( )（向 SERV_CP_info 编程接口注册用户程序）

说明

使用此函数向 SERV_CP_info 编程接口注册用户程序。

其后，可以注册报警和信息事件的回调函数。

语法

```
PNIO_UINT32  SERV_CP_info_open(
    PNIO_UINT32                 CpIndex,          // in
    PNIO_UINT32                *Handle,           // out
    PNIO_UINT32                 Reserved
);
```

参数

名称	说明
CpIndex	模块索引 - 用于唯一标识通信模块。
Handle	注册成功后返回给用户程序的句柄。 必须在后续所有函数调用中包含该句柄。
Reserved	保留

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_CREATE_INSTANCE
- PNIO_ERR_DRIVER_IOCTL_FAILED
- PNIO_ERR_INTERNAL
- PNIO_ERR_MAX_REACHED
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_NO_RESOURCE
- PNIO_ERR_PRM_CP_ID

8.10 SERV_CP_info_close( )（从 SERV_CP_info 编程接口取消注册用户程序）

说明

使用此函数从 SERV_CP_info 编程接口取消注册用户应用程序。

语法

```
PNIO_UINT32  SERV_CP_info_close(
    PNIO_UINT32                 Handle          // in
);
```

参数

名称	说明
Handle	来自 SERV_CP_info_open( ) 的句柄

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_DRIVER_IOCTL_FAILED
- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_NO_RESOURCE
- PNIO_ERR_WRONG_HND

8.11 SERV_CP_INFO_PRM（回调事件参数）

说明

SERV_CP_info 接口上的各种回调事件具有相同的数据类型
SERV_CP_INFO_PRM，该数据类型使用“联合体”对各回调事件的不同参数进行分组。

语法

```
typedef struct
{
    PNIO_UINT32         Version;
    PNIO_UINT32         Handle;
    PNIO_UINT32         CpIndex;
    SERV_CP_INFO_TYPE   InfoType;
    union
    {
        ...           /*Union over the various
        ...             callback event parameters
                        (details in relevant sections)*/
    }u;
} ATTR_PACKED SERV_CP_INFO_PRM;
```

参数

名称	说明
Version	结构的版本
Handle	来自 SERV_CP_info_open( ) 的句柄
CpIndex	模块索引 - 用于唯一标识通信模块。
InfoType	回调事件类型

8.12 SERV_CP_info_register_cbf( )（回调函数注册）

说明

该函数注册一个回调函数。

语法

```
PNIO_UINT32  SERV_CP_info_register_cbf(
    PNIO_UINT32                 Handle,         // in
    SERV_CP_INFO_TYPE           InfoType,       // in
    SERV_CP_INFO_CBF            Cbf,            // in
    PNIO_UINT32                 Reserved
);
```

参数

名称	说明
Handle	来自 SERV_CP_info_open( ) 的句柄
InfoType	要注册回调函数“Cbf”的回调事件类型；请参见“SERV_CP_INFO_PRM（回调事件参数） (页 285)”部分。
Cbf	回调函数的地址，该回调函数在回调事件“CbeType”抵达后启动。 如果 Cbf 等于零，则取消注册之前注册的回调函数并且不再将“InfoType”事件转发到用户应用程序。

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_ALREADY_DONE
- PNIO_ERR_DRIVER_IOCTL_FAILED
- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_FW_COMMUNICATION
- PNIO_ERR_NO_RESOURCE
- PNIO_ERR_WRONG_HND

8.13 SERV_CP_INFO_TYPE（回调事件类型）

说明

SERV_CP_info 用户编程接口可识别以下回调事件类型：

回调事件类型	回调事件
SERV_CP_INFO_PDEV_DATA	端口报警到达。
SERV_CP_INFO_PDEV_INIT_DATA	当前端口状态到达。*
SERV_CP_INFO_LED_STATE	LED 状态变化到达。
SERV_CP_INFO_STOP_REQ	表示远程固件下载或重新组态请求

***** 此回调事件类型不要求显式注册并且在注册 PNIO_CP_INFO_PDEV_DATA 事件时自动注册一次。

8.14 回调事件 SERV_CP_INFO_STOP_REQ（注册远程下载请求）

说明

SERV_CP_INFO_STOP_REQ
回调事件向用户程序报告已收到远程下载请求（固件更新或重新组态）。
该回调事件必须已经向 SERV_CP_info_register_cbf( ) 函数注册。

接收该回调后，用户程序必须发送 PNIO_controller_close( )。
之后，用户程序可再次调用 SERV_CP_info_open( )。

但是，函数 SERV_CP_info_close( ) 及 SERV_CP_info _open( ) 不得在属于回调事件 SERV_CP_INFO_STOP_REQ 的函数内执行。

