User's Manual

R-IN32M3 Module (RY9012A0)

User's Manual: Software
API Description

RENESAS MCU
R-IN32M3-EC

Renesas Electronics
www.renesas.com

Rev.1.00 June 2020

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

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

(Rev.4.0-1 November 2017)

Corporate Headquarters

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

www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit: www.renesas.com/contact/.

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)
   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on
   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state
   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins
   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals
   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin
   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses
   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products
   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

How to Use This Manual

1. Purpose and Target Readers

This manual is designed to provide the user with an understanding of the hardware functions and electrical characteristics of the R-IN32M3 module. It is intended for users designing application systems incorporating the MCU. A basic knowledge of electric circuits, logical circuits, and MCUs is necessary in order to use this manual.

> Particular attention should be paid to the precautionary notes when using the manual. These notes occur within the body of the text, at the end of each section, and in the Usage Notes section.

> The revision history summarizes the locations of revisions and additions. It does not list all revisions. Refer to the text of the manual for details.

The following documents apply to the R-IN32M3 module. Make sure to refer to the latest versions of these documents. Last four digits of document number (described as ****) indicate version information of each document. The newest versions of the documents listed may be obtained from the Renesas Electronics Web site.

Document Type	Description	Document Title	Document No.
Data Sheet	Hardware overview and electrical characteristics	R-IN32M3 Module Datasheet	R19DS0100ED****
User's manual for Hardware	Hardware specifications (pin assignments, peripheral function specifications, electrical characteristics, timing charts) and operation description	R-IN32M3 Module User's Manual: Hardware	R19UH0111ED****
User's manual for Software	Description of API	R-IN32M3 Module User's Manual: Software	This User's manual
Quick Start Guide	Information on application examples Sample program for Host CPU.	R-IN32M3 Module Application Note: Quick Start Guide	R12QS0043ED****
Renesas Technical Update	Product specifications, updates on documents, etc.	Available from Renesas Electronics Web site.	

2. Notation for Numbers and Symbols

Note:
Explanation of a point marked “Note” in the text

Caution:
Item deserving extra attention

Remark:
Supplementary explanation to the text

3. List of Abbreviations and Acronyms

Abbreviation	Full Form
AC	Application Controller
ACK	Acknowledge
API	Application Programming Interface
APMS	Acyclic Protocol Machine Sender
CC	Communication Controller
CIP	Common Industrial Protocol
CM	Configuration Manager
CPU	Central Processing Unit
DD	Device Detection
DHCP	Dynamic Host Configuration Protocol
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
PLC	Programmable Logic Controller
RPC	Remote Procedure Call
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

Table of Contents

1. Introduction 15

2. Document Structure 16

3. Device Architecture 17
3.1 Architecture 17
3.2 Interface 17
3.2.1 Hardware Interface 17
3.2.2 SPI Software Interface 18
3.2.3 Integration of Communication Layers / Middleware 18

4. Application 19
4.1 Introduction 19
4.2 Hardware Setup 19
4.3 Basic Application Setup 20
4.4 Default Features 20
4.5 Features 20
4.5.1 Device Detection 20
4.5.2 PNIO 21
4.5.3 EtherNet/IP 21
4.5.4 HTTPD 22
4.5.5 Network Channels 22
4.5.6 Generic Data Provider 22
4.6 External Reset 22
4.7 RPC Synchronization Reset 23
4.7.1 Communication Controller (CC) 23
4.7.2 Application Controller (AC) 23
4.8 IP Setting 23
4.9 Management Interface CC 23
4.9.1 Management Interface DD 24
4.9.2 PNIO 24
4.9.3 Logging Manager (LM) 24
4.9.4 Net 25
4.9.5 BOOT 26
4.9.6 Config Manager (CM) 26
4.9.7 ETH 26
4.9.8 EIP 27
4.9.9 HTTPD 27
4.9.10 CCM 27
4.10 Firmware Update 29
4.10.1 Update the Communication Controller 29
4.10.1.1 Firmware package 29
4.10.1.2 Control interface 29
4.10.1.3 Firmware update sequence 29
4.10.1.4 Keep update functionality while disabling DD 29
4.10.2 Update Possibilities for Application Controller 30
4.10.2.1 AC firmware update over HTTP 30
4.10.2.2 AC firmware update over TCP 31

5. Communication Stack 33
5.1 Introduction 33

5.2 SPI Data Exchange 33
5.2.1 Clock Domains and Communication Cycle 33
5.2.2 Technical Data 34
5.2.3 SPI Timing 35
5.2.3.1 SPI Speed 35
5.2.3.2 SPI Setup Timing 35
5.2.3.3 SPI Cycle Time 35
5.2.4 SPI Frame Structure 35
5.2.4.1 Fletcher 16 Checksum (16 Bits) 36
5.2.4.2 Sequence Counter (8 Bits) 36
5.2.4.3 Data Length (8 Bits) 36
5.3 Remote Procedure Call (RPC) 37
5.3.1 RPC Frame 37
5.3.1.1 Structure 37
5.3.1.2 Fletcher-16 Checksum (16 Bits) 37
5.3.1.3 Local Sequence (8 Bits) 37
5.3.1.4 Remote Sequence Acknowledge (8 Bits) 37
5.3.1.5 Data Length (8 Bits) 37
5.3.2 Flags (8 Bits) 38
5.3.2.1 Structure 38
5.3.2.2 Sync Request 38
5.3.2.3 Sync Acknowledge 38
5.3.2.4 Request Acknowledge 38
5.3.3 RPC Synchronization 38
5.3.4 RPC Protocol 39
5.3.4.1 Introduction 39
5.3.5 RPC Request/Response 39
5.3.5.1 Structure 39
5.3.5.2 Static Identifier 39
5.3.5.3 Data Length 40
5.3.5.4 RPC Id 40
5.3.5.5 Function Id 40
5.3.5.6 CTC Id 40
5.3.5.7 Flags 40
5.3.5.8 Data 40
5.3.5.9 Fletcher-16 Checksum (16 Bits) 40
5.4 Communicational Stack - PROFINET 41
5.4.1 Introduction 41
5.4.2 goal_pnioCfgVendorId – Set Vendor Id 41
5.4.3 goal_pnioCfgDeviceIdSet – Set Device Id 41
5.4.4 goal_pnioCfgVendorNameSet - Set Vendor Name 41
5.4.5 goal_pnioCfgPortDescSet - Set LLDP Port Description 41
5.4.6 goal_pnioCfgSystemDescSet - Set LLDP System Description 42
5.4.7 goal_pnioCfgOrderIdSet - Set Order Id 42
5.4.8 goal_pnioCfgSerialNumSet - Set Serial Number 42
5.4.9 goal_pnioCfgHwRevSet - Set Hardware Revision 43
5.4.10 goal_pnioCfgSwRevPrefixSet - Set Software Revision Prefix 43
5.4.11 pnioCfgSwRevFuncEnhSet - Set Software Revision Functional Enhancement 43
5.4.12 goal_pnioCfgSwRevBugfixSet - Set Software Revision Bugfix 44
5.4.13 goal_pnioCfgSwRevIntChgSet - Set Software Revision Internal Change 44
5.4.14 goal_pnioCfgSwRevCntSet - Set Software Revision Counter 44
5.4.15 goal_pnioCfgIm1TagFuncSet - Set I&M1 Tag Function 44
5.4.16 goal_pnioCfgIm1TagLocSet - Set I&M1 Tag Location 45

5.4.17 goal_pnioCfgIm2DateSet - Set I&M2 Date 45
5.4.18 goal_pnioCfgIm3DescSet - Set I&M3 Description 45
5.4.19 goal_pnioCfgIm4SigSet - Set I&M4 Signature (Functional Safety) 45
5.4.20 goal_pnioCfgLldpOrgExtSet - Configure LLDP Organizationally-specific Extension 46
5.4.21 goal_pnioCfgLldpOptTlvSet - Configure LLDP Optional TLV Parameters 46
5.4.22 goal_pnioCfgLldpGenMacSet - Configure LLDP Port MAC Address Generation 46
5.4.23 goal_pnioCfgIm14SupportSet - Configure I&M 1-4 Support 47
5.4.24 goal_pnioCfgIm14CbSet - Configure I&M 1-4 Callback 47
5.4.25 goal_pnioCfgIm0CbSet - Configure I&M 0 Callback 47
5.4.26 goal_pnioCfgIm0FilterDataCbSet - Configure I&M 0 Filter Data Callback 48
5.4.27 goal_pnioCfgRecDataBusyBufsizeSet - Configure Record Handle Storage Count 48
5.4.28 goal_pnioCfgRpcFragReqLenMaxSet - Configure Maximum Record Size 48
5.4.29 goal_pnioCfgRpcFragMaxCntSet – Configure Maximum RPC Fragment Number 49
5.4.30 goal_pnioCfgRpcFragEnableSet - Configure RPC Fragmentation 49
5.4.31 goal_pnioCfgRpcSessionMaxCntSet - Configure Maximum RPC Session Count 49
5.4.32 goal_pnioVendorIdSet - Set the vendor id 49
5.4.33 goal_pnioDeviceIdSet - Set the device id 49
5.4.34 goal_pnioHwRevSet - Set the hardware revision 50
5.4.35 goal_pnioSwRevSet - Set the software revision 50
5.4.36 goal_pnioProfileIdSet - Set the profile id 50
5.4.37 goal_pnioOrderIdSet - Set the order id 51
5.4.38 goal_pnioSerialNumSet - Set the serial number 51
5.4.39 goal_pnioVendorNameSet - Set the vendor name 51
5.4.40 goal_pnioPortDescSet - Set the port description 52
5.4.41 goal_pnioSystemDescSet - Set the system description 52
5.4.42 goal_pnioSubslotStateSet - Permit usage of wrong submodules (substitute) 52
5.4.43 goal_pnioConfClassGet - Get active conformance class 53
5.4.44 goal_pnioConfClassSet - Set active conformance class 53
5.4.45 goal_pnioConfTestGet - Get current PROFINET test environment 53
5.4.46 Data Mapper API 54
5.4.47 Map Subslot Data – goal_pnioDmSubslotAdd 54
5.4.48 Map Subslot IOCS/IOPS - goal_pnioDmSubslotIoxsAdd 54
5.4.49 Map APDU Status – goal_pnioDmApduAdd 55
5.4.50 Map Data Provider Status – goal_pnioDmDpAdd 55
5.5 Application Callbacks – PROFINET 56
5.5.1 Introduction 56
5.5.2 GOAL_PNIO_CB_ID_ALARM_ACK_TIMEOUT - Timeout Waiting for Alarm ACK 56
5.5.3 GOAL_PNIO_CB_ID_ALARM_NOTIFY_ACK - Alarm Notification ACK Received 57
5.5.4 GOAL_PNIO_CB_ID_ALARM_NOTIFY - Alarm Notification Received 57
5.5.5 GOAL_PNIO_CB_ID_APPL_READY - Application Ready Response Received 57
5.5.6 GOAL_PNIO_CB_ID_BLINK - Blink Request 58
5.5.7 GOAL_PNIO_CB_ID_CONNECT_FINISH - Connect Request Done 59
5.5.8 GOAL_PNIO_CB_ID_CONNECT_REQUEST - Connect Request 59
5.5.9 GOAL_PNIO_CB_ID_CONNECT_REQUEST_EXP_START- Expected Submodule Block Start 59
5.5.10 GOAL_PNIO_CB_ID_END_OF_PARAM - Param End Received 60
5.5.11 GOAL_PNIO_CB_ID_END_OF_PARAM_PLUG – Plug Param End Received 60
5.5.12 GOAL_PNIO_CB_ID_EXP_SUBMOD - Expected Submodule 60
5.5.13 GOAL_PNIO_CB_ID_FACTORY_RESET - Factory Reset 60
5.5.14 GOAL_PNIO_CB_ID_IO_DATA_TIMEOUT - Cyclic Timeout 60
5.5.15 GOAL_PNIO_CB_ID_NET_IP_SET - IP Configuration Update 61
5.5.16 GOAL_PNIO_CB_ID_NEW_AR - New Application Relation 61
5.5.17 GOAL_PNIO_CB_ID_NEW_IO_DATA - New IO Data 61

5.5.18 GOAL_PNIO_CB_ID_PLUG_READY - Plug Ready Response Received 61
5.5.19 GOAL_PNIO_CB_ID_READ_RECORD - Read Record Data 62
5.5.20 GOAL_PNIO_CB_ID_RELEASE_AR - Release Application Relation 63
5.5.21 GOAL_PNIO_CB_ID_RESET_TO_FACTORY - Reset To Factory 63
5.5.22 GOAL_PNIO_CB_ID_STATION_NAME - Station Name Changed 63
5.5.23 GOAL_PNIO_CB_ID_WRITE_RECORD - Write Record Data 64
5.5.24 GOAL_PNIO_CB_ID_INIT - Stack Initialized 65
5.5.25 GOAL_PNIO_CB_ID_LLDP_UPDATE - LLDP Update 65
5.5.26 GOAL_PNIO_CB_ID_CONN_REQ_EXP_FINISH - Connect Request Expected Submodule Block Finish 65
5.5.27 GOAL_PNIO_CB_ID_STATION_NAME_VERIFY - DCP Station Name Verification 65
5.5.28 GOAL_PNIO_CB_ID_NET_IP_SET_VERIFY - DCP IP Configuration Verification 66
5.6 Communication Stack – Ethernet IP 66
5.6.1 goal_eipCfgVendorIdSet 66
5.6.2 goal_eipCfgDeviceTypeSet 66
5.6.3 goal_eipCfgProductCodeSet 67
5.6.4 goal_eipCfgRevisionSet 67
5.6.5 goal_eipCfgSerialNumSet 67
5.6.6 goal_eipCfgProductNameSet 68
5.6.7 goal_eipCfgDomainNameSet 68
5.6.8 goal_eipCfgHostNameSet 68
5.6.9 goal_eipCfgNumExplicitConSet 69
5.6.10 goal_eipCfgNumImplicitConSetImpl 69
5.6.11 goal_eipCfgEthLinkCountersOn 69
5.6.12 goal_eipCfgEthLinkControlOn 70
5.6.13 goal_eipCfgChangeEthAfterResetOn 70
5.6.14 goal_eipCfgChangeIpAfterResetOn 70
5.6.15 goal_eipCfgNumSessionsSet 71
5.6.16 goal_eipCfgTickSet 71
5.6.17 goal_eipCfgO2TRunIdleHeaderOn 71
5.6.18 goal_eipCfgT2ORunIdleHeaderOn 71
5.6.19 goal_eipCfgQoSOn 72
5.6.20 goal_eipCfgNumDelayedEncapMsgSet 72
5.6.21 goal_eipCfgDhcpOn 72
5.6.22 goal_eipCfgDlrOn 73
5.7 Application Callbacks – EtherNet IP 74
5.7.1 Introduction 74
5.7.2 GOAL_EIP_CB_ID_INIT 75
5.7.3 GOAL_EIP_CB_ID_READY 75
5.7.4 GOAL_EIP_CB_ID_CONNECT_EVENT 75
5.7.5 GOAL_EIP_CB_ID_ASSEMBLY_DATA_RECV 75
5.7.6 GOAL_EIP_CB_ID_ASSEMBLY_DATA_SEND 76
5.7.7 GOAL_EIP_CB_ID_RUN_IDLE_CHANGED 76
5.7.8 GOAL_EIP_CB_ID_LED_CHANGED 76
5.7.9 GOAL_EIP_CB_ID_DEVICE_RESET 77

6. Application Programming Interface 78
6.1 Device Specific Functions 78
6.1.1 appl_ccmRpcInit 78
6.1.2 appl_ccmUpdateAllow 78
6.1.3 appl_ccmUpdateDeny 79
6.1.4 appl_ccmInfo 79
6.1.5 appl_ccmUpdateDeny 80

6.1.6 appl_ccmCommResetSet 80
6.1.7 appl_ccmLogEnable 81
6.2 Device Detection 82
6.2.1 goal_ddInit - Register GOAL dd API in GOAL (appl_init) 82
6.2.2 goal_ddNew - Register GOAL dd API in GOAL (appl_setup) 82
6.2.3 goal_ddCustomerIdSet - Configure the customer id of GOAL dd instance 83
6.2.4 goal_ddModuleNameSet - Configure the name of GOAL dd instance 84
6.2.5 goal_ddFeaturesSet - Configure the features of the GOAL dd instance 84
6.2.6 goal_ddCallbackReg - Configure callback for GOAL dd instance 85
6.2.7 goal_ddSessionFeatureActivate - Temporary activation of features of GOAL dd instance 86
6.2.8 goal_ddFilterAdd - Limit access to CM variables 87
6.2.9 Definition of Filter – GOAL_DD_ACCESS_FILTER_SET_ALL 88
6.2.10 Definition of Filter – GOAL_DD_ACCESS_FILTER_SET_BASIC 88
6.2.11 Definition of Filter – GOAL_DD_ACCESS_FILTER_SET_HIDDEN 88
6.3 PROFINET Stack Application Programming Interface 90
6.3.1 goal_pnioInit - Register GOAL PROFINET in GOAL (appl_init) 90
6.3.2 goal_pnioNew - Create a GOAL PROFINET Instance (appl_setup) 90
6.3.3 goal_pnioCfgDcpFactoryResetDisableSet - Configure DCP Factory Reset 91
6.3.4 goal_pnioCfgDcpAcceptMixcaseStationSet - Configure DCP Mixcase Stationname Acceptance 91
6.3.5 goal_pnioCfgDevDapSimpleSet - Configure Device DAP Simple Mode 91
6.3.6 goal_pnioCfgDevDapApiSet - Set Device DAP API Number 92
6.3.7 goal_pnioCfgDevDapSlotSet - Set Device DAP Slot Number 92
6.3.8 goal_pnioCfgDevDapSubslotSet - Set Device DAP Subslot Number 92
6.3.9 goal_pnioCfgDevDapModuleSet - Set Device DAP Module Id 93
6.3.10 goal_pnioCfgDevDapSubmoduleSet - Set Device DAP Submodule Id 93
6.3.11 goal_pnioCfgNetLinkSafetySet - Configure Device Port Disable Behavior 93
6.3.12 goal_pnioCfgNewIoDataCbSet - Configure New IO Data Callback 94
6.3.13 goal_pnioCfgDiagBufMaxCntSet - Configure Maximum Diagnosis Entries 94
6.3.14 goal_pnioCfgDiagBufMaxDataSizeSet - Configure Maximum Diagnosis Data Size 95
6.3.15 goal_pnioCfgIocrBlocksMaxSet – Configure Maximum IOCR Block Buffers 95
6.3.16 goal_pnioCfgCrMaxCntSet - Configure Maximum Communication Relation Count 95
6.3.17 goal_pnioCfgArMaxCntSet - Configure Maximum Application Relation Count 95
6.3.18 goal_pnioCfgApiMaxCntSet - Configure Maximum API Count 96
6.3.19 goal_pnioCfgSlotMaxCntSet - Configure Maximum Slot Count 96
6.3.20 goal_pnioCfgSubslotMaxCntSet - Configure Maximum Subslot Count 97
6.3.21 goal_pnioCfgSubslotIfSet - Configure Interface Subslot 97
6.3.22 goal_pnioCfgSubslotPortSet - Configure Port Subslot 97
6.3.23 goal_pnioCfgSnmpIdSet - Configure SNMP Instance Id 98
6.3.24 goal_pnioSlotNew - Create a new slot 98
6.3.25 goal_pnioSubslotNew - Create a new subslot 98
6.3.26 goal_pnioModNew - Create a new module 99
6.3.27 goal_pnioSubmodNew - Create a new submodule 99
6.3.28 goal_pnioModPlug - Plug a module into a slot 99
6.3.29 goal_pnioSubmodPlug - Plug a submodule into a subslot 100
6.3.30 goal_pnioModPull - Pull a module from a slot 100
6.3.31 goal_pnioSubmodPull - Pull a submodule from a subslot 101
6.3.32 goal_pnioDataOutputGet - Get output data from a submodule 101
6.3.33 goal_pnioDataInputSet - Set input data for a submodule 101
6.3.34 goal_pnioApduStatusGet - Get the application protocol data unit status 102
6.3.35 goal_pnioAlarmNotifySend - Send an alarm notification 102
6.3.36 goal_pnioAlarmNotifySendAck - Send an alarm notification acknowledge 103

6.3.37 goal_pnioAlarmProcessSend - Send a process alarm 103
6.3.38 goal_pnioRecReadFinish - Answer a record read request 103
6.3.39 goal_pnioRecWriteFinish - Answer a record write request 104
6.3.40 goal_pnioDiagExtChanDiagAdd - Add an extended channel diagnosis entry 104
6.3.41 goal_pnioDiagChanDiagRemove - Remove a channel diagnosis entry 104
6.3.42 goal_pnioCyclicCtrl - Control cyclic data received callback 105
6.4 EtherNet/IP Application Programming Interface 106
6.4.1 Introduction 106
6.4.2 goal_eipInit 106
6.4.3 goal_eipNew 107
6.4.4 goal_eipCipClassRegister 107
6.4.5 goal_eipCreateAssemblyObject 107
6.4.6 goal_eipAssemblyObjectGet 108
6.4.7 goal_eipAddExclusiveOwnerConnection 108
6.4.8 goal_eipAddInputOnlyConnection 108
6.4.9 goal_eipAddListenOnlyConnection 109
6.4.10 goal_eipGetVersion 109
6.4.11 goal_eipDeviceStatusSet 109
6.4.12 goal_eipDeviceStatusClear 110
6.5 Networking 111
6.5.1 goal_netRpcInit - Initialize RPC functionality for networking 111
6.5.2 goal_maNetOpen - Open network 111
6.5.3 goal_maNetClose - Close network 112
6.5.4 goal_maNetGetById - Get network media adapter (MA) handle 112
6.5.5 goal_maNetIpSet – Set ip address 112
6.6 TCP Channel 114
6.6.1 goal_maChanTcpOpen - Open the TCP channel media adapter (MA) 114
6.6.2 goal_maChanTcpNew - Create a new TCP channel 114
6.6.3 goal_maChanTcpActive - Activate a created TCP channel 114
6.6.4 goal_maChanTcpSetNonBlocking - Set channel to non-blocking 115
6.6.5 goal_maChanTcpGetRemoteAddr - Get remote address of TCP channel 115
6.6.6 goal_maChanTcpSend - Send data through TCP channel 116
6.7 UDP Channel 117
6.7.1 goal_maChanUdpOpen - Open the UDP channel MA 117
6.7.2 goal_maChanUdpGetById - Get the UDP channel MA handle 117
6.7.3 goal_maChanUdpNew - Create a new UDP channel 117
6.7.4 goal_maChanUdpClose - Close the UDP channel MA 118
6.7.5 goal_maChanUdpSetNonBlocking - Set the opened channel to non-blocking access 118
6.7.6 goal_maChanUdpSetBroadcast - Set the opened UDP channel to broadcast operation 118
6.7.7 goal_maChanUdpGetRemoteAddr - Get remote address of the UDP channel 119
6.7.8 goal_maChanUdpActivate - Activate a UDP channel 119
6.7.9 goal_maChanUdpSend - Send data to the UDP channel 119

7. Examples 120
7.1 01_pnio_io_mirror 120
7.1.1 Purpose 120
7.1.2 Configuration 120
7.1.3 Usage Hints 120
7.2 02_eip_io_data 121
7.2.1 Purpose 121
7.2.2 Configuration 121
7.2.3 Usage Hints 121
7.3 05_pnio_01_simple_io 122

7.3.1 Purpose 122
7.3.2 Configuration 122
7.3.3 Usage Hints 122
7.4 06_eip_io_data_static_ip 123
7.4.1 Configuration 123
7.4.2 Usage Hints 123
7.5 07_pnio_dsn 124
7.5.1 Purpose 124
7.5.2 Configuration 124
7.5.3 Usage Hints 124
7.6 http_01_get 125
7.6.1 Purpose 125
7.6.2 Configuration 125
7.6.3 Usage Hints 125
7.7 http_02_post 125
7.7.1 Purpose 125
7.7.2 Configuration 125
7.7.3 Usage Hints 125
7.8 http_03_list_res 126
7.8.1 Purpose 126
7.8.2 Configuration 126
7.8.3 Usage Hints 126
7.9 http_03_list_res 127
7.9.1 Purpose 127
7.9.2 Configuration 127
7.9.3 Usage Hints 127
7.10 http_05_template_cm 128
7.10.1 Purpose 128
7.10.2 Configuration 128
7.10.3 Usage Hints 128
7.11 http_06_template_list 128
7.11.1 Configuration 128
7.11.2 Usage Hints 128
7.12 http_07_template_table 129
7.12.1 Purpose 129
7.12.2 Configuration 129
7.12.3 Usage Hints 129
7.13 net_01_udp_receive 130
7.13.1 Purpose 130
7.13.2 Configuration 130
7.13.3 Usage Hints 130
7.14 net_02_tcp_client 130
7.14.1 Purpose 130
7.14.2 Configuration 130
7.14.3 Usage Hints 130
7.15 net_03_tcp_server 131
7.15.1 Purpose 131
7.15.2 Configuration 131
7.15.3 Usage Hints 131

8. Trouble Shooting 132
8.1 Startup Issues 132
8.2 Connection Issues 132

8.3 IP Configuration 133
8.4 Downgrade to Version 1.0 133

9. Renesas Synergy™ 134
9.1 Development Environment 134
9.1.1 Support Hardware 134
9.2 Adaption for Customer Hardware 134
9.2.1 SPI Configuration 134
9.2.2 LED Control 136
9.2.3 Timer 137
9.2.4 ThreadX 137
9.3 Logging 138
9.4 Object Dictionary 139

10. Webserver 143
10.1 General 143
10.2 Configuration 144
10.2.1 Compiler-defines 144
10.2.2 CM-variables 144
10.3 Web-templates 149
10.3.1 CM-variables 149
10.3.2 Application-specific variables 149
10.3.3 Lists 149
10.4 Characters 151
10.5 Callback Functions 152
10.6 Implementation Guideline 152
10.6.1 Upload a webpage 152
10.6.2 Upload a webpage 153
10.6.3 Read application specific variable 154
10.6.4 Read a list 156
10.6.5 Set a user level 158
10.6.6 Download files 159

1. Introduction

This manual describes the various application programming interface (API) functions which have been developed to ease the design of application software for the R-IN32M3 module, which is available as a dual-port device.
The R-IN32M3 module has been designed for the fast development of industrial communications applications. The R-IN32M3 module includes a microcontroller which acts as a communications controller (CC). The CC with its firmware (binary software) executes the network communication with industrial communication standards such as EtherNet/IP®, PROFINET® and other networks. The application controller (AC) is a user defined microcontroller. This AC with a set of software modules runs the application specific software without the need to take care for the specific requirements of the communication protocol of the targeted industrial network. It communicates with the CC using a set of APIs which are supplied by Renesas in source code for an easy integration with the user's application software. Communication between CC and AC uses the protocol "Core to Core" (C2C). The following diagram illustrates the controllers and their interfaces.

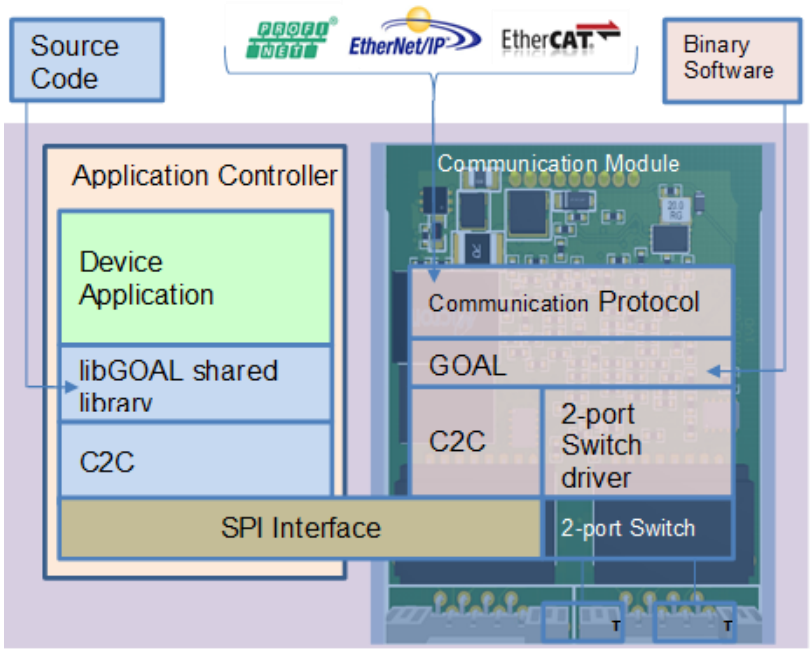

Figure 1.1 Interfaces between Communication Controller and Application Controller

GOAL is a Generic Open Abstraction Layer and libGOAL the source coded API to control the communication controller from the application software. Both components GOAL and C2C are combined under the term of OSAL.

