acontis technologies GmbH

SOFTWARE

EC-Master

Feature Pack Cable-Redundancy

Version 3.2

Edition: September 23, 2025

Contents

1 Introduction

1.1 Licensing

Redundancy support is not included within the standard EC-Master license, thus the Redundancy support has to be licensed separately. To activate Redundancy support, the EC-Master Stack has to be compiled with the option INCLUDE_RED_DEVICE set, which is the default, if not explicitly disabled with EXCLUDE_RED_DEVICE.

1.2 Motivation

Cables expose cause of defects. Without cable redundancy one broken line will lead to unreachable devices. Applying cable redundancy eliminates the line as single point of failure within the EtherCAT® solution. This feature pack offers cable redundancy with simple and intuitive integration.

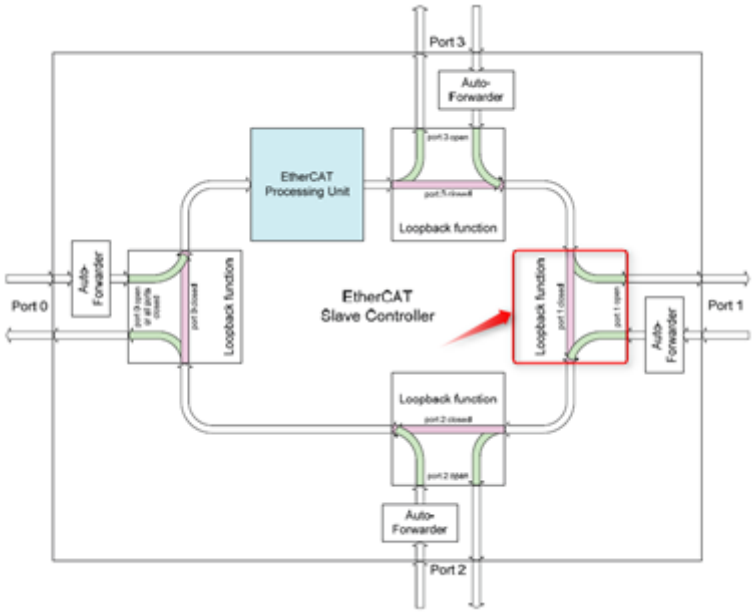
1.3 Overview

The acontis EC-Master EtherCAT® MainDevice allows the support of two Network Interfaces to provide a Redundancy feature.

A typical redundant configuration can be:

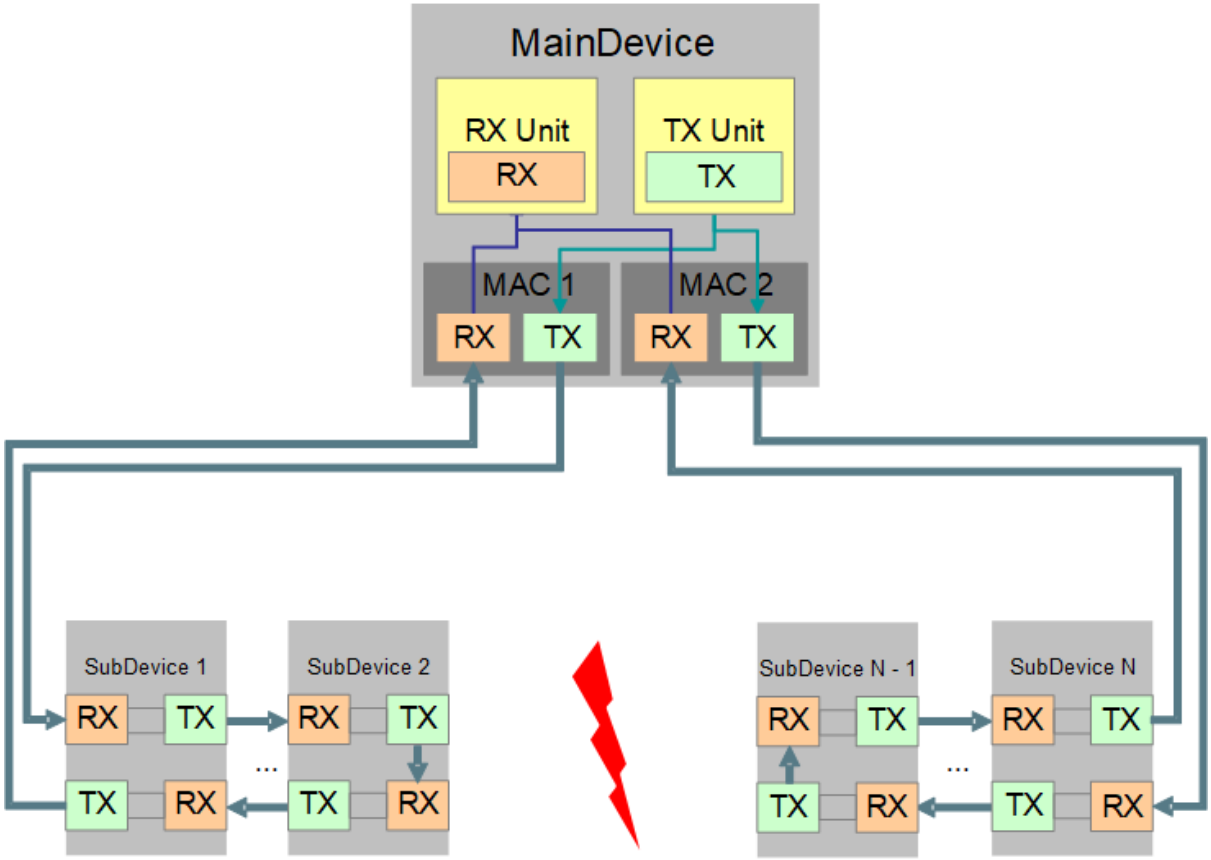
In this configuration the bus is connected from both sides (front and back) to two different network interfaces (main and redundant) on the Control Unit, carrying the EC-Master EtherCAT® MainDevice Stack. The frames are sent from and received on both interfaces. The main interface leads to the front and the redundancy interface to the back of the Control Unit.

