acontis technologies GmbH

SOFTWARE

EC-Master

Feature Pack Hot-Connect

Version 3.2

Edition: December 13, 2024

Contents

1 Introduction

Achieving a maximum flexibility with Hot-Connect.

The concept of Hot-Connect first of all refers to the connecting and disconnecting of slaves in a running (hot) system. However, this is just one of several possible scenarios. Much more often it is required to operate the system without a perfect match between the EtherCAT bus configuration (ENI file) and the actually connected slaves or wiring.

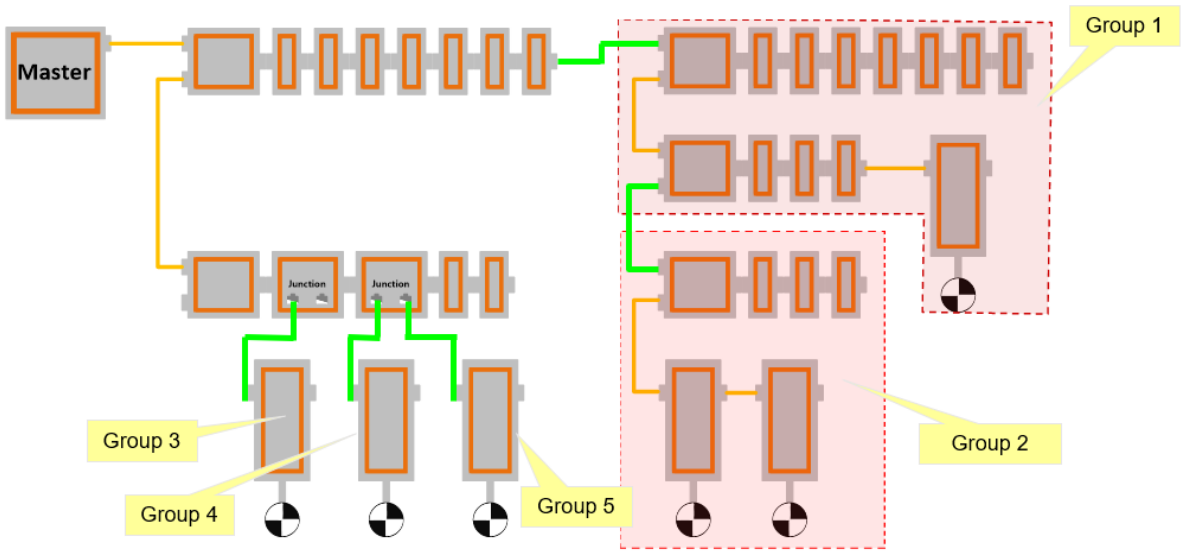
Thus the following additional use cases can be covered, without the need to change the ENI configuration file:

- Setting up a complex control system, while parts of the system are not available, powered-off or disconnected.
- Running a system that consists of mandatory as well as optional devices, e.g. in a test & measurement environment.
- Flexibility within the wiring: slaves can be connected to different ports, e.g. analogue to CAN.

In order for Hot-Connect to be used, no special EtherCAT functionality in the slave is required; in fact, any EtherCAT slave may be a member of a Hot-Connect group (HC group). Every HC group just has to be uniquely identifiable, most often this is implemented using DIP switches. This unique slave address then appears either in the Station Alias Register or at some address location within the slave memory. Both methods are supported by the EC-Master. Furthermore, the application may program the station alias address by means of the EC-Master, e.g. for first-time system initialization.

All EtherCAT activities required to support HC are automatically handled by the EC-Master in the background. There is no need for the application to interact. Besides, as soon as a slave is connected or disconnected, the application will be informed by a call-back function (notification). At any time the application may determine the actually connected slave devices using the appropriate EC-Master API function.

Within the HC feature pack the functionality Border Close offers additional security against connecting of slaves to an incorrect port. By activating that function, all EtherCAT ports are closed, except the ports which are permitted by the configuration. Therefore slaves which are connected to these ports, are simply ignored and the system continues to run perfectly undisturbed.



1.1 Functionality

- EtherCAT network can transferred to an operational state if optional slaves are missing.
- Operational state remains if an optional slave fails. Fail means, that the slave is driven powerless. The port of the previous slave is driven to closed loop automatically.
- Only one EtherCAT network configuration ENI for all possible slave connection combinations.
- Open/close individual ports via the application. To disconnect slaves from the bus or to allow them to participate again.

1.2 Limitations

- A slave is in an undefined state if it is electrically connected but fails in such a way that it cannot be set to the operational state. The slave must then be switched off manually.
- Mandatory HC-Groups are not allowed to be connected at optional slaves.
- A slave that is not configured in the ENI is not allowed.
- A slave with an unknown Hot-Connect ID is not allowed. If such a slave is detected, the bus can be cut off at this point by the master application.
- Synchronization with Distributed Clocks (DC):

  Connecting and disconnecting of slaves while DC is active is possible, but the SYNC signals are not absolute synchronous until the propagation delay measurement and compensation is finished.

  Technical background information:
  Each slave adds a small propagation of approximately one microsecond. In case of adding a new slave between to DC slaves this results in a shift of the SYNC signals.

2 Integration

This section gives an overview of the possibilities and usage of Hot-Connect. It describes the sequence of the notification messages that the application receives in case a Hot-Connect group was connected or disconnected.

No special adjustment of the EtherCAT application is necessary at all. Hot-Connect is configured in the EtherCAT configurator such as the EC-Engineer or the Beckhoff ET9000. These tools can be used to define so-called Hot-Connect groups consisting of one or more EtherCAT slaves.