Note: Some of the industrial networking protocols mentioned in the figure above may not yet be released.

2. Document Structure

Table 2.1 Contents of This Document

Section	Contents
Introduction	Introduction of this document
Document Structure	This section
Device architecture	Introduction of the R-IN32M3 module architecture
Application	Guideline for application programming
Communication stack	Description of the communication interface of the module
API	Listing and explaination of the available APIs of the R-IN32M3 module
Examples	Description of the examples
Trouble Shooting	List of common usage problems
Targets	Target specific information

3. Device Architecture

3.1 Architecture

The module software contains a domain specific middleware (GOAL) with several software stacks that can be utilized to build applications in the domain of industrial communication. The following software stacks are provided:

- Device Detection : A simple protocol for device management
- PROFINET : A communication stack for PROFINET communication
- EtherNet/IP : A communication stack for EtherNet/IP communication
- Webserver : A simple web server for application specific management and information provision
- TCP/IP Stack : A TCP/IP stack which provides UDP and TCP channels

The device is utilized by an application controller (AC). Both controllers (application and communication controller) communicate cyclically, where the communication is dictated by the application controller. The application controller (AC) contains the application which runs services of the module to build a customer application.

3.2 Interface

3.2.1 Hardware Interface

The R-IN32M3 module pins interface with power supply and SPI, which is a slave interface.

Table 3.1 Pin Description

Pin	Signal	I/O	Description
1	V_{cc}	—	3.3V ±0.15V DC power supply
2	GND	—	Ground
3	/SS	I	Slave Select. Active low to enable slave device
4	/RESET	I	Reset of the whole R-IN32M3 module. Active low
5	MISO	O	Master In Slave Out. Data from slave to master
6	MOSI	I	Master Out Slave In. Data from master to slave
7	SCLK	I	Serial clock. Master provides the clock to shift the data
8	SYNC0	O	EtherCAT Sync signal for distributed clocks
9	SYNC1	O	EtherCAT Sync signal for distributed clocks

Note. EtherCAT® will be supported in a future software update. EtherCAT Sync signals are used only for EtherCAT protocol.

3.2.2 SPI Software Interface

The software interface of the R-IN32M3 module contains various layers, which provide communication channels according to the requirements of the communication data. Over SPI a frame of up to 128 bytes of data is cyclically transferred, where the communication is initiated by the AC.

The SPI frame contains multiple segments of data, typically real time data from communication stacks and non-real-time data, which originates from RPC (Remote Procedure Call). A detailed description of the communication stack is given in the following sections.

3.2.3 Integration of Communication Layers / Middleware

All layers of the required communication protocol and the AC applications are implemented using the GOAL middleware. The AC application requires to be based on an implementation of this middleware. To utilize a certain feature set of the CC module (as fieldbus stack PROFINET), wrapper functionality is required that implements the RPC functions. These wrappers require GOAL, thus the target platform must support the GOAL middleware.

4. Application

4.1 Introduction

This document provides information about how to build an application for the R-IN32M3 module.

4.2 Hardware Setup

The module provides a management interface, where initial configuration can be done. These properties can be stored permanently within the device. Following properties are provided to configure the SPI interface:

Table 4.1 SPI Configuration

Variable	Description	Default Value
SPI_TYPE	SPI master/slave configuration	1 = SLAVE
SPI_MODE	SPI timing mode	0 = MODE0
SPI_UNITWIDTH	SPI single transfer size	0 = 8 bits
SPI_BITORDER	SPI bit transfer direction	0 = MSB
SPI_TRANSFERSIZE	SPI packet transfer size	128

The module does only support
SPI_TYPE slave,
SPI_UNITWIDTH of 8 bits,
SPI_TRANSFERSIZE of 128 bytes,
SPI bitorder MSB.

The SPI mode can be configured according to Table 4.2, SPI Mode Configuration.

Table 4.2 SPI Mode Configuration

Value	SPI MODE
0	Mode 0
1	Mode 1
2	Mode 2
3	Mode 3

4.3 Basic Application Setup

A basic application consists of the optional function calls appl_init, appl_setup and appl_loop. It follows the design philosophy of the GOAL middleware.

4.4 Default Features

By default, the CC module will start a web server (on port 8080) for the firmware update feature and an instance of Device Detection for management using the management tool.

It is not possible by an AC application to disable the web server on the CC. However, the AC can start another instance of the web server.

It is possible for an AC application to limit by default full management access through Device Detection in different ways. Please refer to the corresponding section in the documentation.

4.5 Features

4.5.1 Device Detection

4.5.1.1 Initial state

On the communication module the feature “Device Detection“(DD) is enabled by default. This allows full access to all variables and logs on the CC module. DD bases on a simple UDP based protocol with no encryption whatsoever.

Thus, an application can and should limit the access to a feasible range. Limitation of features and access is possible on various levels.

4.5.1.2 Initial disabling of features by application

This property is applied at application start-up, when the AC application setup is executed.

With the call of goal_ddNew() from appl_setup() a bitmask can be passed with bits representing single functions of DD. Please refer to the API documentation of DD for possible values.

If all bits are set, all features (hello, get, set, get_list, wink) are enabled.

4.5.1.3 Initial disabling of features by CM variable

The mechanism (described in section 4.5.1.2) can also be set before application start by setting the corresponding Configuration Manager (CM) variable FEATURE_DISABLE (see section 4.9.1). Each request processed by the device detection module will utilize the value of this variable to determine, if the request can be processed.

Please note, that any call of goal_ddNew will overwrite the value of this variable, if a different initialization value is passed.

4.5.1.4 Temporary enable features of the device detection

This property is applied at an arbitrary time, for example when configured using a web site provided by the AC application.

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

- The parameter bitmask contains bits representing features being enabled
- Those bits overwrite the disable bits from CM for the current session

4.5.1.5 Filtering and access rights

This property is applied at start-up of the AC application.

Using filters, it is possible to limit access to CM variables by groups (module ids) and variables (variable ids). Currently some filters are predefined for usage. These can be enabled using following function:

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

This function works on a created instance of DD, thus requires passing of a valid DD handle. The **setId** can be used with following values:

Table 4.3 DD Filter IDs

Set UD	Description
GOAL_DD_ACCESS_FILTER_SET_ALL	Complete access to all variables
GOAL_DD_ACCESS_FILTER_SET_BASIC	Filter for minimal function of the management tool
GOAL_DD_ACCESS_FILTER_SET_HIDDEN	Filter for hiding information

Note: Please refer to the API documentation of DD for detailed information.

4.5.2 PNIO

The communication module contains a full featured PROFINET slave stack. For application see the PROFINET documentation

4.5.3 EtherNet/IP

The communication module contains a full featured EtherNet/IP slave stack. For application see the proper EtherNet/IP documentation.

4.5.4 HTTPD

The communication module contains a web server for basic management functionality. Please refer to the http documentation.

4.5.5 Network Channels

The communication module provides TCP and UDP communication channels. Please refer to the API documentation within this document for detailed description.

4.5.6 Generic Data Provider

For fieldbus communication applications a generic data provider is available, which contains generalized information and LED status for the application.

Table 4.4 MCTC Status Information

Data Provider Information	PROFINET	EtherNet/IP
GOAL_MCTC_DP_STATUS_FLG_CONN	Connection	Connection
GOAL_MCTC_DP_STATUS_FLG_ERR	Error	Error
GOAL_MCTC_DP_STATUS_FLG_VALID	-	Process data valid
GOAL_MCTC_DP_LED_WINK	DCP blink signaling	-
GOAL_MCTC_DP_LED_RED1	Error	Module Status red
GOAL_MCTC_DP_LED_RED2	Maintainance	Network Status red
GOAL_MCTC_DP_LED_GREEN1	Connection	Module Status green
GOAL_MCTC_DP_LED_GREEN2	DCP blink signaling	Network Status green

4.6 External Reset

The module provides an external reset input, where the AC can perform a reset of the R-IN32M3 module.

Caution:

Do not perform this reset during a firmware update of the module. This will prevent proper update functionality. Therefore, it is recommended to utilize this reset only if the AC module is initially powered up (cold start).

4.7 RPC Synchronization Reset

4.7.1 Communication Controller (CC)

By default, the module (CC) does not perform a reset implicitly.
The CC requests the AC to perform a power cycle if necessary. This occurs if a previously booted and configured AC application is connected to a newly started CC module.

4.7.2 Application Controller (AC)

The AC performs a reset of its application on request of the CC, e.g. when the CC restarts. This can happen during a firmware update.
The AC can perform a reset of the CC module by using the hardware RESET signal.

Caution:
Do not perform this reset during a firmware update of the module. This will prevent proper update functionality.

4.8 IP Setting

IP settings can be done with the CM interface.

Table 4.5 IP Configuration with CM

Step	Action	Remark
1	Set CM Variable GOAL_ID_NET:IP	Configure IP address
2	Set CM Variable GOAL_ID_NET:NETMASK	Configure Netmask
3	Set CM Variable GOAL_ID_NET:GW	Configure Gateway
4	Set CM Variable GOAL_ID_NET:Valid to 1	Set IP configuration to valid
5	Set CM Variable GOAL_ID_NET:DHCP_ENABLED to 0	Disable DHCP
6	Set CM Variable GOAL_ID_NET:COMMIT to 1	Apply IP settings

Note: To enable DHCP set the CM Variable GOAL_ID_NET:DHCP_ENABLED to 1 and perform a power cycle.

4.9 Management Interface CC

See goal_db, Page “Variables”: The column “Long description” contains documentation of single variables.
The management interface has following functions:

- Responding to scan requests for devices
- Listing CM variables
- Reading CM variables
- Writing CM variables
- Setting IP address
- Wink command

Each of those functions can be permanently disabled by setting a bit in the CM variable GOAL_ID_DD: FEATURE_DISABLE.

4.9.1 Management Interface DD

Module Id = GOAL_ID_DD (34)

Table 4.6 DD Management Interface

Variable Name	Variable ID	Type	Max. Size	Long Description
MODULENAME	0	GOAL_CM_STRING	20	Customer specific name of the module
CUSTOMERID	1	GOAL_CM_UINT32	4	Customer Id
RESERVED	2	GOAL_CM_UINT8	1	-
FEATURE_DISABLE	3	GOAL_CM_UINT32	4	Each bit disables a function: bit 0, disable "HELLO DETECTION" bit 1, disable WINK bit 2, disable GETLIST bit 3, disable GET VALUE bit 4, disable SET VALUE

4.9.2 PNIO

Module Id = GOAL_ID_PNIO (27)
• Internal variables used by the PROFINET stack

4.9.3 Logging Manager (LM)

These variables provide an interface for the logging manager of the R-IN32M3 module.
Module Id = GOAL_ID_LM (35)

Table 4.7 LM Management Interface

Variable Name	Variable ID	Type	Max. Size	Long Description
VERSION	0	GOAL_CM_UINT8	1	Version information for LM interface
READBUFFER	1000	GOAL_CM_GENERIC	128	Buffer for reading online logging from deivce
CNT	1001	GOAL_CM_UINT16	2	Control word for online log access
EXLOG_READBUFFER	1002	GOAL_CM_GENERIC	128	Buffer for reading exception logging from deivce
EXLOG_CNT	1003	GOAL_CM_UINT16	2	Control word for exception log access
EXLOG_SIZE	1004	GOAL_CM_UINT32	4	Indicator for exception log size
EXLOG_USAGE	1005	GOAL_CM_UINT8	1	Indicator for exception log usage
EXLOG_ERASE	1006	GOAL_CM_UINT8	1	Command: *, Erase Exception Log

4.9.4 Net

Module Id = GOAL_ID_NET (12)

Table 4.8 NET Management Interface

Variable Name	Variable ID	Type	Max. Size	Long Description
IP	0	GOAL_CM_IPV4	4	IP address of first interface
NETMASK	1	GOAL_CM_IPV4	4	NETMASK of first interface
GW	2	GOAL_CM_IPV4	4	GATEWAY of first interface
VALID	3	GOAL_CM_UINT8	1	Validity of IP address: 0, Stored IP address is not valid, interface settings originate from network stack of system 1, Stored IP address is valid, will be applied to interface at start of device
DHCP_ENABLED	4	GOAL_CM_UINT8	1	DHCP enable: 0, DHCP disabled 1, DHCP enabled
DHCP_STATE	5	GOAL_CM_UINT8	1	DHCP state: 0, DHCP initialized 1, DHCP server selecting 2, DHCP requesting configuration 3, DHCP ip address bound 4, DHCP renewing configuration 5, DHCP rebinding ip address to interface
DNS0	6	GOAL_CM_IPV4	4	First DNS server of first interface
DNS1	7	GOAL_CM_IPV4	4	Second DNS server of first interface
HOSTNAME	8	GOAL_CM_STRING	20	Hostname of first interface
COMMIT	1000	GOAL_CM_UINT8	1	Command: *, Apply IP settings

4.9.5 BOOT

Module Id = GOAL_ID_BOOT (37)

Table 4.9 BOOTLOADER Management Interface

Variable Name	Variable ID	Type	Max. Size	Long Description
SIGNATURE	0	GOAL_CM_GENERIC	16	Signature of booted image
BLVERSION	1	GOAL_CM_STRING	16	Bootloader Version
FWVERSION	2	GOAL_CM_STRING	16	Firmware Version
RESET_CAUSE	1000	GOAL_CM_UINT8	1	Reset cause: 0, Unspecified 1, Firmware Update Requested 2, Watchdog 3, Firmware Commit Required 4, Reserved
IMAGE_NUMBER	1001	GOAL_CM_UINT8	1	Booted image number
IMAGE_COUNTER	1002	GOAL_CM_UINT8	1	Booted image counter

4.9.6 Config Manager (CM)

Module Id = GOAL_ID_CM (2)

Table 4.10 CM Management Interface

Variable Name	Variable ID	Type	Max. Size	Long Description
VERSION	0	GOAL_CM_UINT32	4	Version information for CM interface
SAVE	1000	GOAL_CM_UINT8	1	Command: *, Save CM to Flash

4.9.7 ETH

Module Id = GOAL_ID_ETH (4)

Table 4.11 ETH Management Interface

Variable Name	Variable ID	Type	Max. Size	Long Description
MAC	0	GOAL_CM_GENERIC	6	—
LINK	1000	GOAL_CM_UINT32	4	Link status mask of interfaces
SPEED	1001	GOAL_CM_UINT32	4	Port speed mask of interfaces
DUPLEX	1002	GOAL_CM_UINT32	4	Port Duplex mask of interfaces
PORTCNT	1003	GOAL_CM_UINT32	4	Number of interfaces

4.9.8 EIP

Module Id = GOAL_ID_EIP (23)
• Internal variables used by the EtherNet/IP stack

4.9.9 HTTPD

Module Id = GOAL_ID_HTTP (25)

Table 4.12 HTTP Management Interface

Variable Name	Variable ID	Type	Max. Size	Long Description
HTTP_CHANNELS_MAX	0	GOAL_CM_UINT16	2	Determines the number of possible connections to the HTTP server
HTTPS_CHANNELS_MAX	1	GOAL_CM_UINT16	2	Determines the number of possible connections to the HTTPS server
USERLEVEL0	2	GOAL_CM_STRING	32	Authentication data for level 0
USERLEVEL1	3	GOAL_CM_STRING	32	Authentication data for level 1
USERLEVEL2	4	GOAL_CM_STRING	32	Authentication data for level 2
USERLEVEL3	5	GOAL_CM_STRING	32	Authentication data for level 3

4.9.10 CCM

Interface for Management Tool for informative and configuration purpose.
Module Id = GOAL_ID_CCM (72)

Table 4.13 R-IN32M3 Module Management Interface

Variable Name	Variable ID	Type	Max. Size	Long Description
SPI_TYPE	0	GOAL_CM_UINT8	1	SPI Type (currently only slave supported): 0, SPI Master 1, SPI Slave
SPI_MODE	1	GOAL_CM_UINT8	1	SPI Mode: 0, CPOL=0; CPHA=0 1, CPOL=0; CPHA=1 2, CPOL=1; CPHA=0 3, CPOL=1; CPHA=1

SPI_SPEED	2	GOAL_CM_UINT8	1	SPI Speed in Master Mode
SPI_UNITWIDTH	3	GOAL_CM_UINT8	1	Bitsize of one single transfer unit: 0, 8 bits 1, 16 bits 2, 32 bits
SPI_BITORDER	4	GOAL_CM_UINT8	1	Bitorder of SPI transfers: 0, MSB first 1, LSB first
SPI_TRANSFERSIZE	5	GOAL_CM_UINT16	2	Minimum transfer size of single transmission frame
COMM_FAULT_ERROR_STATE	6	GOAL_CM_UINT8	1	Fault action to execute when communicatio n to AC was lost during a cyclic connection: 0, Enter fault state (disable connection) 1, Keep running (keep connection)
COMM_SYNC_RESET	7	GOAL_CM_UINT8	1	Behavior when a sync reset request was received from AC: 0, Do nothing 1, Perform reset of CC controller
FW_UPDATE_COMMIT_DISABLE	8	GOAL_CM_UINT8	1	Optional disable of the additional commit step during firmware update: 0, Firmware update requires commit step 1, Firmware update doesn't require a commit step
UPTIME	1000	GOAL_CM_UINT32	4	Number of seconds since

				start of device

4.10 Firmware Update

4.10.1 Update the Communication Controller

Firmware update of the communication controller is possible in the field. It is done using the management tool. This tool uses the http protocol to transfer a firmware package to the device.

4.10.1.1 Firmware package

Firmware image is bundled within a package. This package contains a signed firmware, which secures only acceptance of firmware from the device originator. Thus, it is possible for the device to check authenticity of the firmware image.

4.10.1.2 Control interface

By default, a firmware update is possible at any time. The communication module provides an interface to enable and disable the firmware update process. An application should use this interface as there are situations where a firmware update should not be accepted. This should be deactivated during an active cyclic connection to the PLC.
For usage of this interface see section 6.1, Device Specific Functions.

4.10.1.3 Firmware update sequence

Firmware update is a two-step process. At first the firmware is uploaded to the http server of the CC module. Following checks are done during this check:

1. This upload uses authentication with authentication level 0, where credentials are checked if configured for the HTTPD module. Those variables are configurable through the RPC interface of the HTTPD service. By default, these credentials are empty. Thus, any attempt to authenticate is accepted.

2. The firmware update must be allowed by the AC. By default, this is the case. However, the AC can disable this function using the RPC interface.

If at the end of the transfer a valid firmware is detected, the module restarts and enters the bootloader. This software module checks the signature of the firmware and writes the correctly signed firmware to the memory of the device. As a fallback, the previously run firmware is kept. After restart of the module a successful communication to the management tool is required to permanently enable the updated firmware. If this is possible, the module will restart again, and the bootloader will mark the new firmware as the current firmware. If this fails, the bootloader will revert to the original firmware with the next power cycle.

4.10.1.4 Keep update functionality while disabling DD

An application integrator might want to disable the DD feature to not expose any internal information of the device. Disabling is possible. However, to allow a firmware update using the management tool, the following steps need to be implemented:

1. Disabling DD by default

When calling the API function goal_ddNew, a bitmask can be passed, that determines the active DD features. If the predefined value GOAL_DD_FEAAT_NO is used, DD is disabled completely. Internally this value is stored to the CM variable FEATURE_DISABLE, which state also can be saved permanently to flash.

2. Introducing a switch to temporarily enable DD features

If a firmware update shall be processed, minimal DD features need to be set temporarily. These are:

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

This can be done using the API function goal_ddSessionFeatureActivate() with the following arguments:

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

The temporary switch, to enable the management interface, can be implemented using the web server.

As a result, the device will be invisible for the management tool. For firmware update, the required interface is temporarily activated, thus allowing to update the communication module firmware.

4.10.2 Update Possibilities for Application Controller

By default, there is no firmware update available for the application controller. However, the module provides mechanisms for implementing such a feature within the customer application. Following, two possible solutions are shown in next sections.

4.10.2.1 AC firmware update over HTTP

The application controller can provide a web site or tool-based firmware update over HTTP transport, utilising the provided RPC interface of the HTTPD service. As a starting point, the HTTPD example goal_http/02_post can be used on the AC and the example program 01_pnio_io_mirror can be used on the R-IN32M3 module.

For latter, data for firmware update transported using a HTTP POST request will be processed in the call-back function httpDataCbPost, for example as shown in the following sample code:

```
1.  /*****************************************************************************/
2.  /** goal http data callback
3.   *
4.   */
5.  static GOAL_STATUS_T httpDataCbPost(
6.      GOAL_HTTP_APPLCB_DATA_T *pCbInfo          /**< pointer to callback info struct */
7.  )
8.  {
9.      GOAL_STATUS_T res = GOAL_OK;              /* result */
10.     static uint32_t uploadDataLen = 0;        /* recent uploaded data length */
11.
12.     if (hdlUplHtml == pCbInfo->hdlRes) {
```

```
13.       /* check request method */
14.       switch (pCbInfo->reqType)
15.       {
16.       case GOAL_HTTP_FW_POST_START:
17.          /* reset data length */
18.          uploadDataLen = 0;
19.          GOAL_HTTP_RETURN_OK_204(pCbInfo);
20.          break;
21.
22.       case GOAL_HTTP_FW_POST_DATA:
23.
24.          /* process data */
25.          GOAL_MEMCPY(pDst, pCbInfo->cs.pData, pCbInfo->cs.lenData);
26.          uploadDataLen += pCbInfo->cs.lenData;
27.
28.          GOAL_HTTP_RETURN_OK_204(pCbInfo);
29.          break;
30.
31.       case GOAL_HTTP_FW_POST_END:
32.          GOAL_HTTP_RETURN_OK_204(pCbInfo);
33.          break;
34.
35.       case GOAL_HTTP_FW_REQ_DONE_OK:
36.       case GOAL_HTTP_FW_REQ_DONE_ERR:
37.          res = GOAL_OK;
38.          break;
39.
40.       default:
41.          /* return error */
42.          GOAL_HTTP_RETURN_ERR_403(pCbInfo);
43.          break;
44.       }
45.    }
46.    else {
47.       /* return error */
48.       GOAL_HTTP_RETURN_ERR_404(pCbInfo);
49.    }
50.    return res;
51.}
```

4.10.2.2 AC firmware update over TCP

```
1.  /*****************************************************************************/
2.  /** TCP Server Callback
3.   *
4.   * Process TCP data stream segments
5.   */
6.  static void tcpCallback(
7.     GOAL_MA_CHAN_TCP_T *pMaTcpHdl,          /**< MA handle */
8.     GOAL_NET_CB_TYPE_T cbType,              /**< callback type */
9.     struct GOAL_NET_CHAN_T *pChan,          /**< channel descriptor */
10.    struct GOAL_BUFFER_T *pBuf              /**< GOAL buffer */
11.)
12.{
13.    GOAL_NET_ADDR_T remote;                 /* remote address */
14.    GOAL_STATUS_T res;                      /* result */
15.
16.    if (cbType == GOAL_NET_CB_NEW_SOCKET) {
17.       goal_logInfo("new TCP listener");
18.    }
19.    else if (cbType == GOAL_NET_CB_NEW_DATA) {
```

```
20.
21.         /* process data */
22.
23.         goal_logDbg("Data received on tcp socket %p", (void *) pChan);
24.     }
25.}
```

5. Communication Stack

5.1 Introduction

This section lists all functions that are used to configure the communication stacks. These functions must be called within appl_setup() before goal_New() was called. Otherwise these functions have no effect. Each configuration has a default value that will be applied if no other value was set.

5.2 SPI Data Exchange

The communication with the R-IN32M3 module uses the well-known Serial Peripheral Interface (SPI). A transfer must always be triggered by the application core (AC) and must at least be once during the heartbeat timeout. The implementation in the R-IN32M3 module updates the SPI data after each transfer to make sure it doesn't interrupt a running transfer.

5.2.1 Clock Domains and Communication Cycle

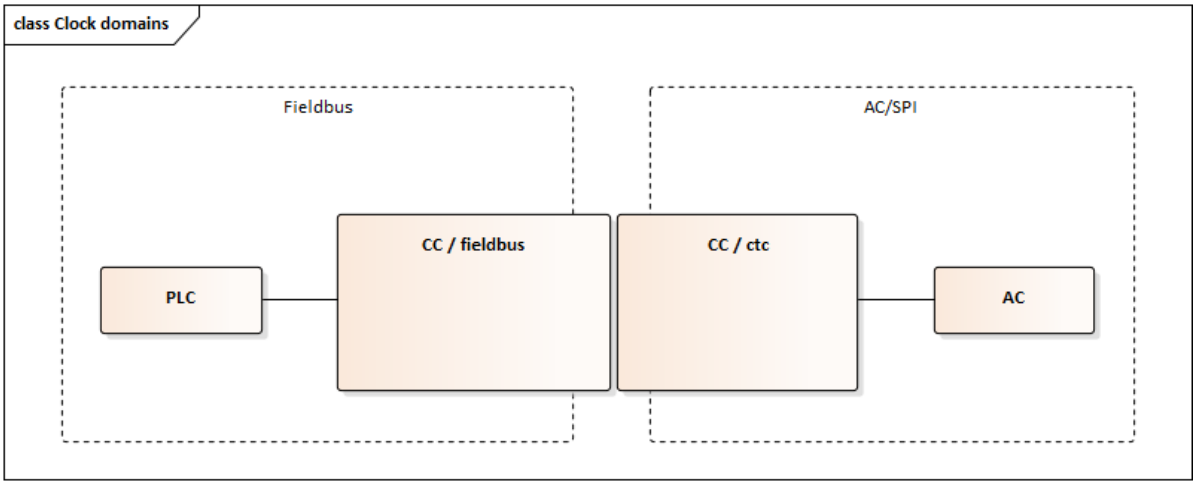

Figure 5.1 ctc Clock Domains

Operation of the device includes two clock domains, which run independently of each other. The first clock domain is on the fieldbus side. Commonly the PLC interacts with the device in one clock domain, where the PLC controls the timing of the output data. The second clock domain is driven by the AC which, through initiation of the SPI cycle, reads the output data from the device and updates input data for the next fieldbus cycle. Therefore, a specific timing of the process data is set up as shown in Figure 5.2, Communication Cycle.

