User's Manual

R-IN32M3 Module (RY9012A0)

ユーザーズマニュアル　ソフトウェア編
API 説明

RENESAS MCU
R-IN32M3-EC

ルネサス エレクトロニクス
www.renesas.com

Rev.2.05　2023.05

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品、本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等
   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等
   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
10. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
12. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.4.0-1　2017.11)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。
5. クロックについて
   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

このマニュアルの使い方

1. 目的と対象者

このマニュアルは、R-IN32M3 Module のハードウェア機能と電気的特性をユーザに理解していただくためのマニュアルです。本マイコンを用いた応用システムを設計するユーザを対象にしています。このマニュアルを使用するには、電気回路、論理回路、マイクロコンピュータに関する基本的な知識が必要です。

本マイコンは、注意事項を十分確認の上、使用してください。注意事項は、各章の本文中、各章の最後、注意事項の章に記載しています

改訂記録は旧版の記載内容に対して訂正または追加した主な箇所をまとめたものです。改訂内容すべてを記録したものではありません。詳細は、このマニュアルの本文でご確認ください。

R-IN32M3 Module では次のドキュメントを用意しています。ドキュメントは最新版を使用してください。下記資料番号の末尾****部分は版数です。当社ホームページより最新版をダウンロードしてご参照ください。

ドキュメントの種類	記載内容	資料名	資料番号
データシート	ハードウェアの概要と電気的特性	R-IN32M3 Module データシート	R19DS0109JJ****
ユーザーズマニュアル ハードウェア編	ハードウェアの仕様（ピン配置、周辺機能の仕様、電気的特性、タイミング）と動作説明	R-IN32M3 Module ユーザーズマニュアル ハードウェア編	R19UH0122JJ****
ユーザーズマニュアル ソフトウェア編	API の説明	R-IN32M3 Module ユーザーズマニュアル　ソフトウェア編	本ユーザーズマニュアル
クイックスタートガイド	応用例情報 ホスト CPU 用サンプルプログラム	R-IN32M3 Module アプリケーションノート： クイックスタートガイド	R12QS0042JJ****
ルネサステクニカル アップデート	製品の仕様、ドキュメント等に関する速報	当社ホームページよりダウンロード	

2. 数や記号の表記

注：

本文中につけた注の説明

注意：

気をつけて読んでいただきたい内容

備考：

本文の補足説明

3. 略語および略称の説明

略語／略称	英語名
AC	Application Controller
ACK	Acknowledge
ADS	Automation Device Specification
API	Application Programming Interface
APMS	Acyclic Protocol Machine Sender
CC	Communication Controller
CIP	Common Industrial Protocol
CM	Configuration Manager
CPU	Central Processing Unit
DC	Distributed Clock
DD	Device Detection
DHCP	Dynamic Host Configuration Protocol
EDT	EtherCAT Design Tool
ESC	EtherCAT Slave controller
GOAL	Generic Open Abstraction Layer
HTTP	Hypertext Transfer Protocol
HTTPD	Hypertext Transfer Protocol Daemon
I/O	Input / Output
IP	Internet Protocol
IOCS	Input Output Object Consumer Status
IOPS	Input Output Object Provider Status
LLDP	Link Layer Discovery Protocol
MA	Media Adapter
MAC	Media Access Control
MCTC	Micro Core to Core
NVS	Non-Volatile Storage
OSAL	Operating System Abstraction Layer
PDI	Process Data Interface
PDO	Process Data Object
PLC	Programmable Logic Controller
RPC	Remote Procedure Call
SDO	Service Data Objects
SII	Slave Information Interface
SNMP	Simple Network Management Protocol
SPI	Serial Peripheral Interface
TCP	Transmission Control Protocol
TLV	Type Length Value
UART	Universal Asynchronous Receiver Transmitter
UDP	User Datagram Protocol
uGOAL	micro Generic Open Abstraction Layer

目　次

1. 概要

本書では、デュアルポートデバイスとして使用される R-IN32M3 Module のアプリケーションソフトウェアの設計を容易化するために開発された各種 API について説明します。

R-IN32M3 Module は、産業通信アプリケーションの早期開発のために設計されたモジュールで、通信コントローラ（CC）として動作するマイコンを搭載しています。ファームウェア（Binary Software）を実装する CC は、PROFINET, EtherNet/IP™ や EtherCAT®等の産業通信規格でのネットワーク通信を行います。アプリケーションコントローラ（AC）は、ユーザ定義のマイコンです。CC の設定を実装する AC は、対象となる産業ネットワークの通信プロトコル固有の要件を意識せずに、アプリケーション固有のソフトウェアを動作させることができます。AC は、ユーザのアプリケーションソフトウェアとの統合を容易にするための、ソースコードでルネサスが提供する API のセットを使用して CC と通信します。CC と AC 間の通信では、「コア to コア（C2C）」プロトコルが使用されます。以下の図に、各コントローラとそのインタフェースを示します。

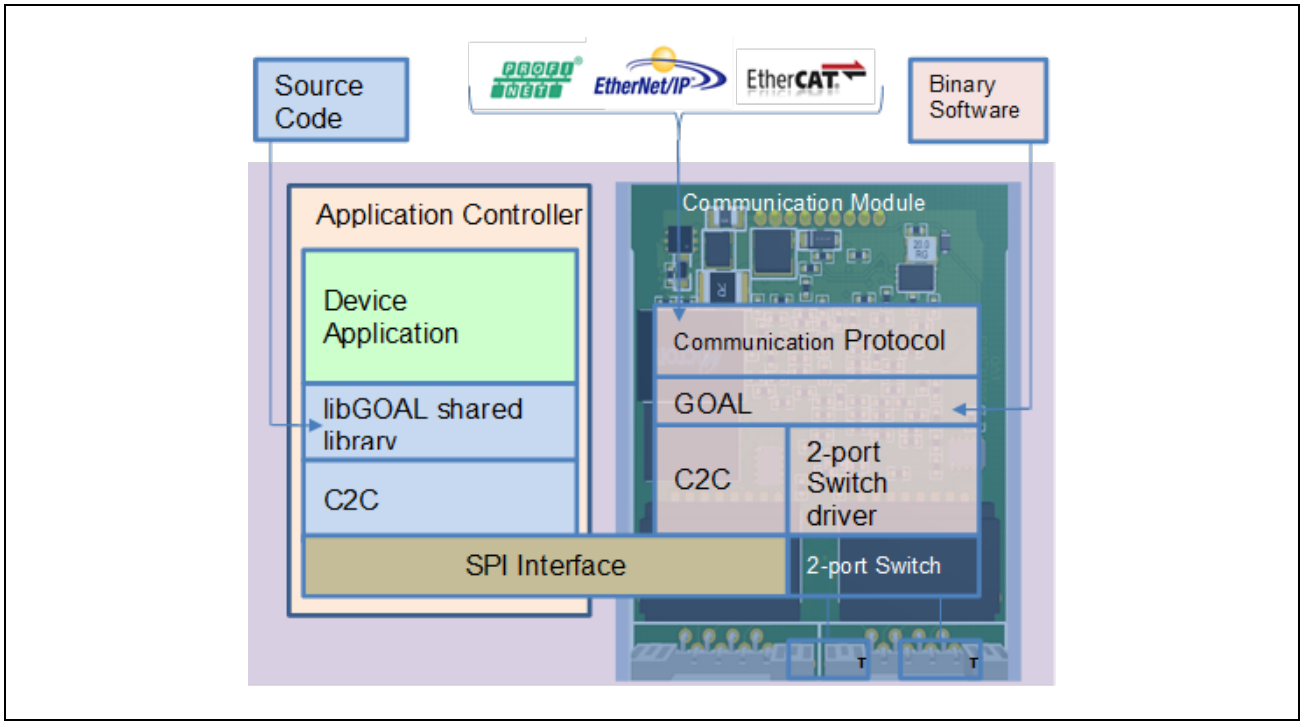

図 1.1 通信コントローラとアプリケーションコントローラ間のインタフェース

GOAL は、Generic Open Abstraction Layer（汎用オープン抽象化層）で、libGOAL は、アプリケーションソフトウェアから通信コントローラを制御するソースコード API です。GOAL と C2C の両コンポーネントは、OSAL として組み合わせて使用します。

uGOAL は、GOAL を継承しつつもその構成を簡素化し、使用するメモリサイズを大幅に削減したものになります。

1.1 産業用イーサネットプロトコル仕様

R-IN32M3 Module がサポートするプロトコルの主要機能を示します。
本機能は、ファームウェア Version 2.1.0.0 以降の条件となります。

■ **PROFINET**

・PROFINET IO Conformance Class B
・サイクリック通信周期 最小 1ms
・プロセスデータ 送受信 最大 1434Byte (*1)
・LLDP, SNMP プロトコルサポート
・MRP(Media Redundancy Protocol) サポート
・Alarm Queue サポート
・web ブラウザ http サポート

■ **EtherNet/IP**

・サイクリック通信周期 最小 1ms
・最大コネクションサイズ 495 Bytes (*1)
・DLR (Device Level Ring) サポート
・QoS サポート
・ACD サポート
・web ブラウザ http サポート

■ **EtherCAT**

・EtherCAT 通信プロファイル
CAN application protocol over EtherCAT (CoE)
Ethernet over EtherCAT (EoE)
File Access over EtherCAT (FoE)
・Explicit Device ID
・EtherCAT 同期機能 Free-Run, Sync Manager, ディストリビュートクロック (DC)
・PDO 最大 1408 Byte (*1)

(*1) サイクリック通信のデータサイズによって通信周期の制限があります。

2. ドキュメント構成

表 2.1 本書の内容

章	内容
概要	本書の概要
ドキュメント構成	本章
デバイスアーキテクチャ	R-IN32M3 Module のアーキテクチャの概要
アプリケーション	アプリケーションプログラミングのガイドライン
通信スタック	R-IN32M3 Module の通信インタフェースの説明
API	R-IN32M3 Module の使用可能な API の一覧と説明
サンプル	サンプルの説明
トラブルシューティング	一般的な使用上の問題の一覧
対象	対象の特定情報

3. デバイスアーキテクチャ

3.1 アーキテクチャ

R-IN32M3 Module のソフトウェアには、産業通信ドメインでのアプリケーション構築に利用可能なソフトウェアスタックで構成される、GOAL / uGOAL（ドメイン固有のミドルウェア）が含まれています。ソフトウェアスタックには以下があります。

- デバイス検出 : デバイス管理用簡易プロトコル
- PROFINET : PROFINET 通信用通信スタック
- EtherNet/IP : EtherNet/IP 通信用通信スタック
- EtherCAT : EtherCAT 通信用通信スタック
- Web サーバ : アプリケーション固有の管理と情報提供を行う簡易 Web サーバ
- TCP/IP スタック : UDP チャネルと TCP チャネルを提供する TCP/IP スタック

デバイスはアプリケーションコントローラ（AC）によって使用されます。2 つのコントローラ（アプリケーションコントローラと通信コントローラ）はサイクリックに通信を行います。通信はアプリケーションコントローラからの命令によって行われます。アプリケーションコントローラ（AC）には、ユーザーアプリケーションを構築するための通信コントローラ（CC）のサービスを実行するアプリケーションがあります。

3.2 インタフェース

3.2.1 ハードウェアインタフェース

R-IN32M3 Module の端子は、電源と SPI（スレーブインタフェース）に接続されます。

表 3.1 端子の説明

ピン	信号	I/O	機能
1	V_{CC}	—	3.3V±0.15V DC 電源
2	GND	—	グランド
3	/SS	I	スレーブ選択： アクティブ Low でスレーブデバイスを有効にする
4	/RESET	I	R-IN32M3 Module 全体のリセット： アクティブ Low
5	MISO	O	マスターイン、スレーブアウト： スレーブからマスターへのデータ
6	MOSI	I	マスターアウト、スレーブイン： マスターからスレーブへのデータ
7	SCLK	I	シリアルクロック： 本クロックに同期して、データが出力されます。
8	SYNC0	O	ディストリビュートクロック用の EtherCAT 同期信号
9	SYNC1	O	ディストリビュートクロック用の EtherCAT 同期信号

3.2.2 SPI ソフトウェアインタフェース

R-IN32M3 Module のソフトウェアインタフェースは、通信データの要件に応じて通信チャネルを提供する各種層で構成されています。SPI では、最大 128 バイトのデータのフレームがサイクリックに送信されます。通信は AC によって行われます。

SPI フレームは、RPC（Remote Procedure Call）によるデータ（通常、通信スタックからのリアルタイムデータおよび非リアルタイムデータ）の複数のセグメントで構成されています。通信スタックの詳細な説明は、以降の章に記載します。

3.2.3 通信層とミドルウェアの統合

要求された通信プロトコルのすべての層と AC は、GOAL / uGOAL ミドルウェアを使用して実装されます。AC は GOAL / uGOAL ミドルウェアの実装をベースにする必要があります。（フィールドバススタック PROFINET として）CC の特定の機能セットを利用するためには、RPC 機能を実装するラッパー機能が必要です。本ラッパーは GOAL / uGOAL を必要とするため、対象プラットフォームには GOAL / uGOAL ミドルウェアがサポートされている必要があります。

4. アプリケーション

4.1 概要

本章では、R-IN32M3 Module のアプリケーションの構築方法について説明します。

4.2 ハードウェアセットアップ

モジュールには管理インタフェースがあり、管理インタフェースで初期設定を行うことができます。プロパティはデバイス内で恒久的に保存されます。SPI インタフェースを設定するためのプロパティには以下があります。

表 4.1 SPI 設定

変数	内容	デフォルト値
SPI_TYPE	SPI マスター/スレーブ設定	1＝SLAVE
SPI_MODE	SPI タイミングモード	0＝MODE0
SPI_UNITWIDTH	SPI シングル転送モード	0＝8 BIT
SPI_BITORDER	SPI ビット転送方向	0＝MSB
SPI_TRANSFERSIZE	SPI パケット転送サイズ	128

本モジュールは、

SPI_TYPE スレーブ、
8 ビットの SPI_UNITWIDTH、
128 バイトの SPI_TRANSFERSIZE、
SPI ビット転送方向 MSB

のみに対応しています。

SPI モードは「**表 4.2 SPI モードの設定**」に従って設定することが可能です。

表 4.2 SPI モードの設定

値	SPI モード
0	モード 0
1	モード 1
2	モード 2
3	モード 3

4.3 基本アプリケーションのセットアップ

基本アプリケーションは、オプションの関数呼び出し appl_init、appl_setup、appl_loop で構成され、GOAL/uGOAL ミドルウェアの設計思想に基づいた構成となっています。

4.4 デフォルト機能

デフォルトでは、CC が Management Tool を使用して、ファームウェア更新のために（ポート 8080 上の）Web サーバを起動し、管理用にデバイス検出のインスタンスを起動します。

CC 上の Web サーバを AC によって無効にすることはできませんが、AC は Web サーバの別のインスタンスを起動することが可能です。

AC は、様々な方法でデバイス検出によるフル管理アクセスをデフォルトで制限することが可能です。該当章を参照してください。

4.5 機能

4.5.1 デバイス検出

4.5.1.1 初期状態

通信コントローラ（CC）では、「デバイス検出（DD）」機能がデフォルトで有効になっています。これにより、CC のすべての変数とログへのフルアクセスが可能になります。DD は、暗号化を行わない UDP ベースのプロトコルに基づいています。

このため、アプリケーションは可能な範囲までアクセス制限を行う必要があります。様々なレベルでの機能制限とアクセス制限が可能です。

4.5.1.2 アプリケーションによる機能の初期無効化

AC のセットアップが行われるときに、このプロパティがアプリケーション起動時に適用されます。

appl_setup()からの goal_ddNew()の呼び出しで、DD の単一機能を示すビットとともにビットマスクを渡すことが可能です。使用可能な値については、DD の API ドキュメントを参照してください。

全ビットが設定された場合、すべての機能（hello、get、set、get_list、win）が有効になります。

4.5.1.3 CM 変数による機能の初期無効化

「**4.5.1.2 アプリケーションによる機能の初期無効化**」に記載の仕組みは、対応するコンフィグレーションマネージャ（CM）変数 FEATURE_DISABLE（「**4.9.1 デバイス検出（DD**」を参照）を設定することで、アプリケーション起動前に設定することもできます。デバイス検出モジュールで処理される各要求は、要求の処理が可能であるかを判断するために変数の値を利用します。

異なる初期設定値が渡された場合には、goal_ddNew の呼び出しでこの変数の値が上書きされることに注意してください。

4.5.1.4 デバイス検出の機能の一時有効化

このプロパティは、AC によって提供される Web サイトを使用して設定する場合等、任意のタイミングで適用されます。

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

- パラメータ bitmask には、有効に設定されている機能を表すビットが含まれています。
- これらのビットは現在のセッションの CM から無効ビットを上書きします。

4.5.1.5 フィルタリングおよびアクセス権

このプロパティは AC 起動時に適用されます。

フィルタを使用して、グループ（モジュール ID）および変数（変数 ID）ごとに CM 変数へのアクセス制限を行うことが可能です。現在、一部のフィルタの使用が定義されています。これらのフィルタは、以下の関数を使用して有効にできます。

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

本関数は DD の作成されたインスタンス上で動作するため、有効な DD 処理の受け渡しが必要になります。

setId は、以下の値と合わせて使用することが可能です。

表 4.3 DD フィルタ ID

Set UD	内容
GOAL_DD_ACCESS_FILTER_SET_ALL	すべての変数へのフルアクセス
GOAL_DD_ACCESS_FILTER_SET_BASIC	Management Tool の最低限機能のフィルタ
GOAL_DD_ACCESS_FILTER_SET_HIDDEN	情報隠蔽フィルタ

備考 詳細については、DD の API ドキュメントを参照してください。

4.5.2 PNIO

通信コントローラ（CC）は、PROFINET スレーブスタックを実装しています。アプリケーションについては、5.4 通信スタック-PROFINET を参照してください。

4.5.3 EtherNet/IP

通信コントローラ（CC）は、EtherNet/IP スレーブスタックを実装しています。アプリケーションについては、5.6 通信スタック-EtherNet/IP を参照してください。

4.5.4 EtherCAT

通信コントローラ（CC）は、EtherCAT スレーブスタックを実装しています。アプリケーションについては、5.8 通信スタック-EtherCAT を参照してください。

4.5.5 汎用データプロバイダ

フィールドバス通信アプリケーションに対して、一般化情報および LED ステータスを含む汎用データプロバイダを使用できます。

表 4.4 MCTC ステータス情報

データプロバイダ情報	PROFINET	EtherNet/IP	EtherCAT
GOAL_MCTC_DP_STATUS_FLG_CONN	接続	接続	接続 (ESM-State == OP)
GOAL_MCTC_DP_STATUS_FLG_ERR	エラー	エラー	アプリケーション層エラー
GOAL_MCTC_DP_STATUS_FLG_VALID	—	プロセスデータ有効	—
GOAL_MCTC_DP_LED_WINK	DCP 点灯信号	—	—
GOAL_MCTC_DP_LED_RED1	エラー	モジュールステータス（赤）	EC ERR LED
GOAL_MCTC_DP_LED_RED2	維持	ネットワークステータス（赤）	—
GOAL_MCTC_DP_LED_GREEN1	接続	モジュールステータス（緑）	EC RUN LED
GOAL_MCTC_DP_LED_GREEN2	DCP 点灯信号	ネットワークステータス（緑）	—

4.5.6 イーサーネットスタートアップディレイ

通信コントローラ（CC）のバージョン 2.0.0.0 に EtherCAT を統合すると、アプリケーションコントローラ（AC）によるネットワークの専用スタートアップを可能にする追加機能が統合されます。選択した通信スタック（PROFINET や EtherCAT など）に応じて、ネットワークインターフェイスは必要なモードで起動されます。

ただし、アプリケーションコントローラ（AC）のない通信コントローラ（CC）は、このシナリオではネットワークを開始しません。この場合、TCP/UDP モードのフォールバックが実行されます。これはデフォルトで、通信コントローラ（CC）の起動後 5 秒後にネットワークを起動します。この動作は、ccm 変数（モジュール ID 72、変数 ID 13）を使用して構成できます。

次の表に、構成オプションを示します。

表 4.5 イーサーネット・タイムアウト・コンフィギュレーション

タイムアウト 値	機能
0	default timeout enabled (5 seconds)
1 ... 254	value determines timeout in seconds
255	timeout disabled

AC への電力供給と AC と CC 間の通信確立するまでに、設定されたタイムアウトよりも時間がかかる場合は、タイムアウトを増やす必要があります。

4.6 外部リセット

外部リセット入力を使用できます。AC による R-IN32M3 Module のリセットが可能です。

注意：CC のファームウェア更新中は、本リセットを行わないでください。ファームウェア更新中にリセットを行うと、更新機能が正しく動作しません。AC への最初の電源投入時（コールドスタート時）以外で本リセットを使用することは推奨しません。

4.7 RPC 同期リセット

4.7.1 通信コントローラ（CC）

デフォルトでは、CC は暗黙的にリセットを行いません。

CC は、必要に応じて電源の再投入を行うことを AC に要求します。これは、先に起動し、設定された AC が、新たに起動した CC に接続された場合に発生します。

4.7.2 アプリケーションコントローラ（AC）

AC は、CC の再起動時等に、CC の要求に応じて自身のアプリケーションのリセットを行います。ファームウェア更新中もリセットできます。

ハードウェア RESET 信号を使用して CC のリセットを行うことも可能です。

注意：CC のファームウェア更新中は、本リセットを行わないでください。ファームウェア更新中にリセットを行うと、更新機能が正しく動作しません。

4.8 IP 設定

IP 設定は、CM インタフェースで行うことができます。

表 4.6 CM での IP 設定

手順	操作	備考
1	CM 変数 GOAL_ID_NET：IP を設定	IP アドレスの設定
2	CM 変数 GOAL_ID_NET：NETMASK を設定	ネットマスクの設定
3	CM 変数 GOAL_ID_NET：GW を設定	ゲートウェイの設定
4	CM 変数 GOAL_ID_NET：Valid を 1 に設定	IP 設定を有効に設定
5	CM 変数 GOAL_ID_NET：DHCP_ENABLED を 0 に設定	DHCP の無効化
6	CM 変数 GOAL_ID_NET：COMMIT を 1 に設定	IP 設定の適用

注： DHCP を有効にする場合は、CM 変数 GOAL_ID_NET：DHCP_ENABLED を 1 に設定し、電源の再投入を行います。

4.9 R-IN32M3 Module 管理インタフェース

R-IN32M3 Module を構成する各変数は、用途ごとにモジュール ID でグループ分けされた管理インタフェースがあります。

管理インタフェース(モジュール ID)毎の表は以下の通りで、それぞれの表は R-IN32M3 module ファームウェア Version 2.1.0.0 時点の内容となっております。

なお、R-IN32M3 Module を構成する各変数は、モジュール ID と変数 ID を用いることでアプリケーションからアクセスできます。

- appl_ccmCfgVarGet()関数によって値を取得
- appl_ccmCfgVarSet()関数によって値を設定
- appl_ccmCfgSave()関数によって永続的に値を不揮発性メモリに保存

4.9.1 デバイス検出（DD）

モジュール ID＝GOAL_ID_DD (34)

表 4.7 DD 管理インタフェース

変数名	変数 ID	タイプ	最大サイズ	詳細説明
MODULENAME	0	STRING	20	ユーザ固有のモジュール名
CUSTOMERID	1	UINT32	4	ユーザ ID
RESERVED	2	UINT8	1	—
FEATURE_DISABLE	3	UINT32	4	各機能を無効にします。 ビット 0：“HELLO DETECTION”無効 ビット 1：WINK 無効 ビット 2：GETLIST 無効 ビット 3：GET VALUE 無効 ビット 4：SET VALUE 無効

4.9.2 PNIO

モジュール ID＝GOAL_ID_PNIO (27)

表 4.8 PNIO 管理インタフェース

変数名	変数 ID	タイプ	最大 サイズ	詳細説明
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 ログマネージャ（LM）

モジュール ID＝GOAL_ID_LM (35)

表 4.9 LM 管理インタフェース

変数名	変数 ID	タイプ	最大サイズ	詳細説明
VERSION	0	UINT8	1	LM インタフェースのバージョン情報
READBUFFER	1000	GENERIC	128	デバイスからオンラインログを読み出すバッファ
CNT	1001	UINT16	2	オンラインログアクセスのコントロールワード
EXLOG_READBUFFER	1002	GENERIC	128	デバイスから例外ログを読み出すバッファ
EXLOG_CNT	1003	UINT16	2	例外ログアクセスのコントロールワード
EXLOG_SIZE	1004	UINT32	4	例外ログサイズのインジケータ
EXLOG_USAGE	1005	UINT8	1	例外ログ使用のインジケータ
EXLOG_ERASE	1006	UINT8	1	コマンド *：例外ログ消去

4.9.4 NET

モジュール ID＝GOAL_ID_NET (12)