下载期间，SERV_CP_info_open( )
函数会返回错误代码“PNIO_ERR_CONFIG_IN_UPDATE”或“PNIO_ERR_NO_FW_COMMUNICATION”。

成功下载后，SERV_CP_info_open( ) 函数返回“PNIO_OK”错误代码。

说明

如果用户程序不注册该回调，则只要用户程序处于活动状态，就不能进行远程下载请求或重新组态。

语法

此事件类型无其它具体参数作为 SERV_CP_INFO_PRM
结构中“union”的一部分；请参见“SERV_CP_INFO_PRM（回调事件参数）
(页 285)”部分。

8.15 回调事件 SERV_CP_INFO_PDEV_DATA（端口报警到达）

说明

SERV_CP_INFO_PDEV_DATA 回调事件向用户程序报告端口报警到达。

端口报警通知用户程序在端口状态参数变化，例如，链路中断、帧丢失及冗余信息。
该回调事件必须已经向 SERV_CP_info_register_cbf( ) 函数注册。

以下语法显示此事件的具体参数，将其作为 SERV_CP_INFO_PRM 结构中“联合体”的一部分；请参见“SERV_CP_INFO_PRM（回调事件参数） (页 285)”部分。

PNIO_ALARM_TINFO 的语法

```
typedef struct
{
    PNIO_UINT16         CompatDevGeoaddr;
    PNIO_UINT8          ProfileType;
    PNIO_UINT8          AinfoType;
    PNIO_UINT8          ControllerFlags;
    PNIO_UINT8          DeviceFlag;
    PNIO_UINT16         PnioVendorIdent;
    PNIO_UINT16         PnioDevIdent;
    PNIO_UINT16         PnioDevInstance;
} ATTR_PACKED PNIO_ALARM_TINFO;
```

PNIO_ALARM_TINFO 的元素

名称	说明
CompatDevGeoaddr	位 0 - 10 = 设备编号 位 11 - 14 = io-subsys-nr 位 15 = 1（pnio 标识符）
ProfileType	0x08 => 位 0 - 3 用于 PNIO 位 4 - 7 用于 DP
AinfoType	位 0-3 alinfotyp = 0x00 （透明）
ControllerFlags	位 0 = 1 （外部 dp 接口） 位 1 - 7 保留
DeviceFlag	位 0 用于 PNIO： apdu-stat-failure 位 1 - 7 保留
PnioVendorIdent	对于 PNIO： 供应商标识
PnioDevIdent	对于 PNIO： 设备标识符
PnioDevInstance	对于 PNIO： 设备实例

SERV_CIB_PDEV_EXTCHNL_DIAG 的语法

```
typedef struct            /* 详细信息请参见 IEC 61158-6 */
{
    PNIO_UINT16      Channel;
    PNIO_UINT16      Properties;
    PNIO_UINT16      ErrType;
    PNIO_UINT16      ExtErrType;
    PNIO_UINT16      ExtAdvAlLo;
    PNIO_UINT16      ExtAdvAlHi;
} ATTR_PACKED SERV_CIB_PDEV_EXTCHNL_DIAG;
                /* ExtChannelDiagnosisData 的编码 */
```

SERV_CIB_PDEV_EXTCHNL_DIAG 的元素

名称	说明	
Channel	章节“区域 ChannelNumber 的编码” 0x0000-0x7FFF 制造商指定 0x8000 子模块： 子模块 0x8001 - 0xFFFF：保留	
Properties	章节“区域 ChannelProperties 的编码”	
	位 0 - 7： 位 8 ： 位 9 - 10 位 11 - 12： 位 13 - 15：	ChannelProperties.Type ChannelProperties.Accumulative ChannelProperties.Maintenance ChannelProperties.Specifier ChannelProperties.Direction
ErrType	章节“区域 ChannelErrorType 的编码”	
ExtErrType	章节“区域 ExtChannelErrorType 的编码”	
ExtAdvAlLo	章节“区域 ExtChannelAddValue 的编码”	
ExtAdvAlHi	章节“区域 ExtChannelAddValue 的编码”	

SERV_CIB_PDEV_DATA 的语法

```
typedef struct
{
    PNIO_UINT32                 Version;
    PNIO_IO_TYPE                IODataType;        /* in, out */
    PNIO_UINT32                 LogAddr;
    PNIO_UINT32                 Slot;
    PNIO_UINT32                 Subslot;
    PNIO_ALARM_TYPE             AlarmType;
    PNIO_APRIO_TYPE             AlarmPriority;
    PNIO_UINT32                 AlarmSequence;
    PNIO_UINT32                 StationNr;
    PNIO_UINT8                  Reserved1[4];
    PNIO_UINT8                  Reserved2[48];
    PNIO_ALARM_TINFO            AlarmTinfo;
    SERV_CIB_PDEV_EXTCHNL_DIAG  Diagnostics;
    PNIO_UINT8                  DiagDs[4];
    PNIO_ALARM_INFO             AlarmAinfo;
} ATTR_PACKED SERV_CIB_PDEV_DATA;
```

