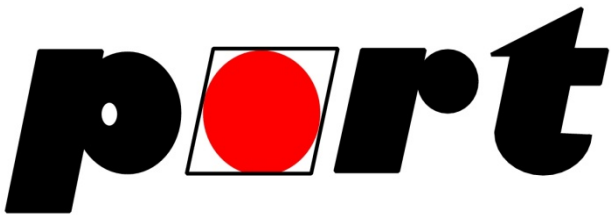
GOAL EtherNet/IP
Generic Open Abstraction Layer
Unified Ethernet/IP Layer
User Manual



Documentation Revision: 1.5 / GOAL EIP Version: 2.13.0

Contents

1 General Information

1.1 Changelog

Table 1: Changelog

Date	Revision	Authors	Comment
2017-04-04	1.0	maz	Initial revision.
2017-04-05	1.1	maz	Usage of pointer for EIP handles
2017-04-25	1.2	maz	Remove description of goal_eipHdlAppl
2018-05-24	1.3	meh	Update API descriptions
2018-09-28	1.4	mis	Update API descriptions, added configuration API
2018-12-04	1.5	mis	Added introduction to EtherNet/IP

1.2 Authors

The following authors have been working on the GOAL Ethernet/IP Layer.

Table 2: Authors of the GOAL Ethernet/IP Layer

Sign	Name	Mail
mis	Michael Struwe	mis@port.de
maz	Marcus Züche	maz@port.de

2 Introduction to EtherNet/IP

EtherNet/IP is an Industrial Ethernet protocol implementing the Common Industrial Protocol (CIP) over Ethernet in combination with the standard protocols IP, TCP and UDP. This protocol is maintained by the ODVA. The association also manages other implementations of the CIP protocol, e.g. DeviceNet (CIP over CAN).

Scanner

A Scanner is the master in an EtherNet/IP network. Usually it is the originator of a request.

Adapter

The Adapter is an EtherNet/IP slave. It creates responses to requests from a Scanner.

2.1 Encapsulation Protocol

On EtherNet/IP networks CIP messages are encapsulated by the Encapsulation Protocol. The Encapsulation Protocol defines an Encapsulation header, address items and a data items. It provides several commands for device identification, session management and CIP message transport. A session corresponds to a TCP connection between two endpoints.

2.2 CIP Object Model

The Common Industrial Protocol groups functionalities, attributes and behavior into classes. Each class has a class ID that is used to address a class. The ODVA manages the lists of classes that are officially defined. Vendors can also implement their own classes within a certain range of class IDs.

A class can have class attributes that describe the class and class services that influence the behavior of the class. An object is an instance of a CIP class. Each instance has its own set of instance attributes. All instances share the same instance services.

The CIP protocol addresses a certain service by specifying the class ID, instance ID and service ID. If a service is only applicable to an attribute, its attribute ID is also specified in the request. Class attributes are addressed via instance ID 0.

2.3 CIP Messages

2.3.1 Explicit Messages

Explicit Messages are sent acyclically. A client issues a request that explicitly addresses the resource of a server. The server sends a response. Explicit Messages only allow Point-to-Point connections. An Explicit Message can be either unconnected or connected. A connected message requires an established connection. An unconnected message can be sent without

establishing a connection beforehand. On EtherNet/IP networks Explicit Messages are used for acyclic access to CIP attributes and to invoke CIP services.

The Encapsulation Protocol for unconnected Explicit Messages consists of the Encapsulation header, the Null Address Item and the Unconnected Data Item. The Unconnected Data Item contains the actual CIP message. A connected Explicit Message has an Encapsulation Header a Connected Address Item and a Connected Data Item. The Connected Address Item contains the Connection ID associated with this Explicit Connection.

2.3.2 Implicit Messages

Implicit Messages are sent cyclically. A producer sends messages that can be consumed by one or multiple consumers, i.e. they are sent over Point-to-Point or Multicast connections. Implicit Messages contain a message ID that indicates the meaning of the data conveyed in these messages. Thus it can never be unconnected. On EtherNet/IP networks Implicit Messages are used to send cyclic IO data. These messages do not have an Encapsulation Header but have a Sequenced Address Item and a Connected Data Item. The Sequenced Address Item contains a sequence number and a Connection ID used to implicitly map the message data to a resource.

2.4 CIP Connections

For EtherNet/IP devices CIP connections are established via the CIP service “ForwardOpen” of the Connection Manager Object. This service is invoked by an unconnected Explicit Message.

2.4.1 Transport Class 3 Connections

The ForwardOpen service can open a Class 3 connection. This is a connected Explicit Message. Each message is prepended with a sequence count to detect duplicate messages. The Connection Path for Explicit Messages points to the Message Router Object.