表 4.10 NET 管理インタフェース

変数名	変数 ID	タイプ	最大サイズ	詳細説明
IP	0	IPV4	4	第 1 インタフェースの IP アドレス
NETMASK	1	IPV4	4	第 1 インタフェースのネットマスク
GW	2	IPV4	4	第 1 インタフェースのゲートウェイ
VALID	3	UINT8	1	IP アドレスの有効性 0：格納された IP アドレスは無効、インタフェース設定はシステムのネットワークスタックから送信 1：格納された IP アドレスは有効、デバイス開始時にインタフェースに適用
DHCP_ENABLED	4	UINT8	1	DHCP 有効化 0：DHCP 無効 1：DHCP 有効
DHCP_STATE	5	UINT8	1	DHCP 状態 0：DHCP 初期化 1：DHCP サーバ選択中 2：DHCP 設定要求中 3：DHCP IP アドレス割り振り 4：DHCP 設定更新中 5：DHCP は IP アドレスをインタフェースに再割り振り中
DNS0	6	IPV4	4	第 1 インタフェースの第 1DNS サーバ
DNS1	7	IPV4	4	第 1 インタフェースの第 2DNS サーバ
HOSTNAME	8	STRING	20	第 1 インタフェースのホスト名
COMMIT	1000	UINT8	1	コマンド *：IP 設定割り当て

4.9.5 BOOT

モジュール ID＝GOAL_ID_BOOT (37)

表 4.11 BOOT 管理インタフェース

変数名	変数 ID	タイプ	最大サイズ	詳細説明
SIGNATURE	0	GENERIC	16	ブートイメージの署名
BLVERSION	1	STRING	16	ブートローダバージョン
FWVERSION	2	STRING	16	ファームウェアバージョン
RESET_CAUSE	1000	UINT8	1	リセット要因 0：不定 1：ファームウェア更新要求 2：ウォッチドッグ 3：ファームウェアコミット要求 4：予約
IMAGE_NUMBER	1001	UINT8	1	ブートイメージ番号
IMAGE_COUNTER	1002	UINT8	1	ブートイメージカウンタ

4.9.6 コンフィグマネージャ（CM）

モジュール ID＝GOAL_ID_CM (2)

表 4.12 CM 管理インタフェース

変数名	変数 ID	タイプ	最大サイズ	詳細説明
VERSION	0	UINT32	4	CM インタフェースのバージョン情報
SAVE	1000	UINT8	1	コマンド *：Flash に CM を保存

4.9.7 ETH

モジュール ID＝GOAL_ID_ETH (4)

表 4.13 ETH 管理インタフェース

変数名	変数 ID	タイプ	最大サイズ	詳細説明
MAC	0	GENERIC	6	—
LINK	1000	UINT32	4	インタフェースのリンクステータスマスク
SPEED	1001	UINT32	4	インタフェースのポート速度マスク
DUPLEX	1002	UINT32	4	インタフェースのポート二重化マスク
PORTCNT	1003	UINT32	4	インタフェース数

4.9.8 EIP

モジュール ID＝GOAL_ID_EIP (23)

表 4.14 EIP 管理インタフェース

変数名	変数 ID	タイプ	最大サイズ	詳細説明
INCARNATIONID	0	UINT32	4	電源再投入の間の固有の接続 ID の作成に使用
DOMAIN	1	GENERIC	EIP_CM_DOMAIN_LEN	TCP/IP インタフェースオブジェクトの一部の属性 5
HOST	2	GENERIC	EIP_CM_HOST_LEN	TCP/IP インタフェースオブジェクトの属性 6
ENCAPTIMEOUT	3	UINT16	2	TCP/IP インタフェースオブジェクトの属性 13
PORTCFG	4	GENERIC	sizeof(OPENER_PORT_CFG_T)	Ethernet リンクオブジェクトの属性 6、9（インタフェース制御・管理状態）
QOS_VLAN	5	UINT8	1	QoS オブジェクトの属性 1
QOS_URGENT	6	UINT8	1	QoS オブジェクトの属性 4
QOS_SCHEDULED	7	UINT8	1	QoS オブジェクトの属性 5
QOS_HIGH	8	UINT8	1	QoS オブジェクトの属性 6
QOS_LOW	9	UINT8	1	QoS オブジェクトの属性 7
QOS_EXPLICIT	10	UINT8	1	QoS オブジェクトの属性 8
TTL	11	UINT8	1	TCP/IP インタフェースオブジェクトの属性 8 マルチキャストメッセージの IP ヘッダの TTL 値
MC_CTRL	12	UINT8	1	TCP/IP インタフェースオブジェクトの属性 9 マルチキャストアドレスの割り当て方法を選択 0：デバイスの IP アドレスからアドレスを算出 1： マルチキャスト開始アドレス＋オフセットを使用
MC_NUM	13	UINT16	2	TCP/IP インタフェースオブジェクトの属性 9 割り当てられた EtherNet/IP 用マルチキャストアドレスの数
MC_START	14	UINT32	4	TCP/IP インタフェースオブジェクトの属性 9 最初に割り当てられたマルチキャストアドレス
ACD_ENABLED	15	UINT8	1	ACD 有効化 0：ACD 無効 1：ACD 有効
ACD_LAST_CONFLICT	16	GENERIC	8	直前のアドレス競合を起こしたデバイスの MAC

4.9.9 HTTP

モジュール ID＝GOAL_ID_HTTP (25)

表 4.15 HTTP 管理インタフェース

変数名	変数 ID	タイプ	最大サイズ	詳細説明
HTTP_CHANNELS_MAX	0	UINT16	2	HTTP サーバへの有効接続数の決定
HTTPS_CHANNELS_MAX	1	UINT16	2	HTTPS サーバへの有効接続数の決定
USERLEVEL0	2	STRING	32	レベル 0 の認証データ
USERLEVEL1	3	STRING	32	レベル 1 の認証データ
USERLEVEL2	4	STRING	32	レベル 2 の認証データ
USERLEVEL3	5	STRING	32	レベル 3 の認証データ

4.9.10 CCM

モジュール ID＝GOAL_ID_CCM (72)

表 4.16 R-IN32M3 Module 管理インタフェース

変数名	変数 ID	タイプ	最大サイズ	詳細説明
SPI_TYPE	0	UINT8	1	SPI タイプ（現在、スレーブのみ対応） 0：SPI マスター 1：SPI スレーブ
SPI_MODE	1	UINT8	1	SPI モード 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	マスタモード時の SPI 速度
SPI_UNITWIDTH	3	UINT8	1	1 データ転送のビットサイズ 0：8 ビット 1：16 ビット 2：32 ビット
SPI_BITORDER	4	UINT8	1	SPI 転送のビットオーダー 0：MSB ファースト 1：LSB ファースト
SPI_TRANSFERSIZE	5	UINT16	2	1 送信フレームの最小転送サイズ
COMM_FAULT_ERROR_STATE	6	UINT8	1	サイクリック接続中に AC への通信が切断された場合に行うフォルト処理 0：フォルト状態に移行（接続無効） 1：動作を維持（接続維持）
COMM_SYNC_RESET	7	UINT8	1	AC からの同期リセット要求受け付け時の動作 0：何もしない 1：CC のリセットを実行
FW_UPDATE_COMMIT_DISABLE	8	UINT8	1	ファームウェア更新時追加コミットステップの任意の無効化 0：ファームウェア更新にはコミットステップが必要 1：ファームウェア更新にはコミットステップが不要
FOE_FILENAME	9	STRING	32	EtherCAT FoE アップデート用ファイル名
FOE_PASSWORD	10	UINT32	4	EtherCAT FoE アップデート用パスワード
FOE_UPDATE_REQUIRES_BOOT	11	UINT8	1	EtherCAT FoE アップデート用要求する状態
FOE_FILENAME_MATCH_LEN	12	UINT8	1	EtherCAT FoE アップデート用ファイル名一致を判定する長さ
ETH_SWITCH_MODE_TIMEOUT	13	UINT8	1	イーサネットインターフェイスアクティベーションの一般的なタイムアウト
UPTIME	1000	UINT32	4	デバイス開始からの秒数

4.9.11 QUEUE

モジュール ID＝GOAL_ID_QUEUE (42)

表 4.17 QUEUE 管理インタフェース

変数名	変数 ID	タイプ	最大サイズ	詳細説明
SMALLBUFSIZE	0	UINT16	2	小メモリバッファのバッファサイズ
SMALLBUFNUM	1	UINT16	2	小メモリバッファの受信量
MEDBUFSIZE	2	UINT16	2	中メモリバッファのバッファサイズ
MEDBUFNUM	3	UINT16	2	中メモリバッファの受信量
BIGBUFSIZE	4	UINT16	2	大メモリバッファのバッファサイズ
BIGBUFNUM	5	UINT16	2	大メモリバッファの受信量

4.9.12 SNMP

モジュール ID＝GOAL_ID_SNMP (75)

表 4.18 SNMP 管理インタフェース

変数名	変数 ID	タイプ	最大サイズ	詳細説明
MIB2_SYS_LOCATION	0	STRING	255	—
MIB2_SYS_CONTACT	1	STRING	255	—
MIB2_SYS_NAME	2	STRING	255	—

4.9.13 MCTC

モジュール ID＝GOAL_ID_MCTC (94)

表 4.19 MCTC 管理インタフェース

変数名	変数 ID	タイプ	最大サイズ	詳細説明
PRC_DISABLE	0	UINT8	4	RPC モード 0：有効 1：無効
ID	1000	UINT32	4	インスタンス ID
STAT_RESET	1001	UINT32	4	MCTC 通信リセット数
STAT_RPC_TIMEOUTS	1002	UINT32	4	MCTC RPC タイムアウト数
STAT_RPC_DELAY_MIN	1003	UINT32	4	RPC 要求の最小時間
STAT_RPC_DELAY_MAX	1004	UINT32	4	RPC 要求の最大時間
STAT_RPC_DELAY_MEAN	1005	UINT32	4	RPC 要求の平均時間
STAT_RPC_COUNT	1006	UINT32	4	RPC 要求数

4.10 ファームウェア更新

4.10.1 通信コントローラ（CC）の更新

通信コントローラ（CC）のファームウェア更新が可能です。ファームウェア更新は Management Tool を使用して行われます。Management Tool は、HTTP プロトコルを使用して、ファームウェアパッケージをデバイスに転送します。

4.10.1.1 ファームウェアパッケージ

パッケージ内にはファームウェアイメージが同梱されています。このパッケージにはデバイス作成者による署名付きファームウェアが含まれています。これによりファームウェアイメージの真正性をデバイスが確認することができます。

4.10.1.2 制御インタフェース

ファームウェア更新は、デフォルトで常時可能となっています。通信コントローラ（CC）には、ファームウェアの更新処理を有効/無効にするインタフェースがあります。アプリケーションコントローラ（AC）は、ファームウェア更新を拒否しなければならない場合に、本インタフェースを使用する必要があります。PLC へのサイクリック接続中は、ファームウェア更新を無効にする必要があります。

本インタフェースの使用については、「**6.1 デバイス固有関数**」を参照してください。

オプションで、アプリケーションコントローラ（AC）はファームウェアアップデートに関するイベントに登録できます。 次に、転送の開始と終了がアプリケーションコントローラ（AC）に通知されます。 そのタスクは、ブートローダの再起動と実際のファームウェアアップデートのパフォーマンスをトリガすることも意味します。 失敗または成功したファームウェアアップデートのほかに、アプリケーションコントローラ（AC）に報告されます。

コールバック関数は、appl_ccmFwUpdateCbReg（）を使用して登録できます。

```
/*****************************************************************************/
/** 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 ファームウェア更新手順

ファームウェア更新は、2 ステップで行われます。はじめに、ファームウェアを CC の HTTP サーバにアップロードします。この間、以下の検証が行われます。

1. アップロードでは、認証レベル 0 の認証が行われます。この認証で、HTTPD モジュールに対して資格情報が設定されているかどうかの確認が行われます。資格情報の変数は、HTTPD サービスの RPC インタフェースを介して設定可能です（資格情報はデフォルトで空白になっています）。これにより、認証が受け付けられます。

2. ファームウェア更新は AC によって許可される必要があります。デフォルトではそのように設定されていますが、AC は RPC インタフェースを使用してこの機能を無効にすることもできます。

転送終了時に有効なファームウェアが検出されると、CC が再起動し、ブートローダに移行します。この CC は、ファームウェアの署名を確認し、デバイスのメモリに正しく署名されたファームウェアを書き込みます。フォールバックとして、以前実行されたファームウェアが保持されます。

更新されたファームウェアを恒久的に有効にするためには、CC 再起動後、Management Tool への通信が正常に行われる必要があります。成功すると、CC が再度再起動し、ブートローダが新しいファームウェアを現在のファームウェアとして示します。失敗すると、次回電源投入時に元のファームウェアに戻します。

4.10.1.4 DD 無効状態での更新機能維持

アプリケーションインテグレータは、デバイスの内部情報を表示しないようにするために、DD 機能を無効にしたい場合があります。本機能を無効にすることはできませんが、Management Tool を使用してファームウェア更新を許可するためには、以下の手順を行う必要があります。

1. デフォルトで DD を無効にする

API 関数 goal_ddNew を呼び出すと、有効な DD 機能を決定するビットマスクが渡されます。所定値 GOAL_DD_FEAAT_NO を使用すると、DD は完全に無効になります。

この値は内部で CM 変数 FEATURE_DISABLE に格納され、その状態を Flash に恒久的に保存することもできます。

2. DD 機能を一時的に有効にするための切り替え

ファームウェア更新処理を行う場合、以下の最低限の DD 機能を一時的に設定する必要があります。

- HELLO REQUEST
- SET CONFIG
- GET CONFIG
- SET IP

この設定は、以下の引数で API 関数 goal_ddSessionFeatureActivate()を使用して行うことが可能です。

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

管理インタフェースを有効にするための一時的な切り替えは、Web サーバを使用して行うことができます。

これにより、デバイスが Management Tool 側から見えなくなります。ファームウェア更新に必要なインタフェースが一時的に有効化され、通信コントローラ（CC）の更新が可能になります。

4.10.2 アプリケーションコントローラ（AC）の機能更新

デフォルトでは、アプリケーションコントローラ（AC）に対するファームウェア更新機能はありませんが、R-IN32M3 Module では、この機能をユーザーアプリケーション内に実装するための仕組みを提供しています。

以下の項で、可能な 2 つの方法を示します。

4.10.2.1 HTTP 経由の AC ファームウェア更新

アプリケーションコントローラ（AC）は、HTTPD サービスの RPC インタフェースを利用して、Web サイトまたはツールベースのファームウェア更新機能を HTTP トランスポート経由で提供することが可能です。はじめに、AC に HTTPD サンプル goal_http/02_post を、CC にサンプルプログラム 01_pnio_io_mirror を使用できます。

後者については、例として以下のサンプルコードに示すように、HTTP POST リクエストを使用して伝送されたファームウェア更新用データが、コールバック関数 httpDataCbPost で処理されます。

```
/*************************************************************/
/** goal http data callback
 *
 */
static GOAL_STATUS_T httpDataCbPost(
  GOAL_HTTP_APPLCB_DATA_T *pCbInfo             /**< pointer to callback info struct */
)
{
  GOAL_STATUS_T res = GOAL_OK;              /* result */
  static uint32_t uploadDataLen = 0;     /* recent uploaded data length */

  if (hdlUplHtml == pCbInfo->hdlRes) {
      /* check request method */
      switch (pCbInfo->reqType)
      {
      case GOAL_HTTP_FW_POST_START:
        /* reset data length */
        uploadDataLen = 0;
        GOAL_HTTP_RETURN_OK_204(pCbInfo);
        break;

      case GOAL_HTTP_FW_POST_DATA:

        /* process data */
        GOAL_MEMCPY(pDst, pCbInfo->cs.pData, pCbInfo->cs.lenData);
        uploadDataLen += pCbInfo->cs.lenData;

        GOAL_HTTP_RETURN_OK_204(pCbInfo);
        break;

      case GOAL_HTTP_FW_POST_END:
        GOAL_HTTP_RETURN_OK_204(pCbInfo);
        break;

      case GOAL_HTTP_FW_REQ_DONE_OK:
      case GOAL_HTTP_FW_REQ_DONE_ERR:
        res = GOAL_OK;
        break;

      default:
        /* return error */
        GOAL_HTTP_RETURN_ERR_403(pCbInfo);
        break;
```

```
44.         }
45.       }
46.     else {
47.       /* return error */
48.       GOAL_HTTP_RETURN_ERR_404(pCbInfo);
49.     }
50.     return res;
51.   }
```

4.10.2.2 TCP 経由の AC ファームウェア更新

```
/*****************************************************************/
/** 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. 通信スタック

5.1 概要

本章では、通信スタックの設定に使用されるすべての関数について記載します。本関数は、goal_New()が呼び出される前に、appl_setup() before goal_New()内で呼び出す必要があります。そうしないと、本関数は無効となります。各設定にはデフォルト値があり、他の値が設定されない場合にデフォルト値が適用されます。

5.2 SPI データ交換

R-IN32M3 Module との通信には、シリアルペリフェラルインタフェース（SPI）を使用します。転送は、常にアプリケーションコア（AC）によってトリガされ、ハートビートタイムアウト中に少なくとも一度行われる必要があります。R-IN32M3 Module では、実行中の転送が中断されないように、各転送後に SPI データの更新を行います。

5.2.1 クロックドメインと通信サイクル

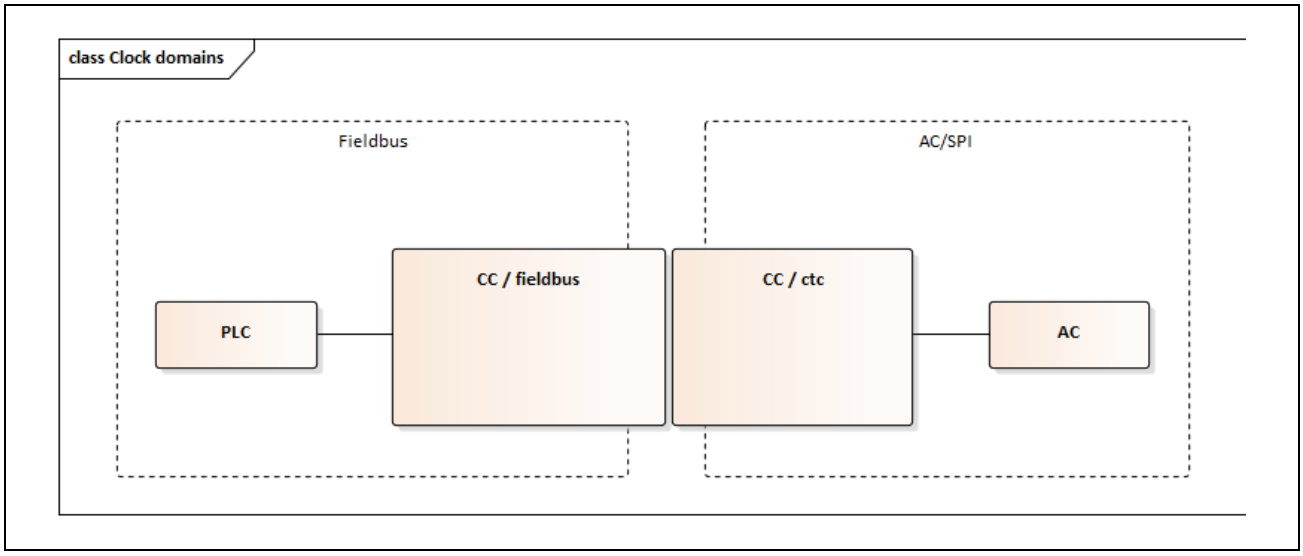

図 5.1 ctc クロックドメイン

デバイスは、それぞれ独立して動作する 2 つのクロックドメインで動作します。1 つ目のクロックドメインはフィールドバス側にあります。通常、PLC はこのクロックドメインでデバイスと交信し、出力データのタイミングを制御します。2 つ目のクロックドメインは SPI 転送開始時に AC によって駆動されます。AC は、SPI 転送で、デバイスから出力データを読み出し、次のフィールドバスサイクルで入力データを更新します。プロセスデータの特定のタイミングは、「**図 5.2 通信サイクル**」に示すように設定されます。

データ転送は以下のように行われます。

1. PLC から新規データ出力
2. CC で出力データを処理、SPI 転送バッファをプリロード
3. AC による SPI 転送の完了後、AC と CC 間でデータをやり取り
4. 次の SPI 転送の新規入力データをプリロード
5. 連続 SPI 転送の完了後、AC と CC 間でデータのやり取りが行われ、次のフィールドバス転送の新規入力データが提供されます。

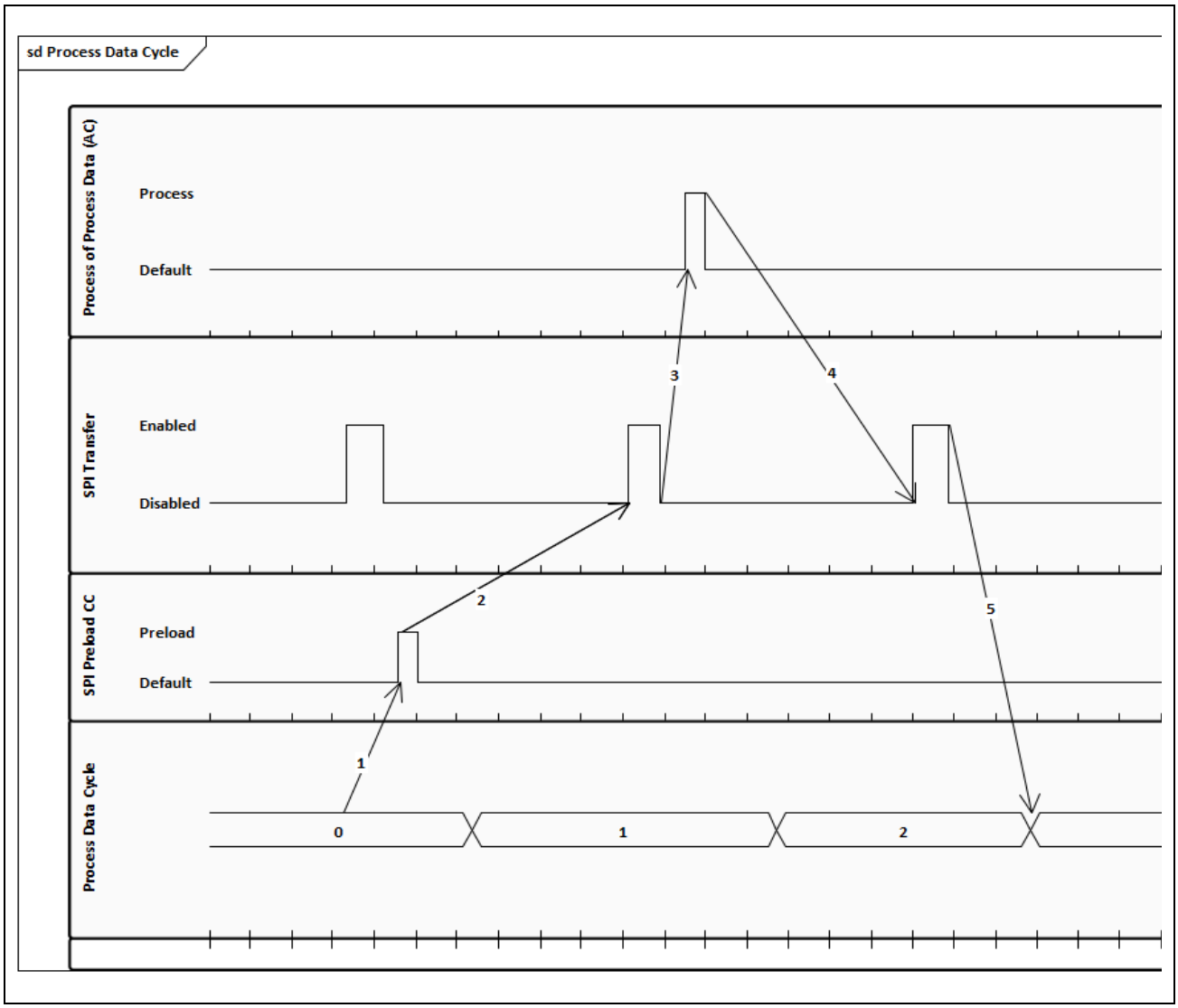

図 5.2 通信サイクル

5.2.2 技術データ

- 転送長：
  - ― サイクリックのみ：72 バイト
  - ― サイクリック＋RPC データ：128 バイト
- ボーレート：Min～Max
- SPI 転送間の遅延：0.5ms～ハートビートタイムアウト（1 秒）
- 最小往復時間：4～6ms（使用するプロトコルおよび設定に依存）