Following data transports occur:

1. New output data from PLC
2. Processing of output data in CC module and preloading of SPI transfer buffer
3. A finished SPI transfer initiated by the AC executes data exchange between AC and CC
4. Preloading of new input data for the next SPI transfer
5. A finished sequential SPI transfer executes data exchange between AC and CC, thus providing new input data for the next fieldbus transfer

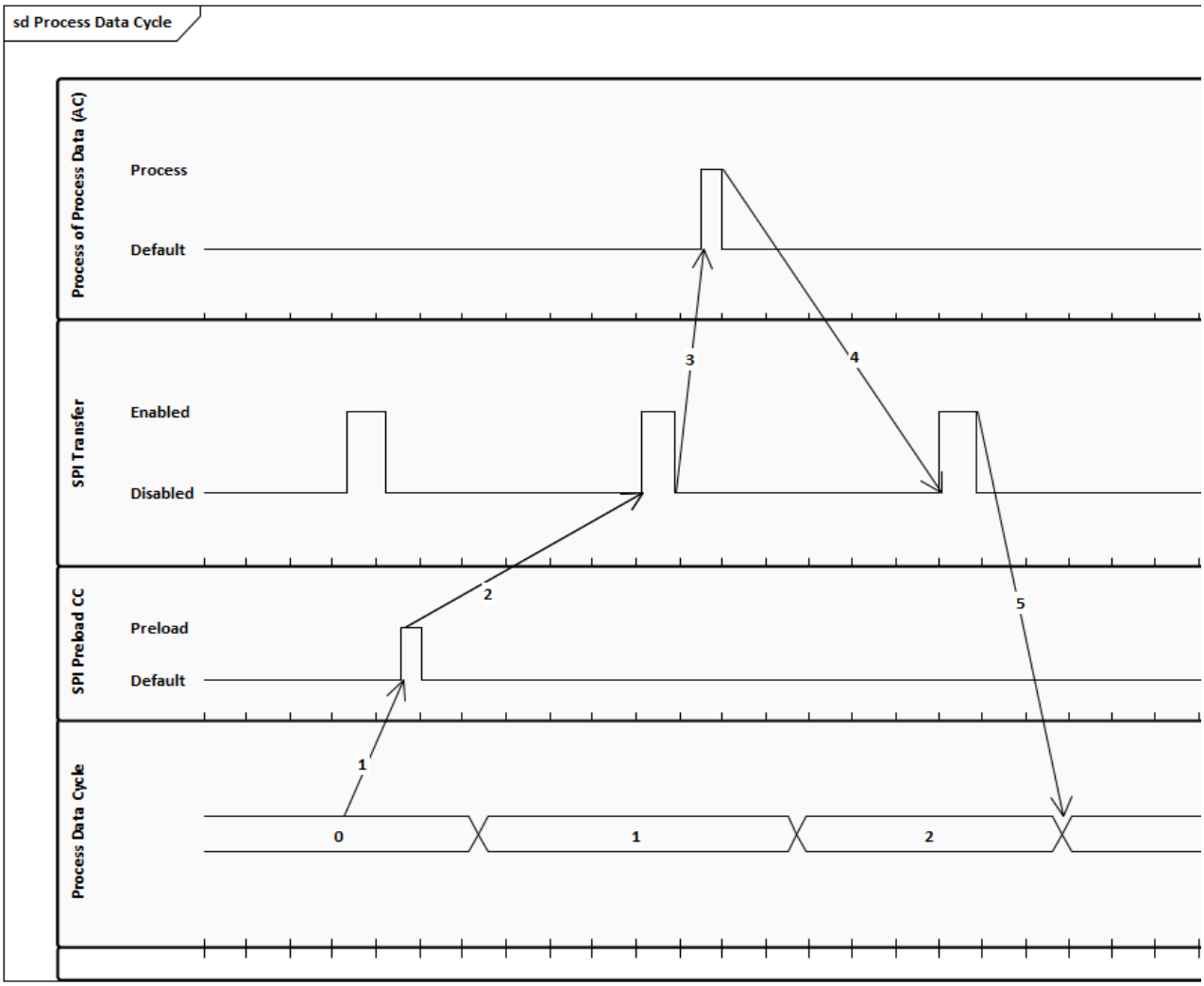

Figure 5.2 Communication Cycle

5.2.2 Technical Data

- Transfer length:
  - Cyclic only: 72 Bytes
  - Cyclic + RPC data: 128 Bytes
- Baud-rate: min, ..., max
- Delay between SPI transfers: 0.5 ms ... Heartbeat Timeout (1 second)
- Minimal round-trip time: 4 ... 6 ms (depending on the used protocol and setup)

5.2.3 SPI Timing

Following simplified diagram shows basic SPI timing which must be considered by the application controller.

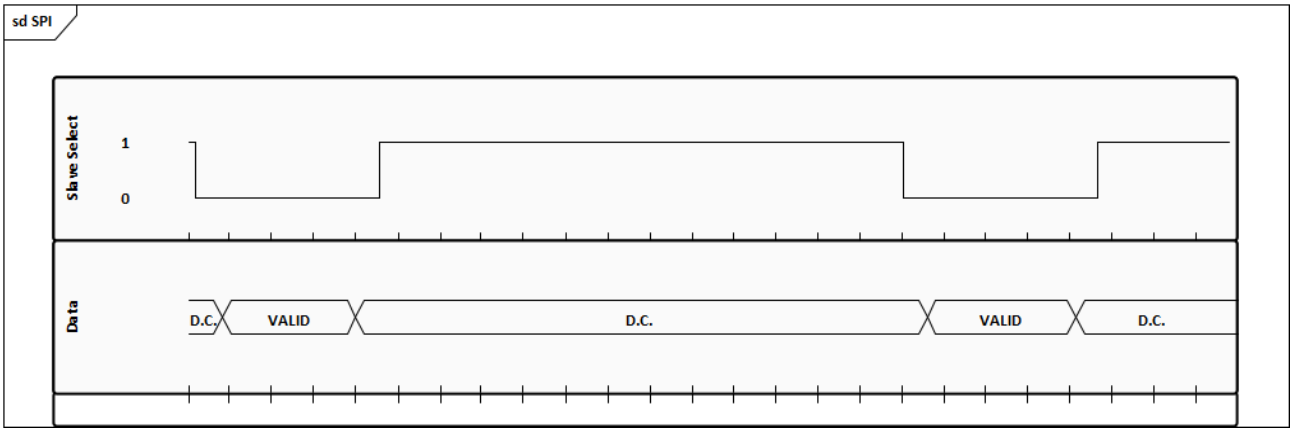

Figure 5.3 Basic SPI Timing

5.2.3.1 SPI Speed

The communication module supports SPI speed in the range of 29.3 kHz and 10 MHz. Default SPI speed is 2 MHz.

5.2.3.2 SPI Setup Timing

In Figure 5.3, Basic SPI Timing, the communication scheme between AC and CC is shown. Following time needs to be considered by the Application Controller during communication.

SPI Setup Time .. Time between activation of the module using the Slave Select signal and first data

The SPI Setup Time must at least be 10 ns, before the module accepts data over SPI.

5.2.3.3 SPI Cycle Time

This time is the distance between two consecutive SPI transfers. Between two transfers a minimum time of 250 us must be kept to secure proper processing in the module.

5.2.4 SPI Frame Structure

The SPI frame must contain the structure from Table 5.1, SPI Frame Structure to get accepted by the R-IN32M3 module.

Table 5.1 SPI Frame Structure

Bytes 0..1	**Byte 2**	**Byte 3**	**Bytes 4 .. 76**	**Bytes 77 .. 127**
Fletcher-16 Checksum with Offset 0x0007 (little endian)	Sequence	Data Length	Cyclic Data	RPC Data

The same layout is sent back by the device containing its local sequence counter. The sequence

counter is tracked and if it doesn’t change during the heartbeat timeout the communication gets stopped until the sequence is updated again.

5.2.4.1 Fletcher 16 Checksum (16 Bits)

To calculate the Fletcher-16 checksum can be used.
Start index is byte 4 and end is at 127. After the calculation the value 0x0007 needs to be added to not have false positives if the whole area is set to zeros. In the frame the value has a width of 16 bits and needs to have the little-endian encoding.

5.2.4.2 Sequence Counter (8 Bits)

The sequence counter must be incremented on each sent out frame to be recognized as new data. It can start at any 8-bit value.

5.2.4.3 Data Length (8 Bits)

If only cyclic data is transferred the data length can be set from 0 to 73 bytes. When using RPC over SPI the data length needs to have a fixed value of 124.

5.3 Remote Procedure Call (RPC)

The RPC protocol used by the C2C implementation of the R-IN32M3 module is transferred in byte 77 – 127. RPC transfers are always acknowledged to minimize data loss on erroneous transfers as good as possible. Also, the RPC calls can be larger than the available 50 bytes as the R-IN32M3 module internally stores each received RPC frame in a ring-buffer and waits until a partitioned transfer is completed before proceeding with the request.

5.3.1 RPC Frame

5.3.1.1 Structure

The RPC frame must contain the structure from Table 5.2, RPC Frame Structure to be accepted by the R-IN32M3 module.

Table 5.2 RPC Frame Structure

Bytes 0..1	Byte 2	Byte 3	Byte 4	Byte 5	Bytes 6 .. 49
Fletcher-16 Checksum with Offset 0x0007 (little endian)	Local Sequence	Remote Sequence Acknowledge	Data Length	Flags	Data

Each time a new local sequence is sent the R-IN32M3 module will respond with the corresponding acknowledge in the second transferred frame. If no acknowledge was received the AC can retransmit its frame to re-request the acknowledge.

5.3.1.2 Fletcher-16 Checksum (16 Bits)

To calculate the Fletcher-16 checksum can be used.
Start index is byte 6 and end is at the included data length. After the calculation the value 0x0007 needs to be added to not have false positives if the whole area is set to zeros. In the frame the value has a width of 16 bits and needs to have the little-endian encoding.

5.3.1.3 Local Sequence (8 Bits)

The sequence counter must be incremented for each RPC frame with changed data. For each unseen incremental frame, the R-IN32M3 module will put the transferred data into its internal ring-buffer and after the length matches it will process the RPC call.

5.3.1.4 Remote Sequence Acknowledge (8 Bits)

For each processed received RPC frame the AC needs to send out an acknowledge. If the received frame doesn't match the expected sequence the AC must leave the acknowledge on the previous sequence to trigger a resend from the R-IN32M3 module. If the resend fails the AC can also perform a re-sync.

5.3.1.5 Data Length (8 Bits)

Contains the length of the RPC data and must be between 0 (acknowledge-only frame) and 44.

5.3.2 Flags (8 Bits)

5.3.2.1 Structure

Table 5.3 RPC Header Flags

Bit 0	Bit 1	Bit 2	Bit 3
Sync Request	Sync Acknowledge	Reserved	Request Acknowledge

5.3.2.2 Sync Request

If set to 1 the R-IN32M3 module enters the RPC synchronization.

5.3.2.3 Sync Acknowledge

During the Sync Request both sides need to set the Sync Acknowledge flag, see section 5.3.3, RPC Synchronization for details.

5.3.2.4 Request Acknowledge

Forces an acknowledge from the partner device.

5.3.3 RPC Synchronization

Before the R-IN32M3 module is ready to operate and after a synchronization is lost the RPC must be synchronized. This is triggered by setting the Sync Request flag to 1.

Table 5.4 RPC Synchronization

AC	R-IN32M3 Module
Sync Request = 1 Local Sequence = 0 Local Sequence Acknowledge = 0 Remote Sequence Acknowledge = 0	
	Sync Request = 1 Local Sequence = 0 Local Sequence Acknowledge = 0 Remote Sequence Acknowledge = 0
Sync Request = 0 Sync Acknowledge = 1 Local Sequence = 1 RPC_Send(<empty>)	
	RPC_Receive() → Remote Sequence Acknowledge = 1 Sync Request = 0 Sync Acknowledge = 1 Local Sequence = 1 RPC_Send(<empty>)
RPC_Receive() → Local Sequence Acknowledge = 1	

→ Remote Sequence Acknowledge = 1 Sync Acknowledge = 0 RPC_Start()	
	RPC_Receive() → Local Sequence Acknowledge = 1 RPC_Start()

5.3.4 RPC Protocol

5.3.4.1 Introduction

The GOAL RPC protocol uses a virtual push/pop stack to call remote API functions with arguments. Each call must have a return value that is usually set to GOAL_OK when the call succeeded.

5.3.5 RPC Request/Response

5.3.5.1 Structure

An RPC request/response consists of the parts shown in Table 5.5, RPC Request Structure and Table 5.6, RPC Response Structure.

Table 5.5 RPC Request Structure

Byte 0..1	Byte 2..5	Byte 6..9	Byte 10..13	Byte 14	Byte 15	Byte 16..x	Byte (x+1)..(x+3)
Static Identifier 0xaa, 0xee	Data Length from Byte 6 to x	Function Id (little endian)	RPC Id (little endian)	CTC Id	Flags	Data	Fletcher-16 Checksum with Offset 0x0007 (little endian)

Table 5.6 RPC Response Structure

Byte 0..1	Byte 2..5	Byte 6 .. x	Byte x+1	Byte x+2	Byte (x+3)..(x+4)
Static Identifier 0xaa, 0xee	Data Length From Byte 6 to Flags (included)	Response Data	CTC Id	Flags	Fletcher-16 Checksum with Offset 0x0007 (little endian)

5.3.5.2 Static Identifier

The 2-byte static identifier is used on the R-IN32M3 module to detect the start of a new RPC request or response. If the identifier is also contained in the data bytes the Fletcher-16 checksum will make sure that it isn't handled as a thread in the same way as an RPC request/response.

5.3.5.3 Data Length

The data length contains the count of bytes starting with the “Function Id” and ending with the last data byte.

5.3.5.4 RPC Id

The RPC id is the module or group id. For example, GOAL PROFINET uses the RPC id GOAL_ID_PNIO to register its calls.

5.3.5.5 Function Id

The function id is used as a sub id to map the function calls in the specific module to their handlers.

5.3.5.6 CTC Id

The CTC id is used to match requests and responses to their specific MCTC internal handle. A response must use the same CTC id as the request.

5.3.5.7 Flags

The flags define the type of the request, see Table 5.7, RPC Request/Response Flags for details.

Table 5.7 RPC Request/Response Flags

Bit **0..1**
0 – Response 1 – Request 2 – Info (Request without waiting for a response)

5.3.5.8 Data

The data part contains the virtual stack of the RPC request/response.

5.3.5.9 Fletcher-16 Checksum (16 Bits)

To calculate the Fletcher-16 checksum can be used.
Start index is byte 16 and end is at the included data length. After the calculation the value 0x0007 needs to be added to not have false positives if the whole area is set to zeros. In the frame the value has a width of 16 bits and needs to have the little-endian encoding.

5.4 Communicational Stack - PROFINET

5.4.1 Introduction

This section lists the API functions that are provided by GOAL PROFINET.

5.4.2 goal_pnioCfgVendorId – Set Vendor Id

Configures the vendor id. Default: 0x02c7 (Renesas Electronics); It returns a GOAL_STATUS_T status.

This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Table 5.8 goal_pnioCfgVendorIdSet Parameter

Parameter	Description
uint16_t idVendor	Vendor ID

5.4.3 goal_pnioCfgDeviceIdSet – Set Device Id

Configures the device id. Default: 0x0001; It returns a GOAL_STATUS_T status.
This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Table 5.9 goal_pnioCfgDeviceIdSet Parameter

Parameter	Description
uint16_t idDevice	Device ID

5.4.4 goal_pnioCfgVendorNameSet - Set Vendor Name

Configures the vendor name. Returns a GOAL_STATUS_T status.
This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Table 5.10 goal_pnioCfgVendorIdSet Parameter

Parameter	Description
const char *strVendor	Vendor Name

5.4.5 goal_pnioCfgPortDescSet - Set LLDP Port Description

Configures the LLDP port description. Default: “TestPort”; It returns a GOAL_STATUS_T status.
This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Table 5.11 goal_pnioCfgPortDescSet Parameter

Parameter	Description
const char *strDescPort	Port Description

5.4.6 goal_pnioCfgSystemDescSet - Set LLDP System Description

Configures the LLDP system description. Default: “PROFINET System”; It returns a GOAL_STATUS_T status.
This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Table 5.12 goal_pnioCfgSystemDescSet Parameters

Parameter	Description
const char *strSystem	System Description

5.4.7 goal_pnioCfgOrderIdSet - Set Order Id

Configures the order id. It returns a GOAL_STATUS_T status.
This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Table 5.13 goal_pnioCfgOrderIdSet Parameters

Parameter	Description
const char *strOrder	Order Id

5.4.8 goal_pnioCfgSerialNumSet - Set Serial Number

Configures the serial number. It returns a GOAL_STATUS_T status.
This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Table 5.14 goal_pnioCfgSerialNumSet Parameters

Parameter	Description
const char *strNumSerial	Serial Number

5.4.9 goal_pnioCfgHwRevSet - Set Hardware Revision

Configures the hardware revision.; It returns a GOAL_STATUS_T status.
This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Table 5.15 goal_pnioCfgHwRevSet Parameters

Parameter	Description
uint16_t idRevHw	Hardware Revision

5.4.10 goal_pnioCfgSwRevPrefixSet - Set Software Revision Prefix

Configures the software revision prefix. Default: “P”

- “V” - official
- “R” - revision
- “P” - prototype
- “U” - under test
- “T” - test device

It returns a GOAL_STATUS_T status.
This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Table 5.16 goal_pnioCfgSwRevPrefixSet Parameters

Parameter	Description
const char chrRevSwPrefix	Software Revision Prefix

See example 15_config_set for a demonstration.

5.4.11 pnioCfgSwRevFuncEnhSet - Set Software Revision Functional Enhancement

Configures the software revision functional enhancement. Default: 0x50
It returns a GOAL_STATUS_T status.
This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Table 5.17 goal_pnioCfgSwRevFuncEnhSet Parameters

Parameter	Description
uint8_t idRevSwFuncEnh	Software Revision Functional Enhancement

5.4.12 goal_pnioCfgSwRevBugfixSet - Set Software Revision Bugfix

Configures the software revision bugfix. Default: 3
It returns a GOAL_STATUS_T status.
This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Table 5.18 goal_pnioCfgSwRevBugfixSet Parameters

Parameter	Description
uint8_t idRevSwBugfix	Software Revision Bugfix

5.4.13 goal_pnioCfgSwRevIntChgSet - Set Software Revision Internal Change

Configures the software revision internal change. Default: 0x18
It returns a GOAL_STATUS_T status.
This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Table 5.19 goal_pnioCfgSwRevIntChgSet Parameters

Parameter	Description
uint8_t idRevSwIntChg	Software Revision Internal Change

5.4.14 goal_pnioCfgSwRevCntSet - Set Software Revision Counter

Configures the software revision counter. Default: 0x0000
It returns a GOAL_STATUS_T status.
This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Table 5.20 goal_pnioCfgSwRevBugfixSet Parameters

Parameter	Description
uint16_t idRevSwRevCnt	Software Revision Counter

5.4.15 goal_pnioCfgIm1TagFuncSet - Set I&M1 Tag Function

Configures the I&M1 tag function. Default: “”
It returns a GOAL_STATUS_T status.
This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Table 5.21 goal_pnioCfgIm1TagFuncSet Parameters

Parameter	Description

const char *strIm1TagFunc	I&M1 Tag Function

5.4.16 goal_pnioCfgIm1TagLocSet - Set I&M1 Tag Location

Configures the I&M1 tag location. Default: “”
It returns a GOAL_STATUS_T status.
This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Table 5.22 goal_pnioCfgIm1TagLocSet Parameters

Parameter	Description
const char *strIm1TagLoc	I&M1 Tag Location

5.4.17 goal_pnioCfgIm2DateSet - Set I&M2 Date

Configures the I&M2 date. Default: “”
It returns a GOAL_STATUS_T status.
This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Table 5.23 goal_pnioCfgIm2DateSet Parameters

Parameter	Description
const char *strIm2Date	I&M2 Date

5.4.18 goal_pnioCfgIm3DescSet - Set I&M3 Description

Configures the I&M3 description. Default: “”
It returns a GOAL_STATUS_T status.
This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Table 5.24 goal_pnioCfgIm3DescSet Parameters

Parameter	**Description**
const char *strIm3Desc	**I&M3 Description**

5.4.19 goal_pnioCfgIm4SigSet - Set I&M4 Signature (Functional Safety)

Configures the I&M4 signature. Default: “”
It returns a GOAL_STATUS_T status.
This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Table 5.25 goal_pnioCfgIm4SigSet Parameters

Parameter	Description
const char *strIm4Sig	I&M4 Signature

5.4.20 goal_pnioCfgLldpOrgExtSet - Configure LLDP Organizationally-specific Extension

Configures the LLDP organizationally-specific extension.
Default: GOAL_TRUE It returns a GOAL_STATUS_T status.
This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.
Warning: This function should not be used as the behavior can break conformity.

Table 5.26 goal_pnioCfgLldpOrgExtSet Parameters

Parameter	Description
GOAL_BOOL_T flgLldpOrgExt	LLDP Organizationally-specific Extension Flag

5.4.21 goal_pnioCfgLldpOptTlvSet - Configure LLDP Optional TLV Parameters

Configures the LLDP optional TLV parameters. Default: GOAL_TRUE These parameters contain:

- port description
- system name
- system description
- system capabilities
- management address
- object ID

It returns a GOAL_STATUS_T status.
This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.
Warning: This function should not be used as the behavior can break conformity.

Table 5.27 goal_pnioCfgLldpOptTlvSet Parameters

Parameter	Description
GOAL_BOOL_T flgLldpOptTlv	LLDP Optional TLV Parameters Flag

5.4.22 goal_pnioCfgLldpGenMacSet - Configure LLDP Port MAC Address Generation

Configures the automatic LLDP port MAC address generation. If set to GOAL_TRUE the LLDP port-specific MAC addresses are automatically generated by adding the port id to the host port MAC address.
Default: GOAL_TRUE It returns a GOAL_STATUS_T status.
This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Warning: If this functionality is enabled (default) every device uses at least two MAC addresses (real plus one per port) in an ascending order. To use specific virtual port MAC addresses this feature must be disabled and the driver must provide different MAC addresses per request (GOAL_ETH_PORT_HOST and port specific requests).

Table 5.28 goal_pnioCfgLldpGenMacSet Parameters

Parameter	Description
GOAL_BOOL_T flgLldpGenMac	Automatic LLDP MAC Generation Flag

5.4.23 goal_pnioCfgIm14SupportSet - Configure I&M 1-4 Support

Configures the GOAL PROFINET stack I&M 1-4 support. If set to GOAL_FALSE the stack denies I&M 1-4 requests. Default: GOAL_TRUE It returns a GOAL_STATUS_T status.
This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.
Warning: This function should be used carefully because it can break conformity.

Table 5.29 goal_pnioCfgIm14SupportSet Parameters

Parameter	Description
GOAL_BOOL_T flgIm14Support	I&M 1-4 Support Flag

5.4.24 goal_pnioCfgIm14CbSet - Configure I&M 1-4 Callback

Configures the GOAL PROFINET stack I&M 1-4 callback. If set to GOAL_TRUE the application callback must handle the I&M 1-4 get and set requests. Default: GOAL_FALSE It returns a GOAL_STATUS_T status.
This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.
Warning: This function should be used carefully because it can break conformity.

Table 5.30 goal_pnioCfgIm14CbSet Parameters

Parameter	Description
GOAL_BOOL_T flgIm14Cb	I&M 1-4 Callback Flag

5.4.25 goal_pnioCfgIm0CbSet - Configure I&M 0 Callback

Configures the GOAL PROFINET stack I&M 0 callback. If set to GOAL_TRUE the application callback must handle the I&M 0 get and set requests.
Default: GOAL_FALSE It returns a GOAL_STATUS_T status.
This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.
Warning: This function should be used carefully because it can break conformity.

Table 5.31 goal_pnioCfgIm0CbSet Parameters

Parameter	Description
GOAL_BOOL_T flgIm0Cb	I&M 0 Callback Flag

5.4.26 goal_pnioCfgIm0FilterDataCbSet - Configure I&M 0 Filter Data Callback

Configures the GOAL PROFINET stack I&M 0 filter data callback. If set to GOAL_TRUE the application callback must handle the I&M 0 filter data get and set requests.
Default: GOAL_FALSE
It returns a GOAL_STATUS_T status.
This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.
Warning: This function should be used carefully because it can break conformity.

Table 5.32 goal_pnioCfgIm0FilterDataCbSet Parameters

Parameter	Description
GOAL_BOOL_T flgIm0FilterCb	I&M 0 Filter Data Callback Flag

5.4.27 goal_pnioCfgRecDataBusyBufsizeSet - Configure Record Handle Storage Count

Configures the count of parallel record handles.
Default: 2 It returns a GOAL_STATUS_T status.
This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Table 5.33 goal_pnioCfgRecDataBusyBufsizeSet Parameters

Parameter	Description
GOAL_BOOL_T cntRecDataBusyBufsize	Record Handle Storage Count

5.4.28 goal_pnioCfgRpcFragReqLenMaxSet - Configure Maximum Record Size

Configures the maximum size in bytes of a record request. This must match the MaxSupportedRecordSize attribute in the GSDML file.
Default: 4068 It returns a GOAL_STATUS_T status.
This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.
Warning: Changing this value to a smaller number can break conformity.

Table 5.34 goal_pnioCfgRpcFragReqLenMaxSet Parameters

Parameter	Description
unsigned int sizeRpcFragMaxReqLen	Maximum Record Size

5.4.29 goal_pnioCfgRpcFragMaxCntSet – Configure Maximum RPC Fragment Number

This function is obsolete. The GOAL PROFINET stack now allows 64 fragmented frames.

5.4.30 goal_pnioCfgRpcFragEnableSet - Configure RPC Fragmentation

Configures the RPC fragmentation feature. If set to GOAL_TRUE RPC fragmentation is enabled. Default: GOAL_TRUE It returns a GOAL_STATUS_T status.
This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.
Warning: This function should not be used as the behavior can break conformity.

Table 5.35 goal_pnioCfgRpcFragEnableSet Parameters

Parameter	Description
GOAL_BOOL_T flgRpcFragSupport	RPC Fragmentation Enable Flag

5.4.31 goal_pnioCfgRpcSessionMaxCntSet - Configure Maximum RPC Session Count

Configures the maximum count of RPC sessions. Default: 8; It returns a GOAL_STATUS_T status.
This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.
Warning: This function should not be used as the behavior can break conformity.

Table 5.36 goal_pnioCfgRpcSessionMaxCntSet Parameters

Parameter	Description
unsigned int numRpcSessions	RPC Session Count

5.4.32 goal_pnioVendorIdSet - Set the vendor id

Set the vendor id. Returns a GOAL_STATUS_T status.

Table 5.37 goal_pnioVendorIdSet Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET data
uint16_t id	vendor id

5.4.33 goal_pnioDeviceIdSet - Set the device id

Set the device id. Returns a GOAL_STATUS_T status.

Table 5.38 goal_pnioDeviceIdSet Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET data
uint16_t id	device id

5.4.34 goal_pnioHwRevSet - Set the hardware revision

Set the hardware revision. Returns a GOAL_STATUS_T status.

Table 5.39 goal_pnioHwRevSet Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET data
uint16_t revHw	hardware revision

5.4.35 goal_pnioSwRevSet - Set the software revision

Set the software revision. Returns a GOAL_STATUS_T status.

Table 5.40 goal_pnioSwRevSet Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET data
uint8_t chrPrefix	prefix
uint8_t idFuncEnh	functional enhancement
uint8_t idBugfix	bugfix
uint8_t idIntChange	internal change
uint16_t cntRev	revision counter

5.4.36 goal_pnioProfileIdSet - Set the profile id

Set the profile id. Returns a GOAL_STATUS_T status.

Table 5.41 goal_pnioProfileIdSet Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET data
uint16_t id	profile id
uint16_t type	profile specific type

5.4.37 goal_pnioOrderIdSet - Set the order id

Set the order id. Returns a GOAL_STATUS_T status.

Table 5.42 goal_pnioOrderIdSet Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET data
const char *strOrder	order id
uint32_t lenOrder	order id length

5.4.38 goal_pnioSerialNumSet - Set the serial number

Set the serial number. Returns a GOAL_STATUS_T status.

Table 5.43 goal_pnioSerialNumSet Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET data
const char *strSerial	serial number
uint32_t lenSerial	serial number length

5.4.39 goal_pnioVendorNameSet - Set the vendor name

Set the vendor name. Returns a GOAL_STATUS_T status.

Table 5.44 goal_pnioVendorNameSet Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET data
const char *strVendor	vendor name
uint32_t lenVendor	vendor name length

5.4.40 goal_pnioPortDescSet - Set the port description

Set the port description. Returns a GOAL_STATUS_T status.