SERV_CIB_PDEV_DATA 的元素

名称	说明
Version	结构的版本 - 总是 0。
IODataType	IO 数据的类型
LogAddr	组态期间分配的逻辑地址。
Slot	插槽
Subslot	子插槽
AlarmType	报警类型
AlarmPriority	报警优先级
AlarmSequence	因每个新报警递增的数字
StationNr	组态期间分配的站编号。
RESERVED1 [4]	保留
Reserved2[48]	保留
AlarmTinfo	扩展的组态参数
诊断	请参见“SERV_CIB_PDEV_EXTCHNL_DIAG 的语法”部分

名称	说明
DiagDs	对应诊断中断 OB (OB82) 中数据的附加数据，请参见 STEP 7 文档。
AlarmAinfo	报警数据

8.16 回调事件 SERV_CP_INFO_PDEV_INIT_DATA（当前端口状态数据到达）

说明

SERV_CP_INFO_PDEV_INIT_DATA 回调事件向用户程序报告当前端口状态消息到达。

在 PROFINET IO 中，每个以太网接口包含一个接口子模块（子模块号 = 0x8i00）和 n 个端口子模块（子模块号 = 0x8ipp，pp = 端口号，i = 分配的接口号）。

以下语法显示此事件的具体参数，将其作为 SERV_CP_INFO_PRM 结构中“联合体”的一部分；请参见“SERV_CP_INFO_PRM（回调事件参数） (页 285)”部分。

语法

```
typedef struct
{
    PNIO_UINT8          Reserved1;
    PNIO_UINT8          Reserved2;
    PNIO_UINT16         Reserved3;

    PNIO_IO_TYPE        IODataType;     /* 输入、输出 */
    PNIO_UINT32         LogAddr;        /* 逻辑地址，
                                           使用工程工具
                                           组态 */
    PNIO_UINT32         Slot;           /* 插槽号 */
    PNIO_UINT32         Subslot;        /* 子插槽号 */

    PNIO_UINT16         PortState;      /* 位域，
                                           SERV_CIB_PS_... */

    PNIO_UINT16         Reserved4;
} ATTR_PACKED SERV_CIB_PDEV_INIT_STATE;

typedef struct
{
    PNIO_UINT32       ItemNumber;
    SERV_CIB_PDEV_INIT_STATE
              InitState[SERV_CIB_PDEV_MAX_PORT_ITEMS];
} ATTR_PACKED SERV_CIB_PDEV_INIT_DATA;
```

参数

名称	说明
Reserved1	保留
Reserved2	保留
Reserved3	保留
IODataType	IO 数据（输入/输出数据）的类型
LogAddr	组态期间分配的逻辑地址。
Slot	插槽
Subslot	子插槽
PortState	端口状态
Reserved4	保留

说明

如果还没有模块端口的组态，则 SERV_CP_INFO_PDEV_INIT_DATA 回调事件的“ItemNumber”参数为 0 值。

8.17 回调事件 SERV_CP_INFO_LED_STATE（LED 状态变化到达）

说明

SERV_CP_INFO_LED_STATE 回调事件向用户程序报告 LED 状态变化到达。

例如，用户程序可以将此信息转换为屏幕显示。 该回调事件必须已经向 SERV_CP_info_register_cbf( ) 函数注册。

以下语法显示此事件的具体参数，将其作为 SERV_CP_INFO_PRM 结构中“联合体”的一部分；请参见“SERV_CP_INFO_PRM（回调事件参数） (页 285)”部分。

语法

```
typedef enum
{
  SERV_CIB_LED_TYPE_BF = 1,         /* 总线故障 LED */
  SERV_CIB_LED_TYPE_SF = 2          /* 组故障 LED */
} SERV_CIB_LED_TYPE;

typedef enum
{
  SERV_CIB_LED_STATE_OFF = 1,      /* 熄灭 */
  SERV_CIB_LED_STATE_ON  = 2       /* 点亮 */
} SERV_CIB_LED_STATE;

typedef struct
{
    SERV_CIB_LED_TYPE     LedType;
    SERV_CIB_LED_STATE    LedState;
    PNIO_UINT8            Reserved1[4];
} ATTR_PACKED SERV_CIB_LED;
```

