# DLR - Device Level Ring Protocol User Manual

port GmbH, 2016

Documentation Revision: 1.1 / DLR Version: 1.1.0

# Contents

# 1 General Information

## 1.1 Changelog

This section provides a short change description for documentation updates.

| Date | Revision | Author | Comment |
|---|---|---|---|
| 2015-08-27 | 1.0 | mis | Initial revision. |
| 2016-02-02 | 1.1 | mis | - updated DLR changelog |
| | | | - added API DLR_loop() |
| | | | - removed platform specific content |
| | | | - added list of supported hardware |

Table 1: Document Revisions

## 1.2 Authors

The following authors have been working on the DLR stack.

| Sign | Name | Mail |
|---|---|---|
| mis | Michael Struwe | mis@port.de |

Table 2: Authors of DLR manual

## 1.3 Supported Platforms

The DLR stack by *port* requires GOAL (Generic OS Abstraction Layer). Every platform that is supported by GOAL is also supported by the DLR stack. However the hardware requires a 3-port switch.

Announce-based Ring Nodes can run on any platform with a 3-port switch. However Beacon-based Ring Nodes require special hardware support and a special GOAL driver.

Currently the following platforms are suppported for Beacon-based Ring Nodes:

```
Renesas R-IN32M3 (with integrated 3-port switch)
```

If your hardware isn't listed here please ask us for an offer.

# 2 Overview

## 2.1 Introduction

The *Device Level Ring Protocol (DLR)* is a redundancy protocol for *EtherNet/IP* and operates on OSI Layer 2. It can detect bus faults in a single line topology. This bus fault can be compensated by activating a redundant communication path. Thus DLR allows to build fast recovering and redundant network topologies that do not influence the controlling applications.

The DLR stack requires *port's* GOAL (Generic OS Abstraction Layer).

A DLR participant always has two external Ethernet ports. Thus it requires an Embedded 3-port-switch. Furthermore Beacon-based Ring Nodes or DLR Supervisors need additional hardware support to process and generate Beacon frames at a high level of speed.

# 3 Device Level Ring Protocol

This chapter provides further information about the DLR protocol. It is part of the official *EtherNet/IP* specification of the ODVA.

## 3.1 DLR Classes

### 3.1.1 Ring Supervisor

The Ring Supervisor controls the DLR network. It must be able to send out and process Beacon Frames and Announce Frames within the specified time intervals. A Beacon Frame interval of at least 400 microseconds must be supported.

The Ring Supervisor also opens and closes the ring. In Normal State Forwarding between its external Ethernet ports is disabled, i.e. the ring is open. If a fault is detected somewhere in the network, the Ring Supervisor enables Forwarding between its ports and the ring is closed. As a result the line topology is restored and the network fault does not interfere with the logical communication channels.

### 3.1.2 Beacon-based Ring Node

A Beacon-based Ring Node must be able to process Beacon Frames sent by a Ring Supervisor. Since Beacon Frames can have an interval from 100 microseconds to 100 milliseconds hardware support for processing these frames is needed. If it detects a link loss, it must also inform the Ring Supervisor.

### 3.1.3 Announce–based Ring Nodes

An Announce-based Ring Node does not have the hardware support for Beacon Frames. Therefore these frames are not processed and only forwarded. The Ring Node must process the Announce Frames, though.

### 3.1.4 Redundant Gateway

A Redundant Gateway has DLR ports and uplink ports. It must separate the DLR traffic from other ports. It might act as a DLR Ring Supervisor or a Ring Node. Besides the normal DLR functionality the gateway can also act as an Active Gateway. In this mode it forwards frames from the DLR ports to the uplink ports. An Active Gateway propagates its presence to other DLR nodes. If a message from another Active Gateway with higher precedence was received, the gateway loses its active state and must stop forwarding frames from the DLR ports.

## 3.2 Principle of Operation

A DLR network consists of a Ring Supervisor and Ring nodes. Each DLR device must have at least two Ethernet ports as part of an embedded switch. During normal operation the Ring Supervisor deactivates Forwarding for one of its ports. Thus the ring is open and the network has a normal single line topology.