Table 5.45 goal_pnioPortDescSet Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET data
const char *strPort	port description
uint32_t lenPort	port description length

5.4.41 goal_pnioSystemDescSet - Set the system description

Set the system description. Returns a GOAL_STATUS_T status.

Table 5.46 goal_pnioSystemDescSet Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET data
const char *strSys	system description
uint32_t lenSys	system description length

5.4.42 goal_pnioSubslotStateSet - Permit usage of wrong submodules (substitute)

If a not-matching submodule is plugged the application can use this function to mark the submodule as substitute. Possible values for state are:

- GOAL_PNIO_SUBSLOT_STATE_OK
- GOAL_PNIO_SUBSLOT_STATE_SUBST

The default value is GOAL_PNIO_SUBSLOT_STATE_OK and handles the subslot state internally. The subslot state is overwritten by GOAL_PNIO_SUBSLOT_STATE_SUBST to indicate that the submodule is to be treated as a substitute if a submodule is plugged and only has a non-matching module or submodule id.

Table 5.47 goal_pnioSystemDescSet Parameters

Parameter	Description
uint16_t idSlot	slot id
uint16_t idSubslot	subslot id
uint16_t state	subslot state

5.4.43 goal_pnioConfClassGet - Get active conformance class

The GOAL PROFINET stack supports Conformance Class A and B. There is currently one special case where the GOAL PROFINET needs to know the used Conformance Class. The PdevData record must only be supported for Conformance Class B and higher. Valid return values for pIdConfClass are GOAL_PNIO_CC_A and GOAL_PNIO_CC_B.

Table 5.48 goal_pnioConfClassGet Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET instance
GOAL_PNIO_CC_T *pIdConfClass	Conformance Class storage

5.4.44 goal_pnioConfClassSet - Set active conformance class

The GOAL PROFINET stack supports Conformance Class A and B. There is currently one special case where the GOAL PROFINET needs to know the used Conformance Class. The PdevData record must only be supported for Conformance Class B and higher. Valid values for idConfClass are GOAL_PNIO_CC_A and GOAL_PNIO_CC_B.

Table 5.49 goal_pnioConfClassSet Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET instance
GOAL_PNIO_CC_T idConfClass	Conformance Class

5.4.45 goal_pnioConfTestGet - Get current PROFINET test environment

This delivers detailed information about the currently used test environment.

Table 5.50 goal_pnioConfTestGet Parameters

Parameter	Description
const char **pStrPnioStd	PROFINET Standard
const char **pStrTestSpec	PROFINET Test Specification
const char **pStrTestcases	PROFINET Testcases
const char **pStrArt	PROFINET ART
const char **pStrTedCheck	PROFINET TED Check
const char **pStrPnio	PROFINET Version
const char **pStrGsd	PROFINET GSDML Version

5.4.46 Data Mapper API

The GOAL PROFINET Data Mapper API allows to map PROFINET subslots, producer & consumer states and the connection information to the cyclic data stream of the Data Mapper that is used for example in MCTC transfers.

5.4.47 Map Subslot Data – goal_pnioDmSubslotAdd

The API goal_pnioDmSubslotAdd maps input and output data of the given subslot to the also given instances of the DM.

Table 5.51 goal_pnioDmSubslotAdd API Description

Parameter	Description
GOAL_PNIO_T *pPnio	GOAL PROFINET instance
uint32_t idMiDmPeerFrom	Handle that receives data from CC
uint32_t idMiDmPeerTo	Handle that sends data to CC
GOAL_MI_DM_PART_T **ppPartDataOut	Output pointer to store the DM partition
GOAL_MI_DM_PART_T **ppPartDataIn	Output pointer to store the DM partition
uint32_t idApi	API id
uint16_t idSlot	Slot id
uint16_t idSubslot	Subslot id
uint32_t lenDataOut	Output data length
uint32_t lenDataIn	Input data length

5.4.48 Map Subslot IOCS/IOPS - goal_pnioDmSubslotIoxsAdd

The API goal_pnioDmSubslotIoxsAdd maps IOCS and IOPS states of the given subslot to the given instances of the DM.

Table 5.52 goal_pnioDmSubslotAdd API Description

Parameter	Description
GOAL_PNIO_T *pPnio	GOAL PROFINET instance
uint32_t idMiDmPeerFrom	Handle that receives data from CC
uint32_t idMiDmPeerTo	Handle that sends data to CC
GOAL_MI_DM_PART_T **ppPartIocsOut	Output pointer to store the DM partition
GOAL_MI_DM_PART_T **ppPartIopsOut	Output pointer to store the DM partition
GOAL_MI_DM_PART_T **ppPartIocsIn	Output pointer to store the DM partition
GOAL_MI_DM_PART_T **ppPartIopsIn	Output pointer to store the DM partition
uint32_t idApi	API id
uint16_t idSlot	Slot id
uint16_t idSubslot	Subslot id

5.4.49 Map APDU Status – goal_pnioDmApduAdd

The API goal_pnioDmApduAdd maps the APDU status to the given instance of the DM.

Table 5.53 goal_pnioDmSubslotAdd API Description

Parameter	Description
GOAL_PNIO_T *pPnio	GOAL PROFINET instance
uint32_t idMiDmPeerTo	(Ignored) Handle that sends data to CC
GOAL_MI_DM_PART_T **ppPartApduOut	Output pointer to store the DM partition

5.4.50 Map Data Provider Status – goal_pnioDmDpAdd

The API goal_pnioDmDpAdd maps the Data Provider status to the given instance of the DM.

Table 5.54 goal_pnioDmApduAdd API Description

Parameter	Description
GOAL_PNIO_T *pPnio	GOAL PROFINET instance
uint32_t idMiDmPeerTo	(Ignored) Handle that sends data to CC
GOAL_MI_DM_PART_T **ppPartDp	Output pointer to store the DM partition

5.5 Application Callbacks – PROFINET

5.5.1 Introduction

To use the full potential of PROFINET the stack allows you to interact at several stages of the protocol. For example, you can withhold the sending of the application ready signal to the PLC or handle record data reads and writes to specific slots.
The first thing to use the callback system is to provide a callback function that the stack can call every time it wants to communicate with the application. The callback function must have the following prototype:

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

It is registered with the call to goal_pnioInit which takes the pointer to the callback function as second parameter.
res = goal_pnioInit(&pPnio, main_callback);
Now every time the main_callback is called, the GOAL_PNIO_CB_ID_T value contains the reason of the call. The GOAL_PNIO_CB_DATA_T structure contains an array of multiple unions where each array element has a special meaning depending on the GOAL_PNIO_CB_ID_T value.
The following subsections will provide you with the details on the available callback ids.

5.5.2 GOAL_PNIO_CB_ID_ALARM_ACK_TIMEOUT - Timeout Waiting for Alarm ACK

Indicate that a timeout occurred in APMS while waiting for an alarm acknowledge.

Table 5.55 TIMEOUT - Timeout Waiting for Alarm ACK

Parameter	Description
cb->data[0].idAr	application relation id
cb->data[1].u16	timed out alarm sequence number

5.5.3 GOAL_PNIO_CB_ID_ALARM_NOTIFY_ACK - Alarm Notification ACK Received

Indicates that an alarm notification ACK was received.

Table 5.56 Alarm Notification ACK Received

Parameter	Description
cb->data[0].idAr	application relation id
cb->data[1].u16	alarm priority (low, high)
cb->data[2].pAlarmNotifyAck	GOAL_PNIO_ALARM_NOTIFY_ACK_T structure pointer

5.5.4 GOAL_PNIO_CB_ID_ALARM_NOTIFY - Alarm Notification Received

Indicates that an alarm notification was received. If the callback isn't handled or GOAL_OK is returned the PROFINET stack will automatically acknowledge the received alarm notification.

Table 5.57 Alarm Notification Received

Parameter	Description
cb->data[0].idAr	application relation id
cb->data[1].u16	alarm priority (low, high)
cb->data[2].pAlarmNotify	GOAL_PNIO_ALARM_NOTIFY_T structure pointer
cb->data[3].u32	user data length
cb->data[4].pCu8	user data pointer

5.5.5 GOAL_PNIO_CB_ID_APPL_READY - Application Ready Response Received

Indicate that an application ready response was received. The return value must be GOAL_OK.

Table 5.58 Application Ready Response Received

Parameter	Description
cb->data[0].idAr	application relation id

5.5.6 GOAL_PNIO_CB_ID_BLINK - Blink Request

Indicates that a DCP signal request was received and fills the callback data with the necessary information.

cb->data[0].stateDcpBlink tells the application the initial signal request (START), the correct time to toggle the signal (TOGGLE) and when the signal phase is over (FINISH). This information can be used if the application wants to fully represent the signal by itself.

The second possibility is to use the cb->data[1].stateDcpLight data which contains either GOAL_PNIO_DCP_LIGHT_OFF or GOAL_PNIO_DCP_LIGHT_ON which can be used to handover the signal indicator directly to a LED or blink function.
If variant 1 or 2 is used then the callback has to return GOAL_OK_SUPPORTED otherwise the stack handles the blinking by itself by calling goal_targetSetLeds with the GOAL_PNIO_LED_SIGNAL define. For stack internal handling the implementation of goal_targetGetLeds and goal_targetSetLeds is mandatory.

Table 5.59 Blink Request

Parameter	Description
cb->data[0].stateDcpBlink	DCP blink state (start, toggle, finish)
cb->data[1].stateDcpLight	DCP light state (on, off)

Return Values	Description
GOAL_OK	signal handled stack internal
GOAL_OK_SUPPORTED	application handles signal

```
case GOAL_PNIO_CB_ID_BLINK:

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

case
    GOAL_PNIO_DCP_BLINK_START:
    /* start blinking */ break;

case
    GOAL_PNIO_DCP_BLINK_TOGGL
    E: /* 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 - Connect Request Done

Indicates that a connect request was fully processed and allows the application to cancel it by setting the error status and return a value that is not ***GOAL_OK***.

Table 5.60 Connect Request Done

Parameter	Description
cb->data[0].idAr	application relation id
cb->data[1].pStatus	pointer to PROFINET status

5.5.8 GOAL_PNIO_CB_ID_CONNECT_REQUEST - Connect Request

Indicates that the processing of a connect request has been started. The parameter pointer is set to ***NULL***.

5.5.9 GOAL_PNIO_CB_ID_CONNECT_REQUEST_EXP_START- Expected Submodule Block Start

Indicates that an expected submodule block starts. This can be used to unplug all modules before the request callback for each module comes in.

Table 5.61 Expected Submodule Block Start

Parameter	Description
cb->data[0].idAr	application relation id

5.5.10 GOAL_PNIO_CB_ID_END_OF_PARAM - Param End Received

Indicates that the parameter end information was received.

Table 5.62 Param End Received

Parameter	Description
cb->data[0].idAr	application relation id

5.5.11 GOAL_PNIO_CB_ID_END_OF_PARAM_PLUG – Plug Param End Received

Indicates the plug parameter end information was received.

Table 5.63 Plug Param End Received

Parameter	Description
cb->data[0].idAr	application relation id
cb->data[1].u16	plug handle

5.5.12 GOAL_PNIO_CB_ID_EXP_SUBMOD - Expected Submodule

Indicates an expected submodule which can be plugged immediately.

Table 5.64 Expected Submodule

Parameter	Description
cb->data[0].idAr	application relation id
cb->data[1].u32	API
cb->data[2].u16	slot number
cb->data[3].u16	subslot number
cb->data[4].u32	module ident number
cb->data[5].u32	submodule ident number
cb->data[6].valBool	module flag (true = module, false = submodule)

5.5.13 GOAL_PNIO_CB_ID_FACTORY_RESET - Factory Reset

Indicates a factory reset. The parameter pointer is set to ***NULL*.**

5.5.14 GOAL_PNIO_CB_ID_IO_DATA_TIMEOUT - Cyclic Timeout

Indicates that an output endpoint timed out.

5.5.15 GOAL_PNIO_CB_ID_NET_IP_SET - IP Configuration Update

Indicate that the IP configuration was updated. The internal change flag indicates if the change was triggered by PROFINET DCP (PN_TRUE) or by an external caller like DHCP (PN_FALSE).

Table 5.65 IP Configuration Update

Parameter	Description
cb->data[0].u32	ip address
cb->data[1].u32	netmask
cb->data[2].u32	gateway
cb->data[3].valBool	temporary setting flag
cb->data[4].valBool	internal update flag

5.5.16 GOAL_PNIO_CB_ID_NEW_AR - New Application Relation

Indicates a new application relation. A return value different from ***GOAL_OK*** drops the AR and replies with an error.

Table 5.66 New Application Relation

Parameter	Description
cb->data[0].idAr	application relation id

5.5.17 GOAL_PNIO_CB_ID_NEW_IO_DATA - New IO Data

Indicates reception of new IO data.

Table 5.67 New IO Data

Parameter	Description
cb->data[0].u16	cyclic frame id
cb->data[1].u16	data length
cb->data[2].pCu8	data pointer

5.5.18 GOAL_PNIO_CB_ID_PLUG_READY - Plug Ready Response Received

Indicates that a plug ready response was received.

Table 5.68 Plug Ready Response Received

Parameter	Description
cb->data[0].idAr	application relation id

5.5.19 GOAL_PNIO_CB_ID_READ_RECORD - Read Record Data

Indicates that record data that couldn't be handled by the stack itself should be read. Before this callback is called, the stack checks if the API, slot and subslot combination is valid.
If the function returns GOAL_OK, the request will be denied with "invalid index".
Otherwise, if the application can handle the request, it needs to return GOAL_OK_SUPPORTED and can provide the answer either directly in the callback by calling the API goal_pnioRecReadFinish or store the callback information and respond outside the callback, also by calling the API goal_pnioRecReadFinish. The API goal_pnioRecReadFinish also allows to overwrite the PROFINET status if the matching parameter is set.
If the response isn't GOAL_OK or GOAL_OK_SUPPORTED, the stack will automatically respond with application read error.
The parameter "sequence number handle" is used later when the response is sent to detect if the request has expired.

Table 5.69 Read Record Data

Parameter	Description
cb->data[0].pStatus	unused (pointer to PROFINET status)
cb->data[1].idAr	application relation id
cb->data[2].u32	API
cb->data[3].u16	slot
cb->data[4].u16	subslot
cb->data[5].u16	record index
cb->data[6].pU8	unused (pointer to store record data)
cb->data[7].u32	maximum record read data length
cb->data[8].i32	internal record handle
cb->data[9].u32	sequence number handle

Example callback code:

```
    case GOAL_PNIO_CB_ID_READ_RECORD:
    goal_logInfo("read request received"); goal_logInfo("address:
                                    %"FMT_u32":%u:%u, idx: %u",
    pCb->data[2].u32,                       /*    API      */
pCb->data[3].u16,                          /*    slot     */
pCb->data[4].u16,           /*             subslot       */
pCb->data[5].u16 /* record index */);

    if (GOAL_TRUE == flgAnswerInCallback) {

    /* send answer direct in callback */
    res = goal_pnioRecReadFinish(
    pPnio,                                  /* PROFINET handle */
                    pCb->data[8].i32,                   /* internal record handle */
                     NULL,                             /* PROFINET Status or NULL */
                     DATA_BUFFER,                       /* answer data buffer */
        DATA_LENGTH,                                  /* answer data length */
        pCb->data[9].u32                              /* sequence number handle */
```

```
);

if (GOAL_RES_ERR(res)) { goal_logErr("failed to answer record read request"); }

} else {            /* postpone answer by storing the request */
                    GOAL_MEMCPY(REQUEST_STORAGE, pCb,
                        sizeof(GOAL_PNIO_CB_DATA_T));}
/* tell the PROFINET stack that the record was or will be handled */
return GOAL_OK_SUPPORTED;
```

5.5.20 GOAL_PNIO_CB_ID_RELEASE_AR - Release Application Relation

Indicates that an application relation was released.

Table 5.70 Release Application Relation

Parameter	Description
cb->data[0].idAr	application relation id

5.5.21 GOAL_PNIO_CB_ID_RESET_TO_FACTORY - Reset To Factory

Indicates a reset to factory request. If there is a separate request handling outside of the stack the application must return GOAL_OK_SUPPORTED.
If the application doesn't return from the callback with GOAL_OK or GOAL_OK_SUPPORTED the DCP error "0x04 suboption not set" is sent back to the DCP set request sender.

Table 5.71 Reset To Factory

Parameter	Description
cb->data[0].u16	reset to factory action

5.5.22 GOAL_PNIO_CB_ID_STATION_NAME - Station Name Changed

Indicates a station name change. If the permanent flag is set to GOAL_TRUE, the station name is stored in NVS, otherwise the station name is only stored temporarily and the NVS value is cleared.

Table 5.72 Station Name Changed

Parameter	Description
cb->data[0].pCu8	name of station
cb->data[1].u32	name of station length
cb->data[2].valBool	permanent flag

5.5.23 GOAL_PNIO_CB_ID_WRITE_RECORD - Write Record Data

Indicates that record data that couldn't be handled by the stack itself should be written. Before this callback is called, the stack checks if the API, slot and subslot combination is valid.
If the function returns GOAL_OK the request will be denied with "invalid index".
Otherwise, if the application can handle the request, it needs to return GOAL_OK_SUPPORTED and can provide the answer either directly in the callback by calling the API goal_pnioRecWriteFinish or store the callback information and respond outside the callback, also by calling the API goal_pnioRecWriteFinish. The API goal_pnioRecReadFinish also allows to overwrite the PROFINET status if the matching parameter is set.
If the response isn't GOAL_OK or GOAL_OK_SUPPORTED, the stack will automatically respond with application read error.
The parameter "sequence number handle" is used later when the response is sent to detect if the request has expired.

Table 5.73 Write Record Data

Parameter	Description
cb->data[0].pStatus	unused (pointer to PROFINET status)
cb->data[1].idAr	application relation id
cb->data[2].u32	API
cb->data[3].u16	slot
cb->data[4].u16	subslot
cb->data[5].u16	record index
cb->data[6].pCu8	pointer to read record data from
cb->data[7].u32	record data length
cb->data[8].i32	internal record handle
cb->data[9].valBool	subslot locked status flag
cb->data[10].u32	sequence number handle

Example callback code:

```
case GOAL_PNIO_CB_ID_WRITE_RECORD:
goal_logInfo("write request received"); goal_logInfo("address: "FMT_u32":%u:%u, idx: %u",
pCb->data[2].u32, /* API */
pCb->data[3].u16, /* slot */
pCb->data[4].u16, /* subslot */
pCb->data[5].u16 /* record index */);
if (GOAL_TRUE == flgAnswerInCallback) {

/* send answer direct in callback */
  res = goal_pnioRecWriteFinish(
                          pPnio,                          /* PROFINET handle */
                         pCb->data[8].i32,                  /* internal record handle */
                          NULL,                          /* PROFINET Status or NULL */
                        pCb->data[10].u32                  /* sequence number handle */
);
  if (GOAL_RES_ERR(res)) { goal_logErr("failed to answer record write request"); }
} else {
/* postpone answer by storing the request */
```

```
    GOAL_MEMCPY(REQUEST_STORAGE, pCb, sizeof(GOAL_PNIO_CB_DATA_T));
}
    /* tell the PROFINET stack that the record was or will be handled */
return GOAL_OK_SUPPORTED
```

5.5.24 GOAL_PNIO_CB_ID_INIT - Stack Initialized

Indicates that the PROFINET stack is fully initialized. The parameter pointer is set to *NULL*. At this point it is safe to create the device configuration.

5.5.25 GOAL_PNIO_CB_ID_LLDP_UPDATE - LLDP Update

Signalizes that the device on the partner port changed.

Table 5.74 LLDP Update

Parameter	Description
cb->data[0].u32	GOAL Ethernet Port Id

5.5.26 GOAL_PNIO_CB_ID_CONN_REQ_EXP_FINISH - Connect Request Expected Submodule Block Finish

This callback is called after the Expected Submodule Block in the Connect Request has been parsed.

Table 5.75 Connect Request Expected Submodule Block Finish

Parameter	Description
cb->data[0].idAr	application relation id

5.5.27 GOAL_PNIO_CB_ID_STATION_NAME_VERIFY - DCP Station Name Verification

Let the application verify a DCP Station Name Set Request.

If the function returns a GOAL error status the set request will be denied.

Table 5.76 DCP Station Name Verification

Parameter	Description
cb->data[0].pCu8	pointer to station name (unterminated)
cb->data[1].u32	length of station name
cb->data[2].valBool	permanent flag (permanent = true)

5.5.28 GOAL_PNIO_CB_ID_NET_IP_SET_VERIFY - DCP IP Configuration Verification

Let the application verify a DCP IP Configuration Set Request.

If the function returns a GOAL error status the set request will be denied.

Table 5.77 DCP IP Configuration Verification

Parameter	Description
cb->data[0].u32	IP address
cb->data[1].u32	netmask
cb->data[2].u32	gateway
cb->data[3].valBool	temporary flag (temporary = true)

5.6 Communication Stack – EtherNet/IP

5.6.1 goal_eipCfgVendorIdSet

Set the Vendor ID of this EtherNet/IP stack instance. The vendor ID is assigned by the ODVA.

Default Value: 1105

Table 5.78 goal_eipCfgVendorIdSet Parameters

Parameter	Description
uint16_t vendorId	Vendor ID

res = goal_eipCfgVendorIdSet(1105);

5.6.2 goal_eipCfgDeviceTypeSet

Set the Device Type of this EtherNet/IP stack instance. Valid values are defined by the CIP specification.
Default Value: 0x2B

Table 5.79 goal_eipCfgVendorIdSet Parameters

Parameter	Description
uint16_t deviceType	Device Type

res = goal_eipCfgDeviceTypeSet(0x2B);

5.6.3 goal_eipCfgProductCodeSet

Set the Product Code of this EtherNet/IP stack instance. This value is defined by the device vendor.

Default Value: 1

Table 5.80 goal_eipCfgProductCodeSet Parameters

Parameter	Description
uint16_t productCode	Product Code

```
res = goal_eipCfgProductCodeSet(0xABCD);
```

5.6.4 goal_eipCfgRevisionSet

Set the Revision of the EtherNet/IP stack instance. The revision consists of a major and a minor number. It represents changes in the firmware that affect the device's behavior.

Default Value: 1.1

Table 5.81 goal_eipCfgRevisionSet Parameters

Parameter	Description
uint8_t revMajor	major revision number
uint8_t revMinor	minor revision number

```
res = goal_eipCfgRevisionSet(1, 3);
```

5.6.5 goal_eipCfgSerialNumSet

Set the Serial Number of the EtherNet/IP stack instance. This number is assigned by the vendor and should be unique for each device.

Default Value: 1

Table 5.82 goal_eipCfgSerialNumSet Parameters

Parameter	Description
uint32_t serial	Serial Number

```
res = goal_eipCfgSerialNumSet(12345678);
```

5.6.6 goal_eipCfgProductNameSet

Set the Product Name of the EtherNet/IP stack instance. This name is assigned by the vendor. Its maximum length is 32 characters.

Default Value: “EtherNet/IP Adapter”

Table 5.83 goal_eipCfgProductNameSet Parameters

Parameter	Description
const char *strName	Product Name

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

5.6.7 goal_eipCfgDomainNameSet

Set the default Domain Name of the EtherNet/IP stack instance. This name is used as the device’s domain name if no valid configuration was found. Its maximum length is 48 characters.

Default Value: “port.de”

Table 5.84 goal_eipCfgDomainNameSet Parameters

Parameter	Description
const char *strName	default Domain Name

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

5.6.8 goal_eipCfgHostNameSet

Set the default Host Name of the EtherNet/IP stack instance. This name is used as the device’s host name if no valid configuration was found. Its maximum length is 64 characters.
Default Value: “eipdevice”

Table 5.85 goal_eipCfgHostNameSet Parameters

Parameter	Description
const char *strName	default Host Name

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

5.6.9 goal_eipCfgNumExplicitConSet

Set the maximum number of explicit Connections the device can handle at once.

Default Value: 6

Table 5.86 goal_eipCfgHostNameSet Parameters

Parameter	Description
uint16_t num	number of explicit connections

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

5.6.10 goal_eipCfgNumImplicitConSetImpl

Set the maximum number of implicit Connections the device can handle at once.

Default Value: 6

Table 5.87 goal_eipCfgNumImplicitConSetImpl Parameters

Parameter	Description
uint16_t num	number of implicit connections

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

5.6.11 goal_eipCfgEthLinkCountersOn

Enable support for Ethernet Link attributes 4, 5, 12 and 13. The hardware must support the appropriate counters.

Default Value: GOAL_TRUE

Table 5.88 goal_eipCfgEthLinkCountersOn Parameters

Parameter	Description
GOAL_BOOL_T enable	enable or disable feature

```
res = goal_eipCfgEthLinkCountersOn(GOAL_TRUE);
```

5.6.12 goal_eipCfgEthLinkControlOn

Enable support for Ethernet Link attribute 6. The hardware must support forced link speeds and duplex modes.

Default Value: GOAL_TRUE

Table 5.89 goal_eipCfgEthLinkControlOn Parameters

Parameter	Description
GOAL_BOOL_T enable	enable or disable feature

```
res = goal_eipCfgEthLinkControlOn(GOAL_TRUE);
```

5.6.13 goal_eipCfgChangeEthAfterResetOn

A change of Ethernet link speeds or duplex modes requires a reset of the device. The hardware must support forced link speeds and duplex modes.

Default Value: GOAL_FALSE

Table 5.90 goal_eipCfgEthLinkControlOn Parameters

Parameter	Description
GOAL_BOOL_T enable	enable or disable feature

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

5.6.14 goal_eipCfgChangeIpAfterResetOn

A change of the IP address requires a reset of the device.

Default Value: GOAL_FALSE

Table 5.91 goal_eipCfgChangeIpAfterResetOn Parameters

Parameter	Description
GOAL_BOOL_T enable	enable or disable feature

5.6.15 goal_eipCfgNumSessionsSet

Set the number of Encapsulation sessions the device can handle at once.

Default Value: 20

Table 5.92 goal_eipCfgNumSessionsSet Parameters

Parameter	Description
uint16_t num	number of sessions

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

5.6.16 goal_eipCfgTickSet

Set the number of milliseconds of one tick. The stack uses a tick as the smallest unit of time.

Default Value: 10

Table 5.93 goal_eipCfgTickSet Parameters

Parameter	Description
uint32_t ticks	size of 1 tick in ms

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

5.6.17 goal_eipCfgO2TRunIdleHeaderOn

Enable the Run/Idle Header for consumed (Originator-to-Target) cyclic data.

Default Value: GOAL_TRUE

Table 5.94 goal_eipCfgO2TRunIdleHeaderOn Parameters

Parameter	Description
GOAL_BOOL_T enable	enable or disable feature

5.6.18 goal_eipCfgT2ORunIdleHeaderOn

Enable the Run/Idle Header for produced (Target-to-Originator) cyclic data.

Default Value: GOAL_FALSE

Table 5.95 goal_eipCfgT2ORunIdleHeaderOn Parameters

Parameter	Description
GOAL_BOOL_T enable	enable or disable feature

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

5.6.19 goal_eipCfgQoSOn

Enable support of the QoS object.

Default Value: GOAL_TRUE

Table 5.96 goal_eipCfgQoSOn Parameters

Parameter	Description
GOAL_BOOL_T enable	enable or disable feature

