用户手册

R-IN32M3 Module (RY9012A0)

用户手册：软件
API 说明

RENESAS MCU
R-IN32M3-EC

Renesas Electronics
www.renesas.com

版本 2.05　2023 年 5 月

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.
   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

以下“注意事项”为从英语原稿翻译的中文译文，仅作为参考译文，英文版的“Notice”具有正式效力。

注意事项

1. 本文档中所记载的关于电路、软件和其他相关信息仅用于说明半导体产品的操作和应用实例。用户如在产品或系统设计中应用本文档中的电路、软件和相关信息或将此等内容用于其他目的时，请自行负责。对于用户或第三方因使用上述电路、软件或信息而遭受的任何损失和损害，瑞萨电子概不承担任何责任。
2. 瑞萨电子在此明确声明，对于因使用瑞萨电子产品或本文档中所述技术信息（包括但不限于产品数据、图、表、程序、算法、应用实例）而造成的与第三方专利、版权或其他知识产权相关的侵权或任何其他索赔，瑞萨电子不作任何保证并概不承担责任。
3. 本文档所记载的内容不应视为对瑞萨电子或其他人所有的专利、版权或其他知识产权作出任何明示、默示或其它方式的许可及授权。
4. 用户不得对瑞萨电子的任何产品进行全部或部分更改、修改、复制或反向工程。对于用户或第三方因上述更改、修改、复制或对瑞萨电子产品进行反向工程的行为而遭受的任何损失或损害，瑞萨电子概不承担任何责任。
5. 瑞萨电子产品根据其质量等级分为两个等级：“标准等级”和“高质量等级”。每种瑞萨电子产品的预期用途均取决于产品的质量等级，如下所示：
   标准等级： 计算机、办公设备、通讯设备、测试和测量设备、视听设备、家用电器、机械工具、个人电子设备、工业机器人等。
   高质量等级： 运输设备（汽车、火车、轮船等）、交通控制系统（交通信号灯）、大型通讯设备、关键金融终端系统、安全控制设备等。
   除非是瑞萨电子数据表或其他瑞萨电子文档中明确指定为高可靠性产品或用于恶劣环境的产品，否则瑞萨电子产品不能用于、亦未授权用于可能对人类生命造成直接威胁的产品或系统及可能造成人身伤害的产品或系统（人工生命维持装置或系统、植埋于体内的装置等）中，或者可能造成重大财产损失的产品或系统（太空系统、海底增音机、核能控制系统、飞机控制系统、关键装置系统、军用设备等）中。对于用户或任何第三方因使用不符合瑞萨电子数据表、使用说明书或其他瑞萨电子文档的瑞萨电子产品而遭受的任何损害或损失，瑞萨电子概不承担任何责任。
6. 使用瑞萨电子产品时，请参阅最新产品信息（数据表、使用说明书、应用指南、可靠性手册中的“半导体元件处理和使用一般注意事项”等），并确保使用条件在瑞萨电子指定的最大额定值、电源工作电压范围、散热特性、安装条件等范围内使用。对于在上述指定范围之外使用瑞萨电子产品而产生的任何故障、失效或事故，瑞萨电子概不承担任何责任。

7. 虽然瑞萨电子一直致力于提高瑞萨电子产品的质量和可靠性，但是，半导体产品有其自身的具体特性，如一定的故障发生率以及在某些使用条件下会发生故障等。除非是瑞萨电子数据表或其他瑞萨电子文档中指定为高可靠性产品或用于恶劣环境的产品，否则瑞萨电子产品未进行防辐射设计。用户负责执行安全保护措施，以避免因瑞萨电子产品失效或发生故障而造成身体伤害、火灾导致伤害或损害和/或其他对公众构成危险的事故。例如进行软硬件安全设计（包括但不限于冗余设计、防火控制以及故障预防等）、适当的老化处理或其他适当的措施等。由于对微机软件单独进行评估非常困难且不实际，所以请用户自行负责对最终产品或系统进行安全评估。
8. 关于环境保护方面的详细内容，例如每种瑞萨电子产品的环境兼容性等，请与瑞萨电子的营业部门联系。用户负责仔细并充分查阅对管制物质的使用或含量进行管理的所有适用法律法规（包括但不限于《欧盟RoHS指令》），并在使用瑞萨电子产品时遵守所有适用法律法规。对于因用户未遵守相应法律法规而导致的损害或损失，瑞萨电子概不承担任何责任。
9. 不可将瑞萨电子产品和技术用于或者嵌入日本国内或海外相应的法律法规所禁止生产、使用及销售的任何产品或系统中。也不可将瑞萨电子产品或技术用于(1)与大规模杀伤性武器（例如核武器、化学武器、生物武器或运送此等武器的导弹，包括无人机(UAV)）的开发、设计、制造、使用、存储等相关的任何目的；(2)与常规武器的开发、设计、制造或使用相关的任何目的；(3)扰乱国际和平与安全的任何其他目的，并且不可向任何第三方销售、出口、租赁、转让、或让与瑞萨电子产品或技术，无论直接或间接知悉或者有理由知悉该第三方或任何其他方将从事上述活动。用户必须遵守对各方或交易行使司法管辖权的任意国家/地区政府所公布和管理的任何适用出口管制法律法规。
10. 瑞萨电子产品的买方或分销商，或者分销、处置产品、或以其他方式向第三方出售或转让产品的任何其他方有责任事先向所述第三方通知本文档规定的内容和条件。
11. 在事先未得到瑞萨电子书面认可的情况下，不得以任何形式部分或全部再版、转载或复制本文档。
12. 如果对本文档所记载的信息或瑞萨电子产品有任何疑问，请向瑞萨电子的营业部门咨询。

（注1） 瑞萨电子：在本文档中指瑞萨电子株式会社及其控股子公司。

（注2） 瑞萨电子产品：指瑞萨电子开发或生产的任何产品。

（版本 4.0-1　2017 年 11 月）

公司总部

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

联系信息

有关产品、技术、文件最新版本或最近的销售办事处的详细信息，请访问：

www.renesas.com/contact/..

商标

处理微处理器和微处理器产品的通用预防措施

下列使用说明适用于 Renesas 的所有微处理器和微处理器产品。有关本文件中所涵盖产品的详细使用说明，请参阅文件的相关章节以及发布的任何产品技术更新。

1 防静电放电(ESD)

暴露于CMOS设备的强电场会导致栅氧化层破坏，并最终导致设备的运行退化。必须采取措施尽可能阻止静电的产生，并在静电产生时迅速使之消散。必须采取适当的环境控制措施。天气干燥时，应使用加湿器。建议避免使用容易聚积静电的绝缘体。必须在防静电容器、静电屏蔽袋或导电材料中储存和运输半导体器件。必须将所有测试和测量工具（包括工作台和地板）接地。操作员还必须佩戴防静电手环进行接地。禁止徒手触摸半导体器件。对于装有半导体器件的印刷电路板，也必须采取类似的预防措施。

2 通电时的处理

产品在通电时的状态是不确定的。LSI内部电路的状态是不确定的，寄存器设置和引脚在通电时的状态也是不确定的。在将复位信号应用于外部复位引脚的成品中，从开始通电直到复位过程完成，都不能保证引脚的状态。同样，对于通过片上通电复位功能复位的产品中的引脚状态，从开始通电直到功率达到规定的复位水平，也不能保证。

3 断电状态下的信号输入

当设备断电时，不要输入信号或I/O上拉电源。输入此类信号或I/O上拉电源而引起的电流注入可能导致故障，此时通过设备的异常电流可能导致内部元件降级。请按照产品文档中所述的断电状态下的输入信号指南进行操作。

4 未使用引脚的处理

根据手册中“未使用引脚的处理”下方给出的说明处理未使用的引脚。CMOS产品的输入引脚通常处于高阻抗状态。在未使用的引脚处于开路状态下运行时，在LSI附近会产生额外的电磁噪声，相关的贯穿电流在内部流动，可能出现的输入信号导致对引脚状态的错误识别，从而造成故障。

5 时钟信号

应用复位后，只有在运行中的时钟信号稳定后才会释放复位线。在程序执行期间切换时钟信号时，等待直到目标时钟信号稳定。当在复位期间由外部谐振器或外部振荡器产生时钟信号时，确保仅在时钟信号充分稳定后释放复位线。另外，在程序执行过程中，当切换到由外部谐振器或外部振荡器产生的时钟信号时，等待直到目标时钟信号稳定。

6 输入引脚处的电压施加波形

输入噪声或反射波引起的波形失真可能会导致故障。如果CMOS设备的输入由于噪声而停留在VIL（最大值）和VIH（最小值）之间的区域，则该设备可能会发生故障。当输入电平固定以及当输入电平通过VIL（最大值）和VIH（最小值）之间的区域时，注意防止抖振噪声进入设备。

7 禁止访问保留地址

对保留地址的访问是被禁止的。提供保留地址以供将来可能的功能扩展使用。不要访问这些地址，否则无法保证LSI的正确运行。

8 产品之间的差异

从一种产品更换到另一种产品之前，例如更换为型号不同的产品，应确认更换不会导致问题。在同一组别但件号不同的微处理器或微处理器产品的特性可能在内部存储容量、布局模式和其他因素方面存在差异，这些差异可能会影响电气特性的范围，例如特性值、运行裕度、抗噪性和辐射噪声量。更换为件号不同的产品时，应对给定的产品进行系统评估测试。

如何使用本手册

1. 目的和目标读者

本手册旨在让用户了解R-IN32M3 Module 的硬件功能和电气特性。供设计包含MCU的应用系统的用户使用。使用本手册之前，必须具备电路、逻辑电路和MCU的基本知识。

使用本手册时，应特别注意防范说明。这些防范说明在正文、每个章节的末尾和使用说明部分。

修订记录总结了修订和添加内容的位置。并未列出所有修订版本。有关详细信息，请参见本手册的正文。

以下文件适用于R-IN32M3 Module 。务必参考这些文件的最新版本。文件编号的最后四位（描述为****）表明每个文件的版本信息。所列文件的最新版本可从瑞萨电子公司网站获得。

文件类型	说明	文件标题	文件编号
数据手册	硬件概述和电气特性	R-IN32M3 Module 数据表	R19DS0109ED****
硬件用户手册	硬件规范（引脚分配、外围功能规范、电气特性、时序图）和操作说明	R-IN32M3 Module 用户手册：硬件	R19UH0122ED****
软件用户手册	API说明	R-IN32M3 Module 用户手册： 软件	本用户手册
快速启动指南	应用示例信息 主机CPU的示例程序。	R-IN32M3 Module 应用说明： 快速启动指南	R12QS0042CJ****
Renesas技术更新	产品规格、文件更新等	可从瑞萨电子公司网站获得。	

2. 编号和符号注释

注：
对正文中标记有“注”的项目进行解释

注意：
值得特别注意的项目

备注：
正文的补充说明

3. 缩略语和首字母缩略词列表

缩略语	全称
AC	应用程序控制器
ACK	确认
API	应用程序编程接口
APMS	非周期性协议机发送方
CC	通信控制器
CIP	通用工业协议
CM	配置管理器
CPU	中央处理单元
DD	设备检测
DHCP	动态主机配置协议
GOAL	通用开放抽象层
HTTP	超文本传输协议
HTTPD	超文本传输协议守候服务机
I/O	输入/输出
IP	网络协议
IOCS	输入输出对象使用程序状态
IOPS	输入输出对象提供程序状态
LLDP	链路层发现协议
MA	媒介适配器
MAC	介质访问控制
MCTC	微核到核
NVS	非易失性存储
OSAL	操作系统抽象层
PLC	可编程序控制器
RPC	远程过程调用
SNMP	简单网络管理协议
SPI	串行外围接口
TCP	传输控制协议
TLV	类型长度值
UART	通用非同步收发传输器
UDP	用户数据报协议
uGOAL	微型通用开放抽象层

目录

1. 简介 16
1.1 通信协议特性 17

2. 文件结构 18

3. 设备架构 19
3.1 架构 19
3.2 接口 19
3.2.1 硬件接口 19
3.2.2 SPI 软件接口 20
3.2.3 通信层/中间件的集成 20

4. 应用程序 21
4.1 简介 21
4.2 硬件设置 21
4.3 基本应用程序设置 22
4.4 默认功能 22
4.5 功能 22
4.5.1 设备检测 22
4.5.2 PNIO 23
4.5.3 EtherNet/IP 24
4.5.4 EtherCAT 24
4.5.5 通用数据提供程序 24
4.5.6 以太网启动延迟 24
4.6 外部复位 25
4.7 RPC 同步复位 25
4.7.1 通信控制器(CC) 25
4.7.2 应用程序控制器(AC) 25
4.8 IP设置 26
4.9 R-IN32M3 Module管理接口 27
4.9.1 设备检测(DD) 27
4.9.2 PNIO 28
4.9.3 日志管理器(LM) 29
4.9.4 NET 30
4.9.5 BOOT 31
4.9.6 配置管理器(CM) 31
4.9.7 ETH 31
4.9.8 EIP 32
4.9.9 HTTP 33
4.9.10 CCM 34
4.9.11 QUEUE 35
4.9.12 SNMP 35
4.9.13 MCTC 36
4.10 固件更新 37
4.10.1 更新通信控制器(CC) 37
4.10.1.1 固件包 37
4.10.1.2 控制接口 38
4.10.1.3 固件更新顺序 39
4.10.1.4 禁用DD时保留更新功能 40
4.10.2 应用程序控制器(AC)的功能更新 40
4.10.2.1 通过 HTTP 更新 AC 固件 41

4.10.2.2 通过 TCP 更新 AC 固件 42

5. 通信栈 43
5.1 简介 43
5.2 SPI数据交换 43
5.2.1 时钟域和通信周期 43
5.2.2 技术数据 44
5.2.3 SPI时序 45
5.2.3.1 SPI速度 45
5.2.3.2 SPI设置时序 45
5.2.3.3 SPI周期时间 45
5.2.4 SPI帧结构 46
5.2.4.1 Fletcher 16校验和（16位） 46
5.2.4.2 序列计数器（8位） 46
5.2.4.3 数据长度（8位） 46
5.3 远程过程调用(RPC) 47
5.3.1 RPC帧 47
5.3.1.1 结构 47
5.3.1.2 Fletcher-16校验和（16位） 47
5.3.1.3 本地序列（8位） 47
5.3.1.4 远程序列确认（8位） 47
5.3.1.5 数据长度（8位） 47
5.3.2 标志（8位） 48
5.3.2.1 结构 48
5.3.2.2 同步请求 48
5.3.2.3 同步确认 48
5.3.2.4 请求确认 48
5.3.3 RPC同步 48
5.3.4 RPC协议 49
5.3.4.1 简介 49
5.3.5 RPC请求/响应 49
5.3.5.1 结构 49
5.3.5.2 静态标识符 49
5.3.5.3 数据长度 50
5.3.5.4 RPC ID 50
5.3.5.5 函数ID 50
5.3.5.6 CTC ID 50
5.3.5.7 标志 50
5.3.5.8 数据 50
5.3.5.9 Fletcher-16校验和（16位） 50
5.4 通信栈 – PROFINET 51
5.4.1 goal_pnioCfgVendorIdSet – 设置供应商ID 51
5.4.2 goal_pnioCfgVendorId – 设置设备 ID 51
5.4.3 goal_pnioCfgVendorNameSet - 设置供应商名称 51
5.4.4 goal_pnioCfgPortDescSet - 设置LLDP端口描述 52
5.4.5 goal_pnioCfgSystemDescSet - 设置LLDP系统描述 52
5.4.6 goal_pnioCfgOrderIdSet - 设置订单ID 52
5.4.7 goal_pnioCfgSerialNumSet - 设置序列号 52
5.4.8 goal_pnioCfgHwRevSet - 设置硬件版本 53
5.4.9 goal_pnioCfgSwRevPrefixSet - 设置软件版本前缀 53
5.4.10 pnioCfgSwRevFuncEnhSet - 设置软件版本功能增强 53
5.4.11 goal_pnioCfgSwRevBugfixSet - 设置软件版本错误修复 54
5.4.12 goal_pnioCfgSwRevIntChgSet - 设置软件版本内部更改 54

5.4.13 goal_pnioCfgSwRevCntSet - 设置软件版本计数器 54
5.4.14 goal_pnioCfgIm1TagFuncSet - 设置I&M1标签函数 55
5.4.15 goal_pnioCfgIm1TagLocSet - 设置I&M1标签位置 55
5.4.16 goal_pnioCfgIm2DateSet - 设置I&M2日期 55
5.4.17 goal_pnioCfgIm3DescSet - 设置I&M3说明 56
5.4.18 goal_pnioCfgIm4SigSet - 设置I&M4签名（功能安全） 56
5.4.19 goal_pnioCfgLldpOrgExtSet - 配置特定于LLDP组织的扩展 56
5.4.20 goal_pnioCfgLldpOptTlvSet - 配置LLDP可选TLV参数 57
5.4.21 goal_pnioCfgLldpGenMacSet - 配置LLDP端口MAC地址生成 57
5.4.22 goal_pnioCfgIm14SupportSet - 配置I&M 1-4支持 58
5.4.23 goal_pnioCfgIm14CbSet - 配置I&M 1-4回调 58
5.4.24 goal_pnioCfgIm0CbSet - 配置I&M 0回调 58
5.4.25 goal_pnioCfgIm0FilterDataCbSet - 配置I&M 0筛选数据回调 59
5.4.26 goal_pnioCfgRecDataBusyBufsizeSet - 配置记录句柄存储计数 59
5.4.27 goal_pnioCfgRpcFragReqLenMaxSet - 配置最大记录大小 59
5.4.28 goal_pnioCfgRpcFragMaxCntSet–配置最大RPC片段数 59
5.4.29 goal_pnioCfgRpcFragEnableSet - 配置RPC碎片 60
5.4.30 goal_pnioCfgRpcSessionMaxCntSet - 配置最大RPC会话计数 60
5.4.31 goal_pnioDmSubslotAdd–映射subslot数据 60
5.4.32 goal_pnioDmSubslotIoxsAdd - 映射subslot IOCS/IOPS 60
5.4.33 goal_pnioDmApduAdd–映射APDU状态 62
5.4.34 goal_pnioDmDpAdd–映射数据提供程序状态 62
5.5 应用程序回调 – PROFINET 63
5.5.1 简介 63
5.5.2 GOAL_PNIO_CB_ID_ALARM_ACK_TIMEOUT - 报警 ACK 的超时等待 63
5.5.3 GOAL_PNIO_CB_ID_ALARM_NOTIFY_ACK - 收到的报警通知 ACK 64
5.5.4 GOAL_PNIO_CB_ID_ALARM_NOTIFY - 收到的报警通知 64
5.5.5 GOAL_PNIO_CB_ID_APPL_READY - 收到的应用程序就绪响应 64
5.5.6 GOAL_PNIO_CB_ID_BLINK - 闪烁请求 65
5.5.7 GOAL_PNIO_CB_ID_CONNECT_FINISH - 连接请求完成 67
5.5.8 GOAL_PNIO_CB_ID_CONNECT_REQUEST - 连接请求 67
5.5.9 GOAL_PNIO_CB_ID_CONNECT_REQUEST_EXP_START- 预期子模块阻塞开始 67
5.5.10 GOAL_PNIO_CB_ID_END_OF_PARAM - 收到的参数结束信息 68
5.5.11 GOAL_PNIO_CB_ID_END_OF_PARAM_PLUG –收到的插入参数结束信息 68
5.5.12 GOAL_PNIO_CB_ID_EXP_SUBMOD - 预期子模块 68
5.5.13 GOAL_PNIO_CB_ID_FACTORY_RESET - 恢复出厂设置 68
5.5.14 GOAL_PNIO_CB_ID_IO_DATA_TIMEOUT - 周期性超时 68
5.5.15 GOAL_PNIO_CB_ID_NET_IP_SET - IP配置更新 69
5.5.16 GOAL_PNIO_CB_ID_NEW_AR - 新应用程序关系 69
5.5.17 GOAL_PNIO_CB_ID_NEW_IO_DATA - 新IO数据 69
5.5.18 GOAL_PNIO_CB_ID_PLUG_READY - 收到的插入就绪响应 69
5.5.19 GOAL_PNIO_CB_ID_READ_RECORD - 读取记录数据 70
5.5.20 GOAL_PNIO_CB_ID_RELEASE_AR - 发布应用程序关系 72
5.5.21 GOAL_PNIO_CB_ID_RESET_TO_FACTORY - 恢复到出厂设置 72
5.5.22 GOAL_PNIO_CB_ID_STATION_NAME - 站名已更改 72
5.5.23 GOAL_PNIO_CB_ID_WRITE_RECORD - 写入记录数据 72
5.5.24 GOAL_PNIO_CB_ID_INIT - 堆栈已初始化 74
5.5.25 GOAL_PNIO_CB_ID_LLDP_UPDATE - LLDP更新 74
5.5.26 GOAL_PNIO_CB_ID_CONN_REQ_EXP_FINISH - 连接请求预期子模块阻塞完成 74
5.5.27 GOAL_PNIO_CB_ID_STATION_NAME_VERIFY - DCP站名验证 74
5.5.28 GOAL_PNIO_CB_ID_NET_IP_SET_VERIFY - DCP IP配置验证 75
5.6 通信栈 - EtherNet/IP 76
5.6.1 goal_eipCfgVendorIdSet 76

5.6.2 goal_eipCfgDeviceTypeSet 76
5.6.3 goal_eipCfgProductCodeSet 76
5.6.4 goal_eipCfgRevisionSet 77
5.6.5 goal_eipCfgSerialNumSet 77
5.6.6 goal_eipCfgProductNameSet 78
5.6.7 goal_eipCfgDomainNameSet 78
5.6.8 goal_eipCfgHostNameSet 78
5.6.9 goal_eipCfgNumExplicitConSet 79
5.6.10 goal_eipCfgNumImplicitConSetImpl 79
5.6.11 goal_eipCfgEthLinkCountersOn 79
5.6.12 goal_eipCfgEthLinkControlOn 80
5.6.13 goal_eipCfgChangeEthAfterResetOn 80
5.6.14 goal_eipCfgChangeIpAfterResetOn 80
5.6.15 goal_eipCfgNumSessionsSet 81
5.6.16 goal_eipCfgTickSet 81
5.6.17 goal_eipCfgO2TRunIdleHeaderOn 82
5.6.18 goal_eipCfgT2ORunIdleHeaderOn 82
5.6.19 goal_eipCfgQoSOn 82
5.6.20 goal_eipCfgNumDelayedEncapMsgSet 83
5.6.21 goal_eipCfgDhcpOn 83
5.6.22 goal_eipCfgDlrOn 83
5.6.23 goal_eipCfgAcdOn 84
5.6.24 goal_eipCfgAcdConflictFallbackIp 84
5.7 应用程序回调 – EtherNet/IP 85
5.7.1 简介 85
5.7.2 GOAL_EIP_CB_ID_INIT 86
5.7.3 GOAL_EIP_CB_ID_READY 86
5.7.4 GOAL_EIP_CB_ID_CONNECT_EVENT 86
5.7.5 GOAL_EIP_CB_ID_ASSEMBLY_DATA_RECV 87
5.7.6 GOAL_EIP_CB_ID_ASSEMBLY_DATA_SEND 87
5.7.7 GOAL_EIP_CB_ID_RUN_IDLE_CHANGED 88
5.7.8 GOAL_EIP_CB_ID_LED_CHANGED 88
5.7.9 GOAL_EIP_CB_ID_DEVICE_RESET 88
5.7.10 GOAL_EIP_CB_ID_REVISION_CHECK 89
5.7.11 GOAL_EIP_CB_ID_ACD_CONFLICT 89
5.8 通信栈 –EtherCAT 90
5.8.1 CoE API 90
5.8.2 EoE API 90
5.8.3 FoE API 91
5.8.4 EtherCAT 状态机 91
5.8.5 数据层指示函数 92
5.8.6 分布时钟 API 92
5.9 应用程序回调 – EtherCAT 93
5.9.1 简介 93
5.9.2 GOAL_ECAT_CB_ID_SDO_DOWNLOAD 95
5.9.3 GOAL_ECAT_CB_ID_SDO_UPLOAD 95
5.9.4 GOAL_ECAT_CB_ID_RxPDO_RECEIVED 95
5.9.5 GOAL_ECAT_CB_ID_TxPDO_PREPARE 95
5.9.6 GOAL_ECAT_CB_ID_SYNC_FAIL 95
5.9.7 GOAL_ECAT_CB_ID_NEW_DL_STATE 95
5.9.8 GOAL_ECAT_CB_ID_NEW_DC_CONFIG 95
5.9.9 GOAL_ECAT_CB_ID_DC_FAIL 95
5.9.10 GOAL_ECAT_CB_ID_SM_WATCHDOG_EXPIRED 95

5.9.11 GOAL_ECAT_CB_ID_EXPLICIT_DEV_ID 95
5.9.12 GOAL_ECAT_CB_ID_NEW_ESM_STATE_ENTERED 96
5.9.13 GOAL_ECAT_CB_ID_NEW_ESM_STATE_REQUESTED 96
5.9.14 GOAL_ECAT_CB_ID_FOE_READ_REQ 96
5.9.15 GOAL_ECAT_CB_ID_FOE_READ_DATA 96
5.9.16 GOAL_ECAT_CB_ID_FOE_WRITE_REQ 96
5.9.17 GOAL_ECAT_CB_ID_FOE_WRITE_DATA 96
5.9.18 GOAL_ECAT_CB_ID_FOE_ERROR 96
5.9.19 GOAL_ECAT_CB_ID_RUN_LED_STATE 96
5.9.20 GOAL_ECAT_CB_ID_ERROR_LED_STATE 96

6. 应用程序编程接口 97
6.1 设备特定函数 97
6.1.1 appl_ccmRpcInit 97
6.1.2 appl_ccmUpdateAllow 98
6.1.3 appl_ccmUpdateDeny 98
6.1.4 appl_ccmInfo 99
6.1.5 appl_ccmFaultStateSet 100
6.1.6 appl_ccmCommResetSet 100
6.1.7 appl_ccmLogEnable 101
6.1.8 appl_ccmLogToAcEnable 101
6.1.9 appl_ccmFwUpdateStart 102
6.1.10 appl_ccmFwUpdateExecute 102
6.1.11 appl_ccmEcatSsiUpdate 103
6.1.12 appl_ccmFoeUpdateSettings 104
6.1.13 appl_ccmEthMacAddressSet 105
6.1.14 appl_ccmNetworkDefaultUp 106
6.1.15 appl_ccmNetworkEoEUp 106
6.1.16 appl_ccmCfgVarGet 107
6.1.17 appl_ccmCfgVarSet 108
6.1.18 appl_ccmCfgSave 108
6.2 设备检测 109
6.2.1 goal_ddInit - 在GOAL中注册GOAL dd API (appl_init) 109
6.2.2 goal_ddNew - 在GOAL中注册GOAL dd API (appl_setup) 109
6.2.3 goal_ddCustomerIdSet - 配置GOAL dd实例的客户ID 111
6.2.4 goal_ddModuleNameSet - 配置GOAL dd实例的名称 112
6.2.5 goal_ddFeaturesSet - 配置GOAL dd实例的功能 113
6.2.6 goal_ddCallbackReg - 配置GOAL dd实例的回调 113
6.2.7 goal_ddSessionFeatureActivate - GOAL dd实例功能的临时激活 115
6.2.8 goal_ddFilterAdd - CM变量的限制访问 116
6.2.9 筛选项定义 - GOAL_DD_ACCESS_FILTER_SET_ALL 116
6.2.10 筛选项定义 – GOAL_DD_ACCESS_FILTER_SET_BASIC 117
6.2.11 筛选项定义 – GOAL_DD_ACCESS_FILTER_SET_HIDDEN 118
6.3 PROFINET堆栈应用程序编程接口 119
6.3.1 goal_pnioInit - 在GOAL中注册GOAL PROFINET (appl_init) 119
6.3.2 goal_pnioNew - 创建 GOAL PROFINET 实例 (appl_setup) 119
6.3.3 goal_pnioCfgDcpFactoryResetDisableSet - 配置DCP恢复出厂设置 120
6.3.4 goal_pnioCfgDcpAcceptMixcaseStationSet - 配置DCP MixcaseStationname接受 120
6.3.5 goal_pnioCfgDevDapSimpleSet - 配置设备DAP简单模式 121
6.3.6 goal_pnioCfgDevDapApiSet - 设置设备DAP API号 121
6.3.7 goal_pnioCfgDevDapSlotSet - 设置设备DAP slot号 122
6.3.8 goal_pnioCfgDevDapSubslotSet - 设置设备DAP subslot号 122
6.3.9 goal_pnioCfgDevDapModuleSet - 设置设备DAP module id 122

6.3.10 goal_pnioCfgDevDapSubmoduleSet - 设置设备DAP子 module id 123
6.3.11 goal_pnioCfgNetLinkSafetySet - 配置设备端口禁用行为 123
6.3.12 goal_pnioCfgNewIoDataCbSet - 配置新的IO数据回调 124
6.3.13 goal_pnioCfgDiagBufMaxCntSet - 配置最大诊断条目数 124
6.3.14 goal_pnioCfgDiagBufMaxDataSizeSet - 配置最大诊断数据大小 125
6.3.15 goal_pnioCfgIocrBlocksMaxSet – 配置最大IOCR块缓冲区 125
6.3.16 goal_pnioCfgCrMaxCntSet - 配置最大通信关系计数 125
6.3.17 goal_pnioCfgArMaxCntSet - 配置最大应用程序关系计数 126
6.3.18 goal_pnioCfgApiMaxCntSet - 配置最大API计数 126
6.3.19 goal_pnioCfgSlotMaxCntSet - 配置最大slot计数 126
6.3.20 goal_pnioCfgSubslotMaxCntSet - 配置最大subslot计数 127
6.3.21 goal_pnioCfgSubslotIfSet - 配置接口subslot 127
6.3.22 goal_pnioCfgSubslotPortSet - 配置端口subslot 127
6.3.23 goal_pnioCfgSnmpIdSet - 配置SNMP实例ID 128
6.3.24 goal_pnioSlotNew - 创建新slot 128
6.3.25 goal_pnioSubslotNew - 创建新slot 129
6.3.26 goal_pnioModNew - 创建新模块 129
6.3.27 goal_pnioSubmodNew - 创建新的子模块 130
6.3.28 goal_pnioModPlug - 将模块插入slot 130
6.3.29 goal_pnioSubmodPlug - 将Submodule插入Subslot以进cyclic传输 131
6.3.30 goal_pnioSubmodPlug - 将Submodule插入Subslot以进RPC传输 131
6.3.31 goal_pnioModPull - 从slot中拔出模块 132
6.3.32 goal_pnioSubmodPull - 从subslot中拔出子模块 133
6.3.33 goal_pnioDataOutputGet - 从子模块获取输出数据 133
6.3.34 goal_pnioDataInputSet - 设置子模块的输入数据 134
6.3.35 goal_pnioApduStatusGet - 获取应用程序协议数据单元状态 134
6.3.36 goal_pnioAlarmNotifySend - 发送报警通知 135
6.3.37 goal_pnioAlarmNotifySendAck - 发送报警通知确认 135
6.3.38 goal_pnioAlarmProcessSend - 发送过程报警 136
6.3.39 goal_pnioRecReadFinish - 回复记录读取请求 136
6.3.40 goal_pnioRecWriteFinish - 回复记录写入请求 137
6.3.41 goal_pnioDiagExtChanDiagAdd - 添加扩展通道诊断条目 137
6.3.42 goal_pnioDiagChanDiagRemove - 删除通道诊断条目 138
6.3.43 goal_pnioCyclicCtrl - 控制周期性数据接收回调 138
6.4 EtherNet/IP应用程序编程接口 139
6.4.1 goal_eipInit 139
6.4.2 goal_eipNew 139
6.4.3 goal_eipCipClassRegister 139
6.4.4 goal_eipCreateAssemblyObject 140
6.4.5 goal_eipCreateAssemblyObjectRpc 140
6.4.6 goal_eipAssemblyObjectGet 141
6.4.7 goal_eipAddExclusiveOwnerConnection 141
6.4.8 goal_eipAddInputOnlyConnection 142
6.4.9 goal_eipAddListenOnlyConnection 143
6.4.10 goal_eipGetVersion 143
6.4.11 goal_eipDeviceStatusSet 144
6.4.12 goal_eipDeviceStatusClear 144
6.4.13 goal_eipAssemblyObjectWrite 145
6.4.14 goal_eipAssemblyObjectRead 145
6.4.15 goal_eipIdentitySerialNumberSet 145
6.5 EtherCAT 应用程序设计接口 146
6.5.1 goal_ecatInit 146
6.5.2 goal_ecatNew 147

6.5.3 goal_ecatCfgEmergencyOn 148
6.5.4 goal_ecatCfgEmergencyQueueNum 148
6.5.5 goal_ecatCfgFoeOn 149
6.5.6 goal_ecatCfgExplDevIdOn 149
6.5.7 goal_ecatCfgBootstrapOn 150
6.5.8 goal_ecatCfgDcRequiredOn 150
6.5.9 goal_ecatCfgSizePdoStreamBufSet 151
6.5.10 配置指示灯模拟行为 151
6.5.11 goal_ecatObjAddrGet 152
6.5.12 goal_ecatObjValGet 153
6.5.13 goal_ecatObjValSet 154
6.5.14 goal_ecatdynOdObjAdd 155
6.5.15 goal_ecatdynOdSubIndexAdd 156
6.5.16 goal_ecatdynOdSubIndexRpcAdd 157
6.5.17 goal_ecatdynOdObjNameAdd 158
6.5.18 goal_ecatdynOdSubIndexNameAdd 158
6.5.19 goal_ecatdynOdFinish 159
6.5.20 goal_ecatEsmStateGet 159
6.5.21 goal_ecatEmcyReqWrite 160
6.5.22 goal_ecatGetVersion 161
6.5.23 goal_ecatEsmLocalErrorSet 161
6.6 网络 162
6.6.1 goal_netRpcInit - RPC功能网络初始化 162
6.6.2 goal_maNetOpen - 打开网络 163
6.6.3 goal_maNetClose - 关闭网络 163
6.6.4 goal_maNetGetById - 获取网络媒介适配器 (MA) 句柄 164
6.6.5 goal_maNetIpSet – 设置 IP 地址 164
6.7 TCP通道 165
6.7.1 goal_maChanTcpOpen - 打开 TCP 通道媒介适配器 (MA) 165
6.7.2 goal_maChanTcpNew - 创建新的 TCP 通道 165
6.7.3 goal_maChanTcpActive - 激活创建的 TCP 通道 166
6.7.4 goal_maChanTcpSetNonBlocking - 将通道设置为非阻塞 166
6.7.5 goal_maChanTcpGetRemoteAddr - 获取 TCP 通道的远程地址 167
6.7.6 goal_maChanTcpSend - 通过 TCP 通道发送数据 167
6.8 UDP通道 168
6.8.1 goal_maChanUdpOpen - 打开 UDP 通道MA 168
6.8.2 goal_maChanUdpGetById - 获取 UDP 通道 MA 句柄 168
6.8.3 goal_maChanUdpNew - 创建新的 UDP 通道 169
6.8.4 goal_maChanUdpClose - 关闭 UDP 通道 MA 169
6.8.5 goal_maChanUdpSetNonBlocking - 将打开的通道设置为非阻塞访问 170
6.8.6 goal_maChanUdpSetBroadcast - 将打开的 UDP 通道设置为广播操作 170
6.8.7 goal_maChanUdpGetRemoteAddr - 获取 UDP 通道的远程地址 171
6.8.8 goal_maChanUdpActivate - 激活 UDP 通道 171
6.8.9 goal_maChanUdpSend - 将数据发送到 UDP 通道 171