The device is aware at which ports other devices are connected and opens and closes the ports accordingly by means of the port's loopback function. The following figure shows the four ports of the ET1100, each with loopback control:



In case of line break the loopback function closes the connection to the next EtherCAT®-device. If the redundancy line is not equipped all devices behind the closed loop would be not accessible anymore!

In order to keep all EtherCAT®-devices reachable in case of line break each used EtherCAT®-device has to provide at least two EtherCAT® ports. If somewhere in the middle a line breaks up, the loop is closed within the “bordering” SubDevices and both bus stubs are still communicating with the EC-Master stack. The following figure illustrates this:



All EtherCAT® devices still receive frames either from the main interface or from the redundant interface.

From an application's point of view, the bus operates as usual as long all SubDevices are still powered and operating. By means of the API calls still operate the same like before the bus split up. Especially Auto Increment addressed and logical commands are still unchanged after the bus split. Process data is sent to and received from all devices accordingly and the integrity check by means of working counters (WKC) is still ensured. Even Hot Connect Groups that are configured to be immediately connected to each other and get separated by a line break are still functional. Therefore the application developer doesn't need to change any calls or addresses during runtime.

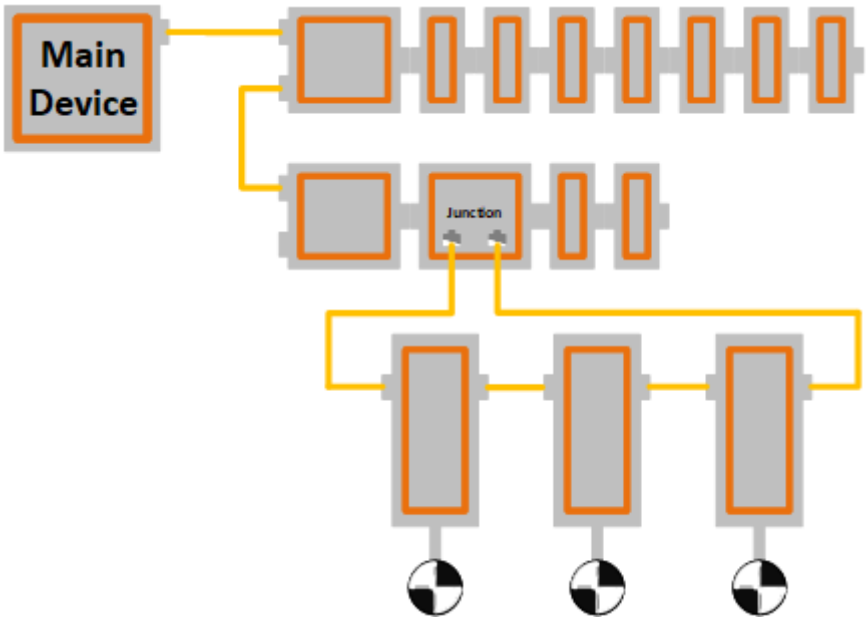
The system automatically recovers from the situation as soon as the EtherCAT®-devices report the link to be established again. The EC-Master Stack scans the bus to verify that all lines are available while continuing the operation.

In case of a line break the EC-Master Stack detects the redundancy requirement and throws a notification (*emNotify - EC_NOTIFY_RED_LINEBRK*) to the application, which enables a front-end or application user to be informed about the redundancy situation. The situation is handled immediately without any further needed interaction by the application. When the line break is fixed another notification (*emNotify - EC_NOTIFY_RED_LINEFIXED*) is thrown to inform about the absence of the error situation.

The system can handle at most one line break at a time having all EtherCAT® devices reachable. If there are two or more lines broken the reachability of the EtherCAT® devices cannot be ensured anymore. The system automatically recovers from multiple line breaks as soon as the lines are fixed.

1.3.1 Junction Redundancy

If the EtherCAT® MainDevice stack is running on systems with just a single Ethernet controller, it is possible to implement cable redundancy using junction devices, this is called "Junction Redundancy".



Behind the junction it is possible to set up a local ring which supports redundant operation. The whole system will stay operational in case of a break in this local ring. Multiple such local rings are supported, each of them will become an independent redundant system.

By default, junction redundancy is disabled. By calling *emIoControl - EC_IOCTL_SB_SET_JUNCTION_REDUNDANCY_MODE* it must be enabled by the user's application.

In case of a line break the EC-Master Stack detects the redundancy requirement and throws a notification (*emNotify - EC_NOTIFY_JUNCTION_RED_CHANGE*) to the application, which enables a front-end or application user to be informed about the redundancy situation. The situation is handled immediately without any further needed interaction by the application. Whenever the line break is fixed the same notification is thrown again to inform about the absence of the error situation.

Important: Just one single line break can be handled no matter how many local rings you have. Single fault tolerant. This can be explained by the circulating frame bit of the datagram header described in the EtherCAT® SubDevice Controller (ESC) Hardware Data Sheet ET1100 Section I – Technology Chapter 3.5. A frame will not be processed and destroyed when it traverses a second ESC with a auto closed port 0.

3.5 Circulating Frames

The ESCs incorporate a mechanism for prevention of circulating frames. This mechanism is very important for proper watchdog functionality.