```
res = goal_eipCfgQoSOn(GOAL_TRUE);
```

5.6.20 goal_eipCfgNumDelayedEncapMsgSet

Set the number of Encapsulation messages that can be delayed at the same time.

Default Value: 2

Table 5.97 goal_eipCfgNumDelayedEncapMsgSet Parameters

Parameter	Description
uint16_t num	number of messages

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

5.6.21 goal_eipCfgDhcpOn

Enable the DHCP client if it is supported by the platform.

Default Value: GOAL_FALSE

Table 5.98 goal_eipCfgDhcpOn Parameters

Parameter	Description
GOAL_BOOL_T enable	enable or disable feature

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

5.6.22 goal_eipCfgDlrOn

Enable support of the DLR object. This feature requires the DLR stack and support of the hardware.

Default Value: GOAL_FALSE

Table 5.99 goal_eipCfgDlrOn Parameters

Parameter	Description
GOAL_BOOL_T enable	enable or disable feature

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

5.7 Application Callbacks – EtherNet/IP

5.7.1 Introduction

To use the full potential of ***EtherNet/IP***, the starck allows you to interact at several stages of the protocol. For example, you may receive a callback after new assembly data has arrived.

The stack calls the callback handler registered via the function goal_eipNew(). The callback handler returns a GOAL_STATUS_T value.

Table 5.100 Parameters of the Callback Handler

Parameter	Description
GOAL_EIP_T *pHdlEip	GOAL EtherNet/IP handle
GOAL_EIP_CB_ID_T id	callback id
GOAL_EIP_CB_DATA_T *pCb	callback parameters

The callback parameter is an array of unions that has up to 10 elements.

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

5.7.2 GOAL_EIP_CB_ID_INIT

The stack was initialized. The application can initialize its resources.

Parameter <none>

Return Value

- GOAL_OK - success
- other - fail

5.7.3 GOAL_EIP_CB_ID_READY

Initialization is done.

Parameter <none>

Return Value <ignored>

5.7.4 GOAL_EIP_CB_ID_CONNECT_EVENT

Inform the application about a connection event

Parameter

Table 5.101 Callback Parameters of GOAL_EIP_CB_ID_CONNECT_EVENT

Element	Member	Data type	Description
0	outputAssembly	uint32_t	instance id of output assembly
1	inputAssembly	uint32_t	instance id of input assembly
2	connectionEvent	uint32_t	GOAL_EIP_CONNECTION_EVENT_*

Return Value <ignored>

5.7.5 GOAL_EIP_CB_ID_ASSEMBLY_DATA_RECV

New data for an assembly has been received.

Parameter

Table 5.102 Callback Parameters of GOAL_EIP_CB_ID_ASSEMBLY_DATA_RECV

Element	Member	Data type	Description
0	instanceNr	uint32_t	instance id of assembly

Return Value

- GOAL_OK - success
- Other - fail

5.7.6 GOAL_EIP_CB_ID_ASSEMBLY_DATA_SEND

Inform application that data of an assembly will be sent.

Parameter

Table 5.103 Callback Parameters of GOAL_EIP_CB_ID_ASSEMBLY_DATA_SEND

Element	Member	Data type	Description
0	instanceNr	uint32_t	instance id of assembly

Return Value

- GOAL_OK - data has changed
- Other - data has not changed

5.7.7 GOAL_EIP_CB_ID_RUN_IDLE_CHANGED

Inform application that the Run/Idle header of consumed cyclic data has changed.

Parameter

Table 5.104 Callback Parameters of GOAL_EIP_CB_ID_RUN_IDLE_CHANGED

Element	Member	Data type	Description
0	outputAssembly	uint32_t	instance id of output assembly
1	inputAssembly	uint32_t	instance id of input assembly
2	runIdleValue	uint32_t	current value of the run/idle flag

Return Value <ignored>

5.7.8 GOAL_EIP_CB_ID_LED_CHANGED

Inform the application that a Module Status or Network Status LED must be changed. The parameter contains a bitmap made of GOAL_EIP_LED_* macros. If a bit is set the corresponding LED must be set. Otherwise it must be cleared.

Parameter

Table 5.105 Callback Parameters of GOAL_EIP_CB_ID_LED_CHANGED

Element	Member	Data type	Description
0	leds	uint32_t	bitmap of active LEDs

Return Value <ignored>

5.7.9 GOAL_EIP_CB_ID_DEVICE_RESET

Inform the application that the device will be reset. Depending on the platform the device is either reset or the reset is only simulated. Therefore, the application must reinitialize its resources. The callback parameter indicates the reset type.

Parameter

Table 5.106 Callback Parameters of GOAL_EIP_CB_ID_DEVICE_RESET

Element	Member	Data type	Description
0	resetState	uint32_t	GOAL_EIP_RESET_*

Return Value <ignored>

6. Application Programming Interface

This section lists the API functions that are provided by the R-IN32M3 module.

6.1 Device Specific Functions

6.1.1 appl_ccmRpcInit

Purpose: Register R-IN32M3 module API in GOAL (appl_init)

This function registers the R-IN32M3 module specific API in GOAL and must be called in appl_init. It returns a GOAL_STATUS_T status and has no parameters.

Function Prototype:

```
1.  GOAL_STATUS_T goal_ddRpcInit(
2.      void
3.  );
```

Example:

```
24. /*****************************************************************************/
25. /** Application Init
26.  *
27.  * Build up the device structure and initialize the Profinet stack.
28.  */
29. GOAL_STATUS_T appl_init(
30.     void
31. )
32. {
33.     GOAL_STATUS_T res = GOAL_OK;              /* result */
34.
35.     /* initialize RPC interface */
36.     res = appl_ccmRpcInit();
37.     if (GOAL_RES_ERR(res)) {
38.         goal_logErr("Initialization of RPC failed");
39.     }
40.
41.     return res;
42. }
```

6.1.2 appl_ccmUpdateAllow

Purpose: Enable firmware update in the R-IN32M3 module

This function enables the possibility to update the firmware of the R-IN32M3 module. It returns a GOAL_STATUS_T status and has no parameters.

Function Prototype

```
1.  GOAL_STATUS_T appl_ccmUpdateAllow(
2.      void
```

```
3. );
```

Example:
For an example, where firmware update capability can be enabled and disabled using the web server please check example project 01pnio_io_mirror.

6.1.3 appl_ccmUpdateDeny

Purpose: Disable firmware update in the R-IN32M3 module

This function disables the possibility to update the firmware of the R-IN32M3 module. A possible use of this function is to disable firmware update possibilities during a cyclic communication relation.

It returns a GOAL_STATUS_T status and has no parameters.

Function Prototype:

```
1. GOAL_STATUS_T appl_ccmUpdateDeny(
2.     void
3. );
```

Example:

For an example, where firmware update capability can be enabled and disabled using the web server please check example project 01pnio_io_mirror.

6.1.4 appl_ccmInfo

Purpose: Get version and device information

This function reads information from the R-IN32M3 module:

- MAC address (serial number)
- Software version
- Device type

Function Prototype:

```
1. GOAL_STATUS_T appl_ccmInfo(
2.     char *strVersion,                    /**< target string for version */
3.     uint8_t lenStrVersion,               /**< length of str buffer */
4.     GOAL_ETH_MAC_ADDR_T *macAddress,     /**< mac address buffer */
5.     uint16_t *pDevType                   /**< device type */
6. );
```

Example:

```
1. /* 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_ccmUpdateDeny

Purpose: Set behavior of fieldbus communication on fault

This function determines the behavior of the R-IN32M3 module regarding cyclic communication. By default, a cyclic communication is stopped when communication to the application controller (AC) is lost.

Function Prototype:

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

Example:

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

6.1.6 appl_ccmCommResetSet

Purpose: Set behavior of the R-IN32M3 module on SPI sync reset request

A sync reset request is requested by the AC upon restart, if the CC had previously been set up by a running AC application.

This function determines the behavior of the R-IN32M3 module regarding this reset request. By default, no reset is done. If this option is enabled, reset is done.

The state of this setting is stored in non-volatile memory on the CC. A value of 0 disables the reset.

A value of 1 enables the reset.

Function Prototype:

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

Example:

```
1. /* set sync reset behavior */
2. if (GOAL_RES_OK(res)) {
3.     /* enable reset on sync reset request from AC */
4.     res = appl_ccmCommResetSet(1);
5. }
```

6.1.7 appl_ccmLogEnable

Purpose: Enable transport of AC log messages to the CC

Once this function is executed, log messages from the AC's logging buffer are continuously transferred to the CC module. Those are then accessible as the CC's own log messages through the management interface.

Function Prototype:

```
1. GOAL_STATUS_T appl_ccmLogEnable(
2.     void
3. );
```

Example:

```
1. /* enable logging to CC */
2. if (GOAL_RES_OK(res)) {
3.     res = appl_ccmLogEnable();
4. }
```

6.2 Device Detection

6.2.1 goal_ddInit - Register GOAL dd API in GOAL (appl_init)

Purpose: This function registers the GOAL dd specific API in GOAL and must be called in appl_init. It returns a GOAL_STATUS_T status and has no parameters.

Function Prototype:

```
1.  GOAL_STATUS_T goal_ddInit (
2.      void
3.  );
```

Example:

```
1.  GOAL_STATUS_T appl_init( void
2.  )
3.  {
4.      GOAL_STATUS_T res;                              /**< GOAL result */
5.
6.          /* initialize GOAL dd API */
7.          res = goal_ddInit();
8.          if (GOAL_RES_ERR(res)) {
9.              goal_logErr("failed to initialize GOAL dd API");
10.         }
11.     return res;
12. }
```

6.2.2 goal_ddNew - Register GOAL dd API in GOAL (appl_setup)

Purpose: This function creates an instance of GOAL dd in GOAL and must be called in appl_setup. A valid instance is requirement for using the GOAL dd API.

It returns a GOAL_STATUS_T status and has the following parameters.

Table 6.1 goal_ddNew Parameters

Parameter	Description
GOAL_DD_T **ppHdl	returned GOAL dd instance handle
uint32_t bitmaskFeatures	initial instance features to be enabled

The parameter bitmaskFeatures is a bitmask which disables single features of GOAL dd if set:

Table 6.2 Feature Bitmask Parameter

Bit	Feature
0	disable HELLO request (used for device scan detection)
1	disable WINK command
2	disable GETLIST command (read list of available CM variables)
3	disable GETCONFIG command (read cm variables value)
4	disable SETCONFIG command (write cm variables value)
5	disable SETIP command (configure IP through GOAL dd)

Function Prototype:

```
1.  GOAL_STATUS_T goal_ddNew(
2.      GOAL_DD_T **ppHdlDd,                /**< DD handle */
3.      uint32_t bitmaskFeatures            /**< initial features */
4.  );
```

Example:

```
1.      static GOAL_DD_T *pHdlDd;               /**< GOAL dd handle */
2.
3.      GOAL_STATUS_T appl_setup(
4.          void
5.      )
6.      {
7.          GOAL_STATUS_T res;                  /**< GOAL result */
8.              /* create GOAL dd instance */
9.              res = goal_ddNew(&pHdlDd, GOAL_DD_FEAT_ALL);
10.             if (GOAL_RES_ERR(res)) {
11.                 goal_logErr("failed to create GOAL dd instance");
12.             }
13.         return res;
14.     }
```

6.2.3 goal_ddCustomerIdSet - Configure the customer id of GOAL dd instance

Purpose: This function configures the customer Id of the given GOAL dd instance. The customer Id is a property of the underlying protocol which is contained in each request using GOAL dd. There is a special customer Id with value of zero, which causes every request to be processed. If the customer Id is not equal to zero, a request will only be processed if the customer Id of the request equals the customer Id of the GOAL dd instance.

The customer Id is stored in non-volatile memory.

Table 6.3 goal_ddCustomerIdSet Parameters

Parameter	Description
GOAL_DD_T *pHdl	GOAL dd instance handle
uint32_t customerId	instance customer Id

Function Prototype:

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

Example:

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

6.2.4 goal_ddModuleNameSet - Configure the name of GOAL dd instance

Purpose: This function configures the module name of the given GOAL dd instance. The module Id is a property of the device stored in non-volatile memory. It is used by the management tool for device naming. The module name length is limited to 20 bytes.

Table 6.4 goal_ddModuleNameSet Parameters

Parameter	Description
GOAL_DD_T *pHdl	GOAL dd instance handle
uint8_t *str	instance module name

Function Prototype:

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

Example:

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

6.2.5 goal_ddFeaturesSet - Configure the features of the GOAL dd instance

Purpose: This function configures the features to be disabled for the given GOAL dd instance. This property is stored in the device stored in non-volatile memory.

*/

Table 6.5 goal_ddFeaturesSet Parameters

Parameter	Description
GOAL_DD_T *pHdl	GOAL dd instance handle
uint32_t bitmaskFeatures	instance features bitmask

For parameter description see function “goal_ddNew”.

Function Prototype:

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

Example:

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

6.2.6 goal_ddCallbackReg - Configure callback for GOAL dd instance

Purpose: This function registers a callback to the given GOAL dd instance.

Type of the callback:

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

Function Prototype:

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

Example:

```
1.  /*****************************************************************************/
2.  /** Application Setup
3.   *
4.   * This function must setup all used protocol stacks.
5.   *
6.   * Notes if CSAP is active:
7.   *  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 - Temporary activation of features of GOAL dd instance

Purpose: This function temporarily enabled features for the given GOAL dd instance. This property is ***not*** stored in non-volatile memory.

Table 6.6 goal_ddSessionFeatureActivation Parameters

Parameter	Description
GOAL_DD_T *pHdl	GOAL dd instance handle
uint32_t bitmaskFeatures	instance features bitmask

ATTENTION: The parameter "bitmaskFeatures" is used to revert to the function for permanent configuration of the features, and a bit set here enables the given feature.

Function Prototype:

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

Example:

```
1.  /* temporarily enable capability to respond to hello requests (device detection) */
2.     res = goal_ddSessionFeatureActivte (pHdlDd, GOAL_DD_FEAT_HELLO)
3.  );
```

6.2.8 goal_ddFilterAdd - Limit access to CM variables

Purpose: By default, an external application as the management tool has total access to all CM variables of the device. This is a handy feature for development, but for production purpose one wants to limit access to only the variables that are required for minimal functionality using the management tool. Therefore, filters were introduced, which do this task. Following filters are predefined:

Table 6.7 goal_ddFilterAdd filter Sets

Filter ID	Filter Name
0	GOAL_DD_ACCESS_FILTER_SET_ALL
1	GOAL_DD_ACCESS_FILTER_SET_BASIC
2	GOAL_DD_ACCESS_FILTER_SET_HIDDEN

Table 6.8 goal_ddFilterAdd predefined Filters

Filter ID	Filter Actions	Purpose
0	Full access granted to all variables	Development
1	Read Access to all variables of the NET module (IP settings), Read Access to all variables of the ETH module (MAC, status), Full access to all variables of the LM module (logging)	Production Code with minimal support of the Management Tool
2	Disables read access to the web server authentication strings	Production Code

6.2.9 Definition of Filter – 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 Definition of Filter – 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 Definition of Filter – 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)
    },
};
```

Function Prototype:

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

Example:

```
    /* 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 Stack Application Programming Interface

6.3.1 goal_pnioInit - Register GOAL PROFINET in GOAL (appl_init)

This function registers GOAL PROFINET in GOAL and must be called in appl_init. Its main functionality is to register a set config manager variable before the NVS storage is initialized.

It returns a GOAL_STATUS_T status and has no parameters.

```
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 - Create a GOAL PROFINET Instance (appl_setup)

This function creates a new GOAL PROFINET instance by taking a snapshot of all previously defined variables (goal_pnioCfg API) and allocating the necessary resources. The application must provide a GOAL PROFINET instance id and a call-back handler. The call-back handler can also be NULL to only use GOAL PROFINET stack default behavior.

It returns a GOAL_STATUS_T status and a GOAL PROFINET instance handle.

Table 6.9 goal_pnioNew Parameters

Parameter	Description
GOAL_PNIO_T **ppPnio	returned GOAL PROFINET instance handle
const uint32_t id	GOAL PROFINET instance id
GOAL_PNIO_FUNC_CB_T pFunc	GOAL PROFINET callback function

6.3.3 goal_pnioCfgDcpFactoryResetDisableSet - Configure DCP Factory Reset

Controls if DCP factory reset is available. If set to GOAL_TRUE DCP factory reset will be denied. Default: GOAL_FALSE. It returns a GOAL_STATUS_T status.

This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Warning: This function should not be used as the behavior can break conformity.

Table 6.10 goal_pnioCfgDcpFactoryResetDisableSet Parameters

Parameter	Description
GOAL_BOOL_T flgDcpFactoryResetDisable	DCP Factory Reset Disable Flag

6.3.4 goal_pnioCfgDcpAcceptMixcaseStationSet - Configure DCP Mixcase Stationname Acceptance

Controls if DCP accepts station names with mixed case spelling. If set to GOAL_TRUE DCP accepts mixed case station names. Default: GOAL_FALSE.

It returns a GOAL_STATUS_T status.

This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Warning: This function should not be used as the behavior can break conformity.

Table 6.11 goal_pnioCfgDcpAcceptMixcaseStationSet Parameters

Parameter	Description
GOAL_BOOL_T flgDcpAcceptMixcaseStation	DCP Mixcase Stationname Acceptance Flag

6.3.5 goal_pnioCfgDevDapSimpleSet - Configure Device DAP Simple Mode

Configures the device DAP simple mode. If flgDevDapSimple is GOAL_TRUE the GOAL PROFINET stack automatically creates the DAP and port slots and modules.

Default: GOAL_TRUE

It returns a GOAL_STATUS_T status.

This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Warning: This function should be used carefully because it can break conformity.

Table 6.12 goal_pnioCfgDcpAcceptMixcaseStationSet Parameters

Parameter	Description
GOAL_BOOL_T flgDevDapSimple	Device DAP Simple Mode Flag

6.3.6 goal_pnioCfgDevDapApiSet - Set Device DAP API Number

Configures the device DAP API number. Default: 0

It returns a GOAL_STATUS_T status.

This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Warning: This function should be used carefully because it can break conformity.

Table 6.13 goal_pnioCfgDevDapApiSet Parameters

Parameter	Description
uint32_t idDevDapApi	Device DAP API Number

6.3.7 goal_pnioCfgDevDapSlotSet - Set Device DAP Slot Number

Configures the device DAP slot number. Default: 0

It returns a GOAL_STATUS_T status.

This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Warning: This function should be used carefully because it can break conformity.

Table 6.14 goal_pnioCfgDevDapSlotSet Parameters

Parameter	Description
uint16_t idDevDapSlot	Device DAP Slot Number

6.3.8 goal_pnioCfgDevDapSubslotSet - Set Device DAP Subslot Number

Configures the device DAP subslot number. Default: 1 It returns a GOAL_STATUS_T status.

This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Warning: This function should be used carefully because it can break conformity.

Table 6.15 goal_pnioCfgDevDapSubslotSet Parameters

Parameter	Description
uint16_t idDevDapSubslot	Device DAP Subslot Number

6.3.9 goal_pnioCfgDevDapModuleSet - Set Device DAP Module Id

Configures the device DAP module id. Default: 1 It returns a GOAL_STATUS_T status.

This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Warning: This function should be used carefully because it can break conformity.

Table 6.16 goal_pnioCfgDevDapModuleSet Parameters

Parameter	Description
uint32_t idDevDapMod	Device DAP Module Id

6.3.10 goal_pnioCfgDevDapSubmoduleSet - Set Device DAP Submodule Id

Configures the device DAP submodule id. Default: 1 It returns a GOAL_STATUS_T status.

This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Warning: This function should be used carefully because it can break conformity.

Table 6.17 goal_pnioCfgDevDapSubmoduleSet Parameters

Parameter	Description
uint32_t idDevDapSubmod	Device DAP Submodule Id

6.3.11 goal_pnioCfgNetLinkSafetySet - Configure Device Port Disable Behavior

Configures the device port disable behavior. If flgNetLinkSafety is set to GOAL_TRUE the GOAL PROFINET stack doesn't allow the master to disable all Ethernet interfaces at the same time.

Default: GOAL_TRUE It returns a GOAL_STATUS_T status.

This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Warning: This function should be used carefully because it can break conformity.

Table 6.18 goal_pnioCfgNetLinkSafetySet Parameters

Parameter	Description
GOAL_BOOL_T flgNetLinkSafety	Device Port Disable Behavior

6.3.12 goal_pnioCfgNewIoDataCbSet - Configure New IO Data Callback

Configures the new IO data callback. If flgCbNewIoData is set to GOAL_TRUE, the registered callback function is also called for each arrived cyclic frame. It must return a value as fast as possible as this happens in real time context.

Default: GOAL_FALSE It returns a GOAL_STATUS_T status.

This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Warning: If enabled it produces a very high application load as the callback is called 1000 times a second if the cycle time is set to 1 ms.

Table 6.19 goal_pnioCfgNewIoDataCbSet Parameters

Parameter	Description
GOAL_BOOL_T flgCbNewIoData	New IO Data Callback Flag

6.3.13 goal_pnioCfgDiagBufMaxCntSet - Configure Maximum Diagnosis Entries

Configures the maximum count of diagnosis entries.

Default: 20 It returns a GOAL_STATUS_T status.

This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Warning: Using a too small value can break conformity.

Table 6.20 goal_pnioCfgDiagBufMaxCntSet Parameters

Parameter	Description
unsigned int numDiagBufMax	Maximum Diagnosis Entries

6.3.14 goal_pnioCfgDiagBufMaxDataSizeSet - Configure Maximum Diagnosis Data Size

Configures the maximum size in bytes of each diagnosis entry. Default: 28 It returns a GOAL_STATUS_T status.

This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Warning: Using a value below 28 can break conformity.

Table 6.21 goal_pnioCfgDiagBufMaxDataSizeSet Parameters

Parameter	Description
unsigned int numDiagDataSizeMax	Maximum Diagnosis Data Size

6.3.15 goal_pnioCfglocrBlocksMaxSet – Configure Maximum IOCR Block Buffers

Configures the maximum count of IOCR block buffers. Default: 10 It returns a GOAL_STATUS_T status.

This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Warning: This function should be used carefully because it can break conformity.

Table 6.22 goal_pnioCfglocrBlocksMaxSet Parameters

Parameter	Description
unsigned int numIocrBlocksMax	Maximum IOCR Block Buffers

6.3.16 goal_pnioCfgCrMaxCntSet - Configure Maximum Communication Relation Count

Configures the maximum count of communication relations.

Default: 10 It returns a GOAL_STATUS_T status.

This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Warning: This function should be used carefully because it can break conformity.

Table 6.23 goal_pnioCfgCrMaxCntSet Parameters

Parameter	Description
unsigned int numCrMax	Maximum CR Count

6.3.17 goal_pnioCfgArMaxCntSet - Configure Maximum Application Relation Count

Configures the maximum count of application relations.

Default: 2; It returns a GOAL_STATUS_T status.

This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Warning: This function should be used carefully because it can break conformity.

Table 6.24 goal_pnioCfgArMaxCntSet Parameters

Parameter	Description
unsigned int numArMax	Maximum AR Count

6.3.18 goal_pnioCfgApiMaxCntSet - Configure Maximum API Count

Configures the maximum count of application process identifiers. Setting this value only affects the ModuleDiffBlock logic and needs to be big enough to hold the largest possible GSDML configuration.

Default: 1 It returns a GOAL_STATUS_T status.

This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Warning: This function should not be used as the behavior can break conformity and the GOAL PROFINET stack only supports API 0.

Table 6.25 goal_pnioCfgApiMaxCntSet Parameters

Parameter	Description
unsigned int numApiMax	Maximum API Count

6.3.19 goal_pnioCfgSlotMaxCntSet - Configure Maximum Slot Count

Configures the maximum count of slots. Setting this value only affects the ModuleDiffBlock logic and needs to be big enough to hold the largest possible GSDML configuration.

Default: 3; It returns a GOAL_STATUS_T status.

This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Table 6.26 goal_pnioCfgSlotMaxCntSet Parameters

Parameter	Description
unsigned int numSlotMax	Maximum Slot Count

6.3.20 goal_pnioCfgSubslotMaxCntSet - Configure Maximum Subslot Count

Configures the maximum count of subslots per slot. Setting this value only affects the ModuleDiffBlock logic and needs to be big enough to hold the largest possible GSDML configuration. Default: 6; It returns a GOAL_STATUS_T status.

This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Table 6.27 goal_pnioCfgSubslotMaxCntSet Parameters

Parameter	Description
unsigned int numSubslotMax	Maximum Subslot Count Per Slot

6.3.21 goal_pnioCfgSubslotIfSet - Configure Interface Subslot

Configures the interface subslot id.

Default: 0x8000 It returns a GOAL_STATUS_T status.

This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Warning: This function should be used carefully because it can break conformity.

Table 6.28 goal_pnioCfgSubslotIfSet Parameters

Parameter	Description
unsigned int idSubslotIf	Interface Subslot Id

6.3.22 goal_pnioCfgSubslotPortSet - Configure Port Subslot

Configures the port subslot base id.

Default: 0x8001 It returns a GOAL_STATUS_T status.

This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Warning: This function should not be used as the behavior can break conformity.

Table 6.29 goal_pnioCfgSubslotPortSet Parameters

Parameter	Description
unsigned int idSubslotPort	Port Subslot Id

6.3.23 goal_pnioCfgSnmpIdSet - Configure SNMP Instance Id

Configures the GOAL SNMP instance id that is used by the GOAL PROFINET stack. Default: GOAL_ID_INVALID

It returns a GOAL_STATUS_T status.

This function configures the GOAL PROFINET instance and must be called before goal_pnioNew to have an effect.

Table 6.30 goal_pnioCfgSnmpIdSet Parameters

Parameter	Description
idSnmp	GOAL SNMP Instance Id

6.3.24 goal_pnioSlotNew - Create a new slot

Create a new slot. Returns a GOAL_STATUS_T status.

Table 6.31 goal_pnioSlotNew Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET data
uint32_t idApi	API number (always 0)
uint32_t idSlot	slot number
GOAL_BOOL_T flgAutoGen	generate API automatically if not exist

```
/* 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 - Create a new subslot

Create a new subslot. Returns a GOAL_STATUS_T status.

Table 6.32 goal_pnioSubslotNew Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET data
uint32_t idApi	API number (always 0)
uint32_t idSlot	slot number
uint32_t idSubslot	subslot number
GOAL_BOOL_T flgAutoGen	generate API and slot automatically if not exist

```
/* 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);
```

See example 01_simple_io for a demonstration.

6.3.26 goal_pnioModNew - Create a new module

Create a new module. Returns a GOAL_STATUS_T status.

Table 6.33 goal_pnioModNew Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET data
uint32_t idMod	module id

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

6.3.27 goal_pnioSubmodNew - Create a new submodule

Create a new submodule. Returns a GOAL_STATUS_T status.

Table 6.34 goal_pnioSubmodNew Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET data
uint32_t idMod	module id
uint32_t idSubmod	submodule id
GOAL_PNIO_MOD_TYPE_T type	submodule type (input, output, both)
uint16_t sizeInput	size of input module
uint16_t sizeOutput	size of output module
GOAL_BOOL_T flgAutogen	generate module if not exist

```
/* 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 - Plug a module into a slot

Plug a module into a slot. Returns a GOAL_STATUS_T status.

Table 6.35 goal_pnioModPlug Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET data
uint32_t idApi	API (always 0)
uint16_t idSlot	slot number
uint32_t idMod	module id

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

6.3.29 goal_pnioSubmodPlug - Plug a submodule into a subslot

Plug a submodule into a subslot. Returns a GOAL_STATUS_T status.

Table 6.36 goal_pnioSubmodPlug Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET data
uint32_t idApi	API (always 0)
uint16_t idSlot	slot number
uint16_t idSubslot	subslot number
uint32_t idMod	module id
uint32_t idSubmod	submodule id

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

6.3.30 goal_pnioModPull - Pull a module from a slot

Pull a module from a slot. Returns a GOAL_STATUS_T status.

Table 6.37 goal_pnioModPull Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET data
uint32_t idApi	API (always 0)
uint16_t idSlot	slot number

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

6.3.31 goal_pnioSubmodPull - Pull a submodule from a subslot

Pull a submodule from a subslot. Returns a GOAL_STATUS_T status.

Table 6.38 goal_pnioSubmodPull Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET data
uint32_t idApi	API (always 0)
uint16_t idSlot	slot number
uint16_t idSubslot	subslot number

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

6.3.32 goal_pnioDataOutputGet - Get output data from a submodule

Get output data from a submodule. Returns a GOAL_STATUS_T status.