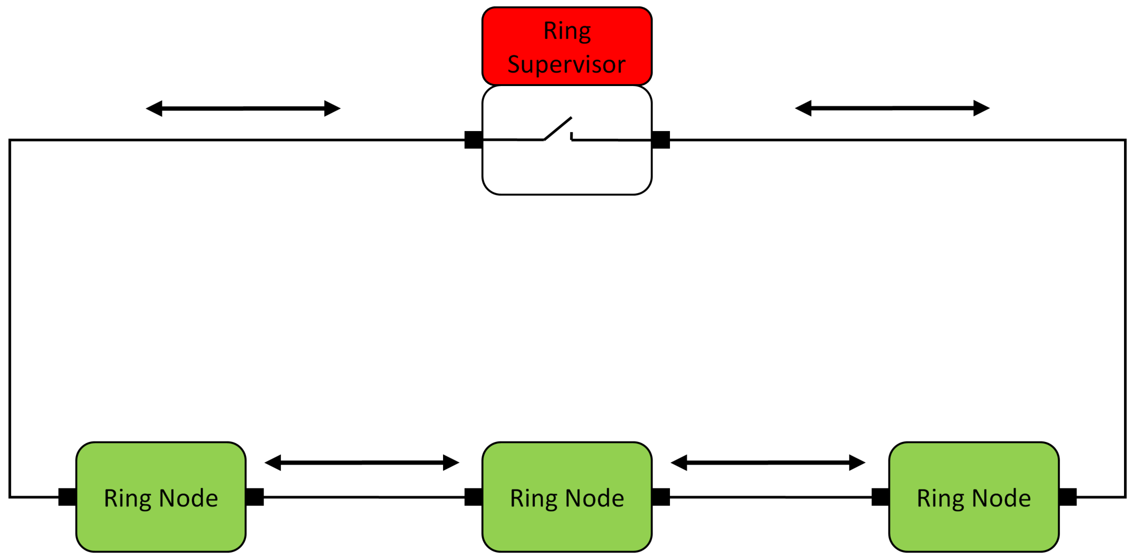

Figure 1: DLR network in Normal State

The Ring Supervisor cyclically sends out Beacon Frames and Announce Frames on both ports. They are received on one port of a Ring Node, processed and passed on to the next ring node via the other port. By default the Beacon Frames are sent every 400 microseconds and the Announce Frames are sent every second.

If a Ring Node detects a link loss on one of its ports, it will inform the Ring Supervisor. The Ring Supervisor then informs all Ring Nodes that the network is now in Fault State. Also if a Ring Supervisor does not receive its Beacon Frame on its other port, it will assume a link failure and asks the Ring nodes to detect the location of the failure.

During the Fault State all devices will flush their MAC tables and the Ring Supervisor activates Forwarding for both ports. Thus a new line topology is created and the logical communication path is retained.

If the physical path is fixed again, the ring is closed. Once the Supervisor has received on both of its ports a Beacon Frame from its other port it disables Forwarding between its external port. Now the ring is opened again and the normal Linear topology is restored. The Supervisor will inform the other participants that the network is in Normal State again.

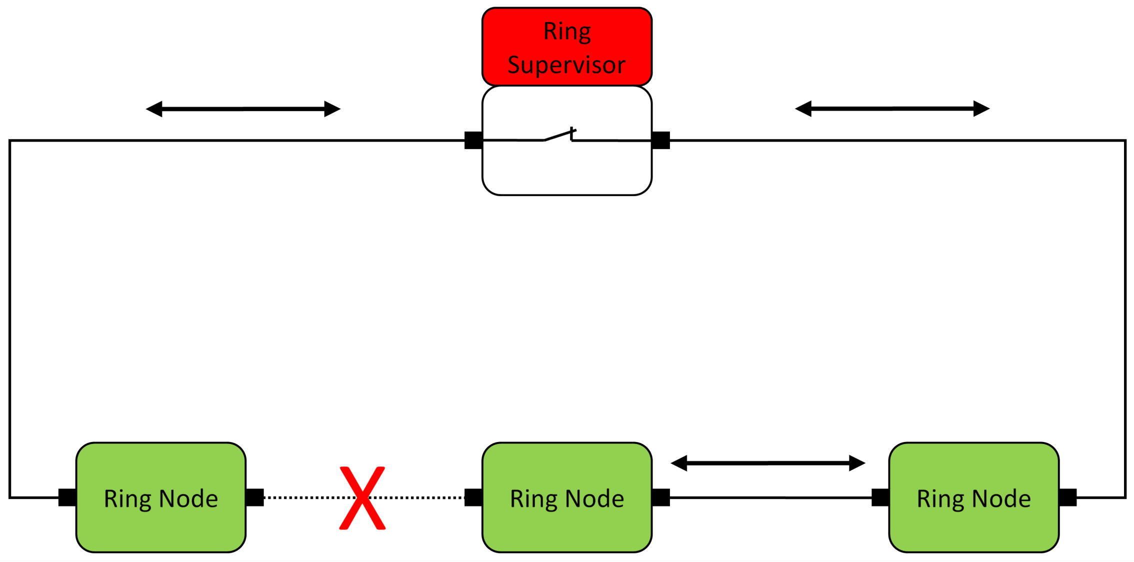

Figure 2: DLR network in Fault State

It is also possible to have multiple Ring Supervisors in one DLR ring. But only one Supervisor can be active. The others act as normal Ring Nodes. The active Supervisor is always the Supervisor with the highest precedence value. This value can be set by via a DLR Object attribute of *EtherNet/IP*. If two Supervisors within a network have the same precedence value, their MAC addresses are compared. The Supervisor with the numerically higher MAC address becomes the active Supervisor.

# 4 Configuration of the DLR stack

The DLR stack needs the application specific configuration file *dlr_config.h*. It provides the following options:

- *DLR_CONFIG_BEACON_RINGNODE*
  Enable support for a Beacon-based Ring Node.
- *DLR_CONFIG_BEACON_HW_SUPPORT*
  This option must be set for targets that completly process the beacon frame in hardware.
- *DLR_CONFIG_FLUSHTABLES*
  Enable support for LearningUpdate Frames sent by Redundant Gateways.