5.2.3 SPI タイミング

以下の簡略図に、アプリケーションコントローラ（AC）が考慮すべき SPI の基本タイミングを示します。

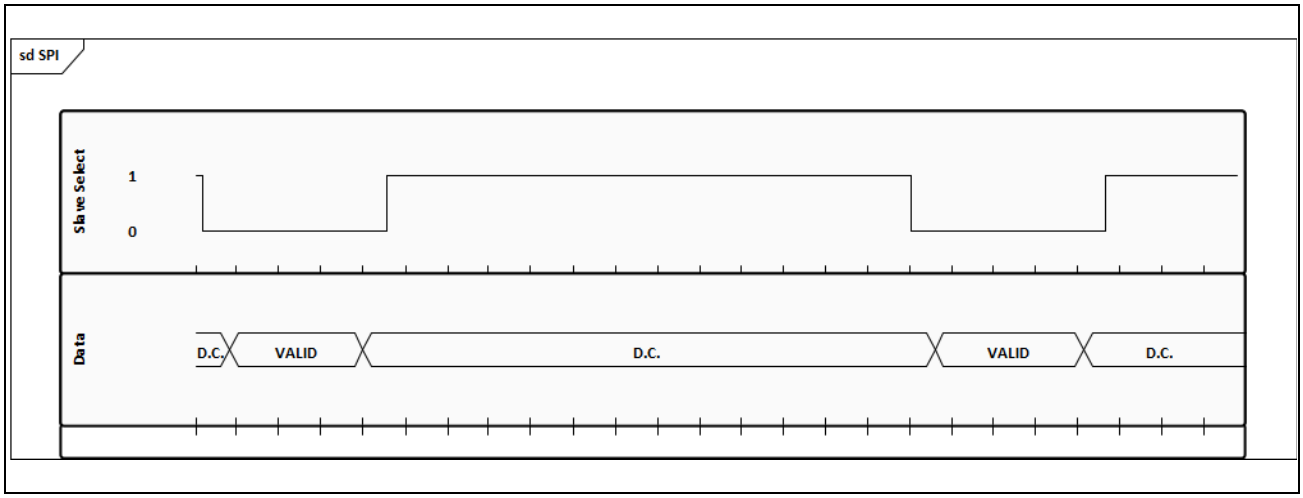

図 5.3 SPI 基本タイミング

5.2.3.1 SPI 速度

通信コントローラ（CC）のデフォルトの SPI 速度は 2MHz です。

5.2.3.2 SPI セットアップタイミング

「**図 5.3 SPI 基本タイミング**」には、AC と CC 間の通信スキームが示されています。通信中に AC が考慮すべきタイミングは以下です。

- SPI セットアップ時間：
  スレーブセレクト信号を使用した CC 起動から最初のデータまでの時間

CC が SPI 経由でデータを受け付けるまでの SPI セットアップ時間は 10ns 以上としてください。

5.2.3.3 SPI サイクル時間

SPI サイクル時間は、2 つの SPI 転送間の間隔を示します。転送間の間隔は、CC 内で適切な処理を行うため、250us 以上確保する必要があります。

5.2.4 SPI フレーム構造

SPI フレームは、R-IN32M3 Module で受け付け可能な「**表 5.1 SPI フレーム構造**」を持つ必要があります。

表 5.1 SPI フレーム構造

バイト 0、1	バイト 2	バイト 3	バイト 4～76	バイト 77～127
オフセット 0x0007 の Fletcher-16 チェックサム（リトルエンディアン）	シーケンス	データ長	サイクリックデータ	RPC データ

同じレイアウトがローカルシーケンスカウンタを持つデバイスから送り返されます。シーケンスカウンタは監視されており、ハートビートタイムアウト中に変更がない場合、シーケンスが再度更新されるまで通信は停止します。

5.2.4.1 Fletcher-16 チェックサム（16 ビット）

計算には Fletcher-16 チェックサムを使用します。

開始インデックスはバイト 4 で、127 で終了します。計算後は、全領域が 0 に設定されている場合に偽陽性とならないよう、0x0007 の値を追加する必要があります。本フレームでは、値は 16 ビット幅であり、リトルエンディアンエンコードを使用します。

5.2.4.2 シーケンスカウンタ（8 ビット）

シーケンスカウンタは、新しいデータとして認識される送信フレームごとにインクリメントされます。

5.2.4.3 データ長（8 ビット）

サイクリックデータのみの転送の場合、データ長は 0～73 バイトの範囲で設定可能です。SPI 経由で RPC を使用する場合、データ長は 124 の固定値である必要があります。

5.3 リモートプロシージャコール（RPC）

R-IN32M Module の C2C で使用される RPC プロトコルは、バイト 77～127 で転送されます。誤送信によるデータ損失をできるだけ最小限に抑えるため、RPC 転送では常に受信確認（ACK）が行われます。また、R-IN32M3 Module は受信した各 RPC フレームを内部でリングバッファに格納し、分割転送が完了するまで待ってから要求処理を行うため、RPC 呼び出しは使用可能な 50 バイトより大きくなる場合があります。

5.3.1 RPC フレーム

5.3.1.1 構造

RPC フレームは、R-IN32M3 Module で受け付け可能な「**表 5.2 RPC フレーム構造**」を持つ必要があります。

表 5.2 RPC フレーム構造

バイト 0、1	バイト 2	バイト 3	バイト 4	バイト 5	バイト 6～49
オフセット 0x0007 の Fletcher-16 チェックサム（リトルエンディアン）	ローカル シーケンス	リモート シーケンス ACK	データ長	フラグ	データ

R-IN32M3 Module は、新たなローカルシーケンスが送信されるたびに次の転送フレーム内の対応する ACK で応答します。ACK が受信されない場合、AC はそのフレームを再送し、ACK を再要求することができます。

5.3.1.2 Fletcher-16 チェックサム（16 ビット）

計算には Fletcher-16 チェックサムを使用します。

開始インデックスはバイト 6 で、データ長で終了します。計算後は、全領域が 0 に設定されている場合に偽陽性とならないよう、0x0007 の値を追加する必要があります。本フレームでは、値は 16 ビット幅であり、リトルエンディアンエンコードを使用します。

5.3.1.3 ローカルシーケンス（8 ビット）

変更されたデータを持つ RPC フレームごとに、シーケンスカウンタがインクリメントされます。観測されない各インクリメンタルフレームに対しては、R-IN32M3 Module は内部リングバッファに転送データを入れ、データ長一致後に RPC コールを処理します。

5.3.1.4 リモートシーケンス ACK（8 ビット）

AC は、処理された各受信 RPC フレームに対して ACK を送信する必要があります。受信フレームが期待されたシーケンスと一致しない場合、AC は直前のシーケンスの ACK を返さず、R-IN32M3 Module からの再送信をトリガする必要があります。再送信されない場合、AC は再同期を行うこともできます。

5.3.1.5 データ長（8 ビット）

RPC データの長さを示します。データ長は、0（ACK 専用フレーム）～44 である必要があります。

5.3.2 フラグ（8 ビット）

5.3.2.1 構造

表 5.3 RPC ヘッダフラグ

ビット 0	ビット 1	ビット 2	ビット 3
同期要求	同期 ACK	予約	要求 ACK

5.3.2.2 同期要求

1 に設定されると、R-IN32M3 Module が RPC 同期に移行します。

5.3.2.3 同期 ACK

同期要求中、両サイドで同期 ACK フラグを設定する必要があります。詳細は、「**5.3.3 RPC 同期**」を参照してください。

5.3.2.4 要求 ACK

対向デバイスからの ACK を強制します。

5.3.3 RPC 同期

R-IN32M3 Module が動作可能になる前、および同期が外れた後は、RPC を同期する必要があります。この処理は、同期要求フラグを 1 に設定することで行われます。

表 5.4 RPC 同期

AC	R-IN32M3 Module
同期要求＝1 ローカルシーケンス＝0 ローカルシーケンス ACK＝0 リモートシーケンス ACK＝0	
	同期要求＝1 ローカルシーケンス＝0 ローカルシーケンス ACK＝0 リモートシーケンス ACK＝0
同期要求＝0 同期 ACK＝1 ローカルシーケンス＝1RPC_Send(<empty>)	
	RPC_Receive() →リモートシーケンス ACK＝1 同期要求＝0 同期 ACK＝1 ローカルシーケンス＝1 RPC_Send(<empty>)
RPC_Receive() →ローカルシーケンス ACK＝1 →リモートシーケンス ACK＝1 同期 ACK＝0 RPC_Start()	
	RPC_Receive() →ローカルシーケンス ACK＝1 RPC_Start()

5.3.4 RPC プロトコル

5.3.4.1 概要

GOAL RPC プロトコルは、仮想プッシュ／ポップスタックを使用して、引数でリモート API 関数を呼び出します。各呼び出しには、呼び出しが正常に行われたときに通常 GOAL_OK に設定される戻り値があります。

5.3.5 RPC 要求/応答

5.3.5.1 構造

RPC 要求/応答は、それぞれ「**表 5.5 RPC 要求構造**」、「**表 5.6 RPC 応答構造**」に示すように構成されています。

表 5.5 RPC 要求構造

バイト 0、1	バイト 2～5	バイト 6～9	バイト 10～13	バイト 14	バイト 15	バイト 16～x	バイト (x+1)～(x+3)
静的識別子 0xaa, 0xee	データ長 バイト 6～x まで	関数 ID（リトルエンディアン）	RPC ID（リトルエンディアン）	CTC ID	フラグ	データ	オフセット 0x0007 の Fletcher-16 チェックサム（リトルエンディアン）

表 5.6 RPC 応答構造

バイト 0、1	バイト 2～5	バイト 6～x	バイト x+1	バイト x+2	バイト (x+3)～(x+4)
静的識別子 0xaa, 0xee	データ長 バイト 6～フラグ（含む）まで	応答データ	CTC ID	フラグ	オフセット 0x0007 の Fletcher-16 チェックサム（リトルエンディアン）

5.3.5.2 静的識別子

2 バイトの静的識別子は、R-IN32M3 Module で新たな RPC 要求または応答の開始を検出するために使用します。識別子がデータバイトにもある場合、RPC 要求／応答のようにスレッドとして処理されないことを Fletcher-16 チェックサムで確認します。

5.3.5.3 データ長

データ長は、「関数 ID」から最終データバイトまでのバイト数を示します。

5.3.5.4 RPC ID

RPC ID は、モジュール ID またはグループ ID を示します。GOAL PROFINET は、RPC ID GOAL_ID_PNIO を使用して、自身の呼び出しを登録します。

5.3.5.5 関数 ID

関数 ID は、特定の関数呼び出しをハンドラに割り当てる、サブ ID として使用されます。

5.3.5.6 CTC ID

CTC ID は、特定の MCTC 内部処理への要求と応答を一致させるために使用します。応答には要求と同じ CTC ID を使用する必要があります。

5.3.5.7 フラグ

フラグは要求タイプを定義します。詳細は、「**表 5.7 RPC 要求／応答フラグ**」を参照してください。

表 5.7 RPC 要求／応答フラグ

ビット 0、1
0：応答 1：要求 2：情報（応答を待たない要求）

5.3.5.8 データ

データ部には、RPC 応答/要求の仮想スタックが入ります。

5.3.5.9 Fletcher-16 チェックサム（16 ビット）

計算には Fletcher-16 チェックサムを使用します。

開始インデックスはバイト 16 で、データ長で終了します。計算後、全領域が 0 に設定されている場合に偽陽性とならないよう、0x0007 の値を追加する必要があります。本フレームでは、値は 16 ビット幅であり、リトルエンディアンエンコードを使用します。

5.4 通信スタック — PROFINET

本節では、GOAL PROFINET が提供する API 関数について記載します。

5.4.1 goal_pnioCfgVendorIdSet — ベンダ ID 設定

ベンダ ID を設定します。

デフォルト：0x02c7（ルネサスエレクトロニクス）

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

表 5.8 goal_pnioCfgVendorIdSet パラメータ

パラメータ	内容
uint16_t idVendor	ベンダ ID

5.4.2 goal_pnioCfgDeviceIdSet — デバイス ID 設定

デバイス ID を設定します。

デフォルト：0x0300

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

表 5.9 goal_pnioCfgDeviceIdSet パラメータ

パラメータ	内容
uint16_t idDevice	デバイス ID

5.4.3 goal_pnioCfgVendorNameSet — ベンダ名設定

ベンダ名を設定します。

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

表 5.10 goal_pnioCfgVendorNameSet パラメータ

パラメータ	内容
const char *strVendor	ベンダ名

5.4.4 goal_pnioCfgPortDescSet — LLDP ポート記述設定

LLDP ポート記述を設定します。

デフォルト：“TestPort”

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

表 5.11 goal_pnioCfgPortDescSet パラメータ

パラメータ	内容
const char *strDescPort	ポート記述

5.4.5 goal_pnioCfgSystemDescSet — LLDP システム記述設定

LLDP システム記述を設定します。

デフォルト：“PROFINET System”

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

表 5.12 goal_pnioCfgSystemDescSet パラメータ

パラメータ	内容
const char *strSystem	システム記述

5.4.6 goal_pnioCfgOrderIdSet — オーダーID 設定

オーダーID を設定します。

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

表 5.13 goal_pnioCfgOrderIdSet パラメータ

パラメータ	内容
const char *strOrder	オーダーID

5.4.7 goal_pnioCfgSerialNumSet — シリアル番号設定

シリアル番号を設定します。

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

表 5.14 goal_pnioCfgSerialNumSet パラメータ

パラメータ	内容
const char *strNumSerial	シリアル番号

5.4.8 goal_pnioCfgHwRevSet — ハードウェアリビジョン設定

ハードウェアリビジョンを設定します。

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

表 5.15 goal_pnioCfgHwRevSet パラメータ

パラメータ	内容
uint16_t idRevHw	ハードウェアリビジョン

5.4.9 goal_pnioCfgSwRevPrefixSet — ソフトウェアリビジョン接頭辞設定

ソフトウェアリビジョン接頭辞を設定します。

デフォルト：“P”

- “V”：オフィシャル
- “R”：リビジョン
- “P”：プロトタイプ
- “U”：テスト中
- “T”：テストデバイス

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

表 5.16 goal_pnioCfgSwRevPrefixSet パラメータ

パラメータ	内容
const char chrRevSwPrefix	ソフトウェアリビジョン接頭辞

デモについては、サンプル 15_config_set を参照してください。

5.4.10 pnioCfgSwRevFuncEnhSet — ソフトウェアリビジョン機能拡張設定

ソフトウェアリビジョンの機能拡張を設定します。

デフォルト：0x50

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

表 5.17 goal_pnioCfgSwRevFuncEnhSet パラメータ

パラメータ	内容
uint8_t idRevSwFuncEnh	ソフトウェアリビジョン機能拡張

5.4.11 goal_pnioCfgSwRevBugfixSet — ソフトウェアリビジョンバグ修正設定

ソフトウェアリビジョンのバグ修正を設定します。

デフォルト：3

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

表 5.18 goal_pnioCfgSwRevBugfixSet パラメータ

パラメータ	内容
uint8_t idRevSwBugfix	ソフトウェアリビジョンバグ修正

5.4.12 goal_pnioCfgSwRevIntChgSet — ソフトウェアリビジョン内部変更設定

ソフトウェアリビジョンの内部変更を設定します。

デフォルト：0x18

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

表 5.19 goal_pnioCfgSwRevIntChgSet パラメータ

パラメータ	内容
uint8_t idRevSwIntChg	ソフトウェアリビジョン内部変更

5.4.13 goal_pnioCfgSwRevCntSet — ソフトウェアリビジョンカウンタ設定

ソフトウェアリビジョンカウンタを設定します。

デフォルト：0x0000

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

表 5.20 goal_pnioCfgSwRevCntSet パラメータ

パラメータ	内容
uint16_t idRevSwRevCnt	ソフトウェアリビジョンカウンタ

5.4.14 goal_pnioCfgIm1TagFuncSet — I&M1 タグ機能設定

I&M1 タグ機能を設定します。

デフォルト：“”

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

表 5.21 goal_pnioCfgIm1TagFuncSet パラメータ

パラメータ	内容
const char *strIm1TagFunc	I&M1 タグ機能

5.4.15 goal_pnioCfgIm1TagLocSet — I&M1 タグ位置設定

I&M1 タグ位置を設定します。

デフォルト：“”

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

表 5.22 goal_pnioCfgIm1TagLocSet パラメータ

パラメータ	内容
const char *strIm1TagLoc	I&M1 タグ位置

5.4.16 goal_pnioCfgIm2DateSet — I&M2 日付設定

I&M2 日付を設定します。

デフォルト：“”

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

表 5.23 goal_pnioCfgIm2DateSet パラメータ

パラメータ	内容
const char *strIm2Date	I&M2 日付

5.4.17 goal_pnioCfgIm3DescSet — I&M3 記述設定

I&M3 記述を設定します。

デフォルト：“”

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

表 5.24 goal_pnioCfgIm3DescSet パラメータ

パラメータ	内容
const char *strIm3Desc	I&M3 記述

5.4.18 goal_pnioCfgIm4SigSet — I&M4 署名（機能セーフティ）設定

I&M4 署名を設定します。

デフォルト：“”

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

表 5.25 goal_pnioCfgIm4SigSet パラメータ

パラメータ	内容
const char *strIm4Sig	I&M4 署名

5.4.19 goal_pnioCfgLldpOrgExtSet — LLDP 組織的固有の拡張機能設定

LLDP 組織的固有の拡張機能を設定します。

デフォルト：GOAL_TRUE

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

注意：整合性が失われる可能性があるため、本関数は使用しないでください。

表 5.26 goal_pnioCfgLldpOrgExtSet パラメータ

パラメータ	内容
GOAL_BOOL_T flgLldpOrgExt	LLDP 組織的固有拡張フラグ

5.4.20 goal_pnioCfgLldpOptTlvSet — LLDP 任意 TLV パラメータ設定

LLDP の任意の TLV パラメータを設定します。

デフォルト：GOAL_TRUE

本パラメータには、

- ポート記述
- システム名
- システム記述
- システム機能
- 管理アドレス
- オブジェクト ID

があります。

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

注意：整合性が失われる場合があるため、本関数は使用しないでください。

表 5.27 goal_pnioCfgLldpOptTlvSet パラメータ

パラメータ	内容
GOAL_BOOL_T flgLldpOptTlv	LLDP 任意 TLV パラメータフラグ

5.4.21 goal_pnioCfgLldpGenMacSet — LLDP ポート MAC アドレス生成設定

LLDP ポートの MAC アドレス自動生成を設定します。GOAL_TRUE に設定されると、ホストポートの MAC アドレスにポート ID を追加することによって、LLDP ポート固有の MAC アドレスが自動生成されます。

デフォルト：GOAL_TRUE

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

注意：本機能が有効（デフォルト）の場合、各デバイスは 2 つ以上の MAC アドレス（実アドレス＋ポートごとのアドレス）を昇順に使用します。特定の仮想ポートの MAC アドレスを使用する場合、本機能を無効にし、ドライバで要求（GOAL_ETH_PORT_HOST およびポート固有の要求）ごとに異なる MAC アドレスを割り付ける必要があります。

表 5.28 goal_pnioCfgLldpGenMacSet パラメータ

パラメータ	内容
GOAL_BOOL_T flgLldpGenMac	自動 LLDP MAC 生成フラグ

5.4.22 goal_pnioCfgIm14SupportSet ― I&M 1-4 サポート設定

GOAL PROFINET スタックの I&M 1-4 サポートを設定します。GOAL_FALSE に設定されると、I&M 1-4 要求はスタックによって拒否されます。

デフォルト：GOAL_TRUE

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

注意：整合性が失われる場合があるため、本関数の使用時には注意が必要です。

表 5.29 goal_pnioCfgIm14SupportSet パラメータ

パラメータ	内容
GOAL_BOOL_T flgIm14Support	I&M 1-4 サポートフラグ

5.4.23 goal_pnioCfgIm14CbSet ― I&M 1-4 コールバック設定

GOAL PROFINET スタックの I&M 1-4 コールバックを設定します。GOAL_TRUE に設定されると、アプリケーションコールバックによって I&M 1-4 取得・設定要求の処理が行われます。

デフォルト：GOAL_TRUE

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

注意：整合性が失われる場合があるため、本関数の使用時には注意が必要です。

表 5.30 goal_pnioCfgIm14CbSet パラメータ

パラメータ	内容
GOAL_BOOL_T flgIm14Cb	I&M 1-4 コールバックフラグ

5.4.24 goal_pnioCfgIm0CbSet ― I&M 0 コールバック設定

GOAL PROFINET スタックの I&M 0 コールバックを設定します。GOAL_TRUE に設定されると、アプリケーションコールバックによって I&M 0 取得・設定要求の処理が行われます。

デフォルト：GOAL_TRUE

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

注意：整合性が失われる場合があるため、本関数の使用時には注意が必要です。

表 5.31 goal_pnioCfgIm0CbSet パラメータ

パラメータ	内容
GOAL_BOOL_T flgIm0Cb	I&M 0 コールバックフラグ

5.4.25 goal_pnioCfgIm0FilterDataCbSet — I&M 0 フィルタデータコールバック設定

GOAL PROFINET スタックの I&M 0 フィルタデータのコールバックを設定します。GOAL_TRUE に設定されると、アプリケーションコールバックによって I&M 0 フィルタデータ取得・設定要求の処理が行われます。

デフォルト：GOAL_TRUE

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

注意：整合性が失われる場合があるため、本関数の使用時には注意が必要です。

表 5.32 goal_pnioCfgIm0FilterDataCbSet パラメータ

パラメータ	内容
GOAL_BOOL_T flgIm0FilterCb	I&M 0 フィルタコールバックフラグ

5.4.26 goal_pnioCfgRecDataBusyBufsizeSet — 記録処理格納数設定

並行記録処理の回数を設定します。

デフォルト：2

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

表 5.33 goal_pnioCfgRecDataBusyBufsizeSet パラメータ

パラメータ	内容
GOAL_BOOL_T cntRecDataBusyBufsize	記録処理格納数

5.4.27 goal_pnioCfgRpcFragReqLenMaxSet — 最大記録サイズ設定

記録要求のバイトの最大サイズを設定します。GSDML ファイルの MaxSupportedRecordSize 属性と一致している必要があります。

デフォルト：4068

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

注意：デフォルト値より小さい値に変更すると、整合性が失われる可能性があります。

表 5.34 goal_pnioCfgRpcFragReqLenMaxSet パラメータ

パラメータ	内容
unsigned int sizeRpcFragMaxReqLen	最大記録サイズ

5.4.28 goal_pnioCfgRpcFragMaxCntSet — 最大 RPC フラグメント番号設定

本関数は廃止されています。GOAL PROFINET スタックでは、現在 64 個のフラグメントフレームを使用できます。

5.4.29 goal_pnioCfgRpcFragEnableSet — RPC フラグメンテーション設定

RPC フラグメンテーション機能を設定します。GOAL_TRUE RPC に設定されると、フラグメンテーションが有効になります。

デフォルト：GOAL_TRUE

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

注意：整合性が失われる場合があるため、本関数は使用しないでください。

表 5.35 goal_pnioCfgRpcFragEnableSet パラメータ

パラメータ	内容
GOAL_BOOL_T flgRpcFragSupport	RPC フラグメンテーション有効フラグ

5.4.30 goal_pnioCfgRpcSessionMaxCntSet — 最大 RPC セッション数設定

RPC セッションの最大数を設定します。

デフォルト：8

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

注意：整合性が失われる場合があるため、本関数は使用しないでください。

表 5.36 goal_pnioCfgRpcSessionMaxCntSet パラメータ

パラメータ	内容
unsigned int numRpcSessions	RPC セッション数

5.4.31 goal_pnioDmSubslotAdd — サブスロットデータのマッピング

API 関数 goal_pnioDmSubslotAdd は、任意のサブスロットの入出力データを DM の任意のインスタンスにマッピングします。

表 5.37 goal_pnioDmSubslotAdd API 内容

パラメータ	内容
GOAL_PNIO_T *pPnio	GOAL PROFINET インスタンス
uint32_t idMiDmPeerFrom	CC からのデータ受信処理
uint32_t idMiDmPeerTo	CC へのデータ送信処理
GOAL_MI_DM_PART_T **ppPartDataOut	DM パーティションを格納する出力ポインタ
GOAL_MI_DM_PART_T **ppPartDataIn	DM パーティションを格納する出力ポインタ
uint32_t idApi	API ID
uint16_t idSlot	スロット ID
uint16_t idSubslot	サブスロット ID
uint32_t lenDataOut	出力データ長
uint32_t lenDataIn	入力データ長

5.4.32 goal_pnioDmSubslotIoxsAdd — サブスロット IOCS/IOPS のマッピング

API 関数 goal_pnioDmSubslotIoxsAdd は、任意のサブスロットの IOCS、IOPS の状態を DM の任意のインスタンスにマッピングします。

表 5.38 goal_pnioDmSubslotIoxsAdd API 内容

パラメータ	内容
GOAL_PNIO_T *pPnio	GOAL PROFINET インスタンス
uint32_t idMiDmPeerFrom	CC からのデータ受信処理
uint32_t idMiDmPeerTo	CC へのデータ送信処理
GOAL_MI_DM_PART_T **ppPartIocsOut	DM パーティションを格納する出力ポインタ
GOAL_MI_DM_PART_T **ppPartIopsOut	DM パーティションを格納する出力ポインタ
GOAL_MI_DM_PART_T **ppPartIocsIn	DM パーティションを格納する出力ポインタ
GOAL_MI_DM_PART_T **ppPartIopsIn	DM パーティションを格納する出力ポインタ
uint32_t idApi	API ID
uint16_t idSlot	スロット ID
uint16_t idSubslot	サブスロット ID