7. Web服务器 172
7.1 概述 172
7.2 配置 173
7.2.1 编译程序定义 173
7.2.2 CM变量 174
7.3 Web模板 179
7.3.1 CM变量 179
7.3.2 应用程序特定变量 179

7.3.3 列表 180
7.4 字符 181
7.5 回调函数 182
7.6 实现指南 183
7.6.1 上传网页 183
7.6.2 上传网页 184
7.6.3 读取应用程序特定的变量 185
7.6.4 读取列表 187
7.6.5 设置用户级别 189
7.6.6 下载文件 190

8. 故障检修 192
8.1 启动问题 192
8.2 连接问题 192
8.3 IP配置 193
8.4 降级到版本 1.0 193

1. 简介

本手册介绍各种应用程序编程接口(API)功能，这些功能旨在简化可作为双端口设备使用的 R-IN32M3 Module 的应用软件设计。
R-IN32M3 Module 的设计目的是加快工业通信应用程序的开发。R-IN32M3 Module 包括用作通信控制器(CC)的微控制器。CC 及其固件（二进制软件）根据工业通信标准（例如 PROFINET、EtherNet/IP™、EtherCAT®和其他网络）进行网络通信。应用程序控制器(AC)是用户定义的微控制器。该 AC 包含一组软件模块，在应用程序特定的软件上运行，无需考虑目标工业网络通信协议的特定要求。它使用 Renesas 在源代码中提供的一组 API 与 CC 通信，以便于用户的应用软件集成。CC 和 AC 之间的通信采用“核对核”(C2C)协议。下图显示了控制器及其接口。

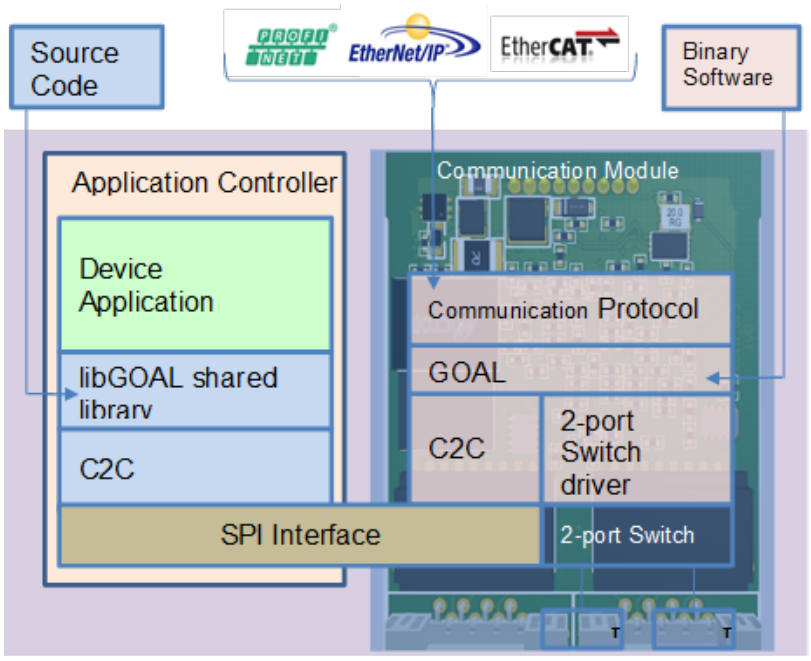

图 1.1 通信控制器与应用程序控制器之间的接口

GOAL 是一个通用开放抽象层，libGOAL 是源代码 API，用于从应用软件控制通信控制器。GOAL 和 C2C 二个组成部分，基于 OSAL 结合使用。

uGOAL 继承了 GOAL 但简化了它的配置并大大减少了使用的内存大小。

注：上图中提到的一些工业网络协议可能尚未发布。

1.1 通信协议特性

此功能是固件版本 2.1.0.0 或更高版本的条件。

PROFINET
- PROFINET IO Conformance Class B
- 最小循环时间 1 ms
- 最大 读/写 1434 Byte 处理数据(*1)
- 支持 LLDP, SNMP
- 支持 MRP
- 支持 Alarm Queue
- 支持 web browser, http

EtherNet/IP
- 最小循环时间 1 ms
- 最大连接大小 495 Bytes (*1)
- 支持 DLR (Device Level Ring)
- 支持 QoS
- 支持 ACD
- 支持 web browser and http

EtherCAT
- EtherCAT communication profile
  - CAN application protocol over EtherCAT (CoE)
  - Ethernet over EtherCAT (EoE)
  - File Access over EtherCAT (FoE)
- 支持 Explicit Device ID
- 支持; Free-Run, Sync Manager and Distributed Clocks (DC)
- 最大 PDO 1408 Byte (*1)

(*1) 循环过程数据的大小限制了通信周期

2. 文件结构

表 2.1 本文档内容

章节	内容
简介	介绍本文件
文件结构	本章节
设备架构	介绍 R-IN32M3 Module 的架构
应用程序	应用程序设计指南
通信栈	说明模块的通信接口
API	列出和说明 R-IN32M3 Module 的可用 API
示例	示例说明
故障检修	常见使用问题列表
目标	目标特定信息

3. 设备架构

3.1 架构

模块软件是由可用于工业通信领域应用构建的软件栈所构成，包含了一个领域特有的中间件(GOAL / uGOAL)。提供了以下软件栈：

- 设备检测： 简单的设备管理协议
- PROFINET： PROFINET 通信的通信栈
- EtherNet/IP： EtherNet/IP 通信的通信栈
- EtherCAT： EtherCAT 通信的通信栈
- Web 服务器： 用于应用程序特定管理和信息提供的简单 Web 服务器
- TCP/IP 栈： 提供 UDP 和 TCP 通道的 TCP/IP 栈

设备根据应用程序控制器(AC)使用。两个控制器（应用程序和通信控制器）周期性地通信，其中通信由应用程序控制器发出指令。应用程序控制器(AC)包含运行通信控制器(CC)服务以构建用户应用程序的应用程序。

3.2 接口

3.2.1 硬件接口

R-IN32M3 Module 引脚与电源和作为从属接口的 SPI 连接。

表 3.1 引脚说明

引脚	信号	I/O	说明
1	V_{cc}	—	3.3V ±0.15V DC 电源
2	GND	—	接地
3	/SS	I	从属选择。低电位有效，以启用从设备
4	/RESET	I	复位整个 R-IN32M3 Module 。低电位有效
5	MISO	O	主设备输入，从设备输出。数据从从设备传输到主设备
6	MOSI	I	主设备输出，从设备输入。数据从主设备传输到从设备
7	SCLK	I	串行时钟。主设备提供时钟，以转移数据
8	SYNC0	O	分布式时钟的 EtherCAT 同步信号
9	SYNC1	O	分布式时钟的 EtherCAT 同步信号

3.2.2 SPI 软件接口

R-IN32M3 Module 的软件接口包含多个层，可根据通信数据的要求提供通信通道。在 SPI 上，周期性地传输高达 128 字节的帧，其中通信由 AC 发起。

SPI 帧包含多个数据段，通常为来自通信栈的实时数据和源自 RPC（远程过程调用）的非实时数据。下文对通信栈进行了详细说明。

3.2.3 通信层/中间件的集成

所需通信协议的所有层和 AC 均通过搭载 GOAL / uGOAL 中间件实现。AC 需要 GOAL / uGOAL 中间件的搭载为基础。为了要利用 CC 的特定特征集（例如现场总线堆栈 PROFINET），需要搭载 RPC 功能的包装程序功能。这些包装程序需要 GOAL / uGOAL，因此目标平台必须支持 GOAL / uGOAL 中间件。

4. 应用程序

4.1 简介

本文件提供有关如何构建 R-IN32M3 Module 应用程序的信息。

4.2 硬件设置

模块提供一个管理接口，可在其中进行初始配置。这些属性可以永久性存储在设备中。为了配置 SPI 接口，提供了以下属性：

表 4.1SPI 配置

变量	说明	默认值
SPI_TYPE	SPI 主/从配置	1 = SLAVE
SPI_MODE	SPI 时序模式	0 = MODE0
SPI_UNITWIDTH	SPI 单一传输大小	0 = 8 bits
SPI_BITORDER	SPI 位传输方向	0 = MSB
SPI_TRANSFERSIZE	SPI 包传输大小	128

该模块仅支持
SPI_TYPE 从属,
SPI_UNITWIDTH 8 位,
SPI_TRANSFERSIZE 128 位,
SPI 位序 MSB。

SPI 模式可以根据表 4.2, SPI 模式配置。

表 4.2SPI 模式配置

值	SPI 模式
0	模式 0
1	模式 1
2	模式 2
3	模式 3

4.3 基本应用程序设置

基本应用程序由可选函数调用 appl_init、appl_setup 和 appl_loop 组成。它遵循 GOAL / uGOAL 中间件的设计原理。

4.4 默认功能

默认情况下，CC 将启动 Web 服务器（在端口 8080 上）,使用管理工具（Management Tool）来进行固件更新,并启动管理的设备检测实例。

AC 无法禁用 CC 上的 Web 服务器。但是，AC 可以启动 Web 服务器的另一个实例。

通过设备检测，AC 可通过不同的方式限制默认的完整管理访问。请参见本文件的相应部分。

4.5 功能

4.5.1 设备检测

4.5.1.1 初始状态

在通信控制器上，“设备检测”(DD)特性默认启用。这允许完全访问 CC 上的所有变量和日志。DD 基于不加密的，以 UDP 为基础的协议。

因此，应用程序可以而且应该将访问限制在可行范围内。可在不同的级别上实现对功能和访问的限制。

4.5.1.2 通过应用程序对功能进行初始禁用

在执行设置 AC 时，该属性应用于应用程序启动。

通过从 appl_setup()调用 goal_ddNew()，可以传递位掩码，其中位代表 DD 的单个函数。有关可能的值，请参见 DD 的 API 文档。

如果设置了所有位，则所有功能（hello、get、set、get_list、wink）都将启用。

4.5.1.3 通过 CM 变量对功能进行初始禁用

在启动应用程序之前，也可以通过设置相应的配置管理器 (CM) 变量 FEATURE_DISABLE（参见第 4.9.1 节）设置该机制（如第 4.5.1.2 节所述）。设备检测模块处理的每个请求将利用该变量的值确定是否可以处理请求。

请注意，如果传递了不同的初始化值，则 goal_ddNew 的任何调用将覆盖该变量的值。

4.5.1.4 临时启用设备检测功能

该属性在任意时间应用，例如使用 AC 提供的网站进行配置时。

```
1. GOAL_STATUS_T goal_ddSessionFeatureActivate(
2.     GOAL_DD_T *pHdlDd,                    /**< dd handle */
3.     uint32_t bitmaskFeatures              /**< bitmask with feature enable bits set */
4. );
```

- 参数位掩码包含代表正在启用的功能的位
- 这些位将从当前会话的CM覆盖写入无效位

4.5.1.5 筛选和访问权限

该属性在启动 AC 时应用。

使用筛选项，可以按组（模块 ID）和变量（变量 ID）限制对 CM 变量的访问。目前，已预先定义了一些使用筛选项。可使用以下函数启用这些筛选项：

```
1. GOAL_STATUS_T goal_ddFilterAdd(
2.     GOAL_DD_T *pHdlDd,                          /**< dd handle */
3.     GOAL_DD_ACCESS_FILTER_SET_T setId           /**< set id */
4. );
```

该函数作用于已创建的 DD 实例，因此需要传递有效的 DD 句柄。**setId** 可以与以下值一起使用：

表 4.3DD 筛选项 ID

设置 UD	说明
GOAL_DD_ACCESS_FILTER_SET_ALL	完全访问所有变量
GOAL_DD_ACCESS_FILTER_SET_BASIC	筛选管理工具的极小函数
GOAL_DD_ACCESS_FILTER_SET_HIDDEN	筛选隐藏信息

注： 有关详细信息，请参见 DD 的 API 文档。

4.5.2 PNIO

该通信控制器(CC)包含一个功能齐全的 PROFINET 从栈。有关其应用，请参见 5.4 通信栈 - PROFINET。

4.5.3 EtherNet/IP

该通信控制器(CC)包含一个功能齐全的 EtherNet/IP 从栈。有关其应用，请参见 5.6 通信栈 - EtherNet/IP。

4.5.4 EtherCAT

该通信控制器(CC)包含一个功能齐全的 EtherCAT 辅助栈。有关其应用，请参见 5.8 通信栈 - EtherCAT。

4.5.5 通用数据提供程序

对于现场总线通信应用程序，通用数据提供程序可用，其中包含应用程序的广义信息和 LED 状态。

表 4.4 MCTC 状态信息

数据提供程序信息	PROFINET	EtherNet/IP	EtherCAT
GOAL_MCTC_DP_STATUS_FLG_CONN	连接	连接	连接 (ESM-状态 == OP)
GOAL_MCTC_DP_STATUS_FLG_ERR	出错	出错	应用层出错
GOAL_MCTC_DP_STATUS_FLG_VALID	-	过程数据有效	
GOAL_MCTC_DP_LED_WINK	DCP 闪烁信号	-	
GOAL_MCTC_DP_LED_RED1	出错	模块状态：红色	EC ERR LED
GOAL_MCTC_DP_LED_RED2	维护	网络状态：红色	-
GOAL_MCTC_DP_LED_GREEN1	连接	模块状态：绿色	EC RUN LED
GOAL_MCTC_DP_LED_GREEN2	DCP 闪烁信号	网络状态：绿色	-

4.5.6 以太网启动延迟

在将 EtherCAT 整合入 2.0.0.0 版通信控制器(CC)后，便整合了一项附加功能。该功能允许通过应用程序控制器(AC)专门启动网络。根据所选通信栈（例如：PROFINET 或 EtherCAT），以规定模式引出网络接口。

但是，在该场景中，无应用程序控制器(AC)的通信控制器(CC)不会启动网络。因此，执行低效运行，默认会在通信控制器(CC)启动 5 秒后引出网络。在这种情况下，该模块仅支持 TCP/UDP。可以使用 ccm 变量（模块编号 72，变量编号 13）配置该行为。

下表显示了相关的配置选项：

表4.5 以太网超时配置

超时值	功能
0	默认超时启用（5秒）
1 ... 254	值决定超时（秒）
255	超时禁用

如果 AC 的加电以及 AC 和 CC 之间的通信建立超过配置的超时时间，则需相应增加时间。

4.6 外部复位

该模块提供外部复位输入，其中 AC 可以执行 R-IN32M3 Module 的复位。

注意：
在 CC 固件更新期间，请勿执行该复位。这将妨碍功能的正确更新。因此，建议仅在 AC 初始通电（冷启动）时使用该复位。

4.7 RPC 同步复位

4.7.1 通信控制器(CC)

默认情况下，该 CC 不会隐式执行复位。
必要时，CC 请求 AC 执行重启。如果先前启动和配置的 AC 连接到新启动的 CC，则会出现这种情况。

4.7.2 应用程序控制器(AC)

AC 根据 CC 的请求执行应用程序的复位，例如在 CC 重启时。在固件更新期间可能会出现此情况。
AC 可以使用硬件复位信号执行 CC 的复位。

注意：
在 CC 固件更新期间，请勿执行该复位。这将妨碍功能的正确更新。

4.8 IP 设置

IP 设置可通过 CM 接口完成。

表 4.6 使用 CM 配置 IP

步骤	操作	备注
1	设置 CM 变量 GOAL_ID_NET:IP	配置 IP 地址
2	设置 CM 变量 GOAL_ID_NET:NETMASK	配置网络掩码
3	设置 CM 变量 GOAL_ID_NET:GW	配置网关
4	设置 CM 变量 GOAL_ID_NET：有效：1	将 IP 配置设置为有效
5	将 CM 变量 GOAL_ID_NET:DHCP_ENABLED 设置为 0	禁用 DHCP
6	将 CM 变量 GOAL_ID_NET:COMMITGOAL_ID_NET:COMMIT 设置为 1	应用 IP 设置

注： 要启用 DHCP，将 CM 变量 GOAL_ID_NET:DHCP_ENABLED 设置为 1 并执行重启。

4.9 R-IN32M3 Module管理接口

配置 R-IN32M3 Module 的每个变量都有一个按模块 ID 分组的管理接口具体取决于每个用途。

每个管理接口（每个模块 ID）的表格如下，每个表格信息从 R-IN32M3 Module 固件版本 2.1.0.0 开始。

可以使用模块 ID 和变量 ID 从应用程序访问配置变量。

- 通过 appl_ccmCfgVarGet()函数读取值
- 通过 appl_ccmCfgVarSet()函数写入值
- 通过 appl_ccmCfgSave()函数将值永久保存到非易失性存储器

4.9.1 设备检测(DD)

module id = GOAL_ID_DD (34)

表 4.7 DD 管理接口

量名称	变量 ID	类型	最大大小	详细说明
MODULENAME	0	STRING	20	模块的客户特定名称
CUSTOMERID	1	UINT32	4	客户 ID
RESERVED	2	UINT8	1	-
FEATURE_DISABLE	3	UINT32	4	每个位禁用一个函数： 位0，禁用"HELLO DETECTION" 位1，禁用 WINK 位2，禁用 GETLIST 位3，禁用 GET VALUE 位4，禁用 SET VALUE

4.9.2 PNIO

module id = GOAL_ID_PNIO (27)

表 4.8 PNIO 管理接口

量名称	变量 ID	类型	最大大小	详细说明
STATION_NAME	0	GENERIC	255	—
VENDOR_ID	1	UINT16	2	—
DEVICE_ID	2	UINT16	2	—
VENDOR	3	GENERIC	255	—
IM0HWREV	4	UINT16	2	—
IM0SWREVPREF	5	UINT8	1	—
IM0SWREVENH	6	UINT8	1	—
IM0SWREVBUGFIX	7	UINT8	1	—
IM0SWREVINTCHG	8	UINT8	1	—
IM0SWREVCNT	9	UINT16	2	—
IM0PROFILEID	10	UINT16	2	—
IM0PROFILETYPE	11	UINT16	2	—
IM0ORDERID	12	GENERIC	(20+1)	—
IM0SERIALNR	13	GENERIC	(16+1)	—
IM1TAGFUNC	14	GENERIC	32	—
IM1TAGLOC	15	GENERIC	22	—
IM2DATA	16	GENERIC	16	—
IM3DESC	17	GENERIC	54	—
IM4SIG	18	GENERIC	54	—
HOLDFCT	19	UINT32	4	—
SYSCAP	20	UINT32	4	—
PORTDESC	21	GENERIC	LLDP_PORT_DESC_LEN	—
SYSDESC	22	GENERIC	LLDP_SYS_DESC_LEN	—
TXINTERV	23	UINT32	4	—
MANADDR	24	UINT32	4	—
SYSTEM_NAME	25	STRING	LLDT_SYS_NAME_LEN	—
IFSUBTYPE	26	UINT32	4	—
PDPORTDATA	27	GENERIC	sizeof(PN_REC_PDPORT DATA_CFT_T	—
FS_HELLO_MODE	28	UINT32	4	—
FS_HELLO_INTERVAL	29	UINT32	4	—
FS_HELLO_RETRY	30	UINT32	4	—
FS_HELLO_DELAY	31	UINT32	4	—

4.9.3 日志管理器(LM)

module id = GOAL_ID_LM (35)

表 4.9 LM 管理接口

变量名称	变量 ID	类型	最大大小	详细说明
VERSION	0	UINT8	1	LM 接口的版本信息
READBUFFER	1000	GENERIC	128	从设备读取联机日志的缓冲区
CNT	1001	UINT16	2	联机日志访问的控制字
EXLOG_READBUFFER	1002	GENERIC	128	从设备读取异常日志的缓冲区
EXLOG_CNT	1003	UINT16	2	异常日志访问的控制字
EXLOG_SIZE	1004	UINT32	4	异常日志大小指示器
EXLOG_USAGE	1005	UINT8	1	异常日志使用指示器
EXLOG_ERASE	1006	UINT8	1	命令： *，擦除异常日志

4.9.4 NET

module id = GOAL_ID_NET (12)

表 4.10 NET 管理接口

变量名称	变量 ID	类型	最大大小	详细说明
IP	0	IPV4	4	第一个接口的 IP 地址
NETMASK	1	IPV4	4	第一个接口的网络掩码
GW	2	IPV4	4	第一个接口的网关
VALID	3	UINT8	1	IP 地址的有效性： 0，存储的 IP 地址无效，接口设置源自系统的网络堆栈 1，存储的 IP 地址有效，将在设备启动时应用于接口
DHCP_ENABLED	4	UINT8	1	DHCP 启用： 0，DHCP 禁用 1，DHCP 启用
DHCP_STATE	5	UINT8	1	DHCP 状态： 0，DHCP 已初始化 1，DHCP 服务器选择 2，DHCP 请求配置 3，DHCP IP 地址绑定 4，DHCP 更新配置 5，DHCP 将 IP 地址重新绑定到接口
DNS0	6	IPV4	4	第一个接口的第一个 DNS 服务器
DNS1	7	IPV4	4	第一个接口的第二个 DNS 服务器
HOSTNAME	8	STRING	20	第一个接口的主机名
COMMIT	1000	UINT8	1	命令： *，应用 IP 设置

4.9.5 BOOT

module id = GOAL_ID_BOOT (37)

表 4.11 BOOT 管理接口

变量名称	变量 ID	类型	最大大小	详细说明
SIGNATURE	0	GENERIC	16	启动映像的签名
BLVERSION	1	STRING	16	引导加载程序版本
FWVERSION	2	STRING	16	固件版本
RESET_CAUSE	1000	UINT8	1	复位原因： 0，未指定 1，请求固件更新 2，监视器 3，需要固件提交 4，保留
IMAGE_NUMBER	1001	UINT8	1	启动映像编号
IMAGE_COUNTER	1002	UINT8	1	启动映像计数器

4.9.6 配置管理器(CM)

module id = GOAL_ID_CM (2)

表 4.12 CM 管理接口

变量名称	变量 ID	类型	最大大小	详细说明
VERSION	0	UINT32	4	CM 接口的版本信息
SAVE	1000	UINT8	1	命令： *，将 CM 保存到闪存

4.9.7 ETH

module id = GOAL_ID_ETH (4)

表 4.13 ETH 管理接口

变量名称	变量 ID	类型	最大大小	详细说明
MAC	0	GENERIC	6	—
LINK	1000	UINT32	4	接口的链路状态掩码
SPEED	1001	UINT32	4	接口的端口速度掩码
DUPLEX	1002	UINT32	4	接口的端口双工掩码
PORTCNT	1003	UINT32	4	接口数量

4.9.8 EIP

module id = GOAL_ID_EIP (23)

表 4.14 EIP 管理接口

变量名称	变量 ID	类型	最大大小	详细说明
INCARNATIONID	0	UINT32	4	用于创建跨电源周期的唯一连接 ID
DOMAIN	1	GENERIC	EIP_CM_DOMAIN_LEN	部分 TCP/IP 接口对象属性 5
HOST	2	GENERIC	EIP_CM_HOST_LEN	TCP/IP 接口对象属性 6
ENCAPTIMEOUT	3	UINT16	2	TCP/IP 接口对象属性 13
PORTCFG	4	GENERIC	sizeof(OPENER_PORT_CFG_T)	以太网链接对象属性 6 和 9（界面控制和管理状态）
QOS_VLAN	5	UINT8	1	QoS 对象属性 1
QOS_URGENT	6	UINT8	1	QoS 对象属性 4
QOS_SCHEDULED	7	UINT8	1	QoS 对象属性 5
QOS_HIGH	8	UINT8	1	QoS 对象属性 6
QOS_LOW	9	UINT8	1	QoS 对象属性 7
QOS_EXPLICIT	10	UINT8	1	QoS 对象属性 8
TTL	11	UINT8	1	TCP/IP 接口对象属性 8 多播消息的 IP 报头的 TTL 值
MC_CTRL	12	UINT8	1	TCP/IP 接口对象属性 9 选择如何分配多播地址 0：从设备的 IP 地址计算地址 1：使用 Mcast 起始地址 + 偏移量
MC_NUM	13	UINT16	2	TCP/IP 接口对象属性 9 为 EtherNet/IP 分配的多播地址数
MC_START	14	UINT32	4	TIP/IP 接口对象属性 9 首个已分配的多播地址
ACD_ENABLED	15	UINT8	1	ACD 启用： 0，ACD 禁用 1，ACD 启用
ACD_LAST_CONFLICT	16	GENERIC	8	导致最后一次地址冲突的设备的 MAC

4.9.9 HTTP

module id = GOAL_ID_HTTP (25)

表 4.15 HTTP 管理接口

变量名称	变量 ID	类型	最大大小	详细说明
HTTP_CHANNELS_MAX	0	UINT16	2	确定到 HTTP 服务器的可能连接数目
HTTPS_CHANNELS_MAX	1	UINT16	2	确定到 HTTPS 服务器的可能连接数目
USERLEVEL0	2	STRING	32	0级身份验证数据
USERLEVEL1	3	STRING	32	1级身份验证数据
USERLEVEL2	4	STRING	32	2级身份验证数据
USERLEVEL3	5	STRING	32	3级身份验证数据

4.9.10 CCM

module id = GOAL_ID_CCM (72)

表 4.16 R-IN32M3 Module 管理接口

变量名称	变量 ID	类型	最大大小	详细说明
SPI_TYPE	0	UINT8	1	SPI 类型（目前仅支持从设备）： 0，SPI 主设备 1，SPI 从设备
SPI_MODE	1	UINT8	1	SPI 模式： 0, CPOL=0; CPHA=0 1, CPOL=0; CPHA=1 2, CPOL=1; CPHA=0 3, CPOL=1; CPHA=1
SPI_SPEED	2	UINT8	1	主模式下的 SPI 速度
SPI_UNITWIDTH	3	UINT8	1	单个传输单元的位大小： 0，8位 1，16位 2，32位
SPI_BITORDER	4	UINT8	1	SPI 传输的位序： 0，MSB 第一 1，LSB 第一
SPI_TRANSFERSIZE	5	UINT16	2	单个传输帧的最小传输大小
COMM_FAULT_ERROR_STATE	6	UINT8	1	在周期性连接期间与 AC 失去通信时要执行的故障动作： 0，进入故障状态（禁用连接） 1，保持运行（保持连接）
COMM_SYNC_RESET	7	UINT8	1	从 AC 收到同步复位请求时的行为： 0，无动作 1，执行 CC 的复位
FW_UPDATE_COMMIT_DISABLE	8	UINT8	1	在固件更新期间可选禁用附加提交步骤： 0，固件更新需要提交步骤 1，固件更新不需要提交步骤
FOE_FILENAME	9	STRING	32	EtherCAT FoE 更新文件名
FOE_PASSWORD	10	UINT32	4	EtherCAT FoE 更新密码
FOE_UPDATE_REQUIRES_BOOT	11	UINT8	1	EtherCAT FoE 更新所需状态
FOE_FILENAME_MATCH_LEN	12	UINT8	1	EtherCAT FoE 更新文件名匹配长度
ETH_SWITCH_MODE_TIMEOUT	13	UINT8	1	以太网接口激活的一般超时
UPTIME	1000	UINT32	4	设备启动后的秒数

4.9.11 QUEUE

module id = GOAL_ID_QUEUE (42)

表 4.17 QUEUE 管理接口

变量名称	变量 ID	类型	最大大小	详细说明
SMALLBUFSIZE	0	UINT16	2	用于小内存缓冲区的缓冲区大小
SMALLBUFNUM	1	UINT16	2	保留的小内存缓冲区数量
MEDBUFSIZE	2	UINT16	2	用于中等内存缓冲区的缓冲区大小
MEDBUFNUM	3	UINT16	2	保留的中等内存缓冲区数量
BIGBUFSIZE	4	UINT16	2	用于大内存缓冲区的缓冲区大小
BIGBUFNUM	5	UINT16	2	保留的大内存缓冲区数量

4.9.12 SNMP

module id = GOAL_ID_SNMP (75)

表 4.18 SNMP 管理接口

变量名称	变量 ID	类型	最大大小	详细说明
MIB2_SYS_LOCATION	0	STRING	255	—
MIB2_SYS_CONTACT	1	STRING	255	—
MIB2_SYS_NAME	2	STRING	255	—

4.9.13 MCTC

module id = GOAL_ID_MCTC (94)

表 4.19 MCTC 管理接口

变量名称	变量 ID	类型	最大大小	详细说明
PRC_DISABLE	0	UINT8	4	RPC 模式： 0，启用 1，禁用
ID	1000	UINT32	4	实例 ID
STAT_RESET	1001	UINT32	4	MCTC 通信复位数
STAT_RPC_TIMEOUTS	1002	UINT32	4	MCTC RPC 超时的次数
STAT_RPC_DELAY_MIN	1003	UINT32	4	RPC 请求的最小时间
STAT_RPC_DELAY_MAX	1004	UINT32	4	RPC 请求的最大时间
STAT_RPC_DELAY_MEAN	1005	UINT32	4	RPC 请求的中位时间
STAT_RPC_COUNT	1006	UINT32	4	RPC 请求的次数

4.10 固件更新

4.10.1 更新通信控制器(CC)

可以在现场进行通信控制器(CC)的固件更新。使用 Management Tool 进行更新。该工具使用 http 协议将固件包传输到设备。

4.10.1.1 固件包

固件映像捆绑在一个包中。该固件包包含一个已签名的固件，保证仅从设备发起者接受固件。因此，设备可以检查固件映像的真实性。

4.10.1.2 控制接口

默认情况下，可以随时更新固件。通信控制器(CC)提供一个启用和禁用固件更新过程的接口。应用程序控制器(AC)应使用该接口，因为存在固件更新不被接受的某些情况。在与 PLC 的主动周期性连接过程中，应禁用该固件更新的接口。
有关该接口的使用，请参见第 6.1, 设备特定函数节。

作为可选方案，应用程序控制器(AC)可以就固件更新相关事件进行登记。之后，传输开始和结束以信令方式传至应用程序控制器(AC)。其任务亦会触发引导程序重启和执行实际固件更新。除此之外，将会向应用程序控制器(AC)报告故障或成功的固件更新事件。

可以使用 appl_ccmFwUpdateCbReg()，登记回调函数。

```
/*****************************************************************************/
/** firmware update callback
 *
 */
static void appl_fwUpdateCb(
    FW_UPDATE_SOURCE_T source,                  /**< fw update source */
    FW_UPDATE_STATUS_T state,                   /**< fw update status */
    uint8_t progress                            /**< fw update progress */
)
{
    switch (state) {
        default:
        case FW_UPDATE_IDLE:
            goal_logInfo("fw update state : IDLE");
            break;
        case FW_UPDATE_TRANSFER_INIT:
            goal_logInfo("fw update state : TRANSFER INIT");
            switch (source) {
                case FW_UPDATE_SOURCE_RPC:
                    goal_logInfo("fw update source : RPC");
                    break;
                case FW_UPDATE_SOURCE_HTTP:
                    goal_logInfo("fw update source : HTTP");
                    break;
                case FW_UPDATE_SOURCE_FOE:
                    goal_logInfo("fw update source : FOE");
                    break;
            }
            break;
        case FW_UPDATE_TRANSFER:
            goal_logInfo("fw update state : TRANSFER, progress = %d", progress);
            break;
        case FW_UPDATE_TRANSFER_DONE:
            goal_logInfo("fw update state : TRANSFER DONE");

            goal_logInfo("performing update, rebooting CCM module");
```

```
            /* perform actual update */
            appl_ccmFwUpdateExecute();
            break;
        case FW_UPDATE_ABORT:
            goal_logInfo("fw update state : ABORT");
            break;
        case FW_UPDATE_COMMIT_PENDING:
            goal_logInfo("fw update state : PENDING");
            break;
        case FW_UPDATE_COMMIT_DONE:
            goal_logInfo("fw update state : UPDATE DONE");
            break;
    }
}
```

4.10.1.3 固件更新顺序

固件更新过程包含两个步骤。首先，将固件上传到 CC 的 http 服务器。在该检查期间执行以下检查：

1. 该上传使用 0 级身份验证，其中会检查针对 HTTPD 模块配置的凭据。这些变量可通过 HTTPD 服务的 RPC 接口配置。默认情况下，这些凭据为空。因此，接受任何身份验证尝试。

2. AC 必须允许固件更新。默认情况下允许。但是，AC 可使用 RPC 接口禁用该函数。

如果在传输结束时检测到有效固件，则 CC 重新启动并进入引导加载程序。该 CC 检查固件的签名，并将正确签名的固件写入到设备内存。作为回退，先前运行的固件保留。
CC 重启后，需要与 Management Tool 建立成功的通信才能永久启用更新的固件。如果可能，CC 将再次重启，引导加载程序将新固件标记为当前固件。如果失败，引导加载程序将在下一次重启时还原为原始固件。

4.10.1.4 禁用DD时保留更新功能

应用程序集成器可能希望禁用 DD 功能，以免公开设备的任何内部信息。可以禁用。但是，需要执行以下步骤允许使用 Management Tool 进行固件更新：

1. 默认禁用 DD
调用 API 函数 goal_ddNew 时，可以传递一个确定有效 DD 功能的位掩码。如果使用预定义值 GOAL_DD_FEAAT_NO，则完全禁用 DD。
在内部，该值存储到 CM 变量 FEATURE_DISABLE，其状态也可以永久保存到闪存。