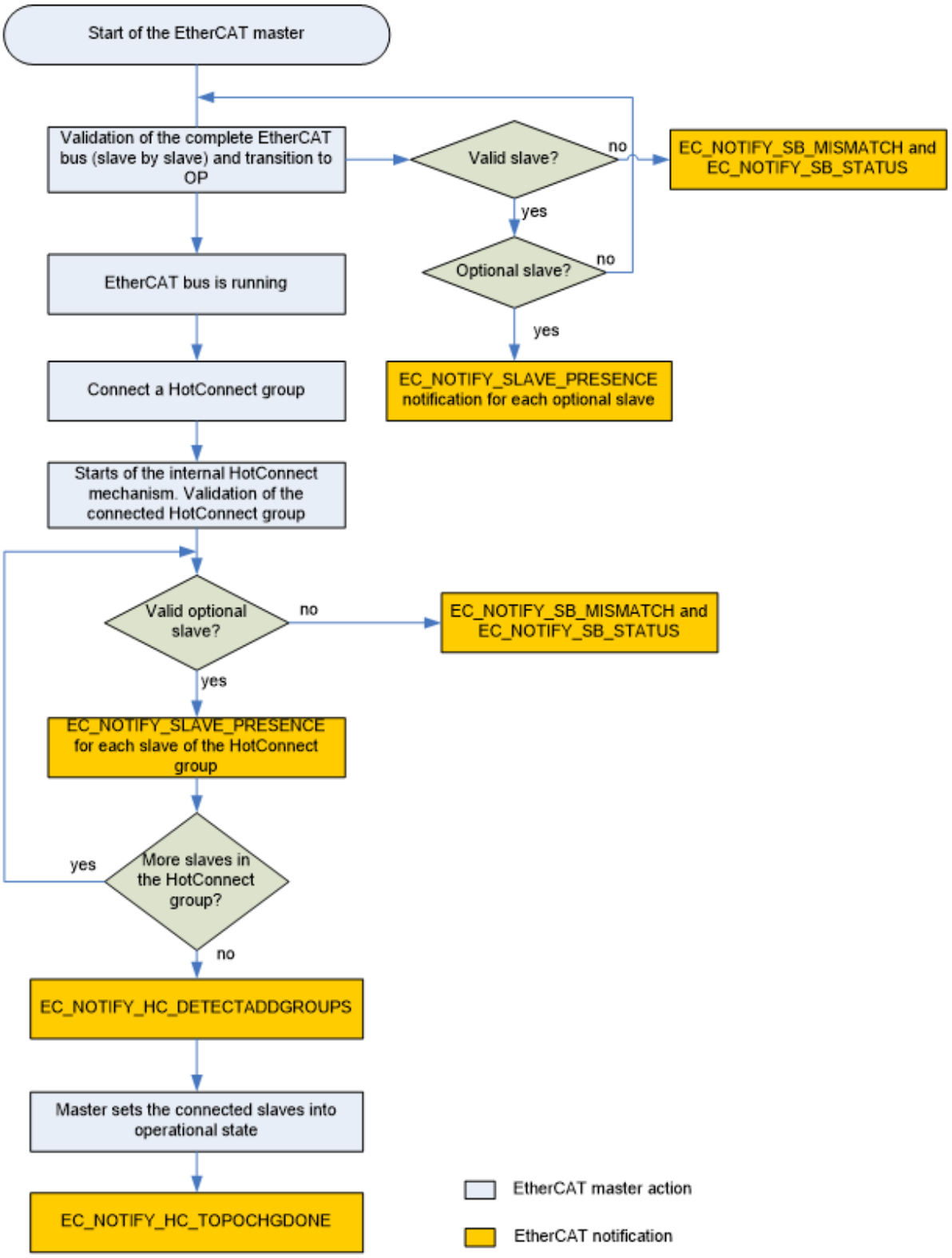
See also:

Configuration

2.1 Hot-Connect group connection sequence

The following sequence shows which notifications the application receives by connecting an EtherCAT slave that has been defined as a Hot-Connect group.

It is assumed that the EtherCAT network configuration file (ENI) has been correctly created and exported with an EtherCAT configurator and a valid Hot-Connect group has been connected to the EtherCAT bus.



1. The EC-Master will be started. The validation of the EtherCAT network against the ENI is executed and the transition to operational is done for all slaves. The application receives *emNotify - EC_NOTIFY_SLAVE_PRESENCE* notification for each slave of the currently connected optional Hot-Connect groups. If the validation fails the application receives an *emNotify - EC_NOTIFY_SB_MISMATCH* and an *emNotify - EC_NOTIFY_SB_STATUS* notification.
2. Now the EtherCAT network is running and a new Hot-Connect group will be connected, either by plugging or

power on.

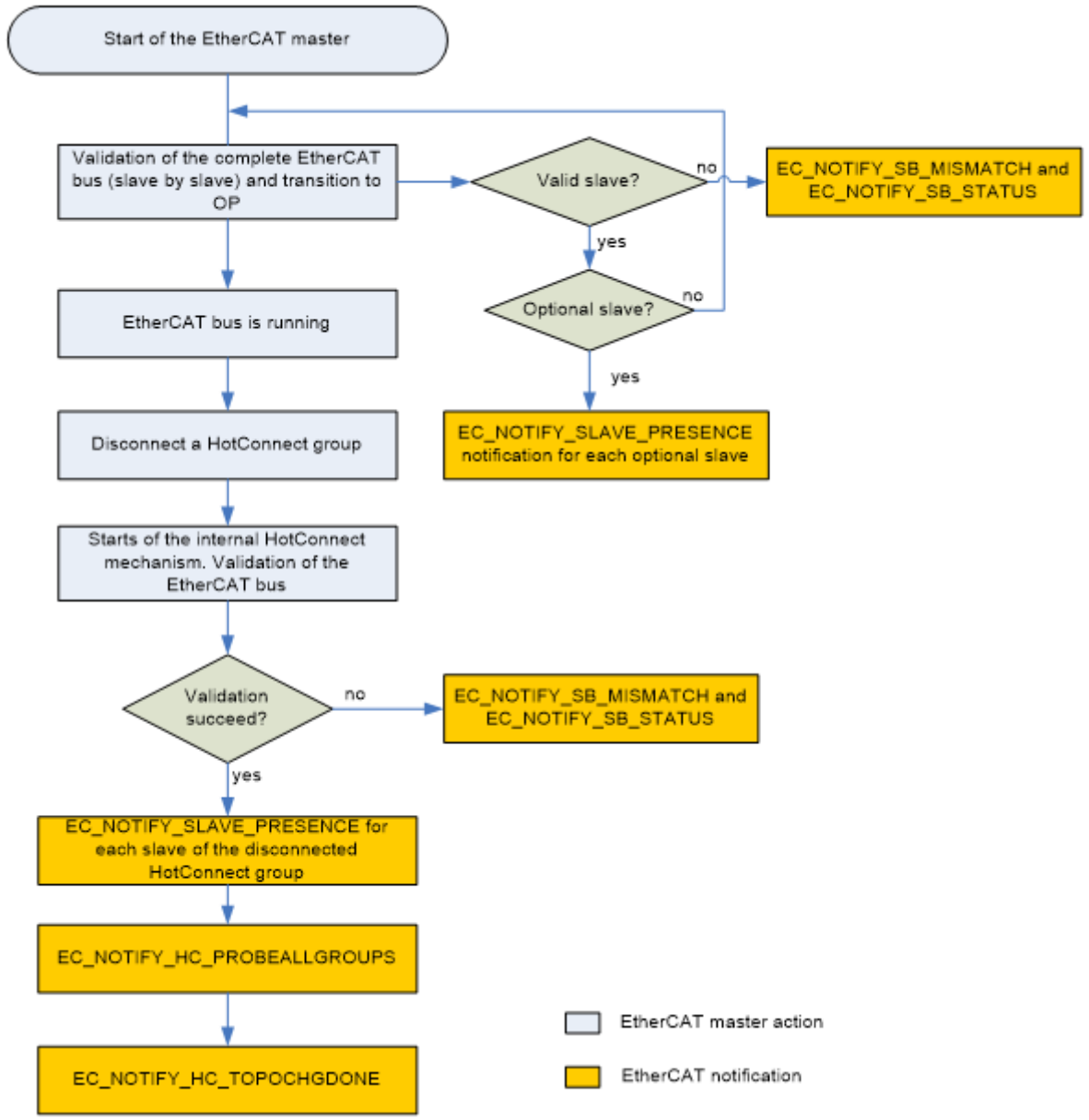
3. The EC-Master recognizes the topology change and starts the internal Hot-Connect state machine.
4. The EC-Master check whether the new connected Hot-Connect group is valid and therefore scans the whole network to identify the connected Hot-Connect group.
5. To identify the Hot-Connect group the vendor ID, product code and identification value of the Hot-Connect group must match to the ENI file.
6. If the EC-Master has identified the Hot-Connect group successfully, the application receives the *emNotify - EC_NOTIFY_SLAVE_PRESENCE* notification for each slave of the connected Hot-Connect group. If a slave could not successful validated, the application receives an *emNotify - EC_NOTIFY_SB_MISMATCH* and *emNotify - EC_NOTIFY_SB_STATUS* notification.
7. If the EC-Master could validate all the Hot-Connect groups successfully, the application receives the *emNotify - EC_NOTIFY_HC_DETECTADDGROUPS* notification which indicates that the detection of the all HotConnect group was successful finished.
8. The EC-Master now sets the connected slaves in operational state.
9. Finally the EC-Master signalizes with the *emNotify - EC_NOTIFY_HC_TOPOCHGDONE* notification, that the topology change was finished. The error code in the notification indicates whether the topology change was successful or not.