5.4.33 goal_pnioDmApduAdd — APDU ステータスのマッピング

API 関数 goal_pnioDmApduAdd は、APDU ステータスを DM の任意のインスタンスにマッピングします。

表 5.39 goal_pnioDmApduAdd API 内容

パラメータ	内容
GOAL_PNIO_T *pPnio	GOAL PROFINET インスタンス
uint32_t idMiDmPeerTo	CC へのデータ送信処理（無視）
GOAL_MI_DM_PART_T **ppPartApduOut	DM パーティションを格納する出力ポインタ

5.4.34 goal_pnioDmDpAdd — データプロバイダステータスのマッピング

API 関数 goal_pnioDmDpAdd は、データプロバイダのステータスを DM の任意のインスタンスにマッピングします。

表 5.40 goal_pnioDmDpAdd API の内容

パラメータ	内容
GOAL_PNIO_T *pPnio	GOAL PROFINET インスタンス
uint32_t idMiDmPeerTo	CC へのデータ送信処理（無視）
GOAL_MI_DM_PART_T **ppPartDp	DM パーティションを格納する出力ポインタ

5.5 アプリケーションコールバック – PROFINET

5.5.1 概要

PROFINET を最大限に活用するために、スタックはユーザが複数段階のプロトコルでやり取りを行うことを可能にします。例えば、ユーザはアプリケーションレディ信号を PLC に送信することを保留し、特定スロットへの記録データの読み書きを処理することができます。

コールバックシステムを使用するためには、まず、アプリケーションとやり取りを行なうときに、その都度スタックで呼び出し可能なコールバック関数を提供します。コールバック関数は以下のプロトタイプを持つ必要があります。

```
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 */
);
```

プロトタイプは、コールバック関数へのポインタを第 2 パラメータとして取る goal_pnioNew への呼び出しとともに登録されます。

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

これにより、appl_pnioCb が呼び出されるたびに、GOAL_PNIO_CB_ID_T 値は呼び出しの理由を示します。GOAL_PNIO_CB_DATA_T 構造体は、各配列要素が GOAL_PNIO_CB_ID_T 値に応じて特別な意味を持つ、複数の共用体の配列を持ちます。
以下の項で、使用可能なコールバック ID の詳細について説明します。

5.5.2 GOAL_PNIO_CB_ID_ALARM_ACK_TIMEOUT ― アラーム ACK 待機中タイムアウト

アラーム ACK の待機中に APMS でタイムアウトが発生したことを示します。

表 5.41 TIMEOUT ― アラーム ACK 待機中タイムアウト

パラメータ	内容
cb->data[0].idAr	Application Relations ID
cb->data[1].u16	タイムアウトアラームシーケンス番号

5.5.3 GOAL_PNIO_CB_ID_ALARM_NOTIFY_ACK ― アラーム通知 ACK 受信

アラーム通知 ACK を受信したことを示します。

表 5.42 アラーム通知 ACK 受信

パラメータ	内容
cb->data[0].idAr	Application Relations ID
cb->data[1].u16	アラーム優先度（低、高）
cb->data[2].pAlarmNotifyAck	GOAL_PNIO_ALARM_NOTIFY_ACK_T 構造体ポインタ

5.5.4 GOAL_PNIO_CB_ID_ALARM_NOTIFY — アラーム通知受信

アラーム通知を受信したことを示します。コールバックを処理しない場合や、GOAL_OK が返された場合、PROFINET スタックは受信したアラーム通知の ACK を自動で返信します。

表 5.43 アラーム通知受信

パラメータ	内容
cb->data[0].idAr	Application Relations ID
cb->data[1].u16	アラーム優先度（低、高）
cb->data[2].pAlarmNotify	GOAL_PNIO_ALARM_NOTIFY_T 構造体ポインタ
cb->data[3].u32	ユーザデータ長
cb->data[4].pCu8	ユーザデータポインタ

5.5.5 GOAL_PNIO_CB_ID_APPL_READY — アプリケーション準備完了応答受信

アプリケーション準備完了の応答を受信したことを示します。

戻り値は GOAL_OK となります。

表 5.44 アプリケーション準備完了応答受信

パラメータ	内容
cb->data[0].idAr	Application Relations ID

5.5.6 GOAL_PNIO_CB_ID_BLINK — 点滅要求

DCP 信号要求を受信したことを示し、コールバックデータを必要な情報で埋めます。

cb->data[0].stateDcpBlink は、最初の信号要求（START）と、信号をトグルするための正確な時間（TOGGLE）、信号位相の終了時（FINISH）をアプリケーションに伝えます。この情報は、アプリケーションが自身で信号を完全に指定したい場合に使用することができます。

次に、信号インジケータを LED または点滅関数に直接引き渡すために使用される GOAL_PNIO_DCP_LIGHT_OFF または GOAL_PNIO_DCP_LIGHT_ON を含む cb->data[1].stateDcpLight データを使用することができます。

バリアント 1 または 2 を使用する場合、コールバックで GOAL_OK_SUPPORTED を返す必要があります。それ以外の場合、GOAL_PNIO_LED_SIGNAL 定義で goal_targetSetLeds を呼び出すことで、スタック自身で点滅処理を行います。スタック内部処理には、goal_targetGetLeds、goal_targetSetLeds の実装が必須です。

表 5.45 点滅要求

パラメータ	内容
cb->data[0].stateDcpBlink	DCP 点滅状態（開始、トグル、終了）
cb->data[1].stateDcpLight	DCP 点灯状態（オン、オフ）

戻り値	内容
GOAL_OK	スタックの内部信号処理
GOAL_OK_SUPPORTED	アプリケーションの信号処理

```
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 — 接続要求終了

接続要求が完全に処理されたことを示します。エラーステータスを設定することで、アプリケーションは要求を取り消し、GOAL_OK 以外の値を返すことができます。

表 5.46 接続要求終了

パラメータ	内容
cb->data[0].idAr	Application Relations ID
cb->data[1].pStatus	PROFINET ステータスへのポインタ

5.5.8 GOAL_PNIO_CB_ID_CONNECT_REQUEST — 接続要求

接続要求の処理が開始されたことを示します。パラメータへのポインタは ***NULL*** に設定されます。

5.5.9 GOAL_PNIO_CB_ID_CONNECT_REQUEST_EXP_START — 期待サブモジュールブロック開始

期待されたサブモジュールブロックの開始を示します。各モジュールに対する要求コールバックが来る前に、すべてのモジュールを取り外す際に使用できます。

表 5.47 期待サブモジュールブロック開始

パラメータ	内容
cb->data[0].idAr	Application Relations ID

5.5.10 GOAL_PNIO_CB_ID_END_OF_PARAM — パラメータ終了受信

パラメータ終了の情報を受信したことを示します。

表 5.48 パラメータ終了受信

パラメータ	内容
cb->data[0].idAr	Application Relations ID

5.5.11 GOAL_PNIO_CB_ID_END_OF_PARAM_PLUG — プラグパラメータ終了受信

プラグパラメータ終了の情報を受信したことを示します。

表 5.49 プラグパラメータ終了受信

パラメータ	内容
cb->data[0].idAr	Application Relations ID
cb->data[1].u16	プラグ処理

5.5.12 GOAL_PNIO_CB_ID_EXP_SUBMOD — 期待サブモジュール

即時に取り外し可能な、期待されたサブモジュールを示します。

表 5.50 期待サブモジュール

パラメータ	内容
cb->data[0].idAr	Application Relations ID
cb->data[1].u32	API
cb->data[2].u16	スロット番号
cb->data[3].u16	サブスロット番号
cb->data[4].u32	モジュール識別番号
cb->data[5].u32	サブモジュール識別番号
cb->data[6].valBool	モジュールフラグ（true：モジュール、false：サブモジュール）

5.5.13 GOAL_PNIO_CB_ID_FACTORY_RESET — ファクトリリセット

ファクトリリセットを示します。パラメータへのポインタは NULL に設定されます。

5.5.14 GOAL_PNIO_CB_ID_IO_DATA_TIMEOUT — サイクリックタイムアウト

出力エンドポイントがタイムアウトしたことを示します。

5.5.15 GOAL_PNIO_CB_ID_NET_IP_SET — IP 設定更新

IP 設定が更新されたことを示します。内部変更フラグは、変更が PROFINET DCP (PN_TRUE)によって行われたのか、または DHCP (PN_FALSE)のような外部呼び出し元によって行われたのかを示します。

表 5.51 IP 設定更新

パラメータ	内容
cb->data[0].u32	IP アドレス
cb->data[1].u32	ネットマスク
cb->data[2].u32	ゲートウェイ
cb->data[3].valBool	一時設定フラグ
cb->data[4].valBool	内部更新フラグ

5.5.16 GOAL_PNIO_CB_ID_NEW_AR — 新規 Application Relations

新規 Application Relations を示します。

戻り値が GOAL_OK 以外の場合、AR を削除し、エラーを返します。

表 5.52 新規 Application Relations

パラメータ	内容
cb->data[0].idAr	Application Relations ID

5.5.17 GOAL_PNIO_CB_ID_NEW_IO_DATA — 新規 I/O データ

新規 I/O データの受信を示します。

表 5.53 新規 I/O データ

パラメータ	内容
cb->data[0].u16	サイクリックフレーム ID
cb->data[1].u16	データ長
cb->data[2].pCu8	データポインタ

5.5.18 GOAL_PNIO_CB_ID_PLUG_READY — プラグ準備完了応答受信

プラグ準備完了の応答を受信したことを示します。

表 5.54 プラグ準備完了応答受信

パラメータ	内容
cb->data[0].idAr	Application Relations ID

5.5.19 GOAL_PNIO_CB_ID_READ_RECORD — 記録データ読み出し

スタック自身で処理できない記録データを読み出す必要があることを示します。スタックは、このコールバックが呼び出される前に、API、スロット、サブスロットの組み合わせが有効かどうかを確認します。

本関数が GOAL_OK を返す場合、「無効インデックス」で要求が拒否されます。

要求を処理するには、コールバックは GOAL_OK_SUPPORTED を返す必要があります。直接応答することも、API goal_pnioRecReadFinish を呼び出して後で応答することもできます。また、現際の PROFINET ステータスは、API goal_pnioRecReadFinish を介して設定できます。

応答が GOAL_OK または GOAL_OK_SUPPORTED ではない場合、スタックはアプリケーションリードエラーを自動で返します。

パラメータ「シーケンス番号処理」は、要求の有効期限切れを検出するために応答が送信された場合、後で使用します。

表 5.55 記録データ読み出し

パラメータ	内容
cb->data[0].pStatus	未使用（PROFINET ステータスへのポインタ）
cb->data[1].idAr	Application Relations ID
cb->data[2].u32	API
cb->data[3].u16	スロット
cb->data[4].u16	サブスロット
cb->data[5].u16	記録インデックス
cb->data[6].pU8	未使用（記録データ格納用ポインタ）
cb->data[7].u32	最大記録リードデータ長
cb->data[8].i32	内部記録処理
cb->data[9].u32	シーケンス番号処理

コールバックコード例：

```
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 — Application Relations リリース

Application Relations がリリースされたことを示します。

表 5.56 Application Relations リリース

パラメータ	内容
cb->data[0].idAr	Application Relations ID

5.5.21 GOAL_PNIO_CB_ID_RESET_TO_FACTORY — ファクトリリセット

ファクトリリセット要求を示します。スタック外で別の要求処理があった場合、アプリケーションは GOAL_OK_SUPPORTED を返す必要があります。

アプリケーションがコールバックから GOAL_OK または GOAL_OK_SUPPORTED を返さない場合には、DCP エラー「0x04 サブオプション未設定」が DCP 設定要求の送信者に返されます。

表 5.57 ファクトリリセット

パラメータ	内容
cb->data[0].u16	ファクトリリセット動作

5.5.22 GOAL_PNIO_CB_ID_STATION_NAME — ステーション名変更

ステーション名の変更を示します。永久フラグが GOAL_TRUE に設定されると、ステーション名が NVS に格納されます。それ以外の場合、ステーション名は一時的に格納されるのみで、NVS 値はクリアされます。

表 5.58 ステーション名変更

パラメータ	内容
cb->data[0].pCu8	ステーション名
cb->data[1].u32	ステーション名長さ
cb->data[2].valBool	永久フラグ

5.5.23 GOAL_PNIO_CB_ID_WRITE_RECORD — 記録データ書き込み

スタック自身で処理できない記録データを書き込む必要があることを示します。スタックは、このコールバックが呼び出される前に、API、スロット、サブスロットの組み合わせが有効かどうかを確認します。

本関数が GOAL_OK を返した場合、「無効インデックス」で要求が拒否されます。

要求を処理するには、コールバックは GOAL_OK_SUPPORTED を返す必要があります。直接応答することも、API goal_pnioRecWriteFinish を呼び出して後で応答することもできます。また、現際の PROFINET ステータスは、API goal_pnioRecWriteFinish を介して設定できます。

応答が GOAL_OK または GOAL_OK_SUPPORTED ではない場合、スタックはアプリケーションリードエラーを自動で返します。

パラメータ「シーケンス番号処理」は、要求の有効期限切れを検出するために応答が送信された場合、後で使用します。

表 5.59 記録データ書き込み

パラメータ	内容
cb->data[0].pStatus	未使用（PROFINET ステータスへのポインタ）
cb->data[1].idAr	Application Relations ID
cb->data[2].u32	API
cb->data[3].u16	スロット
cb->data[4].u16	サブスロット
cb->data[5].u16	記録インデックス
cb->data[6].pCu8	記録データの読み出し用ポインタ
cb->data[7].u32	リードデータ長
cb->data[8].i32	内部記録処理
cb->data[9].valBool	サブスロットロック状態フラグ
cb->data[10].u32	シーケンス番号処理

コールバックコード例：

```
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 — スタック初期化

PROFINET スタックが完全に初期化されたことを示します。パラメータへのポインタは NULL に設定されます。この時点でデバイス設定を作成することは安全です。

5.5.25 GOAL_PNIO_CB_ID_LLDP_UPDATE — LLDP 更新

対向ポートのデバイスが変更したことを示します。

表 5.60 LLDP 更新

パラメータ	内容
cb->data[0].u32	GOAL Ethernet ポート ID

5.5.26 GOAL_PNIO_CB_ID_CONN_REQ_EXP_FINISH — 接続要求の期待サブモジュールブロック終了

このコールバックは、接続要求における期待サブモジュールブロックが解析された後に呼び出されます。

表 5.61 接続要求の期待サブモジュールブロック終了

パラメータ	内容
cb->data[0].idAr	Application Relations ID

5.5.27 GOAL_PNIO_CB_ID_STATION_NAME_VERIFY — DCP ステーション名検証

アプリケーションで DCP ステーション名の設定要求を検証します。

GOAL エラーステータスを返すと、設定要求は拒否されます。

表 5.62 DCP ステーション名検証

パラメータ	内容
cb->data[0].pCu8	ステーション名へのポインタ（未終端）
cb->data[1].u32	ステーション名長さ
cb->data[2].valBool	永久フラグ（永久＝true）

5.5.28 GOAL_PNIO_CB_ID_NET_IP_SET_VERIFY — DCP IP 設定検証

アプリケーションで DCP IP 設定の設定要求を検証します。

GOAL エラーステータスを返すと、設定要求は拒否されます。

表 5.63 DCP IP 設定検証

パラメータ	内容
cb->data[0].u32	IP アドレス
cb->data[1].u32	ネットマスク
cb->data[2].u32	ゲートウェイ
cb->data[3].valBool	一時フラグ（一時＝true）

5.6 通信スタック — EtherNet/IP

本節では、GOAL EtherNet/IP が提供する API 関数について記載します。

5.6.1 goal_eipCfgVendorIdSet

EtherNet/IP スタックインスタンスのベンダ ID を設定します。ベンダ ID は ODVA によって割り当てられます。

デフォルト値：1105

表 5.64 goal_eipCfgVendorIdSet パラメータ

パラメータ	内容
uint16_t vendorId	ベンダ ID

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

5.6.2 goal_eipCfgDeviceTypeSet

EtherNet/IP スタックインスタンスのデバイスタイプを設定します。有効な値は CIP 規格で定義されます。

デフォルト値：0x2B

表 5.65 goal_eipCfgDeviceTypeSet パラメータ

パラメータ	内容
uint16_t deviceType	デバイスタイプ

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

5.6.3 goal_eipCfgProductCodeSet

EtherNet/IP スタックインスタンスの製品コードを設定します。この値はデバイスベンダで定義されます。

デフォルト値：768

表 5.66 goal_eipCfgProductCodeSet パラメータ

パラメータ	内容
uint16_t productCode	製品コード

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

5.6.4 goal_eipCfgRevisionSet

EtherNet/IP スタックインスタンスのリビジョンを設定します。リビジョンはメジャー番号とマイナー番号で構成され、デバイスの動作に影響を及ぼすファームウェアの変更を表します。

デフォルト値：1.1

表 5.67 goal_eipCfgRevisionSet パラメータ

パラメータ	内容
uint8_t revMajor	メジャーバージョン番号
uint8_t revMinor	マイナーバージョン番号

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

5.6.5 goal_eipCfgSerialNumSet

EtherNet/IP スタックインスタンスのシリアル番号を設定します。シリアル番号はベンダによって割り当てられます。番号はデバイス固有である必要があります。

デフォルト値：1

表 5.68 goal_eipCfgSerialNumSet パラメータ

パラメータ	内容
uint32_t serial	シリアル番号

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

5.6.6 goal_eipCfgProductNameSet

EtherNet/IP スタックインスタンスの製品名を設定します。製品名はベンダによって割り当てられます。最大長は 32 文字です。

デフォルト値：“EtherNet/IP Adapter”

表 5.69 goal_eipCfgProductNameSet パラメータ

パラメータ	内容
const char *strName	製品名

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

5.6.7 goal_eipCfgDomainNameSet

EtherNet/IP スタックインスタンスのデフォルトドメイン名を設定します。有効な設定が見つからない場合には、デバイスのドメイン名として使用します。最大長は 48 文字です。

デフォルト値： “port.de”

表 5.70 goal_eipCfgDomainNameSet パラメータ

パラメータ	内容
const char *strName	デフォルトドメイン名

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

5.6.8 goal_eipCfgHostNameSet

EtherNet/IP スタックインスタンスのデフォルトホスト名を設定します。有効な設定が見つからない場合には、デバイスのホスト名として使用します。最大長は 64 文字です。

デフォルト値： “eipdevice”

表 5.71 goal_eipCfgHostNameSet パラメータ

パラメータ	内容
const char *strName	デフォルトホスト名

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

5.6.9 goal_eipCfgNumExplicitConSet

デバイスが一度に処理できる明示的接続の最大数を設定します。

デフォルト値：6

表 5.72 goal_eipCfgNumExplicitConSet パラメータ

パラメータ	内容
uint16_t num	明示的接続の数

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

5.6.10 goal_eipCfgNumImplicitConSetImpl

デバイスが一度に処理できる暗黙的接続の最大数を設定します。

デフォルト値：6

表 5.73 goal_eipCfgNumImplicitConSetImpl パラメータ

パラメータ	内容
uint16_t num	暗黙的接続の数

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

5.6.11 goal_eipCfgEthLinkCountersOn

Ethernet リンクの属性 4, 5, 12, 13 に対するサポートを無効にします。
本機能は、関数の有効設定(GOAL_TRUE)に関係なくデフォルトで有効となっています。デフォルト値：GOAL_TRUE

表 5.74 goal_eipCfgEthLinkCountersOn パラメータ

パラメータ	内容
GOAL_BOOL_T enable	機能有効／無効

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

5.6.12 goal_eipCfgEthLinkControlOn

Ethernet リンクの属性 6 に対するサポートを無効にします。
本機能は、関数の有効設定(GOAL_TRUE)に関係なくデフォルトで有効となっています。デフォルト値：GOAL_TRUE

表 5.75 goal_eipCfgEthLinkControlOn パラメータ

パラメータ	内容
GOAL_BOOL_T enable	機能有効／無効

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

5.6.13 goal_eipCfgChangeEthAfterResetOn

Ethernet リンクの速度と二重モードの変更には、デバイスのリセットを必要とします。

デフォルト値：GOAL_FALSE

表 5.76 goal_eipCfgChangeEthAfterResetOn パラメータ

パラメータ	内容
GOAL_BOOL_T enable	機能有効／無効

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

5.6.14 goal_eipCfgChangeIpAfterResetOn

IP アドレスの変更には、デバイスのリセットを必要とします。

デフォルト値：GOAL_FALSE

表 5.77 goal_eipCfgChangeIpAfterResetOn パラメータ

パラメータ	内容
GOAL_BOOL_T enable	機能有効／無効

5.6.15 goal_eipCfgNumSessionsSet

デバイスが一度に処理できるカプセル化セッション数を設定します。

デフォルト値：20

表 5.78 goal_eipCfgNumSessionsSet パラメータ

パラメータ	内容
uint16_t num	セッション数

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

5.6.16 goal_eipCfgTickSet

各ティックのミリ秒数を設定します。本スタックは、1 ティックを最小時間単位として使用します。

デフォルト値：10

表 5.79 goal_eipCfgTickSet パラメータ

パラメータ	内容
uint32_t ticks	ms 単位の 1 ティックの長さ

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

注意： この関数は非推奨であり、効果はありません。 古いファームウェアバージョンとの互換性のために保持されています。

5.6.17 goal_eipCfgO2TRunIdleHeaderOn

消費される（発信者から対象者への）サイクリックデータの Run/Idle ヘッダを無効にします。
本機能は、関数の有効設定(GOAL_TRUE)に関係なくデフォルトで有効となっています。

デフォルト値：GOAL_TRUE

表 5.80 goal_eipCfgO2TRunIdleHeaderOn パラメータ

パラメータ	内容
GOAL_BOOL_T enable	機能有効／無効

```
res = goal_eipCfgO2TRunIdleHeaderOn(GOAL_FALSE);
```

5.6.18 goal_eipCfgT2ORunIdleHeaderOn

生成される（発信者から対象者への）サイクリックデータの Run/Idle ヘッダを有効にします。

デフォルト値：GOAL_FALSE

表 5.81 goal_eipCfgT2ORunIdleHeaderOn パラメータ

パラメータ	内容
GOAL_BOOL_T enable	機能有効／無効

```
res = goal_eipCfgT2ORunIdleHeaderOn(GOAL_FALSE);
```

5.6.19 goal_eipCfgQoSOn

QoS オブジェクト 属性 4, 5, 6, 7 のサポートを無効にします。
本機能は、関数の有効設定(GOAL_TRUE)に関係なくデフォルトで有効となっています。デフォルト値：GOAL_TRUE

表 5.82 goal_eipCfgQoSOn パラメータ

パラメータ	内容
GOAL_BOOL_T enable	機能有効／無効

```
res = goal_eipCfgQoSOn(GOAL_FALSE);
```

5.6.20 goal_eipCfgNumDelayedEncapMsgSet

同時に遅らせることが可能なカプセル化メッセージ数を設定します。

デフォルト値：2

表 5.83 goal_eipCfgNumDelayedEncapMsgSet パラメータ

パラメータ	内容
uint16_t num	メッセージ数

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

5.6.21 goal_eipCfgDhcpOn

プラットフォームでサポートされている場合に、DHCP クライアントを有効にします。

デフォルト値：GOAL_FALSE

表 5.84 goal_eipCfgDhcpOn パラメータ

パラメータ	内容
GOAL_BOOL_T enable	機能有効／無効

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

5.6.22 goal_eipCfgDlrOn

DLR オブジェクトのサポートを有効にします。本機能は、DLR スタックとハードウェアのサポートを必要とします。

デフォルト値：GOAL_FALSE

表 5.85 goal_eipCfgDlrOn パラメータ

パラメータ	内容
GOAL_BOOL_T enable	機能有効／無効

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

5.6.23 goal_eipCfgAcdOn

ACD(Address Conflict Detection)のサポートを有効にします。本機能を実行するには TCP/IP オブジェクト属性 10 を 1 (Enable ACD) とする必要があります。

デフォルト値：GOAL_FALSE

表 5.86 goal_eipCfgAcdOn パラメータ

パラメータ	内容
GOAL_BOOL_T enable	機能有効／無効

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

5.6.24 goal_eipCfgAcdConflictFallbackIp

ACD がアドレス競合を検出した場合は、フォールバック IP 構成を構成します。 静的 IP を持つデバイスとして構成されている場合、デバイスは構成された IP を適用します。

TCP/IP オブジェクト 属性 3 が DHCP に設定されている場合、デバイスは一定のタイムアウト後に DHCP プロセスを開始します。

- ipAddress のみを引数に設定(subnetMAsk, gateway = 0)：エラー状態に移行し、何もしない
- 有効な IP 構成を引数に設定：フォールバック IP 構成を設定

表 5.87 goal_eipCfgAcdConfilctFallbackIp パラメータ

パラメータ	内容
uint32_t ipAddress	フォールバック IP アドレス
uint32_t subnetMask	フォールバックサブネットマスク
uint32_t gateway	フォールバックゲートウェイ