This is an example network with a link failure between slave 1 and slave 2:

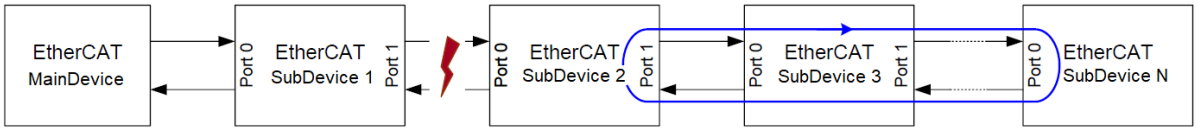


Figure 6: Circulating Frames

Both slave 1 and slave 2 detect the link failure and close their ports (port 1 at slave 1 and port 0 at slave 2). A frame currently traveling through the ring at the right side of slave 2 might start circulating. If such a frame contains output data, it might trigger the built-in watchdog of the ESCs, so the watchdog never expires, although the EtherCAT master cannot update the outputs anymore.

To prevent this, a slave with loop closed at port 0 and loop control for port 0 set to Auto or Auto close (ESC DL Control register 0x0100) will do the following inside the EtherCAT Processing Unit:

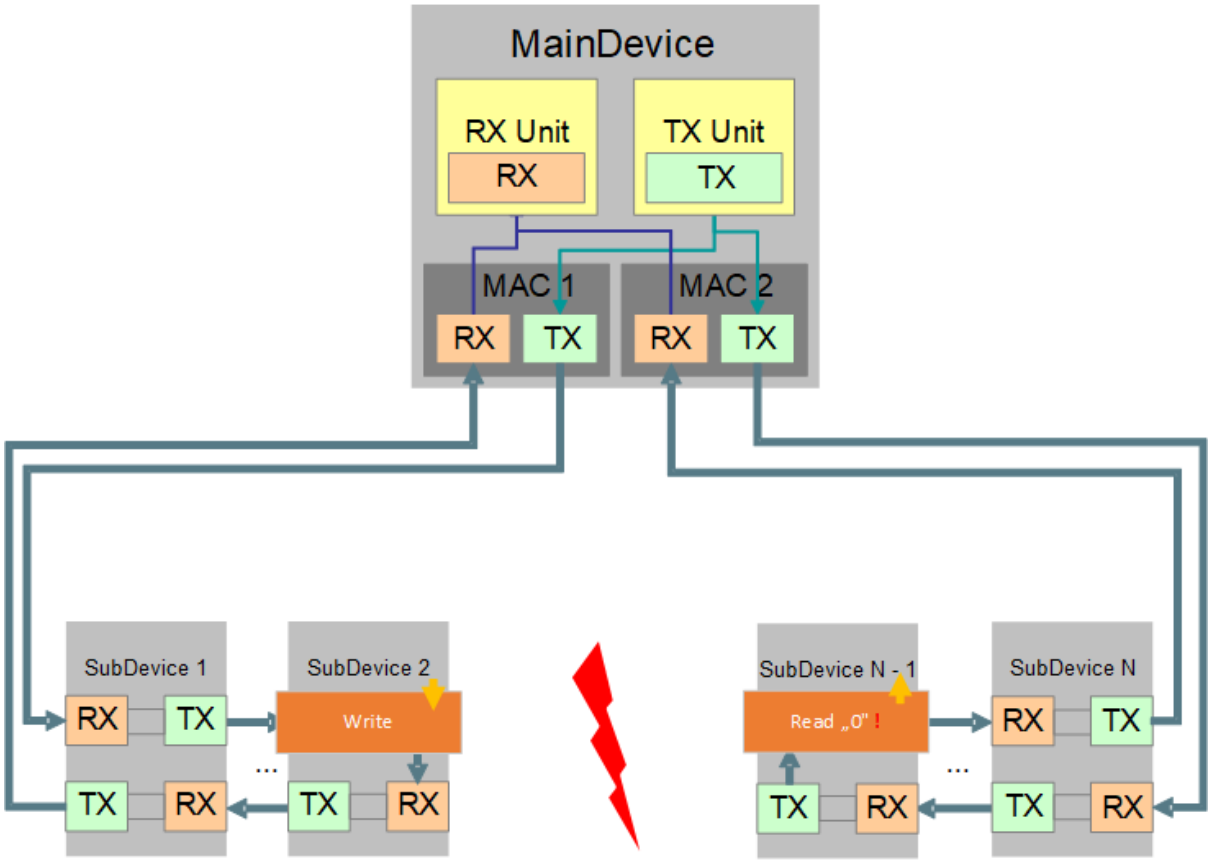
- If the Circulating bit of the EtherCAT datagram is 0, set the Circulating bit to 1
- If the Circulating bit is 1, do not process the frame and destroy it

The result is that circulating frames are detected and destroyed. Since the ESCs do not store the frames for processing, a fragment of the frame will still circulate triggering the Link/Activity LEDs. Nevertheless, the fragment is not processed.