Note:

- *emNotify - EC_NOTIFY_SB_MISMATCH* and *emNotify - EC_NOTIFY_SB_STATUS* are common notifications that the application receives after a scan bus was performed. Please refer to the EC-Master manual for more information.
- In case the application tries to access a slave by the EC-Master API that is currently not connected to the bus, `EC_E_SLAVE_NOT_PRESENT` will be returned.

2.2 Hot-Connect group disconnection sequence

The disconnect case works analog to the connect case of a Hot-Connect group. The EC-Master of course does not need to validate new slaves, but must make sure that the network is valid after a Hot-Connect group was disconnected. If a mandatory slave, a slave that was not configured as a Hot-Connect group, was disconnected, the network validation will fail and the application receives *emNotify - EC_NOTIFY_SB_MISMATCH* and *emNotify - EC_NOTIFY_SB_STATUS* notifications. Finally the application will receive an *emNotify - EC_NOTIFY_HC_TOPOCHGDONE* with an error code. If the network could be validated successfully the application receives an *emNotify - EC_NOTIFY_SLAVE_PRESENCE* and finally an *emNotify - EC_NOTIFY_HC_TOPOCHGDONE* notification with `EC_E_NOERROR`.



2.3 Modes of operation

There are different modes how the EC-Master handles a Hot-Connect process and how the application can influence it.

2.3.1 Automatic mode

EC_T_EHOTCONNECTMODE::echm_automatic is the default Hot-Connect mode. The detection, topology change handling and slave state transition is automatically handled by the EC-Master.

2.3.2 Manual mode

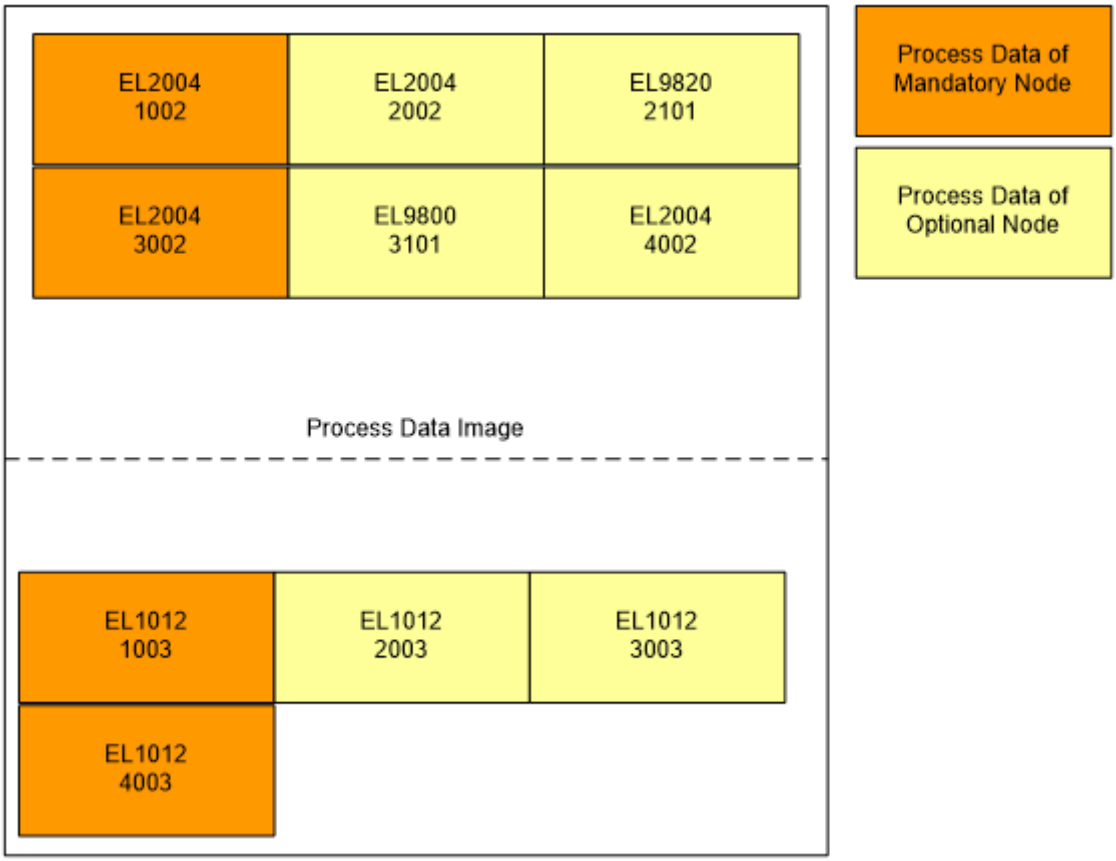
In the modes *EC_T_EHOTCONNECTMODE::echm_manual_preop* and *EC_T_EHOTCONNECTMODE::echm_manual_noreset* the EC-Master detects and notifies a topology change. The application can accept the topology change and allow the EC-Master a slave transition to the master state. *EC_T_EHOTCONNECTMODE::echm_manual_noreset* does not perform a slave state change via DEVICE_STATE_INIT.

2.3.3 Border close

EC_T_EHOTCONNECTMODE::echm_borderclose is an option and can be combined with the other modes. If this option is set, the still open ports in the configuration and in the EtherCAT network, respectively are closed automatically.

3 Process Data Image

All mandatory and optional slave are configured into a fixed process data image. The process data of absent slaves is not updated. The process data for each slave is mapped to its own process data image offset.