```
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 アプリケーションコールバック — EtherNet/IP

5.7.1 概要

EtherNet/IP を最大限に活用するために、スタックはユーザが複数段階のプロトコルでやり取りを行うことを可能にします。例えば、ユーザは新たなアセンブリデータが来てからコールバックを受信することができます。

スタックは、goal_eipNew()関数を通じて登録されたコールバックハンドラを呼び出します。コールバックハンドラは GOAL_STATUS_T 値を返します。

表 5.88 コールバックハンドラのパラメータ

パラメータ	内容
GOAL_EIP_T *pHdlEip	GOAL EtherNet/IP handle
GOAL_EIP_CB_ID_T id	コールバック ID
GOAL_EIP_CB_DATA_T *pCb	コールバックパラメータ

コールバックパラメータは、最大 10 個の要素を持つ共用体の配列です。

```
/*****************************************************************************/
/** EtherNet/IP Callback Handler
 * This function collects all callbacks from the stack and decides if the
 * callback must be handled.
 */

GOAL_STATUS_T main_eipCallback(
  GOAL_EIP_T *pHdlEip,                                  /**< EtherNet IP handle */
  GOAL_EIP_CB_ID_T id,                                  /**< callback id */
  GOAL_EIP_CB_DATA_T *pCb                               /**< callback parameters */
 )
{

GOAL_STATUS_T res = GOAL_OK;                        /* result */
  switch (id) {
  case
   GOAL_EIP_CB_ID_INIT:
   /* initialize application resources */
   break;
   /* ...*/
  }

  return res;
}
```

以下の項で、使用可能なコールバック ID の詳細について説明します。

5.7.2 GOAL_EIP_CB_ID_INIT

スタックが初期化されます。アプリケーションは自身のリソースを初期化することができます。

パラメータ <none>

戻り値

- GOAL_OK：成功
- その他：失敗

5.7.3 GOAL_EIP_CB_ID_READY

初期化を行います。

パラメータ <none>

戻り値 <ignored>

5.7.4 GOAL_EIP_CB_ID_CONNECT_EVENT

接続イベントをアプリケーションに通知します。

パラメータ

表 5.89 GOAL_EIP_CB_ID_CONNECT_EVENT のコールバックパラメータ

要素	メンバ	データ型	内容
0	outputAssembly	uint32_t	出力アセンブリのインスタンス ID
1	inputAssembly	uint32_t	入力アセンブリのインスタンス ID
2	connectionEvent	uint32_t	GOAL_EIP_CONNECTION_EVENT_*

戻り値 <ignored>

5.7.5 GOAL_EIP_CB_ID_ASSEMBLY_DATA_RECV

アセンブリの新規データを受信します。

パラメータ

表 5.90 GOAL_EIP_CB_ID_ASSEMBLY_DATA_RECV のコールバックパラメータ

要素	メンバ	データ型	内容
0	instanceNr	uint32_t	アセンブリのインスタンス ID

戻り値

- GOAL_OK：成功
- その他：失敗

5.7.6 GOAL_EIP_CB_ID_ASSEMBLY_DATA_SEND

アセンブリのデータを送信することをアプリケーションに通知します。

パラメータ

表 5.91 GOAL_EIP_CB_ID_ASSEMBLY_DATA_SEND のコールバックパラメータ

要素	メンバ	データ型	内容
0	instanceNr	uint32_t	アセンブリのインスタンス ID

戻り値

- GOAL_OK：データ変更
- その他：データ未変更

5.7.7 GOAL_EIP_CB_ID_RUN_IDLE_CHANGED

消費されるサイクリックデータの Run/Idle ヘッダが変更されたことをアプリケーションに通知します。

パラメータ

表 5.92 GOAL_EIP_CB_ID_RUN_IDLE_CHANGED のコールバックパラメータ

要素	メンバ	データ型	内容
0	outputAssembly	uint32_t	出力アセンブリのインスタンス ID
1	inputAssembly	uint32_t	入力アセンブリのインスタンス ID
2	runIdleValue	uint32_t	Run/Idle フラグの現在値

戻り値 <ignored>

5.7.8 GOAL_EIP_CB_ID_LED_CHANGED

モジュールステータスまたはネットワークステータス LED を変更する必要があることをアプリケーションに通知します。パラメータには GOAL_EIP_LED_*マクロで構成されるビットマップが含まれています。ビットが設定されると、対応する LED が設定されます。それ以外の場合はクリアされます。

パラメータ

表 5.93 GOAL_EIP_CB_ID_LED_CHANGED のコールバックパラメータ

要素	メンバ	データ型	内容
0	leds	uint32_t	アクティブ LED のビットマップ

戻り値 <ignored>

5.7.9 GOAL_EIP_CB_ID_DEVICE_RESET

デバイスをリセットすることをアプリケーションに通知します。プラットフォームによって、デバイスをリセットするか、またはリセットのシミュレーションのみ行われます。そのため、アプリケーションは自身のリソースを再初期化する必要があります。コールバックパラメータはリセットタイプを示します。

パラメータ

表 5.94 GOAL_EIP_CB_ID_DEVICE_RESET のコールバックパラメータ

要素	メンバ	データ型	内容
0	resetState	uint32_t	GOAL_EIP_RESET_*

戻り値 <ignored>

5.7.10 GOAL_EIP_CB_ID_REVISION_CHECK

Electronic Key Segment に互換性ビットが設定されている場合、アプリケーションは Minor Revision がサポートされているかどうかを判断できます。これは、アプリケーションが以前のリビジョンと互換性があるかどうかを判断出来る事を意味します。

パラメータ

表 5.95 GOAL_EIP_CB_ID_REVISION_CHECK のコールバックパラメータ

要素	メンバ	データ型	内容
0	minorRevision	uint8_t	要求したアプリケーションのマイナーバージョン

戻り値

- GOAL_OK：リビジョンをサポート
- その他：リビジョン非サポート

5.7.11 GOAL_EIP_CB_ID_ACD_CONFLICT

検出されたアドレスの競合についてアプリケーションに通知します。アプリケーションが GOAL_OK 又は GOAL_ERROR を返すかどうかに応じて、デバイスは構成された動作との競合を解決しようとするかエラーフォルトに移行します。構成には、関数 goal_eipCfgAcdconflictFallbackIp()が使用されます。

パラメータ

表 5.96 GOAL_EIP_CB_ID_ACD_CONFLICT のコールバックパラメータ

要素	メンバ	データ型	内容
0	acdActivity	uint8_t	ACD ステータスは次の通り: 0 – 競合なし 1 – 検出継続 2 – IP アドレス調査 3 – セミアクティブプロービング
1	acdRemoteMac	uint8_t *	アドレス競合を起こしたデバイスの MAC の配列
2	acdArpPru	uint8_t *	アドレス競合を起こした raw フレームの配列
3	flgDhcpEnabled	GOAL_BOOL_T	DHCP 有効フラグ

戻り値

- GOAL_OK：デバイスは構成された動作を追加します
- GOAL_ERROR：デバイスはエラーフォルトに移行します

5.8 通信スタック — EtherCAT

本節では、GOAL EtherCAT が提供する API 関数について記載します。

5.8.1 CoE API

CoE モジュールは、アプリケーションをオブジェクトおよびプロセスデータに接続するための表示イベントを提供します。 CoE および SDO モジュールをトリガする API 関数もあります。

パラメータ

表 5.97 CoE Indication events

コールバック ID	内容
GOAL_ECAT_CB_ID_SDO_UPLOAD	indicate SDO Read access to an object
GOAL_ECAT_CB_ID_SDO_DOWNLOAD	check an object's new data before it is written
GOAL_ECAT_CB_ID_RxPDO_RECEIVED	indicate reception of output process data (objects updated)
GOAL_ECAT_CB_ID_TxPDO_PREPARE	update input process data objects before they are mapped to Sync Manager

5.8.2 EoE API

EoE API が内部で GOAL フレームワークに接続されることによって有効になっているため、EoE API パラメータは、ありません。

5.8.3 FoE API

FoE モジュールは、アプリケーションで実装する必要のあるインディケーションイベントを提供します。

パラメータ

表 5.98 FOE indication events

イベント	内容
GOAL_ECAT_CB_ID_FOE_READ_REQ	open a file for reading
GOAL_ECAT_CB_ID_FOE_WRITE_REQ	open a file for writing
GOAL_ECAT_CB_ID_FOE_READ_DATA	get fragment of read file
GOAL_ECAT_CB_ID_FOE_WRITE_DATA	write fragment of write file
GOAL_ECAT_CB_ID_FOE_ERROR	error occurred, close file

5.8.4 EtherCAT State machine

これらの関数は、EtherCAT ステートマシン(ESM)の変更についてアプリケーションに通知します。

パラメータ

表 5.99 ESM API

パラメータ	内容
goal_ecatEsmStateGet()	get current ESM state

表 5.100 ESM Events

イベント	内容
GOAL_ECAT_CB_ID_NEW_ESM_STATE_ENTERED	indicate new ESM state and possible error
GOAL_ECAT_CB_ID_NEW_ESM_STATE_REQUESTED	indicate an ESM state change request
GOAL_ECAT_CB_ID_EXPLICIT_DEV_ID	get the Explicit Device ID (during state change)

表 5.101 ESM states

Symbol	内容
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_BOOTSTRAP	ESM state BOOTSTRAP
GOAL_ECAT_ESM_STATE_SAFEOP	ESM state SafeOP
GOAL_ECAT_ESM_STATE_OP	ESM state OP

5.8.5 Data Layer indication functions

これらのイベントは、Data Link レイヤーイベントが発生した場合に EtherCAT ライブラリによって呼び出されます。

パラメータ

表 5.102 Data Link Events

パラメータ	内容
GOAL_ECAT_CB_ID_SM_WATCHDOG_EXPIRED	process data reception timeout (Hardware Watchdog)
GOAL_ECAT_CB_ID_NEW_DL_STATE	port state change

5.8.6 Distributed Clock API

ディストリビュートクロックを介した時刻同期がアクティブ化されている場合、次のイベントがライブラリによって呼び出されます。

パラメータ

表 5.103 DC Events

パラメータ	内容
GOAL_ECAT_CB_ID_NEW_DC_CONFIG	check the synchronization settings, e.g. cycle time
GOAL_ECAT_CB_ID_DC_FAIL	indicate loss of Sync0 interrupts

5.9 アプリケーションコールバック — EtherCAT

5.9.1 概要

EtherCAT を最大限に活用するために、スタックはユーザが複数段階のプロトコルでやり取りを行うことを可能にします。

スタックは、goal_ecatNew()関数を通じて登録されたコールバックハンドラを呼び出します。コールバックハンドラは GOAL_STATUS_T 値を返します。

表 5.104 コールバックハンドラのパラメータ

パラメータ	内容
GOAL_ECAT_T *pHdlEcat	GOAL EtherAT handle
GOAL_ECAT_CB_ID_T id	コールバック ID
GOAL_ECAT_CB_DATA_T *pCb	コールバックパラメータ

コールバックパラメータは、最大 6 個の要素を持つ共用体の配列です。

```
/*****************************************************************************/
/** EtherCAT Callback Handler
 * This function collects all callbacks from the stack and decides if the
 * callback must be handled.
 */

GOAL_STATUS_T appl_ecatCallback(
  GOAL_ECAT_T *pHdlEcat,                                    /**< GOAL EtherCAT handle */
  GOAL_ECAT_CB_ID_T id,                                     /**< callback id */
  GOAL_ECAT_CB_DATA_T *pCb                                  /**< callback parameters */
 )
{

  GOAL_STATUS_T res = GOAL_OK;                          /* result */
  switch (id) {
  case
    GOAL_ECAT_CB_ID_.....:
    break;
  }

  return res;
}
```

パラメータは、データ構造 GOAL_ECAT_CB_DATA_T の配列としてコールバック関数に伝播されます。

コールバックデータのコンテンツ構造

```
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;
```

コールバック ID に応じて、パラメータと戻り値がコールバックデータ配列内の ID 固有の位置で配置されて、提供されます。

以下の項で、使用可能なコールバック ID の詳細について説明します。

5.9.2 GOAL_ECAT_CB_ID_SDO_DOWNLOAD

SDO 書き込みアクセスをアプリケーションに通知します。

5.9.3 GOAL_ECAT_CB_ID_SDO_UPLOAD

SDO 読み出しアクセスをアプリケーションに通知します。

5.9.4 GOAL_ECAT_CB_ID_RxPDO_RECEIVED

出力データを取得します。

5.9.5 GOAL_ECAT_CB_ID_TxPDO_PREPARE

新しい入力データを提供します。

5.9.6 GOAL_ECAT_CB_ID_SYNC_FAIL

同期操作失敗をアプリケーションに通知します。

5.9.7 GOAL_ECAT_CB_ID_NEW_DL_STATE

Data Link レイヤーのステータス変化をアプリケーションに通知します。

5.9.8 GOAL_ECAT_CB_ID_NEW_DC_CONFIG

DC 構成の変化をアプリケーションに通知します。

5.9.9 GOAL_ECAT_CB_ID_DC_FAIL

DC 同期失敗をアプリケーションに通知します。

5.9.10 GOAL_ECAT_CB_ID_SM_WATCHDOG_EXPIRED

プロセスデータ受信タイムアウト(HW ウォッチドッグ)をアプリケーションに通知します。

5.9.11 GOAL_ECAT_CB_ID_EXPLICIT_DEV_ID

Explicit Device Identification をアプリケーションに通知します。

5.9.12 GOAL_ECAT_CB_ID_NEW_ESM_STATE_ENTERED

新しい ESM が入力されたことをアプリケーションに通知します。

5.9.13 GOAL_ECAT_CB_ID_NEW_ESM_STATE_REQUESTED

新しい ESM が要求されたことをアプリケーションに通知します。

5.9.14 GOAL_ECAT_CB_ID_FOE_READ_REQ

FoE 読み出し要求をアプリケーションに通知します。

5.9.15 GOAL_ECAT_CB_ID_FOE_READ_DATA

FoE データ読み出しをアプリケーションに通知します。

5.9.16 GOAL_ECAT_CB_ID_FOE_WRITE_REQ

FoE 書き込み要求をアプリケーションに通知します。

5.9.17 GOAL_ECAT_CB_ID_FOE_WRITE_DATA

FoE データ書き込みをアプリケーションに通知します。

5.9.18 GOAL_ECAT_CB_ID_FOE_ERROR

FoE エラーをアプリケーションに通知します。

5.9.19 GOAL_ECAT_CB_ID_RUN_LED_STATE

RUN 状態の LED の変化をアプリケーションに通知します。

5.9.20 GOAL_ECAT_CB_ID_ERROR_LED_STATE

ERROR 状態の LED の変化をアプリケーションに通知します。

6. アプリケーションプログラミングインタフェース

本章では、R-IN32M3 Module が提供する API 関数について記載します。

6.1 デバイス固有関数

6.1.1 appl_ccmRpcInit

目的：GOAL (appl_init)への R-IN32M Module API 登録

R-IN32M3 Module 固有の API を GOAL に登録します。呼び出しは appl_init 内で行われる必要があります。

本関数は GOAL_STATUS_T ステータスを返します。パラメータはありません。

関数プロトタイプ）

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

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：R-IN32M3 Module のファームウェア更新有効化

R-IN32M3 Module のファームウェア更新を有効にします。

本関数は GOAL_STATUS_T ステータスを返します。パラメータはありません。

関数プロトタイプ）

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

6.1.3 appl_ccmUpdateDeny

目的：R-IN32M3 Module のファームウェア更新無効化

R-IN32M3 Module のファームウェア更新を無効にします。本関数は、サイクリック通信中のファームウェア更新を無効にする場合に使用します。

本関数は GOAL_STATUS_T ステータスを返します。パラメータはありません。

関数プロトタイプ）

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

6.1.4 appl_ccmInfo

目的：バージョン&デバイス情報取得

R-IN32M3 Module から情報を読み出します。

- MAC アドレス（シリアル番号）
- ソフトウェアバージョン
- デバイスタイプ

関数プロトタイプ）

```
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 */
);
```

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：フォルト時のフィールドバス通信動作設定

サイクリック通信に関する R-IN32M3 Module の動作を決定します。デフォルトでは、アプリケーションコントローラ（AC）に対する通信が切断すると、サイクリック通信は停止します。

関数プロトタイプ）

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

例）

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

6.1.6 appl_ccmCommResetSet

目的：SPI 同期リセット要求に応じた R-IN32M3 Module の動作設定

実行中の AC で CC が予めセットアップされている場合、再起動時に同期リセットが AC によって要求されます。

本関数は、リセット要求に関する R-IN32M3 Module の動作を決定します。デフォルトでは、リセットは行われません。本オプションを有効に設定すると、リセットが行われます。

本設定は CC の不揮発性メモリに格納されます。“0”でリセットが無効になり、“1”で有効になります。

関数プロトタイプ）

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

例）

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

6.1.7 appl_ccmLogEnable

目的：CC への AC ログメッセージ送信許可

本関数が実行されると、AC のログバッファからのログメッセージが CC に連続して転送されます。これらのメッセージは、CC 自身のログメッセージとして、管理インタフェースを介してアクセス可能になります。

関数プロトタイプ）

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

例）

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

6.1.8　appl_ccmLogToAcEnable

目的：AC への CC ログメッセージの転送を有効にする

本関数が実行されると、CC のログバッファからのログメッセージは継続的に AC に転送されます。 これらは、シリアルコンソール、または端末などのローカルログメカニズムを介して AC 独自のログメッセージとしてアクセスできます。

関数プロトタイプ）

```
GOAL_STATUS_T appl_ccmLogToAcEnable(

  void
);
```

例）

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

6.1.9　appl_ccmFwUpdateStart

目的：rpc を使用して CC のファームウェアアップデートを開始する

本関数は、ファームウェア更新データを含む指定されたデータバッファを CC に転送します。

関数プロトタイプ）

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

例）

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

6.1.10 appl_ccmFwUpdateExecute

目的：ファームウェアアップデートにて、アップデート処理を実行する

本関数は、CC の実際の更新を実行します。 ファームウェアデータの転送後に呼び出す必要があります。これには、appl_ccmFwUpdateCbReg()を使用したファームウェアアップデートイベントの登録が必要です。アプリケーションがイベントに登録しない場合、関数 appl_ccmFwUpdateExecute は CC によって暗黙的に呼び出されます

関数プロトタイプ）

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

例）

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

6.1.11 appl_ccmEcatSsiUpdate

目的：eeprom で EtherCAT SSI データの更新を実行する

本関数は、オプションで EEPROM の EtherCAT SSI データを初期化できます。

一部の EtherCAT マスターは、上書きしてはならない eeprom の設定（構成済みステーションエイリアス）に依存しているため、これは 1 回だけ呼び出す必要があります。

関数プロトタイプ）

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

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：FoE ファームウェアアップデートの要件を構成する

本関数は、FoE ファームウェアアップデートに必要な設定を構成します。

関数プロトタイプ）

```
GOAL_STATUS_T appl_ccmFoeUpdateSettings(
      char *strName,                          /**< foe firmware filename */
      uint8_t matchLength,                    /**< string match length */
      uint32_t password,                      /**< foe password */
      GOAL_BOOL_T flgBootstrapReq             /**< bootstrap required */
);
```

例）

```
/* enable FoE */
res = goal_ecatCfgFoeOn(GOAL_TRUE);
if (GOAL_RES_ERR(res)) {
    goal_logErr("failed to enable FoE support");
}

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

/* set settings for ccm firmware update via FoE */
res = appl_ccmFoeUpdateSettings(
    "ccm.efw",                              /* filename beginning */
    0,                                      /* 0 -> match all characters */
    0,                                      /* password */
    GOAL_TRUE);                             /* only update in ESM state bootstrap */
if (GOAL_RES_ERR(res)) {
    goal_logErr("failed to configure FoE firmware update of CC");
    return res;
}
```

注意：FoE を介したファームウェアの更新を機能させるには、goal_ecatCfgFoeOn(GOAL_TRUE)を使用して FoE を有効にする必要があります。ファームウェアの更新がブートストラップ ESM 状態で行われるように構成されている場合、goal_ecatCfgBootstrapOn(GOAL_TRUE) を使用してこの状態のサポートを有効にする必要があります。 appl_ccmFoeUpdateSettings のパラメータは、ファームウェアの更新を実行するために必要なものを定義します。

表 6.1 appl_ccmFoeUpdateSettings パラメータ

パラメータ	型	内容
strName	Char *	ファームウェアアップデートに必要なファイル名
umatchLength	uint8_t	設定された敵ファイル名と更新プロセスの敵ファイル名を一致させるために必要な文字数。 matchLength がゼロの場合、strName のすべてのバイトが一致する必要があります。 strName が長さ 0 の文字列の場合、デフォルトのファイル名「ccm.efw」が必要です。
password	uint32_t	ファームウェアアップデートのための FoE パスワード
flgBootstrapReq	bool	有効の場合、ファームウェアの更新は esm 状態のブートストラップでのみ受け入れられます

6.1.13 appl_ccmEthMacAddressSet

目的：デバイスのカスタム MAC アドレスを構成する

本関数は、デバイスの MAC アドレスを変更できます。 ネットワーク関連の機能（通信スタックの開始、ネットワークの初期化）の前に呼び出す必要があります。

関数プロトタイプ）

```
GOAL_STATUS_T appl_ccmEthMacAddressSet(
uint8_t *pMacAddr
);
```

例）

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

6.1.14 appl_ccmNetworkDefaultUp

目的：デフォルトのネットワークを開始する

本関数は、CC を標準のイーサネットモードで起動します。 プロトコルスタックはネットワークを適切なモードで自動的に開始するため、通常、この関数を呼び出す必要はありません。

関数プロトタイプ）

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

例）

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

6.1.15 appl_ccmNetworkEoEUp

目的：EtherCAT EoE ネットワーキングを開始する

本関数は、CC を EtherCAT モードで起動します。 EtherCAT プロトコルスタックはネットワークを適切なモードで自動的に開始するため、通常、この関数を呼び出す必要はありません

関数プロトタイプ）

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

例）

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

6.1.16 appl_ccmCfgVarGet

目的：構成変数を取得する

本関数は、CC の構成変数を読み取るメカニズムを提供します。モジュール ID と変数 ID が必要です。これについては、4.9 章に記載されています。

関数プロトタイプ）

```
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 */
);
```

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：構成変数を設定する

本関数は、CC の構成変数を書き込むメカニズムを提供します。 モジュール ID と変数 ID が必要です。これについては、4.9 章に記載されています。

関数プロトタイプ）

```
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 */
);
```

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：構成変数の変更を永続的に保存する

本関数は、構成変数を含む不揮発性ストレージを更新するメカニズムを提供します。 構成変数の現在の値が保存されます。

関数プロトタイプ）

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

例）

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

6.2　デバイス検出

6.2.1　goal_ddInit ― GOAL (appl_init)への GOAL DD API 登録

目的：GOAL DD 固有の API を GOAL に登録します。

呼び出しは appl_init 内で行われる必要があります。

本関数は GOAL_STATUS_T ステータスを返します。パラメータはありません。

関数プロトタイプ）

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

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 ― GOAL (appl_setup)への GOAL DD API 登録

目的：GOAL DD のインスタンスを GOAL に作成します。

呼び出しは appl_setup 内で行われる必要があります。GOAL DD API の使用には有効なインスタンスが必要です。

本関数は GOAL_STATUS_T ステータスを返します。パラメータはありません。

表 6.2　goal_ddNew パラメータ

パラメータ	内容
GOAL_DD_T **ppHdl	返された GOAL DD ハンドル
uint32_t bitmaskFeatures	有効化する初期インスタンス機能

パラメータ bitmaskFeatures は、GOAL DD の単一機能を無効にするビットマスクです。

表 6.3　　機能ビットマスクパラメータ

ビット	機能
0	HELLO 要求（デバイススキャン検出に使用）の無効化
1	WINK コマンド無効化
2	GETLIST コマンド（使用可能な CM 変数リスト読み出し）の無効化
3	GETCONFIG コマンド（CM 変数値読み出し）の無効化
4	SETCONFIG コマンド（CM 変数値書き込み）の無効化
5	SETIP コマンド（GOAL DD 経由の IP 設定）の無効化

関数プロトタイプ）

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

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 ― GOAL DD インスタンスのユーザ ID 設定

目的：本関数は、任意の GOAL DD インスタンスのユーザ ID を設定します。

ユーザ ID は、GOAL DD を使用した各要求に含まれる、基本となるプロトコルのプロパティです。0 の値を持つ特別ユーザ ID があり、これによってすべての要求が処理されます。ユーザ ID が 0 以外の値の場合は、要求のユーザ ID が GOAL DD インスタンスのユーザ ID と同じ場合にのみ、要求が処理されます。