# 5 Application Programming Interface

The DLR protocol is part of the EtherNet/IP specification. There is a DLR CIP object. That needs to communicate with the DLR stack. Also the stack must be integrated into the target platform. The stack was designed to be used in conjunction with *port's* GOAL.

This chapter provides an overview of all necessary functions. For further details please refer to the Reference Manual.

## 5.1 API for GOAL

These functions should be called from within GOAL to execute the DLR stack.
To connection to GOAL can be enabled in the GOAL configuration file *goal_config.h*.

| Function | Description |
|---|---|
| GOAL_STATUS_T DLR_SM_init | Initialize the DLR State Machine.<br>**Parameters:**<br><none> |
| void DLR_SM_exec | Execute the DLR State Machine whenever an event occurs.<br>**Parameters:**<br>DLR_EVENT_T event<br>uint8_t port |
| GOAL_STATUS_T DLR_receive | Process received DLR frames and generate events for the DLR state machine.<br>**Parameters:**<br>GOAL_BUFFER_T *pBuf |
| void DLR_loop | process received DLR frames and creates the appropriate event<br>**Parameters:**<br><none> |

Table 3: API used by GOAL

## 5.2 API for the EtherNet/IP stack

These functions provide all necessary information to connect the DLR object to the DLR stack.

| Function | Description |
|---|---|
| uint32_t DLR_getCapability | Get the Capability Flags of the Ring Node: Announce-based Ring Node or Beacon-based Ring Node, Supervisor Capable, Redundant Gateway Capable, Flush_Table frame Capable<br>**Parameters:**<br><none> |
| uint8_t DLR_getNetStatus | Get the current network status from the point of view of the device: Normal, Ring Fault, Unexpected Loop, Partial Network Fault, Rapid Fault/Restore Cycle<br>**Parameters:**<br><none> |
| uint8_t DLR_getNetTopology | Get the DLR network topology: Linear or Ring<br>**Parameters:**<br><none> |
| void DLR_getSupervisorAddr | Get the MAC address and IP address of the active Ring Supervisor.<br>**Parameters:**<br>uint32_t *pIPAddr<br>uint8_t *pMAC |

Table 4: API used by the EtherNet/IP stack

# 6 Logging

Since the DLR stack is based on GOAL it uses GOAL's logging mechanism. In order to enable logging you have to define the logging ID *GOAL_LOG_ID_DLR* in *goal_conf.h*.

```
#define GOAL_LOG_ID_DLR (1<<0)
```

Also you have to enable logging for this module by calling the function *goal_logEnableModule*.

```
goal_logEnableModule(GOAL_LOG_ID_DLR);
```

Furthermore the desired debug level must be enabled with the function *goal_logSetLevel*.

```
goal_logSetLevel(GOAL_DBG_DEFAULT); /* Errors, Warnings, Infos */
```

If logging was successfully enabled for the DLR stack, the messages are written along with the other GOAL messages to the configured output.

**Logging should only be enabled during development. For productive code logging must be disabled.**

If logging is enabled, the logging messages are also compiled into the binary and use up additional space. Also the process of writing these messages to the configured output is blocking the CPU. Thus realtime communication might not be possible while logging is active. However there is the option to use a non-blocking variant of GOAL's logging mechanisms. The disadvantage is that logging messages might get lost, if the ringbuffer cannot be emptied fast enough. Therefore the logging buffer size should always be bigger in non-blocking mode.

# 7 Code Optimization

Various platforms with various compilers have shown, that there is no predictable way how compiler optimized code works. Most of the times, targets suffered from different problems, like:

- Network packets where not transferred from interrupt to application.
- Data loss / unwanted data manipulation.
- Imprecise code steps / data views in debugger.

For these reasons, please turn of optimization when asking for support.

# 8 Terms

| | |
|---|---|
| API | - Application Programming Interface |
| BSP | - Board Support Package |
| CIP | - Common Industrial Protocol |
| DLR | - Device Level Ring Protocol |
| EIP | - EtherNet/IP |
| GOAL | - Generic OS Abstraction Layer |
| ODVA | - Open DeviceNet Vendor Association |
| OS | - Operating System |
| OSI | - Open Systems Interconnection Model |
| SDK | - Software Development Kit |

# 9 DLR Changelog

| Version | Date | Changes |
|---|---|---|
| 1.0.0 | 2015-08-27 | - initial release of DLR stack |
| | | - added support for Renesas R-IN32M3 |
| 1.1.0 | 2016-02-02 | - separated reception & processing of frames |
| | | - new API DLR_loop (called cyclically) |
| | | - fixed flushing of Unicast MAC Table during state transition |
| | | - fixed VLAN Priority Code Point in generated DLR frames |
| | | - forced enum DLR_NODESTATE_T to have fixed values |
| | | - DLR stack is now compatible to GOAL 1.2 |

Table 5: DLR Changelog