参数

名称	说明
LedType	LED 类型
LedState	LED 状态

产品特有的用于 PN 驱动程序的函数

9

作为 IO 控制器，还为 PN 驱动程序提供以下函数：

- SERV_CP_init
- SERV_CP_undo_init
- SERV_CP_get_network_adapters
- SERV_CP_startup
- SERV_CP_shutdown
- SERV_CP_set_trace_level

9.1 SERV_CP_init

说明

该函数具有多个任务，需要在 PN 驱动程序的启动序列中调用。它执行以下任务：

- 跟踪子系统（如果启用）的初始化
- PNIO 堆栈的初始化
- 邮箱子系统的初始化
- 线程的初始化
- 启动线程

说明

函数“SERV_CP_init”必须作为启动序列中第一个函数进行调用。随后才可调用函数“SERV_CP_startup”和“PNIO_controller_open”或“PNIO_interface_open”。

说明

如果组态或系统自适应存在问题，将使用跟踪功能进行故障排除。
欲分析跟踪数据并咨询如何设置正确的跟踪级别，请与 Siemens 支持团队联系。

语法

```
PNIO_UINT32 SERV_CP_init (PNIO_DEBUG_SETTINGS_PTR_TYPE DebugSetting);
```

参数

名称	说明
DebugSetting	通过此结构，用户可使一个回调函数可用，跟踪缓冲区已满时将立即调用该函数。还可以设置跟踪级别。若参数值为“NULL”，将不使用内部跟踪功能。

返回值

如果成功，则函数将返回“PNIO_OK”，否则将返回“PNIO_ERR_SEQUENCE”。

9.2 SERV_CP_undo_init

说明

此函数完成关闭序列。

说明

函数“SERV_CP_undo_init”必须作为关闭序列中的最后一个函数进行调用。在这之前，需要先调用函数“PNIO_controller_close”或“PNIO_interface_close”以及“SERV_CP_shutdown”。

语法

```
PNIO_UINT32 SERV_CP_undo_init ();
```

参数

该函数无参数。

返回值

该函数无返回值。

9.3 SERV_CP_get_network_adapters

说明

此函数返回所有检测到的网络适配器的列表。

说明

在调用该函数之前，必须先调用函数“SERV_CP_init”。

语法

```
PNIO_UINT32 SERV_CP_get_network_adapters (
 PNIO_CP_ID_PTR_TYPE CpList,
 PNIO_UINT8 *NrOfCp
);
```

参数

名称	说明
CpList	网络适配器列表
NrOfCp	“CpList”中的列表元素数。

返回值

如果成功，则函数将返回“PNIO_OK”，否则将返回“PNIO_ERR_NO_ADAPTER_FOUND”。

9.4 SERV_CP_startup

说明

此函数根据组态启动所有内部任务并组态 PNIO 堆栈。

下列参数由用户程序设置：

- 以太网接口的选择，在使用 WinPcap 驱动程序时根据 MAC 地址选择，在使用所有其它不同版本时根据 PCI 位置选择。PN Driver V1.0 仅支持单个接口的操作。也就是说，“CpList”只能包含一个元素，并且“NrOfCp”必须始终为“1”。
- PROFINET 接口的准备工作。

说明

必须先调用“SERV_CP_startup”函数，才能调用“PNIO_controller_open()”函数。

语法

```
PNIO_UINT32 SERV_CP_startup (
 PNIO_CP_ID_PTR_TYPE CpList,
 PNIO_UINT32 NrOfCp,
 PNIO_UINT8 *pConfigData,
 PNIO_UINT32 ConfigDataLen,
 PNIO_UINT8 *pRemaData,
 PNIO_UINT32 RemaDataLen,
 PNIO_SYSTEM_DESCR *pSysDescr
);
```

参数

名称	说明
CpList	标识数组，用于选择要使用的网络适配器。
NrOfCp	“CpList”中的元素数。必须始终设置为“1”。

名称	说明
pConfigData	字符串形式的指针，指向包含 XML PROFINET 组态的存储器。必须用最后一个字节中的“0”指示字符串结束（终止 NULL）。
ConfigDataLen	以字节表示的“pConfigData”长度，不包含终止 NULL。
pRemaData	指向保持性数据的指针，可以为 NULL。
RemaDataLen	“pRemaData”参数所引用存储区的长度。
pSysDescr	指向结构“PNIO_SYSTEM_DESCR”的指针。

返回值

如果成功，则返回
PNIO_OK。如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_SEQUENCE
- PNIO_ERR_PRM_POINTER
- PNIO_ERR_CREATE_INSTANCE
- PNIO_ERR_INTERNAL
- PNIO_ERR_CORRUPTED_DATA