ユーザ ID は不揮発性メモリに格納されます。

表 6.4　　goal_ddCustomerIdSet パラメータ

パラメータ	内容
GOAL_DD_T *pHdl	GOAL DD ハンドル
uint32_t customerId	インスタンスユーザーID

関数プロトタイプ）

```
GOAL_STATUS_T goal_ddCustomerIdSet(
  GOAL_DD_T *pHdlDd,                    /**< dd handle */
  uint32_t cid                          /**< customer ID */
);
```

例）

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

6.2.4　goal_ddModuleNameSet ― GOAL DD インスタンスのモジュール名設定

目的：任意の GOAL DD インスタンスのモジュール名を設定します。

モジュール ID は、不揮発性メモリに格納されているデバイスのプロパティです。Management Tool によってデバイスの命名に使用されます。モジュール名の長さは 20 バイトに制限されています。

表 6.5　goal_ddModuleNameSet パラメータ

パラメータ	内容
GOAL_DD_T *pHdl	GOAL DD ハンドル
uint8_t *str	インスタンスモジュール名

関数プロトタイプ）

```
GOAL_STATUS_T goal_ddModuleNameSet(
  GOAL_DD_T *pHdlDd,                    /**< dd handle */
  uint8_t *str                          /**< module name */
);
```

例）

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

6.2.5　goal_ddFeaturesSet — GOAL DD インスタンスの機能設定

目的：任意の GOAL DD インスタンスに対して無効化する機能を設定します。

このプロパティは、不揮発性メモリに格納されているデバイスに格納されます。

表 6.6　goal_ddFeaturesSet パラメータ

パラメータ	内容
GOAL_DD_T *pHdl	GOAL DD ハンドル
uint32_t bitmaskFeatures	インスタンス機能ビットマスク

パラメータの内容については、関数「goal_ddNew」を参照してください。

関数プロトタイプ）

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

例）

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

6.2.6　goal_ddCallbackReg — GOAL DD インスタンスに対するコールバック設定

目的：任意の GOAL DD インスタンスに対するコールバックを登録します。

コールバックのタイプ）

```
typedef GOAL_STATUS_T (* GOAL_DD_FUNC_CB_T)(
  struct GOAL_DD_T *pHdlDd,         /**< dd handle */
  GOAL_DD_CB_ID_T cbId,             /**< callback id */
  GOAL_DD_CB_DATA_T *pCbData        /**< callback data */
);
```

関数プロトタイプ）

```
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 */
);
```

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 インスタンス機能の一時有効化

目的：任意の GOAL DD インスタンスの機能を一時的に有効にします。

このプロパティは不揮発性メモリには格納されません。

表 6.7　goal_ddSessionFeatureActivation パラメータ

パラメータ	内容
GOAL_DD_T *pHdl	GOAL DD ハンドル
uint32_t bitmaskFeatures	インスタンス機能ビットマスク

注意：パラメータ"bitmaskFeatures"は、恒久的な機能設定を行う関数に戻すために使用され、ここで設定されるビットは任意の機能を有効化します。

関数プロトタイプ）

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

例）

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

6.2.8　goal_ddFilterAdd ― CM 変数へのアクセス制限

目的：Management Tool としての外部アプリケーションは、デフォルトでデバイスのすべての CM 変数にアクセスできます。

これは開発には便利ですが、生産目的で、Management Tool を使用する最低限の機能に必要な変数のみにアクセスを制限したい場合があります。そのため、アクセス制限を行うフィルタを導入しています。定義済みのフィルタを以下に示します。

表 6.8　goal_ddFilterAdd フィルタセット

フィルタ ID	フィルタ名
0	GOAL_DD_ACCESS_FILTER_SET_ALL
1	GOAL_DD_ACCESS_FILTER_SET_BASIC
2	GOAL_DD_ACCESS_FILTER_SET_HIDDEN

表 6.9　goal_ddFilterAdd 定義済みフィルタ

フィルタ ID	フィルタ動作	目的
0	全変数へのフルアクセス	開発
1	NET モジュールの全変数（IP 設定）へのリードアクセス、ETH モジュールの全変数（MAC、ステータス）へのリードアクセス、LM モジュールの全変数（ログ）へのフルアクセス	Management Tool の最低限のサポートが付いたプロダクションコード
2	Web サーバの認証文字列へのリードアクセス無効	プロダクションコード

6.2.9 フィルタの定義 — GOAL_DD_ACCESS_FILTER_SET_ALL

```
/**< complete access */
static GOAL_DD_VAR_T ddAccessAll[] = {
  {
     .pNext = NULL,
     .modId = GOAL_DD_VAR_ALL,
     .varId = GOAL_DD_VAR_ALL,
     .access = (1 << GOAL_DD_ACCESS_READ) | (1 << GOAL_DD_ACCESS_WRITE)
  }
};
```

6.2.10 フィルタの定義 — 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　フィルタの定義 ― 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)
  },
};
```

関数プロトタイプ）

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

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 スタックのアプリケーションプログラミングインタフェース

6.3.1 goal_pnioInit — GOAL への GOAL PROFINET 登録 (appl_init)

GOAL PROFINET を GOAL に登録します。呼び出しは appl_init 内で行われる必要があります。主な機能として、NVS ストレージが初期化される前に設定コンフィグマネージャ変数を登録します。

本関数は GOAL_STATUS_T ステータスを返します。パラメータはありません。

```
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 — GOAL PROFINET インスタンス作成 (appl_setup)

すべての定義済みの変数(goal_pnioCfg API)のスナップショットを取り、必要なリソースを割り当てることで、新たな GOAL PROFINET インスタンスを生成します。アプリケーションは、GOAL PROFINET インスタンス ID とコールバックハンドラを提供する必要があります。コールバックハンドラは、GOAL PROFINET スタックのデフォルト動作のみを使用するために NULL である場合もあります。

本関数は GOAL_STATUS_T ステータスと GOAL PROFINET ハンドルを返します。

表 6.10 goal_pnioNew パラメータ

パラメータ	内容
GOAL_PNIO_T **ppPnio	返された GOAL PROFINET ハンドル
const uint32_t id	GOAL PROFINET インスタンス ID
GOAL_PNIO_FUNC_CB_T pFunc	GOAL PROFINET コールバック関数

6.3.3 goal_pnioCfgDcpFactoryResetDisableSet — DCP ファクトリリセット設定

DCP ファクトリリセットを使用可能にするかどうかを制御します。GOAL_TRUE に設定されると、DCP ファクトリリセットは拒否されます。

デフォルト：GOAL_FALSE

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

注意：整合性が失われる場合があるため、本関数は使用しないでください。

表 6.11 goal_pnioCfgDcpFactoryResetDisableSet パラメータ

パラメータ	内容
GOAL_BOOL_T flgDcpFactoryResetDisable	DCP ファクトリリセット無効フラグ

6.3.4 goal_pnioCfgDcpAcceptMixcaseStationSet — DCP 大小文字混在ステーション名承認

大小文字が混在するステーション名を DCP が承認するかどうかを制御します。GOAL_TRUE に設定されると、DCP は大小文字が混在するステーション名を承認します。

デフォルト：GOAL_FALSE

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

注意：整合性が失われる場合があるため、本関数は使用しないでください。

表 6.12 goal_pnioCfgDcpAcceptMixcaseStationSet パラメータ

パラメータ	内容
GOAL_BOOL_T flgDcpAcceptMixcaseStation	DCP 大小文字混在ステーション名承認フラグ

6.3.5　goal_pnioCfgDevDapSimpleSet ― デバイス DAP 簡易モード設定

デバイス DAP の簡易モードを設定します。flgDevDapSimple が GOAL_TRUE の場合、GOAL PROFINET スタックは DAP、ポートスロット、モジュールを自動で作成します。

デフォルト：GOAL_TRUE

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

注意：整合性が失われる場合があるため、本関数は使用しないでください。

表 6.13　goal_pnioCfgDevDapSimpleSet パラメータ

パラメータ	内容
GOAL_BOOL_T flgDevDapSimple	デバイス DAP 簡易モードフラグ

6.3.6　goal_pnioCfgDevDapApiSet ― デバイス DAP API 番号設定

デバイス DAP の API 番号を設定します。

デフォルト：0

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

注意：整合性が失われる場合があるため、本関数の使用時には注意が必要です。

表 6.14　goal_pnioCfgDevDapApiSet パラメータ

パラメータ	内容
uint32_t idDevDapApi	デバイス DAP API 番号

6.3.7　goal_pnioCfgDevDapSlotSet ― デバイス DAP スロット番号設定

デバイス DAP のスロット番号を設定します。

デフォルト：0

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

注意：整合性が失われる場合があるため、本関数の使用時には注意が必要です。

表 6.15　goal_pnioCfgDevDapSlotSet パラメータ

パラメータ	内容
uint16_t idDevDapSlot	デバイス DAP スロット番号

6.3.8 goal_pnioCfgDevDapSubslotSet — デバイス DAP サブスロット番号設定

デバイス DAP のサブスロット番号を設定します。

デフォルト：1

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

注意：整合性が失われる場合があるため、本関数の使用時には注意が必要です。

表 6.16 goal_pnioCfgDevDapSubslotSet パラメータ

パラメータ	内容
uint16_t idDevDapSubslot	デバイス DAP サブスロット番号

6.3.9 goal_pnioCfgDevDapModuleSet — デバイス DAP モジュール ID 設定

デバイス DAP のモジュール ID を設定します。

デフォルト：1

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

注意：整合性が失われる場合があるため、本関数の使用時には注意が必要です。

表 6.17 goal_pnioCfgDevDapModuleSet パラメータ

パラメータ	内容
uint32_t idDevDapMod	デバイス DAP モジュール ID

6.3.10 goal_pnioCfgDevDapSubmoduleSet — デバイス DAP サブモジュール ID 設定

デバイス DAP のサブモジュール ID を設定します。

デフォルト：1

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

注意：整合性が失われる場合があるため、本関数の使用時には注意が必要です。

表 6.18 goal_pnioCfgDevDapSubmoduleSet パラメータ

パラメータ	内容
uint32_t idDevDapSubmod	デバイス DAP サブモジュール ID

6.3.11　goal_pnioCfgNetLinkSafetySet ― デバイスポート無効動作設定

デバイスポート無効動作を設定します。flgNetLinkSafety が GOAL_TRUE に設定されると、GOAL PROFINET スタックは、マスターが全 Ethernet インタフェースを同時に無効化することを禁止します。

デフォルト：GOAL_TRUE

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

注意：整合性が失われる場合があるため、本関数の使用時には注意が必要です。

表 6.19　goal_pnioCfgNetLinkSafetySet パラメータ

パラメータ	内容
GOAL_BOOL_T flgNetLinkSafety	デバイスポート無効動作

6.3.12　goal_pnioCfgNewIoDataCbSet ― 新規 I/O データコールバック設定

新規 I/O データコールバックを設定します。flgCbNewIoData が GOAL_TRUE に設定されると、登録コールバック関数が到着した各サイクリックフレームに対して呼び出されます。これはリアルタイムコンテキストで起こるため、コールバック関数は値をできるだけ早く返す必要があります。

デフォルト：GOAL_FALSE

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

注意：有効に設定されると、サイクルタイムが 1ms に設定されている場合にコールバックの呼び出しが 1 秒に 1000 回行われるため、非常に高いアプリケーション負荷がかかります。

表 6.20　goal_pnioCfgNewIoDataCbSet パラメータ

パラメータ	内容
GOAL_BOOL_T flgCbNewIoData	新規 I/O データコールバックフラグ

6.3.13 goal_pnioCfgDiagBufMaxCntSet — 最大 Diagnostics エントリ設定

Diagnostics エントリの最大数を設定します。

デフォルト：20

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

注意：値が小さすぎると、整合性が失われることがあります。

表 6.21 goal_pnioCfgDiagBufMaxCntSet パラメータ

パラメータ	内容
unsigned int numDiagBufMax	最大 Diagnostics エントリ

6.3.14 goal_pnioCfgDiagBufMaxDataSizeSet — 最大 Diagnostics データサイズ設定

各 Diagnostics エントリのバイトの最大サイズを設定します。

デフォルト：28

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

注意：値が 28 より小さいと、整合性が失われることがあります。

表 6.22 goal_pnioCfgDiagBufMaxDataSizeSet パラメータ

パラメータ	内容
unsigned int numDiagDataSizeMax	最大 Diagnostics データサイズ

6.3.15 goal_pnioCfgIocrBlocksMaxSet — 最大 IOCR ブロックバッファ設定

IOCR ブロックバッファの最大数を設定します。

デフォルト：10

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

注意：整合性が失われる場合があるため、本関数の使用時には注意が必要です。

表 6.23 goal_pnioCfgIocrBlocksMaxSet パラメータ

パラメータ	内容
unsigned int numIocrBlocksMax	最大 IOCR ブロックバッファ

6.3.16　goal_pnioCfgCrMaxCntSet ― 最大 Communication Relations 数設定

Communication Relations（CR）の最大数を設定します。

デフォルト：10

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

注意：整合性が失われる場合があるため、本関数の使用時には注意が必要です。

表 6.24　goal_pnioCfgCrMaxCntSet パラメータ

パラメータ	内容
unsigned int numCrMax	最大 CR 数

6.3.17　goal_pnioCfgArMaxCntSet ― 最大 Application Relations 数設定

Application Relations（AR）の最大数を設定します。

デフォルト：2

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

注意：整合性が失われる場合があるため、本関数の使用時には注意が必要です。

表 6.25　goal_pnioCfgArMaxCntSet パラメータ

パラメータ	内容
unsigned int numArMax	最大 AR 数

6.3.18　goal_pnioCfgApiMaxCntSet ― 最大 API 数設定

アプリケーションプロセス識別子の最大数を設定します。この値の設定は ModuleDiffBlock ロジックのみに影響します。値は、最大限の GSDML 設定を保持するのに十分大きい値である必要があります。

デフォルト：1

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

注意：整合性が失われる場合があるため、また GOAL PROFINET スタックで API 0 以外はサポートしていないため、本関数は使用しないでください。

表 6.26　goal_pnioCfgApiMaxCntSet パラメータ

パラメータ	内容
unsigned int numApiMax	最大 API 数

6.3.19　goal_pnioCfgSlotMaxCntSet ― 最大 Slot 数設定

Slot の最大数を設定します。この値の設定は ModuleDiffBlock ロジックのみに影響します。値は、最大限の GSDML 設定を保持するのに十分大きい値である必要があります。

デフォルト：12

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

表 6.27　goal_pnioCfgSlotMaxCntSet パラメータ

パラメータ	内容
unsigned int numSlotMax	最大 Slot 数

6.3.20　goal_pnioCfgSubslotMaxCntSet ― 最大 sub slot 数設定

Slot ごとの sub slot の最大数を設定します。この値の設定は ModuleDiffBlock ロジックのみに影響します。値は、最大限の GSDML 設定を保持するのに十分大きい値である必要があります。

デフォルト：4

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

表 6.28　goal_pnioCfgSubslotMaxCntSet パラメータ

パラメータ	内容
unsigned int numSubslotMax	Slot ごとの最大 sub slot 数

6.3.21　goal_pnioCfgSubslotIfSet ― インタフェース sub slot 設定

インタフェースの sub slot ID を設定します。

デフォルト：0x8000

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

注意：整合性が失われる場合があるため、本関数の使用時には注意が必要です。

表 6.29　goal_pnioCfgSubslotIfSet パラメータ

パラメータ	内容
unsigned int idSubslotIf	インタフェース sub slot ID

6.3.22　goal_pnioCfgSubslotPortSet ― ポート sub slot 設定

ポートの sub slot ID を設定します。

デフォルト：0x8001

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

注意：整合性が失われる場合があるため、本関数は使用しないでください。

表 6.30　goal_pnioCfgSubslotPortSet パラメータ

パラメータ	内容
unsigned int idSubslotPort	ポート sub slot ID

6.3.23　goal_pnioCfgSnmpIdSet ― SNMP インスタンス ID 設定

GOAL PROFINET スタックが使用する GOAL SNMP インスタンス ID を設定します。

デフォルト：GOAL_ID_INVALID

GOAL_STATUS_T ステータスを返します。

本関数は、GOAL PROFINET インスタンスを設定します。goal_pnioNew が有効になる前に呼び出す必要があります。

表 6.31　goal_pnioCfgSnmpIdSet パラメータ

パラメータ	内容
idSnmp	GOAL SNMP インスタンス ID

6.3.24　goal_pnioSlotNew ― 新規 Slot 作成

新規 Slot を作成します。

GOAL_STATUS_T ステータスを返します。

表 6.32　goal_pnioSlotNew パラメータ

パラメータ	内容
GOAL_PNIO_T *pPnio	PROFINET データ
uint32_t idApi	API 番号（常に 0）
uint32_t idSlot	Slot 番号
GOAL_BOOL_T flgAutoGen	API を存在しない場合に自動で生成

```
/* 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 ― 新規 sub slot 作成

新規 sub slot を作成します。

GOAL_STATUS_T ステータスを返します。

表 6.33　goal_pnioSubslotNew パラメータ

パラメータ	内容
GOAL_PNIO_T *pPnio	PROFINET データ
uint32_t idApi	API 番号（常に 0）
uint32_t idSlot	Slot 番号
uint32_t idSubslot	sub slot 番号
GOAL_BOOL_T flgAutoGen	API とスロットを存在しない場合に自動で生成

```
/* 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 ― 新規 Module 作成

新規 Module を作成します。

GOAL_STATUS_T ステータスを返します。

表 6.34　goal_pnioModNew パラメータ

パラメータ	内容
GOAL_PNIO_T *pPnio	PROFINET データ
uint32_t idMod	モジュール ID

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

6.3.27　goal_pnioSubmodNew ― 新規 sub module 作成

新規 sub module を作成します。

GOAL_STATUS_T ステータスを返します。

表 6.35　goal_pnioSubmodNew パラメータ

パラメータ	内容
GOAL_PNIO_T *pPnio	PROFINET データ
uint32_t idMod	モジュール ID
uint32_t idSubmod	サブモジュール ID
GOAL_PNIO_MOD_TYPE_T type	サブモジュールタイプ（入力、出力、入出力）
uint16_t sizeInput	入力モジュールのサイズ
uint16_t sizeOutput	出力モジュールのサイズ
GOAL_BOOL_T flgAutogen	モジュールを存在しない場合に生成

```
/* 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 ― Slot へのモジュール差し込み

モジュールを Slot に差し込みます。

GOAL_STATUS_T ステータスを返します。

表 6.36　goal_pnioModPlug パラメータ

パラメータ	内容
GOAL_PNIO_T *pPnio	PROFINET データ
uint32_t idApi	API（常に 0）
uint16_t idSlot	Slot 番号
uint32_t idMod	モジュール ID

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

6.3.29　goal_pnioSubmodPlug ― Cyclic 通信 sub slot へのサブモジュール差し込み

サブモジュールを cyclic 通信用 sub slot に差し込みます。

GOAL_STATUS_T ステータスを返します。

表 6.37　goal_pnioSubmodPlug パラメータ

パラメータ	内容
GOAL_PNIO_T *pPnio	PROFINET データ
uint32_t idApi	API（常に 0）
uint16_t idSlot	Slot 番号
uint16_t idSubslot	sub slot 番号
uint32_t idMod	モジュール ID
uint32_t idSubmod	サブモジュール ID

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

6.3.30　goal_pnioRpcSubmodPlug ― RPC 通信 sub slot へのサブモジュール差し込み

サブモジュールを RPC 通信用 sub slot に差し込みます。

GOAL_STATUS_T ステータスを返します。

表 6.38　goal_pnioSubmodPlug パラメータ

パラメータ	内容
GOAL_PNIO_T *pPnio	PROFINET データ
uint32_t idApi	API（常に 0）
uint16_t idSlot	Slot 番号
uint16_t idSubslot	sub slot 番号
uint32_t idMod	モジュール ID
uint32_t idSubmod	サブモジュール ID

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

6.3.31 goal_pnioModPull — Slot からのモジュール取り外し

Slot からモジュールを取り外します。

GOAL_STATUS_T ステータスを返します。

表 6.39 goal_pnioModPull パラメータ

パラメータ	内容
GOAL_PNIO_T *pPnio	PROFINET データ
uint32_t idApi	API（常に 0）
uint16_t idSlot	Slot 番号

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

6.3.32 goal_pnioSubmodPull — sub slot からのサブモジュール取り外し

sub slot からサブモジュールを取り外します。

GOAL_STATUS_T ステータスを返します。

表 6.40 goal_pnioSubmodPull parameters

パラメータ	内容
GOAL_PNIO_T *pPnio	PROFINET データ
uint32_t idApi	API（常に 0）
uint16_t idSlot	Slot 番号
uint16_t idSubslot	sub slot 番号

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

6.3.33　goal_pnioDataOutputGet ― サブモジュールからの出力データ取得

サブモジュールから出力データを取得します。

GOAL_STATUS_T ステータスを返します。

表 6.41　goal_pnioDataOutputGet パラメータ

パラメータ	内容
GOAL_PNIO_T *pPnio	PROFINET データ
uint32_t idApi	API（常に 0）
uint16_t idSlot	Slot 番号
uint16_t idSubslot	sub slot 番号
char *pBuf	データバッファ
uint16_t len	データバッファ長
GOAL_PNIO_IOXS_T *pStateIops	生産者ステータスへのポインタ

```
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 ― サブモジュールへの入力データ設定

サブモジュールへの入力データを設定します。

GOAL_STATUS_T ステータスを返します。

表 6.42　goal_pnioDataInputSet パラメータ

パラメータ	内容
GOAL_PNIO_T *pPnio	PROFINET データ
uint32_t idApi	API（常に 0）
uint16_t idSlot	Slot 番号
uint16_t idSubslot	sub slot 番号
char *pBuf	データバッファ
uint16_t len	データバッファ長
GOAL_PNIO_IOXS_T statelops	生産者ステータス

```
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 — データユニットステータス取得

アプリケーションプロトコルのデータユニットステータスを取得します。

GOAL_STATUS_T ステータスを返します。

表 6.43 goal_pnioApduStatusGet パラメータ

パラメータ	内容
GOAL_PNIO_T *pPnio	PROFINET データ
GOAL_PNIO_AR_ID_T idAr	Application Relations ID
GOAL_PNIO_APDU_STATUS_T *pStatusApdu	APDU ステータスポインタ

6.3.36 goal_pnioAlarmNotifySend — アラーム通知送信

アラーム通知を送信します。

GOAL_STATUS_T ステータスを返します。

表 6.44 goal_pnioAlarmNotifySend パラメータ

パラメータ	内容
GOAL_PNIO_T *pPnio	PROFINET データ
uint16_t *pNrAlarmSeq	アラームシーケンス番号格納用ポインタ
GOAL_PNIO_AR_ID_T idAr	Application Relations ID
uint32_t prio	優先度
const GOAL_PNIO_ALARM_ NOTIFY_T*pAlarmNotify	アラーム通知へのポインタ
uint16_t lenDataUser	ユーザデータ長
void *pDataUser	ユーザデータへのポインタ

6.3.37 goal_pnioAlarmNotifySendAck — アラーム通知 ACK 送信

アラーム通知の ACK を送信します。

GOAL_STATUS_T ステータスを返します。

表 6.45 goal_pnioAlarmNotifySendAck パラメータ

パラメータ	内容
GOAL_PNIO_T *pPnio	PROFINET データ
GOAL_PNIO_AR_ID_T idAr	Application Relations ID
uint32_t prio	優先度
const GOAL_PNIO_ALARM_ NOTIFY_T*pAlarmNotify	アラーム通知へのポインタ
GOAL_PNIO_STATUS_T *pStatus	PROFINET ステータス
uint16_t lenDataUser	ユーザデータ長
void *pDataUser	ユーザデータへのポインタ

6.3.38 goal_pnioAlarmProcessSend — プロセスアラーム送信

プロセスアラームを送信します。

GOAL_STATUS_T ステータスを返します。

表 6.46 goal_pnioAlarmProcessSend パラメータ

パラメータ	内容
GOAL_PNIO_T *pPnio	PROFINET データ
uint32_t api	API
uint16_t slot	Slot 番号
uint16_t subslot	sub slot 番号
uint16_t usi	ユーザ構造識別子
uint16_t lenData	データ長
uint8_t *pData	ユーザデータへのポインタ

6.3.39 goal_pnioRecReadFinish — 記録読み出し要求への応答

記録読み出し要求への応答を行います。要求の有効期限切れを検出するために、シーケンス番号が使用されます。検出された場合、応答は自動的に中断します。

本関数は GOAL_STATUS_T ステータスを返します。

表 6.47 goal_pnioRecReadFinish パラメータ

パラメータ	内容
GOAL_PNIO_T *pPnio	PROFINET データ
int32_t hdl	記録処理
GOAL_PNIO_STATUS_T *pStatusPnio	PROFINET ステータスポインタ
const uint8_t *pData	記録データ
uint16_t len	記録データ長
uint32_t numSeq	シーケンス番号

6.3.40 goal_pnioRecWriteFinish — 記録書き込み要求への応答

記録書き込み要求への応答を行います。要求の有効期限切れを検出するために、シーケンス番号が使用されます。検出された場合、応答は自動的に中断します。

本関数は GOAL_STATUS_T ステータスを返します。

表 6.48 goal_pnioRecWriteFinish パラメータ

パラメータ	内容
GOAL_PNIO_T *pPnio	PROFINET データ
int32_t hdl	記録処理
GOAL_PNIO_STATUS_T *pStatusPnio	PROFINET ステータスポインタ
uint32_t numSeq	シーケンス番号

6.3.41　goal_pnioDiagExtChanDiagAdd ― 拡張チャネル Diagnostics エントリ追加

拡張チャネルの Diagnostics エントリを追加します。

GOAL_STATUS_T ステータスを返します。

表 6.49　goal_pnioDiagExtChanDiagAdd パラメータ

パラメータ	内容
GOAL_PNIO_T *pPnio	PROFINET データ
GOAL_PNIO_DIAG_HANDLE_T *pHdl	Diagnostics 処理格納用ポインタ
uint32_t api	API
uint16_t slot	Slot 番号
uint16_t subslot	sub slot 番号
uint16_t chan	チャネルインデックス
uint16_t numErr	エラー番号
uint16_t typeExtChanErr	拡張チャネルエラータイプ
uint32_t valExtChanAdd	拡張チャネルエラー値
GOAL_BOOL_T flgMaintReq	維持要求フラグ
GOAL_BOOL_T flgMaintDem	維持要請フラグ
GOAL_BOOL_T flgSubmitAlarm	発行アラームフラグ
uint16_t typeAlarm	アラームタイプ

6.3.42　goal_pnioDiagChanDiagRemove ― チャネル Diagnostics エントリ削除

チャネルの Diagnostics エントリを削除します。

GOAL_STATUS_T ステータスを返します。

表 6.50　goal_pnioDiagChanDiagRemove パラメータ

パラメータ	内容
GOAL_PNIO_T *pPnio	PROFINET データ
GOAL_PNIO_DIAG_HANDLE_T hdl	Diagnostics ハンドル
GOAL_BOOL_T flgSubmitAlarm	発行アラームフラグ

6.3.43　goal_pnioCyclicCtrl ― サイクリックデータ受信コールバック制御

サイクリックデータ受信のコールバックを制御します。flgCyclic が GOAL_TRUE に設定されると、コールバック GOAL_PNIO_CB_ID_NEW_IO_DATA が新規 I/O データの受信を示します。これにより、システム負荷が増大します。ほとんどのアプリケーションでは、goal_pnioDataOutputGet でサイクリックデータをポーリングするだけで十分です。

本関数は GOAL_STATUS_T ステータスを返します。

表 6.51　goal_pnioCyclicCtrl パラメータ

パラメータ	内容
GOAL_PNIO_T *pPnio	PROFINET データ
GOAL_BOOL_T flgCyclic	サイクリックデータ有効フラグ

6.4 EtherNet/IP アプリケーションプログラミングインタフェース

6.4.1 goal_eipInit

GOAL EtherNet/IP を初期化します。GOAL_STATUS_T を結果として返します。本関数の呼び出しは appl_init()関数内で行われる必要があります。パラメータは不要です。

GOAL での EtherNet/IP サポートを有効にするために、goal_eipInit 関数の呼び出しは appl_init 関数内で行われる必要があります。