1.4 Application notes

1.4.1 Use LRD/LWR instead of LRW

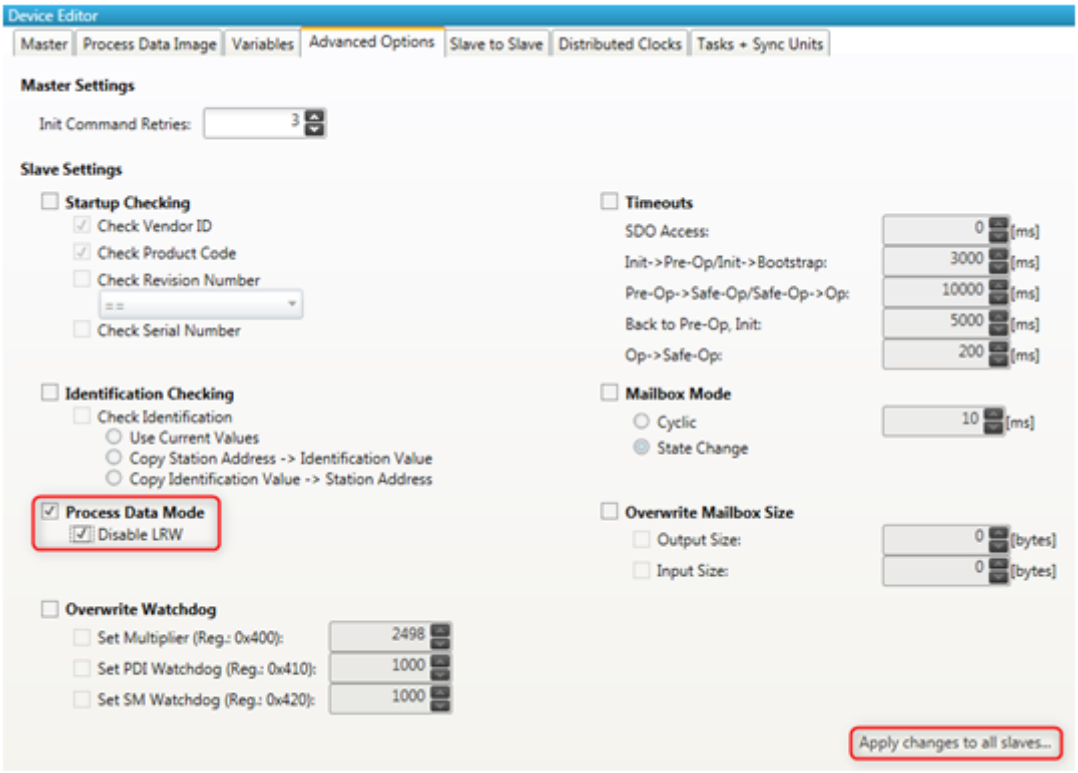
Important: It is not allowed to use LRW EtherCAT® commands because they force that multiple EtherCAT® devices operate on the same frame. In the following figure, “SubDevice 1” gets the frame from MAC 1 and “SubDevice N+1” gets the frame from MAC 2. These are different frames. MainDevice and/or SubDevices may receive data other than what is expected.



LDR/LWR instead of LRW must be configured if multiple SubDevices are addressed by the same EtherCAT® Cmd. If in doubt, configure LDR/LWR. The following screenshots show how to configure the SubDevices.

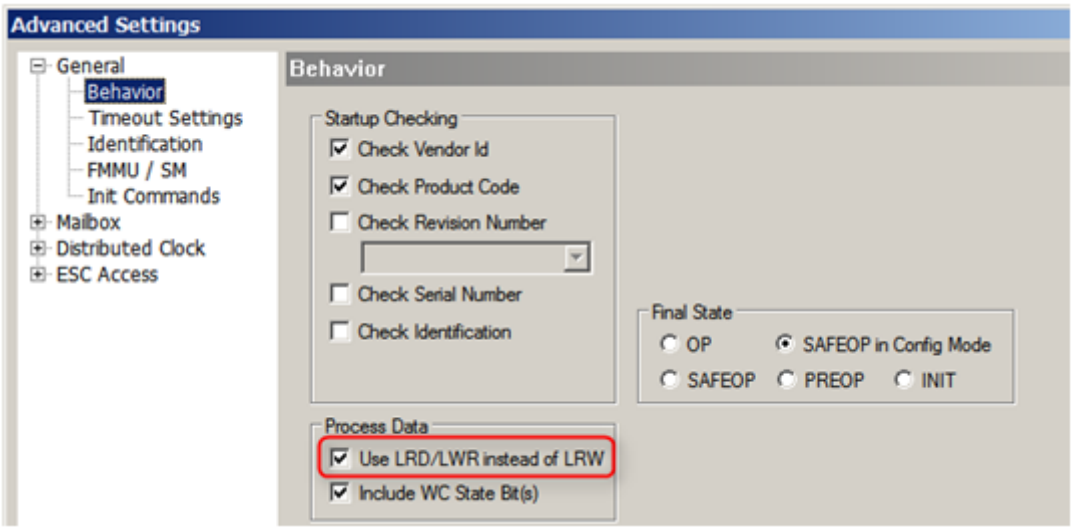
1.4.1.1 Configuring with EC-Engineer

In the MainDevice Advanced Options tab (Expert Mode View) the user can configure LRD/LWR usage instead of LRW for all SubDevices:



1.4.1.2 Configuring with ET9000

Chosing LDR/LWR instead of LRW within the ET9000 is located at the Advanced settings of the EtherCAT® SubDevice in the section General > Behavior.



1.4.2 Distributed clocks

As an outstanding solution the acontis EC-Master EtherCAT® MainDevice Stack is fully capable of distributed clocks operation for bus shift as well as Master shift.

2 Programmer's Guide

2.1 Configuration

To activate redundancy support, the EC-Master Stack has to be configured additionally with a second Network Interface. This is done by the parameter *EC_T_INIT_MASTER_PARMS::pLinkParmsRed* while initializing the stack by aims of the command *emInitMaster()*.

EC_T_DWORD **emInitMaster** (
    EC_T_DWORD dwInstanceID,
    EC_T_INIT_MASTER_PARMS *pParms
)

struct **EC_T_INIT_MASTER_PARMS**
EC-Master init parameters.

Public Members

EC_T_LINK_PARMS ***pLinkParmsRed**
[in] Link layer parameters for red device (cable redundancy)

2.1.1 emIoControl - EC_IOCTL_SB_SET_RED_ENHANCED_LINE_CROSSED_DETECTION_E

Enable or disable the enhanced line crossed detection if redundancy is configured.

emIoControl - EC_IOCTL_SB_SET_RED_ENHANCED_LINE_CROSSED_DETECTION_ENABLED

Parameter

- `pbyInBuf`: [in] Pointer to EC_T_BOOL. If value is EC_TRUE enhanced line crossed detection is enabled, if EC_FALSE it is not.
- `dwInBufSize`: [in] Should be set to sizeof(EC_T_BOOL).
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

Return
EC_E_NOERROR or error code

The line crossed detection is restricted by using redundancy. Only the simple detection works per default. No precise line crossed location will be notified.