2. 引入交换机以临时启用 DD 功能
如果应处理固件更新，则需要临时设置最小数量的 DD 功能。
分别是：
- HELLO REQUEST
- SET CONFIG
- GET CONFIG
- SET IP

这可以使用具有以下参数的 API 函数 goal_ddSessionFeatureActivate()完成：

```
goal_ddSessionFeatureActivate(pHdlDd, GOAL_DD_FEAT_GETCONFIG |
                                      GOAL_DD_FEAT_SETCONFIG |
                                      GOAL_DD_FEAT_SETIP);
```

启用管理接口的临时交换机可以使用 Web 服务器实现。

因此，设备对 Management Tool 不可见。对于固件更新，临时激活所需的接口，从而允许更新通信控制器(CC)固件。

4.10.2 应用程序控制器(AC)的功能更新

默认情况下，应用程序控制器(AC)没有可用的固件更新功能。不过，该模块提供了在客户应用程序中实现此类功能的机制。
下文介绍了两种可能的解决方案。

4.10.2.1 通过 HTTP 更新 AC 固件

应用控制器(AC)可以利用 HTTPD 服务提供的 RPC 接口，通过 HTTP 传输提供基于网站或工具的固件更新。作为起点，HTTPD 示例 goal_http/02_post 可在 AC 上使用，示例程序 01_pnio_io_mirror 可在 CC 上使用。

对于后者，将在回调函数 httpDataCbPost 中处理使用 HTTP POST 请求传输的固件更新数据，例如，如下列示例代码中所示：

```
1.  /*****************************************************************************/
2.  /** goal http data callback
3.   *
4.   */
5.  static GOAL_STATUS_T httpDataCbPost(
6.      GOAL_HTTP_APPLCB_DATA_T *pCbInfo          /**< pointer to callback info struct */
7.  )
8.  {
9.      GOAL_STATUS_T res = GOAL_OK;              /* result */
10.     static uint32_t uploadDataLen = 0;        /* recent uploaded data length */
11.
12.     if (hdlUplHtml == pCbInfo->hdlRes) {
13.         /* check request method */
14.         switch (pCbInfo->reqType)
15.         {
16.         case GOAL_HTTP_FW_POST_START:
17.             /* reset data length */
18.             uploadDataLen = 0;
19.             GOAL_HTTP_RETURN_OK_204(pCbInfo);
20.             break;
21.
22.         case GOAL_HTTP_FW_POST_DATA:
23.
24.             /* process data */
25.             GOAL_MEMCPY(pDst, pCbInfo->cs.pData, pCbInfo->cs.lenData);
26.             uploadDataLen += pCbInfo->cs.lenData;
27.
28.             GOAL_HTTP_RETURN_OK_204(pCbInfo);
29.             break;
30.
31.         case GOAL_HTTP_FW_POST_END:
32.             GOAL_HTTP_RETURN_OK_204(pCbInfo);
33.             break;
34.
35.         case GOAL_HTTP_FW_REQ_DONE_OK:
36.         case GOAL_HTTP_FW_REQ_DONE_ERR:
37.             res = GOAL_OK;
38.             break;
39.
40.         default:
41.             /* return error */
42.             GOAL_HTTP_RETURN_ERR_403(pCbInfo);
43.             break;
44.         }
45.     }
46.     else {
47.         /* return error */
48.         GOAL_HTTP_RETURN_ERR_404(pCbInfo);
49.     }
50.     return res;
51. }
```

4.10.2.2 通过 TCP 更新 AC 固件

```
/*****************************************************************************/
/** TCP Server Callback
 *
 * Process TCP data stream segments
 */
static void tcpCallback(
    GOAL_MA_CHAN_TCP_T *pMaTcpHdl,          /**< MA handle */
    GOAL_NET_CB_TYPE_T cbType,              /**< callback type */
    struct GOAL_NET_CHAN_T *pChan,          /**< channel descriptor */
    struct GOAL_BUFFER_T *pBuf              /**< GOAL buffer */
)
{
    GOAL_NET_ADDR_T remote;                 /* remote address */
    GOAL_STATUS_T res;                      /* result */

    if (cbType == GOAL_NET_CB_NEW_SOCKET) {
        goal_logInfo("new TCP listener");
    }
    else if (cbType == GOAL_NET_CB_NEW_DATA) {

        /* process data */

        goal_logDbg("Data received on tcp socket %p", (void *) pChan);
    }
}
```

5. 通信栈

5.1 简介

本节列出了在配置通信栈时使用的所有函数。在调用 goal_New()之前，必须在 appl_setup()中调用这些函数。否则，这些函数将不起作用。每个配置有一个默认值，如果未设置其他值，则将应用该默认值。

5.2 SPI数据交换

与 R-IN32M3 Module 的通信使用串行外围接口(SPI)。传输必须始终通过应用程序核心(AC)触发，在 Heartbeat 心跳超时期间必须至少触发一次。在每次传输后，R-IN32M3 Module 中的实现会更新 SPI 数据，以确保不会中断正在运行的传输。

5.2.1 时钟域和通信周期

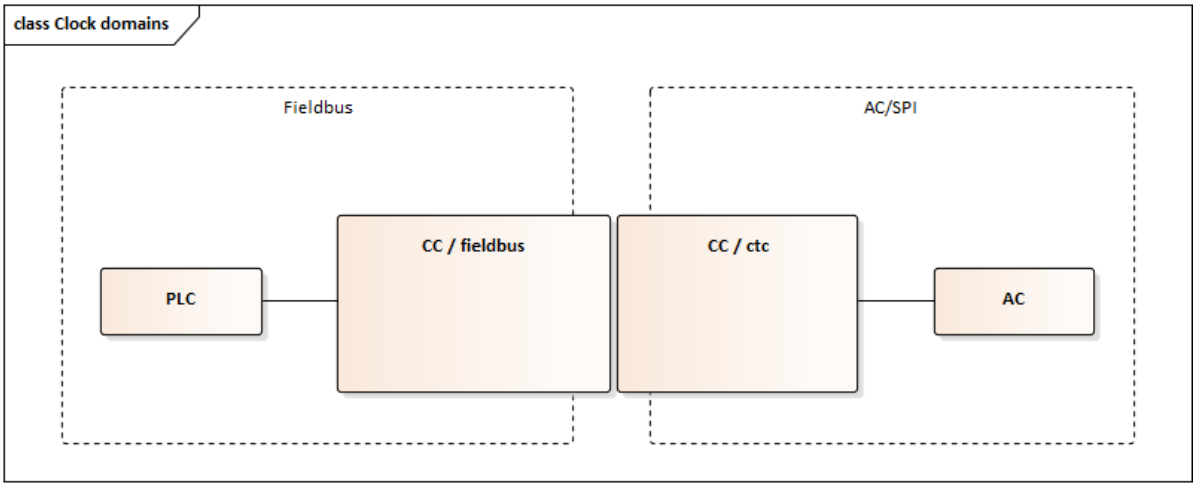

图 5.1ctc 时钟域

设备的操作包括两个互相独立运行的时钟域。第一个时钟域在现场总线侧。通常，PLC 在一个时钟域内与设备交互，其中 PLC 控制输出数据的时序。第二个时钟域由 AC 驱动，AC 通过启动 SPI 周期从设备读取输出数据，并更新下一个现场总线周期的输入数据。因此，过程数据的特定时序如图 5.2, 通信周期所示设置。

进行以下数据传输：

1. 来自 PLC 的新输出数据
2. 处理 CC 中的输出数据，预先加载 SPI 传输缓冲区
3. 完成基于 AC 的 SPI 传输执行后，AC 和 CC 之间的数据交换
4. 预先加载新的输入数据，进行下一次 SPI 传输
5. 完成的连续 SPI 传输执行后，AC 和 CC 之间的数据交换，从而为下一个现场总线传输提供新的输入数据

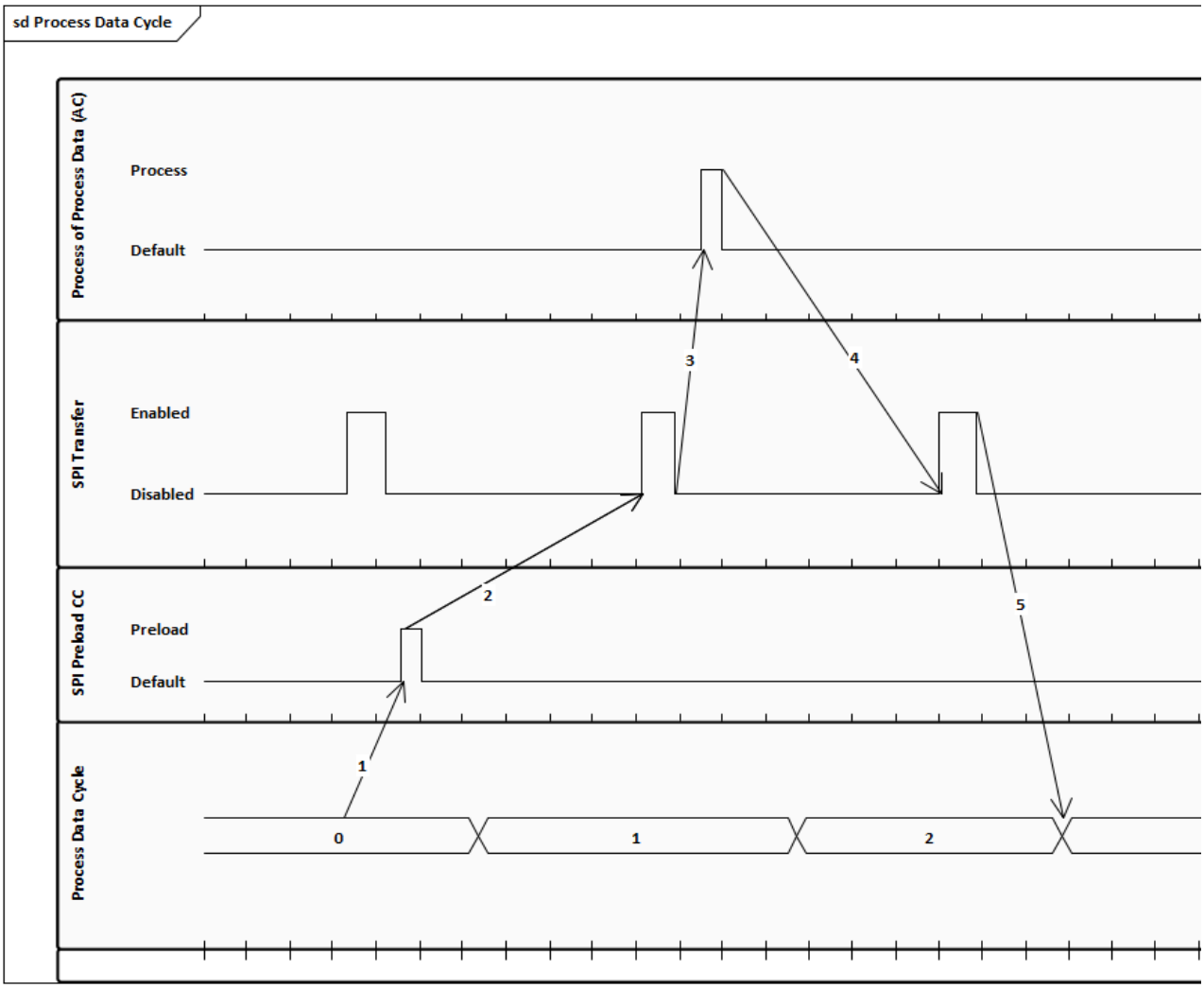

图 5.2 通信周期

5.2.2 技术数据

- 传输长度：
  - 仅周期性：72 字节
  - 周期性 + RPC 数据：128 字节
- 波特率：最小， ...，最大
- SPI 传输之间的延迟：0.5 ms ...心跳超时(1 s)
- 最短往返时间：4 ...6 ms（取决于使用的协议和设置）

5.2.3 SPI时序

下列简图显示了应用程序控制器(AC)必须考虑的基本 SPI 时序。

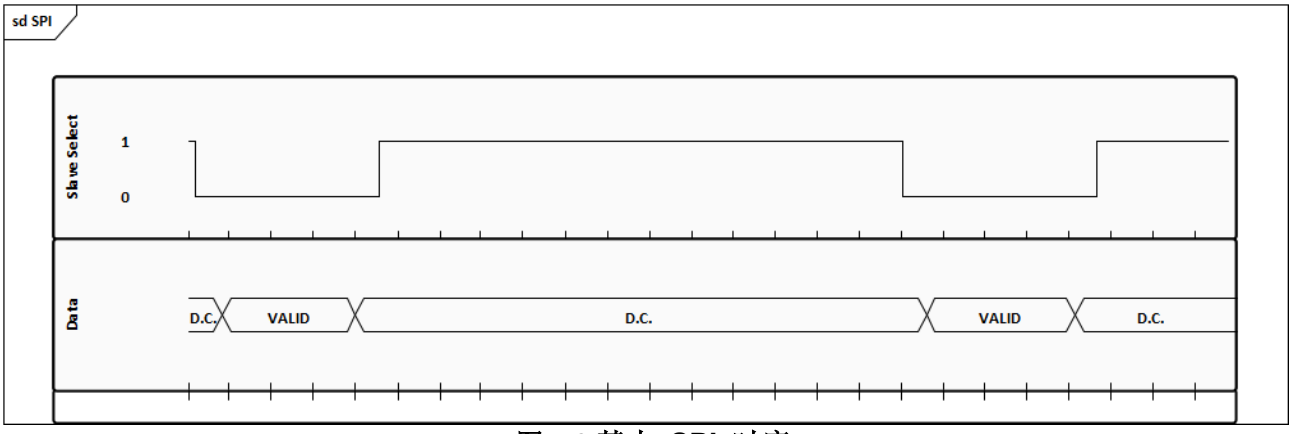

图 5.3 基本 SPI 时序

5.2.3.1 SPI速度

通信控制器 （CC） 的默认 SPI 速度为 2MHz。

5.2.3.2 SPI设置时序

图5.3, 基本 SPI 时序中显示了 AC 与 CC 之间的通信方案。在通信过程中，AC 需考虑以下时间。

SPI 设置时间 ..使用从属选择信号激活模块与第一个数据之间的时间

在 CC 通过 SPI 接受数据之前，SPI 设置时间必须至少为 10 ns。

5.2.3.3 SPI周期时间

该时间是两次连续 SPI 传输之间的距离。在两次传输之间，必须保持至少 250 us 的时间，以确保在 CC 内部正确处理。

5.2.4 SPI帧结构

SPI 帧必须包含表 5.1, SPI 帧结构中的结构，才能被 R-IN32M3 Module 所接受。

表 5.1SPI 帧结构

字节 0..1	**字节 2**	**字节 3**	**字节 4 ..76**	**字节 77 ..127**
偏移量为 0x0007 的 Fletcher-16 校验和（小端模式）	序列	数据长度	周期性数据	RPC 数据

相同的布局由包含本地序列计数器的设备发回。将跟踪序列计数器。如果序列计数器在心跳超时期间没有变化，通信将停止直到序列再次更新。

5.2.4.1 Fletcher 16校验和（16位）

计算时，可以使用 Fletcher-16 校验和。
起始索引是字节 4，结束索引是 127。如果整个区域被设置为零，则在计算后需要添加值 0x0007 以避免误报。在帧中，值的宽度为 16 位，需要具有小端编码。

5.2.4.2 序列计数器（8位）

序列计数器必须在每个发出的帧上递增，以便被识别为新数据。可以从任何 8 位值开始。

5.2.4.3 数据长度（8位）

如果仅传输周期性数据，则数据长度可以设置为 0 到 73 字节。通过 SPI 使用 RPC 时，数据长度需要达到固定值 124。

5.3 远程过程调用(RPC)

R-IN32M3 Module 的 C2C 实现所使用的 RPC 协议以字节 77 – 127 传输。将始终确认 RPC 传输，以尽量减少错误传输造成的数据丢失。此外，RPC 调用可能大于可用的 50 字节，因为 R-IN32M3 Module 在内部将每个接收到的 RPC 帧存储在环形缓冲区中，并等待分区传输完成后再继续处理请求。

5.3.1 RPC帧

5.3.1.1 结构

RPC 帧必须包含“表 5.2, RPC 帧结构”中的结构，才能被 R-IN32M3 Module 所接受。

表 5.2RPC 帧结构

字节 0..1	字节 2	字节 3	字节 4	字节 5	字节 6 ..49
偏移量为0x0007的Fletcher-16校验和 （小端模式）	本地 序列	远程 序列确认	数据 长度	标志	数据

每次发送新的本地序列时，R-IN32M3 Module 将在第二个传输的帧中响应相应的确认。如果没有收到确认，AC 可以重新传输其帧，以重新请求确认。

5.3.1.2 Fletcher-16校验和（16位）

计算时，可以使用 Fletcher-16 校验和。
起始索引是字节 6，结束索引是包含的数据长度。如果整个区域被设置为零，则在计算后需要添加值 0x0007 以避免误报。在帧中，值的宽度为 16 位，需要具有小端编码。

5.3.1.3 本地序列（8位）

对于每个数据已更改的 RPC 帧，序列计数器必须递增。对于每个不可见的增量帧，R-IN32M3 Module 将传输的数据放入其内部环形缓冲区，并在长度匹配后处理 RPC 调用。

5.3.1.4 远程序列确认（8位）

对于每个接收到的已处理 RPC 帧，AC 需要发送确认。如果接收到的帧与预期序列不匹配，AC 必须保留前一序列上的确认，以触发来自 R-IN32M3 Module 的重新发送。如果重新发送失败，AC 也可以执行重新同步。

5.3.1.5 数据长度（8位）

包含 RPC 数据的长度，必须介于 0（仅确认帧）和 44 之间。

5.3.2 标志（8位）

5.3.2.1 结构

表 5.3RPC 头标志

位 0	位 1	位 2	位 3
同步请求	同步确认	保留	请求确认

5.3.2.2 同步请求

如果设置为 1，R-IN32M3 Module 将进入 RPC 同步。

5.3.2.3 同步确认

在同步请求期间，两侧都需要设置同步确认标志。有关详细信息，请参见 5.3.3, RPC 同步。

5.3.2.4 请求确认

从合作伙伴设备强制确认。

5.3.3 RPC同步

在 R-IN32M3 Module 准备好运行之前以及在同步丢失之后，必须将 RPC 同步。通过将同步请求标志设置为 1 来触发。

表 5.4RPC 同步

AC	R-IN32M3 Module
同步请求 = 1 本地序列 = 0 本地序列确认 = 0 远程序列确认 = 0	
	同步请求 = 1 本地序列 = 0 本地序列确认 = 0 远程序列确认 = 0
同步请求 = 0 同步确认 = 1 本地序列 = 1 RPC_发送(<空>)	
	RPC_接收() →远程序列确认 = 1 同步请求 = 0 同步确认 = 1 本地序列 = 1 RPC_发送(<空>)

RPC_接收() →本地序列确认 = 1 →远程序列确认 = 1 同步确认= 0 RPC_启动()	
	RPC_接收() →本地序列确认 = 1 RPC_启动()

5.3.4 RPC协议

5.3.4.1 简介

GOAL RPC 协议使用虚拟出栈/入栈调用包含参数的远程 API 函数。每次调用都必须有一个返回值，该值通常在调用成功时设置为 GOAL_OK。

5.3.5 RPC请求/响应

5.3.5.1 结构

RPC 请求/响应由“表 5.5、RPC 请求结构”和“表 5.6RPC 响应结构”所示的部分组成。

表 **5.5RPC 请求结构**

字节 0..1	**字节 2..5**	**字节 6..9**	**字节 10..13**	**字节 14**	**字节 15**	**字节 16..x**	**字节 (x+1)..(x+3)**
静态标识符 0xaa, 0xee	数据长度 从字节 6 到 x	函数 ID （小端模式）	RPC ID （小端模式）	CTC ID	标志	数据	Fletcher-16 校验和，偏移量为 0x0007 （小端模式）

表 **5.6RPC 响应结构**

字节 0..1	**字节 2..5**	**字节 6 .. x**	**字节 x+1**	**字节 x+2**	**字节 (x+3)..(x+4)**
静态标识符 0xaa, 0xee	数据长度 从字节 6 到标志（含）	响应数据	CTC ID	标志	Fletcher-16 校验和，偏移量为 0x0007 （小端模式）

5.3.5.2 静态标识符

在 R-IN32M3 Module 上使用 2 字节静态标识符检测新 RPC 请求或响应的开始。如果标识符也包

含在数据字节中，Fletcher-16 校验和将确保不会按照与 RPC 请求/响应相同的方式作为线程处理。

5.3.5.3 数据长度

数据长度包含从“函数 ID”开始到最后一个数据字节结束的字节数。

5.3.5.4 RPC ID

RPC ID 是模块或组 ID。例如，GOAL PROFINET 使用 RPC IDGOAL_ID_PNIO 注册其自身调用。

5.3.5.5 函数ID

函数 ID 用作子 ID，用于将特定的函数调用映射到其处理程序。

5.3.5.6 CTC ID

CTC ID 用于将请求和响应与其特定的 MCTC 内部句柄相匹配。响应必须使用与请求相同的 CTC ID。

5.3.5.7 标志

标志定义请求的类型，详细信息请参见表 5.7, RPC 请求/响应标志。

表 5.7RPC 请求/响应标志

位 **0..1**
0 – 响应
1 – 请求
2 – 信息（请求，而不等待响应）

5.3.5.8 数据

数据部分包含 RPC 请求/响应的虚拟栈。

5.3.5.9 Fletcher-16校验和（16位）

计算时，可以使用 Fletcher-16 校验和。
起始索引是字节 16，结束索引是包含的数据长度。如果整个区域被设置为零，则在计算后需要添加值 0x0007 以避免误报。在帧中，值的宽度为 16 位，需要具有小端编码。

5.4 通信栈 – PROFINET

本节列出 GOAL PROFINET 提供的 API 函数。

5.4.1 goal_pnioCfgVendorIdSet – 设置供应商ID

配置供应商 ID。默认值：0x02c7 (Renesas Electronics)；返回 GOAL_STATUS_T 状态。

该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

表 5.8goal_pnioCfgVendorIdSet 参数

参数	说明
uint16_t idVendor	供应商 ID

5.4.2 goal_pnioCfgVendorId – 设置设备 ID

配置设备 ID。默认值：0x0300；返回 GOAL_STATUS_T 状态。
该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

表 5.9goal_pnioCfgVendorIdSet 参数

参数	说明
uint16_t idDevice	设备 ID

5.4.3 goal_pnioCfgVendorNameSet - 设置供应商名称

配置供应商名称。返回 GOAL_STATUS_T 状态。
该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

表 5.10goal_pnioCfgVendorNameSet 参数

参数	说明
const char *strVendor	供应商名称

5.4.4 goal_pnioCfgPortDescSet - 设置LLDP端口描述

配置 LLDP 端口描述。默认值：“TestPort”；返回 GOAL_STATUS_T 状态。该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

表 5.11goal_pnioCfgPortDescSet 参数

参数	说明
const char *strDescPort	端口描述

5.4.5 goal_pnioCfgSystemDescSet - 设置LLDP系统描述

配置 LLDP 系统描述。默认值：“PROFINET System”；返回 GOAL_STATUS_T 状态。
该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

表 5.12goal_pnioCfgSystemDescSet 参数

参数	说明
const char *strSystem	系统描述

5.4.6 goal_pnioCfgOrderIdSet - 设置订单ID

配置订单 ID。返回 GOAL_STATUS_T 状态。
该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

表 5.13goal_pnioCfgOrderIdSet 参数

参数	说明
const char *strOrder	订单 ID

5.4.7 goal_pnioCfgSerialNumSet - 设置序列号

配置序列号。返回 GOAL_STATUS_T 状态。
该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

表 5.14goal_pnioCfgSerialNumSet 参数

参数	说明
const char *strNumSerial	序列号

5.4.8 goal_pnioCfgHwRevSet - 设置硬件版本

配置硬件版本；返回 GOAL_STATUS_T 状态。
该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

表 5.15goal_pnioCfgHwRevSet 参数

参数	说明
uint16_t idRevHw	硬件版本

5.4.9 goal_pnioCfgSwRevPrefixSet - 设置软件版本前缀

配置软件版本前缀。默认值：“P”

- “V” - 正式
- “R” - 修订
- “P” - 原型
- “U” - 测试中
- “T” - 试验设备

返回 GOAL_STATUS_T 状态。
该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

表 5.16goal_pnioCfgSwRevPrefixSet 参数

参数	说明
const char chrRevSwPrefix	软件版本前缀

有关演示，请参见示例 15_config_set。

5.4.10 pnioCfgSwRevFuncEnhSet - 设置软件版本功能增强

配置软件版本功能增强。默认值：0x50
返回 GOAL_STATUS_T 状态。
该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

表 5.17goal_pnioCfgSwRevFuncEnhSet 参数

参数	说明
uint8_t idRevSwFuncEnh	软件版本功能增强

5.4.11 goal_pnioCfgSwRevBugfixSet - 设置软件版本错误修复

配置软件版本错误修复。默认值：3
返回 GOAL_STATUS_T 状态。
该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

表 5.18goal_pnioCfgSwRevBugfixSet 参数

参数	说明
uint8_t idRevSwBugfix	软件版本错误修复

5.4.12 goal_pnioCfgSwRevIntChgSet - 设置软件版本内部更改

配置软件版本内部更改。默认值：0x18
返回 GOAL_STATUS_T 状态。
该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

表 5.19goal_pnioCfgSwRevIntChgSet 参数

参数	说明
uint8_t idRevSwIntChg	软件版本内部更改

5.4.13 goal_pnioCfgSwRevCntSet - 设置软件版本计数器

配置软件版本计数器。默认值：0x0000
返回 GOAL_STATUS_T 状态。
该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

表 5.20goal_pnioCfgSwRevCntSet 参数

参数	说明
uint16_t idRevSwRevCnt	软件版本计数器

5.4.14 goal_pnioCfgIm1TagFuncSet - 设置I&M1标签函数

配置 I&M1 标签函数。默认值：“ ”
返回 GOAL_STATUS_T 状态。
该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

表 5.21 goal_pnioCfgIm1TagFuncSet 参数

参数	说明
const char *strIm1TagFunc	I&M1 标签函数

5.4.15 goal_pnioCfgIm1TagLocSet - 设置I&M1标签位置

配置 I&M1 标签位置。默认值：“ ”
返回 GOAL_STATUS_T 状态。
该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

表 5.22 goal_pnioCfgIm1TagLocSet 参数

参数	说明
const char *strIm1TagLoc	I&M1 标签位置

5.4.16 goal_pnioCfgIm2DateSet - 设置I&M2日期

配置 I&M2 日期。默认值：“ ”
返回 GOAL_STATUS_T 状态。
该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

表 5.23 goal_pnioCfgIm2DateSet 参数

参数	说明
const char *strIm2Date	I&M2 日期

5.4.17 goal_pnioCfgIm3DescSet - 设置I&M3说明

配置 I&M3 说明。默认值：“ ”
返回 GOAL_STATUS_T 状态。
该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

表 5.24 goal_pnioCfgIm3DescSet 参数

参数	说明
const char *strIm3Desc	**I&M3 说明**

5.4.18 goal_pnioCfgIm4SigSet - 设置I&M4签名（功能安全）

配置 I&M4 签名。默认值：“ ”
返回 GOAL_STATUS_T 状态。
该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

表 5.25 goal_pnioCfgIm4SigSet 参数

参数	说明
const char *strIm4Sig	I&M4 签名

5.4.19 goal_pnioCfgLldpOrgExtSet - 配置特定于LLDP组织的扩展

配置特定于 LLDP 组织的扩展。
默认值：GOAL_TRUE；返回 GOAL_STATUS_T 状态。
该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。
警告：不应使用该函数，否则可能会破坏一致性。

表 5.26goal_pnioCfgLldpOrgExtSet 参数

参数	说明
GOAL_BOOL_T flgLldpOrgExt	特定于 LLDP 组织的扩展标志

5.4.20 goal_pnioCfgLldpOptTlvSet - 配置LLDP可选TLV参数

配置 LLDP 可选 TLV 参数。默认值：GOAL_TRUE；这些参数包括：

- 端口描述
- 系统名称
- 系统描述
- 系统能力
- 管理地址
- 对象 ID

返回 GOAL_STATUS_T 状态。
该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。
警告：不应使用该函数，否则可能会破坏一致性。

表 5.27goal_pnioCfgLldpOptTlvSet 参数

参数	说明
GOAL_BOOL_T flgLldpOptTlv	LLDP 可选 TLV 参数标志

5.4.21 goal_pnioCfgLldpGenMacSet - 配置LLDP端口MAC地址生成

配置自动 LLDP 端口 MAC 地址生成。如果设置为 GOAL_TRUE，则通过向主机端口 MAC 地址添加端口 ID 自动生成 LLDP 端口特定的 MAC 地址。
默认值：GOAL_TRUE；返回 GOAL_STATUS_T 状态。
该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。
警告：如果启用该功能（默认），则每个设备至少按升序使用两个 MAC 地址（每个端口为实际数目加一）。要使用特定的虚拟端口 MAC 地址，必须禁用该功能，并且驱动程序必须为每个请求提供不同的 MAC 地址（GOAL_ETH_PORT_HOST 和端口特定请求）。

表 5.28goal_pnioCfgLldpGenMacSet 参数

参数	说明
GOAL_BOOL_T flgLldpGenMac	自动 LLDP MAC 生成标志

5.4.22 goal_pnioCfgIm14SupportSet - 配置I&M 1-4支持

配置 GOAL PROFINET 堆栈 I&M 1-4 支持。如果设置为 GOAL_FALSE，堆栈将拒绝 I&M 1-4 请求。默认值：GOAL_TRUE；返回 GOAL_STATUS_T 状态。
该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。
警告：应小心使用该函数，因为可能会破坏一致性。

表 5.29 goal_pnioCfgIm14SupportSet 参数

参数	说明
GOAL_BOOL_T flgIm14Support	I&M 1-4 支持标志

5.4.23 goal_pnioCfgIm14CbSet - 配置I&M 1-4回调

配置 GOAL PROFINET 堆栈 I&M 1-4 回调。如果设置为 GOAL_TRUE，则应用程序回调必须处理 I&M 1-4 获取和设置请求。默认值：GOAL_FALSE；返回 GOAL_STATUS_T 状态。
该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。
警告：应小心使用该函数，因为可能会破坏一致性。

表 5.30 goal_pnioCfgIm14CbSet 参数

参数	说明
GOAL_BOOL_T flgIm14Cb	I&M 1-4 回调标志

5.4.24 goal_pnioCfgIm0CbSet - 配置I&M 0回调

配置 GOAL PROFINET 堆栈 I&M 0 回调。如果设置为 GOAL_TRUE，则应用程序回调必须处理 I&M 0 获取和设置请求。
默认值：GOAL_FALSE；返回 GOAL_STATUS_T 状态。
该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。
警告：应小心使用该函数，因为可能会破坏一致性。

表 5.31 goal_pnioCfgIm0CbSet 参数

参数	说明
GOAL_BOOL_T flgIm0Cb	I&M 0 回调标志

5.4.25 goal_pnioCfgIm0FilterDataCbSet - 配置I&M 0筛选数据回调

配置 GOAL PROFINET 堆栈 I&M 0 筛选数据回调。如果设置为 GOAL_TRUE，则应用程序回调必须处理 I&M 0 筛选数据获取和设置请求。
默认值：GOAL_FALSE
返回 GOAL_STATUS_T 状态。
该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。
警告：应小心使用该函数，因为可能会破坏一致性。

表 5.32 goal_pnioCfgIm0FilterDataCbSet 参数

参数	说明
GOAL_BOOL_T flgIm0FilterCb	I&M 0 筛选数据回调标志

5.4.26 goal_pnioCfgRecDataBusyBufsizeSet - 配置记录句柄存储计数

配置并行记录句柄计数。
默认值：2；返回 GOAL_STATUS_T 状态。
该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

表 5.33goal_pnioCfgRecDataBusyBufsizeSet 参数

参数	说明
GOAL_BOOL_T cntRecDataBusyBufsize	记录句柄存储计数

5.4.27 goal_pnioCfgRpcFragReqLenMaxSet - 配置最大记录大小

配置记录请求的最大字节数。这必须与 GSDML 文件中的 MaxSupportedRecordSize 属性匹配。
默认值：4068；返回 GOAL_STATUS_T 状态。
该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。
警告：将该值更改为一个较小的数字可能会破坏一致性。

表 5.34goal_pnioCfgRpcFragReqLenMaxSet 参数

参数	说明
unsigned intsizeRpcFragMaxReqLen	最大记录大小

5.4.28 goal_pnioCfgRpcFragMaxCntSet – 配置最大RPC片段数

该函数已过时。GOAL PROFINET 堆栈现在允许 64 个片段帧。

5.4.29 goal_pnioCfgRpcFragEnableSet - 配置RPC碎片

配置 RPC 碎片功能。如果设置为 GOAL_TRUE RPC，则启用碎片。
默认值：GOAL_TRUE；返回 GOAL_STATUS_T 状态。
该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。
警告：不应使用该函数，否则可能会破坏一致性。

表 5.35goal_pnioCfgRpcFragEnableSet 参数

参数	说明
GOAL_BOOL_T flgRpcFragSupport	RPC 碎片启用标志

5.4.30 goal_pnioCfgRpcSessionMaxCntSet - 配置最大RPC会话计数

配置 RPC 会话的最大计数。默认值：8；返回 GOAL_STATUS_T 状态。
该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。
警告：不应使用该函数，否则可能会破坏一致性。

表 5.36goal_pnioCfgRpcSessionMaxCntSet 参数

参数	说明
unsigned intnumRpcSessions	RPC 会话计数

5.4.31 goal_pnioDmSubslotAdd – 映射subslot数据

API goal_pnioDmSubslotAdd 将给定 subslot 的输入和输出数据映射到 DM 的给定实例。

表 5.37goal_pnioDmSubslotAdd API 说明

参数	说明
GOAL_PNIO_T *pPnio	GOAL PROFINET实例
uint32_t idMiDmPeerFrom	从CC接收数据的句柄
uint32_t idMiDmPeerTo	向CC发送数据的句柄
GOAL_MI_DM_PART_T **ppPartDataOut	存储DM分区的输出指标
GOAL_MI_DM_PART_T **ppPartDataIn	存储DM分区的输出指标
uint32_t idApi	API ID
uint16_t idSlot	插槽ID
uint16_t idSubslot	subslot ID
uint32_t lenDataOut	输出数据长度
uint32_t lenDataIn	输入数据长度