```
1.  static GOAL_EIP_T *pHdlEip; /**< GOAL Ethernet/ IP handle */
2.  GOAL_STATUS_T appl_init( void
3.  )
4.  {
5.     GOAL_STATUS_T res;              /* result */
6.
7.     res = goal_eipInit();
8.     if (GOAL_RES_ERR(res)) {
9.        goal_logErr("Initialization of Ethernet/IP failed");
10.    }
11.    return res;
12. }
```

6.4.2 goal_eipNew

任意 ID の GOAL EtherNet/IP インスタンスを作成し、コールバックを登録します。本関数の呼び出しは appl_setup()関数内で行われる必要があります。

表 6.52 goal_eipNew パラメータ

パラメータ	内容
GOAL_EIP_T ** ppEip	[out] EtherNet/IP インスタンス参照
const uint32_t id	EtherNet/IP インスタンスの ID
GOAL_EIP_FUNC_CB_T pFunc	GOAL Ethernet/IP 処理アプリケーションコールバック

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

6.4.3 goal_eipCipClassRegister

アプリケーション固有の CIP クラスを登録します。登録された CIP クラスの要求が EtherNet/IP スタックで受信されると、ハンドラ関数に渡されます。

表 6.53 goal_eipCipClassRegister パラメータ

パラメータ	内容
GOAL_EIP_T *pHdlEip	GOAL Ethernet/IP ハンドル
uint16_t classId	登録するクラス ID
GOAL_EIP_REQ_HANDLER_T pFunc	要求ハンドラ

```
res = goal_eipCipClassRegister(pEip, APPL_CLASS_ID_PARAMETER, appl_parameterClassHandler);
```

6.4.4 goal_eipCreateAssemblyObject

選択されたクラスに CIP インスタンスを追加します。GOAL_STATUS_T を結果として返します。

表 6.54 goal_eipCreateAssemblyObject パラメータ

パラメータ	内容
GOAL_EIP_T *pHdlEip	GOAL EtherNet/IP ハンドル
uint32_t instanceId	作成するアセンブリオブジェクトのインスタンス番号
uint16_t len	アセンブリオブジェクトのデータ長

```
res = goal_eipCreateAssemblyObject(pHdlEip, GOAL_APP_ASM_ID_INPUT, GOAL_APP_ASM_SIZE_INPUT);
```

6.4.5 goal_eipCreateAssemblyObjectRpc

選択されたクラスに RPC 通信用の CIP インスタンスを追加します。GOAL_STATUS_T を結果として返します。

表 6.55 goal_eipCreateAssemblyObjectRpc パラメータ

パラメータ	内容
GOAL_EIP_T *pHdlEip	GOAL EtherNet/IP ハンドル
uint32_t instanceId	作成するアセンブリオブジェクトのインスタンス番号
uint16_t len	アセンブリオブジェクトのデータ長

```
res = goal_eipCreateAssemblyObjectRpc(pHdlEip, GOAL_APP_ASM_ID_INPUT,
GOAL_APP_ASM_SIZE_INPUT);
```

6.4.6　goal_eipAssemblyObjectGet

アセンブリのデータアレイへのポインタを取得します。

表 6.56　goal_eipAssemblyObjectGet パラメータ

パラメータ	内容
GOAL_EIP_T *pHdlEip	GOAL Ethernet/IP ハンドル
uint32_t instanceId	アセンブリオブジェクトのインスタンス番号
uint8_t **ppData	[Out] アセンブリデータへのポインタ
uint16_t *pLen	[Out] アセンブリデータ長

```
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

生産・消費エンドポイントとの Exclusive Owner（独占オーナー）接続を作成します。

表 6.57　goal_eipAddExclusiveOwnerConnection パラメータ

パラメータ	内容
GOAL_EIP_T *pHdlEip	GOAL EtherNet/IP ハンドル
uint32_t outputAssembly	本接続に使用する O→T ポイント
uint32_t inputAssembly	本接続に使用する T→O ポイント
uint32_t configAssembly	本接続に使用する設定ポイント

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

6.4.8　goal_eipAddInputOnlyConnection

同じ生産・消費エンドポイントとの複数の Input Only（入力オンリー）接続を作成します。

表 6.58　goal_eipAddInputOnlyConnection パラメータ

パラメータ	内容
GOAL_EIP_T *pHdlEip	GOAL EtherNet/IP ハンドル
uint32_t connNum	Input Only 接続数
uint32_t outputAssembly	本接続に使用する O→T ポイント
uint32_t inputAssembly	本接続に使用する T→O ポイント
uint32_t configAssembly	本接続に使用する設定ポイント

```
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

同じ生産・消費エンドポイントとの複数の Listen Only（リッスンオンリー）接続を作成します。

表 6.59 goal_eipAddListenOnlyConnection パラメータ

パラメータ	内容
GOAL_EIP_T *pHdlEip	GOAL EtherNet/IP ハンドル
uint32_t connNum	Listen Only 接続数
uint32_t outputAssembly	本接続に使用する O→T ポイント
uint32_t inputAssembly	本接続に使用する T→O ポイント
uint32_t configAssembly	本接続に使用する設定ポイント

```
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

EtherNet/IP バージョンを取得します。GOAL_STATUS_T を結果として返します。

表 6.60 goal_eipGetVersion パラメータ

パラメータ	内容
GOAL_EIP_T *pHdlEip	GOAL EtherNet/IP ハンドル
char **ppVersion	[out] Ethernet/IP バージョン

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

6.4.11 goal_eipDeviceStatusSet

デバイスステータスビットを設定します。パラメータ statusBits は GOAL_EIP_STATUS_マクロを使用します。これらのビットは論理和で組み合わせることが可能です。

表 6.61 goal_eipDeviceStatusSet パラメータ

パラメータ	内容
GOAL_EIP_T *pHdlEip	GOAL EtherNet/IP ハンドル
uint16_t statusBits	ステータスビットマップ

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

6.4.12 goal_eipDeviceStatusClear

デバイスステータスビットをクリアします。statusBits の全ビットがクリアされます。本パラメータは GOAL_EIP_STATUS_マクロを使用します。これらのビットは論理和で組み合わせることが可能です。

表 6.62 goal_eipDeviceStatusClear パラメータ

パラメータ	内容
GOAL_EIP_T *pHdlEip	GOAL EtherNet/IP ハンドル
uint16_t statusBits	ステータスビットマップ

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

6.4.13 goal_eipAssemblyObjectWrite

アセンブリオブジェクトにデータを書き込みます。

表 6.63 goal_eipAssemblyObjectWrite パラメータ

パラメータ	内容
GOAL_EIP_T *pHdlEip	GOAL Ethernet/IP handle
uint32_t instanceId	instance ID of the Assembly object
uint8_t *pData	buffer with write data
uint16_t len	number of bytes to write

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

6.4.14 goal_eipAssemblyObjectRead

アセンブリオブジェクトからデータを読み取ります。

表 6.64 goal_eipAssemblyObjectRead パラメータ

パラメータ	内容
GOAL_EIP_T *pHdlEip	GOAL Ethernet/IP handle
uint32_t instanceId	instance ID of the Assembly object
uint8_t *pData	buffer for read data
uint16_t len	number of bytes to read

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

6.4.15 goal_eipIdentitySerialNumberSet

デバイスのシリアル番号を設定します。 各デバイスには固有のシリアル番号が必要です。 この関数は、Identity インスタンス属性を更新します。つまり、EtherNet/IP インスタンスが作成された後にシリアル番号を変更します。

表 6.65 goal_eipIdentitySerialNumberSet parameters パラメータ

パラメータ	内容
GOAL_EIP_T *pHdlEip	GOAL Ethernet/IP handle
uint32_t serialNum	new Serial Number

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

6.5 EtherCAT アプリケーションプログラミングインタフェース

6.5.1 goal_ecatInit

目的：EtherCAT ライブラリの初期登録。

関数プロトタイプ）

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

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：EtherCAT ライブラリのインスタンスを作成する。

関数プロトタイプ）

```
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 */
);
```

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：Emergency メッセージのサポートを有効にする

関数プロトタイプ）

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

例）

```
/* enable CoE emergency */
res = goal_ecatCfgEmergencyOn(GOAL_TRUE);
if (GOAL_RES_ERR(res)) {
    goal_logErr("failed to enable CoE Emergency support");
}
```

注意：この関数は、goal_ecatNew の前に呼び出す必要があります。

6.5.4 goal_ecatCfgEmergencyQueueNum

目的：非同期な Emergency メッセージの数を設定する。

関数プロトタイプ）

```
GOAL_STATUS_T goal_ecatCfgEmergencyQueueNum(
    int16_t num                                 /**< number of queue slots */
);
```

例）

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

注意：この関数は、goal_ecatNew の前に呼び出す必要があります。

6.5.5 goal_ecatCfgFoeOn

目的：FoE support を有効にする。

関数プロトタイプ）

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

例）

```
/* enable FoE */
res = goal_ecatCfgFoeOn(GOAL_TRUE);
if (GOAL_RES_ERR(res)) {
    goal_logErr("failed to enable FoE support");
}
```

注意：この関数は、goal_ecatNew の前に呼び出す必要があります。

6.5.6 goal_ecatCfgExplDevIdOn

目的：Explicit Device ID のサポートを有効にする。

関数プロトタイプ）

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

例）

```
/* enable Explicit Device Identification */
res = goal_ecatCfgExplDevIdOn(GOAL_TRUE);
if (GOAL_RES_ERR(res)) {
    goal_logErr("failed to enable Explicit Device Identification support");
}
```

注意：この関数は、goal_ecatNew の前に呼び出す必要があります。

6.5.7 goal_ecatCfgBootstrapOn

目的：BOOTSTRAP ESM 状態のサポートを有効にする。

関数プロトタイプ）

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

例）

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

注意：この関数は、goal_ecatNew の前に呼び出す必要があります。

6.5.8 goal_ecatCfgDcRequiredOn

目的：DC モードの実行を有効にする。

関数プロトタイプ）

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

例）

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

注意：この関数は、goal_ecatNew の前に呼び出す必要があります。

6.5.9　goal_ecatCfgSizePdoStreamBufSet

目的：PDO データバッファを構成する

関数プロトタイプ）

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

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);
}
```

注意：この関数は、goal_ecatNew の前に呼び出す必要があります。また、CC の最大プロセスデータ長を利用するには、バッファを 2 * 68 = 136 バイトに構成する必要があります。

6.5.10　goal_ecatCfgLedStatusIndicator

目的：LED エミュレーションの動作を構成する。

関数プロトタイプ）

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

例）

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

注意：この関数は、goal_ecatNew の前に呼び出す必要があります。
FALSE に設定すると、LED エミュレーションは専用の RUN および ERROR インジケータをサポートします。
TRUE に設定すると、LED エミュレーションは結合された STATUS インジケータをサポートします。

6.5.11　goal_ecatObjAddrGet

目的：オブジェクト値へのポインタを取得します。

関数プロトタイプ）

```
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 */
);
```

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);
}
```

注意：無し。

6.5.12 goal_ecatObjValGet

目的：オブジェクトの値を取得する。

関数プロトタイプ）

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

例）

```
GOAL_STATUS_T res;                          /* return */
uint32_t value = 0;                         /* variable value */
uint32_t size = 0;                          /* variable size */

res = goal_ecatObjValGet(pHdlEcat, 0x1001, 0, (uint8_t *) &value, &size);
if (GOAL_RES_ERR(res)) {
    return res;
}
```

注意：無し。

6.5.13 goal_ecatObjValSet

目的：オブジェクトの値を設定する。

関数プロトタイプ）

```
GOAL_STATUS_T goal_ecatObjValSet(
    GOAL_ECAT_T *pEcat,                     /**< EtherCAT instance data */
    uint16_t index,                         /**< object index */
    uint8_t subIndex,                       /**< object sub-index */
    uint8_t *pData,                         /**< [in] new object value */
    uint32_t size                           /**< size of new data */
);
```

例）

```
res = goal_ecatObjValSet(
          pHdlEcat,
          0x1008,
          0x00,
          (uint8_t *) APPL_ECAT_PRODUCT_NAME,
          GOAL_STRLEN(APPL_ECAT_PRODUCT_NAME));
if (GOAL_RES_ERR(res)) {
    goal_logErr("failed to set product name in object dictionary");
    return res;
}
```

注意：無し。

6.5.14 goal_ecatdynOdObjAdd

目的：オブジェクトを追加する。

関数プロトタイプ）

```
GOAL_STATUS_T goal_ecatdynOdObjAdd(
    GOAL_ECAT_T *pEcat,                    /**< EtherCAT instance data */
    uint16_t index,                        /**< object index */
    uint16_t objTpye,                      /**< object type (VAR, ARRAY, RECORD) */
    uint16_t dataType                      /**< data type of object */
);
```

例）

```
res = goal_ecatdynOdObjAdd(
    pHdlEcat,
    0x1000,
    GOAL_ECAT_OBJCODE_VAR,
    GOAL_ECAT_DATATYPE_UNSIGNED32);
```

注意：無し。

6.5.15　goal_ecatdynOdSubIndexAdd

目的：オブジェクトに Cyclic 通信サブインデックスを追加する。

関数プロトタイプ）

```
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 */
);
```

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);
```

注意：無し。

6.5.16　goal_ecatdynOdSubIndexRpcAdd

目的：オブジェクトに RPC 通信サブインデックスを追加する。

関数プロトタイプ）

```
GOAL_STATUS_T goal_ecatdynOdSubIndexRpcAdd(
    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 */
);
```

例）

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

res = goal_ecatdynOdSubIndexRpcAdd(
    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);
```

注意：無し。

6.5.17 goal_ecatdynOdObjNameAdd

目的：オブジェクトに名前を割り当てる。

関数プロトタイプ）

```
GOAL_STATUS_T goal_ecatdynOdObjNameAdd(
    GOAL_ECAT_T *pEcat,                     /**< EtherCAT instance data */
    uint16_t index,                         /**< object index */
    char *pName                             /**< name of object */
);
```

例）

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

注意：無し。

6.5.18 goal_ecatdynOdSubIndexNameAdd

目的：サブインデックスに名前を割り当てる

関数プロトタイプ）

```
GOAL_STATUS_T goal_ecatdynOdSubIndexNameAdd(
    GOAL_ECAT_T *pEcat,                     /**< EtherCAT instance data */
    uint16_t index,                         /**< object index */
    uint8_t subIndex,                       /**< object sub-index */
    char *pName                             /**< name of sub-index */
);
```

例）

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

注意：無し。

6.5.19 goal_ecatdynOdFinish

目的：オブジェクトの作成を終了します。

関数プロトタイプ）

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

例）

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

注意：この関数は、オブジェクトの動的作成を終了します。 最後のオブジェクトが作成されたときに呼び出す必要があります。 この関数を呼び出した後は、追加のオブジェクトを作成できません。

6.5.20 goal_ecatEsmStateGet

目的：現在の ESM 状態を取得する。

関数プロトタイプ）

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

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) {
}
```

注意：無し。

6.5.21 goal_ecatEmcyReqWrite

目的：送信用の Emergency メッセージを生成する。

関数プロトタイプ）

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

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);
```

注意：無し。

6.5.22 goal_ecatGetVersion

目的：EtherCAT ライブラリに関するバージョン情報を取得する。

関数プロトタイプ）

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

例）

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

注意：無し。

6.5.23 goal_ecatEsmLocalErrorSet

目的：アプリケーションからのエラーを設定する。

関数プロトタイプ）

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

例）

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

注意：無し。

6.6 ネットワーク

6.6.1 goal_netRpcInit ― ネットワークの RPC 機能初期化

目的：ネットワーク機能（IP 設定、UDP チャネル、TCP チャネル）が必要な場合に、アプリケーション初期化中に本関数を呼び出す必要があります。

関数プロトタイプ）

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

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 — ネットワークオープン

目的：使用するネットワークメディアアダプタ（MA）を開きます。

関数プロトタイプ）

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

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 — ネットワーククローズ

目的：ネットワークメディアアダプタ（MA）を閉じます。

関数プロトタイプ）

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

6.6.4　goal_maNetGetById ― ネットワークメディアアダプタ（MA）ハンドル取得

目的：使用する、先に開いたネットワークメディアアダプタ（MA）ハンドルを取得します。

関数プロトタイプ）

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

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 ― IP アドレス設定

目的：ネットワークインターフェイスの IP アドレスを設定します。

表 6.66　goal_maNetIpSet パラメータ

パラメータ	内容
GOAL_MA_NET_T *pNetHdl	GOAL NET ハンドル
uint32_t addrIp	IP アドレス
uint32_t addrMask	ネットマスク
uint32_t addrGw	ゲートウェイ
GOAL_BOOL_T flgTemp	GOAL_TRUE：変数"VALID"の設定値を維持 GOAL_FALSE：変数"VALID"を 1 に設定

関数プロトタイプ）

```
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 */
);
```

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 チャネル

6.7.1　goal_maChanTcpOpen ― TCP チャネルメディアアダプタ（MA）オープン

目的：再度使用するネットワークメディアアダプタ（MA）を開きます。

アプリケーションのセットアップ時に一度これを行う必要があります。

関数プロトタイプ）

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

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 ― 新規 TCP チャネル作成

目的：新規 TCP チャネルを作成します。

関数プロトタイプ）

```
GOAL_STATUS_T goal_maChanTcpNew(
  GOAL_MA_CHAN_TCP_T *pChanTcpHdl,          /**< pointer to store CHAN_TCP handle */
  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 */
);
```

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 ― 作成した TCP チャネルの起動

目的：先に作成した TCP チャネルを起動します。起動後、接続を確立するか、または入ってきた接続要求を受け付けます。

関数プロトタイプ）

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

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 ― チャネルのノンブロッキング設定

目的：ソケットをノンブロッキングモードに設定します。

関数プロトタイプ）

```
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 */
);
```

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 — TCP チャネルのリモートアドレス取得

目的：TCP チャネルのリモートエンドポイントの IP アドレスを取得します。

関数プロトタイプ）

```
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 */
);
```

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 — TCP チャネルへのデータ送信

目的：先に開いた TCP チャネルにデータを送信します。

関数プロトタイプ）

```
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 */
);
```

例）

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

6.8　UDP Channel

6.8.1　goal_maChanUdpOpen ― UDP チャネル MA オープン

目的：UDP チャネル MA を開きます。

アプリケーションのセットアップ時に一度これを行う必要があります。

関数プロトタイプ）

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

例）

```
/* pen 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 ― UDP チャネル MA ハンドル取得

目的：先に開いた UDP チャネル MA ハンドルを取得します。

関数プロトタイプ）

```
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 ― 新規 UDP チャネル作成

目的：新規 UDP チャネルを作成します。

関数プロトタイプ）

```
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 */
);
```

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 ― UDP チャネル MA クローズ

目的：現在のチャネルを閉じます。

関数プロトタイプ）

```
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 — オープンチャネルのノンブロッキングアクセス設定

目的：チャネルをノンブロッキング動作に設定します。

関数プロトタイプ）

```
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 */
);
```

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 — オープン UDP チャネルのブロードキャスト動作設定

目的：チャネルをブロードキャストに設定します。

関数プロトタイプ）

```
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 */
);
```

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 ― UDP チャネルのリモートアドレス取得

目的：UDP チャネルのリモートアドレス（データを受信したアドレス）を取得します。

関数プロトタイプ）

```
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 ― UDP チャネルの起動

目的：チャネルを起動します。

関数プロトタイプ）

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

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 ― UDP チャネルへのデータ送信

目的：オープン UDP チャネルにデータを送信します。

関数プロトタイプ）

```
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 */
);
```

例）

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

7. Web サーバ

7.1 概要

GOAL には、組み込みシステム用の小型 Web サーバがあります。Web サーバは、

- ファイルダウンロード
- 現在のデバイスの状態とプロパティに関する情報を取得

するように設計されています。

Web サーバは、以下のプロパティをサポートしています。

転送プロトコル：HTTP / HTTPS

要求方法；GET / POST

1 ページまたは複数ページの Web ページを、Web サーバの 1 つのインスタンスに割り当てることができます。Web ページはアプリケーションの一部であり、アプリケーションによって使用可能にする必要があります。Web サーバは、このためのコールバック機能を提供しています。「**8.5 コールバック関数**」に記載の cbHttpReqFunc()を参照してください。

現在のデバイスの状態とプロパティは、CM 変数およびアプリケーション固有変数から読み出せます。アプリケーション固有変数は、簡易変数または 1 次元リストとして構成可能です。

アプリケーション固有変数の現在値のプレースホルダを持つ Web ページ用テンプレートを格納することが可能です。テキスト置換については、「**8.3 Web テンプレート**」に記載されています。Web テンプレートは、Web ページの動的管理を可能にします。

Web サーバへのアクセスは、ユーザレベルで制限することが可能です。アプリケーションは、どのユーザレベルがデバイスでサポートされるのか、またどの権限をユーザレベルが持つかを指定することができます。各ユーザレベルのユーザ名とパスワードから成る認証データは、CM 変数で設定可能です。GOAL の Web サーバは、最大 4 つのユーザレベルを提供します。

ユーザレベルは、Web サーバの全インスタンスで適用可能です。Web サーバの各インスタンスでは、登録中に有効なユーザレベルを指定できます。Web 要求は、認証に成功しないとアプリケーションに転送されません。例えば、「**8.5 コールバック関数**」のコールバック関数 cbHttpReqFunc()は、ログインに成功しないと呼び出されません。HTTP 転送プロトコルを使用した Web サーバインスタンス経由のユーザ名とパスワードの転送は安全ではありません。ポートは、HTTPS 転送プロトコルが動作するポートを推奨します。

HTTPS は、コンパイラ定義の GOAL_CONFIG_HTTPS によって起動します。HTTPS は、暗号化と認証に外部ソフトウェアコンポーネント mbedTLS を使用します。mbedTLS へのアクセスは、TLS のメディアアダプタで実現されます。HTTPS の TLS は、goal_httpsNew()関数で初期化され、開かれます。