Enabling the enhanced line crossed detection with this IOCTL permit the EC-Master to give more precise information about detected line crossed. During bus scans triggered by topology change, the EC-Master stack will close the redundancy link. This generate a controlled line break. Under this condition the EC-Master stack is able to detect all the line crossed. To achieve this, the Real-time Ethernet Driver has to support `EC_LINKIOCTL_SET_FORCEDISCONNECTION`. At this stage only the *emllIntelGbe* Real-time Ethernet Driver supports this IOCTL.

2.1.2 emIoControl - EC_IOCTL_SB_SET_JUNCTION_REDUNDANCY_MODE

Set junction redundancy mode

emIoControl - EC_IOCTL_SB_SET_JUNCTION_REDUNDANCY_MODE

Parameter

- `pbyInBuf`: [in] Pointer to value of EC_T_JUNCTION_REDUNDANCY_MODE
- `dwInBufSize`: [in] Should be set to sizeof(EC_T_JUNCTION_REDUNDANCY_MODE)
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

Return
EC_E_NOERROR or error code

enum **EC_T_JUNCTION_REDUNDANCY_MODE**
Values:

enumerator **eJunctionRedundancyMode_Disabled**

enumerator **eJunctionRedundancyMode_Automatic**

enumerator **eJunctionRedundancyMode_Strict**

Enabling the junction redundancy support will generate *emNotify - EC_NOTIFY_JUNCTION_RED_CHANGE* in case of junction redundancy status change is detected. In case of junction redundancy line break, the SubDevices connected at port B are connected backward. This would generate an `EC_NOTIFY_LINE_CROSSED` if this IOCTL is not called before.

Enabling the junction redundancy support will generate *emNotify - EC_NOTIFY_JUNCTION_RED_CHANGE* in case of junction redundancy line break is detected.

In the Disabled Mode, Line break will lead to an `EC_NOTIFY_LINE_CROSSED` notification, because the SubDevices connected at port B are connected backward. In the Automatic Mode, the junction connection are detected if present during startup. Line break can notified afterwards.

In the Strict Mode, the junction redundancy ports have to be explicitly defined using the following sequence instead of `emConfigureNetwork()`:

1. *`emConfigLoad()`*
2. *`emConfigAddJunctionRedundancyConnection()`*
3. *`emConfigApply()`*

In this mode, the line break can be notified during the startup.

2.1.3 emConfigLoad

static EC_T_DWORD **ecatConfigLoad**(
EC_T_CNF_TYPE eCnfType,
EC_T_PBYTE pbyCnfData,
EC_T_DWORD dwCnfDataLen
)

EC_T_DWORD **emConfigLoad** (
EC_T_DWORD dwInstanceID,
EC_T_CNF_TYPE eCnfType,
EC_T_PBYTE pbyCnfData,
EC_T_DWORD dwCnfDataLen
)

Load the network configuration.

In combination with emConfigApply, this function replaces emConfigureNetwork and must be called after the initialization. Among others the EtherCAT topology defined in the given XML configuration file will be stored internally.

Note: A client must not be registered prior to calling this function. Existing client registrations will be dropped.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **eCnfType** – [in] Type of configuration data provided
- **pbyCnfData** – [in] Configuration data
- **dwCnfDataLen** – [in] Length of configuration data in byte

Returns

- EC_E_NOERROR if successful
- EC_E_INVALIDSTATE if master isn't initialized
- EC_E_INVALIDPARM if dwInstanceID is out of range, the input pointer is EC_NULL or contains EC_NULL pointer
- EC_E_NOMEMORY if some memory cannot be allocated
- EC_E_ADS_IS_RUNNING if ADS server is running

enum **EC_T_CNF_TYPE**

Values:

enumerator **eCnfType_Unknown**

enumerator **eCnfType_Filename**
pbyCnfData: ENI filename to read

enumerator **eCnfType_Data**
pbyCnfData: ENI data

enumerator **eCnfType_Datadiag**
pbyCnfData: ENI data for diagnosis

enumerator **eCnfType_GenPreopENI**
Generate ENI based on bus-scan result to get into PREOP state

enumerator **eCnfType_GenPreopENIWithCRC**
Same as eCnfType_GenPreopENI with CRC protection

enumerator **eCnfType_GenOpENI**
Generate ENI based on bus-scan result to get into OP state. The default PDO mapping read from the slaves is activated. See ETG2010 “SII Specification”, Table 14 “Structure Category TXPDO and RXPDO for each PDO”

enumerator **eCnfType_None**
Reset configuration

enumerator **eCnfType_ConfigData**
pbyCnfData: Binary structured configuration

enumerator **eCnfType_GenOpENINoStrings**
Generate ENI based on bus-scan result to get into OP state , does not read strings from EEPROM

enumerator **eCnfType_FileByApp**
File access provided by user application, See EC_T_CNF_FILEBYAPP_DESC