5.4.32 goal_pnioDmSubslotIoxsAdd - 映射subslot IOCS/IOPS

API goal_pnioDmSubslotIoxsAdd 将给定 subslot 的 IOCS 和 IOPS 状态映射到 DM 的给定实例。

表 5.38goal_pnioDmSubslotIoxsAdd API 说明

参数	说明
GOAL_PNIO_T *pPnio	GOAL PROFINET 实例
uint32_t idMiDmPeerFrom	从 CC 接收数据的句柄
uint32_t idMiDmPeerTo	向 CC 发送数据的句柄
GOAL_MI_DM_PART_T **ppPartIocsOut	存储 DM 分区的输出指标
GOAL_MI_DM_PART_T **ppPartIopsOut	存储 DM 分区的输出指标
GOAL_MI_DM_PART_T **ppPartIocsIn	存储 DM 分区的输出指标
GOAL_MI_DM_PART_T **ppPartIopsIn	存储 DM 分区的输出指标
uint32_t idApi	API ID
uint16_t idSlot	slotID
uint16_t idSubslot	subslot ID

5.4.33 goal_pnioDmApduAdd – 映射APDU状态

API goal_pnioDmApduAdd 将 APDU 状态映射到 DM 的给定实例。

表 5.39goal_pnioDmApduAdd API 说明

参数	说明
GOAL_PNIO_T *pPnio	GOAL PROFINET 实例
uint32_t idMiDmPeerTo	（忽略）向 CC 发送数据的句柄
GOAL_MI_DM_PART_T **ppPartApduOut	存储 DM 分区的输出指标

5.4.34 goal_pnioDmDpAdd – 映射数据提供程序状态

API goal_pnioDmDpAdd 将数据提供程序状态映射到 DM 的给定实例。

表 5.40goal_pnioDmDpAdd API 说明

参数	说明
GOAL_PNIO_T *pPnio	GOAL PROFINET 实例
uint32_t idMiDmPeerTo	（忽略）向 CC 发送数据的句柄
GOAL_MI_DM_PART_T **ppPartDp	存储 DM 分区的输出指标

5.5 应用程序回调 - PROFINET

5.5.1 简介

为了充分利用 PROFINET，该堆栈允许在协议的几个阶段进行交互。例如，可以拒绝向 PLC 发送应用程序就绪信号，或处理特定 slot 的记录数据写入和读取。
使用回调系统前，应提供一个回调函数，使堆栈可以在每次希望与应用程序通信时调用该回调函数。回调函数必须具有以下原型：

```
GOAL_STATUS_T appl_pnioCb (
GOAL_PNIO_T *pPnio,                    /**< PROFINET handle */
GOAL_PNIO_CB_ID_T id,                  /**<callback id */
GOAL_PNIO_CB_DATA_T *pCb               /**<callback parameters */
);
```

通过调用 goal_pnioNew 注册，该调用将回调函数的指标作为第二个参数。

```
res = goal_pnioNew(&pPnio, APPL_PNIO_ID, appl_pnioCb);
```

现在每次调用 appl_pnioCb 时，GOAL_PNIO_CB_ID_T 值中都包含调用原因。GOAL_PNIO_CB_DATA_T 结构包含多个联合体的阵列，其中根据 GOAL_PNIO_CB_ID_T 值每个阵列元素具有特殊含义。
下文介绍有关可用回调 ID 的详细信息。

5.5.2 GOAL_PNIO_CB_ID_ALARM_ACK_TIMEOUT - 报警 ACK 的超时等待

指示等待报警确认时 APMS 中发生的超时。

表 5.41 超时 - 报警 ACK 的超时等待

参数	说明
cb->data[0].idAr	应用程序关系 ID
cb->data[1].u16	超时报警的序列号

5.5.3 GOAL_PNIO_CB_ID_ALARM_NOTIFY_ACK - 收到的报警通知 ACK

指示已收到报警通知 ACK。

表 5.42 收到的报警通知 ACK

参数	说明
cb->data[0].idAr	应用程序关系 ID
cb->data[1].u16	报警优先级（低、高）
cb->data[2].pAlarmNotifyAck	GOAL_PNIO_ALARM_NOTIFY_ACK_T 结构指标

5.5.4 GOAL_PNIO_CB_ID_ALARM_NOTIFY - 收到的报警通知

指示已收到报警通知。如果未处理回调或返回 GOAL_OK，PROFINET 堆栈将自动确认收到的报警通知。

表 5.43 收到的报警通知

参数	说明
cb->data[0].idAr	应用程序关系 ID
cb->data[1].u16	报警优先级（低、高）
cb->data[2].pAlarmNotify	GOAL_PNIO_ALARM_NOTIFY_T 结构指标
cb->data[3].u32	用户数据长度
cb->data[4].pCu8	用户数据指标

5.5.5 GOAL_PNIO_CB_ID_APPL_READY - 收到的应用程序就绪响应

指示已收到应用程序就绪响应。返回值必须是 GOAL_OK。

表 5.44 收到的应用程序就绪响应

参数	说明
cb->data[0].idAr	应用程序关系 ID

5.5.6 GOAL_PNIO_CB_ID_BLINK - 闪烁请求

指示已收到 DCP 信号请求，并使用必要的信息填充回调数据。

cb->data[0].stateDcpBlink 向应用程序表明初始信号请求（开始）、切换信号的正确时间（切换）以及信号相位结束的时间（完成）。如果应用程序希望完全表示信号本身，则可以使用该信息。

第二种可能是使用包含 GOAL_PNIO_DCP_LIGHT_OFF 或 GOAL_PNIO_DCP_LIGHT_ON 的 cb->data[1].stateDcpLight 数据，可用于将信号指示灯直接切换到 LED 或闪烁功能。
如果使用变量 1 或 2，则回调必须返回 GOAL_OK_SUPPORTED，否则堆栈将通过使用 GOAL_PNIO_LED_SIGNAL define 调用 goal_targetSetLeds，自行处理闪烁。对于堆栈内部处理，必须实现 goal_targetGetLeds 和 goal_targetSetLeds。

表 5.45 闪烁请求

参数	说明
cb->data[0].stateDcpBlink	DCP 闪烁状态（开始、切换、完成）
cb->data[1].stateDcpLight	DCP 灯状态（开、关）

返回值	说明
GOAL_OK	堆栈内部处理的信号
GOAL_OK_SUPPORTED	应用程序处理的信号

```
case GOAL_PNIO_CB_ID_BLINK:

#if FIRST_SCENARIO
    /*************************/
    /* first scenario: act on blink state */
    /*************************/
    swtich (cb->data[0].stateDcpBlink) {

        case GOAL_PNIO_DCP_BLINK_START:
            /* start blinking */
            break;

        case GOAL_PNIO_DCP_BLINK_TOGGLE:
            /* toggle signal */
            break;

        case GOAL_PNIO_DCP_BLINK_FINISH:
            /* switch signal off */
            break;
    }

    res = GOAL_OK_SUPPORTED;
    break;
#endif /* FIRST_SCENARIO */

#if SECOND_SCENARIO
    /*************************************/
    /* second scenario: act on light state */
    /*************************************/
    vendorBlinkFunction(cb->data[1].stateDcpLight);
    res = GOAL_OK_SUPPORTED;
    break;
#endif /* SECOND_SCENARIO */

#if THIRD_SCENARIO
    /************************************************/
    /* third scenario: do nothing / PROFINET stack handles signal */
    /************************************************/
#endif /* THIRD_SCENARIO */
```

5.5.7 GOAL_PNIO_CB_ID_CONNECT_FINISH - 连接请求完成

指示已完全处理连接请求，并允许应用程序通过设置错误状态取消该请求，并返回一个不是 ***GOAL_OK*** 的值。

表 5.46 连接请求完成

参数	说明
cb->data[0].idAr	应用程序关系 ID
cb->data[1].pStatus	PROFINET 状态指标

5.5.8 GOAL_PNIO_CB_ID_CONNECT_REQUEST - 连接请求

指示已开始处理连接请求。参数指标设置为*空*。

5.5.9 GOAL_PNIO_CB_ID_CONNECT_REQUEST_EXP_START- 预期子模块阻塞开始

指示预期的子模块阻塞开始。可用于在每个模块的请求回调传入之前拔掉所有模块。

表 5.47 预期子模块阻塞开始

参数	说明
cb->data[0].idAr	应用程序关系 ID

5.5.10 GOAL_PNIO_CB_ID_END_OF_PARAM - 收到的参数结束信息

指示已收到参数结束信息。

表 5.48 收到的参数结束信息

参数	说明
cb->data[0].idAr	应用程序关系 ID

5.5.11 GOAL_PNIO_CB_ID_END_OF_PARAM_PLUG –收到的插入参数结束信息

指示已收到插入参数结束信息。

表 5.49 收到的插入参数结束信息

参数	说明
cb->data[0].idAr	应用程序关系 ID
cb->data[1].u16	插入句柄

5.5.12 GOAL_PNIO_CB_ID_EXP_SUBMOD - 预期子模块

指示可以立即插入的预期子模块。

表 5.50 预期子模块

参数	说明
cb->data[0].idAr	应用程序关系 ID
cb->data[1].u32	API
cb->data[2].u16	slot 号
cb->data[3].u16	子槽号
cb->data[4].u32	模块标识号
cb->data[5].u32	子模块标识号
cb->data[6].valBool	模块标志（真=模块，假=子模块）

5.5.13 GOAL_PNIO_CB_ID_FACTORY_RESET - 恢复出厂设置

指示恢复出厂设置。参数指标设置为空。

5.5.14 GOAL_PNIO_CB_ID_IO_DATA_TIMEOUT - 周期性超时

指示输出端点超时。

5.5.15 GOAL_PNIO_CB_ID_NET_IP_SET - IP配置更新

指示 IP 配置已更新。内部更改标志指示更改由 PROFINET DCP (PN_TRUE)触发或由 DHCP (PN_FALSE)等外部调用程序触发。

表 5.51 IP 配置更新

参数	说明
cb->data[0].u32	IP 地址
cb->data[1].u32	网络掩码
cb->data[2].u32	网关
cb->data[3].valBool	临时设置标志
cb->data[4].valBool	内部更新标志

5.5.16 GOAL_PNIO_CB_ID_NEW_AR - 新应用程序关系

指示新应用程序关系。不同于 ***GOAL_OK*** 的返回值会删除 AR 并返回一个错误。

表 5.52 新应用程序关系

参数	说明
cb->data[0].idAr	应用程序关系 ID

5.5.17 GOAL_PNIO_CB_ID_NEW_IO_DATA - 新IO数据

指示接收到新的 IO 数据。

表 5.53 新 IO 数据

参数	说明
cb->data[0].u16	周期性帧 ID
cb->data[1].u16	数据长度
cb->data[2].pCu8	数据指标

5.5.18 GOAL_PNIO_CB_ID_PLUG_READY - 收到的插入就绪响应

指示已收到插入就绪响应。

表 5.54 收到的插入就绪响应

参数	说明
cb->data[0].idAr	应用程序关系 ID

5.5.19 GOAL_PNIO_CB_ID_READ_RECORD - 读取记录数据

指示应该读取堆栈本身无法处理的记录数据。在调用该回调之前，堆栈检查 API、slot 和 subslot 组合是否有效。如果该函数返回 GOAL_OK，将通过“无效索引”拒绝请求。
否则，如果应用程序可以处理请求，则需要返回 GOAL_OK_SUPPORTED，并且可以通过调用 API goal_pnioRecReadFinish 直接在回调中提供答复，或者存储回调信息并在回调范围之外响应，也可以通过调用 API goal_pnioRecReadFinish。如果设置了匹配参数，则 API goal_pnioRecReadFinish 还允许覆盖 PROFINET 状态。
如果响应不是 GOAL_OK 或 GOAL_OK_SUPPORTED，堆栈将自动响应应用程序读取错误。
在发送响应以检测请求是否过期时，可以稍后使用参数“序列号句柄”。

表 5.55 读取记录数据

参数	说明
cb->data[0].pStatus	未使用（PROFINET 状态指标）
cb->data[1].idAr	应用程序关系 ID
cb->data[2].u32	API
cb->data[3].u16	slot
cb->data[4].u16	subslot
cb->data[5].u16	记录指数
cb->data[6].pU8	未使用（存储记录数据指标）
cb->data[7].u32	最大记录读取数据长度
cb->data[8].i32	内部记录句柄
cb->data[9].u32	序列号句柄

回调代码示例：

```
case GOAL_PNIO_CB_ID_READ_RECORD:
    GOAL_PNIO_STATUS_T statusPnio = { 0, 0, 0, 0 }; /* PNIO Status */

    /* only process application index */
    if (RECORD_DATA_INDEX != pCb->data[5].u16) {
        return GOAL_OK;
    }

    goal_logInfo("CB: Read Request for record %x at api %"FMT_u32", slot %u, subslot %u",
                 pCb->data[5].u16, pCb->data[2].u32, pCb->data[3].u16, pCb->data[4].u16);

    /*
     * Process the record data and set error codes if something went wrong.
     * The following example shows a length check. Length should be 3
     * according to GSD.
     */
    if (RECORD_DATA_LENGTH > pCb->data[7].u32) {
        goal_logInfo("CB: Invalid data size in read request %"FMT_u32"!", pCb->data[7].u32);

        statusPnio.decode = GOAL_PNIO_ERR_DECODE_PNIORW;
        statusPnio.code1 = GOAL_PNIO_ERR_CODE1_PNIORW_ACCESS_INVAL_RANGE;
    }

    goal_pnioRecReadFinish(pHdlPnio, pCb->data[8].i32, &statusPnio, &recordBuf[0],
                           RECORD_DATA_LENGTH, pCb->data[9].u32);

    return GOAL_OK_SUPPORTED;
```

5.5.20 GOAL_PNIO_CB_ID_RELEASE_AR - 发布应用程序关系

指示已发布应用程序关系。

表 5.56 发布应用程序关系

参数	说明
cb->data[0].idAr	应用程序关系 ID

5.5.21 GOAL_PNIO_CB_ID_RESET_TO_FACTORY - 恢复到出厂设置

指示恢复到出厂设置请求。如果存在堆栈范围外的单独请求处理，则应用程序必须返回 GOAL_OK_SUPPORTED。
如果应用程序未通过 GOAL_OK 或 GOAL_OK_SUPPORTED 从回调返回，DCP 错误“0x04 未设置子选项”将发送回 DCP 设置请求发送程序。

表 5.57 恢复到出厂设置

参数	说明
cb->data[0].u16	恢复到出厂设置操作

5.5.22 GOAL_PNIO_CB_ID_STATION_NAME - 站名已更改

指示站名更改。如果永久标志设置为 GOAL_TRUE，则站名存储在 NVS 中，否则仅临时存储站名，并且 NVS 值将清除。

表 5.58 站名已更改

参数	说明
cb->data[0].pCu8	站名
cb->data[1].u32	站名长度
cb->data[2].valBool	永久标志

5.5.23 GOAL_PNIO_CB_ID_WRITE_RECORD - 写入记录数据

指示应该写入堆栈本身无法处理的记录数据。在调用该回调之前，堆栈检查 API、slot 和 subslot 组合是否有效。
如果该函数返回 GOAL_OK，将通过“无效索引”拒绝请求。
否则，如果应用程序可以处理请求，则需要返回 GOAL_OK_SUPPORTED，并且可以通过调用 API goal_pnioRecWriteFinish 直接在回调中提供答复，或者存储回调信息并在回调范围之外响应，也可以通过调用 API goal_pnioRecWriteFinish。如果设置了匹配参数，则 API goal_pnioRecReadFinish 还允许覆盖 PROFINET 状态。
如果响应不是 GOAL_OK 或 GOAL_OK_SUPPORTED，堆栈将自动响应应用程序读取错误。
在发送响应以检测请求是否过期时，可以稍后使用参数“序列号句柄”。

表 5.59 写入记录数据

参数	说明
cb->data[0].pStatus	未使用（PROFINET 状态指标）
cb->data[1].idAr	应用程序关系 ID
cb->data[2].u32	API
cb->data[3].u16	slot
cb->data[4].u16	subslot
cb->data[5].u16	记录指数
cb->data[6].pCu8	从中读取记录数据的指标
cb->data[7].u32	记录数据长度
cb->data[8].i32	内部记录句柄
cb->data[9].valBool	subslot 锁定状态标志
cb->data[10].u32	序列号句柄

回调代码示例：

```
case GOAL_PNIO_CB_ID_WRITE_RECORD:
    GOAL_PNIO_STATUS_T statusPnio = { 0, 0, 0, 0 }; /* PNIO Status */

    /* only process application index */
    if (RECORD_DATA_INDEX != pCb->data[5].u16) {
        return GOAL_OK;
    }

    goal_logInfo("CB: Write Request for record %x at api %"FMT_u32", slot %u, subslot %u",
                 pCb->data[5].u16, pCb->data[2].u32, pCb->data[3].u16, pCb->data[4].u16);

    /*
     * Process the record data and set error codes if something went wrong.
     * The following example shows a length check. Length should be 3
     * according to GSD.
     */
    if (RECORD_DATA_LENGTH > pCb->data[7].u32) {
        goal_logInfo("CB: Invalid data size in write request %"FMT_u32"!", pCb->data[7].u32);

        statusPnio.decode = GOAL_PNIO_ERR_DECODE_PNIORW;
        statusPnio.code1 = GOAL_PNIO_ERR_CODE1_PNIORW_ACCESS_INVAL_RANGE;
    } else {
        goal_logInfo("CB: Record data: 0x%x 0x%x 0x%x", pCb->data[6].pCu8[0],
                     pCb->data[6].pCu8[1], pCb->data[6].pCu8[2]);
        GOAL_MEMCPY(&recordBuf[0], &pCb->data[6].pCu8[0], RECORD_DATA_LENGTH);
    }
    goal_pnioRecWriteFinish(pHdlPnio, pCb->data[8].i32, &statusPnio, pCb->data[10].u32);

    return GOAL_OK_SUPPORTED;
```

5.5.24 GOAL_PNIO_CB_ID_INIT - 堆栈已初始化

指示 PROFINET 堆栈已完全初始化。参数指标设置为*空*。此时，可以安全创建设备配置。

5.5.25 GOAL_PNIO_CB_ID_LLDP_UPDATE - LLDP更新

指示合作伙伴端口上的设备已更改。

表 5.60 LLDP 更新

参数	说明
cb->data[0].u32	GOAL 以太网端口 ID

5.5.26 GOAL_PNIO_CB_ID_CONN_REQ_EXP_FINISH - 连接请求预期子模块阻塞完成

在解析连接请求中的预期子模块阻塞后，调用该回调。

表 5.61 连接请求预期子模块阻塞完成

参数	说明
cb->data[0].idAr	应用程序关系 ID

5.5.27 GOAL_PNIO_CB_ID_STATION_NAME_VERIFY - DCP站名验证

让应用程序验证 DCP 站名设置请求。

如果该函数返回 GOAL 错误状态，则将拒绝设置请求。

表 5.62 DCP 站名验证

参数	说明
cb->data[0].pCu8	站名指标（未终结）
cb->data[1].u32	站名长度
cb->data[2].valBool	永久标志（永久=真）

5.5.28 GOAL_PNIO_CB_ID_NET_IP_SET_VERIFY - DCP IP配置验证

让应用程序验证 DCP IP 配置设置请求。

如果该函数返回 GOAL 错误状态，则将拒绝设置请求。

表 5.63 DCP IP 配置验证

参数	说明
cb->data[0].u32	IP 地址
cb->data[1].u32	网络掩码
cb->data[2].u32	网关
cb->data[3].valBool	临时标志（临时=真）

5.6 通信栈 - EtherNet/IP

本节列出 GOAL EtherNet/IP 提供的 API 函数。

5.6.1 goal_eipCfgVendorIdSet

设置该 EtherNet/IP 堆栈实例的供应商 ID。供应商 ID 由 ODVA 分配。

默认值：1105

表 5.64goal_eipCfgVendorIdSet 参数

参数	说明
uint16_t vendorId	供应商 ID

```
res = goal_eipCfgVendorIdSet(1105);
```

5.6.2 goal_eipCfgDeviceTypeSet

设置该 EtherNet/IP 堆栈实例的设备类型。有效值通过 CIP 规范定义。
默认值：0x2B

表 5.65goal_eipCfgDeviceTypeSet 参数

参数	说明
uint16_t deviceType	设备类型

```
res = goal_eipCfgDeviceTypeSet(0x2B);
```

5.6.3 goal_eipCfgProductCodeSet

设置该 EtherNet/IP 堆栈实例的产品代码。该值由设备供应商定义

默认值：768

表 5.66goal_eipCfgProductCodeSet 参数

参数	说明
uint16_t productCode	产品代码

```
res = goal_eipCfgProductCodeSet(768);
```

5.6.4 goal_eipCfgRevisionSet

设置 EtherNet/IP 堆栈实例的版本。版本由一个主要编号和一个次要编号组成。它代表固件中影响设备行为的更改。

默认值：1.1

表 5.67goal_eipCfgRevisionSet 参数

参数	说明
uint8_t revMajor	版本主要编号
uint8_t revMinor	版本次要编号

```
res = goal_eipCfgRevisionSet(1, 1);
```

5.6.5 goal_eipCfgSerialNumSet

设置 EtherNet/IP 堆栈实例的序列号。该编号由供应商分配，每个设备的编号应唯一。

默认值：1

表 5.68goal_eipCfgSerialNumSet 参数

参数	说明
uint32_t serial	序列号

```
res = goal_eipCfgSerialNumSet(1);
```

5.6.6 goal_eipCfgProductNameSet

设置 EtherNet/IP 堆栈实例的产品名称。该名称由供应商分配。最大长度是 32 个字符。

默认值：“EtherNet/IP Adapter”

表 5.69goal_eipCfgProductNameSet 参数

参数	说明
const char *strName	产品名称

```
res = goal_eipCfgProductNameSet("EtherNet/IP Adapter");
```

5.6.7 goal_eipCfgDomainNameSet

设置 EtherNet/IP 堆栈实例的默认域名。如果找不到有效配置，该名称将用作设备的域名。最大长度是 48 个字符。

默认值：“port.de”

表 5.70goal_eipCfgDomainNameSet 参数

参数	说明
const char *strName	默认域名

```
res = goal_eipCfgDomainNameSet("example.org");
```

5.6.8 goal_eipCfgHostNameSet

设置 EtherNet/IP 堆栈实例的默认主机名。如果找不到有效配置，该名称将用作设备的主机名。最大长度是 64 个字符。
默认值：“eipdevice”

表 5.71goal_eipCfgHostNameSet 参数

参数	说明
const char *strName	默认主机名

```
res = goal_eipCfgHostNameSet("example_device");
```

5.6.9 goal_eipCfgNumExplicitConSet

设置设备一次可以处理的显式连接的最大数目。

默认值：6

表 5.72goal_eipCfgNumExplicitConSet 参数

参数	说明
uint16_t num	显式连接的数目

```
res = goal_eipCfgNumExplicitConSet(10);
```

5.6.10 goal_eipCfgNumImplicitConSetImpl

设置设备一次可以处理的隐式连接的最大数目。

默认值：6

表 5.73goal_eipCfgNumImplicitConSetImpl 参数

参数	说明
uint16_t num	隐式连接的数目

```
res = goal_eipCfgNumImplicitConSetImpl(10);
```

5.6.11 goal_eipCfgEthLinkCountersOn

禁用对以太网链路属性 4、5、12 和 13 的支持。
无论功能启用 (GOAL_TRUE) 为何，都会启用此功能。

默认值：GOAL_TRUE

表 5.74goal_eipCfgEthLinkCountersOn 参数

参数	说明
GOAL_BOOL_T enable	启用或禁用功能

```
res = goal_eipCfgEthLinkCountersOn(GOAL_FALSE);
```

5.6.12 goal_eipCfgEthLinkControlOn

禁用对以太网链路属性 6 的支持。
无论功能启用 (GOAL_TRUE) 为何，都会启用此功能。

默认值：GOAL_TRUE

表 5.75goal_eipCfgEthLinkControlOn 参数

参数	说明
GOAL_BOOL_T enable	启用或禁用功能

```
res = goal_eipCfgEthLinkControlOn(GOAL_FALSE);
```

5.6.13 goal_eipCfgChangeEthAfterResetOn

以太网链路速度或双工模式的更改需要复位设备。

默认值：GOAL_FALSE

表 5.76goal_eipCfgChangeEthAfterResetOn 参数

参数	说明
GOAL_BOOL_T enable	启用或禁用功能

```
res = goal_eipCfgChangeEthAfterResetOn(GOAL_FALSE);
```

5.6.14 goal_eipCfgChangeIpAfterResetOn

更改 IP 地址需要复位设备。

默认值：GOAL_FALSE

表 5.77goal_eipCfgChangeIpAfterResetOn 参数

参数	说明
GOAL_BOOL_T enable	启用或禁用功能

5.6.15 goal_eipCfgNumSessionsSet

设置设备一次可以处理的封装会话的最大数目。

默认值：20

表 5.78goal_eipCfgNumSessionsSet 参数

参数	说明
uint16_t num	会话数

```
res = goal_eipCfgNumSessionsSet(10);
```

5.6.16 goal_eipCfgTickSet

设置一个刻度的毫秒数。该堆栈使用刻度作为最小的时间单位。

默认值：10

表 5.79goal_eipCfgTickSet 参数

参数	说明
uint32_t ticks	一个刻度的毫秒数

```
res = goal_eipCfgTickSet(10);
```

注意：

此函数已弃用且无效。 保留它是为了与旧固件版本兼容。

5.6.17 goal_eipCfgO2TRunIdleHeaderOn

禁用使用的（发起者到目标）周期性数据的运行/空闲头。
无论功能启用 (GOAL_TRUE) 为何，都会启用此功能。

默认值：GOAL_TRUE

表 5.80 goal_eipCfgO2TRunIdleHeaderOn 参数

参数	说明
GOAL_BOOL_T enable	启用或禁用功能

res = goal_eipCfgO2TRunIdleHeaderOn(GOAL_FALSE);

5.6.18 goal_eipCfgT2ORunIdleHeaderOn

启用制作的（目标到发起者）周期性数据的运行/空闲头。

默认值：GOAL_FALSE

表 5.81 goal_eipCfgT2ORunIdleHeaderOn 参数

参数	说明
GOAL_BOOL_T enable	启用或禁用功能

res = goal_eipCfgT2ORunIdleHeaderOn(GOAL_FALSE);

5.6.19 goal_eipCfgQoSOn

禁用 QoS 对象 属性 4, 5, 6, 7 的支持。
无论功能启用 (GOAL_TRUE) 为何，都会启用此功能。

默认值：GOAL_TRUE

表 5.82goal_eipCfgQoSOn 参数

参数	说明
GOAL_BOOL_T enable	启用或禁用功能

res = goal_eipCfgQoSOn(GOAL_FALSE);

5.6.20 goal_eipCfgNumDelayedEncapMsgSet

设置可以同时延迟的封装消息数。

默认值：2

表 5.83goal_eipCfgNumDelayedEncapMsgSet 参数

参数	说明
uint16_t num	消息数

```
res = goal__eipCfgNumDelayedEncapMsgSet(2);
```

5.6.21 goal_eipCfgDhcpOn

启用平台支持的 DHCP 客户端。

默认值：GOAL_FALSE

表 5.84goal_eipCfgDhcpOn 参数

参数	说明
GOAL_BOOL_T enable	启用或禁用功能

```
res = goal_eipCfgDhcpOn(GOAL_TRUE);
```

5.6.22 goal_eipCfgDlrOn

启用 DLR 对象的支持。该功能需要 DLR 堆栈和硬件支持。

默认值：GOAL_FALSE

表 5.85goal_eipCfgDlrOn 参数

参数	说明
GOAL_BOOL_T enable	启用或禁用功能

```
res = goal_eipCfgDlrOn(GOAL_TRUE);
```

5.6.23 goal_eipCfgAcdOn

启用对地址冲突检测 (ACD) 的支持。

默认值：GOAL_FALSE

表 5.86goal_eipCfgAcdOn 参数

参数	说明
GOAL_BOOL_T enable	启用或禁用功能

```
res = goal_eipCfgAcdOn(GOAL_TRUE);
```

5.6.24 goal_eipCfgAcdConflictFallbackIp

如果地址冲突检测(ACD)检测到地址冲突，则配置回退 IP 配置。 如果设备配置为具有静态 IP 的设备，则该设备将应用配置的 IP。 如果 TCP/IP Class 的属性 3 设置为 DHCP，设备将在固定超时后启动 DHCP 进程。

- 转换到错误状态并且什么都不做：传递 ipAddress、subnetMask 和 gateway = 0，
- 设置回退 IP 配置：传递有效的 IP 配置。

表 5.87goal_eipCfgAcdConflictFallbackIp 参数

参数	说明
uint32_t ipAddress	回退 IP 地址
uint32_t subnetMask	回退子网掩码
uint32_t gateway	回退网关地址

```
uint32_t ipAddress = GOAL_NET_IPV4(192, 168, 0, 100);
uint32_t subnetMask = GOAL_NET_IPV4(255, 255, 255, 0);
uint32_t gateway = GOAL_NET_IPV4(192, 168, 0, 1);

res = goal_eipCfgAcdConflictFallbackIp(ipAddress, subnetMask, gateway);
```

5.7 应用程序回调 – EtherNet/IP

5.7.1 简介

为了充分利用 EtherNet/IP，该堆栈允许在协议的几个阶段进行交互。例如，在新的程序集数据到达后可能会收到回调。

该堆栈通过函数 goal_eipNew()调用注册的回调处理程序。回调处理程序返回 GOAL_STATUS_T 值。

表 5.88 回调处理程序参数

参数	说明
GOAL_EIP_T *pHdlEip	GOAL EtherNet/IP 句柄
GOAL_EIP_CB_ID_T id	回调 ID
GOAL_EIP_CB_DATA_T *pCb	回调参数

回调参数是一个联合体的阵列，最多包含 10 个元素。

```
1.  /*****************************************************************************/
2.  /** EtherNet/IP Callback Handler
3.   * This function collects all callbacks from the stack and decides if the * callback must
4.   *   be handled.
5.   */
6.
7.  GOAL_STATUS_T main_eipCallback(
8.      GOAL_EIP_T *pHdlEip,                    /**<EtherNet IP handle */
9.      GOAL_EIP_CB_ID_T id,                    /**<callback id */
10.     GOAL_EIP_CB_DATA_T *pCb                 /**<callback parameters */
11.   )
12. {
13.
14. GOAL_STATUS_T res = GOAL_OK;              /* result */
15.   switch (id) {
16.   case
17.     GOAL_EIP_CB_ID_INIT:
18.     /* initialize application resources */
19.     break;
20.     /*  …*/
21.   }
22.
23.   return res;
24. }
```

下文介绍有关可用回调 ID 的详细信息。

5.7.2 GOAL_EIP_CB_ID_INIT

堆栈已初始化。应用程序可以初始化其资源。

参数<无>

返回值

- GOAL_OK - 成功
- 其他 - 失败

5.7.3 GOAL_EIP_CB_ID_READY

初始化完成。

参数<无>

返回值<忽略>

5.7.4 GOAL_EIP_CB_ID_CONNECT_EVENT

向应用程序通知连接事件

参数

表 5.89 GOAL_EIP_CB_ID_CONNECT_EVENT 的回调参数

元素	构件	数据类型	
0	outputAssembly	uint32_t	输出程序集的实例 ID
1	inputAssembly	uint32_t	输入程序集的实例 ID
2	connectionEvent	uint32_t	GOAL_EIP_CONNECTION_EVENT_*

返回值<忽略>

5.7.5 GOAL_EIP_CB_ID_ASSEMBLY_DATA_RECV

收到程序集的新数据。

参数

表 5.90 GOAL_EIP_CB_ID_ASSEMBLY_DATA_RECV 的回调参数

元素	构件	数据类型	说明
0	instanceNr	uint32_t	程序集的实例 ID

返回值

- GOAL_OK - 成功
- 其他 - 失败

5.7.6 GOAL_EIP_CB_ID_ASSEMBLY_DATA_SEND

通知应用程序将发送程序集的数据。

参数

表 5.91 GOAL_EIP_CB_ID_ASSEMBLY_DATA_SEND 的回调参数

元素	构件	数据类型	说明
0	instanceNr	uint32_t	程序集的实例 ID

返回值

- GOAL_OK - 数据已更改
- 其他 - 数据未更改

5.7.7 GOAL_EIP_CB_ID_RUN_IDLE_CHANGED

通知应用程序已更改使用的周期性数据的运行/空闲头。

参数

表 5.92 GOAL_EIP_CB_ID_RUN_IDLE_CHANGED 的回调参数

元素	构件	数据类型	说明
0	outputAssembly	uint32_t	输出程序集的实例 ID
1	inputAssembly	uint32_t	输入程序集的实例 ID
2	runIdleValue	uint32_t	运行/空闲标志的当前值

返回值<忽略>

5.7.8 GOAL_EIP_CB_ID_LED_CHANGED

通知应用程序必须更改模块状态或网络状态 LED。该参数包含由 GOAL_EIP_LED_*宏构成的位图。如果已设置位，则必须设置相应的 LED。否则，必须将其清除。

参数

表 5.93 GOAL_EIP_CB_ID_LED_CHANGED 的回调参数

元素	构件	数据类型	说明
0	leds	uint32_t	活动 LED 的位图

返回值<忽略>

5.7.9 GOAL_EIP_CB_ID_DEVICE_RESET

通知应用程序设备将复位。根据平台，设备将复位或仅模拟复位。因此，应用程序必须重新初始化其资源。回调参数指示复位类型。

参数

表 5.94 GOAL_EIP_CB_ID_DEVICE_RESET 的回调参数

元素	构件	数据类型	
0	resetState	uint32_t	GOAL_EIP_RESET_*

返回值<忽略>

5.7.10 GOAL_EIP_CB_ID_REVISION_CHECK

如果在电子密钥段中设置了兼容性位，则应用程序可以决定是否支持次要修订。 这意味着应用程序可以决定它是否仍然与自身的先前版本兼容。

参数

表 5.95 GOAL_EIP_CB_ID_REVISION_CHECK 的回调参数

元素	构件	数据类型	说明
0	minorRevision	uint8_t	请求对应用程序进行小幅修改

返回值

- GOAL_OK - 支持修订
- 其他 - 不支持修订

5.7.11 GOAL_EIP_CB_ID_ACD_CONFLICT

将检测到的地址冲突通知应用程序。 根据应用程序返回 GOAL_OK 还是 GOAL_ERROR，设备将尝试解决与配置行为的冲突或转换为错误故障。 配置是通过函数 goal_eipCfgAcdConflictFallbackIp()完成的。