Since the frames on the wire are not changed by absent slaves following rules has to be taken in concern:

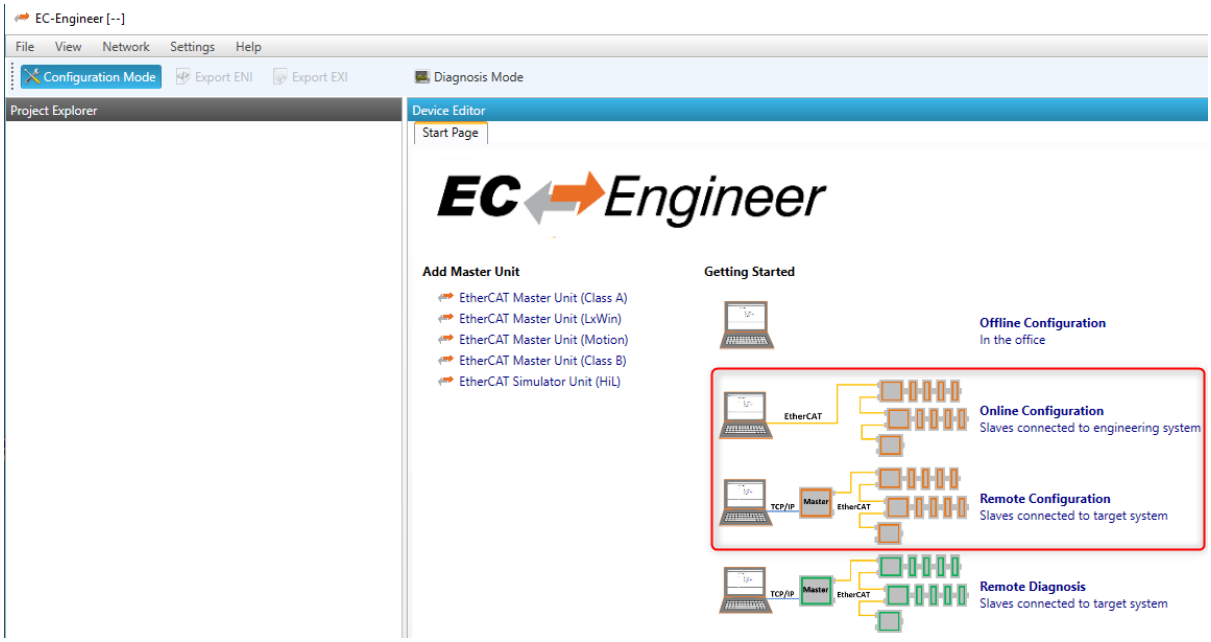
1. A Logical Write (LWR) is send, but the process data, which would hit an absent slave, has no effect.
2. A Logical Read (LRD) is send, but the process data, which would hit an absent slave, is overwritten by `0` because the EC-Master sends empty read frames.
3. A Logical Read/Write (LRW) is send, output process data located on positions where an absent slave would exchange in and out data. The out data sent is stored in the in data portion of the process data. To avoid this mirroring, it is recommended to use LWR/LRD instead of LRW for process data exchange to optional nodes. If this mirroring does not influence the application, it may still be used.

4 Configuration

This chapter explains how to configure Hot-Connect groups with group heads having rotary address switches in EC-Engineer.

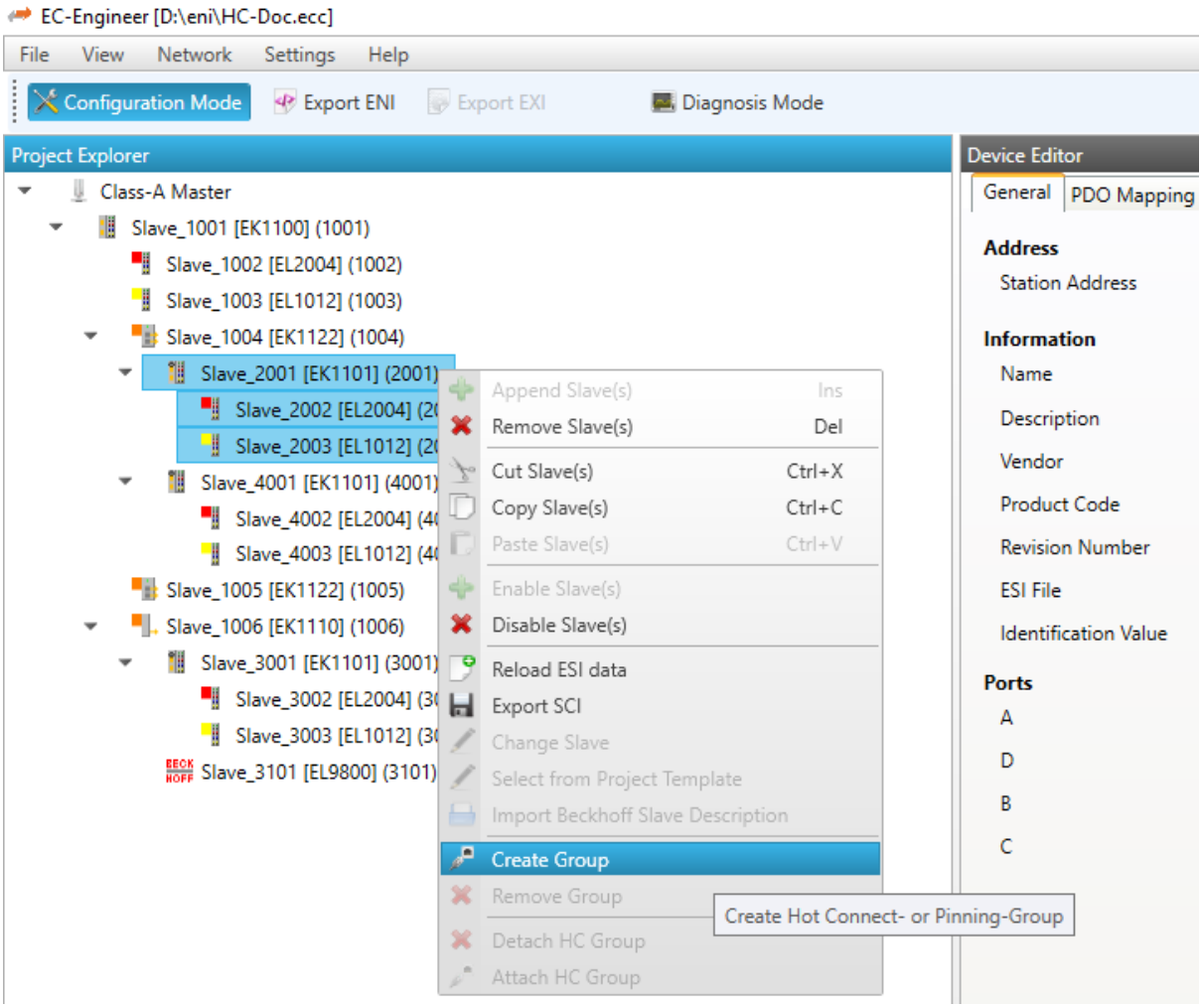
1. Scan the network or create a network by hand.