Table 6.39 goal_pnioDataOutputGet Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET data
uint32_t idApi	API (always 0)
uint16_t idSlot	slot number
uint16_t idSubslot	subslot number
char *pBuf	data buffer
uint16_t len	data buffer length
GOAL_PNIO_IOXS_T *pStateIops	pointer for producer state

```
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);
```

6.3.33 goal_pnioDataInputSet - Set input data for a submodule

Set input data for a submodule. Returns a GOAL_STATUS_T status.

Table 6.40 goal_pnioDataInputSet Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET data
uint32_t idApi	API (always 0)
uint16_t idSlot	slot number
uint16_t idSubslot	subslot number
char *pBuf	data buffer
uint16_t len	data buffer length
GOAL_PNIO_IOXS_T stateIops	producer state

```
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);
```

6.3.34 goal_pnioApduStatusGet - Get the application protocol data unit status

Get the application protocol data unit status. Returns a GOAL_STATUS_T status.

Table 6.41 goal_pnioApduStatusGet Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET data
GOAL_PNIO_AR_ID_T idAr	application relation id
GOAL_PNIO_APDU_STATUS_T *pStatusApdu	APDU status pointer

6.3.35 goal_pnioAlarmNotifySend - Send an alarm notification

Send an alarm notification. Returns a GOAL_STATUS_T status.

Table 6.42 goal_pnioAlarmNotifySend Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET data
uint16_t *pNrAlarmSeq	pointer to store alarm sequence number
GOAL_PNIO_AR_ID_T idAr	application relation id
uint32_t prio	priority
const GOAL_PNIO_ALARM_ NOTIFY_T *pAlarmNotify	pointer to alarm notification
uint16_t lenDataUser	user data length
void *pDataUser	pointer to user data

6.3.36 goal_pnioAlarmNotifySendAck - Send an alarm notification acknowledge

Send an alarm notification acknowledge. Returns a GOAL_STATUS_T status.

Table 6.43 goal_pnioAlarmNotifySendAck Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET data
GOAL_PNIO_AR_ID_T idAr	application relation id
uint32_t prio	priority
const GOAL_PNIO_ALARM_ NOTIFY_T *pAlarmNotify	pointer to alarm notification
GOAL_PNIO_STATUS_T *pStatus	PROFINET status
uint16_t lenDataUser	user data length
void *pDataUser	pointer to user data

6.3.37 goal_pnioAlarmProcessSend - Send a process alarm

Send a process alarm. Returns a GOAL_STATUS_T status.

Table 6.44 goal_pnioAlarmProcessSend Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET data
uint32_t api	API
uint16_t slot	slot number
uint16_t subslot	subslot number
uint16_t usi	user structure identifier
uint16_t lenData	data length
uint8_t *pData	pointer to user data

6.3.38 goal_pnioRecReadFinish - Answer a record read request

Answer a record read request. The sequence number is used to detect if the request has expired. If that is the case the response will be silently dropped. Returns a GOAL_STATUS_T status.

Table 6.45 goal_pnioRecReadFinish Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET data

int32_t hdl	record handle
GOAL_PNIO_STATUS_T *pStatusPnio	PROFINET status pointer
const uint8_t *pData	record data
uint16_t len	record data length
uint32_t numSeq	sequence number

6.3.39 goal_pnioRecWriteFinish - Answer a record write request

Answer a record write request. The sequence number is used to detect if the request has expired. If that is the case the response will be silently dropped. Returns a GOAL_STATUS_T status.

Table 6.46 goal_pnioRecWriteFinish Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET data
int32_t hdl	record handle
GOAL_PNIO_STATUS_T *pStatusPnio	PROFINET status pointer
uint32_t numSeq	sequence number

6.3.40 goal_pnioDiagExtChanDiagAdd - Add an extended channel diagnosis entry

Add an extended channel diagnosis entry. Returns a GOAL_STATUS_T status.

Table 6.47 goal_pnioDiagExtChanDiagAdd Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET data
GOAL_PNIO_DIAG_HANDLE_T *pHdl	pointer to store diagnosis handle
uint32_t api	API
uint16_t slot	slot number
uint16_t subslot	subslot number
uint16_t chan	channel index
uint16_t numErr	error number
uint16_t typeExtChanErr	extended channel error type
uint32_t valExtChanAdd	extended channel error value
GOAL_BOOL_T flgMaintReq	maintenance required flag
GOAL_BOOL_T flgMaintDem	maintenance demanded flag
GOAL_BOOL_T flgSubmitAlarm	submit alarm flag
uint16_t typeAlarm	alarm type

6.3.41 goal_pnioDiagChanDiagRemove - Remove a channel diagnosis entry

Remove a channel diagnosis entry. Returns a GOAL_STATUS_T status.

Table 6.48 goal_pnioDiagChanDiagRemove Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET data
GOAL_PNIO_DIAG_HANDLE_T hdl	diagnosis handle
GOAL_BOOL_T flgSubmitAlarm	submit alarm flag

6.3.42 goal_pnioCyclicCtrl - Control cyclic data received callback

Control cyclic data received callback. If flgCyclic is set to GOAL_TRUE then the callback GOAL_PNIO_CB_ID_NEW_IO_DATA signalizes the reception of new I/O data. This will lead to a much higher system load. For most applications it is sufficient to poll the cyclic data by goal_pnioDataOutputGet. Returns a GOAL_STATUS_T status.

Table 6.49 goal_pnioCyclicCtrl Parameters

Parameter	Description
GOAL_PNIO_T *pPnio	PROFINET data
GOAL_BOOL_T flgCyclic	cyclic data enable flag

6.4 EtherNet/IP Application Programming Interface

6.4.1 Introduction

This section lists the API functions that are provided by GOAL EtherNet/IP. See the applications on appl/goal_eip/01_simple_io or appl/goal_eip/02_dlr_dhcp for a demonstration of the code examples.

6.4.2 goal_eipInit

Initialize GOAL EtherNet/IP. Returns a GOAL_STATUS_T as result. This function must be called within the function appl_init(). No parameters required. res = goal_eipInit();

To enable EtherNet/IP support in GOAL, the goal_eipInit function must be called in the appl_init function. The creation of a new EtherNet/IP instance and its setup is done in appl_setup function.

```
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.  res = goal_eipInit();
7.  if (GOAL_RES_ERR(res)) { goal_logErr("Initialization of Ethernet/IP failed");
8.  }
9.  return res;
10. }
```

Example:

```
1.  /*****************************************************************************/
2.  /** Application Setup *
3.  * Setup the application.
4.  */

5.  GOAL_STATUS_T appl_setup( void
6.  )
7.  {
8.    GOAL_STATUS_T res;            /* result */
9.  /* for a real device the serial number should be unique per device */
10. res = goal_eipCfgSerialNumSet(123456789);
11. if (GOAL_RES_ERR(res)) { goal_logErr("failed to set Serial Number");
12. return res;
13. }
14. goal_logInfo("create new instance");
15. res = goal_eipNew(&pHdlEip, EIP_INSTANCE_DEFAULT, main_eipCallback);
16. if (GOAL_RES_ERR(res)) {
17. goal_logErr("failed to create a new EtherNet/IP instance");
18. return res;
19. }
20. res = main_eipApplInit(pHdlEip);
21. if (GOAL_RES_ERR(res)) {
22. goal_logErr("failed to initialize assembly and attribute configuration");
23. return res;
24. }
25. return GOAL_OK;
}
```

6.4.3 goal_eipNew

Create a GOAL EtherNet/IP instance for the given ID and register a callback. This function must be called within the function appl_setup().

Table 6.50 goal_eipNew Parameters

Parameter	Description
GOAL_EIP_T ** ppEip	[out] EtherNet/IP instance ref
const uint32_t id	ID of EtherNet/IP instance
GOAL_EIP_FUNC_CB_T pFunc	GOAL EtherNet/IP handle application callback

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

6.4.4 goal_eipCipClassRegister

Register an application specific CIP class. When the EtherNet/IP stack receives a request for the registered CIP class it is passed to the handler function.

Table 6.51 goal_eipCipClassRegister Parameters

Parameter	Description
GOAL_EIP_T *pHdlEip	GOAL EtherNet/IP handle
uint16_t classId	class ID to be registered
GOAL_EIP_REQ_HANDLER_T pFunc	request handler

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

6.4.5 goal_eipCreateAssemblyObject

Adds an CIP instance to the selected Class. Returns a GOAL_STATUS_T as result.

Table 6.52 goal_eipCreateAssemblyObject Parameters

Parameter	Description
GOAL_EIP_T *pHdlEip	GOAL EtherNet/IP handle
uint32_t instanceId	instance number of the assembly object to create
uint16_t len	length of the assembly object’s data

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

6.4.6 goal_eipAssemblyObjectGet

Get the pointer to the data array of an assembly.

Table 6.53 goal_eipAssemblyObjectGet Parameters

Parameter	Description
GOAL_EIP_T *pHdlEip	GOAL Ethernet/IP handle
uint32_t instanceId	instance number of the assembly object
uint8_t **ppData	[out] pointer to assembly data
uint16_t *pLen	[out] length of assembly data

```
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

Create one Exclusive Owner connection with producing and consuming endpoints.

Table 6.54 goal_eipAddExclusiveOwnerConnection Parameters

Parameter	Description
GOAL_EIP_T *pHdlEip	GOAL EtherNet/IP handle
uint32_t outputAssembly	the O-to-T point to be used for this connection
uint32_t inputAssembly	the T-to-O point to be used for this connection
uint32_t configAssembly	the configuration point to be used for this connection

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

6.4.8 goal_eipAddInputOnlyConnection

Create multiple Input Only connections with the same producing and consuming endpoints.

Table 6.55 goal_eipAddInputOnlyConnection Parameters

Parameter	Description
GOAL_EIP_T *pHdlEip	GOAL EtherNet/IP handle
uint32_t connNum	the number of the input only connection
uint32_t outputAssembly	the O-to-T point to be used for this connection
uint32_t inputAssembly	the T-to-O point to be used for this connection

uint32_t configAssembly	the configuration point to be used for this connection

```
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

Create multiple Listen Only connections with the same producing and consuming endpoints.

Table 6.56 goal_eipAddListenOnlyConnection Parameters

Parameter	Description
GOAL_EIP_T *pHdlEip	GOAL EtherNet/IP handle
uint32_t connNum	the number of the listen only connection
uint32_t outputAssembly	the O-to-T point to be used for this connection
uint32_t inputAssembly	the T-to-O point to be used for this connection
uint32_t configAssembly	the configuration point to be used for this connection

```
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

Get the EtherNet/IP version. Returns a GOAL_STATUS_T as result.

Table 6.57 goal_eipGetVersion Parameters

Parameter	Description
GOAL_EIP_T *pHdlEip	GOAL EtherNet/IP handle
char **ppVersion	[out] EtherNet/IP version

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

6.4.11 goal_eipDeviceStatusSet

Set device status bits. The parameter statusBits uses the GOAL_EIP_STATUS_ macros. They can be combined with binary OR.

Table 6.58 goal_eipDeviceStatusSet Parameters

Parameter	Description

GOAL_EIP_T *pHdlEip	GOAL EtherNet/IP handle
uint16_t statusBits	status bitmap

res = goal_eipDeviceStatusSet(pHdlEip, GOAL_EIP_STATUS_CFGRD);

6.4.12 goal_eipDeviceStatusClear

Clear device status bits. All bits of statusBits will be cleared. The parameter uses the GOAL_EIP_STATUS_ macros. They can be combined with binary OR.

Table 6.59 goal_eipDeviceStatusClear Parameters

Parameter	Description
GOAL_EIP_T *pHdlEip	GOAL EtherNet/IP handle
uint16_t statusBits	status bitmap

res = goal_eipDeviceStatusClear(pHdlEip,GOAL_EIP_STATUS_CFGRD);

6.5 Networking

6.5.1 goal_netRpcInit - Initialize RPC functionality for networking

Purpose: If networking functionality (IP settings, UDP channel, TCP channel) is required, this function needs to be called during application initialization.

Function Prototype:

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

Example:

```
/*****************************************************************************/
/** 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;
}
```

6.5.2 goal_maNetOpen - Open network

Purpose: Open the network media adapter (MA) for usage.

Function Prototype:

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

Example:

```
/*****************************************************************************/
/** 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;
}
```

6.5.3 goal_maNetClose - Close network

Purpose: Close the network media adapter (MA)

Function Prototype:

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

6.5.4 goal_maNetGetById - Get network media adapter (MA) handle

Purpose:
Get the network media adapter (MA) handle which was previously open for usage
Function Prototype:

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

Example:

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

6.5.5 goal_maNetIpSet – Set ip address

Purpose: Set the network interface IP address

Function Prototype:

```
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 */
);
```

Example:

```
/* set IP address */
ip = MAIN_APPL_IP;
nm = MAIN_APPL_NM;
gw = MAIN_APPL_GW;
res = goal_maNetIpSet(pMaNet, ip, nm, gw, GOAL_FALSE);
```

```
6.    if (GOAL_RES_ERR(res)) {
7.        goal_logErr("error while setting IP address");
8.        return res;
9.    }
```

6.6 TCP Channel

6.6.1 goal_maChanTcpOpen - Open the TCP channel media adapter (MA)

Purpose: Opens the networking media adapter (MA) for further usage. This needs to be done once at application startup.

Function Prototype:

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

Example:

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

6.6.2 goal_maChanTcpNew - Create a new TCP channel

Purpose: This function creates a new TCP channel.

Function Prototype:

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

Example:

```
/* 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_POR
  T + cnt);
    return res;
}
```

6.6.3 goal_maChanTcpActive - Activate a created TCP channel

Purpose: Activate a previously created tcp channel. Once it is activated, it established connection or accepts incoming connection requests.

Function Prototype:

```
GOAL_STATUS_T goal_maChanTcpActivate(
    GOAL_MA_CHAN_TCP_T *pChanTcpHdl,        /**< pointer to store CHAN_TCP handler */
```

```
3.    GOAL_NET_CHAN_T *pChanHandle                    /**< channel handle */
4.);
```

Example:

```
1./* activate channel */
2.res = goal_maChanTcpActivate(pMaTcp, pChanTcp);
3.if (GOAL_OK != res) {
4.    goal_logErr("error while enabling TCP channel");
5.    return;
6.}
```

6.6.4 goal_maChanTcpSetNonBlocking - Set channel to non-blocking

Purpose: Set socket to non blocking mode for reading

Function Prototype:

```
1.GOAL_STATUS_T goal_maChanTcpSetNonBlocking(
2.    GOAL_MA_CHAN_TCP_T *pChanTcpHdl,          /**< pointer to store CHAN_TCP handler */
3.    GOAL_NET_CHAN_T *pChanHandle,                    /**< channel handle */
4.    GOAL_BOOL_T flgOption                     /**< non blocking state */
5.);
```

Example:

```
1./* set TCP channel to non-blocking */
2.res = goal_maChanTcpSetNonBlocking(pMaTcp, pChanTcp, GOAL_TRUE);
3.if (GOAL_OK != res) {
4.    goal_logErr("error while setting TCP channel to non-blocking");
5.    return;
6.}
```

6.6.5 goal_maChanTcpGetRemoteAddr - Get remote address of TCP channel

Purpose: Get the ip address of the remote end point of the TCP channel.

Function Prototype:

```
1.GOAL_STATUS_T goal_maChanTcpGetRemoteAddr(
2.    GOAL_MA_CHAN_TCP_T *pChanTcpHdl,          /**< pointer to store CHAN_TCP handler */
3.    GOAL_NET_CHAN_T *pChanHandle,            /**< channel handle */
4.    GOAL_NET_ADDR_T *pAddr                  /**< remote address */
5.);
```

Example:

```
1./* get IP Address of remote node */
2.res = goal_maChanTcpGetRemoteAddr(pMaTcpHdl, pChan, &remote);
3.if (GOAL_RES_ERR(res)) {
4.    goal_logErr("Failed to get Remote Address for socket %p", (void *) pChan);
5.    return;
6.}
```

6.6.6 goal_maChanTcpSend - Send data through TCP channel

Purpose: Send data to a previously opened TCP channel.
Function Prototype:

```
1. GOAL_STATUS_T goal_maChanTcpSend(
2.     GOAL_MA_CHAN_TCP_T *pChanTcpHdl,          /**< pointer to store CHAN_TCP handler */
3.     GOAL_NET_CHAN_T *pChanHandle,             /**< channel handle */
4.     GOAL_BUFFER_T *pBuf                       /**< buffer with data to send */
5. );
```

Example:

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

6.7 UDP Channel

6.7.1 goal_maChanUdpOpen - Open the UDP channel MA

Purpose:
Open the UDP channel ma. This needs to be done once at application startup.

Function Prototype:

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

Example:

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

6.7.2 goal_maChanUdpGetById - Get the UDP channel MA handle

Purpose: This function gets the handle of the UDP channel ma which was previously opened.

Function Prototype:

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

6.7.3 goal_maChanUdpNew - Create a new UDP channel

Purpose: Create a new UDP channel.

Function Prototype:

```
    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 */
    );
```

Example:

```
    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, udpCall
    back);
```

```
5.  }
```

6.7.4 goal_maChanUdpClose - Close the UDP channel MA

Purpose: Close an existing channel

Function Prototype:

```
1.  GOAL_STATUS_T goal_maChanUdpClose(
2.      GOAL_MA_CHAN_UDP_T *pChanUdpHdl,        /**< pointer to store CHAN_UDP handler */
3.      GOAL_NET_CHAN_T *pChanHandle            /**< pointer to channel handle */
4.  );
```

6.7.5 goal_maChanUdpSetNonBlocking - Set the opened channel to non-blocking access

Purpose: Set a channel to non blocking operation.

Function Prototype:

```
1.  GOAL_STATUS_T goal_maChanUdpSetNonBlocking(
2.      GOAL_MA_CHAN_UDP_T *pChanUdpHdl,        /**< pointer to store CHAN_UDP handler */
3.      GOAL_NET_CHAN_T *pChanHandle,           /**< channel handle */
4.      GOAL_BOOL_T flgOption                   /**< option value */
5.  );
```

Example:

```
1.  res = goal_maChanUdpSetNonBlocking(pMaChanUdp, pChan2, GOAL_TRUE);
2.  if (GOAL_OK != res) {
3.      goal_logErr("error while setting UDP channel to non-blocking");
4.      return res;
5.  }
```

6.7.6 goal_maChanUdpSetBroadcast - Set the opened UDP channel to broadcast operation

Purpose: Set a channel to broadcast.
Function Prototype:

```
1.  GOAL_STATUS_T goal_maChanUdpSetBroadcast(
2.      GOAL_MA_CHAN_UDP_T *pChanUdpHdl,        /**< pointer to store CHAN_UDP handler */
3.      GOAL_NET_CHAN_T *pChanHandle,           /**< channel handle */
4.      GOAL_BOOL_T flgOption                   /**< option value */
5.  );
```

Example:

```
1.  /* enable broadcast reception */
2.  res = goal_maChanUdpSetBroadcast(pMaChanUdp, pChan1, GOAL_TRUE);
3.  if (GOAL_RES_ERR(res)) {
4.      goal_logErr("error while setting UDP channel to receive broadcasts");
5.      return res;
6.  }
```

6.7.7 goal_maChanUdpGetRemoteAddr - Get remote address of the UDP channel

Purpose: Get the remote address of a udp channel, and thus the address from which data were received.

Function Prototype:

```
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 */
);
```

6.7.8 goal_maChanUdpActivate - Activate a UDP channel

Purpose: Activate a channel

Function Prototype:

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

Example:

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

6.7.9 goal_maChanUdpSend - Send data to the UDP channel

Purpose: Send data to a open UDP channel.

Function Prototype:

```
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 */
);
```

Example:

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

7. Examples

7.1 01_pnio_io_mirror

7.1.1 Purpose

This example implements a PROFINET slave including a http management interface.

7.1.2 Configuration

By default, no IP address is set. Using a configuration tool (e.g. Management Tool) a valid configuration must be provided.

The device configuration is following:

Table 7.1 Example Configuration

Slot	Subslot	Module
1	1	„I Signed8“ – 1 Byte Input data
2	1	„O Signed8“ – 1 Byte Output data
3	1	„I Signed16“ – 2 Byte Input data
4	1	„O Signed16“ – 2 Byte Output data

The GSDML file for this example is located beside the application code:

"goal\appl\Renesas\ac\01_pnio_io_mirror\gsdml"

7.1.3 Usage Hints

This example shows how to implement a basic PROFINET slave. It also shows handling of process data by mapping the output data of module „O Signed8“ to the input data of module „I Signed8“ and mapping output data of module „O Signed16“ to the input data of the module “I Signed16”.

Besides that, a management interface under HTTP://<DEVICE-IP> is provided, where under “Device Management” access from the management tool or firmware update can be enabled and disabled.

This example also controls the LEDs on the R-IN32M3 module board.

7.2 02_eip_io_data

7.2.1 Purpose

This example implements an EtherNet/IP slave including a http management interface.

7.2.2 Configuration

This example requires a DHCP server to obtain an IP address.

The device configuration is following:

Table 7.2 Example EIP Configuration

Assembly ID	Size	Properties
150	32	Output Data
100	32	Input Data
151	10	Configuration Data

The EDS file for this example is located beside the application code:

"goal\appl\Renesas\ac\02_eip_io_data"

7.2.3 Usage Hints

This example shows how to implement a basic EtherNet/IP slave. It also shows handling of process data by mapping the output data of Assembly 150 to the input data of Assembly 100.

Besides that, a management interface under HTTP://<DEVICE-IP> is provided, where under “Device Management” access from the management tool or firmware update can be enabled and disabled.

This example also controls the LEDs on the R-IN32M3 module board.

7.3 05_pnio_01_simple_io

7.3.1 Purpose

This example implements a PROFINET.

7.3.2 Configuration

By default, no IP address is set. Using a configuration tool (e.g. Management Tool) a valid configuration must be provided.

The device configuration is following:

Table 7.3 Example PNIO Configuration

Slot	Subslot	Module
1	1	„64 bytes Input“ – 64 byte input data
2	1	„64 bytes Output“ – 64 Byte Output data

7.3.3 Usage Hints

This example shows how to implement a basic PROFINET slave. It also shows handling of process data by mapping the output data of module „64 bytes output“ to the input data of module „64 bytes input“.

Besides that, a management interface under HTTP://<DEVICE-IP> is provided, where under “Device Management” access from the management tool or firmware update can be enabled and disabled.

7.4 06_eip_io_data_static_ip

This example implements an EtherNet/IP slave including a http management interface.

7.4.1 Configuration

This example requires a static IP to be configured using the management tool. The device configuration is following:

Table 7.4 Example EIP IO Data Configuration

Assembly ID	Size	Properties
150	32	Output Data
100	32	Input Data
151	10	Configuration Data

The EDS file for this example is located beside the application code:

"goal\appl\Renesas\ac\02_eip_io_data"

7.4.2 Usage Hints

This example shows how to implement a basic EtherNet/IP slave. It also shows handling of process data by mapping the output data of Assembly 150 to the input data of Assembly 100.

Besides that, a management interface under HTTP://<DEVICE-IP> is provided, where under “Device Management” access from the management tool or firmware update can be enabled and disabled.

This example also controls the LEDs on the R-IN32M3 module board.

7.5 07_pnio_dsn

7.5.1 Purpose

This example implements a PROFINET slave using the PROFINET design tool. The project file is supplied.

7.5.2 Configuration

By default, no IP address is set. Using a configuration tool (e.g. Management Tool) a valid configuration must be provided.

The device configuration is following:

Table 7.5 Example PNIO DSN Configuration

Slot	Subslot	Module
1	1	„I Signed8“ – 1 Byte Input data
2	1	„O Signed8“ – 1 Byte Output data
3	1	„I Signed16“ – 2 Byte Input data
4	1	„O Signed16“ – 2 Byte Output data

The GSDML file for this example is located beside the application code:

"goal\appl\Renesas\ac\07_pnio_dsn\gsdml"

The Design tool project file is located beside the generated gsdml file.

7.5.3 Usage Hints

This example shows how to implement a basic PROFINET slave. It only contains the generated stub code from the design tool, therefore application functions need to be added (refer to example 01_pnio_io_mirror for that).

7.6 http_01_get

7.6.1 Purpose

This example shows usage of the web server. It shows implementing GET-request support.

7.6.2 Configuration

This example uses the configured IP address of the cc module. Check the management tool to get the IP address.

7.6.3 Usage Hints

Open the URL HTTP://<DEVICE-IP>, where a simple web page is shown.

7.7 http_02_post

7.7.1 Purpose

This example shows usage of the web server. It shows implementing POST-request support.

7.7.2 Configuration

This example uses the configured IP address of the cc module. Check the management tool to get the IP address.

7.7.3 Usage Hints

Open the URL HTTP://<DEVICE-IP>, where a simple web page is shown. There a small amount of data can be uploaded to the application.

7.8 http_03_list_res

7.8.1 Purpose

This example shows usage of the web server. It shows supporting of hierarchical URLs.

7.8.2 Configuration

This example uses the configured IP address of the cc module. Check the management tool to get the IP address.

7.8.3 Usage Hints

Open the URL HTTP://<DEVICE-IP>, where a simple web page is shown. It provides several sub-pages shown on the main page.

7.9 http_03_list_res

7.9.1 Purpose

This example shows usage of the web server. It shows supporting of basic authentication.

7.9.2 Configuration

This example uses the configured IP address of the cc module. Check the management tool to get the IP address.

7.9.3 Usage Hints

This example creates 4 pages with separate authentication data:

Table 7.6 Default Login and Password

URL	Credentials
HTTP://<DEVICE-IP>:8080	Index: Level 0
HTTP://<DEVICE-IP>:8080/page1.html	Page 1: Level 1
HTTP://<DEVICE-IP>:8080/page2.html	Page 2: Level 2
HTTP://<DEVICE-IP>:8080/page3.html	Page 3: Level 3

When opening one of the listed URLs the provided login credentials are required.

ATTENTION: This example modifies credentials of authentication level 0. If those settings are stored permanently in the CC module, firmware update will fail with default credentials. Reset settings to default value “” using the management tool.

7.10 http_05_template_cm

7.10.1 Purpose

This example shows usage of the web server. It shows implementing GET-request support with templating for CM variables.

7.10.2 Configuration

This example uses the configured IP address of the cc module. Check the management tool to get the IP address.

7.10.3 Usage Hints

Open the URL HTTP://<DEVICE-IP>:8080, where a simple web page is shown. It demonstrates template replacement of template usage with CM variable reference. It demonstrates the replacement of template usage as templates themselves with the use of reference to CM variable.
The string [CM:12, 0] will be replaced with the IP address of the device.

7.11 http_06_template_list

This example shows usage of the web server. It shows implementing GET-request support with templating for a list.

7.11.1 Configuration

This example uses the configured IP address of the cc module. Check the management tool to get the IP address.

7.11.2 Usage Hints

Open the URL HTTP://<DEVICE-IP>:8080, where a simple web page is shown. It demonstrates the replacement of template usage with list support.

7.12 http_07_template_table

7.12.1 Purpose

This example shows usage of the web server. It shows implementing GET-request support with templating for a table.

7.12.2 Configuration

This example uses the configured IP address of the cc module. Check the management tool to get the IP address.

7.12.3 Usage Hints

Open the URL http://<IP>:8081, where a simple web page is shown. It demonstrates the replacement of template usage with table support.

A table of values from various sensors is generated.

7.13 net_01_udp_receive

7.13.1 Purpose

This example shows usage of the UDP. It shows implementing an echo server on a specific UDP port.

7.13.2 Configuration

This example sets the IP address 192.168.0.100/24.

7.13.3 Usage Hints

Using netcat the UDP server can be tested. Just type some text, and the server will reply this message.