HTTPS の CM 変数に関しては、秘密鍵および秘密鍵の X509 証明書をインストールすることが可能です。証明書が格納されていない場合、Web サーバはポートが提供するデフォルトの証明書を受け取ります。

例）

- …\appl\goal_http\01_get*:
  （Web ページのアップロード）
- …\appl\goal_http\05_template_cm*:
  （CM 変数およびアプリケーション固有変数を持つ Web テンプレートのアップデート）

- …\appl\goal_http\06_template_list*:
  （リストを持つ Web テンプレートのアップロード）
- …\appl\goal_http\04_auth*:
  （ユーザレベルに関する認証）
- …\appl\goal_http\02_post*:
  （ファイルダウンロード）

7.2 設定

7.2.1 コンパイラ定義

Web サーバの設定には、以下のコンパイラ定義を使用します。

GOAL_CONFIG_HTTP：

0：転送プロトコル HTTP を使用しない（デフォルト）
1：転送プロトコル HTTP を使用

GOAL_CONFIG_HTTPS：

0：転送プロトコル HTTPS を使用しない（デフォルト）
1：転送プロトコル HTTPS を使用

7.2.2 CM 変数

Web サーバの設定には、以下の CM 変数を使用します。

CM モジュール ID	GOAL_ID_HTTP
CM 変数 ID	0
CM 変数名	HTTP_CHANNELS_MAX
内容	HTTP 転送プロトコルで使用可能な最大接続数
CM データ型	UINT16
サイズ	2 バイト
デフォルト値	NVS 値または 0

CM モジュール ID	GOAL_ID_HTTP
CM 変数 ID	1
CM 変数名	HTTP_CHANNELS_MAX
内容	HTTPS 転送プロトコルで使用可能な最大接続数
CM データ型	UINT16
サイズ	2 バイト
デフォルト値	NVS 値または 0

CM モジュール ID	GOAL_ID_HTTP
CM 変数 ID	2
CM 変数名	USERLEVEL0
内容	レベル 0 の認証データ
CM データ型	STRING
サイズ	32 バイト
デフォルト値	NVS 値または空文字列

CM モジュール ID	GOAL_ID_HTTP
CM 変数 ID	3
CM 変数名	USERLEVEL1
内容	レベル 1 の認証データ
CM データ型	STRING
サイズ	32 バイト
デフォルト値	NVS 値または空文字列

CM モジュール ID	GOAL_ID_HTTP
CM 変数 ID	4
CM 変数名	USERLEVEL2
内容	レベル 2 の認証データ
CM データ型	STRING
サイズ	32 バイト
デフォルト値	NVS 値または空文字列

CM モジュール ID	GOAL_ID_HTTP
CM 変数 ID	5
CM 変数名	USERLEVEL3
内容	レベル 3 の認証データ
CM データ型	STRING
サイズ	32 バイト
デフォルト値	NVS 値または空文字列

HTTPS で使用される TLS 層の設定には、以下の CM 変数を使用します。

CM モジュール ID	GOAL_ID_HTTPS
CM 変数 ID	0
CM 変数名	TLS_SERVER_CERTIFICATE
内容	Web サーバの証明書
CM データ型	GENERIC
サイズ	1024 バイト
デフォルト値	NVS 値またはポートからの証明書

CM モジュール ID	GOAL_ID_HTTPS
CM 変数 ID	1
CM 変数名	TLS_PRIVATE_KEY
内容	Web サーバの秘密鍵
CM データ型	GENERIC
サイズ	1024 バイト
デフォルト値	NVS 値または空文字列

CM モジュール ID	GOAL_ID_HTTPS
CM 変数 ID	2
CM 変数名	TLS_SRV_CERT_CA_CN
内容	認証機関の共通サーバ名
CM データ型	STRING
サイズ	128 バイト
デフォルト値	NVS 値または空文字列

CM モジュール ID	GOAL_ID_HTTPS
CM 変数 ID	3
CM 変数名	TLS_SRV_CERT_CA_O
内容	認証機関の名称（会社名）
CM データ型	STRING
サイズ	128 バイト
デフォルト値	NVS 値または空文字列

CM モジュール ID	GOAL_ID_HTTPS
CM 変数 ID	4
CM 変数名	TLS_SRV_CERT_CA_C
内容	認証機関が所在する国
CM データ型	STRING
サイズ	8 バイト
デフォルト値	NVS 値または空文字列

CM モジュール ID	GOAL_ID_HTTPS
CM 変数 ID	5
CM 変数名	TLS_SRV_CERT_CN
内容	Web サーバの共通名
CM データ型	STRING
サイズ	128 バイト
デフォルト値	NVS 値または空文字列

CM モジュール ID	GOAL_ID_HTTPS
CM 変数 ID	6
CM 変数名	TLS_SRV_CERT_O
内容	Web サーバ提供機関の名称
CM データ型	STRING
サイズ	128 バイト
デフォルト値	NVS 値または空文字列

CM モジュール ID	GOAL_ID_HTTPS
CM 変数 ID	7
CM 変数名	TLS_SRV_CERT_C
内容	Web サーバ提供機関が所在する国
CM データ型	STRING
サイズ	8 バイト
デフォルト値	NVS 値または空文字列

CM モジュール ID	GOAL_ID_HTTPS
CM 変数 ID	8
CM 変数名	TLS_SRV_CERT_NOT_BEFORE
内容	証明書が有効になる日付
CM データ型	STRING
サイズ	20 バイト
デフォルト値	NVS 値または空文字列

CM モジュール ID	GOAL_ID_HTTPS
CM 変数 ID	9
CM 変数名	TLS_SRV_CERT_NOT_AFTER
内容	証明書が無効になる日付
CM データ型	STRING
サイズ	20 バイト
デフォルト値	NVS 値または空文字列

7.3 Web テンプレート

GOAL の Web サーバでは、最新情報のプレースホルダを持つ Web ページ用テンプレートを実装することが可能です。プレースホルダは、アップロード処理中に、Web サーバによって現在値に置き換えられます。Web サーバは、

- CM 変数、
- アプリケーション固有変数、
- リスト

のプレースホルダを提供します。

7.3.1 CM 変数

CM 変数のプレースホルダには、CM モジュール ID と CM 変数 ID が入っています。Web サーバは、プレースホルダの CM 変数への置き換えを自動で行います。

構文）

```
[CM:<modNum>, <cmVarNum>]
```

例）

```
[CM:0, 2]
```

7.3.2 アプリケーション固有変数

アプリケーション固有変数のプレースホルダには、アプリケーションの変数名が入っています。Web サーバは、コールバック関数を呼び出すことで、アプリケーションから変数の現在値を要求し（「**8.5 コールバック関数**」に記載の cpHttpTemplateFunc()を参照）、Web ページ上のプレースホルダを置き換えます。

構文）

```
[VAR:<appIVarName>]
```

例）

```
[VAR:appIVar]
```

7.3.3 リスト

Web サーバは、HTML テキストでリストを生成するための有効な手法を提供します。単一のリストエントリの HTML テキストは、Web テンプレートのプレースホルダ FOREACH、/FOREACH に入れられます。FOREACH は 1 次元リストを示し、プレースホルダ FOREACH と/FOREACH の間の HTML テキストが各リスト要素に対して実行されます。リストエントリの配置は、任意の変数名のプレースホルダ VAR で HTML テキストに示されます。プレースホルダ VAR の置き換え後、Web サーバは自動的に次のリストエントリに移行します。同じリストエントリを 2 回置き換えることはできません。Web テンプレートには最初のリストエントリのみ記述する必要があります。

Web サーバは登録時、ID、リスト名、リスト要素数のみ取得します。リスト要素の内容はアプリケーションによって管理されます。Web サーバはコールバック関数を呼び出して、次のリスト要素の内容を取得します。「**8.5 コールバック関数**」に記載の cpHttpTemplateFunc()を参照してください。

リストをネストすることも可能です。サポートされているネストの最大深さは 4 です。

構文）

```
[FOREACH:<listName>] … [/FOREACH]
```

HTML リストの例）

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

例…\appl\goal_http\06_template_list*は、HTML リストを生成します。Web ブラウザ上の表示を **図 8.1** に示します。

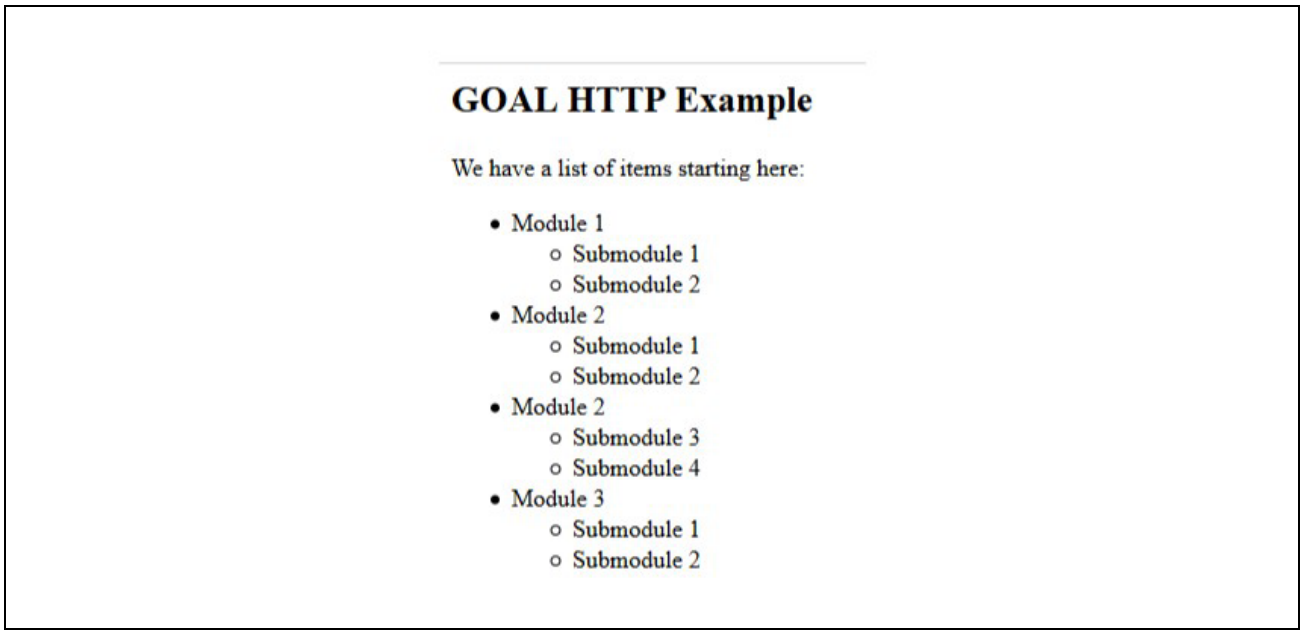

図 8.1 サンプル 06_template_list の Web ページ

プレースホルダ[FOREACH] … [\FOREACH]は、表などの他の HTML フォーマットと統合することもできます。

7.4 文字

角括弧：

Web テンプレートのプレースホルダが角括弧で囲まれているため、HTML テキストで角括弧を表示する場合には、二重に記述する必要があります。

例）配列の指示を Web ページに表示する場合

HTML テキスト : applArray[[5]]
Web ブラウザ表示 : applArray[5]

二重引用符：

C コードの文字列が二重引用符で囲まれているため、HTML テキスト中の二重引用符はバックスラッシュで保護する必要があります。

例）`uint8_t webPage[] = “<html><meta charset = \”utf-8\” > … </html>”;`

HTML テキストの規則は、**その他すべての文字**に対して有効です。

7.5 コールバック関数

プロトタイプ	GOAL_STATUS_T cbHttpReqFunc(GOAL_HTTP_APPLCB_DATA_T *applData)	
説明	受信した有効な Web 要求がアプリケーションに渡されます。アプリケーションは Web 要求を処理し、Web 応答を生成する必要があります。	
パラメータ	applData	アプリケーションへのデータとアプリケーションから返されたデータ
戻り値	GOAL 戻りステータス	
区分	任意	
登録	goal_httpdResReg()関数の実行時	

プロトタイプ	GOAL_STATUS_T cbHttpTemplateFunc(GOAL_HTTP_APPLCB_TEMPL_T *pWebTemplate)	
説明	本コールバック関数に関して、アプリケーションは特定のプレースホルダに最新情報を提供します。本コールバック関数は Web テンプレートの各プレースホルダに対して呼び出されます。同じ情報に対して複数のプレースホルダがある場合には、本コールバック関数は各プレースホルダに対して呼び出されます。	
パラメータ	pWebTemplate	変数やリストを指定するための情報、および指定された変数の現在の戻り値
戻り値	GOAL 戻りステータス	
区分	任意	
登録	関数実行時	

7.6 実装ガイドライン

7.6.1 Web ページのアップロード

Web ページ“device.html”をデバイスから Web ブラウザにアップロードします。Web ページの内容が文字列として変数 webPage[]に格納されます。Web ページ上のテキスト置換は必要ありません。

1. GOAL_FSA_INIT_APPL または GOAL_FSA_INIT_GOAL 状態で、Web サーバを初期化します。

```
goal_httpInit();
```

2. GOAL_FSA_INIT_SETUP 状態で、TCP ポート 8081 を使用して Web サーバの 1 つのインスタンスを開きます。

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

3. GOAL_FSA_INIT_SETUP 状態で、開いた Web サーバに対するコールバック関数と使用方法を登録します。テキスト置換のコールバック関数は登録しません。

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

4. Web ページを文字列変数として提供します。

```
const uint8_t webPage[] = "<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 middlewar.</p> \r\n\
            </body></html>"
```

5. Web 要求を処理するコールバック関数を実装します。

```
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((const char*) webPage));
             res = GOAL_OK;
        }
   }
   return res;
}
```

6. Web 要求の受け付け後、Web サーバからコールバック関数 applWebReqCb()が呼び出されます。

7.6.2 Web ページのアップロード

Web ページ“device.html”をデバイスから Web ブラウザにアップロードします。Web ページはテンプレート webPage[]をベースとします。テンプレートには、CM モジュール ID 34 および CM 変数 ID 0 の CM 変数 DD_CM_VAR_MODULENAME のプレースホルダがあります。Web サーバはプレースホルダを自動的に CM 変数の現在値に置き換えます。Web ページ上のテキスト置換は必要ありません。

1. GOAL_FSA_INIT_APPL または GOAL_FSA_INIT_GOAL 状態で、Web サーバを初期化します。

```
goal_httpInit();
```

2. GOAL_FSA_INIT_SETUP 状態で、TCP ポート 8081 を使用して Web サーバの 1 つのインスタンスを開きます。

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

3. GOAL_FSA_INIT_SETUP 状態で、開いた Web サーバに対するコールバック関数と使用方法を登録します。テキスト置換のコールバック関数は登録しません。

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

4. Web ページ用テンプレートを CM 変数のプレースホルダで文字列変数として提供します。

```
const uint8_t webPage[] = "<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. Web 要求を処理するコールバック関数を実装します。

```
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((const char*) webPage));
             res = GOAL_OK;
        }
   }
   return res;
}
```

6. Web 要求を受け付け後、Web サーバからコールバック関数 applWebReqCb()が呼び出されます。

7.6.3 アプリケーション固有変数の読み出し

Web ページ“device.html”をデバイスから Web ブラウザにアップロードします。Web ページはテンプレート webPage[]をベースとします。テンプレートには、アプリケーション固有変数 applVar のプレースホルダがあります。Web サーバはコールバック関数 applWebGetValCb()を呼び出して、アプリケーション固有変数の現在値を提供します。Web サーバはプレースホルダをアプリケーション固有変数の現在値に置き換えます。

1. GOAL_FSA_INIT_APPL または GOAL_FSA_INIT_GOAL 状態で、Web サーバを初期化します。

```
goal_httpInit();
```

2. GOAL_FSA_INIT_SETUP 状態で、TCP ポート 8081 を使用して Web サーバの 1 つのインスタンスを開きます。

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

3. GOAL_FSA_INIT_SETUP 状態で、開いた Web サーバに対するコールバック関数と使用方法を登録します。

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

4. Web ページ用テンプレートをアプリケーション固有変数のプレースホルダで文字列変数として提供します。

```
const uint8_t webPage[] = "<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. Web 要求を処理するコールバック関数を実装します。

```
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((const char*) webPage));
             res = GOAL_OK;
        }
   }
   return res;
}
```

6. アプリケーションからアプリケーション固有変数の現在値を取得するコールバック関数を実装します。

```
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 要求を受け付け後、Web サーバからコールバック関数 applWebReqCb()が呼び出されます。Web ページの任意のテンプレートが Web サーバで使用可能になります。

8. コールバック関数 applWebGetValCb()が呼び出され、置換が行われます。

7.6.4 リストの読み出し

Web ページ“device.html”をデバイスから Web ブラウザにアップロードします。Web ページはテンプレート webPage[]をベースとします。テンプレートには、デバイスコンポーネントのリストのプレースホルダがあります。Web サーバはコールバック関数 applWebGetValCb()を呼び出して、リストエントリの現在値を提供します。Web サーバはプレースホルダをアプリケーション固有変数の現在値に置き換えます。

1. GOAL_FSA_INIT_APPL または GOAL_FSA_INIT_GOAL 状態で、Web サーバを初期化します。

```
goal_httpInit();
```

2. GOAL_FSA_INIT_SETUP 状態で、TCP ポート 8081 を使用して Web サーバの 1 つのインスタンスを開きます。

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

3. GOAL_FSA_INIT_SETUP 状態で、開いた Web サーバに対するコールバック関数と使用方法を登録します。

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

4. リスト情報を作成し、Web サーバに登録します。リスト情報は、リスト ID、リスト名、リストエントリ数で構成されます。

```
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. Web ページ用テンプレートをリストとリストエントリのプレースホルダで文字列変数として提供します。

```
const uint8_t webPage[] = "<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. Web 要求を処理するコールバック関数を実装します。

```
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((const char*) webPage));
             res = GOAL_OK;
        }
   }
   return res;
}
```

7. アプリケーションからリストエントリの現在値を取得するコールバック関数を実装します。

```
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->in.Path)->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 要求を受け付け後、Web サーバからコールバック関数 applWebReqCb()が呼び出されます。Web ページの任意のテンプレートが Web サーバで使用可能になります。

9. コールバック関数 applWebGetValCb()が各置換に対して呼び出されます。

7.6.5 ユーザレベルの設定

Web ページ“admin.html”のアップロードは、USERLEVEL0 のユーザに対してのみ許可されます。
USERLEVEL0 のログインデータは、以下です。

- ユーザ名：admin
- パスワード：a1b2c3:UL

HTTPS 転送プロトコルが使用されます。

Web ページ“admin.html”にはプレースホルダがありません。Web ページ上のテキスト置換は必要ありません。

1. GOAL_FSA_INIT_APPL または GOAL_FSA_INIT_GOAL 状態で、Web サーバを初期化します。

```
goal_httpInit();
```

2. GOAL_FSA_INIT_SETUP 状態で、TCP ポート 443 を使用して Web サーバの 1 つのインスタンスを開きます。

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

3. GOAL_FSA_INIT_SETUP 状態で、開いた Web サーバに対するコールバック関数と使用方法を登録します。

```
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. GOAL_FSA_INIT_SETUP 状態で、USERLEVEL0 の認証をインストールします。認証データが CM 変数 USERLEVEL0 に書き込まれます。

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

5. Web ページ用テンプレートを文字列変数として提供します。

```
const uint8_t webPage[] = "<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. Web 要求を処理するコールバック関数を実装します。

```
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((const char*) webPage));
             res = GOAL_OK;
        }
   }
   return res;
}
```

7. Web 要求を受け付け後、Web サーバからコールバック関数 applWebReqCb()が呼び出されます。本 Web サーバへのログインが正常に行われている必要があります。

7.6.6 ファイルのダウンロード

Web サーバはダウンロードダイアログを提供します。受信ファイルがアプリケーションに転送されます。

1. GOAL_FSA_INIT_APPL または GOAL_FSA_INIT_GOAL 状態で、Web サーバを初期化します。

```
goal_httpInit();
```

2. GOAL_FSA_INIT_SETUP 状態で、TCP ポート 8081 を使用して Web サーバの 1 つのインスタンスを開きます。

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

3. GOAL_FSA_INIT_SETUP 状態で、開いた Web サーバに対するコールバック関数と使用方法を登録します。

```
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. Web ページを文字列変数として提供します。

```
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. 新しいファームウェアを受信、インストールするアプリケーション固有関数 applDownload() を実装します。

6. Web 要求の結果をアプリケーションに通知するアプリケーション固有関数 applDownloadFinished() を実装します。

7. Web 要求を処理するコールバック関数を実装します。

```
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((const char*) 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 要求を受け付け後、Web サーバからコールバック関数 applWebReqCb()が呼び出されます。

8. トラブルシューティング

本章では、R-IN32M3 Module のサンプル使用時に起こり得る共通の問題について記載します。

8.1 起動に関する問題

サンプルアプリケーションを起動して、期待するアプリケーション動作が見られない場合は、ログを確認してください。

```
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
```

“sync needs local reset to proceed”という最終ログエントリが表示されたら、通信コントローラ（CC）をリセットする必要があります。R-IN32M3 Module ボード上の”RST”ボタンを使用してリセットできます。

8.2 接続に関する問題

SPI 通信が全く行われない場合、以下の最終ログメッセージで確認できます。

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

ボード接続およびアプリケーションコントローラ（AC）のアプリケーションが正常に動作しているか確認してください。

8.3 IP 設定

EtherNet/IP プロジェクトにおいて、再起動後、静的 IP 設定と DHCP との切り替えに失敗した場合は、Management Tool を使用して以下の CM 変数を確認してください。

表 9.1 IP 設定

モジュール ID	変数 ID
GOAL_ID_NET	IP
GOAL_ID_NET	NETMASK
GOAL_ID_NET	GW
GOAL_ID_NET	VALID
GOAL_ID_NET	DHCP_ENABLED

DHCP を無効にするには、変数 DHCP_ENABLED を 0 に設定してください。変数”VALID”が 1 に設定されていることを確認してください。本設定を CC にアップロードし、恒久的に保存してください。DHCP は再起動後に無効になります。

改訂記録	R-IN32M3 Module ユーザーズマニュアル　ソフトウェア編

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2020.08.03	—	初版発行
2.00	2020.11.06	19, 24, 86-90, 135-150,	EtherCAT 機能を追加
		36	4.10.1.2 章 制御インタフェースの追加
		97-102	6.1 章 API に 6.1.8 ~ 6.1.17 章を追加
		134-134	6.4 章に新しい章を追加 6.4.13, 6.4.14, 6.4.15
		161-161	7 章の最適化
		161	9.3 章 ロギングの説明追加
2.01	2021.01.15	51, 73, 118, 118	デフォルト値を変更
2.02	2021.06.14	17	1.1 章産業用イーサネットプロトコル仕様 追加
		44	SPI 速度の記載を削除
		168	7.4 章の説明を更新
2.03	2021.10.15	16	uGOAL の説明を追加
		17	FW2.1.0.0 に合わせて更新
		19, 20, 22	uGOAL の文言を追加
		26	9.4 章の内容を 4.9 章に統合
		51	5.4.1 章を削除、以降の章を繰り上げ
		61	5.4.46 章を削除、以降の章を繰り上げ
		63	5.5.1 章の関数名誤記を修正
		68	5.5.19 章を修正
		70	5.5.23 章を修正
		77	5.6.16 章に注意事項を追記
		80, 80	5.6.23, 5.6.24 章を追加
		84, 85	5.7.10, 5.7.11 章を追加
		86	5.8.1～5.8.6 章を削除(6.5 章で説明)、以降の章を繰り上げ
		89	5.9 章を全体的に見直し(説明の変更、章の追加)
		99	6.1.12 章を追加
		153	6.6.5 章に引数の説明を追加
		162	サンプルソフトウェアを 7.2 章に移動
		—	goal_appl.h を goal_config.h に誤記修正
		178	トラブルシューティングを 9 章に移動
2.04	2022.08.05	—	軽微な修正
2.05	2023.05.31	—	軽微な修正
		—	Renesas Synergy サンプルソフトに関する説明を削除
		172	7.6.4 誤記訂正: [VAR: deviceComponents] > [VAR:devComponent]
		172	7.6.4 コールバック関数処理訂正
		122	6.3.30. RPC 通信用 subslot の module 追加 API 説明追加　pnioRpcSubmodPlug()
		129	6.4.5. RPC 通信用インスタンス追加 API の説明追加 goal_eipCreateAssemblyObjectRpc()
		146	6.5.16. RPC 通信用 Subindex 追加 API の説明追加 goal_ecatdynOdSubIndexRpcAdd()

Rev.	発行日	改訂内容	
		ページ	ポイント

R-IN32M3 Module
ユーザーズマニュアル　ソフトウェア編

発行年月日　2023年5月31日　　Rev.2.05

発行　　ルネサス エレクトロニクス株式会社
　　〒135-0061　東京都江東区豊洲3-2-24（豊洲フォレシア）

R-IN32M3 Module

Renesas Electronics Corporation

R17US0002JJ0205