enumerator **eCnfType_GenEBI**
Generate EBI based on bus-scan result

2.1.4 emConfigAddJunctionRedundancyConnection

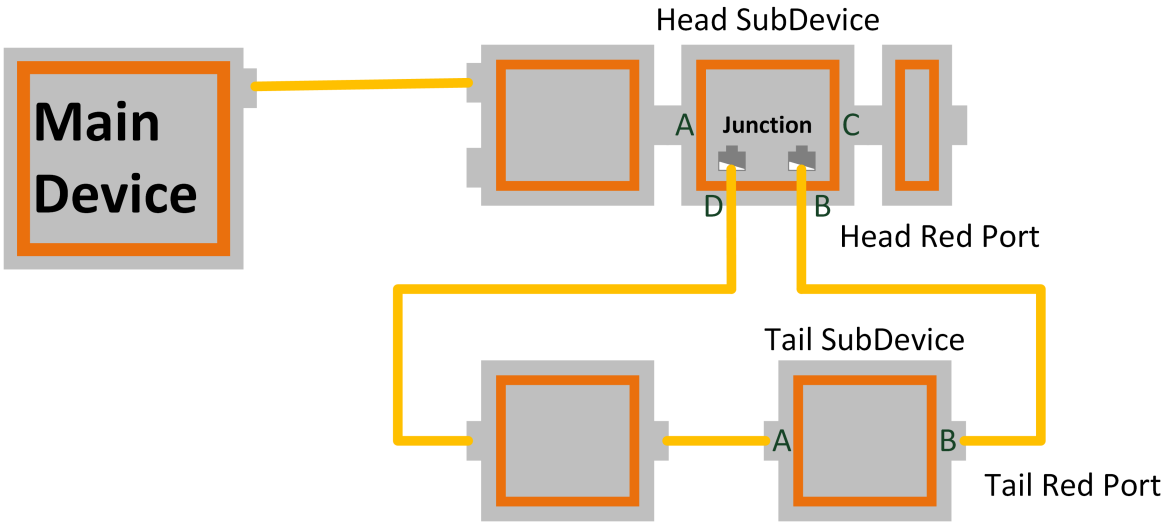
static EC_T_DWORD **ecatConfigAddJunctionRedundancyConnection** (
EC_T_WORD wHeadStationAddress,
EC_T_WORD wHeadRedPort,
EC_T_WORD wTailStationAddress,
EC_T_WORD wTailRedPort
)
EC_T_DWORD **emConfigAddJunctionRedundancyConnection** (
EC_T_DWORD dwInstanceID,
EC_T_WORD wHeadStationAddress,
EC_T_WORD wHeadRedPort,
EC_T_WORD wTailStationAddress,
EC_T_WORD wTailRedPort
)
Add a junction redundancy connection.

Since there is no mechanism to configure junction redundancy in the ENI, this API allows adding junction redundancy connections which will be validated by the EC-Master stack. It has to be called after emConfigLoad and prior to calling emConfigApply. Calling this API enables junction redundancy support implicitly.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **wHeadStationAddress** – [in] Station address of the junction redundancy head slave. Typically this is an EtherCAT junction.
- **wHeadRedPort** – [in] Port at head slave to which the junction redundancy cable is connected. Must be ESC_PORT_B
- **wTailStationAddress** – [in] Station address of the junction redundancy tail slave
- **wTailRedPort** – [in] Port at tail slave to which the junction redundancy cable is connected. May not be ESC_PORT_A.

Returns
EC_E_NOERROR or error code



2.1.5 emConfigApply

static EC_T_DWORD **ecatConfigApply**(EC_T_VOID)

EC_T_DWORD **emConfigApply**(EC_T_DWORD dwInstanceID)

Apply the network configuration.

It has to be called after emConfigLoad.

Parameters

dwInstanceID – [in] Instance ID (Multiple EtherCAT Network Support)

Returns

EC_E_NOERROR or error code

2.2 Polling status information from the EC-Master Stack

2.2.1 emGetNumConnectedSlavesMain

static EC_T_DWORD **ecatGetNumConnectedSlavesMain**(EC_T_VOID)

EC_T_DWORD **emGetNumConnectedSlavesMain**(EC_T_DWORD dwInstanceID)

Get the amount of currently connected Slaves to main interface.

Parameters

dwInstanceID – [in] Instance ID (Multiple EtherCAT Network Support)

Returns

Number of connected slaves at main interface

2.2.2 emGetNumConnectedSlavesRed

static EC_T_DWORD **ecatGetNumConnectedSlavesRed**(EC_T_VOID)

EC_T_DWORD **emGetNumConnectedSlavesRed**(EC_T_DWORD dwInstanceID)

Get the amount of currently connected Slaves to redundancy interface.

Parameters

dwInstanceID – [in] Instance ID (Multiple EtherCAT Network Support)

Returns

Number of connected slaves at redundancy interface

2.2.3 emIoControl - EC_IOCTL_IS_MAIN_LINK_CONNECTED

Determine whether main link is connected.

emIoControl - EC_IOCTL_IS_MAIN_LINK_CONNECTED

Parameter

- `pbyInBuf`: [in] Should be set to EC_NULL
- `dwInBufSize`: [in] Should be set to 0
- `pbyOutBuf`: [out] Pointer to EC_T_DWORD. If value is EC_TRUE link is connected, if EC_FALSE it is not.
- `dwOutBufSize`: [in] Should be set to sizeof(EC_T_DWORD)
- `pdwNumOutData`: [out] Pointer to EC_T_DWORD. Amount of bytes written to the output buffer.

Return

EC_E_NOERROR or error code

2.2.4 emIoControl - EC_IOCTL_IS_RED_LINK_CONNECTED

Determine whether redundancy link is connected.

emIoControl - EC_IOCTL_IS_RED_LINK_CONNECTED

Parameter

- `pbyInBuf`: [in] Should be set to EC_NULL
- `dwInBufSize`: [in] Should be set to 0
- `pbyOutBuf`: [out] Pointer to EC_T_DWORD. If value is EC_TRUE link is connected, if EC_FALSE it is not.
- `dwOutBufSize`: [in] Should be set to sizeof(EC_T_DWORD)
- `pdwNumOutData`: [out] Pointer to EC_T_DWORD. Amount of bytes written to the output buffer.

Return

EC_E_NOERROR or error code

2.3 Receiving notifications

Following notifications are defined for redundancy support.

2.3.1 emNotify - EC_NOTIFY_RED_LINEBRK

Every time the EC-Master detects a line break this error code will be signaled once to the application.