2.4.2 Transport Class 0 & 1 Connections

The ForwardOpen service can also open Class 0 or Class 1 connections. Both classes are used for Implicit Messages. The ForwardOpen service usually opens two connections, one in the Originator-To-Target direction (for data consumed by the Adapter) and the other in the Target-To-Originator direction (for data consumed by the Adapter). Class 0 connections only contain the raw connection data. Class 1 connections prepend a sequence count to the connection data. This sequence count is independent from the one of the Sequenced Address Item in the Encapsulation Protocol. The Connection Path for Implicit Messages points to instances of the Assembly Object class. The Connection Path may contain instances for a producing Assembly, a consuming Assembly or a configuration Assembly. The data attributes

of the assembly instances define the data layout of the consumed and produced IO data and of the configuration data. The configuration data is sent with the ForwardOpen request to configure the application.

2.4.3 Connection Timing

The ForwardOpen service contains Requested Packet Intervals (RPI) for both directions. The Adapter's response contains the Actual Packet Intervals (API). The API is the RPI adjusted to the Adapter's timer resolution. These intervals are used for multiple timers that maintain the connections.

Transmission Trigger Timer

This Timer is loaded with the API value for the Target-To-Originator direction. Whenever the timer expires production of new connection data is started and the timer is reloaded.

Inactivity/Watchdog Timer

This timer is is used for consuming connections. The timer is loaded with the API of the Originator-To-Target direction multiplied by the Connection Timeout Multiplier. The timer is reset whenever valid connection data was consumed. If the timer expires the connection is considered timed out and will be closed. For the first reception the timer is set to at least 10 seconds.

Production Inhibit Timer

This timer is started for Implicit Messages whenever new data was produced because of a change of state or a trigger signal from the application. A new production request shall be delayed as long as this timer has not expired. The Production Inhibit Time loaded into the timer is received via the ForwardOpen service as a Network Segment.

2.4.4 Production Triggers

The production Trigger of a CIP connection determines when connection data is produced.

Cyclic

Data is sent cyclically whenever the Transmission Trigger Timer expires.

Change-of-State

Data is sent if the Application detects a change of state for its produced data. Also the Transmission Trigger Timer can invoke the production of new data to keep the connection alive.

Application Object Triggered

The application decides when to produce new connection data. Also the Transmission Trigger Timer can invoke the production of new data to keep the connection alive.

2.4.5 Application Connection Types for Implicit Messages

When the ForwardOpen service is used to create a class 0 or class 1 connection it opens two connections. One in the Originator-To-Target direction the other in the Target-To-Originator direction. The Application Connection Type defines the relationship between these two connections. The Adapter can recognize the Application Connection Type by the Connection Path of the ForwardOpen request. This means a combination of a producing Assembly and a consuming Assembly can always have only one Application Connection Type.

Listen Only

This connection produces data in the Target-To-Originator direction. The originator only sends a heartbeat. A Listen Only connection always depends on the existence of a non-Listen Only connection with the same Target-To-Originator connection path. Thus this Application Connection Type can be used to "listen" to data produced for other devices, e.g. a supervisor tool.

Input Only

This connection produces data in the Target-To-Originator direction. The originator only sends a heartbeat. It does not depend on any other connection. This Application Connection Type is suitable for connections that produce data that does not depend on other data consumed from other devices.

Exclusive Owner

This connection produces data in both directions, i.e. an Adapter produces data and consumes data. The originator that sends the data consumed by the Adapter is the connection owner. There can only be one active connection to the consuming connection path, i.e. it is exclusive. This Application Connection Type is suitable for connections that produce data that depends on other data consumed from other devices.

Redundant Owner

This Application Connection Type corresponds to Exclusive Owner connections with the exception. that multiple originators can try to claim ownership of this connection. All redundant owners receive the data produced by the Adapter. However the Adapter only consumes data from the current connection owner. If the connection to the current owner is closed or times out another redundant owner can take over the ownership. The current owner of a connection is determined by values in the header of the Originator-To-Target message.

2.4.6 Real time formats for Implicit Messages

As described in previous chapters an Implicit Message for EtherNet/IP consists of a Sequenced Address Item, a Connected Data Item an optional sequence count (for class 1 connections) and the actual connection data. The layout of the data is determined by the Assembly ID in the Connection Path. The Real time format defines how to interpret the data. An Adapter knows from the Connection Path what real time format both directions have. Some Application Connection Types require a certain real time format for a direction.

Modeless Format

The connection data does not indicate any idle information. Only the raw data is transmitted.

Zero Length Data Format

The connection data indicates idle mode if its length is zero. Otherwise it indicates run mode.

Heartbeat Format

The connection data always has the length 0. This format is required for the Originator-To-Target direction of Listen Only and Input Only connections.