   The EC-Engineer automatically scans the network and displays the connected slaves in case of online configuration or remote configuration.



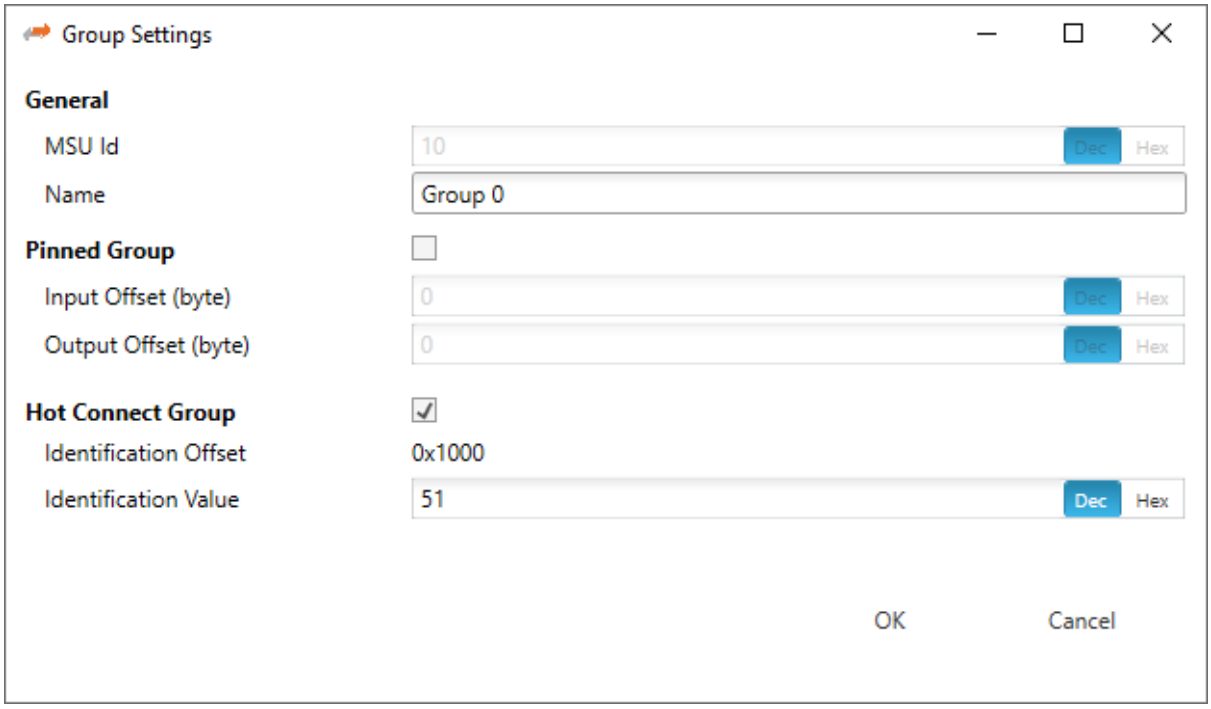
1. Create a Group

   In order to create a Hot-Connect group, a head slave and all topologically following slaves of the stub must be marked. The group can be created via the context menu.

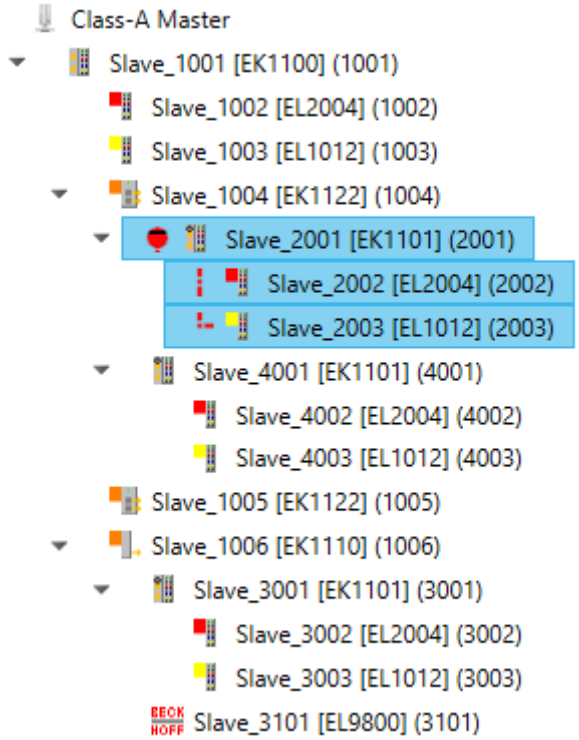


2. Configure Group Settings

   A Hot-Connect group configuration consists of a freely selectable name and an identification value. The identification value must be unique within the network. If the configuration was created by scanning the network, the identification value is automatically read from the slave and predefined in the field.