```
netcat -u 192.168.0.100 1234
5.1s □ Fr 18 Jan 2019 12:09:16 UTC
Testing the echo server using UDP!
Testing the echo server using UDP!
```

7.14 net_02_tcp_client

7.14.1 Purpose

This example shows usage of the TCP as a client. It shows how to connect to a TCP server.

7.14.2 Configuration

This example sets the IP address 192.168.0.100/24.

It expects a TCP server on a remote server with IP address 192.168.0.10 on port 1234.

7.14.3 Usage Hints

Using netcat a tcp server can be provided. Once the application runs, the client will send the following messages.

```
netcat -l 192.168.0.10 1234
Fr 18 Jan 2019 12:22:13 UTC
TESTSTRING 0TESTSTRING 1TESTSTRING 2TESTSTRING 3TESTSTRING 4⏎
```

7.15 net_03_tcp_server

7.15.1 Purpose

This example shows usage of the TCP as a server.

7.15.2 Configuration

This example sets the IP address 192.168.0.100/24.

7.15.3 Usage Hints

Using netcat a connection to the TCP server can be established. Just type some text, and the server will send the following messages.

```
netcat 192.168.0.100 1234
Fr 18 Jan 2019 12:22:13 UTC
Testing the echo server using TCP!
Testing the echo server using TCP!
```

8. Trouble Shooting

This section lists common pitfalls which possibly occur when using the R-IN32M3 module examples.

8.1 Startup Issues

In case an example application is started but the expected application behavior is not shown, please check the log:

```
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
```

If the last log entry regarding „sync needs local reset to proceed“ is shown, the communication controller needs a reset. This can be achieved using the „RST“ button on the R-IN32M3 module board.

8.2 Connection Issues

If the SPI communication is not working at all, this can be seen on the following final log message:

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

Please check the board connection and the proper execution of the application on the application controller.

8.3 IP Configuration

In the EtherNet/IP project, if the switch between static IP configuration and DHCP fails after reboot, please check the following CM variables using the management tool:

Table 8.1 IP Configuration

Module ID	Variable ID
GOAL_ID_NET	IP
GOAL_ID_NET	NETMASK
GOAL_ID_NET	GW
GOAL_ID_NET	VALID
GOAL_ID_NET	DHCP_ENABLED

To disable DHCP set the variable DHCP_ENABLED to 0. Make sure variable „VALID“ is set to 1. Upload these settings to the module and save those values permanently. After a reboot DHCP should be disabled.

8.4 Downgrade to Version 1.0

A downgrade to version 1.0 may fail if the AC application uses the reset input of the cc module. To successfully do a downgrade to the version 1.0, disable any AC connection and update the module in standalone mode.

Versions of the firmware after 1.0 mitigate the influence of an external reset during firmware update.

9. Renesas Synergy™

This section describes sample software for Synergy of the application controller.

9.1 Development Environment

- e2 studio Version 7.5.1
- SSP Version 1.7.0

9.1.1 Support Hardware

Renesas Synergy SK-S7G2

https://www.renesas.com/kr/en/products/synergy/hardware/kits/sk-s7g2.html

- SPI Channel: 8

Table 9.1 Pins Used on Renesas SK-S7G2 Board

Function	Pin
SPI MISO	IOPORT_PORT_01_PIN_04
SPI MOSI	IOPORT_PORT_01_PIN_05
SPI SCK	IOPORT_PORT_01_PIN_06
SPI CS	IOPORT_PORT_05_PIN_07
RESET	IOPORT_PORT_06_PIN_14

9.2 Adaption for Customer Hardware

To apply the provided R-IN32M3 module AC examples on customer hardware, at least the following properties must be adapted from those for this board.

Since the applications are based on the GOAL middleware, the appropriate way to adapt the examples to new hardware would be to create a board specific ***goal_target_board*** module.

The following subsections describe where adaption is required:

9.2.1 SPI Configuration

In ***goal_target_board.h*** there is a configuration option for the chip select pin of the SPI interface. This needs to be adapted to the target hardware:

```
/*****************************************************************************/
/* Configuration */
/*****************************************************************************/
#define GOAL_DRV_SPI_CS_PIN      IOPORT_PORT_05_PIN_07 /**< SPI chip select pin */
```

In ***goal_target_board.c*** within the function *goal_tgtBoardInit* the SPI driver is registered.

```
/* register SPI driver */
res = goal_drvSpiSynReg(GOAL_ID_DEFAULT, 8);
```

```
3.    if (GOAL_RES_ERR(res)) {
4.        goal_logErr("failed to register Synergy SPI driver");
5.        return res;
6.    }
```

The second parameter of *goal_drvSpiSyReg* defines the SPI channel to use.

Configuration of the SPI needs to be done in e2 studio:

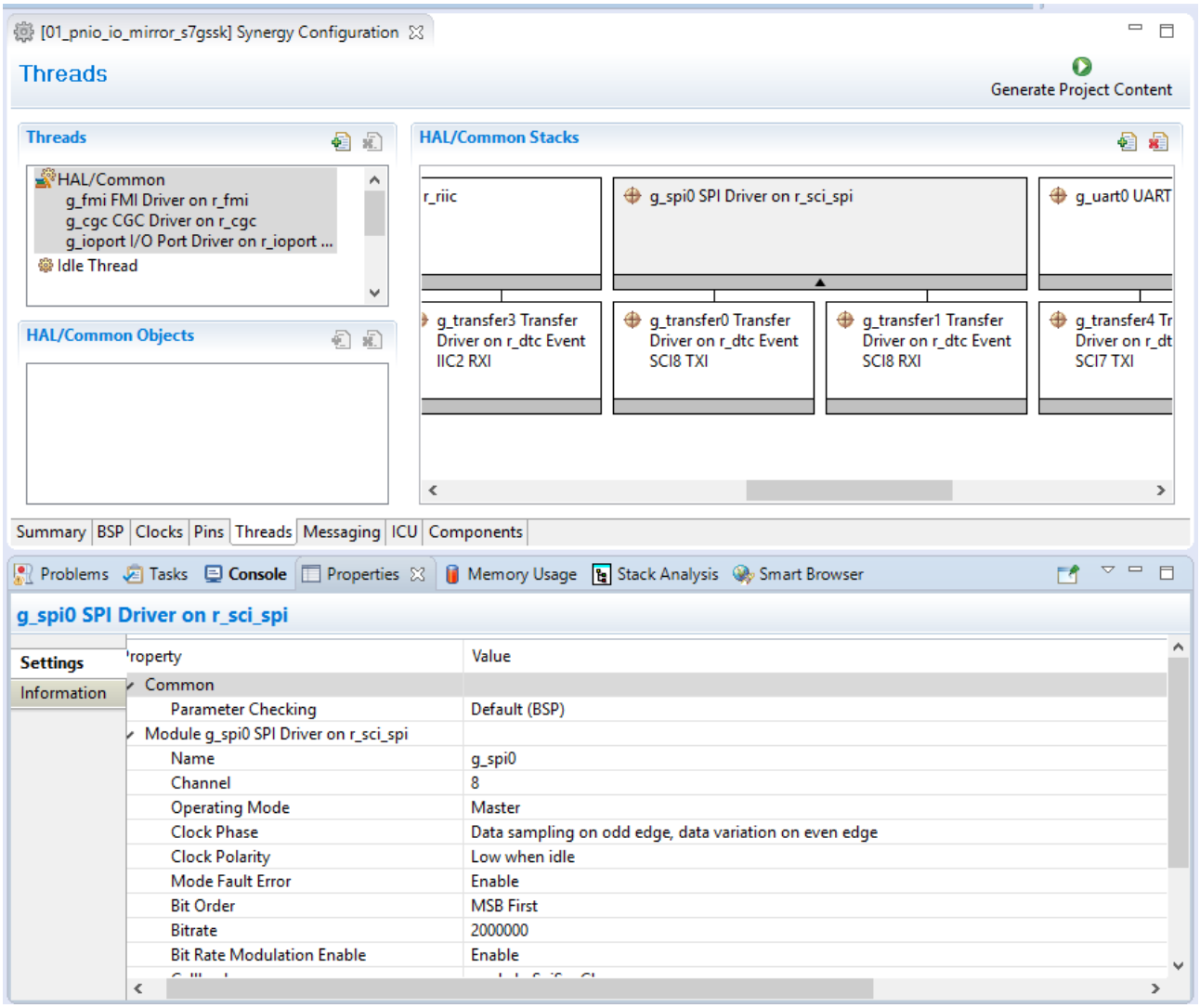

Figure 9.1 e2 studio SPI Properties

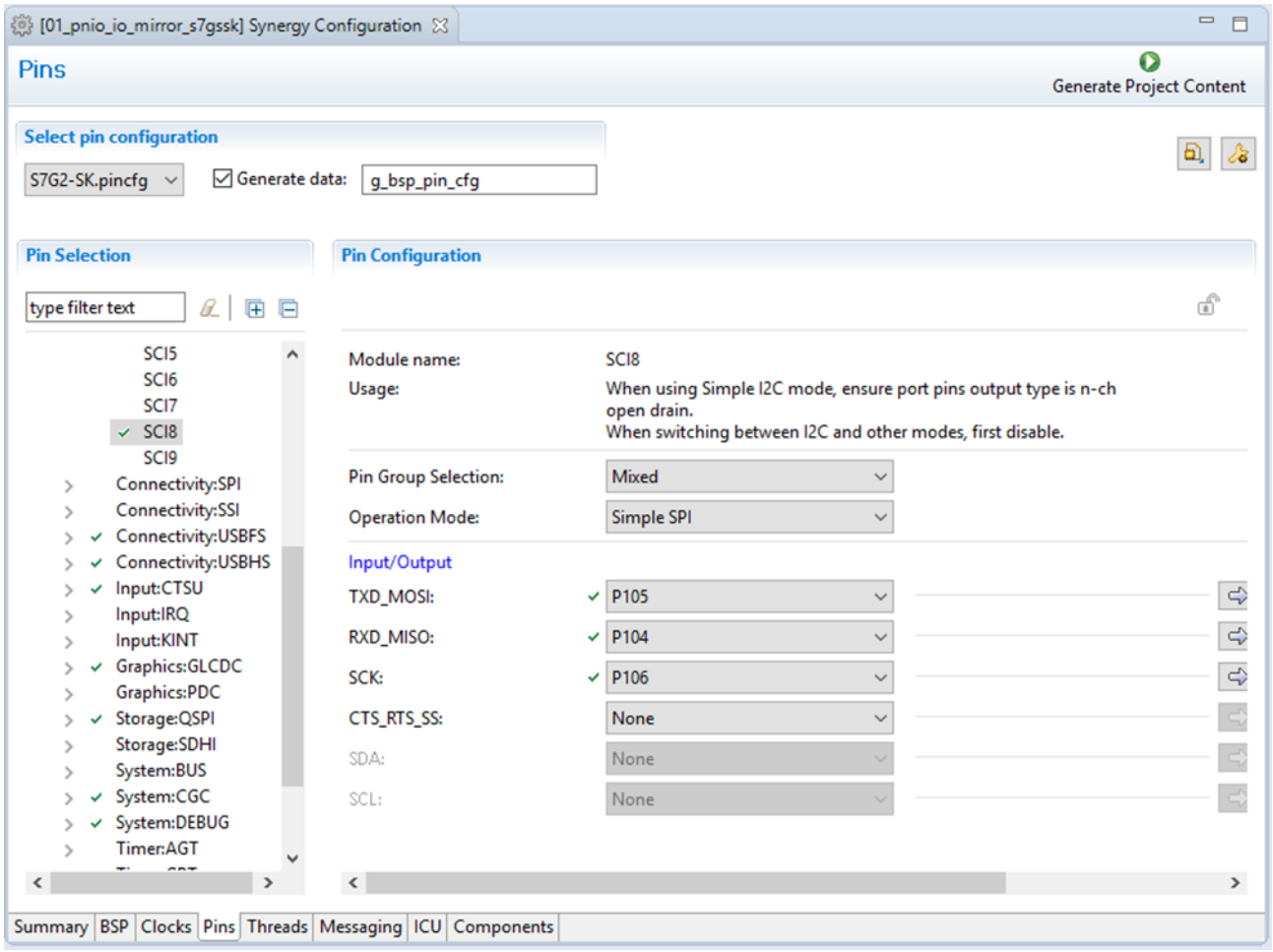

Figure 9.2 e2 studio SPI Pin Selection

9.2.2 LED Control

The GOAL middleware provides mechanism for LED control. The board files of the example code contain registration of a driver for the R-IN32M3 module board, which contains several LEDs. The driver, located in "plat\drv\led\iic\synergy7_irj45shield" controls the LEDs on this additional board, which is connected by an I2C bus.

```
#if GOAL_CONFIG_DRV_LED_IIC_SYNERGY7_IRJ45SHIELD == 1
    /* register LED driver */
    if (GOAL_RES_OK(res)) {
        res = goal_drvLedIicSynergy7Irj45shieldReg(
            LEDS_OFF, GOAL_ID_MA_LED_DEFAULT);
    }

    if (GOAL_RES_OK(res)) {
        res = goal_maLedGetById(&pMaLed, GOAL_ID_MA_LED_DEFAULT);
    }

    if (GOAL_RES_OK(res)) {
        goal_maLedRegisterLed(pMaLed, GOAL_MA_LED_LED1_RED, 0);
        goal_maLedRegisterLed(pMaLed, GOAL_MA_LED_LED1_GREEN, 2);
        goal_maLedRegisterLed(pMaLed, GOAL_MA_LED_LED2_RED, 4);
        goal_maLedRegisterLed(pMaLed, GOAL_MA_LED_LED2_GREEN, 6);
```