9.5 SERV_CP_shutdown

说明

该函数必须是退出用户程序时调用的最后一个函数。
在此函数返回后，所有内部线程均已退出，并且整个局部存储器已重新释放。

说明

必须先调用“PNIO_controller_close( )”函数，才能调用“SERV_CP_shutdown”。

语法

```
PNIO_UINT32 SERV_CP_shutdown ();
```

参数

无参数。

返回值

如果成功，则返回 PNIO_OK。
如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_SEQUENCE

9.6 SERV_CP_set_trace_level

说明

凭借此函数，可在运行过程中更改各内部组件的跟踪级别。调用“SERV_CP_startup( )”函数后，可以随时调用此函数。在调用后，SERV_CP_set_trace_level 函数将立即返回，并会通过“CbfSetTraceLevelDone”回调函数确认跟踪级别已成功更改。

语法

```
PNIO_UINT32 SERV_CP_set_trace_level (

PNIO_UINT32 Component,
PNIO_UINT32 TraceLevel,
PNIO_PNTRC_SET_TRACE_LEVEL_DONE CbfSetTraceLevelDone);
```

参数

名称	说明
组件	选择 PNIO 堆栈组件（servusrx.h 中的 enum pnio_stack_comp）。
TraceLevel	选择 PNIO 堆栈跟踪级别（seruvsrx.h 中的 enum pnio_trace_level）。
CbfSetTraceLevelDone	确认更改的回调函数。

返回值

如果成功，则返回
PNIO_OK。如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_SEQUENCE
- PNIO_ERR_INTERNAL
- PNIO_ERR_PRM_INVALIDARG

9.7 PNIO_IOS_RECONFIG_ MODE

说明

函数“PNIO_IOS_RECONFIG_
MODE()”用于列出“PNIO_interface_reconfig()”函数的“Mode”参数的可能变型。

语法

```
typedef enum {

 PNIO_IOS_RECONFIG_MODE_DEACT = 1,

 PNIO_IOS_RECONFIG_MODE_TAILOR = 2

}PNIO_IOS_RECONFIG_MODE;
```

元素

名称	说明
PNIO_IOS_RECONFIG_MODE_DEACT	取消激活所有 IO 设备。
PNIO_IOS_RECONFIG_MODE_TAILOR	通过用户设置的组态开始重新组态 IO 系统，然后根据此组态激活 IO 设备。

9.8 用于 PN 驱动程序的数据类型

9.8.1 PNIO_CP_SELECT_TYPE

说明

“PNIO_CP_SELECT_TYPE”数据类型用于选择网络适配器的寻址模式。

语法

```
typedef enum
{
 PNIO_CP_SELECT_WITH_PCI_LOCATION = 1,
 PNIO_CP_SELECT_WITH_MAC_ADDRESS = 2
} PNIO_CP_SELECT_TYPE;
```

元素

名称	说明
PNIO_CP_SELECT_WITH_PCI_LOCATION	通过 PCI 位置进行选择
PNIO_CP_SELECT_WITH_MAC_ADDRESS	根据 MAC 地址选择（仅限 WinPcap）

9.8.2 PNIO_MAC_ADDR_TYPE

说明

“PNIO_MAC_ADDR_TYPE”数据类型显示在使用 WinPCAP 时 MAC 地址的传送数组。

语法

```
#define PNIO_MAC_ADDR_SIZE 6

typedef PNIO_UINT8 PNIO_MAC_ADDR_TYPE[PNIO_MAC_ADDR_SIZE];
```

9.8.3 PNIO_PCI_LOCATION_TYPE

说明

“PNIO_PCI_LOCATION_TYPE”显示网络适配器 PCI 位置的传送参数。

语法

```
typedef struct pnio_pci_location_s
{
 PNIO_UINT16 BusNr;
 PNIO_UINT16 DeviceNr;
 PNIO_UINT16 FunctionNr;
} PNIO_PCI_LOCATION_TYPE;
```

元素

名称	说明
BusNr	PCI 总线编号
DeviceNr	PCI 设备编号
FunctionNr	PCI 函数编号

9.8.4 PNIO_DEBUG_SETTINGS_TYPE/PNIO_DEBUG_SETTINGS_PTR_TYPE

说明

此结构包含回调函数，该函数用于发送跟踪子系统的初始跟踪级别。用户将此结构指定为“SERV_CP_init”的参数。如果参数“CbfPntrcBufferFull”设置为“零”(ZERO)，则将不使用内部跟踪功能。否则，如果跟踪缓冲区溢出，将调用此回调函数，且用户可以备份已传送的跟踪数据。