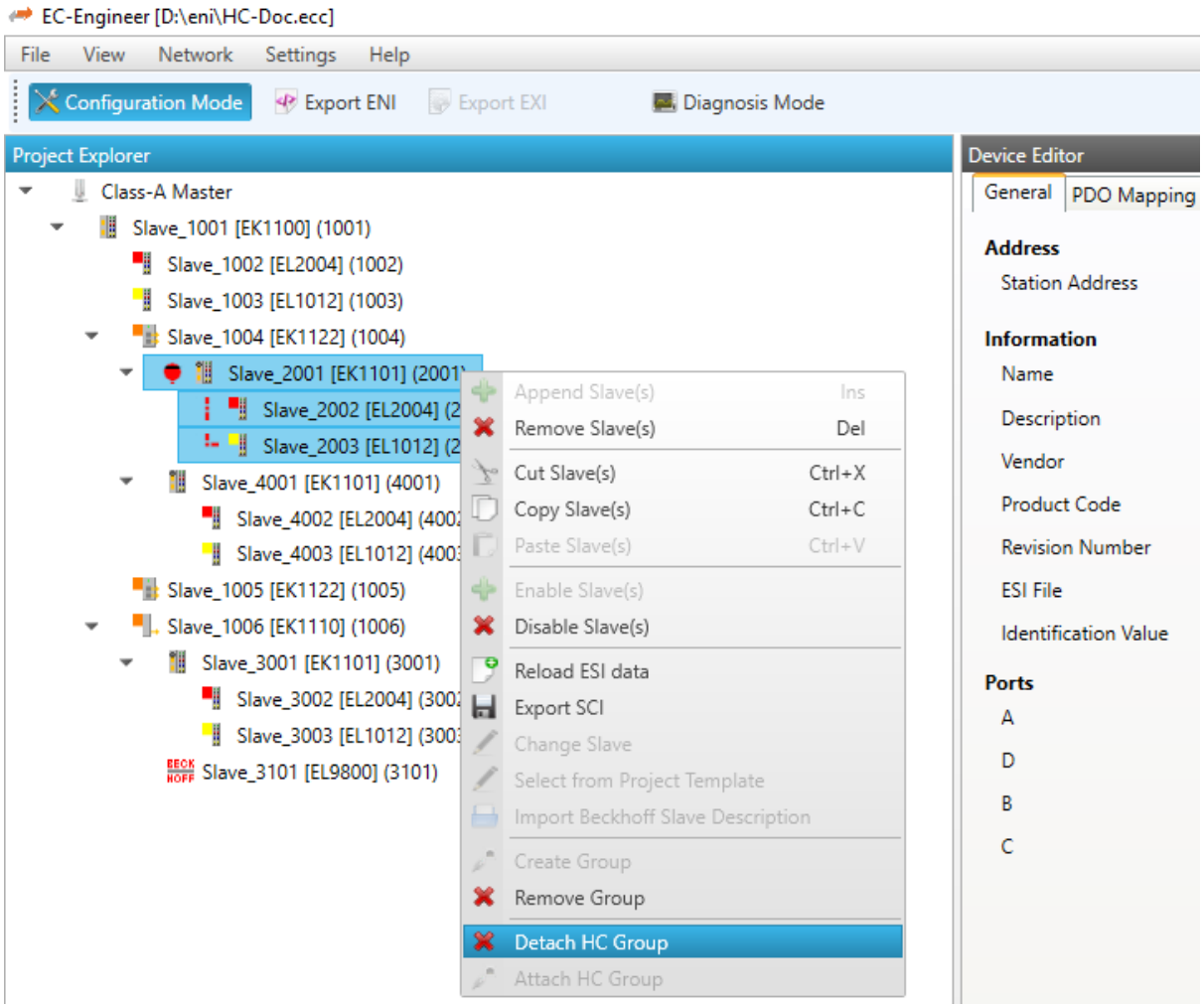


After creating the Hot-Connect group, it is marked with a indicating icon in the slave tree:

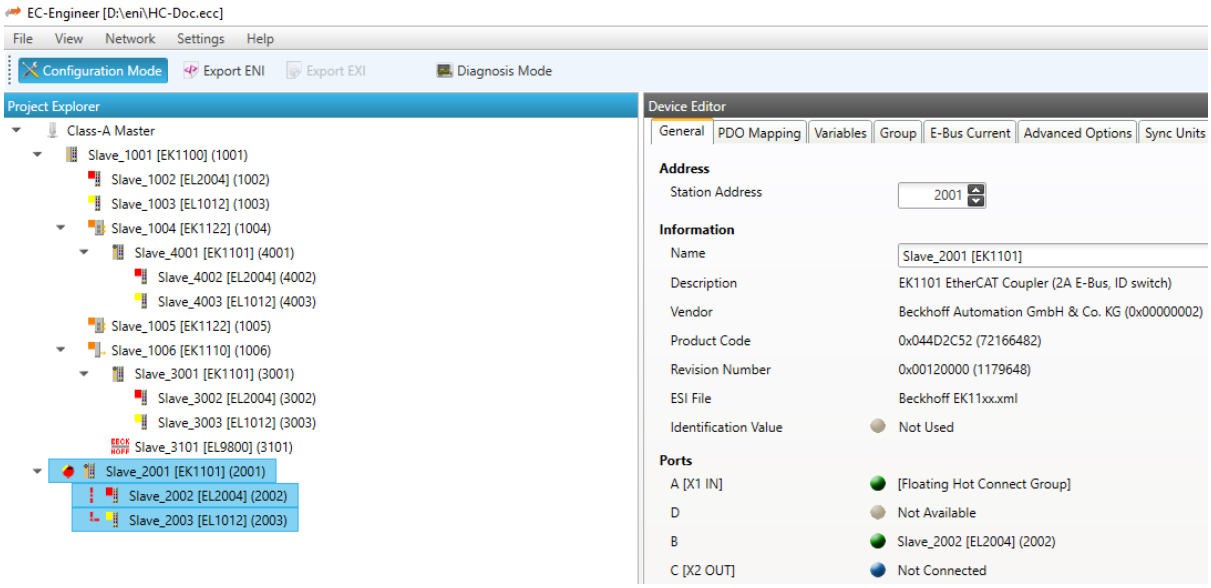


3. Group position restriction

   By default, when creating the Hot-Connect group, its location is restricted to the current position in the topology. The restriction can be removed by detaching the Hot-Connect group.

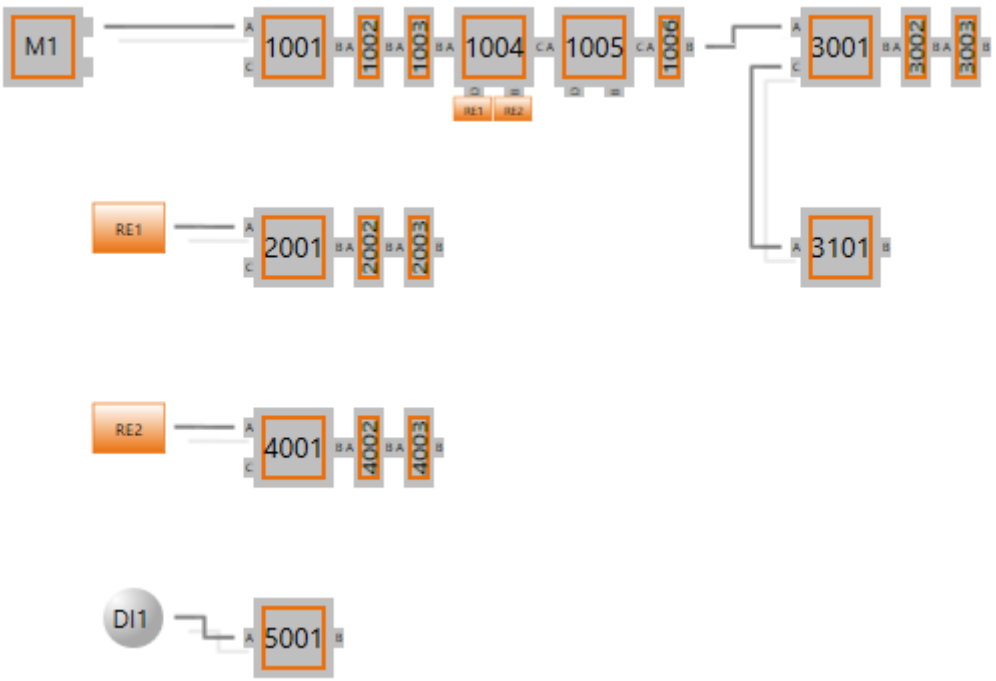


After detaching the group, its floating in the network and positioned at the end of slave tree:



5 Use cases

This chapter describes which use cases are covered by the current revision of EC-Master Hot-Connect Feature Pack. Each use case is described with a small snippet of pseudo-code, for the integration in the application. All use cases which are described refer to following topology:



5.1 Machine (plant) Start-Up

Having a complex hardware application or a huge plant, application engineers often have to manage the task of bringing the device into service section by section. While doing so there is the requirement of starting the device with the final configuration (ENI File). The Hot-Connect Feature Pack allows to startup the EC-Master with such an “incomplete” configuration.