32-Bit Header Format

This format prepends a 32 bit header to the connection data. For class 1 connections the header is inserted in between the sequence count and the connection data. The header has a status bit to indicate run mode or idle mode. Other fields of the header can be used to determine the ownership of a connection. Thus this format must be used Redundant Owner connections in the Originator-To-Target direction.

3 Introduction to GOAL EtherNet/IP Layer

The unified GOAL EtherNet/IP layer provides a common interface to all adapted EtherNet/IP stacks by using the same API in single core mode and also via GOALs Core To Core for multiple cores. This document describes the mandatory design rules for creating new commands for the GOAL CLI.

4 Initialization

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.

```
static GOAL_EIP_T *pHdlEip;   /**< GOAL Ethernet/ IP handle */
GOAL_STATUS_T appl_init(
    void
)
{
    GOAL_STATUS_T res;                          /* result */

    res = goal_eipInit();
    if (GOAL_RES_ERR(res)) {
       goal_logErr("Initialization of Ethernet/IP failed");
    }

    return res;
}


/*****************************************************************************/
/** Application Setup
 *
 * Setup the application.
 */
GOAL_STATUS_T appl_setup(
    void
)
{
    GOAL_STATUS_T res;                         /* result */

    /* for a real device the serial number should be unique per device */
    res = goal_eipCfgSerialNumSet(123456789);
    if (GOAL_RES_ERR(res)) {
        goal_logErr("failed to set Serial Number");
        return res;
    }

    goal_logInfo("create new instance");

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

```
        goal_logErr("failed to create a new EtherNet/IP instance");
        return res;
    }

    res = main_eipApplInit(pHdlEip);
    if (GOAL_RES_ERR(res)) {
        goal_logErr("failed to initialize assembly and attribute configuration");
        return res;
    }

    return GOAL_OK;
}
```

5 Application Programming Interface

This chapter 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. For further details please refer to the Reference Manual.

5.1 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();
```

5.2 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 3: 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);
```

5.3 goal_eipCipClassRegister

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

Table 4: 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);
```

5.4 goal_eipCreateAssemblyObject

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

Table 5: 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);
```

5.5 goal_eipAssemblyObjectGet

Get the pointer to the data array of an assembly.

Table 6: 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);
```

5.6 goal_eipAddExclusiveOwnerConnection

Create one Exclusive Owner connection with producing and consuming endpoints.

Table 7: 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);
```

5.7 goal_eipAddInputOnlyConnection

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

Table 8: 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);
```

5.8 goal_eipAddListenOnlyConnection

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

Table 9: goal_eipAddListenOnlyConnection parameters

Parameter	Description
GOAL_EIP_T *pHdlEip	GOAL EtherNet/IP handle
uint32_t connNum	the number of the input only connection.

Parameter	Description
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);
```

5.9 goal_eipGetVersion

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

Table 10: 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);
```

5.10 goal_eipDeviceStatusSet

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

Table 11: 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);
```

5.11 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 12: 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 Stack Configuration

This chapter lists all functions that are used to configure the EtherNet/IP stack. These functions must be called within appl_setup() **before** goal_eipNew() was called. Otherwise these functions have no effect. Each configuration has a default value that will be applied if no other value was set.

6.1 goal_eipCfgVendorIdSet

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

Default Value: 1114

Table 13: goal_eipCfgVendorIdSet parameters

Parameter	Description
uint16_t vendorId	Vendor ID

```
res = goal_eipCfgVendorIdSet(1114);
```

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 14: goal_eipCfgDeviceTypeSet parameters

Parameter	Description
uint16_t deviceType	Device Type

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

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 15: goal_eipCfgProductCodeSet parameters

Parameter	Description
uint16_t productCode	Product Code

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

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 behaviour.

Default Value: 1.1

Table 16: goal_eipCfgRevisionSet parameters

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

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

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 17: goal_eipCfgSerialNumSet parameters

Parameter	Description
uint32_t serial	SErial Number

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

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 18: goal_eipCfgProductNameSet parameters

Parameter	Description
const char *strName	Product Name

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

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 19: goal_eipCfgDomainNameSet parameters

Parameter	Description
const char *strName	default Domain Name

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

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 20: goal_eipCfgHostNameSet parameters

Parameter	Description
const char *strName	default Host Name

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

6.9 goal_eipCfgNumExplicitConSet

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

Default Value: 6

Table 21: goal_eipCfgNumExplicitConSet parameters

Parameter	Description
uint16_t num	number of explicit connections

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

6.10 goal_eipCfgNumImplicitConSetImpl

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

Default Value: 6

Table 22: goal_eipCfgNumImplicitConSetImpl parameters

Parameter	Description
uint16_t num	number of implicit connections

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

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 23: goal_eipCfgEthLinkCountersOn parameters

Parameter	Description
GOAL_BOOL_T enable	enable or disable feature

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

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 24: goal_eipCfgEthLinkControlOn parameters

Parameter	Description
GOAL_BOOL_T enable	enable or disable feature

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

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 25: goal_eipCfgChangeEthAfterResetOn parameters

Parameter	Description
GOAL_BOOL_T enable	enable or disable feature

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

6.14 goal_eipCfgChangeIpAfterResetOn

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

Default Value: GOAL_FALSE

Table 26: goal_eipCfgChangeIpAfterResetOn parameters

Parameter	Description
GOAL_BOOL_T enable	enable or disable feature