参数

表 5.96 GOAL_EIP_CB_ID_ACD_CONFLICT 的回调参数

元素	构件	数据类型	说明
0	minorRevision	uint8_t	ACD 状态如下： 0 - 未检测到冲突 1 - 持续检测 2 - 探测 IP 地址 3 - 半主动探测
1	acdRemoteMac	uint8_t *	设备 MAC 数组，导致地址冲突
2	acdArpPru	uint8_t *	原始帧数组，导致地址冲突
3	flgDhcpEnabled	GOAL_BOOL_T	启用 DHCP

返回值

- GOAL_OK - 设备将附加配置的行为
- GOAL_ERROR - 设备将转换为错误故障

5.8 通信栈 –EtherCAT

本节列出 GOAL EtherCAT 提供的 API 函数。

5.8.1 CoE API

CoE 模块提供指示事件，将应用程序连接至各对象和程序数据。还有触发 CoE 和 SDO 模块的 API 函数。

参数

表 5.97 CoE 指示事件

回调编号	说明
GOAL_ECAT_CB_ID_SDO_UPLOAD	指示 SDO Read，访问某个对象
GOAL_ECAT_CB_ID_SDO_DOWNLOAD	在写入前，检查对象的新数据
GOAL_ECAT_CB_ID_RxPDO_RECEIVED	指示收到输出程序数据（更新对象）
GOAL_ECAT_CB_ID_TxPDO_PREPARE	在 Sync Manager 映像前，更新输入程序数据对象

5.8.2 EoE API

当所启用的 EoE 内部连接至 GOAL 框架时，不存在 EoE API 参数。

5.8.3 FoE API

FoE 模块提供应用程序必须实施的指示事件。

参数

表 5.98 FOE 指示事件

事件	说明
GOAL_ECAT_CB_ID_FOE_READ_REQ	打开文件，进行读取
GOAL_ECAT_CB_ID_FOE_WRITE_REQ	打开文件，进行写入
GOAL_ECAT_CB_ID_FOE_READ_DATA	获取读取文件碎片
GOAL_ECAT_CB_ID_FOE_WRITE_DATA	写入被写入文件的碎片
GOAL_ECAT_CB_ID_FOE_ERROR	发生错误，关闭文件

5.8.4 EtherCAT 状态机

这些函数告知应用程序：EtherCAT 状态机中所发生的变更。

参数

表 5.99 ESM API

函数	说明
goal_ecatEsmStateGet()	获取现行 ESM 状态

表 5.100 ESM 事件

事件	说明
GOAL_ECAT_CB_ID_NEW_ESM_STATE_ENTERED	指示新 ESM 状态和可能的错误
GOAL_ECAT_CB_ID_NEW_ESM_STATE_REQUESTED	指示 ESM 状态变更请求
GOAL_ECAT_CB_ID_EXPLICIT_DEV_ID	（在状态变更期间）获取 Explicit Device ID

表 5.101 ESM 状态

象征	说明
GOAL_ECAT_ESM_STATE_UNKNOWN	unknown ESM state
GOAL_ECAT_ESM_STATE_INIT	ESM state INIT
GOAL_ECAT_ESM_STATE_PREOP	ESM state PreOP
GOAL_ECAT_ESM_STATE_BOOTSRTAP	ESM state BOOTSTRAP
GOAL_ECAT_ESM_STATE_SAFEOP	ESM state SafeOP
GOAL_ECAT_ESM_STATE_OP	ESM state OP

5.8.5 数据层指示函数

如果发生数据层事件，EtherCAT 库可以调用这些事件。

参数

表 5.102 数据链接事件

函数	说明
GOAL_ECAT_CB_ID_SM_WATCHDOG_EXPIRED	程序数据接收超时（硬件监控）
GOAL_ECAT_CB_ID_NEW_DL_STATE	状态变更端口

5.8.6 分布时钟 API

如果激活分布时钟同步，EtherCAT 库可以调用下述事件。

参数

表 5.103 DC 事件

函数	说明
GOAL_ECAT_CB_ID_NEW_DC_CONFIG	检查同步设置，例如：循环时间
GOAL_ECAT_CB_ID_DC_FAIL	指示 Sync0 干扰的丢失

5.9 应用程序回调 – EtherCAT

5.9.1 简介

为了充分利用 EtherCAT，该堆栈允许在协议的几个阶段进行交互。例如，在新的程序集数据到达后可能会收到回调。

该堆栈通过函数 goal_ecatNew()调用注册的回调处理程序。回调处理程序返回 GOAL_STATUS_T 值。

表 5.104 回调处理程序参数

参数	说明
GOAL_ECAT_T *pHdlEcat	GOAL EtherCAT 句柄
GOAL_ECAT_CB_ID_T id	回调 ID
GOAL_ECAT_CB_DATA_T *pCb	回调参数

回调参数是一个联合体的阵列，最多包含 6 个元素。

```
1.  /*****************************************************************************/
2.  /** EtherCAT Callback Handler
3.    * This function collects all callbacks from the stack and decides if the
4.    * callback must be handled.
5.    */
6.
7.  GOAL_STATUS_T appl_ecatCallback(
8.      GOAL_ECAT_T *pHdlEcat,                          /**< GOAL EtherCAT handle */
9.      GOAL_ECAT_CB_ID_T id,                           /**< callback id */
10.     GOAL_ECAT_CB_DATA_T *pCb                        /**< callback parameters */
11.   )
12. {
13.
14.     GOAL_STATUS_T res = GOAL_OK;                    /* result */
15.     switch (id) {
16.     case
17.       GOAL_ECAT_CB_ID_.....:
18.       break;
19.     }
20.
21.     return res;
22. }
```

将参数分布至回调函数，作为数据结构 GOAL_ECAT_CB_DATA_T 阵列。

回调数据内容结构

```
typedef union {
    uint16_t index;                     /**< object index */
    uint8_t subIndex;                   /**< object sub-index */
    uint8_t *pData;                     /**< object data */
    uint32_t size;                      /**< object size */
    GOAL_BOOL_T completeAccess;         /**< complete object is accessed */
    uint16_t dlState;                   /**< new Data Link layer state */
    uint32_t dcCycleTime;               /**< current DC cycle time */
    uint32_t dcCycleTimeMin;            /**< minimum supported DC cycle time */
    uint16_t explDevId;                 /**< Explicit device ID */
    uint16_t esmState;                  /**< new ESM state */
    GOAL_BOOL_T esmError;               /**< device is in error state */
    uint16_t statusCode;                /**< status code explaining error */
    uint16_t appStatusCode;             /**< application specific error */
    uint32_t foePassword;               /**< password for file access */
    char *pFileName;                    /**< file to be accessed via FoE */
    uint16_t foeError;                  /**< FOE error code */
    uint32_t foeOffset;                 /**< read/write offset in file */
    uint16_t foeMaxSize;                /**< maximum size of a FOE chunk */
    uint16_t foeActSize;                /**< actual size of a FOE chunk */
    GOAL_BOOL_T foeFinished;            /**< write access finished */
    GOAL_BOOL_T bLedEnable;             /**< led state */
} GOAL_ECAT_CB_DATA_ELEM_T;
```

```
typedef struct {
    GOAL_ECAT_CB_DATA_ELEM_T data[GOAL_ECAT_CB_DATA_MAX]; /**< callback data elements */
} GOAL_ECAT_CB_DATA_T;
```

根据回调编号，提供参数和返回值，预期在回调数据阵列内的识别编号特定位置内。

下文介绍有关可用回调 ID 的详细信息。

5.9.2 GOAL_ECAT_CB_ID_SDO_DOWNLOAD

通知应用程序SDO写入访问。

5.9.3 GOAL_ECAT_CB_ID_SDO_UPLOAD

通知应用程序SDO读取访问。

5.9.4 GOAL_ECAT_CB_ID_RxPDO_RECEIVED

通知应用程序更新输出数据。

5.9.5 GOAL_ECAT_CB_ID_TxPDO_PREPARE

通知应用程序提供新输入数据。

5.9.6 GOAL_ECAT_CB_ID_SYNC_FAIL

通知应用程序同步操作故障。

5.9.7 GOAL_ECAT_CB_ID_NEW_DL_STATE

通知应用程序新数据链接层状态。

5.9.8 GOAL_ECAT_CB_ID_NEW_DC_CONFIG

通知应用程序新DC配置。

5.9.9 GOAL_ECAT_CB_ID_DC_FAIL

通知应用程序DC同步失败。

5.9.10 GOAL_ECAT_CB_ID_SM_WATCHDOG_EXPIRED

通知应用程序Sync manager监控失败。

5.9.11 GOAL_ECAT_CB_ID_EXPLICIT_DEV_ID

通知应用程序明确设备识别编号申请。

5.9.12 GOAL_ECAT_CB_ID_NEW_ESM_STATE_ENTERED

通知应用程序输入新ESM状态。

5.9.13 GOAL_ECAT_CB_ID_NEW_ESM_STATE_REQUESTED

通知应用程序申请新ESM状态。

5.9.14 GOAL_ECAT_CB_ID_FOE_READ_REQ

通知应用程序FoE读取申请。

5.9.15 GOAL_ECAT_CB_ID_FOE_READ_DATA

通知应用程序FoE读取数据。

5.9.16 GOAL_ECAT_CB_ID_FOE_WRITE_REQ

通知应用程序FoE写入申请。

5.9.17 GOAL_ECAT_CB_ID_FOE_WRITE_DATA

通知应用程序FoE写入数据。

5.9.18 GOAL_ECAT_CB_ID_FOE_ERROR

通知应用程序FoE错误。

5.9.19 GOAL_ECAT_CB_ID_RUN_LED_STATE

通知应用程序RUN LED状态变更。

5.9.20 GOAL_ECAT_CB_ID_ERROR_LED_STATE

通知应用程序ERROR LED状态变更。

6. 应用程序编程接口

本节列出 R-IN32M3 Module 提供的 API 函数。

6.1 设备特定函数

6.1.1 appl_ccmRpcInit

目的：在 GOAL 中注册 R-IN32M3 Module API (appl_init)

该函数在 GOAL 中注册 R-IN32M3 Module 特定的 API，必须在 appl_init 中调用。它返回 GOAL_STATUS_T 状态，没有参数。

函数原型：

```
GOAL_STATUS_T appl_ccmRpcInit(
    void
);
```

示例：

```
/*****************************************************************************/
/** Application Init
 *
 * Build up the device structure and initialize the Profinet stack.
 */
GOAL_STATUS_T appl_init(
    void
)
{
    GOAL_STATUS_T res = GOAL_OK;            /* result */

    /* initialize RPC interface */
    res = appl_ccmRpcInit();
    if (GOAL_RES_ERR(res)) {
        goal_logErr("Initialization of RPC failed");
    }

    return res;
}
```

6.1.2 appl_ccmUpdateAllow

目的：启用 R-IN32M3 Module 中的固件更新

该函数启用更新 R-IN32M3 Module 固件的可能性。它返回 GOAL_STATUS_T 状态，没有参数。

函数原型

```
GOAL_STATUS_T appl_ccmUpdateAllow(
    void
);
```

6.1.3 appl_ccmUpdateDeny

目的：禁用 R-IN32M3 Module 中的固件更新

该函数禁用更新 R-IN32M3 Module 固件的可能性。该函数的一个可能用途是在周期性通信关系期间禁用固件更新的可能性。

它返回 GOAL_STATUS_T 状态，没有参数。

函数原型：

```
GOAL_STATUS_T appl_ccmUpdateDeny(
    void
);
```

6.1.4 appl_ccmInfo

目的：获取版本和设备信息

该函数从 R-IN32M3 Module 读取信息：

- MAC 地址（序列号）
- 软件版本
- 设备类型

函数原型：

```
GOAL_STATUS_T appl_ccmInfo(
    char *strVersion,                       /**< target string for version */
    uint8_t lenStrVersion,                  /**< length of str buffer */
    GOAL_ETH_MAC_ADDR_T *macAddress,        /**< mac address buffer */
    uint16_t *pDevType                      /**< device type */
);
```

示例：

```
/* get version and information of ccm */
if (GOAL_RES_OK(res)) {
    res = appl_ccmInfo(strVersion, APPL_VERSION_LEN, &mac, &devType);
}

if (GOAL_RES_OK(res)) {
    goal_logInfo("Device Version : %s", strVersion);
    goal_logInfo("Device Type : %u", devType);
    goal_logInfo("Serial Number: %02x:%02x:%02x:%02x:%02x:%02x",
        mac[0], mac[1], mac[2], mac[3], mac[4], mac[5]);
}
```

6.1.5 appl_ccmFaultStateSet

目的：设置在发生故障后现场总线通信行为

该函数确定 R-IN32M3 Module 关于周期性通信的行为。默认情况下，当与应用程序控制器(AC)的通信丢失时，周期性通信停止。

函数原型：

```
GOAL_STATUS_T appl_ccmFaultStateSet(
    APPL_CCM_FAULT_STATE_T faultState        /**< fault state to enter */
);
```

示例：

```
/* set fault state behavior */
if (GOAL_RES_OK(res)) {
    res = appl_ccmFaultStateSet(APPL_CCM_FAULT_STATE_ENTER);
}
```

6.1.6 appl_ccmCommResetSet

目的：设置在 SPI 同步复位请求后 R-IN32M3 Module 的行为

如果先前由正在运行的 AC 设置 CC，则在重启后 AC 会请求同步复位请求。

该函数确定 R-IN32M3 Module 关于该复位请求的行为。默认情况下不会复位。如果启用该选项，则复位完成。

该设置的状态存储在 CC 上的非易失性存储器中。数值 0 禁用复位。数值 1 启用复位。

函数原型：

```
GOAL_STATUS_T appl_ccmCommResetSet(
    uint8_t value                        /**< option value */
);
```

示例：

```
/* set sync reset behavior */
if (GOAL_RES_OK(res)) {
    /* enable reset on sync reset request from AC */
    res = appl_ccmCommResetSet(1);
}
```

6.1.7 appl_ccmLogEnable

目的：启用将 AC 日志消息传输到 CC

执行该函数后，来自 AC 日志缓冲区的日志消息将连续传输到 CC。然后，可通过管理接口将这些消息作为 CC 自身的日志消息访问。

函数原型：

```
GOAL_STATUS_T appl_ccmLogEnable(
    void
);
```

示例：

```
/* enable logging to CC */
if (GOAL_RES_OK(res)) {
    res = appl_ccmLogEnable();
}
```

6.1.8 appl_ccmLogToAcEnable

目的：启用 CC 日志消息传输至 AC

执行该函数后，来自 CC 日志缓冲区的日志消息将连续传输到 AC。然后，可通过本地日志机制（例如：序列控制台或终端）将这些消息作为 AC 自身的日志消息访问。

函数原型：

```
GOAL_STATUS_T appl_ccmLogToAcEnable(
    void
);
```

示例：

```
/* enable logging from CC to AC */
if (GOAL_RES_OK(res)) {
    res = appl_ccmLogToAcEnable();
}
```

6.1.9 appl_ccmFwUpdateStart

目的：使用 rpc，启动 CC 固件更新
该函数传输既定数据缓冲区（其中含有固件更新数据）至 CC。

函数原型：

```
GOAL_STATUS_T appl_ccmFwUpdateStart(
uint8_t *pFwData,                          /**< firmware data */
uint32_t fwSize                            /**< size of firmware data */
);
```

示例：

```
if (GOAL_RES_OK(res)) {
    res = appl_ccmFwUpdateStart(pBufFw, fsize);
}
```

6.1.10 appl_ccmFwUpdateExecute

目的：执行固件更新的实际更新程序
该函数执行 CC 的实际更新。在传输固件数据后，必须调用该函数。其中，要求使用 **appl_ccmFwUpdateCbReg()**对固件更新事件进行登记。如果应用程序未进行事件注册，则 CC 将以不明确方式调用该函数 **appl_ccmFwUpdateExecute**。

函数原型：

```
GOAL_STATUS_T appl_ccmFwUpdateExecute(
    void
);
```

示例：

```
/* perform actual update */
appl_ccmFwUpdateExecute();
```

6.1.11 appl_ccmEcatSsiUpdate

目的：执行 eeprom 内 EtherCAT SSI 数据的更新。
该函数允许以可选方式实现 EEPROM 内 EtherCAT SSI 数据的初始化。由于某些 EtherCAT 主设备依赖 eeprom 中的设置（不得重写（Configured Station Alias）），因此，仅调用一次该函数。

函数原型：

```
GOAL_STATUS_T appl_ccmEcatSsiUpdate(
unsigned char *pData,                    /**< SSI data */
uint32_t dataLen,                        /**< SSI data length */
GOAL_BOOL_T flgEmptyCheck                /**< empty check before writing */
);
```

示例：

```
/* configure SII in EEPROM before creating the EtherCAT instance */
res = appl_ccmEcatSsiUpdate(
&__09_ecat_slave_eeprom_bin[0],          /* data buffer */
__09_ecat_slave_eeprom_bin_len,          /* data buffer length */
  GOAL_FALSE);
if (GOAL_RES_ERR(res)) {
    goal_logErr("failed to configure EEPROM ssi data");
}
```

6.1.12 appl_ccmFoeUpdateSettings

目的：配置 FoE 固件更新要求
此功能允许更改 FoE 固件更新所需的设置。

函数原型：

```
1.  GOAL_STATUS_T appl_ccmFoeUpdateSettings(
2.          char *strName,                          /**< foe firmware filename */
3.          uint8_t matchLength,                    /**< string match length */
4.          uint32_t password,                      /**< foe password */
5.          GOAL_BOOL_T flgBootstrapReq             /**< bootstrap required */
6.  );
```

示例：

```
1.  /* enable FoE */
2.  res = goal_ecatCfgFoeOn(GOAL_TRUE);
3.  if (GOAL_RES_ERR(res)) {
4.          goal_logErr("failed to enable FoE support");
5.  }
6.
7.  /* enable BOOTSTRAP state */
8.  res = goal_ecatCfgBootstrapOn(GOAL_TRUE);
9.  if (GOAL_RES_ERR(res)) {
10.         goal_logErr("failed to enable BOOTSTRAP support");
11. }
12.
13. /* set settings for ccm firmware update via FoE */
14. res = appl_ccmFoeUpdateSettings(
15.         "ccm.efw",                              /* filename beginning */
16.         0,                                      /* 0 -> match all characters */
17.         0,                                      /* password */
18.         GOAL_TRUE);                             /* only update in ESM state bootstrap */
19. if (GOAL_RES_ERR(res)) {
20.         goal_logErr("failed to configure FoE firmware update of CC");
21.         return res;
22. }
```

备注：
为了通过 FoE 进行固件更新，FoE 需要使用函数 goal_ecatCfgFoeOn(GOAL_TRUE) 启用。 如果固件更新配置为在 BOOTSTRAP ESM 状态下进行，则需要使用 goal_ecatCfgBootstrapOn(GOAL_TRUE) 启用对此状态的支持。 除此之外，appl_ccmFoeUpdateSettings 的参数定义了执行固件更新所需的内容：

表 6.1appl_ccmFoeUpdateSettings 参数

参数	类型	说明
strName	Char *	固件更新所需的文件名
umatchLength	uint8_t	配置的目标文件名和更新过程的目标文件名之间匹配所需的字符数。 如果 matchLength 为零，则 strName 的所有字节都需要匹配。 如果 strName 是长度为 0 的字符串，则应使用默认文件名“ccm.efw”。
password	uint32_t	预期接受固件更新的 FoE 密码
flgBootstrapReq	bool	如果启用，固件更新将仅在 esm 状态引导程序中被接受

6.1.13 appl_ccmEthMacAddressSet

目的：配置设备的定制 mac 地址
该函数允许变更设备 mac 地址。应在任何网络相关函数（通信栈启动、网络初始化）之前，调用该函数。

函数原型：

```
1. GOAL_STATUS_T appl_ccmEthMacAddressSet(
2. uint8_t *pMacAddr
3. );
```

示例：

```
1. /* configure MAC address */
2. uint8_t devMacId[8] = {0x02, 0x01, 0x00, 0x00, 0x00, 0x01};
3. res = appl_ccmEthMacAddressSet(&devMacId[0]);
```

6.1.14 appl_ccmNetworkDefaultUp

目的： 启动默认网络
该函数以标准以太网模式启动 CC。通常情况下，无需调用该函数，因为协议栈以正确模式自动启动网络。

函数原型：

```
GOAL_STATUS_T appl_ccmNetworkDefaultUp(
    void
);
```

示例：

```
/* start default networking */
res = appl_ccmNetworkDefaultUp();
```

6.1.15 appl_ccmNetworkEoEUp

目的： 启动 EtherCAT EoE 网络
该函数以 EtherCAT 模式启动 CC。通常情况下，无需调用该函数，因为协议栈以正确模式自动启动网络。

函数原型：

```
GOAL_STATUS_T appl_ccmNetworkEoEUp(
    void
);
```

示例：

```
/* start EtherCAT networking */
res = appl_ccmNetworkEoEUp();
```

6.1.16 appl_ccmCfgVarGet

目的：读取配置变量
该函数提供一种读取 CC 配置变量的机制。其需要模块识别编号和变量识别编号（参见第4.9章）。

函数原型：

```
GOAL_STATUS_T appl_ccmCfgVarGet(
    uint32_t modId,                 /**< module id */
    uint32_t varId,                  /**< variable id */
    void *pBuf,                     /**< [out] output buffer */
    uint32_t bufLength,              /**< buffer length */
    uint32_t *pVarLength,            /**< [out] variable length */
    uint32_t *pVarType              /**< [out] variable type */
);
```

示例：

```
/* get signature of firmware */
if (GOAL_RES_OK(res)) {
res = appl_ccmCfgVarGet(
        37, /* module id */
        0,   /* variable id */
        &fwSignature[0],
        sizeof(fwSignature),
        NULL, NULL);
}
```

6.1.17 appl_ccmCfgVarSet

目的： 写入配置变量
该函数提供一种写入 CC 配置变量的机制。其需要模块识别编号和变量识别编号（参见第4.9章）。

函数原型：

```
GOAL_STATUS_T appl_ccmCfgVarSet(
    uint32_t modId,                 /**< module id */
    uint32_t varId,                 /**< variable id */
    void *pBuf,                     /**< [out] output buffer */
    uint32_t bufLength,             /**< buffer length */
);
```

示例：

```
/* demonstaration for writing of config variables */
valObj = 0x12345678;
if (GOAL_RES_OK(res)) {
res = appl_ccmCfgVarSet(
          34, /* module id dd */
          1,   /* variable id customer id */
           &valObj,
          sizeof(valObj));
}
```

6.1.18 appl_ccmCfgSave

目的：永久存储配置变量变更
该函数提供一种更新非易失性存储（包含配置变量）的机制。将存储现行配置变量值。

函数原型：

```
GOAL_STATUS_T appl_ccmCfgSave(
    void
);
```

示例：

```
/* store current config variable values */
if (GOAL_RES_OK(res)) {
     res = appl_ccmCfgSave();
}
```

6.2 设备检测

6.2.1 goal_ddInit - 在GOAL中注册GOAL dd API (appl_init)

目的：该函数在 GOAL 中注册 GOAL dd 特定的 API，必须在 appl_init 中调用。它返回 GOAL_STATUS_T 状态，没有参数。

函数原型：

```
GOAL_STATUS_T goal_ddInit (
    void
);
```

示例：

```
GOAL_STATUS_T appl_init( void
)
{
    GOAL_STATUS_T res;                                  /**< GOAL result */

    /* initialize GOAL dd API */
    res = goal_ddInit();
    if (GOAL_RES_ERR(res)) {
        goal_logErr("failed to initialize GOAL dd API");
    }
    return res;
}
```

6.2.2 goal_ddNew - 在GOAL中注册GOAL dd API (appl_setup)

目的：该函数在 GOAL 中创建 GOAL dd 的实例，必须在 appl_setup 中调用。使用 GOAL dd API 需要一个有效的实例。

它返回 GOAL_STATUS_T 状态，没有参数。

表 6.2goal_ddNew 参数

参数	说明
GOAL_DD_T **ppHdl	返回的 GOAL dd 实例句柄
uint32_t bitmaskFeatures	要启用的初始实例功能

参数 bitmaskFeatures 是一个位掩码，如果进行如下设置，将禁用 GOAL dd 的单一功能：

表 6.3 功能位掩码参数

位	特性
0	禁用 HELLO 请求（用于设备扫描检测）
1	禁用 WINK 命令
2	禁用 GETLIST 命令（读取可用 CM 变量的列表）
3	禁用 GETCONFIG 命令（读取 CM 变量值）
4	禁用 SETCONFIG 命令（写入 CM 变量值）
5	禁用 SETIP 命令（通过 GOAL dd 配置 IP）

函数原型：

```
GOAL_STATUS_T goal_ddNew(
   GOAL_DD_T **ppHdlDd,                /**< DD handle */
   uint32_t bitmaskFeatures            /**< initial features */
);
```

示例：

```
   static GOAL_DD_T *pHdlDd;              /**< GOAL dd handle */

   GOAL_STATUS_T appl_setup(
      void
   )
   {
      GOAL_STATUS_T res;                  /**< GOAL result */
         /* create GOAL dd instance */
         res = goal_ddNew(&pHdlDd, GOAL_DD_FEAT_ALL);
         if (GOAL_RES_ERR(res)) {
            goal_logErr("failed to create GOAL dd instance");
         }
      return res;
   }
```

6.2.3 goal_ddCustomerIdSet - 配置GOAL dd实例的客户ID

目的：该函数配置给定 GOAL dd 实例的客户 ID。客户 ID 是底层协议的一个属性，它包含在每个使用 GOAL dd 的请求中。存在一个值为零的特殊客户 ID，这将导致处理每个请求。如果客户 ID 不为零，则只有当请求的客户 ID 等于 GOAL dd 实例的客户 ID 时，才会处理请求。

客户 ID 存储在非易失性存储器中。

表 6.4goal_ddCustomerIdSet 参数

参数	说明
GOAL_DD_T *pHdl	GOAL dd 实例句柄
uint32_t customerId	实例客户 ID

函数原型：

```
1.  GOAL_STATUS_T goal_ddCustomerIdSet(
2.      GOAL_DD_T *pHdlDd,                  /**< dd handle */
3.      uint32_t cid                        /**< customer ID */
4.  );
```

示例：

```
1.  /* configure DD properties */
2.  res = goal_ddCustomerIdSet(pHdlDd, APPL_DD_CUSTOMER_ID);
3.  if (GOAL_RES_ERR(res)) {
4.      goal_logErr("failed to configure DD customer id");
5.  }
```

6.2.4 goal_ddModuleNameSet - 配置GOAL dd实例的名称

目的：该函数配置给定 GOAL dd 实例的模块名称。module id 是存储在非易失性存储器中的设备的一个属性。Management Tool 使用该函数进行设备命名。模块名称长度限制为 20 字节。

表 6.5goal_ddModuleNameSet 参数

参数	说明
GOAL_DD_T *pHdl	GOAL dd 实例句柄
uint8_t *str	实例模块名称

函数原型：

```
1.  GOAL_STATUS_T goal_ddModuleNameSet(
2.      GOAL_DD_T *pHdlDd,                  /**< dd handle */
3.      uint8_t *str                        /**< module name */
4.  );
```

示例：

```
1.  res = goal_ddModuleNameSet(pHdlDd, APPL_DD_MODULE_NAME);
2.  if (GOAL_RES_ERR(res)) {
3.      goal_logErr("failed to configure DD module name");
4.  }
```

6.2.5 goal_ddFeaturesSet - 配置GOAL dd实例的功能

目的：该函数配置给定 GOAL dd 实例的禁用功能。该属性存储在非易失性存储器中的设备中。

表 6.6goal_ddFeaturesSet 参数

参数	说明
GOAL_DD_T *pHdl	GOAL dd 实例句柄
uint32_t bitmaskFeatures	实例功能位掩码

有关参数说明，请参见函数“goal_ddNew”。

函数原型：

```
1.  GOAL_STATUS_T goal_ddFeaturesSet(
2.      GOAL_DD_T *pHdlDd,                  /**< dd handle */
3.      uint32_t bitmaskFeatures            /**< bitmask with feature disable bits set */
4.  );
```

示例：

```
1.  res = goal_ddFeaturesSet(pHdlDd, APPL_DD_FEATURES);
2.  if (GOAL_RES_ERR(res)) {
3.      goal_logErr("failed to configure DD features");
4.  }
```

6.2.6 goal_ddCallbackReg - 配置GOAL dd实例的回调

目的：该函数注册给定 GOAL dd 实例的回调。

回调类型：

```
1.  typedef GOAL_STATUS_T (* GOAL_DD_FUNC_CB_T)(
2.      struct GOAL_DD_T *pHdlDd,           /**< dd handle */
3.      GOAL_DD_CB_ID_T cbId,               /**< callback id */
4.      GOAL_DD_CB_DATA_T *pCbData          /**< callback data */
5.  );
```

函数原型：

```
1.  static GOAL_STATUS_T ddCallback(
2.      GOAL_DD_T *pHdlDd,                  /**< dd handle */
3.      GOAL_DD_CB_ID_T cbId,               /**< callback ID */
4.      GOAL_DD_CB_DATA_T *pCbData          /**< callback data */
5.  );
```

示例：

```
/*****************************************************************************/
/** Application Setup
 *
 * This function must setup all used protocol stacks.
 *
 * Notes if CSAP is active:
 *  Steps setup in this stage may be covered by CSAP and therefore must
 *  only contain functions supporting this.
 */
GOAL_STATUS_T appl_setup(
    void
)
{
    GOAL_STATUS_T res;                  /* result */

    res = goal_ddCallbackReg(pHdlDd, (GOAL_DD_FUNC_CB_T) ddCallback);

    return res;
}


/*****************************************************************************/
/** goal dd callback
 *
 */
 static GOAL_STATUS_T ddCallback(
    GOAL_DD_T *pHdlDd,                  /**< dd handle */
    GOAL_DD_CB_ID_T cbId,               /**< callback ID */
    GOAL_DD_CB_DATA_T *pCbData          /**< callback data */
)
{
    UNUSEDARG(pHdlDd);
    UNUSEDARG(pCbData);

    switch (cbId) {
       case GOAL_DD_CB_ID_WINK:
          goal_logInfo("Wink command received");
          break;
       default:
          break;
    }

    return GOAL_OK;
}
```

6.2.7 goal_ddSessionFeatureActivate - GOAL dd实例功能的临时激活

目的：该函数临时启用给定 GOAL dd 实例的功能。该属性*未*存储在非易失性存储器中。

表 6.7goal_ddSessionFeatureActivation 参数

参数	说明
GOAL_DD_T *pHdl	GOAL dd 实例句柄
uint32_t bitmaskFeatures	实例功能位掩码

注意：参数"bitmaskFeatures"用于还原为功能的永久配置函数，此处设置的位将启用给定功能。

函数原型：

```
1.  GOAL_STATUS_T goal_ddSessionFeatureActivate(
2.      GOAL_DD_T *pHdlDd,                  /** dd handle */
3.      uint32_t bitmaskFeatures            /** bitmask with feature enable bits set */
4.  );
```

示例：

```
1.  /* temporarily enable capability to respond to hello requests (device detection) */
2.  res = goal_ddSessionFeatureActivte (pHdlDd, GOAL_DD_FEAT_HELLO);
```

6.2.8 goal_ddFilterAdd - CM变量的限制访问

目的：默认情况下，作为 Management Tool 的外部应用程序可以完全访问设备的所有 CM 变量。该功能对于开发来说很方便，但是对于生产目的，可能需使用 Management Tool 仅限制对最小功能所需的变量的访问。因此，引入了筛选项完成该任务。预先定义了下列筛选项：

表 6.8goal_ddFilterAdd 筛选项集合

筛选项 ID	筛选项名称
0	GOAL_DD_ACCESS_FILTER_SET_ALL
1	GOAL_DD_ACCESS_FILTER_SET_BASIC
2	GOAL_DD_ACCESS_FILTER_SET_HIDDEN

表 6.9goal_ddFilterAdd 预先定义的筛选项

筛选项 ID	筛选操作	目的
0	授予对所有变量的完全访问	开发
1	读取对 NET 模块的所有变量（IP 设置）的访问、读取对 ETH 模块的所有变量（MAC、状态）的访问、完全访问 LM 模块的所有变量 （日志记录）	Management Tool 提供最少支持的生产代码
2	禁用对 Web 服务器身份验证字符串的读取访问	生产代码

6.2.9 筛选项定义 - GOAL_DD_ACCESS_FILTER_SET_ALL

```
1. /**< complete access */
2. static GOAL_DD_VAR_T ddAccessAll[] = {
3.     {
4.         .pNext = NULL,
5.         .modId = GOAL_DD_VAR_ALL,
6.         .varId = GOAL_DD_VAR_ALL,
7.         .access = (1 << GOAL_DD_ACCESS_READ) | (1 << GOAL_DD_ACCESS_WRITE)
8.     }
9. };
```

6.2.10 筛选项定义 - GOAL_DD_ACCESS_FILTER_SET_BASIC

```
/**< list of variables that are required for basic functionality */
static GOAL_DD_VAR_T ddAccessListBasic[] = {
    {   /* read access to network settings */
        .pNext = (struct GOAL_DD_VAR_T *) &ddAccessListBasic[1],
        .modId = GOAL_ID_NET,
        .varId = GOAL_DD_VAR_ALL,
        .access = (1 << GOAL_DD_ACCESS_READ)
    },
    {   /* read access to ethernet properties */
        .pNext = (struct GOAL_DD_VAR_T *) &ddAccessListBasic[2],
        .modId = GOAL_ID_ETH,
        .varId = GOAL_DD_VAR_ALL,
        .access = (1 << GOAL_DD_ACCESS_READ)
    },
    {   /* access to logs */
        .pNext = NULL,
        .modId = GOAL_ID_LM,
        .varId = GOAL_DD_VAR_ALL,
        .access = (1 << GOAL_DD_ACCESS_READ) | (1 << GOAL_DD_ACCESS_WRITE)
    }
};
```