Preface

To cover this use case it is required, that groups of slaves, which form a meaningful module, are combined into a Hot-Connect group within the ENI.

See also:

Create Hot-Connect groups

Operation

A subset of modules (Hot-Connect groups) or all slaves are connected to the control system and the EC-Master stack is started into operational mode. There is no requirement of application intervention for this use case.

5.2 Mandatory slave fails

Since the network configuration can contain optional and mandatory slaves, it may happen that a mandatory slave becomes inaccessible (power loss or disconnected).

-> This is by intention a scenario which Hot-Connect cannot cope with.

Scenario

1. **Slave** All connected slaves are in operational state
2. **Slave** Slave 4001 is shut down (power-off). The EtherCAT port on previous slave (1004 Port B) is loop closed automatically.
3. **Master** Slave fail is detected. Topology change was completed (with error) and notified to the EtherCAT application by means of a notification callback. EC_NOTIFY_HC_TOPOCHGDONE notification is only raised once.
4. **Master** Cyclic commands are continued.
5. **Application** Receives the notification and processes it.
6. **Master** Cyclic working counter errors are notified to the application.
7. **Application** Network is stopped by the application. `emSetMasterState()` INIT

5.3 Mandatory slave returns / is powered up

Since Hot-Connect cannot cover the use case *Mandatory slave fails*, no operation is performed on power up of a mandatory slave.

5.4 Shutdown of optional slave

Since the network configuration can contain optional and mandatory slaves, it may happen that an optional slave becomes inaccessible (power loss or disconnected). This is one of the main purposes of the Hot-Connect Feature Pack.

Scenario

1. **Slave** All connected slaves are in operational state.
2. **Slave** Slave 3101 is shut down (Power off). The EtherCAT Port on previous slave (3001 Port C) is loop closed automatically.
3. **Master** Slave fail is detected. Topology change was completed (with no error) and notified to the application by means of a notification callback. EC_NOTIFY_HC_TOPOCHGDONE notification is only raised once.
4. **Application** Receives the notification and processes it.
5. **Master** Network stays operational
6. **Master** Master corrects expected working counters, cyclic commands are continued.

5.5 Power up of an optional slave

An optional slave is connected to the network. It shall be integrated into the network and set to the master operation state.

Scenario

1. **Slave** All connected slaves are in operational state.
2. **Slave** Slave 3101 is (re-) connected to the Bus. The EtherCAT Port on previous slave (3001 Port C) is opened.
3. **Master** Cyclic commands are continued. Topology change was completed (with no error) and notified to the application by means of a notification callback. EC_NOTIFY _HC_TOPOCHGDONE notification is only raised once.
4. **Application** Receives the notification and processes it.
5. **Master** Master corrects expected working counter values, sets new connected slave to the current master state.
6. **Slave** New connected slave (3101) transits to master operation state.

5.6 Wrong / unknown slave connected to network

If a wrong slave is connected to the network, either not configured, wrong address or invalid location placement, the application may cut off the wrong slave and the subsequent slaves behind it.

Scenario

1. **Slave** All connected slaves are in operational state.
2. **Slave** Unknown slave (5001) is connected to the network. The EtherCAT port on previous slave (3001 Port C) is opened.
3. **Master** Cyclic commands are continued. EC-Master bus scan starts, detects an invalid slave and notifies EC_NOTIFY_SB_MISMATCH to the application. Topology change was completed (with error) and notified to the application by means of a notification callback. EC_NOTIFY _HC_TOPOCHGDONE notification is only raised once.
4. **Application** Receives EC_NOTIFY_SB_MISMATCH and EC_NOTIFY_HC_TOPOCHGDONE
5. **Application** Issues the block of the invalid slave by calling `emBlockNode()`
6. **Master** Issues a manual loop close on the previous slave's port (3001 Port C), which disconnects all slaves behind this port.
7. **Application** May poll manually the closed port(s) to be re-opened to check whether the wrong slave is disappeared. `emOpenBlockedPorts()`

5.7 Slave connected to the network at wrong position

If a known slave with dedicated previous port (Blue Group: Connectible only at the specified position) is connected to the wrong position, the application may handle this like *Wrong / unknown slave connected to network*, therefore this use case is described there.

5.8 Unexpected slave state

It may happen, that a slave is not responding to state change requests, because of a slave application error, or a slave drops back to a lower state because of any kind of slave side errors. In this case the Hot-Connect Feature Pack can be used to lock out such a faulty slave from disturbing the remaining network. Mandatory nodes must not be locked out by closing corresponding ports.

Scenario

1. **Slave** All connected slaves are in operational state.
2. **Slave** Optional slave (2101) is not accepting state change request or state change request results in timeout.
3. **Master** Slave state was not successful. The application is informed by the return value of a state change request, e.g. with `emSetMasterState()`, or of an init command error.
4. **Master** Cyclic commands are continued.
5. **Application** Receives the notification and processes it.
6. **Application** Locks out the faulty node by calling *`emSetSlavePortState()`*
7. **Master** Closes loop at requested position (2004 Port B)
8. **Application** Retries to reach the desired network state.

5.9 Connect of duplicate slave

In field applications it may happen, that an identical slave of one that is already present is connected. Means vendor id-, product code and the identification information (e.g. alias address) are identical. In this case the Hot-Connect instance is going to use the first found slave at the network as the expected one. The second detected slave is treated like connecting an invalid or unknown slave and is therefore handled like described in *Wrong / unknown slave connected to network*.

5.10 Border close operation

In some environment it may be required, that a network is protected against connection of new slaves. It can be required that the network consists of a vast amount of Hot-Connect groups which all can be connected only at a specific port. This can be the case for example on a tooling machine, where the tools are all groups of Hot-Connect slaves. To avoid bus disturbance by connecting any different slave at any point which is not desired one, the application may use the option Border Close of the Hot-Connect Feature Pack (*emIoControl - EC_IOCTL_HC_SETMODE*). This allows the EC-Master to close all ports to which no Hot-Connect groups may be connected.

Configuration

A network configuration ENI file with one or more Hot-Connect groups that can be connected to a specific point in the network (see section Blue Group: Connectible only at the specified position).