```
res = goal_eipCfgChangeIpAfterResetOn(GOAL_FALSE);
```

6.15 goal_eipCfgNumSessionsSet

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

Default Value: 20

Table 27: goal_eipCfgNumSessionsSet parameters

Parameter	Description
uint16_t num	number of sessions

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

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 28: goal_eipCfgTickSet parameters

Parameter	Description
uint32_t ticks	size of 1 tick in ms

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

6.17 goal_eipCfgO2TRunIdleHeaderOn

Default Value: GOAL_TRUE

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

Table 29: goal_eipCfgO2TRunIdleHeaderOn parameters

Parameter	Description
GOAL_BOOL_T enable	enable or disable feature

```
res = goal_eipCfgO2TRunIdleHeaderOn(GOAL_TRUE);
```

6.18 goal_eipCfgT2ORunIdleHeaderOn

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

Default Value: GOAL_FALSE

Table 30: goal_eipCfgT2ORunIdleHeaderOn parameters

Parameter	Description
GOAL_BOOL_T enable	enable or disable feature

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

6.19 goal_eipCfgQoSOn

Enable support of the QoS object.

Default Value: GOAL_TRUE

Table 31: goal_eipCfgQoSOn parameters

Parameter	Description
GOAL_BOOL_T enable	enable or disable feature

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

6.20 goal_eipCfgNumDelayedEncapMsgSet

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

Default Value: 2

Table 32: goal_eipCfgNumDelayedEncapMsgSet parameters

Parameter	Description
uint16_t num	number of messages

```
res = goal_eipCfgNumDelayedEncapMsgSet(2);
```

6.21 goal_eipCfgDhcpOn

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

Default Value: GOAL_FALSE

Table 33: goal_eipCfgDhcpOn parameters

Parameter	Description
GOAL_BOOL_T enable	enable or disable feature

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

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 34: goal_eipCfgDlrOn parameters

Parameter	Description
GOAL_BOOL_T enable	enable or disable feature

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

7 Application Callbacks

To use the full potential of *EtherNet/IP* the stack 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 35: 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.

```
/*****************************************************************************/
/** EtherNet/IP Callback Handler
 *
 * This function collects all callbacks from the stack and decides if the
 * callback must be handled.
 */
GOAL_STATUS_T main_eipCallback(
    GOAL_EIP_T *pHdlEip,                    /**< PROFINET handle */
    GOAL_EIP_CB_ID_T id,                    /**< callback id */
    GOAL_EIP_CB_DATA_T *pCb                 /**< callback parameters */
)
{
    GOAL_STATUS_T res = GOAL_OK;            /* result */

    switch (id) {
        case GOAL_EIP_CB_ID_INIT:
            /* initialize application resources */
            break;

        /* ... */
    }

    return res;
}
```

7.1 GOAL_EIP_CB_ID_INIT

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

Parameter

<none>

Return Value

- GOAL_OK - success
- other - fail

7.2 GOAL_EIP_CB_ID_READY

Initialization is done.

Parameter

<none>

Return Value

<ignored>

7.3 GOAL_EIP_CB_ID_CONNECT_EVENT

Inform the application about a connection event

Parameter

Table 36: 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 output assembly
2	connectionEvent	uint32_t	GOAL_EIP_CONNECTION_EVENT_*

Return Value

<ignored>

7.4 GOAL_EIP_CB_ID_ASSEMBLY_DATA_RECV

New data for an assembly has been received.

Parameter

Table 37: 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

7.5 GOAL_EIP_CB_ID_ASSEMBLY_DATA_SEND

Inform application that data of an assembly will be sent.

Parameter

Table 38: 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

7.6 GOAL_EIP_CB_ID_RUN_IDLE_CHANGED

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

Parameter

Table 39: 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 output assembly
2	runIdleValue	uint32_t	current value of the run/idle flag

Return Value

<ignored>

7.7 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 out of GOAL_EIP_LED_* macros. If a bit is set the corresponding LED must be set. Otherwise it must be cleared.

Parameter

Table 40: 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>

7.8 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 41: callback parameters of GOAL_EIP_CB_ID_LED_CHANGED

Element	Member	data type	Description
0	resetState	uint32_t	GOAL_EIP_RESET_*

Return Value

<ignored>