语法

```
typedef struct pnio_debug_settings_s
{
 PNIO_PNTRC_BUFFER_FULL CbfPntrcBufferFull;
 PNIO_UINT8 TraceLevels[PNIO_TRACE_COMP_NUM];
} PNIO_DEBUG_SETTINGS_TYPE, *PNIO_DEBUG_SETTINGS_PTR_TYPE;
```

元素

名称	说明
CbfPntrcBufferFull	用于传送跟踪数据。
TraceLevels	面向所有内部模块的数组，用于设置初始跟踪级别。

9.8.5 PNIO_PNTRC_BUFFER_FULL()

说明

此回调函数用于接收跟踪数据。因此用户可以选择将跟踪数据保存在文件系统或类似系统中。

语法

```
typedef void (*PNIO_PNTRC_BUFFER_FULL)
 ( PNIO_UINT8 * pBuffer,
 PNIO_UINT32 BufferSize );
```

元素

名称	说明
pBuffer	指向跟踪数据缓冲区的指针
BufferSize	“pBuffer”参数所引用存储区的长度。

9.8.6 PNIO_PNTRC_SET_TRACE_LEVEL_DONE()

说明

此回调函数在跟踪级别设置后立即调用。此回调通过调用 SERV_CP_set_trace_level 进行注册。

语法

```
typedef void (*PNIO_PNTRC_SET_TRACE_LEVEL_DONE)(void);
```

9.8.7 PNIO_CP_ID_TYPE/PNIO_CP_ID_PTR_TYPE

说明

数据类型“PNIO_CP_ID_TYPE”包含关于网络适配器的信息。

语法

```
typedef struct pnio_cp_id_s {
 PNIO_CP_SELECT_TYPE CpSelection;
 PNIO_MAC_ADDR_TYPE CpMacAddr;
 PNIO_PCI_LOCATION_TYPE CpPciLocation;
 PNIO_UINT8 Description[300];
} PNIO_CP_ID_TYPE, *PNIO_CP_ID_PTR_TYPE;
```

元素

名称	说明
CpSelection	用于选择网络适配器的寻址模式。 对于“WINPcap”，需要将该值设为“PNIO_CP_SELECT_WITH_MAC_ADDRESS”。
CpMacAddr	网络适配器的 MAC 地址 (WinPcap)。

名称	说明
CpPciLocation	网络适配器的 PCI 位置。
Description[300]	包含网络适配器的说明。

9.8.8 PNIO_SET_IP_NOS_MODE_TYPE

说明

数据类型“PNIO_SET_IP_NOS_MODE_TYPE”用于列出“PNIO_interface_set_ip_and_nos( )”函数的“Mode”参数的可能变型。

语法

```
typedef enum {

 PNIO_SET_IP_MODE = 0x0001,

 PNIO_SET_NOS_MODE = 0x0002

}PNIO_SET_IP_NOS_MODE_TYPE;
```

元素

名称	说明
PNIO_SET_IP_MODE	可以更改“IP 组合设置”。
PNIO_SET_NOS_MODE	可以更改站名。

9.8.9 PNIO_IPv4

说明

数据类型“PNIO_IPv4”用于设置函数“PNIO_interface_set_ip_and_nos”的“IP 组合设置”。

语法

```
typedef struct {
```

```
 PNIO_UINT8 IpAddress[4];
 PNIO_UINT8 NetMask[4];
 PNIO_UINT8 Gateway[4];
 PNIO_BOOL Remanent;
} PNIO_IPv4;
```

元素

名称	说明
IpAddress[4]	IP 地址
NetMask[4]	子网掩码
Gateway[4]	网关
Retentive	如果已设置参数“PNIO_TRUE”，则可以使用回调函数永久性地保存参数。

9.8.10 PNIO_NOS

说明

数据类型“PNIO_NOS”用于设置函数“PNIO_interface_set_ip_and_nos”的站名。

语法

```
typedef struct {
 PNIO_UINT8 Nos[256];
 PNIO_UINT16 Length;
 PNIO_BOOL Remanent
;} PNIO_NOS;
```

元素

名称	说明
Nos[256]	站名称
Length	站名称长度
Retentive	如果已设置参数“PNIO_TRUE”，则可以使用回调函数永久性地保存参数。

以太网接口数据类型与函数的说明

10

说明

说明

本部分中介绍的函数和回调函数仅可用于产品 PN 驱动程序。

除 IO 控制器功能外，还可以使用本地以太网接口的功能来执行以下操作：

- 读取本地以太网接口的数据记录。
- 接收本地以太网接口生成的报警。
- 调整参数，如本地以太网接口的“IP 组合设置”或站名称。

说明

为了能够使用此功能，需要调用函数“PNIO_interface_open()”。

概述

下表概述了 SIMATIC NET 产品提供的接口功能。

函数组和名称	产品		
	SOFTNET PN IO (RT)	CP 1616/CP 1604 (RT + IRT)，带有版本 V2.0 及更高版本的 DK-16xx	PN 驱动程序
管理函数			
PNIO_interface_open()	-	-	X
PNIO_interface_register_cbf()	-	-	X
PNIO_interface_close( )	-	-	X
PNIO_interface_set_ip_and_nos()	-	-	X
回调事件 PNIO_CBE_IFC_SET_IP_AND_NOS	-	-	X