How many SubDevices are on the main interface and how many are on the redundancy interface is stored in structure *EC_T_RED_CHANGE_DESC* of the union element RedChangeDes:

struct **EC_T_RED_CHANGE_DESC**

Public Members

EC_T_WORD **wNumOfSlavesMain**
Number of Slaves on Main Line

EC_T_WORD **wNumOfSlavesRed**
Number of Slaves on Red Line

2.3.2 emNotify - EC_NOTIFY_RED_LINEFIXED

Every time the EC-Master recovers from a line break this code will be signaled once to the application. How many SubDevices are on the main interface and how many are on the redundancy interface is stored in structure *EC_T_RED_CHANGE_DESC* of the union element RedChangeDes:

struct *EC_T_RED_CHANGE_DESC*

EC_T_WORD *wNumOfSlavesMain*
EC_T_WORD *wNumOfSlavesRed*

2.3.3 emNotify - EC_NOTIFY_JUNCTION_RED_CHANGE

If a SubDevice is detected with port 0 not connected, this error code will be signaled to the application. Either due to a cabling error or a line break in the junction redundancy. This notification is disabled by default.

struct **EC_T_JUNCTION_RED_CHANGE_DESC**

Public Members

EC_T_SLAVE_PROP **SlaveProp**
Slave properties

EC_T_BOOL **bLineBreak**
EC_TRUE for line break, EC_FALSE for line fixed

EC_T_WORD **wPort**
Port

struct **EC_T_SLAVE_PROP**

Public Members

EC_T_WORD **wStationAddress**
: Configured station address or INVALID_FIXED_ADDR

EC_T_WORD **wAutoIncAddr**
: Configured auto increment address or INVALID_AUTO_INC_ADDR

EC_T_CHAR **achName**[MAX_STD_STRLEN]
: Configured name of the slave device (NULL terminated string)

3 Sample integration

3.1 Redundancy support in EcMasterDemo

Since Version 2.5 redundancy support is automatically enabled in EcMasterDemo when giving two Real-time Ethernet Driver definitions at the command line. Different Real-time Ethernet Driver types are supported as well as using the same type twice. Since V3.2.2.03 the parameter -junctionred sets the redundancy mode to automatic, otherwise it is disabled.

The following statement will set the device with IP 192.168.0.120 as primary and the device with IP 192.168.1.120 as secondary EtherCAT® interface, both using winpcap Ethernet Driver:

```
EcMasterDemo -f Network-ENI.xml -ndis 192.168.0.120 1 -ndis 192.168.1.120 1
```

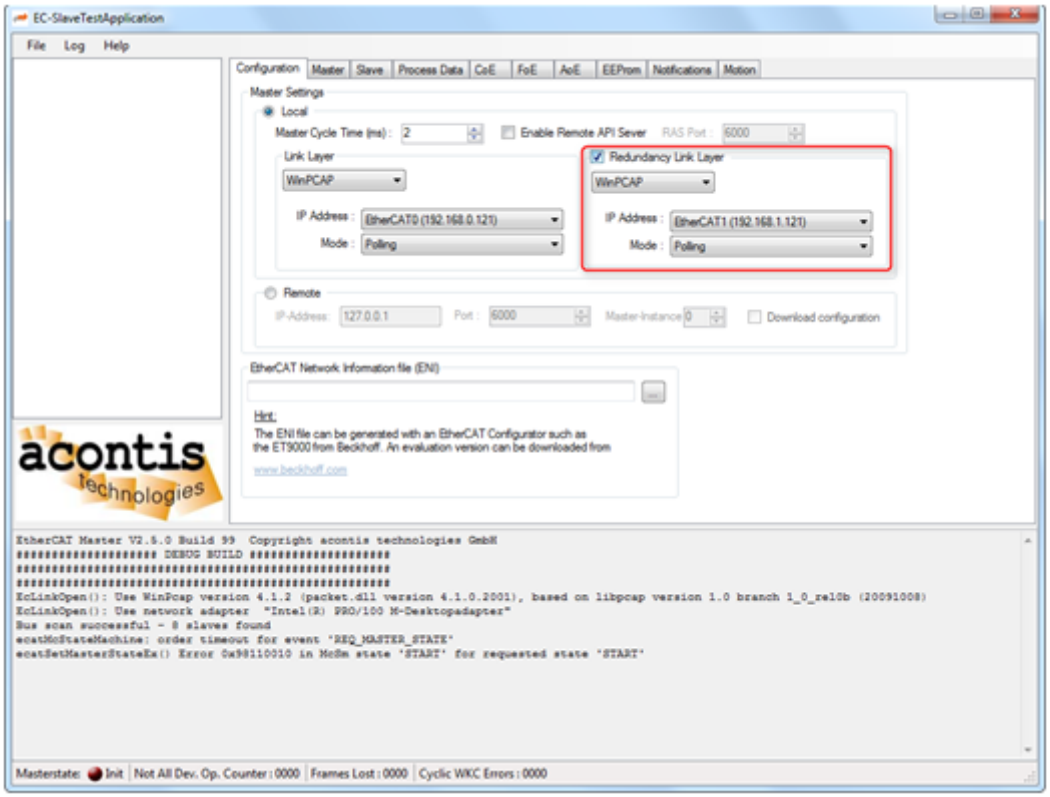
EcMasterDemo logs the usage of both interfaces by means of the following lines:

```
EcLinkOpen(): Use WinPcap version 4.1.2 (packet.dll version 4.1.0.2001), based on␣
↪libpcap version 1.0 branch 1_0_rel0b (20091008)
EcLinkOpen(): Use network adapter  "Intel(R) PRO/100 M-Desktopadapter"
EcLinkOpen(): Use WinPcap version 4.1.2 (packet.dll version 4.1.0.2001), based on␣
↪libpcap version 1.0 branch 1_0_rel0b (20091008)
EcLinkOpen(): Use network adapter  "Realtek RTL8168B/8111B PCI-E Gigabit Ethernet␣
↪NIC"
```

The network adapter names will likely differ from the ones giving in the sample above depending on the EcMaster-Demo host machine and the engaged Real-time Ethernet Driver.