6.2.11 筛选项定义 - GOAL_DD_ACCESS_FILTER_SET_HIDDEN

```
static GOAL_DD_VAR_T ddAccessListHidden[] = {
    {   /* disable read access to HTTP user level 0 */
        .pNext = (struct GOAL_DD_VAR_T *) &ddAccessListHidden[1],
        .modId = GOAL_ID_HTTP,
        .varId = 2, /* USERLEVEL0 */
        .access = (1 << GOAL_DD_ACCESS_WRITE)
    },
    {   /* disable read access to HTTP user level 1 */
        .pNext = (struct GOAL_DD_VAR_T *) &ddAccessListHidden[2],
        .modId = GOAL_ID_HTTP,
        .varId = 3, /* USERLEVEL1 */
        .access = (1 << GOAL_DD_ACCESS_WRITE)
    },
    {   /* disable read access to HTTP user level 2 */
        .pNext = (struct GOAL_DD_VAR_T *) &ddAccessListHidden[3],
        .modId = GOAL_ID_HTTP,
        .varId = 4, /* USERLEVEL2 */
        .access = (1 << GOAL_DD_ACCESS_WRITE)
    },
    {   /* disable read access to HTTP user level 3 */
        .pNext = NULL,
        .modId = GOAL_ID_HTTP,
        .varId = 5, /* USERLEVEL3 */
        .access = (1 << GOAL_DD_ACCESS_WRITE)
    },
};
```

函数原型：

```
GOAL_STATUS_T goal_ddFilterAdd(
    GOAL_DD_T *pHdlDd,                          /**< dd handle */
    GOAL_DD_ACCESS_FILTER_SET_T setId           /**< set id */
);
```

示例：

```
    /* enable for full access */
#if 0
    res = goal_ddFilterAdd(pHdlDd, GOAL_DD_ACCESS_FILTER_SET_ALL);
    if (GOAL_RES_ERR(res)) {
        goal_logErr("failed to set dd access filter");
    }
#endif

    res = goal_ddFilterAdd(pHdlDd, GOAL_DD_ACCESS_FILTER_SET_HIDDEN);
    if (GOAL_RES_ERR(res)) {
        goal_logErr("failed to set dd access filter");
    }
```

6.3 PROFINET堆栈应用程序编程接口

6.3.1 goal_pnioInit - 在GOAL中注册GOAL PROFINET (appl_init)

该函数在 GOAL 中注册 GOAL PROFINET，必须在 appl_init 中调用。其主要功能是在初始化 NVS 存储之前注册设置配置管理器变量。

它返回 GOAL_STATUS_T 状态，没有参数。

```
GOAL_STATUS_T appl_init(
    void
)
{
    GOAL_STATUS_T res;                              /**< GOAL result */

    /* initialize GOAL PROFINET */
    res = goal_pnioInit();
    if (GOAL_RES_ERR(res)) {
        goal_logErr("failed to initialize GOAL PROFINET");
    }

    return res;
}
```

6.3.2 goal_pnioNew - 创建 GOAL PROFINET 实例 (appl_setup)

该函数通过获取所有先前定义的变量(goal_pnioCfg API)的快照并分配必要的资源，创建新的 GOAL PROFINET 实例。应用程序必须提供 GOAL PROFINET 实例 ID 和回调处理程序。回调处理程序也可以为空，以便仅使用 GOAL PROFINET 堆栈默认行为。

它返回 GOAL_STATUS_T 状态和 GOAL PROFINET 实例句柄。

表 6.10goal_pnioNew 参数

参数	说明
GOAL_PNIO_T **ppPnio	返回的 GOAL PROFINET 实例句柄
const uint32_t id	GOAL PROFINET 实例 ID
GOAL_PNIO_FUNC_CB_T pFunc	GOAL PROFINET 回调函数

6.3.3 goal_pnioCfgDcpFactoryResetDisableSet - 配置DCP恢复出厂设置

控制 DCP 恢复出厂设置是否可用。如果设置为 GOAL_TRUE DCP，则将拒绝恢复出厂设置。默认值：GOAL_FALSE。返回 GOAL_STATUS_T 状态。

该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

警告：不应使用该函数，否则可能会破坏一致性。

表 6.11goal_pnioCfgDcpFactoryResetDisableSet 参数

参数	说明
GOAL_BOOL_T flgDcpFactoryResetDisable	DCP 恢复出厂设置禁用标志

6.3.4 goal_pnioCfgDcpAcceptMixcaseStationSet - 配置DCP MixcaseStationname接受

控制 DCP 是否接受大小写混合的站名。如果设置为 GOAL_TRUE DCP，则接受大小写混合的站名。默认值：GOAL_FALSE。

返回 GOAL_STATUS_T 状态。

该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

警告：不应使用该函数，否则可能会破坏一致性。

表 6.12goal_pnioCfgDcpAcceptMixcaseStationSet 参数

参数	说明
GOAL_BOOL_T flgDcpAcceptMixcaseStation	DCP MixcaseStationname 接受标志

6.3.5 goal_pnioCfgDevDapSimpleSet - 配置设备DAP简单模式

配置设备 DAP 简单模式。如果 flgDevDapSimple 为 GOAL_TRUE，则 GOAL PROFINET 堆栈会自动创建 DAP 和端口 slot 及模块。

默认值：GOAL_TRUE

返回 GOAL_STATUS_T 状态。

该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

警告：应小心使用该函数，因为可能会破坏一致性。

表 6.13goal_pnioCfgDevDapSimpleSet 参数

参数	说明
GOAL_BOOL_T flgDevDapSimple	设备 DAP 简单模式标志

6.3.6 goal_pnioCfgDevDapApiSet - 设置设备DAP API号

配置设备 DAP API 号。默认值：0

返回 GOAL_STATUS_T 状态。

该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

警告：应小心使用该函数，因为可能会破坏一致性。

表 6.14goal_pnioCfgDevDapApiSet 参数

参数	说明
uint32_t idDevDapApi	设备 DAP API 号

6.3.7 goal_pnioCfgDevDapSlotSet - 设置设备DAP slot号

配置设备 DAP slot 号。默认值：0

返回 GOAL_STATUS_T 状态。

该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

警告：应小心使用该函数，因为可能会破坏一致性。

表 6.15goal_pnioCfgDevDapSlotSet 参数

参数	说明
uint16_t idDevDapSlot	设备 DAP slot 号

6.3.8 goal_pnioCfgDevDapSubslotSet - 设置设备DAP subslot号

配置设备 DAP subslot 号。默认值：1；返回 GOAL_STATUS_T 状态。

该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

警告：应小心使用该函数，因为可能会破坏一致性。

表 6.16goal_pnioCfgDevDapSubslotSet 参数

参数	说明
uint16_t idDevDapSubslot	设备 DAP subslot 号

6.3.9 goal_pnioCfgDevDapModuleSet - 设置设备DAP module id

配置设备 DAPmodule id。默认值：1；返回 GOAL_STATUS_T 状态。

该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

警告：应小心使用该函数，因为可能会破坏一致性。

表 6.17goal_pnioCfgDevDapModuleSet 参数

参数	说明
uint32_t idDevDapMod	设备 DAPmodule id

6.3.10 goal_pnioCfgDevDapSubmoduleSet - 设置设备DAP子 module id

配置设备 DAP 子 module id。默认值：1；返回 GOAL_STATUS_T 状态。

该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

警告：应小心使用该函数，因为可能会破坏一致性。

表 6.18goal_pnioCfgDevDapSubmoduleSet 参数

参数	说明
uint32_t idDevDapSubmod	设备 DAP 子 module id

6.3.11 goal_pnioCfgNetLinkSafetySet - 配置设备端口禁用行为

配置设备端口禁用行为。如果 flgNetLinkSafety 设置为 GOAL_TRUE，则 GOAL PROFINET 堆栈不允许主设备同时禁用所有以太网接口。

默认值：GOAL_TRUE；返回 GOAL_STATUS_T 状态。

该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

警告：应小心使用该函数，因为可能会破坏一致性。

表 6.19goal_pnioCfgNetLinkSafetySet 参数

参数	说明
GOAL_BOOL_T flgNetLinkSafety	设备端口禁用行为

6.3.12 goal_pnioCfgNewIoDataCbSet - 配置新的IO数据回调

配置新的 IO 数据回调。如果 flgCbNewIoData 设置为 GOAL_TRUE，则还会为每个到达的周期性帧调用已注册的回调函数。必须尽快返回一个值，因为这是在实时环境中进行的。

默认值：GOAL_FALSE；返回 GOAL_STATUS_T 状态。

该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

警告：如果启用，则会产生非常高的应用程序负载，因为如果周期时间设置为 1 ms，则每秒会调用 1000 次回调。

表 6.20goal_pnioCfgNewIoDataCbSet 参数

参数	说明
GOAL_BOOL_T flgCbNewIoData	新的 IO 数据回调标志

6.3.13 goal_pnioCfgDiagBufMaxCntSet - 配置最大诊断条目数

配置诊断条目的最大计数。

默认值：20；返回 GOAL_STATUS_T 状态。

该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

警告：使用太小的值可能会破坏一致性。

表 6.21goal_pnioCfgDiagBufMaxCntSet 参数

参数	说明
unsigned intnumDiagBufMax	最大诊断条目数

6.3.14 goal_pnioCfgDiagBufMaxDataSizeSet - 配置最大诊断数据大小

配置每个诊断条目的最大字节数。默认值：28；返回 GOAL_STATUS_T 状态。

该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

警告：使用小于 28 的值可能会破坏一致性。

表 6.22goal_pnioCfgDiagBufMaxDataSizeSet 参数

参数	说明
unsigned intnumDiagDataSizeMax	最大诊断数据大小

6.3.15 goal_pnioCfgIocrBlocksMaxSet – 配置最大IOCR块缓冲区

配置 IOCR 块缓冲区的最大计数。默认值：10；返回 GOAL_STATUS_T 状态。

该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

警告：应小心使用该函数，因为可能会破坏一致性。

表 6.23goal_pnioCfgIocrBlocksMaxSet 参数

参数	说明
unsigned intnumIocrBlocksMax	最大 IOCR 块缓冲区

6.3.16 goal_pnioCfgCrMaxCntSet - 配置最大通信关系计数

配置通信关系的最大计数。

默认值：10；返回 GOAL_STATUS_T 状态。

该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

警告：应小心使用该函数，因为可能会破坏一致性。

表 6.24goal_pnioCfgCrMaxCntSet 参数

参数	说明
unsigned intnumCrMax	最大 CR 计数

6.3.17 goal_pnioCfgArMaxCntSet - 配置最大应用程序关系计数

配置应用程序关系的最大计数。

默认值：2；返回 GOAL_STATUS_T 状态。

该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

警告：应小心使用该函数，因为可能会破坏一致性。

表 6.25goal_pnioCfgArMaxCntSet 参数

参数	说明
unsigned intnumArMax	最大 AR 计数

6.3.18 goal_pnioCfgApiMaxCntSet - 配置最大API计数

配置应用程序进程标识符的最大计数。设置该值只会影响 ModuleDiffBlock 逻辑，并且应足够大以容纳最大可能的 GSDML 配置。

默认值：1；返回 GOAL_STATUS_T 状态。

该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

警告：不应使用该函数，否则可能会破坏一致性，而且 GOAL PROFINET 堆栈仅支持 API 0。

表 6.26goal_pnioCfgApiMaxCntSet 参数

参数	说明
unsigned intnumApiMax	最大 API 计数

6.3.19 goal_pnioCfgSlotMaxCntSet - 配置最大slot计数

配置 slot 的最大计数。设置该值只会影响 ModuleDiffBlock 逻辑，并且应足够大以容纳最大可能的 GSDML 配置。

默认值：12；返回 GOAL_STATUS_T 状态。

该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

表 6.27goal_pnioCfgSlotMaxCntSet 参数

参数	说明
unsigned intnumSlotMax	最大 slot 计数

6.3.20 goal_pnioCfgSubslotMaxCntSet - 配置最大subslot计数

配置每个 slot 的最大 subslot 计数。设置该值只会影响 ModuleDiffBlock 逻辑，并且应足够大以容纳最大可能的 GSDML 配置。默认值：4；返回 GOAL_STATUS_T 状态。

该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

表 6.28goal_pnioCfgSubslotMaxCntSet 参数

参数	说明
unsigned intnumSubslotMax	每个 slot 的最大 subslot 计数

6.3.21 goal_pnioCfgSubslotIfSet - 配置接口subslot

配置接口 subslot ID。

默认值：0x8000；返回 GOAL_STATUS_T 状态。

该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

警告：应小心使用该函数，因为可能会破坏一致性。

表 6.29goal_pnioCfgSubslotIfSet 参数

参数	说明
unsigned intidSubslotIf	接口 subslotID

6.3.22 goal_pnioCfgSubslotPortSet - 配置端口subslot

配置端口 subslot 基本 ID。

默认值：0x8001；返回 GOAL_STATUS_T 状态。

该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

警告：不应使用该函数，否则可能会破坏一致性。

表 6.30goal_pnioCfgSubslotPortSet 参数

参数	说明
unsigned intidSubslotPort	端口 subslot ID

6.3.23 goal_pnioCfgSnmpIdSet - 配置SNMP实例ID

配置 GOAL PROFINET 堆栈使用的 GOAL SNMP 实例 ID。默认值：GOAL_ID_INVALID

返回 GOAL_STATUS_T 状态。

该函数配置 GOAL PROFINET 实例，必须在 goal_pnioNew 生效之前调用该函数。

表 6.31goal_pnioCfgSnmpIdSet 参数

参数	说明
idSnmp	GOAL SNMP 实例 ID

6.3.24 goal_pnioSlotNew - 创建新slot

创建新 slot。返回 GOAL_STATUS_T 状态。

表 6.32goal_pnioSlotNew 参数

参数	说明
GOAL_PNIO_T *pPnio	PROFINET 数据
uint32_t idApi	API 号（始终为 0）
uint32_t idSlot	slot 号
GOAL_BOOL_T flgAutoGen	如果不存在，则自动生成 API

```
/* create API 0:Slot 1 even if API 0 doesn't exist */
res = goal_pnioSlotNew(pPnio, 0, 1, GOAL_TRUE);
```

6.3.25 goal_pnioSubslotNew - 创建新slot

创建新的 subslot。返回 GOAL_STATUS_T 状态。

表 6.33goal_pnioSubslotNew 参数

参数	说明
GOAL_PNIO_T *pPnio	PROFINET 数据
uint32_t idApi	API 号（始终为 0）
uint32_t idSlot	slot 号
uint32_t idSubslot	subslot 号
GOAL_BOOL_T flgAutoGen	如果不存在，则自动生成 API 和 slot

```
/* create API 0:Slot 1:Subslot 1 even if API 0 and/or * Slot 1 don't exist */
res = goal_pnioSubslotNew(pPnio, 0, 1, 1, GOAL_TRUE);
```

6.3.26 goal_pnioModNew - 创建新模块

创建新模块。返回 GOAL_STATUS_T 状态。

表 6.34goal_pnioModNew 参数

参数	说明
GOAL_PNIO_T *pPnio	PROFINET 数据
uint32_t idMod	module id

```
/* create module with id 0x30 */
res = goal_pnioModNew(pPnio, 0x30);
```

6.3.27 goal_pnioSubmodNew - 创建新的子模块

创建新的子模块。返回 GOAL_STATUS_T 状态。

表 6.35goal_pnioSubmodNew 参数

参数	说明
GOAL_PNIO_T *pPnio	PROFINET 数据
uint32_t idMod	module id
uint32_t idSubmod	submodule id
GOAL_PNIO_MOD_TYPE_T type	子模块类型（输入、输出、两者）
uint16_t sizeInput	输入模块的大小
uint16_t sizeOutput	输出模块的大小
GOAL_BOOL_T flgAutogen	如果不存在，则生成模块

```
/* create 4 byte input submodule with id 0x30:0x01 even if module * doesn't exist */
res = goal_pnioSubmodNew(pPnio, 0x30, 0x01, GOAL_PNIO_MOD_TYPE_INPUT, 4, 0,
GOAL_TRUE);
```

6.3.28 goal_pnioModPlug - 将模块插入slot

将模块插入 slot。返回 GOAL_STATUS_T 状态。

表 6.36goal_pnioModPlug 参数

参数	说明
GOAL_PNIO_T *pPnio	PROFINET 数据
uint32_t idApi	API（始终为 0）
uint16_t idSlot	slot 号
uint32_t idMod	module id

```
/* plug module 0x30 into slot 1 */
res = goal_pnioModPlug(pPnio, 0, 1, 0x30);
```

6.3.29 goal_pnioSubmodPlug - 将Submodule插入Subslot以进cyclic传输

将 Submodule 插入 Subslot 以进 cyclic 传输。返回 GOAL_STATUS_T 状态。

表 6.37goal_pnioSubmodPlug 参数

参数	说明
GOAL_PNIO_T *pPnio	PROFINET 数据
uint32_t idApi	API（始终为 0）
uint16_t idSlot	slot number
uint16_t idSubslot	subslot number
uint32_t idMod	module id
uint32_t idSubmod	submodule id

```
/* plug submodule 0x30:0x01 into subslot 1:1 */
res = goal_pnioSubmodPlug(pPnio, 0, 1, 1, 0x30, 0x01);
```

6.3.30 goal_pnioSubmodPlug - 将Submodule插入Subslot以进RPC传输

将 Submodule 插入 Subslot 以进 RPC 传输。返回 GOAL_STATUS_T 状态。

表 6.38goal_pnioRpcSubmodPlug 参数

参数	说明
GOAL_PNIO_T *pPnio	PROFINET 数据
uint32_t idApi	API（始终为 0）
uint16_t idSlot	slot number
uint16_t idSubslot	subslot number
uint32_t idMod	module id
uint32_t idSubmod	submodule id

```
/* plug submodule 0x30:0x01 into subslot 1:1 */
res = goal_pnioRpcSubmodPlug(pPnio, 0, 1, 1, 0x30, 0x01);
```

6.3.31 goal_pnioModPull - 从slot中拔出模块

从 slot 中拔出模块。返回 GOAL_STATUS_T 状态。

表 6.39goal_pnioModPull 参数

参数	说明
GOAL_PNIO_T *pPnio	PROFINET 数据
uint32_t idApi	API（始终为 0）
uint16_t idSlot	slot number

```
/* pull module from slot 1 */
res = goal_pnioModPull(pPnio, 0, 1);
```

6.3.32 goal_pnioSubmodPull - 从subslot中拔出子模块

从 subslot 中拔出子模块。返回 GOAL_STATUS_T 状态。

表 6.40goal_pnioSubmodPull 参数

参数	说明
GOAL_PNIO_T *pPnio	PROFINET 数据
uint32_t idApi	API（始终为 0）
uint16_t idSlot	slot number
uint16_t idSubslot	subslot number

```
/* pull submodule from subslot 1:1 */
res = goal_pnioSubmodPull(pPnio, 0, 1, 1);
```

6.3.33 goal_pnioDataOutputGet - 从子模块获取输出数据

从子模块获取输出数据。返回 GOAL_STATUS_T 状态。

表 6.41goal_pnioDataOutputGet 参数

参数	说明
GOAL_PNIO_T *pPnio	PROFINET 数据
uint32_t idApi	API（始终为 0）
uint16_t idSlot	slot number
uint16_t idSubslot	subslot number
char *pBuf	数据缓冲区
uint16_t len	数据缓冲区长度
GOAL_PNIO_IOXS_T *pStateIops	制作程序状态指标

```
GOAL_PNIO_IOXS_T iops = GOAL_PNIO_IOXS_GOOD;
/* get 4 bytes of output data for slot 1:1 */
res = goal_pnioDataOutputGet(pPnio, 0, 1, 1, &buf, 4, &iops);
```

6.3.34 goal_pnioDataInputSet - 设置子模块的输入数据

设置子模块的输入数据。返回 GOAL_STATUS_T 状态。

表 6.42goal_pnioDataInputSet 参数

参数	说明
GOAL_PNIO_T *pPnio	PROFINET 数据
uint32_t idApi	API（始终为 0）
uint16_t idSlot	slot number
uint16_t idSubslot	subslot number
char *pBuf	数据缓冲区
uint16_t len	数据缓冲区长度
GOAL_PNIO_IOXS_T statelops	制作程序状态

```
GOAL_PNIO_IOXS_T iops = GOAL_PNIO_IOXS_GOOD;
/* set 4 bytes of input data for slot 2:1 */
res = goal_pnioDataInputSet(pPnio, 0, 2, 1, &buf, 4, GOAL_PNIO_IOXS_GOOD);
```

6.3.35 goal_pnioApduStatusGet - 获取应用程序协议数据单元状态

获取应用程序协议数据单元状态。返回 GOAL_STATUS_T 状态。

表 6.43goal_pnioApduStatusGet 参数

参数	说明
GOAL_PNIO_T *pPnio	PROFINET 数据
GOAL_PNIO_AR_ID_T idAr	应用程序关系 ID
GOAL_PNIO_APDU_STATUS_T *pStatusApdu	APDU 状态指标

6.3.36 goal_pnioAlarmNotifySend - 发送报警通知

发送报警通知。返回 GOAL_STATUS_T 状态。

表 6.44goal_pnioAlarmNotifySend 参数

参数	说明
GOAL_PNIO_T *pPnio	PROFINET 数据
uint16_t *pNrAlarmSeq	存储报警序列号的指标
GOAL_PNIO_AR_ID_T idAr	应用程序关系 ID
uint32_t prio	优先级
const GOAL_PNIO_ALARM_ NOTIFY_T *pAlarmNotify	报警通知指标
uint16_t lenDataUser	用户数据长度
void *pDataUser	用户数据指标

6.3.37 goal_pnioAlarmNotifySendAck - 发送报警通知确认

发送报警通知确认。返回 GOAL_STATUS_T 状态。

表 6.45goal_pnioAlarmNotifySendAck 参数

参数	说明
GOAL_PNIO_T *pPnio	PROFINET 数据
GOAL_PNIO_AR_ID_T idAr	应用程序关系 ID
uint32_t prio	优先级
const GOAL_PNIO_ALARM_ NOTIFY_T *pAlarmNotify	报警通知指标
GOAL_PNIO_STATUS_T *pStatus	PROFINET 状态
uint16_t lenDataUser	用户数据长度
void *pDataUser	用户数据指标

6.3.38 goal_pnioAlarmProcessSend - 发送过程报警

发送过程报警。返回 GOAL_STATUS_T 状态。

表 6.46goal_pnioAlarmProcessSend 参数

参数	说明
GOAL_PNIO_T *pPnio	PROFINET 数据
uint32_t api	API
uint16_t slot	slot number
uint16_t subslot	subslot number
uint16_t usi	用户结构标识符
uint16_t lenData	数据长度
uint8_t *pData	用户数据指标

6.3.39 goal_pnioRecReadFinish - 回复记录读取请求

回复记录读取请求。序列号用于检测请求是否已过期。如果过期，响应将被静默丢弃。返回 GOAL_STATUS_T 状态。

表 6.47goal_pnioRecReadFinish 参数

参数	说明
GOAL_PNIO_T *pPnio	PROFINET 数据
int32_t hdl	记录句柄
GOAL_PNIO_STATUS_T *pStatusPnio	PROFINET 状态指标
const uint8_t *pData	记录数据
uint16_t len	记录数据长度
uint32_t numSeq	序列号

6.3.40 goal_pnioRecWriteFinish - 回复记录写入请求

回复记录写入请求。序列号用于检测请求是否已过期。如果过期，响应将被静默丢弃。返回 GOAL_STATUS_T 状态。

表 6.48goal_pnioRecWriteFinish 参数

参数	说明
GOAL_PNIO_T *pPnio	PROFINET 数据
int32_t hdl	记录句柄
GOAL_PNIO_STATUS_T *pStatusPnio	PROFINET 状态指标
uint32_t numSeq	序列号

6.3.41 goal_pnioDiagExtChanDiagAdd - 添加扩展通道诊断条目

添加扩展通道诊断条目。返回 GOAL_STATUS_T 状态。

表 6.49goal_pnioDiagExtChanDiagAdd 参数

参数	说明
GOAL_PNIO_T *pPnio	PROFINET 数据
GOAL_PNIO_DIAG_HANDLE_T *pHdl	存储诊断句柄的指标
uint32_t api	API
uint16_t slot	slot number
uint16_t subslot	subslot number
uint16_t chan	通道索引
uint16_t numErr	错误号
uint16_t typeExtChanErr	扩展通道错误类型
uint32_t valExtChanAdd	扩展通道错误值
GOAL_BOOL_T flgMaintReq	维护要求标志
GOAL_BOOL_T flgMaintDem	维护需求标志
GOAL_BOOL_T flgSubmitAlarm	提交报警标志
uint16_t typeAlarm	报警类型

6.3.42 goal_pnioDiagChanDiagRemove - 删除通道诊断条目

删除通道诊断条目。返回 GOAL_STATUS_T 状态。

表 6.50goal_pnioDiagChanDiagRemove 参数

参数	说明
GOAL_PNIO_T *pPnio	PROFINET 数据
GOAL_PNIO_DIAG_HANDLE_T hdl	诊断句柄
GOAL_BOOL_T flgSubmitAlarm	提交报警标志

6.3.43 goal_pnioCyclicCtrl - 控制周期性数据接收回调

控制周期性数据接收回调。如果 flgCyclic 设置为 GOAL_TRUE，则回调 GOAL_PNIO_CB_ID_NEW_IO_DATA 将指示接收到新的 I/O 数据。这将产生更高的系统负载。对于大多数应用程序，只需通过 goal_pnioDataOutputGet 轮询周期性数据即可。返回 GOAL_STATUS_T 状态。

表 6.51goal_pnioCyclicCtrl 参数

参数	说明
GOAL_PNIO_T *pPnio	PROFINET 数据
GOAL_BOOL_T flgCyclic	周期性数据启用标志

6.4 EtherNet/IP应用程序编程接口

6.4.1 goal_eipInit

初始化 GOAL EtherNet/IP。返回 GOAL_STATUS_T 作为结果。必须在函数 appl_init()中调用该函数。不需要参数。

要在 GOAL 中启用 EtherNet/IP 支持，必须在 appl_init 函数中调用 goal_eipInit 函数。

```
23. static GOAL_EIP_T *pHdlEip; /**< GOAL Ethernet/ IP handle */
23.  GOAL_STATUS_T appl_init( void
24.  )
25.  {
26.      GOAL_STATUS_T res;                  /* result */
27.
28.      res = goal_eipInit();
29.      if (GOAL_RES_ERR(res)) {
30.          goal_logErr("Initialization of Ethernet/IP failed");
31.      }
32.      return res;
33.  }
```

6.4.2 goal_eipNew

创建给定 ID 的 GOAL EtherNet/IP 实例并注册回调。必须在函数 appl_setup()中调用该函数。

表 6.52goal_eipNew 参数

参数	说明
GOAL_EIP_T ** ppEip	[输出]EtherNet/IP 实例参考
const uint32_t id	EtherNet/IP 实例 ID
GOAL_EIP_FUNC_CB_T pFunc	GOAL Ethernet/IP 句柄应用程序回调

```
res = goal_eipNew(&pHdlEip, EIP_INSTANCE_DEFAULT, main_eipCallback);
```

6.4.3 goal_eipCipClassRegister

注册应用程序特定的 CIP 类。当 EtherNet/IP 堆栈收到注册的 CIP 类的请求时，将传递给处理程序函数。

表 6.53goal_eipCipClassRegister 参数

参数	说明
GOAL_EIP_T *pHdlEip	GOAL EtherNet/IP 句柄
uint16_t classId	要注册的类 ID
GOAL_EIP_REQ_HANDLER_T pFunc	请求处理程序

res = goal_eipCipClassRegister(pEip, APPL_CLASS_ID_PARAMETER, appl_parameterClassHandler);

6.4.4 goal_eipCreateAssemblyObject

将 CIP 实例添加到所选类。返回 GOAL_STATUS_T 作为结果。

表 6.54goal_eipCreateAssemblyObject 参数

参数	说明
GOAL_EIP_T *pHdlEip	GOAL EtherNet/IP 句柄
uint32_t instanceId	要创建的程序集对象的实例号
uint16_t len	程序集对象的数据长度

res = goal_eipCreateAssemblyObject(pHdlEip, GOAL_APP_ASM_ID_INPUT,
GOAL_APP_ASM_SIZE_INPUT);

6.4.5 goal_eipCreateAssemblyObjectRpc

将用于 RCP 将 CIP 实例添加到所选类。返回 GOAL_STATUS_T 作为结果。

表 6.55 goal_eipCreateAssemblyObjectRpc 参数

参数	说明
GOAL_EIP_T *pHdlEip	GOAL EtherNet/IP 句柄
uint32_t instanceId	要创建的程序集对象的实例号
uint16_t len	程序集对象的数据长度

res = goal_eipCreateAssemblyObjectRpc(pHdlEip, GOAL_APP_ASM_ID_INPUT,
GOAL_APP_ASM_SIZE_INPUT);

6.4.6 goal_eipAssemblyObjectGet

获取程序集数据阵列的指标。

表 6.56goal_eipAssemblyObjectGet 参数

参数	说明
GOAL_EIP_T *pHdlEip	GOAL Ethernet/IP 句柄
uint32_t instanceId	程序集对象的实例号
uint8_t **ppData	[输出]程序集数据的指标
uint16_t *pLen	[输出]程序集数据的长度

```
uint8_t *pData; /* assembly data */
uint32_t len;  /* assembly data length */

res = goal_eipAssemblyObjectGet(pHdlEip, GOAL_APP_ASM_ID_INPUT, &pData, &len);
```

6.4.7 goal_eipAddExclusiveOwnerConnection

创建具有生成和使用端点的独占所有者连接。

表 6.57goal_eipAddExclusiveOwnerConnection 参数

参数	说明
GOAL_EIP_T *pHdlEip	GOAL EtherNet/IP 句柄
uint32_t outputAssembly	用于该连接的 O-to-T 点
uint32_t inputAssembly	用于该连接的 T-to-O 点
uint32_t configAssembly	用于该连接的配置点

```
res = goal_eipAddExclusiveOwnerConnection(pHdlEip,GOAL_APP_ASM_ID_OUTPUT,
      GOAL_APP_ASM_ID_INPUT, GOAL_APP_ASM_ID_CONFIG);
```

6.4.8 goal_eipAddInputOnlyConnection

创建具有相同生成和使用端点的多个仅输入连接。

表 6.58goal_eipAddInputOnlyConnection 参数

参数	说明
GOAL_EIP_T *pHdlEip	GOAL EtherNet/IP 句柄
uint32_t connNum	仅输入连接的数目
uint32_t outputAssembly	用于该连接的 O-to-T 点
uint32_t inputAssembly	用于该连接的 T-to-O 点
uint32_t configAssembly	用于该连接的配置点

```
res = goal_eipAddInputOnlyConnection(pHdlEip, GOAL_APP_IOCON_NUM,
GOAL_APP_ASM_ID_HEARTBEAT_IO, GOAL_APP_ASM_ID_INPUT, GOAL_APP_ASM_ID_CONFIG);
```

6.4.9 goal_eipAddListenOnlyConnection

创建具有相同生成和使用端点的多个仅监听连接。

表 6.59goal_eipAddListenOnlyConnection 参数

参数	说明
GOAL_EIP_T *pHdlEip	GOAL EtherNet/IP 句柄
uint32_t connNum	仅监听连接的数目
uint32_t outputAssembly	用于该连接的 O-to-T 点
uint32_t inputAssembly	用于该连接的 T-to-O 点
uint32_t configAssembly	用于该连接的配置点

```
res = goal_eipAddListenOnlyConnection(pHdlEip, GOAL_APP_LOCON_NUM,
      GOAL_APP_ASM_ID_HEARTBEAT_LO, GOAL_APP_ASM_ID_INPUT,
      GOAL_APP_ASM_ID_CONFIG);
```

6.4.10 goal_eipGetVersion

获取 EtherNet/IP 版本。返回 GOAL_STATUS_T 作为结果。

表 6.60goal_eipGetVersion 参数

参数	说明
GOAL_EIP_T *pHdlEip	GOAL EtherNet/IP 句柄
char **ppVersion	[输出] Ethernet/IP 版本

```
char *pVersion; /* version string */
goal_eipGetVersion(pHdlEip, &pVersion);
goal_logInfo("EtherNet/IP stack version: %s", pVersion);
```

6.4.11 goal_eipDeviceStatusSet

设置设备状态位。参数 statusBits 使用 GOAL_EIP_STATUS_宏。可以与二进制 OR 组合。

表 6.61goal_eipDeviceStatusSet 参数

参数	说明
GOAL_EIP_T *pHdlEip	GOAL EtherNet/IP 句柄
uint16_t statusBits	状态位图

```
res = goal_eipDeviceStatusSet(pHdlEip, GOAL_EIP_STATUS_CFGRD);
```

6.4.12 goal_eipDeviceStatusClear

清除设备状态位。将清除 statusBits 的所有位。参数使用 GOAL_EIP_STATUS_宏。可以与二进制 OR 组合。

表 6.62goal_eipDeviceStatusClear 参数

参数	说明
GOAL_EIP_T *pHdlEip	GOAL EtherNet/IP 句柄
uint16_t statusBits	状态位图

```
res = goal_eipDeviceStatusClear(pHdlEip,GOAL_EIP_STATUS_CFGRD);
```

6.4.13 goal_eipAssemblyObjectWrite

将数据写入集合对象。

表 6.63 goal_eipAssemblyObjectWrite 参数

参数	说明
GOAL_EIP_T *pHdlEip	GOAL 以太网/IP 处理
uint32_t instanceId	集合对象实例 ID
uint8_t *pData	缓冲区，带有写入数据
uint16_t len	要写入的字节数

```
res = goal_goal_eipAssemblyObjectWrite (pHdlEip, 100, pData, 32);
```

6.4.14 goal_eipAssemblyObjectRead

从集合对象处读取数据。

表 6.64 goal_eipAssemblyObjectRead 参数

参数	说明
GOAL_EIP_T *pHdlEip	GOAL 以太网/IP 处理
uint32_t instanceId	集合对象实例 ID
uint8_t *pData	缓冲区，用于读取数据
uint16_t len	要读取的字节数

```
res = goal_eipAssemblyObjectRead(pHdlEip, 150, pRdBuf, 32);
```