函数组和名称	产品		
	SOFTNET PN IO (RT)	CP 1616/CP 1604 (RT + IRT)，带有版本 V2.0 及更高版本的 DK-16xx	PN 驱动程序
数据接口			
PNIO_interface_rec_read_req()	-	-	X
回调事件 PNIO_CBE_IFC_REC_READ_CONF	-	-	X
报警接口			
回调事件 PNIO_CBE_IFC_ALARM_IND	-	-	X

10.1 管理函数

10.1.1 PNIO_interface_open()

说明

利用此函数，可以接收报警、读取数据记录以及更改“IP 组合设置”，还能分配本地以太网接口的站名称。必须已调用函数“PNIO_interface_open()”，才能使用本地以太网接口的功能（设置 IP 或“NameOfStation”、读取数据记录、接收报警等...）。

语法

```
PNIO_UINT32 PNIO_interface_open(
 PNIO_UINT32 CpIndex
 PNIO_CBF cbf_rec_read_conf
 PNIO_CBF cbf_alarm_ind
 PNIO_UINT32 *Handle
);
```

参数

名称	说明
CpIndex	模块索引
cbf_rec_read_conf	回调函数的地址，该回调函数在读数据记录作业完成后立即调用。回调事件类型为“PNIO_CBE_IFC_REC_READ_CONF”。也可以传送“零”(ZERO)。
cbf_alarm_ind	回调函数的地址，该回调函数在显示报警时立即调用。回调事件类型为“PNIO_CBE_IFC_ALARM_IND”。也可以传送“零”(ZERO)。
Handle	函数成功执行后返回给用户的句柄，该句柄将引用本地以太网接口。此句柄必须与本地以太网接口的所有后续调用结合使用。

返回值

如果成功，则返回“PNIO_OK”。

如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_NO_RESOURCE
- PNIO_ERR_PRM_CALLBACK
- PNIO_ERR_PRM_CP_ID
- PNIO_ERR_PRM_HND
- PNIO_ERR_SEQUENCE

10.1.2 PNIO_interface_register_cbf()

说明

利用此函数，可在调用“PNIO_interface_open()”函数后，通过以太网接口注册回调函数。

语法

```
PNIO_UINT32 PNIO_interface_register_cbf(

 PNIO_UINT32 Handle
```

```
 PNIO_CBE_TYPE CbeType

 PNIO_CBF Cbf

);
```

参数

名称	说明
Handle	“PNIO_interface_open()”的句柄
CbeType	将注册回调函数的回调事件类型。
Cbf	要调用的回调函数的地址。 **注意** 函数指针不允许为空。 如果函数指针已经注册，则无法再注册其他指针

返回值

如果成功，则返回“PNIO_OK”。

如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_ALLREADY_DONE
- PNIO_ERR_INTERNAL
- PNIO_ERR_PRM_CALLBACK
- PNIO_ERR_PRM_TYPE
- PNIO_ERR_WRONG_HND

10.1.3 PNIO_interface_close()

说明

利用此函数可以关闭本地以太网接口。随后将不会调用注册的回调函数。

说明

在“PNIO_interface_close()”的执行完成之前，仍可调用回调函数。

说明

在关闭本地以太网接口后，随“PNIO_interface_open()”提供的句柄将失效且不再使用。

语法

```
PNIO_UINT32 PNIO_interface_close(
 PNIO_UINT32 Handle
);
```

参数

名称	说明
Handle	“PNIO_interface_open()”的句柄

返回值

如果成功，则返回“PNIO_OK”。

如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_WRONG_HND

10.1.4 PNIO_interface_set_ip_and_nos()

说明

利用此函数，可以更改本地以太网接口的“IP 组合设置”和/或站名称。

说明

要使用此函数，必须已经指定硬件组态，并且可以在本地更改 IP 地址或站名。

语法

```
PNIO_UINT32 PNIO_interface_set_ip_and_nos(
 PNIO_UINT32 Handle,
 PNIO_SET_IP_NOS_MODE_TYPE Mode,
 PNIO_IPv4 IpV4,
 PNIO_NOS NoS,
);
```