3.2 Redundancy support in EC-SlaveTestApplication

The “Redundancy Real-time Ethernet Driver” definition in the EC-SlaveTestApplication activates redundant Ether-CAT® network interfaces usage as demonstrated in the following screenshot:


EC-SlaveTestApplication
File
Log
Help
Configuration
Master
Slave
Process Data
CoE
FoE
AoE
EEProm
Notifications
Motion
Master Settings
Local
Master Cycle Time (ms) :
2
Enable Remote API Sever
RAS Port :
6000
Link Layer
WinPCAP
IP Address :
EtherCAT0 (192.168.0.121)
Mode :
Polling
Redundancy Link Layer
WinPCAP
IP Address :
EtherCAT1 (192.168.1.121)
Mode :
Polling
Remote
IP-Address:
127.0.0.1
Port :
6000
Master Instance
0
Download configuration
EtherCAT Network Information file (ENI)
Hint:
The ENI file can be generated with an EtherCAT Configurator such as
the ET9000 from Beckhoff. An evaluation version can be downloaded from
www.beckhoff.com
acontis
technologies
EtherCAT Master V2.5.0 Build 99 Copyright acontis technologies GmbH
########## DEBUG BUILD ##########
EcLinkOpen(): Use WinPcap version 4.1.2 (packet.dll version 4.1.0.2001), based on libpcap version 1.0 branch 1_0_rel0b (20091008)
EcLinkOpen(): Use network adapter "Intel(R) PRO/100 M-Desktopadapter"
Bus scan successful - 0 slaves found
ecatMoStateMachine: order timeout for event 'REQ_MASTER_STATE'
ecatSetMasterStateEx() Error 0x98110010 in MoSm state 'START' for requested state 'START'
Masterstate:
Init
Not All Dev. Op. Counter : 0000
Frames Lost : 0000
Cyclic WKC Errors : 0000

4 Troubleshooting

4.1 Evaluating the notifications

The EcMasterDemo demonstrates how to consume the notifications (*emNotify - EC_NOTIFY_RED_LINEBRK* and *emNotify - EC_NOTIFY_RED_LINEFIXED*).

Within the EcMasterDemo logs, the messages *Line is fixed* and *Line break between SubDevice <x> and <y>`* indicate the line state changes that application received as notifications.

See also:

EcMasterDemo for how to start with two interfaces and `EcNotification.cpp` for details about the implementation.

4.2 Tracing main and redundancy frames

When tracing a redundancy enabled setup there are two different network adapters that send and receive EtherCAT® frames. In order to know which frames are sent and which are responses from EtherCAT® SubDevices, the source MAC address gives the needed information. It also includes additional state information. The destination MAC address of the EtherCAT® frames are fixed to the EtherCAT® unicast address 01:01:05:01:00:00.

The source MAC address of frames sent from any EtherCAT® interface of the Control Unit contains a part of the main EtherCAT® interface's MAC address (e.g. 00:0e:0c:61:3d:7c). Before sending the frames, the source address gets modified to contain frame state information like the Redundancy Bit, Retry Index and the Redundancy Frame ID. Actual EtherCAT® SubDevices modify bit three in Byte 1 (mask: 0x02) to give clue about the Forwarding mode. According to the ET1100 datasheet, the Forwarding Rule bit is set if the ESC is configured to drop non-EtherCAT® frames. The Redundancy Bit is set for all frames sent at the redundant EtherCAT® interface and it is not set for all frames sent at the main EtherCAT® interface. The retry index is set to indicate sending retries of asynchronous frames. It defaults to Byte 2 of the MAC address and is incremented according to the retries. Byte three through five of the MAIN EtherCAT® interface's MAC address are kept as they are. The Redundancy Frame ID at the last byte is incremented with every frame per interface.

The frame is structured the following way:

Red. Bit Mask 0x80	Forwarding Mask 0x02	Retry Index MAC Byte 2 + Idx	MAC Byte 3	MAC Byte 4	MAC Byte 5	Red-FrameId
0x00	0x00	0x0e	0x0c	0x61	0x3d	0x01

The remaining bits of MAC Byte 1 (mask 0x7d) are reserved.

The following sample trace of EtherCAT® frames with main and redundancy link shows the Redundancy bit 0x08 at the first byte and the incrementing RedFrameId:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	00:0e:0c:61:3d:01	01:01:05:01:00:00	ECAT	60	'BRD': Len: 1, Adp 0x0, Ado 0x1
2	0.000114	08:0e:0c:61:3d:01	01:01:05:01:00:00	ECAT	60	'BRD': Len: 1, Adp 0x0, Ado 0x1
3	0.000992	00:0e:0c:61:3d:02	01:01:05:01:00:00	ECAT	60	'BRD': Len: 1, Adp 0x0, Ado 0x1
4	0.001070	08:0e:0c:61:3d:02	01:01:05:01:00:00	ECAT	60	'BRD': Len: 1, Adp 0x0, Ado 0x1
5	0.001999	00:0e:0c:61:3d:03	01:01:05:01:00:00	ECAT	60	'BRD': Len: 1, Adp 0x0, Ado 0x1
6	0.002066	08:0e:0c:61:3d:03	01:01:05:01:00:00	ECAT	60	'BRD': Len: 1, Adp 0x0, Ado 0x1
7	0.002990	00:0e:0c:61:3d:04	01:01:05:01:00:00	ECAT	60	'BRD': Len: 1, Adp 0x0, Ado 0x1
8	0.003055	08:0e:0c:61:3d:04	01:01:05:01:00:00	ECAT	60	'BRD': Len: 1, Adp 0x0, Ado 0x1

When receiving frames, the EC-Master-Stack performs consistency checks and fast error detection. By means of the Redundancy Bit the EC-Master Stack detects if the line between main and redundancy link is broken. It expects for frames received at the redundant interface that they were sent from main interface and therefore their Redundancy

Bit is not set. Frames received at main interface are marked with the Redundancy Bit if the line between main and redundancy link is not broken.

The Redundancy Frame ID at the last byte is used to merge received frames and to detect frame loss.