6.4.15 goal_eipIdentitySerialNumberSet

设定设备的序列号。每种设备必须具有一个唯一的序列号。该函数更新识别编号实例属性，即：在创建 EtherNet/IP 实例后，其变更序列号。

表 6.62goal_eipIdentitySerialNumberSet 参数

参数	说明
GOAL_EIP_T *pHdlEip	GOAL 以太网/IP 处理
uint32_t serialNum	新序列号

```
res = goal_eipIdentitySerialNumberSet (pHdlEip, 12345);
```

6.5 EtherCAT 应用程序设计接口

6.5.1 goal_ecatInit

目的： **EtherCAT** 库首次注册

函数原型：

```
GOAL_STATUS_T goal_ecatInit(
  void
);
```

示例：

```
/*****************************************************************************/
/** Application Init *
* This function must initialize GOAL and all used protocol stacks.
* Furthermore application specific resources must be initialized.
*/
GOAL_STATUS_T appl_init(
  void
)
{
  GOAL_STATUS_T res;            /* result */
  /* initialize EtherCAT */
  res = goal_ecatInit();
  if (GOAL_RES_ERR(res)) {
    goal_logErr("Initialization of EtherCAT failed");
  }
  return res;
}
```

6.5.2 goal_ecatNew

目的：创建 EtherCAT 库实例

函数原型：

```
GOAL_STATUS_T goal_ecatNew(
    GOAL_ECAT_T **ppEcat,                   /**< [out] EtherCAT instance ref */
    const uint32_t id,                      /**< instance id */
    GOAL_ECAT_FUNC_CB_T pFunc               /**< EtherCAT callback function */
);
```

示例：

```
/*****************************************************************************/
/* Local variables */
/*****************************************************************************/
static GOAL_ECAT_T *pHdlEcat;                   /**< GOAL EtherCAT handle */


/*****************************************************************************/
/** Application Setup
 *
 * Setup the application.
 */
GOAL_STATUS_T appl_setup(
    void
)
{
    GOAL_STATUS_T res;                          /* result */
    goal_logInfo("create new instance");

    res = goal_ecatNew(&pHdlEcat, GOAL_ECAT_INSTANCE_DEFAULT, appl_ecatCallback);
    if (GOAL_RES_ERR(res)) {
        goal_logErr("failed to create a new EtherCAT instance");
        return res;
    }

    return res;
}
```

6.5.3 goal_ecatCfgEmergencyOn

目的： 启用紧急消息支持

函数原型：

```
1.  GOAL_STATUS_T goal_ecatCfgEmergencyOn(
2.      GOAL_BOOL_T enable                          /**< enable or disable feature */
3.  );
```

示例：

```
1.  /* enable CoE emergency */
2.  res = goal_ecatCfgEmergencyOn(GOAL_TRUE);
3.  if (GOAL_RES_ERR(res)) {
4.      goal_logErr("failed to enable CoE Emergency support");
5.  }
```

备注：
应在 goal_ecatNew 前，调用该函数。

6.5.4 goal_ecatCfgEmergencyQueueNum

目的： 设定可排列紧急消息的数目

函数原型：

```
1.  GOAL_STATUS_T goal_ecatCfgEmergencyQueueNum(
2.      int16_t num                                 /**< number of queue slots */
3.  );
```

示例：

```
1.  /* set emergency queue size to 8 */
2.  res = goal_ecatCfgEmergencyQueueNum(8);
3.  if(GOAL_RES_ERR(res)) {
4.      goal_logErr("failed to set CoE Emergency Queue size to 8");
5.  }
```

备注：
应在 goal_ecatNew 前，调用该函数。

6.5.5 goal_ecatCfgFoeOn

目的： 启用 FoE 支持

函数原型：

```
1.  GOAL_STATUS_T goal_ecatCfgFoeOn(
2.      GOAL_BOOL_T enable                  /**< enable or disable feature */
3.  );
```

示例：

```
1.  /* enable FoE */
2.  res = goal_ecatCfgFoeOn(GOAL_TRUE);
3.  if (GOAL_RES_ERR(res)) {
4.      goal_logErr("failed to enable FoE support");
5.  }
```

备注：
应在 goal_ecatNew 前，调用该函数。

6.5.6 goal_ecatCfgExplDevIdOn

目的：启用明确设备识别编号支持

函数原型：

```
1.  GOAL_STATUS_T goal_ecatCfgExplDevIdOn(
2.      GOAL_BOOL_T enable                  /**< enable or disable feature */
3.  );
```

示例：

```
1.  /* enable Explicit Device Identification */
2.  res = goal_ecatCfgExplDevIdOn(GOAL_TRUE);
3.  if (GOAL_RES_ERR(res)) {
4.      goal_logErr("failed to enable Explicit Device Identification support");
5.  }
```

备注：
应在 goal_ecatNew 前，调用该函数。

6.5.7 goal_ecatCfgBootstrapOn

目的：启用 BOOTSTRAP ESM 状态支持

函数原型：

```
);GOAL_STATUS_T goal_ecatCfgBootstrapOn(
    GOAL_BOOL_T enable                          /**< enable or disable feature */
);
```

示例：

```
/* enable BOOTSTRAP state */
res = goal_ecatCfgBootstrapOn(GOAL_TRUE);
if(GOAL_RES_ERR(res)) {
    goal_logErr("failed to enable BOOTSTRAP support");
}
```

备注：
应在 goal_ecatNew 前，调用该函数。

6.5.8 goal_ecatCfgDcRequiredOn

目的： 启用 DC 模式执行

函数原型：

```
);GOAL_STATUS_T goal_ecatCfgDcRequiredOn(
    GOAL_BOOL_T enable                          /**< enable or disable feature */
);
```

示例：

```
/* enable DC mode only */
res = goal_ecatCfgDcRequiredOn(GOAL_TRUE);
if(GOAL_RES_ERR(res)) {
    goal_logErr("failed to enforce DC mode");
}
```

备注：
应在 goal_ecatNew 前，调用该函数。

6.5.9 goal_ecatCfgSizePdoStreamBufSet

目的： 配置 PDO 数据缓冲区

函数原型：

```
GOAL_STATUS_T goal_ecatCfgSizePdoStreamBufSet(
    uint16_t size                                   /**< size of PDO stream buffer */
);
```

示例：

```
/* set size of PDO buffer
 *
 * The buffer must be big enough to hold both the input and output data.
 * The size depends on the possible combinations of mappable objects.
 */
res = goal_ecatCfgSizePdoStreamBufSet(APPL_ECAT_PDO_BUF_SIZE);
if (GOAL_RES_ERR(res)) {
    goal_logErr("failed to set PDO buffer size to %d", APPL_ECAT_PDO_BUF_SIZE);
}
```

备注：

应在 goal_ecatNew 前，调用该函数。

为实现 CC 最大程序数据长度，应将缓冲区配置为 2 * 68 = 136 字节。

6.5.10 配置指示灯模拟行为

目的：配置指示灯模拟行为

函数原型：

```
GOAL_STATUS_T goal_ecatCfgLedStatusIndicator(
    GOAL_BOOL_T enable                              /**< enable or disable feature */
);
```

示例：

```
/* configure dual LED status indicator */
res = goal_ecatCfgLedStatusIndicator(GOAL_FALSE);
if(GOAL_RES_ERR(res)) {
    goal_logErr("failed to configure LED status indicator");
}
```

备注：

应在 goal_ecatNew 前，调用该函数。

如果设定为 FALSE，指示灯模拟将支持专用 RUN 和 ERROR 指示灯。如果设定为 TRUE，指示灯模拟将支持联合 STATUS 指示灯。

6.5.11 goal_ecatObjAddrGet

目的：获得对象值指标。

函数原型：

```
GOAL_STATUS_T goal_ecatObjAddrGet(
    GOAL_ECAT_T *pEcat,        /**< EtherCAT instance data */
    uint16_t index,      /**< object index */
    uint8_t subIndex,    /**< object sub-index */
    uint8_t **ppData,    /**< [out] object data pointer reference */
    uint32_t *pSize      /**< [out] object size reference */
);
```

示例：

```
GOAL_STATUS_T res;                          /* return */
uint8_t errorRegister;                      /* variable */
uint32_t size;                              /* variable size */
res = goal_ecatObjAddrGet(pEcat, 0x1001, 0x0, &errorRegister, &size);
if (GOAL_RES_OK(res)) {
    goal_logInfo("error register contains %d" errorRegister);
}
```

备注：
无。

6.5.12 goal_ecatObjValGet

目的：获取对象值

函数原型：

```
1.  GOAL_STATUS_T goal_ecatObjValGet(
2.      GOAL_ECAT_T *pEcat,       /**< EtherCAT instance data */
3.      uint16_t index,      /**< object index */
4.      uint8_t subIndex,    /**< object sub-index */
5.      uint8_t *pData,      /**< [out] object data buffer */
6.      uint32_t *pSize      /**< [out] object size reference */
7.  );
```

示例：

```
1.  GOAL_STATUS_T res;                          /* return */
2.  uint32_t value = 0;                         /* variable value */
3.  uint32_t size = 0;                          /* variable size */
4.
5.  res = goal_ecatObjValGet(pHdlEcat, 0x1001, 0, (uint8_t *) &value, &size);
6.  if(GOAL_RES_ERR(res)) {
7.   return res;
8.  }
```

备注：

无。

6.5.13 goal_ecatObjValSet

目的： 设定对象值

函数原型：

```
1.  GOAL_STATUS_T goal_ecatObjValSet(
2.      GOAL_ECAT_T *pEcat,         /**< EtherCAT instance data */
3.      uint16_t index,             /**< object index */
4.      uint8_t subIndex,           /**< object sub-index */
5.      uint8_t *pData,             /**< [in] new object value */
6.      uint32_t size               /**< size of new data */
7.  );
```

示例：

```
1.  res = goal_ecatObjValSet(
2.          pHdlEcat,
3.          0x1008,
4.          0x00,
5.          (uint8_t *) APPL_ECAT_PRODUCT_NAME,
6.          GOAL_STRLEN(APPL_ECAT_PRODUCT_NAME));
7.  if (GOAL_RES_ERR(res)) {
8.      goal_logErr("failed to set product name in object dictionary");
9.   return res;
10. }
```

备注：
无。

6.5.14 goal_ecatdynOdObjAdd

目的： 添加对象

函数原型：

```
1.  GOAL_STATUS_T goal_ecatdynOdObjAdd(
2.      GOAL_ECAT_T *pEcat,         /**< EtherCAT instance data */
3.      uint16_t index,             /**< object index */
4.      uint16_t objTpye,           /**< object type (VAR, ARRAY, RECORD) */
5.      uint16_t dataType           /**< data type of object */
6.  );
```

示例：

```
1.  res = goal_ecatdynOdObjAdd(
2.      pHdlEcat,
3.      0x1000,
4.      GOAL_ECAT_OBJCODE_VAR,
5.      GOAL_ECAT_DATATYPE_UNSIGNED32);
```

备注：
无。

6.5.15 goal_ecatdynOdSubIndexAdd

目的：添加对象子索引 cyclic 传输

函数原型：

```
GOAL_STATUS_T goal_ecatdynOdSubIndexAdd(
    GOAL_ECAT_T *pEcat,                 /**< EtherCAT instance data */
    uint16_t index,                     /**< object index */
    uint8_t subIndex,                   /**< object sub-index */
    uint16_t dataType,                  /**< data type of object */
    uint16_t attr,                      /**< attributes (R/W/mappable,...) */
    uint8_t *pDefVal,                   /**< default value of entry */
    uint8_t *pMinVal,                   /**< minimum value of entry */
    uint8_t *pMaxVal,                   /**< maximum value of entry */
    uint32_t size,                      /**< size of non-numerical objects */
    uint8_t *pVar                       /**< application variable */
);
```

示例：

```
uint32ValueMin = 0x0;
uint32ValueDef = 0x00030191;
uint32ValueMax = 0xFFFFFFFF;

res = goal_ecatdynOdSubIndexAdd(
    pHdlEcat,
    0x1000,
    0x00,
    GOAL_ECAT_DATATYPE_UNSIGNED32,
    EC_OBJATTR_RD | EC_OBJATTR_MAN | EC_OBJATTR_NUMERIC | EC_OBJATTR_NO_LIMITS,
    (uint8_t *) &uint32ValueDef,
    (uint8_t *) &uint32ValueMin,
    (uint8_t *) &uint32ValueMax,
    4,
    NULL);
```

备注：

无。

6.5.16 goal_ecatdynOdSubIndexRpcAdd

目的：添加对象子索引 RPC 传输

函数原型：

```
1.  GOAL_STATUS_T goal_ecatdynOdSubIndexRpcAdd(
2.      GOAL_ECAT_T *pEcat,                 /**< EtherCAT instance data */
3.      uint16_t index,                     /**< object index */
4.      uint8_t subIndex,                   /**< object sub-index */
5.      uint16_t dataType,                  /**< data type of object */
6.      uint16_t attr,                      /**< attributes (R/W/mappable,...) */
7.      uint8_t *pDefVal,                   /**< default value of entry */
8.      uint8_t *pMinVal,                   /**< minimum value of entry */
9.      uint8_t *pMaxVal,                   /**< maximum value of entry */
10.     uint32_t size,                      /**< size of non-numerical objects */
11.     uint8_t *pVar                       /**< application variable */
12. );
```

示例：

```
1.  uint32ValueMin = 0x0;
2.  uint32ValueDef = 0x00030191;
3.  uint32ValueMax = 0xFFFFFFFF;
4.
5.  res = goal_ecatdynOdSubIndexRpcAdd(
6.      pHdlEcat,
7.      0x1000,
8.      0x00,
9.      GOAL_ECAT_DATATYPE_UNSIGNED32,
10.     EC_OBJATTR_RD | EC_OBJATTR_MAN | EC_OBJATTR_NUMERIC | EC_OBJATTR_NO_LIMITS,
11.     (uint8_t *) &uint32ValueDef,
12.     (uint8_t *) &uint32ValueMin,
13.     (uint8_t *) &uint32ValueMax,
14.     4,
15.     NULL);
```

备注：

无。

6.5.17 goal_ecatdynOdObjNameAdd

目的： 分配对象名称

函数原型：

```
1.  GOAL_STATUS_T goal_ecatdynOdObjNameAdd(
2.      GOAL_ECAT_T *pEcat,                 /**< EtherCAT instance data */
3.      uint16_t index,                     /**< object index */
4.      char *pName                         /**< name of object */
5.  );
```

示例：

```
1.  res = goal_ecatdynOdObjNameAdd(pHdlEcat, 0x1000, "Device Type");
```

备注：
无。

6.5.18 goal_ecatdynOdSubIndexNameAdd

目的： 分配子索引名称

函数原型：

```
1.  GOAL_STATUS_T goal_ecatdynOdSubIndexNameAdd(
2.      GOAL_ECAT_T *pEcat,                 /**< EtherCAT instance data */
3.      uint16_t index,                     /**< object index */
4.      uint8_t subIndex,                   /**< object sub-index */
5.      char *pName                         /**< name of sub-index */
6.  );
```

示例：

```
1.  res = goal_ecatdynOdSubIndexNameAdd(pHdlEcat, 0x1018, 1, "Vendor Id");
```

备注：
无。

6.5.19 goal_ecatdynOdFinish

目的： 结束对象创建

函数原型：

```
GOAL_STATUS_T goal_ecatdynOdFinish(
    GOAL_ECAT_T *pEcat                          /**< EtherCAT instance data */
);
```

示例：

```
/* finish object dictionary creation */
If(GOAL_RES_OK(res)) {
    res = goal_ecatdynOdFinish(pHdlEcat);
}
```

备注：
该函数终止对象动态创建。当创建最后一个对象时，必须调用该函数。在调用该函数后，不会创建附加对象。

6.5.20 goal_ecatEsmStateGet

目的：获取现行 ESM 状态

函数原型：

```
GOAL_STATUS_T goal_ecatEsmStateGet(
    GOAL_ECAT_T *pEcat,                         /**< EtherCAT instance data */
    uint8_t *pState                             /**< [out] esm state */
);
```

示例：

```
GOAL_STATUS_T res;                              /* return */
uint8_t state;                                  /* EtherCAT state */

/* check if state is BOOTSTRAP */
res = goal_ecatEsmStateGet(pHdlEcat, &state);
if (GOAL_RES_OK(res) && GOAL_ECAT_ESM_STATE_BOOTSTRAP == state) {
}
```

备注：
无。

6.5.21 goal_ecatEmcyReqWrite

目的：生成适用于传输的紧急消息

函数原型：

```
GOAL_STATUS_T goal_ecatEmcyReqWrite(
    GOAL_ECAT_T *pEcat,                 /**< EtherCAT instance data */
    uint16_t errorCode,                 /**< standard error code */
    uint8_t *pManuErr                   /**< manufacturer error code */
);
```

示例：

```
GOAL_STATUS_T res;                      /* result */
uint8_t errData[5];                     /* dummy error data */

errData[0] = 0x01;
errData[1] = 0x02;
errData[2] = 0x03;
errData[3] = 0x04;
errData[4] = 0x05;

res = goal_ecatEmcyReqWrite(
    pEcat,
    APPL_ECAT_EMCY_DEMO_ERROR_CODE,
    errData);
```

备注：
无。

6.5.22 goal_ecatGetVersion

目的： 获取与 EtherCAT 库相关的版本信息

函数原型：

```
1.  GOAL_STATUS_T goal_ecatGetVersion(
2.      char *strVersion,                   /**< [out] stack version */
3.      int size                            /**< buffer size */
4.  );
```

示例：

```
1.  char version[10];
2.  res = goal_ecatGetVersion(version, 10);
```

备注：
无。

6.5.23 goal_ecatEsmLocalErrorSet

目的： 设置来自应用程序的错误

函数原型：

```
1.  GOAL_STATUS_T goal_ecatEsmLocalErrorSet(
2.      GOAL_ECAT_T *pEcat,         /**< Ethercat instance data */
3.      uint16_t errorCode          /**< new AL status code */
4.  );
```

示例：

```
1.  /* set local error */
2.  goal_ecatEsmLocalErrorSet(pEcat, 0x8000);
```

备注：
无。

6.6 网络

6.6.1 goal_netRpcInit - RPC功能网络初始化

目的：如果需要网络功能（IP 设置、UDP 通道、TCP 通道），则需要在应用程序初始化期间调用该函数。

函数原型：

```
GOAL_STATUS_T goal_netRpcInit(
    uint32_t id                         /**< id for MA instance */
)
```

示例：

```
/*****************************************************************************/
/** Application Init
 *
 * Initialize the net stack
 */
GOAL_STATUS_T appl_init(
    void
)
{
    GOAL_STATUS_T res;                  /* result */

    /* initialize goal net */
    res = goal_netRpcInit(GOAL_NET_ID_DEFAULT);
    if (GOAL_RES_ERR(res)) {
        goal_logErr("Initialization of goal net RPC failed");
    }

    return res;
}
```

6.6.2 goal_maNetOpen - 打开网络

目的：打开网络媒介适配器(MA)以供使用。

函数原型：

```
GOAL_STATUS_T goal_maNetOpen(
    uint32_t id,                    /**< id of NET handler to use */
    GOAL_MA_NET_T **ppNetHdl        /**< pointer to store NET handler */
);
```

示例：

```
/*****************************************************************************/
/** Application Setup
 *
 * This function is called by the GOAL init-stage system to open UDP channels.
 *
 * API functions from earlier stages are allowed to be used here.
 */
GOAL_STATUS_T appl_setup(
    void
    )
    {
        GOAL_STATUS_T res;              /* result */
        GOAL_MA_NET_T *pMaNet;              /* net ma handle */

        res = goal_maNetOpen(GOAL_NET_ID_DEFAULT, &pMaNet);
        if (GOAL_RES_ERR(res)) {
            goal_logErr("error opening network MA");
        }

        return res;
    }
```

6.6.3 goal_maNetClose - 关闭网络

目的：关闭网络媒介适配器(MA)。

函数原型：

```
GOAL_STATUS_T goal_maNetClose(
    GOAL_MA_NET_T *pNetHdl          /**< pointer to store NET handler */
);
```

6.6.4 goal_maNetGetById - 获取网络媒介适配器 (MA) 句柄

目的：
获取先前打开以供使用的网络媒介适配器(MA)句柄

函数原型：

```
GOAL_STATUS_T goal_maNetGetById(
    GOAL_MA_NET_T **ppHdlMaNet,        /**< NET handle ref ptr */
    uint32_t id                        /**< MA id */
);
```

示例：

```
res = goal_maNetGetById(&pMaNet, GOAL_NET_ID_DEFAULT);
if (GOAL_RES_ERR(res)) {
    goal_logErr("error getting network MA");
}
```

6.6.5 goal_maNetIpSet - 设置 IP 地址

目的：设置网络接口 IP 地址

表 6.65goal_maNetIpSet 参数

参数	说明
GOAL_MA_NET_T *pNetHdl	GOAL NET 句柄
uint32_t addrIp	IP 地址
uint32_t addrMask	网络掩码
uint32_t addrGw	网关
GOAL_BOOT_T flgTemp	GOAL_TRUE ：保持变量“VALID”的当前设置 GOAL_FALSE ：将变量“VALID”设置为 1

函数原型：

```
GOAL_STATUS_T goal_maNetIpSet(
    GOAL_MA_NET_T *pNetHdl,        /**< pointer to store NET handler */
    uint32_t addrIp,               /**< IP address */
    uint32_t addrMask,             /**< subnet mask */
    uint32_t addrGw,               /**< gateway */
    GOAL_BOOL_T flgTemp            /**< temporary IP config flag */
);
```

示例：

```
ip = MAIN_APPL_IP;
nm = MAIN_APPL_NM;
gw = MAIN_APPL_GW;
res = goal_maNetIpSet(pMaNet, ip, nm, gw, GOAL_FALSE);
if (GOAL_RES_ERR(res)) {
    goal_logErr("error while setting IP address");
    return res;
}
```

6.7 TCP通道

6.7.1 goal_maChanTcpOpen - 打开 TCP 通道媒介适配器 (MA)

目的：打开网络媒介适配器(MA)以供进一步使用。需要在应用程序启动时执行一次。

函数原型：

```
GOAL_STATUS_T goal_maChanTcpOpen(
    uint32_t id,                                /**< MA id */
    GOAL_MA_CHAN_TCP_T **ppHdlMaChanTcp         /**< CHAN_TCP handle ref ptr */
);
```

示例：

```
res = goal_maChanTcpOpen(GOAL_NET_ID_DEFAULT, &pMaTcp);
if (GOAL_RES_ERR(res)) {
    goal_logErr("error getting tcp MA");
    return res;
}
```

6.7.2 goal_maChanTcpNew - 创建新的 TCP 通道

目的：该函数创建新的 TCP 通道。

函数原型：

```
GOAL_STATUS_T goal_maChanTcpNew(
    GOAL_MA_CHAN_TCP_T *pChanTcpHdl,            /**< pointer to store CHAN_TCP handler */
    GOAL_NET_CHAN_T **ppChanHandle,             /**< pointer to channel handle */
    GOAL_NET_CHAN_T *pChanOut,                  /**< pointer to channel handle for output */
    GOAL_NET_ADDR_T *pAddr,                     /**< remote address */
    GOAL_NET_TYPE_T type,                       /**< connection type */
    GOAL_MA_CHAN_TCP_CB_T callback              /**< channel callback */
);
```

示例：

```
/* register TCP server */
GOAL_MEMSET(&addr, 0, sizeof(GOAL_NET_ADDR_T));
addr.localPort = (uint16_t) (MAIN_APPL_TCP_PORT + cnt);
res = goal_maChanTcpNew(pMaTcp, &pChan, NULL, &addr, GOAL_NET_TCP_LISTENER, tcpCallback);
if (GOAL_OK != res) {
    goal_logErr("error while opening TCP server channel on port %"FMT_u32,
                (uint32_t) MAIN_APPL_TCP_PORT + cnt);
    return res;
}
```

6.7.3 goal_maChanTcpActive - 激活创建的 TCP 通道

目的：激活先前创建的 TCP 通道。激活后，建立连接或接受传入的连接请求。

函数原型：

```
GOAL_STATUS_T goal_maChanTcpActivate(
    GOAL_MA_CHAN_TCP_T *pChanTcpHdl,          /**< pointer to store CHAN_TCP handler */
    GOAL_NET_CHAN_T *pChanHandle              /**< channel handle */
);
```

示例：

```
/* activate channel */
res = goal_maChanTcpActivate(pMaTcp, pChanTcp);
if (GOAL_OK != res) {
    goal_logErr("error while enabling TCP channel");
    return;
}
```

6.7.4 goal_maChanTcpSetNonBlocking - 将通道设置为非阻塞

目的：将套接字设置为非阻塞模式以进行读取

函数原型：

```
GOAL_STATUS_T goal_maChanTcpSetNonBlocking(
    GOAL_MA_CHAN_TCP_T *pChanTcpHdl,          /**< pointer to store CHAN_TCP handler */
    GOAL_NET_CHAN_T *pChanHandle,             /**< channel handle */
    GOAL_BOOL_T flgOption                     /**< non blocking state */
);
```

示例：

```
/* set TCP channel to non-blocking */
res = goal_maChanTcpSetNonBlocking(pMaTcp, pChanTcp, GOAL_TRUE);
if (GOAL_OK != res) {
    goal_logErr("error while setting TCP channel to non-blocking");
    return;
}
```

6.7.5 goal_maChanTcpGetRemoteAddr - 获取 TCP 通道的远程地址

目的：获取 TCP 通道远程端点的 IP 地址。

函数原型：

```
GOAL_STATUS_T goal_maChanTcpGetRemoteAddr(
    GOAL_MA_CHAN_TCP_T *pChanTcpHdl,        /**< pointer to store CHAN_TCP handler */
    GOAL_NET_CHAN_T *pChanHandle,           /**< channel handle */
    GOAL_NET_ADDR_T *pAddr                  /**< remote address */
);
```

示例：

```
/* get IP Address of remote node */
res = goal_maChanTcpGetRemoteAddr(pMaTcpHdl, pChan, &remote);
if (GOAL_RES_ERR(res)) {
    goal_logErr("Failed to get Remote Address for socket %p", (void *) pChan);
    return;
}
```

6.7.6 goal_maChanTcpSend - 通过 TCP 通道发送数据

目的：将数据发送到先前打开的 TCP 通道。

函数原型：

```
GOAL_STATUS_T goal_maChanTcpSend(
    GOAL_MA_CHAN_TCP_T *pChanTcpHdl,        /**< pointer to store CHAN_TCP handler */
    GOAL_NET_CHAN_T *pChanHandle,           /**< channel handle */
    GOAL_BUFFER_T *pBuf                     /**< buffer with data to send */
);
```

示例：

```
/* echo message */
goal_maChanTcpSend(pMaTcpHdl, pChan, pBuf);
```

6.8 UDP通道

6.8.1 goal_maChanUdpOpen - 打开 UDP 通道MA

目的：
打开 UDP 通道 MA。需要在应用程序启动时执行一次。

函数原型：

```
GOAL_STATUS_T goal_maChanUdpOpen(
    uint32_t id,                                    /**< MA id */
    GOAL_MA_CHAN_UDP_T **ppHdlMaChanUdp             /**< CHAN_UDP handle ref ptr */
);
```

示例：

```
/* open UDP channel MA and create new channel */
res = goal_maChanUdpOpen(GOAL_NET_ID_DEFAULT, &pMaChanUdp);
if (GOAL_RES_OK(res)) {
    GOAL_MEMSET(&addr, 0, sizeof(GOAL_NET_ADDR_T));
    addr.localPort = MAIN_APPL_UDP_PORT_1;
    res = goal_maChanUdpNew(pMaChanUdp, &pChan1, NULL, &addr, GOAL_NET_UDP_SERVER,
                            udpCallback);
}
```

6.8.2 goal_maChanUdpGetById - 获取 UDP 通道 MA 句柄

目的：该函数获取先前打开的 UDP 通道 MA 的句柄。

函数原型：

```
GOAL_STATUS_T goal_maChanUdpGetById(
    GOAL_MA_CHAN_UDP_T **ppHdlMaChanUdp,            /**< CHAN_UDP handle ref ptr */
    uint32_t id                                     /**< MA id */
);
```

6.8.3 goal_maChanUdpNew - 创建新的 UDP 通道

目的：创建新的 UDP 通道。

函数原型：

```
GOAL_STATUS_T goal_maChanUdpNew(
    GOAL_MA_CHAN_UDP_T *pChanUdpHdl,          /**< pointer to store CHAN_UDP handler */
    GOAL_NET_CHAN_T **ppChanHandle,           /**< pointer to channel handle */
    GOAL_NET_CHAN_T *pChanOut,                /**< pointer to channel handle for output */
    GOAL_NET_ADDR_T *pAddr,                   /**< remote address */
    GOAL_NET_TYPE_T type,                     /**< connection type */
    GOAL_MA_CHAN_UDP_CB_T callback            /**< channel callback */
);
```

示例：

```
if (GOAL_RES_OK(res)) {
    GOAL_MEMSET(&addr, 0, sizeof(GOAL_NET_ADDR_T));
    addr.localPort = MAIN_APPL_UDP_PORT_1;
    res = goal_maChanUdpNew(pMaChanUdp, &pChan1, NULL, &addr, GOAL_NET_UDP_SERVER,
                            udpCallback);
 }
```

6.8.4 goal_maChanUdpClose - 关闭 UDP 通道 MA

目的：关闭现有通道

函数原型：

```
GOAL_STATUS_T goal_maChanUdpClose(
    GOAL_MA_CHAN_UDP_T *pChanUdpHdl,          /**< pointer to store CHAN_UDP handler */
    GOAL_NET_CHAN_T *pChanHandle              /**< pointer to channel handle */
);
```

6.8.5 goal_maChanUdpSetNonBlocking - 将打开的通道设置为非阻塞访问

目的：将通道设置为非阻塞操作。

函数原型：

```
GOAL_STATUS_T goal_maChanUdpSetNonBlocking(
    GOAL_MA_CHAN_UDP_T *pChanUdpHdl,        /**< pointer to store CHAN_UDP handler */
    GOAL_NET_CHAN_T *pChanHandle,           /**< channel handle */
    GOAL_BOOL_T flgOption                   /**< option value */
);
```

示例：

```
res = goal_maChanUdpSetNonBlocking(pMaChanUdp, pChan2, GOAL_TRUE);
if (GOAL_OK != res) {
    goal_logErr("error while setting UDP channel to non-blocking");
    return res;
}
```

6.8.6 goal_maChanUdpSetBroadcast - 将打开的 UDP 通道设置为广播操作

目的：将通道设置为广播。

函数原型：

```
GOAL_STATUS_T goal_maChanUdpSetBroadcast(
    GOAL_MA_CHAN_UDP_T *pChanUdpHdl,        /**< pointer to store CHAN_UDP handler */
    GOAL_NET_CHAN_T *pChanHandle,           /**< channel handle */
    GOAL_BOOL_T flgOption                   /**< option value */
);
```

示例：

```
/* enable broadcast reception */
res = goal_maChanUdpSetBroadcast(pMaChanUdp, pChan1, GOAL_TRUE);
if (GOAL_RES_ERR(res)) {
    goal_logErr("error while setting UDP channel to receive broadcasts");
    return res;
}
```

6.8.7 goal_maChanUdpGetRemoteAddr - 获取 UDP 通道的远程地址

目的：获取 UDP 通道的远程地址以及从中接收数据的地址。

函数原型：

```
GOAL_STATUS_T goal_maChanUdpGetRemoteAddr(
    GOAL_MA_CHAN_UDP_T *pChanUdpHdl,        /**< pointer to store CHAN_UDP handler */
    GOAL_NET_CHAN_T *pChanHandle,           /**< channel handle */
    GOAL_NET_ADDR_T *pAddr                  /**< remote address */
);
```

6.8.8 goal_maChanUdpActivate - 激活 UDP 通道

目的：激活通道

函数原型：

```
GOAL_STATUS_T goal_maChanUdpActivate(
    GOAL_MA_CHAN_UDP_T *pChanUdpHdl,        /**< pointer to store CHAN_UDP handler */
    GOAL_NET_CHAN_T *pChanHandle            /**< channel handle */
);
```

示例：

```
res = goal_maChanUdpActivate(pMaChanUdp, pChan2);
if (GOAL_RES_ERR(res)) {
    goal_logErr("error while enabling UDP channel");
    return res;
}
```

6.8.9 goal_maChanUdpSend - 将数据发送到 UDP 通道

目的：将数据发送到打开的 UDP 通道。

函数原型：

```
GOAL_STATUS_T goal_maChanUdpSend(
    GOAL_MA_CHAN_UDP_T *pChanUdpHdl,        /**< pointer to store CHAN_UDP handler */
    GOAL_NET_CHAN_T *pChanHandle,           /**< channel handle */
    GOAL_BUFFER_T *pBuf                     /**< buffer with data to send */
);
```

示例：

```
/* echo message */
goal_maChanUdpSend(pMaChanUdp, pChan, pBuf);
```

7.Web服务器

7.1 概述

GOAL 为嵌入式系统提供一个智能 Web 服务器。Web 服务器的设计用于：

- 文件下载
- 获取有关当前设备状态和属性的信息。

Web 服务器支持下列属性：

传输协议：HTTP / HTTPS

请求方法：GET / POST

可以将一个或多个网页分配给 Web 服务器的一个实例。网页是应用程序的一部分，必须由应用程序提供。为此，Web 服务器提供一个回调函数。请参见第 8.5 节中的 cbHttpReqFunc()。

当前设备状态和属性可以从 CM 变量和应用程序特定的变量读取。应用程序特定的变量可以组织为简单变量或一维列表。

可以存储网页的模板，包含应用程序特定变量当前值的占位符。第 8.3 节中介绍了文本替换。Web 模板使 Web 页面的动态管理成为可能。

对 Web 服务器的访问可能受到用户级别的限制。应用程序可以指定设备应支持哪些用户级别，用户级别应具有哪些权限。由每个用户级别的用户名和密码组成的身份验证数据可以通过 CM 变量配置。GOAL Web 服务器最多提供 4 个用户级别。

用户级别可以通过 Web 服务器的所有实例应用。对于 Web 服务器的每个实例，可以在注册期间指定有效的用户级别。Web 请求仅在进行成功的身份验证后才能传输到应用程序，即只有在成功登录后才调用第 8.5 节中的回调函数 cbHttpReqFunc()。使用 HTTP 传输协议并通过 Web 服务器实例传输用户名和密码不安全。我们推荐使用运行 HTTPS 传输协议的端口。