参数

名称	说明
Handle	“PNIO_interface_open()”的句柄
Mode	• PNIO_SET_IP_MODE (0x0001)：将更改 IpV4。 • PNIO_SET_NOS_MODE (0x0002)：将更改站名称。
IPv4	将更改“IP 组合设置”。
NoS	将更改站名称。

返回值

如果成功，则返回“PNIO_OK”。

如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_WRONG_HND
- PNIO_ERR_MODE_VALUE
- PNIO_ERR_NOT_REENTERABLE
- PNIO_ERR_PRM_NOS_LEN
- PNIO_ERR_SET_IP_NOS_NOT_ALLOWED
- PNIO_ERR_PRM_IP PNIO_ERR_PRM_NOS

10.2 数据记录接口

10.2.1 PNIO_interface_rec_read_req()

说明

利用此函数，可以读取本地以太网接口的数据记录。

语法

```
PNIO_UINT32 PNIO_interface_rec_read_req(
 PNIO_UINT32 Handle,
 PNIO_ADDR *pAddr,
 PNIO_UINT32 RecordIndex,
 PNIO_REF ReqRef,
 PNIO_UINT32 Length
);
```

参数

名称	说明
Handle	“PNIO_interface_open()”的句柄
pAddr	本地以太网接口子模块的地址，将在该接口上执行读数据记录作业。
RecordIndex	数据记录索引
ReqRef	由您指定的引用。可通过此引用区分多个作业。可为此参数设置任意值，将通过回调事件“PNIO_CBE_IFC_REC_READ_CONF”返回该值。
Length	指定 4096 字节的最大记录长度。通过回调事件“PNIO_CBE_IFC_REC_READ_CONF”返回指向数据及其实际长度的指针。

返回值

如果成功，则返回“PNIO_OK”。

如果发生错误，可能出现以下值（有关值的含义，请参见头文件“pnioerrx.h”中的注释）：

- PNIO_ERR_INTERNAL
- PNIO_ERR_PRM_ADD
- PNIO_ERR_PRM_REC_INDEX
- PNIO_ERR_VALUE_LEN
- PNIO_ERR_WRONG_HND
- PNIO_ERR_NO_RESOURCE

10.2.2 PNIO_CBE_IFC_REC_READ_CONF

说明

此回调函数用于指示上一个读数据记录作业已完成。

语法

```
typedef struct{
 PNIO_UINT32 Length;
 const PNIO_UINT8 *pBuffer;
 PNIO_ADDR *pAddr;
 PNIO_UINT32 RecordIndex;
 PNIO_REF ReqRef;
 PNIO_ERR_STAT Err;
} ATTR_PACKED PNIO_CBE_PRM_REC_READ_CONF;
```

参数

名称	说明
Length	“pBuffer”参数指向的数据记录长度（以字节为单位）。
pBuffer	指向数据记录的指针，该记录将包含请求的数据记录。 仅当执行回调函数时数据缓冲区才有效，完成该函数后缓冲区立即失效。

名称	说明
pAddr	本地以太网接口模块的地址，该接口可响应读数据记录作业。
RecordIndex	数据记录索引
ReqRef	由您指定的引用。可通过此引用区分多个作业。通过调用“PNIO_interface_rec_read_req()”传送此参数。
Err	与作业执行有关的错误代码。

返回值

无返回值。

10.3 报警接口

10.3.1 PNIO_CBE_IFC_ALARM_IND

说明

利用此回调事件，可以指示本地以太网接口的报警。

说明

只要该回调事件尚未被确认，将不再指示其它报警。

语法

```
typedef struct{
 PNIO_ADDR *pAddr;
 PNIO_REF ReqRef;
 const PNIO_CTRL_ALARM_DATA *pAlarmData;
} ATTR_PACKED PNIO_CBE_PRM_ALARM_IND;
```

参数

名称	说明
pAddr	生成报警的模块的地址。
ReqRef	由您指定的引用。可通过此引用区分多个作业。通过调用“PNIO_interface_rec_read_req()”传送此参数。
pAlarmData	包含报警信息的结构。 “pAlarmData”指向“PNIO_CTRL_ALARM_DATA”类型的数据。仅当执行回调函数时数据缓冲区才有效，完成该函数后缓冲区立即失效。 PNIO_CTRL_ALARM_DATA 结构在 pniousrx.h 头文件中定义，且其中还包括“报警类型”和“报警说明符”参数。 有关报警的详细信息，请参见以下文档： • STEP 7 文档，例如 STEP 7 在线帮助中有关 SFB 54 的内容。 • IO 设备的文档

返回值

无返回值。

索引

P

S

Z H

S H

T

M