```
17.         goal_maLedRegisterLed(pMaLed, GOAL_MA_LED_ETHERCAT, 8);
18.         goal_maLedRegisterLed(pMaLed, GOAL_MA_LED_PROFINET, 10);
19.         goal_maLedRegisterLed(pMaLed, GOAL_MA_LED_MODBUS, 12);
20.         goal_maLedRegisterLed(pMaLed, GOAL_MA_LED_ETHERNETIP, 14);
21.         goal_maLedRegisterLed(pMaLed, GOAL_MA_LED_CANOPEN, 16);
22.     }
23. #endif
```

For a customer hardware a board specific driver should be implemented and registered accordingly.

9.2.3 Timer

The GOAL middleware requires a timer, which periodically calls the function goal_targetTimerCallback each millisecond. This timer is also initiated within the e2 studio project.

9.2.4 ThreadX

The e2 studio projects for the R-IN32M3 module requires HWOS features of ThreadX. To reach a specific resolution the ThreadX property “Timer Ticks Per Second” was increased to 1000.

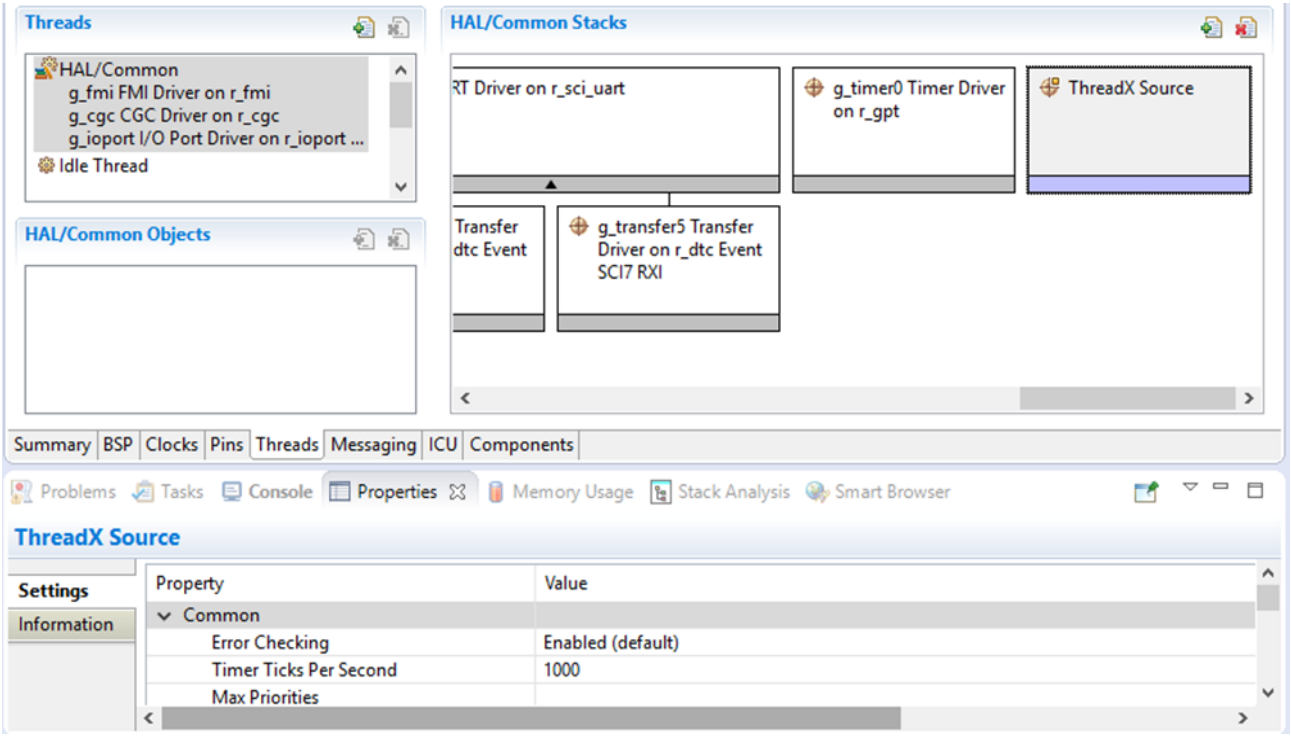

Figure 9.3 ThreadX Configuration

9.3 Logging

There are multiple possibilities for logging. Once the SPI link has been set up, thus RPC communication is established, log messages of the AC can be transferred to the CC. Therefore, these messages can be viewed by using the management tool. This is enabled by using an RPC function (see section 6.1.7, appl_ccmLogEnable).

Besides that, local logging is possible using a UART. For both supported SYNERGY based development boards a UART is provided on the R-IN32M3 module board. Please refer to the R-IN32M3 module board documentation for details.

The UART is initialized with a baud rate of 115200 bps, 8 data bits, no parity, 1 stop bit and no handshake. The connection provides both UART signals with 3.3 Volt level.

9.4 Object Dictionary

Default values are determined by the protocol stacks.

Module Id	Variable Name	Variable ID	Type	Max. Size	Long Diescrition
GOAL_ID_DD	MODULENAME	0	GOAL_CM_STRING	20	Customer specific name of the module
GOAL_ID_DD	CUSTOMERID	1	GOAL_CM_UINT32	4	Customer Id
GOAL_ID_DD	RESERVED	2	GOAL_CM_UINT8	1	—
GOAL_ID_DD	FEATURE_DISABLE	3	GOAL_CM_UINT32	4	Each bit disables a function: bit0, disable “HELLO DETECTION” bit1, disable WINK bit2, disable GETLIST bit3, disable GET VALUE bit4, disable SET VALUE
GOAL_ID_PNIO	STATION_NAME	0	GOAL_CM_GENERIC	255	
GOAL_ID_PNIO	VENDOR_ID	1	GOAL_CM_UINT16	2	
GOAL_ID_PNIO	DEVICE_ID	2	GOAL_CM_UINT16	2	
GOAL_ID_PNIO	VENDOR	3	GOAL_CM_GENERIC	255	
GOAL_ID_PNIO	IM0HWREV	4	GOAL_CM_UINT16	2	
GOAL_ID_PNIO	IM0SWREVPREF	5	GOAL_CM_UINT8	1	
GOAL_ID_PNIO	IM0SWREVENH	6	GOAL_CM_UINT8	1	
GOAL_ID_PNIO	IM0SWREVBUGFIX	7	GOAL_CM_UINT8	1	
GOAL_ID_PNIO	IM0SWREVINTCHG	8	GOAL_CM_UINT8	1	
GOAL_ID_PNIO	IM0SWREVCNT	9	GOAL_CM_UINT16	2	
GOAL_ID_PNIO	IM0PROFILEID	10	GOAL_CM_UINT16	2	
GOAL_ID_PNIO	IM0PROFILETYPE	11	GOAL_CM_UINT16	2	
GOAL_ID_PNIO	IM0ORDERID	12	GOAL_CM_GENERIC	(20 + 1)	
GOAL_ID_PNIO	IM0SERIALNR	13	GOAL_CM_GENERIC	(16 + 1)	
GOAL_ID_PNIO	IM1TAGFUNC	14	GOAL_CM_GENERIC	32	
GOAL_ID_PNIO	IM1TAGLOC	15	GOAL_CM_GENERIC	22	
GOAL_ID_PNIO	IM2DATA	16	GOAL_CM_GENERIC	16	
GOAL_ID_PNIO	IM3DESC	17	GOAL_CM_GENERIC	54	
GOAL_ID_PNIO	IM4SIG	18	GOAL_CM_GENERIC	54	
GOAL_ID_PNIO	HOLDFCT	19	GOAL_CM_UINT32	4	
GOAL_ID_PNIO	SYSCAP	20	GOAL_CM_UINT32	4	
GOAL_ID_PNIO	PORTDESC	21	GOAL_CM_GENERIC	LLDP_PORT_DESC_LEN	
GOAL_ID_PNIO	SYSDESC	22	GOAL_CM_GENERIC	LLDP_SYS_DESC_LEN	
GOAL_ID_PNIO	TXINTERV	23	GOAL_CM_UINT32	4	
GOAL_ID_PNIO	MANADDR	24	GOAL_CM_UINT32	4	
GOAL_ID_PNIO	SYSTEM_NAME	25	GOAL_CM_STRING	LLDT_SYS_NAME_LEN	
GOAL_ID_PNIO	IFSUBTYPE	26	GOAL_CM_UINT32	4	
GOAL_ID_PNIO	PDPORTDATA	27	GOAL_CM_GENERIC	sizeof(PN_REC_PDPORTDATA_CFT_T)	
GOAL_ID_LM	VERSION	0	GOAL_CM_UINT8	1	Version information for LM interface
GOAL_ID_LM	READBUFFER	1000	GOAL_CM_GENERIC	128	Buffer for reading online logging form deivce
GOAL_ID_LM	CNT	1001	GOAL_CM_UINT16	2	Control word for online log access
GOAL_ID_LM	EXLOG_READBUFFER	1002	GOAL_CM_GENERIC	128	Buffer for reading exception logging from deivce
GOAL_ID_LM	EXLOG_CNT	1003	GOAL_CM_UINT16	2	Control word for exception log access
GOAL_ID_LM	EXLOG_SIZE	1004	GOAL_CM_UINT32	4	Indicator for exception log size
GOAL_ID_LM	EXLOG_USAGE	1005	GOAL_CM_UINT8	1	Indicator for exception log usage
GOAL_ID_LM	EXLOG_ERASE	1006	GOAL_CM_UINT8	1	Command: *, Erase Exception Log
GOAL_ID_NET	IP	0	GOAL_CM_IPV4	4	IP address of first interface
GOAL_ID_NET	NETMASK	1	GOAL_CM_IPV4	4	NETMASK of first interface
GOAL_ID_NET	GW	2	GOAL_CM_IPV4	4	GATEWAY of first interface
GOAL_ID_NET	VALID	3	GOAL_CM_UINT8	1	Validity of IP address: 0, Stored IP address is not valid, interface settings originate from network stack of system 1, Stored IP address is valid, will be applied to interface at start of device

Module Id	Variable Name	Variable ID	Type	Max. Size	Long Diescrition
GOAL_ID_NET	DHCP_ENABLED	4	GOAL_CM_UINT8	1	DHCP enable: 0, DHCP disabled 1, DHCP enabled
GOAL_ID_NET	DHCP_STATE	5	GOAL_CM_UINT8	1	DHCP state: 0, DHCP initialized 1, DHCP server selecting 2, DHCP requesting configuration 3, DHCP ip address bound 4, DHCP renewing configuration 5, DHCP rebinding ip address to interface
GOAL_ID_NET	DNS0	6	GOAL_CM_IPV4	4	First DNS server of first interface
GOAL_ID_NET	DNS1	7	GOAL_CM_IPV4	4	Second DNS server of first interface
GOAL_ID_NET	HOSTNAME	8	GOAL_CM_STRING	20	Hostname of first interface
GOAL_ID_NET	COMMIT	1000	GOAL_CM_UINT8	1	Command: *, Apply IP settings
GOAL_ID_BOOT	SIGNATURE	0	GOAL_CM_GENERIC	16	Signature of booted image
GOAL_ID_BOOT	BLVERSION	1	GOAL_CM_STRING	16	Bootloader Version
GOAL_ID_BOOT	FWVERSION	2	GOAL_CM_STRING	16	Firmware Version
GOAL_ID_BOOT	RESET_CAUSE	1000	GOAL_CM_UINT8	1	Reset cause: 0, Unspecified 1, Firmware Update Requested 2, Watchdog 3, Firmware Commit Required 4, Reserved
GOAL_ID_BOOT	IMAGE_NUMBER	1001	GOAL_CM_UINT8	1	Booted image number
GOAL_ID_BOOT	IMAGE_COUNTER	1002	GOAL_CM_UINT8	1	Booted image number
GOAL_ID_CM	VERSION	0	GOAL_CM_UINT32	4	Version information for CM interface
GOAL_ID_CM	SAVE	1000	GOAL_CM_UINT	8	Command: *, Save CM to Flash
GOAL_ID_ETH	MAC	0	GOAL_CM_GENERIC	6	
GOAL_ID_ETH	LINK	1000	GOAL_CM_UINT32	4	Link status mask of interfaces
GOAL_ID_ETH	SPEED	1001	GOAL_CM_UINT32	4	Port speed mask of interfaces
GOAL_ID_ETH	DUPLEX	1002	GOAL_CM_UINT32	4	Port Duplex mask of interfaces
GOAL_ID_ETH	PORTCNT	1003	GOAL_CM_UINT32	4	Number of interfaces
GOAL_ID_PTP	PTPSTATE	0	GOAL_CM_GENERIC	(4 * 32)	
GOAL_ID_PTP	MPDSIGN	1	GOAL_CM_GENERIC	(4 * 1)	
GOAL_ID_PTP	MPDSEC	2	GOAL_CM_GENERIC	(4 * 8)	
GOAL_ID_PTP	MPDNSEC	3	GOAL_CM_GENERIC	(4 * 4)	
GOAL_ID_PTP	OFFSETFROMMASTERSIGN	4	GOAL_CM_UINT8	1	
GOAL_ID_PTP	OFFSETFROMMASTERSEC	5	GOAL_CM_GENERIC	8	
GOAL_ID_PTP	OFFSETFROMMASTERNSEC	6	GOAL_CM_UINT32	4	
GOAL_ID_GPTP	NEIGHBORRATERATIO	0	GOAL_CM_GENERIC	(4 * 4)	
GOAL_ID_GPTP	RATERATIO	1	GOAL_CM_INT32	4	
GOAL_ID_HTTP	HTTP_CHANNELS_MAX	0	GOAL_CM_UINT16	2	Determines the number of possible connections to the HTTP server
GOAL_ID_HTTP	HTTPS_CHANNELS_MAX	1	GOAL_CM_UINT16	2	Determines the number of possible connections to the HTTPS server
GOAL_ID_HTTP	USERLEVEL0	2	GOAL_CM_STRING	32	Authentication data for level 0
GOAL_ID_HTTP	USERLEVEL1	3	GOAL_CM_STRING	32	Authentication data for level 1
GOAL_ID_HTTP	USERLEVEL2	4	GOAL_CM_STRING	32	Authentication data for level 2
GOAL_ID_HTTP	USERLEVEL3	5	GOAL_CM_STRING	32	Authentication data for level 3
GOAL_ID_CSAP	CONFIG	0	GOAL_CM_GENERIC	1024	Configuration Storage Buffer
GOAL_ID_CSAP	MODE	1	GOAL_CM_UINT16	2	Reserved
GOAL_ID_QUEUE	SMALLBUFSIZE	0	GOAL_CM_INT16	2	Buffer size for small memory buffers
GOAL_ID_QUEUE	SMALLBUFNUM	1	GOAL_CM_INT16	2	Amount of small memory buffers reserved
GOAL_ID_QUEUE	MEDBUFSIZE	2	GOAL_CM_INT16	2	Buffer size for medium memory buffers
GOAL_ID_QUEUE	MEDBUFNUM	3	GOAL_CM_INT16	2	Amount of medium memory buffers reserved
GOAL_ID_QUEUE	BIGBUFSIZE	4	GOAL_CM_INT16	2	Buffer size for big memory buffers
GOAL_ID_QUEUE	BIGBUFNUM	5	GOAL_CM_INT16	2	Amount of big memory buffers reserved
GOAL_ID_EIP	INCARNATIONID	0	GOAL_CM_UINT32	4	Used to create unique Connection IDs across power cycles
GOAL_ID_EIP	DOMAIN	1	GOAL_CM_GENERIC	EIP_CM_DOMAIN_LEN	Part of TCP/IP Interface Object attribute 5

Module Id	Variable Name	Variable ID	Type	Max. Size	Long Diescrition
GOAL_ID_EIP	HOST	2	GOAL_CM_GENERIC	EIP_CM_HOST_LEN	TCP/IP Interface Object attribute 6
GOAL_ID_EIP	ENCAPTIMEOUT	3	GOAL_CM_UINT16	2	TCP/IP Interface Object attribute 13
GOAL_ID_EIP	PORTCFG	4	GOAL_CM_GENERIC	sizeof(OPENER_PORT_CFG_T)	Ethernet Link Object attributes 6 & 9 (Interface Control & Admin State)
GOAL_ID_EIP	QOS_VLAN	5	GOAL_CM_UINT8	1	QoS Object attribute 1
GOAL_ID_EIP	QOS_URGENT	6	GOAL_CM_UINT8	1	QoS Object attribute 4
GOAL_ID_EIP	QOS_SCHEDULED	7	GOAL_CM_UINT8	1	QoS Object attribute 5
GOAL_ID_EIP	QOS_HIGH	8	GOAL_CM_UINT8	1	QoS Object attribute 6
GOAL_ID_EIP	QOS_LOW	9	GOAL_CM_UINT8	1	QoS Object attribute 7
GOAL_ID_EIP	QOS_EXPLICIT	10	GOAL_CM_UINT8	1	QoS Object attribute 8
GOAL_ID_EIP	TTL	11	GOAL_CM_UINT8	1	TCP/IP Interface Interface Object attribute 8 TTL value of IP Header for Multicast Messages
GOAL_ID_EIP	MC_CTRL	12	GOAL_CM_UINT8	1	TCP/IP Interface Object attribute 9 Choose how to allocate Multicast Addresses 0: calculate address from device's IP address 1: use Mcast Start Address + offset
GOAL_ID_EIP	MC_NUM	13	GOAL_CM_UINT16	2	TCP/IP Interface Object attribute 9 Number of allocated Multicast Addresses for EtherNet/IP
GOAL_ID_EIP	MC_START	14	GOAL_CM_UINT32	4	TIP/IP Interface Object attribute 9 first allocated Multicast Adress
GOAL_ID_IRJ45	SPI_TYPE	0	GOAL_CM_UINT8	1	SPI Type (currently only slave supported): 0, SPI Master 1, SPI Slave
GOAL_ID_IRJ45	SPI_MODE	1	GOAL_CM_UINT8	1	SPI Mode: 0, CPOL=0; CPHA=0 1, SPOL=0; CPHA=1 2, SPOL=1, CPHA=0 3, CPOL=1; CPHA=1
GOAL_ID_IRJ45	SPI_SPEED	2	GOAL_CM_UINT8	1	SPI Speed in Master Mode
GOAL_ID_IRJ45	SPI_UNITWIDTH	3	GOAL_CM_UINT8	1	Bitsize of one single transfer unit: 0, 8 bits 1, 16 bits 2, 32 bits
GOAL_ID_IRJ45	SPI_BITORDER	4	GOAL_CM_UINT8	1	Bitorder of SPI transfers: 0, MSB first 1, LSB first
GOAL_ID_IRJ45	SPI_TRANSFERSIZE	5	GOAL_CM_UINT16	2	Minimum transfer size of single transmission frame
GOAL_ID_IRJ45	UPTIME	1000	GOAL_CM_UINT32	4	Number of seconds since start of device
GOAL_ID_HTTPS	TLS_SERVER_CERTIFICATE	0	GOAL_CM_GENERIC	1024	TLS server certificate used for HTTPS
GOAL_ID_HTTPS	TLS_PRIVATE_KEY	1	GOAL_CM_GENERIC	1024	TLS server private key used for HTTPS
GOAL_ID_HTTPS	TLS_SRV_CERT_CA_CN	2	GOAL_CM_STRING	128	CA common name
GOAL_ID_HTTPS	TLS_SRV_CERT_CA_O	3	GOAL_CM_STRING	128	CA organization
GOAL_ID_HTTPS	TLS_SRV_CERT_CA_C	4	GOAL_CM_STRING	8	CA country
GOAL_ID_HTTPS	TLS_SRV_CERT_CN	5	GOAL_CM_STRING	128	Subject common name
GOAL_ID_HTTPS	TLS_SRV_CERT_O	6	GOAL_CM_STRING	128	Subject organization
GOAL_ID_HTTPS	TLS_SRV_CERT_C	7	GOAL_CM_STRING	8	Subject country
GOAL_ID_HTTPS	TLS_SRV_CERT_NOT_BEFORE	8	GOAL_CM_STRING	20	TLS server cert valid not before
GOAL_ID_HTTPS	TLS_SRV_CERT_NOT_AFTER	9	GOAL_CM_STRING	20	TLS server cert valid not after
GOAL_ID_MI_MCTC	PRC_DISABLE	0	GOAL_CM_UINT8	4	RPC mode: 0, Enabled 1, Disabled
GOAL_ID_MI_MCTC	ID	1000	GOAL_CM_UINT32	4	Instance ID
GOAL_ID_MI_MCTC	STAT_RESET	1001	GOAL_CM_UINT32	4	Number of MCTC communications resets
GOAL_ID_MI_MCTC	STAT_RPC_TIMEOUTS	1002	GOAL_CM_UINT32	4	Number of MCTC RPC timeouts
GOAL_ID_MI_MCTC	STAT_RPC_DELAY_MIN	1003	GOAL_CM_UINT32	4	Minimum time for RPC request
GOAL_ID_MI_MCTC	STAT_RPC_DELAY_MAX	1004	GOAL_CM_UINT32	4	Maximum time for RPC request
GOAL_ID_MI_MCTC	STAT_RPC_DELAY_MEAN	1005	GOAL_CM_UINT32	4	Median time for RPC request
GOAL_ID_MI_MCTC	STAT_RPC_COUNT	1006	GOAL_CM_UINT32	4	Number of RPC requests
GOAL_ID_PNIO_TETO	DEVICE_ETH_MAC	0	GOAL_CM_GENERIC	6	
GOAL_ID_PNIO_TETO	STAT_CYCLIC_TOT_MIN	1000	GOAL_CM_UINT32	4	

Module Id	Variable Name	Variable ID	Type	Max. Size	Long Diescrition
GOAL_ID_PNIO_TETO	STAT_CYCLIC_TOT_MAX	1001	GOAL_CM_UINT32	4	
GOAL_ID_PNIO_TETO	STAT_CYCLIC_MIN	1002	GOAL_CM_UINT32	4	
GOAL_ID_PNIO_TETO	STAT_CYCLIC_MAX	1003	GOAL_CM_UINT32	4	
GOAL_ID_PNIO_TETO	STAT_CYCLIC_LAST	1004	GOAL_CM_UINT32	4	
GOAL_ID_SNMP	MIB2_SYS_LOCATION	0	GOAL_CM_STRING	255	
GOAL_ID_SNMP	MIB2_SYS_CONTACT	1	GOAL_CM_STRING	255	
GOAL_ID_SNMP	MIB2_SYS_NAME	2	GOAL_CM_STRING	255	
GOAL_ID_PNIO_TETO	DEVICE_PNIO_NAME	1	GOAL_CM_STRING	255	
GOAL_ID_PNIO_TETO	DEVICE_PNIO_RED_RATIO	2	GOAL_CM_UINT16	2	Device reduction ratio (cycle time): 1, 1 ms 2, 2 ms 4, 4 ms 8, 8 ms 16, 16 ms 32, 32 ms 64, 64 ms 128, 128 ms 256, 256 ms 512, 512 ms
GOAL_ID_PNIO_TETO	DEVICE_PNIO_DATA_HOLD_FACT	3	GOAL_CM_UINT16	2	Device data hold factor (watchdog factor): 1, 1 (inofficial) 2, 2 (inofficial) 3, 3 (default) 4, 4 8, 8 16, 16 32, 32
GOAL_ID_PNIO_TETO	DEVICE_NET_IP	4	GOAL_CM_IPV4	4	
GOAL_ID_PNIO_TETO	DEVICE_NET_NM	5	GOAL_CM_IPV4	4	
GOAL_ID_PNIO_TETO	DEVICE_NET_GW	6	GOAL_CM_IPV4	4	

10. Webserver

10.1 General

GOAL provides a smart webserver for embedded systems. The webserver was designed:

- for file downloads and
- to get information about the current device state and properties.

The webserver supports the following properties:

transfer protocols: HTTP / HTTPS

request methods: GET / POST

One or more webpages can be assigned to one instance of the webserver. The webpages are part of the application and must be made available by the application. The web-server provides a callback function for this purpose, see cbHttpReqFunc() in section 10.5.

The current device state and properties can be read from CM-variables and application-specific variables. The application-specific variables can be organized as simple variables or as a one-dimensional list.

It is possible to store templates for webpages with placeholders for current values of application-specific variables. The text substitutions are described in section 10.3. Web-templates make the dynamic management of web-pages possible.

The access to the webserver can be limited by user levels. The application can specify, which user levels shall be supported by the device and which rights the user levels shall have. The authentication data consisting of username and the password for each user level are configurable by CM-variables. The GOAL webserver provides up to 4 user levels.

The user levels can be applied by all instances of the webserver. For each instance of the webserver the valid user level can be specified during registration. Web-requests are only transferred to the application after a successful authentication, i.e. the callback function cbHttpReqFunc() in section 10.5 is only called after a successful login. The transfer of the username and the password via a web-server instance using the HTTP transfer protocol is unsafe. We recommend a port on which the HTTPS transfer protocol is running.

HTTPS is activated by the compiler-define GOAL_CONFIG_HTTPS. HTTPS uses the external software component mbedTLS for encoding and authentication. The access to mbedTLS is realized with the media adapter for TLS. TLS for HTTPS is initialized and opened by function goal_httpsNew().

About the CM-variables for HTTPS it is possible to install a private key and one's own X509-certificate. If no own certificate is stored, the webserver takes a default certificate provided by a port.

example:

- …\goal\appl\goal_http\01_get*:

for upload of a webpage

- …\goal\appl\goal_http\05_template_cm*:

for upload of a web-template with CM-variables and application-specific variables

- …\goal\appl\goal_http\06_template_list*:

for upload of a web-template with lists

- …\goal\appl\goal_http\04_auth*:

for authentication about user levels

- …\goal\appl\goal_http\02_post*:

for file download

10.2 Configuration

10.2.1 Compiler-defines

The following compiler-defines are available to configure the webserver:

GOAL_CONFIG_HTTP:
0: Transfer protocol HTTP is not used (default)
1: Transfer protocol HTTP is used

GOAL_CONFIG_HTTPS:
0: Transfer protocol HTTPS is not used (default)
1: Transfer protocol HTTPS is used

10.2.2 CM-variables

The following CM-variables are available to configure the webserver:

CM-Module-ID	GOAL_ID_HTTPD
CM-variable-ID	0
CM-variable name	HTTPD_CM_VAR_HTTPD_CHANNELS_MAX
Description	Maximal number of connections available for the HTTP transfer protocol
CM data type	GOAL_CM_UINT16
Size	2 bytes

Default value	From NVS or 0

CM-Module-ID	GOAL_ID_HTTPD
CM-variable-ID	1
CM-variable name	HTTPD_CM_VAR_HTTPS_CHANNELS_MAX
Description	Maximal number of connections available for the HTTPS transfer protocol
CM data type	GOAL_CM_UINT16
Size	2 bytes
Default value	From NVS or 0

CM-Module-ID	GOAL_ID_HTTPD
CM-variable-ID	2
CM-variable name	HTTPD_CM_VAR_USERLEVEL0
Description	Authentication data for level 0
CM data type	GOAL_CM_STRING
Size	32 bytes
Default value	From NVS or an empty string

CM-Module-ID	GOAL_ID_HTTPD
CM-variable-ID	3
CM-variable name	HTTPD_CM_VAR_USERLEVEL1
Description	Authentication data for level 1
CM data type	GOAL_CM_STRING
Size	32 bytes
Default value	From NVS or an empty string

CM-Module-ID	GOAL_ID_HTTPD
CM-variable-ID	4
CM-variable name	HTTPD_CM_VAR_USERLEVEL2
Description	Authentication data for level 2
CM data type	GOAL_CM_STRING
Size	32 bytes
Default value	From NVS or an empty string

CM-Module-ID	GOAL_ID_HTTPD
CM-variable-ID	5
CM-variable name	HTTPD_CM_VAR_USERLEVEL3
Description	Authentication data for level 3
CM data type	GOAL_CM_STRING
Size	32 bytes
Default value	From NVS or an empty string

The following CM-variables allow to configure the TLS layer used by HTTPS:

CM-Module-ID	GOAL_ID_HTTPS
CM-variable-ID	0
CM-variable name	HTTPS_CM_VAR_TLS_SERVER_CERTIFICATE
Description	Certificate of the webserver
CM data type	GOAL_CM_GENERIC
Size	1024 bytes
Default value	From NVS or certificate from a port

CM-Module-ID	GOAL_ID_HTTPS
CM-variable-ID	1
CM-variable name	HTTPS_CM_VAR_TLS_PRIVATE_KEY
Description	Private key of the webserver
CM data type	GOAL_CM_GENERIC
Size	1024 bytes
Default value	From NVS or an empty entry

CM-Module-ID	GOAL_ID_HTTPS
CM-variable-ID	2
CM-variable name	HTTPS_CM_VAR_TLS_SRV_CERT_CA_CN
Description	Common name of the server of the certification authority
CM data type	GOAL_CM_STRING
Size	128 bytes
Default value	From NVS or an empty string

CM-Module-ID	GOAL_ID_HTTPS
CM-variable-ID	3
CM-variable name	HTTPS_CM_VAR_TLS_SRV_CERT_CA_O
Description	Name of the certification authority organization, e.g. the company name
CM data type	GOAL_CM_STRING
Size	128 bytes
Default value	From NVS or an empty string

CM-Module-ID	GOAL_ID_HTTPS
CM-variable-ID	4
CM-variable name	HTTPS_CM_VAR_TLS_SRV_CERT_CA_C
Description	Country, in which the certification authority organization is located
CM data type	GOAL_CM_STRING
Size	8 bytes
Default value	From NVS or an empty string

CM-Module-ID	GOAL_ID_HTTPS
CM-variable-ID	5
CM-variable name	HTTPS_CM_VAR_TLS_SRV_CERT_CN
Description	Common name of the webserver
CM data type	GOAL_CM_STRING
Size	128 bytes
Default value	From NVS or an empty string

CM-Module-ID	GOAL_ID_HTTPS
CM-variable-ID	6
CM-variable name	HTTPS_CM_VAR_TLS_SRV_CERT_O
Description	Name of the organization provided the webserver
CM data type	GOAL_CM_STRING
Size	128 bytes
Default value	From NVS or an empty string

CM-Module-ID	GOAL_ID_HTTPS
CM-variable-ID	7
CM-variable name	HTTPS_CM_VAR_TLS_SRV_CERT_C
Description	Country, in which the organization provided the webserver is located
CM data type	GOAL_CM_STRING
Size	8 bytes
Default value	From NVS or an empty string

CM-Module-ID	GOAL_ID_HTTPS
CM-variable-ID	8
CM-variable name	HTTPS_CM_VAR_TLS_SRV_CERT_NOT_BEFORE
Description	From what date and time the certificate is valid
CM data type	GOAL_CM_STRING
Size	20 bytes
Default value	From NVS or an empty string

CM-Module-ID	GOAL_ID_HTTPS
CM-variable-ID	9
CM-variable name	HTTPS_CM_VAR_TLS_SRV_CERT_NOT_AFTER
Description	From what date and time the certificate is invalid
CM data type	GOAL_CM_STRING
Size	20 bytes
Default value	From NVS or an empty string

10.3 Web-templates

The GOAL webserver allows to implement templates for web-pages with placeholders for current information. The placeholders are substituted by the current values by the web-server during the upload process. The webserver provides placeholders for

- CM-variables,
- application-specific variables and
- lists.

10.3.1 CM-variables

The placeholder for CM-variables contains the CM-module-ID and the CM-variable-ID. The web-server executes the substitution of the placeholder by the CM-variable automatically.

syntax:

[CM:<modNum>, <cmVarNum>]

example:

[CM:0, 2]

10.3.2 Application-specific variables

The placeholder for application-specific variables contains the name of the variable in the application. The web-server requires the current value of the variable from the application to be obtained by calling a callback function (see cpHttpTemplateFunc() in section 10.5), and substitutes this for the placeholder on the web-page.

syntax:

[VAR:<applVarName>]

example:

[VAR:applVar]

10.3.3 Lists

The webserver provides an effective method to generate lists in HTML text. The HTML text for a single list entry can be enclosed in the placeholders FOREACH and /FOREACH in the web-template. FOREACH marks a one-dimensional list and the HTML text between the placeholders FOREACH and /FOREACH is executed for each list element. The place for the list entry is marked in the HTML text by the placeholder VAR with the desired variable name. After the substitution of the placeholder VAR the web-server changes to the next list entry automatically, i.e. it is not possible to substitute the same list entry twice. Therewith it is only necessary to describe the first list entry in the web-template.

The webserver only gets the ID and the name of the list and the number of list elements during the registration. The content of the list elements is managed by the application. The web-server calls a callback function to get the content of the next list element, see cpHttpTemplateFunc() in section 10.5.

Nested lists are allowed. The maximal supported nesting depth is 4.

syntax:

[FOREACH:<listName>] … [/FOREACH]

example for HTML listing:

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

Example …\goal\appl\goal_http\06_template_list* generates a HTML listing. The indication in the web-browser is shown in Figure 10.1:

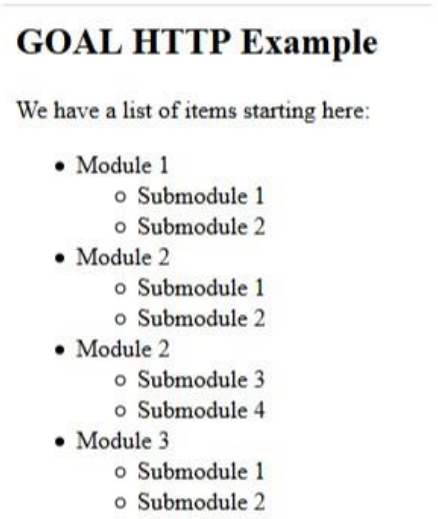

Figure 10.1 Web-page of Example 06_template_list

The placeholder [FOREACH] … [\FOREACH] can also be integrated in other HTML formatting like tables.

10.4 Characters

Square brackets: If the HTML text shall show square brackets, the square brackets must be written double, because placeholders in web-templates are bordered by square brackets.

Example: An array instruction shall be shown on a webpage.

HTML text: applArray[[5]]

Web-browser view: applArray[5]

Double quotes: Double quotes in the HTML-text must be protected by a backslash, because strings in the C code are enclosed by double quotes.

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

The rules for HTML text are valid for **all other characters**.

10.5 Callback Functions

Prototype	GOAL_STATUS_T cbHttpReqFunc(GOAL_HTTP_APPLCB_DATA_T *applData)	
Description	The received and valid web-request is passed to the application. The application has to process the web-request and to produce a web-response.	
Parameters	applData	Contains data for the application and data returned by the application.
Return values	GOAL return status	
Category	Optional	
Registration	During execution of the function goal_httpdResReg()	

Prototype	GOAL_STATUS_T cbHttpTemplateFunc(GOAL_HTTP_APPLCB_TEMPL_T *pWebTemplate)	
Description	About this callback function the application provides the current information for the specified placeholder. This callback function is called for each placeholder in the web-template. If there are multiple placeholders for the same information, this callback function is called for each placeholder.	
Parameters	pWebTemplate	Contains the information to specify a variable or list and the current return value of the specified variable.
Return values	GOAL return status	
Category	Optional	
Registration	During execution of the function	

10.6 Implementation Guideline

10.6.1 Upload a webpage

The webpage “device.html” shall be uploaded from the device to the web-browser. The content of the web-page is stored in variable webPage[] as string. No text substitutions on the webpage are necessary.

1. Initialize the webserver in state GOAL_FSA_INIT_APPL or GOAL_FSA_INIT_GOAL:

```
goal_httpInit();
```

2. Open 1 instance of the web-server using the TCP port 8080 in state GOAL_FSA_INIT_SETUP:

```
GOAL_HTTP_T *pWebInstanceHdl;       /* handle for the web-server instance */
goal_httpNew(&pWebInstanceHdl, 8080, 1);
```

3. Register the callback-functions and allowed methods for the opened web-server in state

GOAL_FSA_INIT_SETUP: No callback function for text substitution is registered.

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

4. Provide the web-page as string variable:

```
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 middleware of port.
</p> \r\n\
</body></html>”
```

5. Implement a callback function to process web-requests:

```
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. The callback function applWebReqCb() is called by the web-server after a web-request is received.

10.6.2 Upload a webpage

The webpage “device.html” shall be uploaded from the device to the web-browser. The web-page bases on the template webPage[]. The template includes a placeholder for the CM-Variable DD_CM_VAR_MODULENAME with the CM-module-ID 34 and the CM-variable-ID 0. The webserver substitutes the placeholder by the current value of the CM-variable automatically. No text substitutions on the webpage are necessary.

1. Initialize the webserver in state GOAL_FSA_INIT_APPL or GOAL_FSA_INIT_GOAL:

```
goal_httpInit();
```

2. Open 1 instance of the webserver using the TCP port 8080 in state GOAL_FSA_INIT_SETUP:

```
GOAL_HTTP_T *pWebInstanceHdl;       /* handle for the web-server instance */
goal_httpNew(&pWebInstanceHdl, 8080, 1);
```

3. Register the callback-functions and allowed methods for the opened webserver in state GOAL_FSA_INIT_SETUP: No callback function for text substitution is registered.

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

4. Provide a template for the webpage as string variable with placeholder for the CM-variable:

```
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 of port.</p> \r\n\
</body></html>”
```

5. Implement a callback function to process web-requests:

```
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. The callback function applWebReqCb() is called by the web-server after a web-request is received.

10.6.3 Read application specific variable

The webpage “device.html” shall be uploaded from the device to the web-browser. The webpage bases on the template webPage[]. The template includes a placeholder for the application-specific Variable applVar. The web-server calls the callback function applWebGetValCb() to provide the current value of the application-specific variable for the web-server. The webserver substitutes the placeholder by the current value of the application-specific variable.

1. Initialize the webserver in state GOAL_FSA_INIT_APPL or GOAL_FSA_INIT_GOAL:

```
goal_httpInit();
```

2. Open 1 instance of the webserver using the TCP port 8080 in state GOAL_FSA_INIT_SETUP:

```
GOAL_HTTP_T *pWebInstanceHdl;       /* handle for the web-server instance */
goal_httpNew(&pWebInstanceHdl, 8080, 1);
```

3. Register the callback-functions and allowed methods for the opened web-server in state GOAL_FSA_INIT_SETUP:

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

4. Provide a template for the webpage as string variable with placeholder for the application-specific variable:

```
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 of port.</p> \r\n\
</body></html>”
```

5. Implement a callback function to process web-requests:

```
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. Implement a callback function to get the current value of the application-specific variable from the application:

```
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. The callback function applWebReqCb() is called by the web-server after a web-request is received. The desired template for the web-page is made available for the web-server.

8. The callback function applWebGetValCb() is called for substitution.

10.6.4 Read a list

The webpage “device.html” shall be uploaded from the device to the web-browser. The web-page bases on the template webPage[]. The template includes a placeholder for the list of device components. The web-server calls the callback function applWebGetValCb() to provide the current value of the list entries. The webserver substitutes the placeholder by the current value of the application-specific variable.

1. Initialize the webserver in state GOAL_FSA_INIT_APPL or GOAL_FSA_INIT_GOAL:

```
goal_httpInit();
```

2. Open 1 instance of the webserver using the TCP port 8080 in state GOAL_FSA_INIT_SETUP:

```
GOAL_HTTP_T *pWebInstanceHdl;       /* handle for the webserver instance */
goal_httpNew(&pWebInstanceHdl, 8080, 1);
```

3. Register the callback-functions and allowed methods for the opened webserver in state GOAL_FSA_INIT_SETUP:

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

4. Create the list information and register the list information for the webserver. The list information contains a list-ID, a list name and the number of list entries.

```
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. Provide a template for the webpage as string variable with placeholders for the list and list entries:

```
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. Implement a callback function to process web-requests:

```
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. Implement a callback function to get the current value of the list entries from the application:

```
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, "deviceComponents",
        GOAL_STRLEN("deviceComponents"))) {

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

```
            }
        }
        return res;
    }
```

8. The callback function applWebReqCb() is called by the web-server after a web-request is received. The desired template for the web-page is made available for the web-server.

9. The callback function applWebGetValCb() is called for each substitution.

10.6.5 Set a user level

The upload of the webpage “admin.html” shall only be allowed for users with the USERLEVEL0. The login data for the USERLEVEL0 are:

- user name: admin
- password: a1b2c3:UL

The HTTPS transfer protocol is used.

The webpage “admin.html” does not contain any placeholder. No text substitutions on the webpage are necessary.

1. Initialize the webserver in state GOAL_FSA_INIT_APPL or GOAL_FSA_INIT_GOAL:

```
goal_httpInit();
```

2. Open 1 instance of the webserver using the TCP port 443 in state GOAL_FSA_INIT_SETUP:

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

3. Register the callback-functions and allowed methods for the opened webserver in state GOAL_FSA_INIT_SETUP:

```
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. Install the authentication for USERLEVEL0 in state GOAL_FSA_INIT_SETUP: The authentication data are written to the CM-variable USERLEVEL0.

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

5. Provide a template for the webpage as string variable:

```
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. Implement a callback function to process web-requests:

```
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. The callback function applWebReqCb() is called by the web-server after a web-request is received. This is only possible following successful login to the given web-server.

10.6.6 Download files

The webserver provides a download dialog. The received file is transferred to the application.

1. Initialize the webserver in state GOAL_FSA_INIT_APPL or GOAL_FSA_INIT_GOAL:

```
goal_httpInit();
```

2. Open 1 instance of the webserver using the TCP port 8080 in state GOAL_FSA_INIT_SETUP:

```
GOAL_HTTP_T *pWebInstanceHdl;       /* handle for the web-server instance */
goal_httpNew(&pWebInstanceHdl, 8080, 1);
```

3. Register the callback-functions and allowed methods for the opened webserver in state GOAL_FSA_INIT_SETUP:

```
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. Provide a web-page as string variable:

```
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. Implement an application-specific function applDownload() to receive and install the new firmware.

6. Implement an application-specific function applDownloadFinished() to report the result of the web-request to the application.

7. Implement a callback function to process web-requests:

```
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. The callback function applWebReqCb() is called by the web-server after a web-request is received.

Revision History	R-IN32M3 Module User's Manual: Software

Rev.	Date	Description	
		Page	Summary
1.00	June 30, 2020	—	First Edition issued

R-IN32M3 Module User’s Manual: Software

Publication Date: Rev.1.00 June 30, 2020

Published by: Renesas Electronics Corporation

R-IN32M3 Module

Renesas Electronics Corporation

R17US0001ED0100