HTTPS 由编译程序定义 GOAL_CONFIG_HTTPS 激活。HTTPS 使用外部软件构件 mbedTLS 进行编码和身份验证。通过 TLS 的媒介适配器实现对 mbedTLS 的访问。HTTPS 的 TLS 由函数 goal_httpsNew()初始化和打开。

关于 HTTPS 的 CM 变量，可以安装私钥和自己的 X509 证书。如果未存储自己的证书，则 Web 服务器将采用端口提供的默认证书。

示例：

- …\appl\goal_http\01_get*:

上传网页

- …\appl\goal_http\05_template_cm*:

上传包含 CM 变量和应用程序特定变量的 Web 模板

- …\appl\goal_http\06_template_list*:

上传包含列表的 Web 模板

- …\appl\goal_http\04_auth*:

有关用户级别的身份验证

- …\appl\goal_http\02_post*:

文件下载

7.2 配置

7.2.1 编译程序定义

以下编译程序定义可用于配置 Web 服务器：

GOAL_CONFIG_HTTP:

0：未使用传输协议 HTTP（默认）

1：使用传输协议 HTTP

GOAL_CONFIG_HTTPS:

0：未使用传输协议 HTTPS（默认）

1：使用传输协议 HTTPS

7.2.2 CM变量

以下 CM 变量可用于配置 Web 服务器：

CM-模块-ID	GOAL_ID_HTTP
CM-变量-ID	0
CM-变量名称	HTTP_CHANNELS_MAX
说明	可用于 HTTP 传输协议的最大连接数
CM 数据类型	UINT16
大小	2 字节
默认值	来自 NVS 或 0

CM-模块-ID	GOAL_ID_HTTP
CM-变量-ID	1
CM-变量名称	HTTPS_CHANNELS_MAX
说明	可用于 HTTPS 传输协议的最大连接数
CM 数据类型	UINT16
大小	2 字节
默认值	来自 NVS 或 0

CM-模块-ID	GOAL_ID_HTTP
CM-变量-ID	2
CM-变量名称	USERLEVEL0
说明	0 级身份验证数据
CM 数据类型	STRING
大小	32 字节
默认值	来自 NVS 或空字符串

CM-模块-ID	GOAL_ID_HTTP
CM-变量-ID	3
CM-变量名称	USERLEVEL1
说明	1 级身份验证数据
CM 数据类型	STRING
大小	32 字节
默认值	来自 NVS 或空字符串

CM-模块-ID	GOAL_ID_HTTP
CM-变量-ID	4
CM-变量名称	USERLEVEL2
说明	2 级身份验证数据
CM 数据类型	STRING
大小	32 字节
默认值	来自 NVS 或空字符串

CM-模块-ID	GOAL_ID_HTTP
CM-变量-ID	5
CM-变量名称	USERLEVEL3
说明	3 级身份验证数据
CM 数据类型	STRING
大小	32 字节
默认值	来自 NVS 或空字符串

下列 CM 变量允许配置 HTTPS 使用的 TLS 层：

CM-模块-ID	GOAL_ID_HTTPS
CM-变量-ID	0
CM-变量名称	TLS_SERVER_CERTIFICATE
说明	Web 服务器证书
CM 数据类型	GENERIC
大小	1024 字节
默认值	来自 NVS 或来自端口的证书

CM-模块-ID	GOAL_ID_HTTPS
CM-变量-ID	1
CM-变量名称	TLS_PRIVATE_KEY
说明	Web 服务器私钥
CM 数据类型	GENERIC
大小	1024 字节
默认值	来自 NVS 或空登记项

CM-模块-ID	GOAL_ID_HTTPS
CM-变量-ID	2
CM-变量名称	TLS_SRV_CERT_CA_CN
说明	认证机构服务器的公用名称
CM 数据类型	STRING
大小	128 字节
默认值	来自 NVS 或空字符串

CM-模块-ID	GOAL_ID_HTTPS
CM-变量-ID	3
CM-变量名称	TLS_SRV_CERT_CA_O
说明	认证机构组织的名称，例如公司名称
CM 数据类型	STRING
大小	128 字节
默认值	来自 NVS 或空字符串

CM-模块-ID	GOAL_ID_HTTPS
CM-变量-ID	4
CM-变量名称	TLS_SRV_CERT_CA_C
说明	认证机构组织所在的国家
CM 数据类型	STRING
大小	8 字节
默认值	来自 NVS 或空字符串

CM-模块-ID	GOAL_ID_HTTPS
CM-变量-ID	5
CM-变量名称	TLS_SRV_CERT_CN
说明	Web 服务器的公用名称
CM 数据类型	STRING
大小	128 字节
默认值	来自 NVS 或空字符串

CM-模块-ID	GOAL_ID_HTTPS
CM-变量-ID	6
CM-变量名称	TLS_SRV_CERT_O
说明	提供 Web 服务器的组织的名称
CM 数据类型	STRING
大小	128 字节
默认值	来自 NVS 或空字符串

CM-模块-ID	GOAL_ID_HTTPS
CM-变量-ID	7
CM-变量名称	TLS_SRV_CERT_C
说明	提供 Web 服务器的组织所在的国家
CM 数据类型	STRING
大小	8 字节
默认值	来自 NVS 或空字符串

CM-模块-ID	GOAL_ID_HTTPS
CM-变量-ID	8
CM-变量名称	TLS_SRV_CERT_NOT_BEFORE
说明	从证书开始生效的日期和时间起
CM 数据类型	STRING
大小	20 字节
默认值	来自 NVS 或空字符串

CM-模块-ID	GOAL_ID_HTTPS
CM-变量-ID	9
CM-变量名称	TLS_SRV_CERT_NOT_AFTER
说明	从证书失效的日期和时间起
CM 数据类型	STRING
大小	20 字节
默认值	来自 NVS 或空字符串

7.3 Web模板

GOAL 服务器允许实现包含当前信息占位符的网页模板。在上传过程中，占位符替换为 Web 服务器的当前值。Web 服务器提供下列占位符：

- CM 变量、
- 应用程序特定变量
- 列表。

7.3.1 CM变量

CM 变量的占位符包含 CM 模块 ID 和 CM 变量 ID。Web 服务器通过 CM 变量自动执行占位符的替换。

句法：

[CM:<modNum>, <cmVarNum>]

示例：

[CM:0, 2]

7.3.2 应用程序特定变量

应用程序特定变量的占位符包含应用程序中变量的名称。Web 服务器通过调用回调函数（参见第 8.5 节 cpHttpTemplateFunc()），要求从应用程序中获取变量的当前值，并将其替换为网页上的占位符。

句法：

[VAR:<applVarName>]

示例：

[VAR:applVar]

7.3.3 列表

Web 服务器提供一种在 HTML 文本中生成列表的有效方法。单个列表项的 HTML 文本可以包含在 Web 模板中的 FOREACH 和/FOREACH 占位符中。FOREACH 标记一个一维列表，对每个列表元素执行 FOREACH 和/FOREACH 占位符之间的 HTML 文本。列表项的位置由具有所需变量名的占位符 VAR 标记在 HTML 文本中。替换占位符 VAR 后，Web 服务器将自动更改为下一个列表项，即不可能两次替换同一个列表项。因此，只需描述 Web 模板中的第一个列表项。

在注册过程中，Web 服务器仅获取列表的 ID、名称和列表元素的数目。列表元素的内容由应用程序进行管理。Web 服务器调用回调函数，获取下一个列表元素的内容，参见第 8.5 节的 cpHttpTemplateFunc()。

允许嵌套的列表。最大支持的嵌套深度为 4。

句法：

[FOREACH:<listName>] … [/FOREACH]

HTML 列表示例：

```
<ul>
    [FOREACH:mainList]
<li> main: [VAR:mainName] </li>
<li> sub-lists:
<ul>
                    [FOREACH]
<li> sub: [VAR:subName] </li>
[/FOREACH]
</ul>
</li>
    [/FOREACH]
</ul>
```

示例…\appl\goal_http\06_template_list*生成 HTML 列表。在 Web 浏览器中的指示如图 8.1 所示：

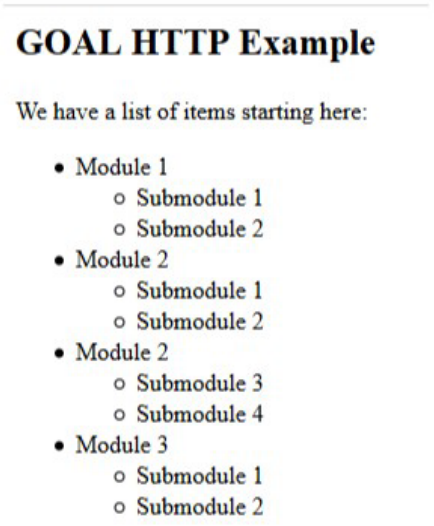

图 8.1 示例 06_template_list 的网页

占位符[FOREACH] … [\FOREACH]也可以集成到表格等其他 HTML 格式中。

7.4 字符

方括号：如果 HTML 文本应显示方括号，则方括号必须重复书写，因为 Web 模板中的占位符由方括号括起。

示例：阵列指令应显示在网页上。

HTML 文本：applArray[[5]]

Web 浏览器视图：applArray[5]

双引号：HTML 文本中的双引号必须通过反斜杠保护，因为 C 代码中的字符串由双引号括起。

示例：uint8_t webPage[] = “<html><meta charset = \” utf-8\” >…</html>”;

HTML 文本的规则适用于**所有其他字符**。

7.5 回调函数

原型设计	GOAL_STATUS_T cbHttpReqFunc(GOAL_HTTP_APPLCB_DATA_T *applData)	
说明	接收到的有效 Web 请求传递到应用程序。应用程序必须处理 Web 请求并生成 Web 响应。	
参数	applData	包含应用程序的数据和应用程序返回的数据。
返回值	GOAL 返回状态	
类别	可选	
注册	在函数 goal_httpdResReg()的执行期间	

原型设计	GOAL_STATUS_T cbHttpTemplateFunc(GOAL_HTTP_APPLCB_TEMPL_T *pWebTemplate)	
说明	关于该回调函数，应用程序提供指定占位符的当前信息。对 Web 模板中的每个占位符调用该回调函数。如果同一信息存在多个占位符，则调用每个占位符的该回调函数。	
参数	pWebTemplate	包含指定变量或列表的信息以及指定变量的当前返回值。
返回值	GOAL 返回状态	
类别	可选	
注册	在函数执行期间	

7.6 实现指南

7.6.1 上传网页

网页“device.html”应从设备上传到 Web 浏览器。网页的内容作为字符串存储在变量 webPage[] 中。无需在网页上进行文本替换。

1. 在 GOAL_FSA_INIT_APPL 或 GOAL_FSA_INIT_GOAL 状态下将 Web 服务器初始化：

```
goal_httpInit();
```

2. 使用 TCP 端口8081，在 GOAL_FSA_INIT_SETUP 状态下打开 Web 服务器的1个实例：

```
GOAL_HTTP_T *pWebInstanceHdl;       /* handle for the web-server instance */
goal_httpNew(&pWebInstanceHdl, 8081, 1);
```

3. 在 GOAL_FSA_INIT_SETUP 状态下，对打开的 Web 服务器注册回调函数和允许方法：未注册文本替换的回调函数。

```
GOAL_HTTP_HDL_T webResourceHdl;
goal_httpResReg(pWebInstanceHdl, (uint8_t*) "/device.html",
GOAL_HTTP_METHOD_ALLW_GET, applWebReqCb, NULL, &webResourceHdl);
```

4. 提供网页作为字符串变量：

```
constuint8_twebPage[] = ”<html><head><meta charset = \”utf-8\” >\
                    <title> Device </title></head> \r\n\
                    <body><h1>Device</h1> \r\n\
                    <p>The device implementation bases on the GOAL middleware.</p> \r\n\
                    </body></html>”
```

5. 实现回调函数以处理 Web 请求：

```
GOAL_STATUS_T applWebReqCb (GOAL_HTTP_APPLCB_DATA_T *pWebData) {
    GOAL_STATUS_T res;     /* GOAL return value */
    res = GOAL_ERROR;      /* no valid web-handle or request method not supported */

    if (webResourceHdl==pWebData->hdlRes) {
        if (GOAL_HTTP_FW_GET ==pWebData->reqType) {
            GOAL_HTTP_GET_RETURN_HTML(pWebData, webPage,
              GOAL_STRLEN((constchar*) webPage));
            res = GOAL_OK;
        }
    }
    return res;
}
```

6. Web 服务器在收到 Web 请求后调用回调函数 applWebReqCb()。

7.6.2 上传网页

网页“device.html”应从设备上传到 Web 浏览器。网页基于 webPage[]模板。该模板包括 CM 变量 DD_CM_VAR_MODULENAME 的占位符，CM 模块 ID 为 34，CM 变量 ID 为 0。Web 服务器自动将占位符替换为 CM 变量的当前值。无需在网页上进行文本替换。

1. 在 GOAL_FSA_INIT_APPL 或 GOAL_FSA_INIT_GOAL 状态下将 Web 服务器初始化：

```
goal_httpInit();
```

2. 使用 TCP 端口8081，在 GOAL_FSA_INIT_SETUP 状态下打开 Web 服务器的1个实例：

```
GOAL_HTTP_T *pWebInstanceHdl;       /* handle for the web-server instance */
goal_httpNew(&pWebInstanceHdl, 8081, 1);
```

3. 在 GOAL_FSA_INIT_SETUP 状态下，对打开的 Web 服务器注册回调函数和允许方法：未注册文本替换的回调函数。

```
GOAL_HTTP_HDL_T webResourceHdl;
goal_httpResReg(pWebInstanceHdl, (uint8_t*) “/device.html”,
GOAL_HTTP_METHOD_ALLW_GET, applWebReqCb, NULL, &webResourceHdl);
```

4. 提供网页模板作为字符串变量，包含 CM 变量的占位符：

```
constuint8_twebPage[] = ”<html><head><meta charset = \”utf-8\”>\
      <title> Device </title></head> \r\n\
      <body><h1>Device</h1> \r\n\
      <p>The device with the name [CM:34, 0] bases on the GOAL middleware.</p> \r\n\
      </body></html>”
```

5. 实现回调函数以处理 Web 请求：

```
GOAL_STATUS_T applWebReqCb (GOAL_HTTP_APPLCB_DATA_T *pWebData) {
    GOAL_STATUS_T res;     /* GOAL return value */
    res = GOAL_ERROR;      /* no valid web-handle or request method not supported */

    if (webResourceHdl==pWebData->hdlRes) {
        if (GOAL_HTTP_FW_GET ==pWebData->reqType) {
            GOAL_HTTP_GET_RETURN_HTML(pWebData, webPage,
              GOAL_STRLEN((constchar*) webPage));
            res = GOAL_OK;
        }
    }
    return res;
}
```

6. Web 服务器在收到 Web 请求后调用回调函数 applWebReqCb()。

7.6.3 读取应用程序特定的变量

网页“device.html”应从设备上传到 Web 浏览器。网页基于 webPage[]模板。该模板包括应用程序特定变量 applVar 的占位符。Web 服务器调用回调函数 applWebGetValCb()，为 Web 服务器提供应用程序特定变量的当前值。Web 服务器将占位符替换为应用程序特定变量的当前值。

1. 在 GOAL_FSA_INIT_APPL 或 GOAL_FSA_INIT_GOAL 状态下将 Web 服务器初始化：

```
goal_httpInit();
```

2. 使用 TCP 端口8081，在 GOAL_FSA_INIT_SETUP 状态下打开 Web 服务器的1个实例：

```
GOAL_HTTP_T *pWebInstanceHdl;       /* handle for the web-server instance */
goal_httpNew(&pWebInstanceHdl, 8081, 1);
```

3. 在 GOAL_FSA_INIT_SETUP 状态下，对打开的 Web 服务器注册回调函数和允许方法：

```
GOAL_HTTP_HDL_T webResourceHdl;
goal_httpResReg(pWebInstanceHdl, (uint8_t*) “/device.html”,
GOAL_HTTP_METHOD_ALLW_GET, applWebReqCb, applWebGetValCb, &webResourceHdl);
```

4. 提供网页模板作为字符串变量，包含应用程序特定变量的占位符：

```
constuint8_twebPage[] = “<html><head><meta charset = \”utf-8\”>\
      <title> Device </title></head> \r\n\
      <body><h1>Device</h1> \r\n\
      <p>The device with the name [VAR:applVar] bases on the GOAL middleware.</p> \r\n\
      </body></html>”
```

5. 实现回调函数以处理 Web 请求：

```
GOAL_STATUS_T applWebReqCb (GOAL_HTTP_APPLCB_DATA_T *pWebData) {
    GOAL_STATUS_T res;      /* GOAL return value */
    res = GOAL_ERROR;       /* no valid web-handle or request method not supported */

    if (webResourceHdl==pWebData->hdlRes) {
        if (GOAL_HTTP_FW_GET ==pWebData->reqType) {
            GOAL_HTTP_GET_RETURN_HTML(pWebData, webPage,
                GOAL_STRLEN((constchar*) webPage));
            res = GOAL_OK;
        }
    }
    return res;
}
```

6. 实现回调函数，以从应用程序中获取应用程序特定变量的当前值：

```
uint8_t deviceName[] = "Sample Gadget";
GOAL_STATUS_T applWebGetValCb (GOAL_HTTP_APPLCB_TEMPL_T *pWebData) {
    if (0==GOAL_MEMCMP(pWebData->in.name, “applVar”,
    GOAL_STRLEN(“applVar”))) {

      /* provide the complete device name */
      if (GOAL_STRLEN(deviceName) <=pWebData->in.retLenMax) {
            GOAL_MEMCPY(pWebData->out.strReturn, deviceName,
                  GOAL_STRLEN(deviceName));
      }
      /* the device name is too long, cut the device name down
                  * to the maximal allowed length */
      else {
          GOAL_MEMCPY(pWebData->out.strReturn, deviceName,
              pWebData->in.retLenMax);
        }
     }
     return GOAL_OK;
 }
```

7. Web 服务器在收到 Web 请求后调用回调函数 applWebReqCb()。Web 服务器可以使用所需的网页模板。

8. 调用回调函数 applWebGetValCb()进行替换。

7.6.4 读取列表

网页“device.html”应从设备上传到 Web 浏览器。网页基于 webPage[]模板。该模板包括设备组件列表的占位符。Web 服务器调用回调函数 applWebGetValCb()，提供列表条目的当前值。Web 服务器将占位符替换为应用程序特定变量的当前值。

1. 在 GOAL_FSA_INIT_APPL 或 GOAL_FSA_INIT_GOAL 状态下将 Web 服务器初始化：

```
goal_httpInit();
```

2. 使用 TCP 端口8081，在 GOAL_FSA_INIT_SETUP 状态下打开 Web 服务器的1个实例：

```
GOAL_HTTP_T *pWebInstanceHdl;       /* handle for the webserver instance */
goal_httpNew(&pWebInstanceHdl, 8081, 1);
```

3. 在 GOAL_FSA_INIT_SETUP 状态下，对打开的 Web 服务器注册回调函数和允许方法：

```
GOAL_HTTP_HDL_T webResourceHdl;
goal_httpResReg(pWebInstanceHdl, (uint8_t*) “/device.html”,
GOAL_HTTP_METHOD_ALLW_GET, applWebReqCb, applWebGetValCb, &webResourceHdl);
```

4. 创建列表信息并注册 Web 服务器的列表信息。列表信息包含列表 ID、列表名称和列表条目数。

```
GOAL_HTTP_TEMPLATE_LIST_INIT_T webList;
GOAL_MEMSET(&webList, 0, sizeof(webList));
webList.listId=1;
webList.cntMemb=4;
GOAL_MEMCPY(webList.listName, “deviceComponents”,
sizeof("devcieComponents“));
goal_httpTmpMgrNewList(pWebInstanceHdl, &webList);
```

5. 提供网页模板作为字符串变量，包含列表和列表条目的占位符：

```
const uint8_twebPage[] = “<html><head><meta charset = \”utf-8\”>\
                        <title> Device </title></head> \r\n\
                        <body><h1>Device</h1> \r\n\
                        <p> The device contains the following components: \r\n\
                        <ul> \r\n\
                              [FOREACH:deviceComponents] \r\n\
                              <li> [VAR:devComponent] </li> \r\n\
                              [/FOREACH]
                        </ul> \r\n\
                        </p> \r\n\
                        </body></html>”
```

6. 实现回调函数以处理 Web 请求：

```
GOAL_STATUS_T applWebReqCb (GOAL_HTTP_APPLCB_DATA_T *pWebData) {
    GOAL_STATUS_T res;     /* GOAL return value */
     res = GOAL_ERROR;     /* no valid web-handle or request method not supported */

     if (webResourceHdl==pWebData->hdlRes) {
          if (GOAL_HTTP_FW_GET ==pWebData->reqType) {
              GOAL_HTTP_GET_RETURN_HTML(pWebData, webPage,
                  GOAL_STRLEN((constchar*) webPage));
          res = GOAL_OK;
          }
     }
     return res;
 }
```

7. 实现回调函数，以从应用程序中获取列表条目的当前值：

```
GOAL_STATUS_T applWebGetValCb (GOAL_HTTP_APPLCB_TEMPL_T *pWebData) {
    GOAL_STATUS_T res;     /* GOAL return value */
    res = GOAL_ERR_NOT_FOUND;
    if (NULL!=pWebData->in.pPath) {
         if (0==GOAL_MEMCMP(pWebData->in.name, “devComponent“,
         GOAL_STRLEN(“devComponent“))) {

            switch ((pWebData->inPath)->path[0].index) {
              case 0:
                   GOAL_MEMCPY(pWebData->out.strReturn, “I/O module“,
                     GOAL_STRLEN(“I/O module“));
                   res = GOAL_OK;
                   break;
              case 1:
                   GOAL_MEMCPY(pWebData->out.strReturn, “drive “,
                     GOAL_STRLEN(“drive“));
                   res = GOAL_OK;
                   break;
              case 2:
                   GOAL_MEMCPY(pWebData->out.strReturn, “encoder “,
                     GOAL_STRLEN(“encoder “));
                   res = GOAL_OK;
                   break;
              case 3:
                   GOAL_MEMCPY(pWebData->out.strReturn, “power supply“,
                      GOAL_STRLEN(“power supply“));
                   res = GOAL_OK;
                   break;
              default:
                   break;
            }
        }
    }
    return res;
 }
```

8. Web 服务器在收到 Web 请求后调用回调函数 applWebReqCb()。Web 服务器可以使用所需的网页模板。

9. 对于每次替换，均调用回调函数 applWebGetValCb()。

7.6.5 设置用户级别

仅允许 USERLEVEL0 的用户上传网页“admin.html”。USERLEVEL0 的登录数据是：

- 用户名：admin
- 密码：a1b2c3:UL

使用 HTTPS 传输协议。

网页“admin.html”不包含任何占位符。无需在网页上进行文本替换。

1. 在 GOAL_FSA_INIT_APPL 或 GOAL_FSA_INIT_GOAL 状态下将 Web 服务器初始化：

```
goal_httpInit();
```

2. 使用 TCP 端口443，在 GOAL_FSA_INIT_SETUP 状态下打开 Web 服务器的1个实例：

```
GOAL_HTTP_T *pWebInstanceHdl;   /* handle for the web-server instance*/
goal_httpsNew(&pWebInstanceHdl, 443, 1);
```

3. 在 GOAL_FSA_INIT_SETUP 状态下，对打开的 Web 服务器注册回调函数和允许方法：

```
GOAL_HTTP_HDL_T webResourceHdl;
goal_httpResReg(pWebInstanceHdl, (uint8_t*) “/admin.html”,
GOAL_HTTP_METHOD_ALLW_GET | GOAL_HTTP_AUTH_USERLEVEL0, applWebReqCb,
NULL, &webResourceHdl);
```

4. 在状态 GOAL_FSA_INIT_SETUP 下安装 USERLEVEL0的身份验证：身份验证数据写入 CM 变量 USERLEVEL0中。

```
goal_httpAuthBasSetUserInfo(pWebInstanceHdl, GOAL_HTTP_AUTH_USERLEVEL0,
“admin”, “a1b2c3:UL”);
```

5. 提供网页模板作为字符串变量：

```
constuint8_twebPage[] = “<html><head><meta charset = \”utf-8\”>\
                         <title> Administration </title></head> \r\n\
                         <body><h1>Administration</h1> \r\n\
                         <p> Internal information:  ⋯</p> \r\n\
                         </body></html>”
```

6. 实现回调函数以处理 Web 请求：

```
GOAL_STATUS_T applWebReqCb (GOAL_HTTP_APPLCB_DATA_T *pWebData) {
    GOAL_STATUS_T res;      /* GOAL return value */
    res = GOAL_ERROR;       /* no valid web-handle or request method not supported */

    if (webResourceHdl==pWebData->hdlRes) {
        if (GOAL_HTTP_FW_GET ==pWebData->reqType) {
            GOAL_HTTP_GET_RETURN_HTML(pWebData, webPage,
                GOAL_STRLEN((constchar*) webPage));
            res = GOAL_OK;
        }
    }
    return res;
}
```

7. Web 服务器在收到 Web 请求后调用回调函数 applWebReqCb()。只有在成功登录到给定的 Web 服务器后，才能进行该操作。

7.6.6 下载文件

Web 服务器提供一个下载对话框。收到的文件将传输到应用程序。

1. 在 GOAL_FSA_INIT_APPL 或 GOAL_FSA_INIT_GOAL 状态下将 Web 服务器初始化：

```
goal_httpInit();
```

2. 使用 TCP 端口8081，在 GOAL_FSA_INIT_SETUP 状态下打开 Web 服务器的1个实例：

```
GOAL_HTTP_T *pWebInstanceHdl;      /* handle for the web-server instance */
goal_httpNew(&pWebInstanceHdl, 8081, 1);
```

3. 在 GOAL_FSA_INIT_SETUP 状态下，对打开的 Web 服务器注册回调函数和允许方法：

```
GOAL_HTTP_HDL_T webResourceHdl;
goal_httpResReg(pWebInstanceHdl, (uint8_t*) „/download.html “,
                    GOAL_HTTP_METHOD_ALLW_GET |
GOAL_HTTP_METHOD_ALLW_POST,
applWebReqCb, NULL, &webResourceHdl);
```

4. 提供网页作为字符串变量：

```
const uint8_t webPage[] = "<html><head><meta charset = \"utf-8\">\
                           <title> Download dialog </title></head> \r\n\
                           <body><h1>Download dialog</h1> \r\n\
                           <form method = \"post\"enctype= \"multipart/form-data\"> \r\n\
                           <input type = \"file\" name = \"file\"><br> \r\n\
                           <input type = \"submit\" value = \"POST\"> \r\n
                           </form> \r\n\
                           </body></html>"
```

5. 实现应用程序特定的函数 applDownload()以接收和安装新固件。

6. 实现应用程序特定的函数 applDownloadFinished()，将 Web 请求的结果报告至应用程序。

7. 实现回调函数以处理 Web 请求：

```
GOAL_STATUS_T applWebReqCb (GOAL_HTTP_APPLCB_DATA_T *pWebData) {
    GOAL_STATUS_T res;        /* GOAL return value */

    res = GOAL_ERROR;         /* no valid web-handle or request method not supported */

    if (webResourceHdl==pWebData->hdlRes) {
        switch (pWebData->reqType) {
            case GOAL_HTTP_FW_GET:
                GOAL_HTTP_GET_RETURN_HTML(pWebData, webPage,
                    GOAL_STRLEN((constchar*) webpage));
                break;
            case GOAL_HTTP_FW_POST_START:
                res =applDownload(pWebData);
                GOAL_HTTP_RETURN_OK_204(pWebData);
                break;
            case GOAL_HTTP_FW_POST_DATA:
                res =applDownload(pWebData);
                GOAL_HTTP_RETURN_OK_204(pWebData);
                break;
            case GOAL_HTTP_FW_POST_END:
                res =applDownload(pWebData);
                GOAL_HTTP_RETURN_OK_204(pWebData);
                break;
            case GOAL_HTTP_FW_REQ_DONE_OK:
            case GOAL_HTTP_FW_REQ_DONE_ERR:
                res =applDownloadFinished(pWebData);
                break;
            default:
                break;
        }
    }
    return res;
}
```

8. Web 服务器在收到 Web 请求后调用回调函数 applWebReqCb()。

8. 故障检修

本节列出使用 R-IN32M3 Module 示例时可能出现的常见缺陷。

8.1 启动问题

如果示例应用程序启动，但未显示预期的应用程序行为，请检查日志：

```
2019-01-18 09:44:45 GOAL_LOG_SEV_INFO 7 [I|goal_miDmPartRegInt:275] part added to 'Write to CC', pos: 1, len: 2
2019-01-18 09:44:45 GOAL_LOG_SEV_INFO 7 [I|appl_setup:798] TX: Slot 3 Subslot 1 DATA: 1
2019-01-18 09:44:45 GOAL_LOG_SEV_INFO 7 [I|appl_httpSetup:100] setup web server
2019-01-18 09:44:45 GOAL_LOG_SEV_INFO 7 [I|goal_httpNewAc:959] HTTP Application Core successfully started
2019-01-18 09:44:45 GOAL_LOG_SEV_INFO 7 [I|appl_httpSetup:178] web server setup done
2019-01-18 09:44:45 GOAL_LOG_SEV_INFO 7 [I|appl_setup:822] Device Version : 1.0.0.0
2019-01-18 09:44:45 GOAL_LOG_SEV_INFO 7 [I|appl_setup:823] Device Type : 1
2019-01-18 09:44:45 GOAL_LOG_SEV_INFO 7 [I|appl_setup:825] Serial Number : b4:e9:a3:00:75:39
2019-01-18 09:44:45 GOAL_LOG_SEV_INFO 7 [I|goal_miMctcRpcSyncLoop:1028] local setup done
2019-01-18 09:44:45 GOAL_LOG_SEV_INFO 7 fixed memory usage: 102096/262144 bytes
2019-01-18 09:44:45 GOAL_LOG_SEV_INFO 7 fixed memory usage: (39%)
2019-01-18 09:47:04 GOAL_LOG_SEV_ERROR 492 [CC_E|goal_miMctcMonitorRx:1250] data channel offline: MCTC SPI
2019-01-18 09:47:13 GOAL_LOG_SEV_INFO 501 [CC_I|goal_miMctcMonitorRx:1239] data channel online: MCTC SPI
2019-01-18 09:47:13 GOAL_LOG_SEV_ERROR 501 [CC_E|goal_miMctcRpcSyncLoop:956] sync needs local reset to proceed
```

如果显示最后一个有关“sync needs local reset to proceed”的日志条目，则通信控制器(CC)需要复位。这可以通过 R-IN32M3 Module 板上的”RST”按钮实现。

8.2 连接问题

如果 SPI 通信完全不工作，可以在下列最后一条日志消息中看到：

```
2019-01-18 09:47:04 GOAL_LOG_SEV_ERROR 492 [CC_E|goal_miMctcMonitorRx:1250] data channel offline: MCTC SPI
```

请检查应用程序控制器(AC)上的板连接和应用程序的正确执行。

8.3 IP配置

在 EtherNet/IP 项目中，如果重启后静态 IP 配置和 DHCP 之间的切换失败，请使用管理工具检查下列 CM 变量：

表 9.1IP 配置

模块 ID	变量 ID
GOAL_ID_NET	IP
GOAL_ID_NET	NETMASK
GOAL_ID_NET	GW
GOAL_ID_NET	VALID
GOAL_ID_NET	DHCP_ENABLED

要禁用 DHCP，将变量 DHCP_ENABLED 设置为 0。确保变量”VALID”设置为 1。将这些设置上传到 CC 并永久保存这些值。重启后，应禁用 DHCP。

8.4 降级到版本 1.0

如果 AC 应用程序使用 CC 模块的复位输入，则降级到版本 1.0 可能会失败。要成功降级到版本 1.0，禁用任何 AC 连接并在独立模式下更新模块。

版本 1.0 之后的固件减轻了固件更新期间外部复位的影响。

版本	日期	说明	
		页码	摘要
1.00	2020年8月3日	—	发布第一版
2.01	2021年1月15日	19, 24, 90-94, 146-161	添加EtherCAT函数
		38	添加第4.10.1.2章
		101-108	添加API函数
		145	添加EIP第6.4.13章, 第6.4.14章, 第6.4.15章
		51, 76, 126, 127	更改默认值
2.02	2021年6月14日	17	追加1.1项说明
		45	更改了SPI速度描述
2.03	2021年10月15日	16	添加了对uGOAL的描述
		17	FW2.1.0.0更新功能
		19, 20, 22	添加了uGOAL的措辞
		27	将第9.4章的内容整合到第4.9章中
		51	删除5.4.1章，上移后续章节
		60	删除5.4.46章，上移后续章节
		63	修复了第5.5.1章中函数名称的拼写错误
		70	修改第5.5.19章
		72	修改第5.5.23章
		81	在5.6.16章节中添加了注意事项
		84, 84	添加第5.6.23章, 5.6.24章
		89, 89	添加第5.7.10章, 5.7.11章
		90	删除了第5.8.1至5.8.6章以在第6.5章中进行描述，并向上移动了以下章节
		93	第5.9章的整体更改（更改描述，增加章节）
		104	添加第6.1.12章
		164	在第6.6.5章中添加了参数说明
		—	修正错字（从goal_appl.h 到goal_config.h）
		192	将 故障检修 移至第 8 章
2.04	2022年8月5日	—	小修正
2.05	2023年5月31日	—	删除了 Renesas Synergy 示例软件的描述
		187	7.6.4 誤記訂正: [VAR: deviceComponents] > [VAR:devComponent]
		187	7.6.4 訂正回调函数以处理
		131	6.3.30 添加 RPC 传输 API: pnioRpcSubmodPlug()
		140	6.4.5 添加 RPC 传输 API: goal_eipCreateAssemblyObjectRpc()
		157	6.5.16 添加 RPC 传输 API: goal_ecatdynOdSubIndexRpcAdd()

R-IN32M3 Module 用户手册：软件

发布日期：　　版本2.05　2023年5月31日

发布方：　　瑞萨电子公司

R-IN32M3 Module

Renesas Electronics Corporation

R17US0002CJ0205