Scenario

1. **Application** Activate Border Close feature by calling of *emIoControl - EC_IOCTL_HC_SETMODE*

2. **Application** Sets the bus into OP state
3. **Master** Transits to state OP
4. **Master** Ports at the bus where no Hot-Connect group can be connected are set to port mode Closed. The Port where Hot-Connect groups may be connected remains in mode Open
5. **Slave** Ports are closed
6. **Slave** A slave is connected anywhere at the network, not at the Hot-Connect connection port
7. **Master** Since the other ports are closed, the EC-Master doesn't even recognize the connection of a slave.
8. **Slave** A Hot-Connect Group is connected to the Hot-Connect connection port
9. **Master** Performs a Topology Change
10. **Slave** Newly connected slave transit to state OPERATIONAL via INIT
11. **Master** Ports at the end of the added Hot-Connect group are also port closed
12. **Slave** Ports are closed
13. **Slave** Hot-Connect group is disconnected
14. **Master** Performs topology change, ports remain unchanged

6 Application programming interface, reference

6.1 emIoControl - EC_IOCTL_HC_SETMODE

Configures the Hot-Connect mode. Can be called at any time after `emInitMaster()`, usually before `emConfigureNetwork()`, but not necessarily. If it is not called, the Hot-Connect instance operates in automatic mode *EC_T_EHOTCONNECTMODE::echm_automatic*.

emIoControl - EC_IOCTL_HC_SETMODE

Parameter

- `pbyInBuf`: [in] Pointer to EC_T_EHOTCONNECTMODE
- `dwInBufSize`: [in] sizeof(EC_T_EHOTCONNECTMODE)
- `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

Enumeration containing Hot-Connect modes of operation. The modes can be or'ed together within the call.

enum **EC_T_EHOTCONNECTMODE**
Values:

enumerator **echm_unknown**
Unknown mode of operation

enumerator **echm_manual_preop**

Manual mode of operation.

The Hot-Connect instance detects and notifies a topology change to the application. During the detection a slave state transition to DEVICE_STATE_PREOP via DEVICE_STATE_INIT is executed for the newly detected slaves. The application can accept the topology change by calling *emHCAcceptTopoChange()*, this sets all newly detected slaves to the master state.

enumerator **echm_automatic**

Automatic mode of operation.

The Hot-Connect instance detects, accepts topology changes and performs a slave state transition to the master state without waiting for the confirmation of the application.

enumerator **echm_fullmanual**
Reserved

enumerator **echm_manual_noreset**

Manual mode of operation without slave reset.

The Hot-Connect instance detects and notifies a topology change to the application. The new slaves retain in their state, no slave state transition via DEVICE_STATE_INIT is executed during the detection.

The application can accept the topology change by calling *emHCAcceptTopoChange()*, this sets all newly detected slaves to the master state.

enumerator **echm_borderclose**

Activates border close.

If this option is set, the still open ports in the configuration and in the EtherCAT network, respectively are closed automatically. As a result, new slaves outside the configuration, which are connected to any port, do not disturb the running bus and trigger a topology change. If a configured Hot-Connect group is removed, the corresponding allowed port are remain open for connection.

6.2 emIoControl - EC_IOCTL_HC_GETMODE

Get the current Hot-Connect operating mode.

emIoControl - EC_IOCTL_HC_GETMODE

Parameter

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

Return
EC_E_NOERROR or error code

See also:

emIoControl - EC_IOCTL_HC_SETMODE

6.3 emIoControl - EC_IOCTL_HC_CONFIGURETIMEOUTS

Configures the Timeout values used by Hot-Connect.

emIoControl - EC_IOCTL_HC_CONFIGURETIMEOUTS

Parameter

- `pbyInBuf`: [in] Pointer to EC_T_HC_CONFIGURETIMEOUTS_DESC
- `dwInBufSize`: [in] sizeof(EC_T_HC_CONFIGURETIMEOUTS_DESC)
- `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

struct **EC_T_HC_CONFIGURETIMEOUTS_DESC**

Public Members

EC_T_DWORD **dwDetectionTimeout**
[in] Timeout [ms] for hot-connect group detection (default: 15000 ms)

6.4 emHCGetNumGroupMembers

static EC_T_DWORD **ecatHCGetNumGroupMembers** (EC_T_DWORD dwGroupIndex)

EC_T_DWORD **emHCGetNumGroupMembers** (
EC_T_DWORD dwInstanceID,
EC_T_DWORD dwGroupIndex
)
Get number of slaves belonging to a specific Hot-Connect group.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwGroupIndex** – [in] Index of Hot-Connect group, 0 is the mandatory group

Returns
Number of slaves

6.5 emHCGetSlaveIdsOfGroup

static EC_T_DWORD **ecatHCGetSlaveIdsOfGroup** (
EC_T_DWORD dwGroupIndex,
EC_T_DWORD *adwSlaveId,
EC_T_DWORD dwMaxNumSlaveIds
)
EC_T_DWORD **emHCGetSlaveIdsOfGroup** (
EC_T_DWORD dwInstanceID,
EC_T_DWORD dwGroupIndex,
EC_T_DWORD *adwSlaveId,
EC_T_DWORD dwMaxNumSlaveIds
)
Get the list of Slave IDs belonging to a specific Hot-Connect group.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwGroupIndex** – [in] Index of Hot-Connect group, 0 is the mandatory group
- **adwSlaveId** – [out] Preallocated Slave ID list buffer
- **dwMaxNumSlaveIds** – [in] Size of Slave ID list buffer

Returns
EC_E_NOERROR or error code

6.6 emHCAcceptTopoChange

static EC_T_DWORD **ecatHCAcceptTopoChange** (EC_T_VOID)

EC_T_DWORD **emHCAcceptTopoChange** (EC_T_DWORD dwInstanceID)

Accept last detected topology change.

If Hot connect is configured in manual mode by EC_IOCTL_HC_SETMODE, the master will generate the notifications EC_NOTIFY_HC_PROBEALLGROUPS or EC_NOTIFY_HC_DETECTADDGROUPS after a topology change was detected. This function will set all new detected slaves to the current master state.

Parameters

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

Returns

EC_E_NOERROR or error code

6.7 emForceTopologyChange

static EC_T_DWORD **ecatForceTopologyChange** (EC_T_VOID)

EC_T_DWORD **emForceTopologyChange** (EC_T_DWORD dwInstanceID)

Force changed topology.

Trigger HC State Machine

Parameters

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

Returns

EC_E_NOERROR or error code

6.8 emSetSlavePortState

static EC_T_DWORD **ecatSetSlavePortState** (
EC_T_DWORD dwSlaveId,
EC_T_WORD wPort,
EC_T_BOOL bClose,
EC_T_BOOL bForce,
EC_T_DWORD dwTimeout
)

EC_T_DWORD **emSetSlavePortState** (
EC_T_DWORD dwInstanceID,
EC_T_DWORD dwSlaveId,
EC_T_WORD wPort,
EC_T_BOOL bClose,
EC_T_BOOL bForce,
EC_T_DWORD dwTimeout
)

Open or close slave port.

This function allows to open or close a specific slave port in different ways. It also can be used to re-open ports closed by a rescue scan.

Parameters

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

- **dwSlaveId** – [in] Slave ID
- **wPort** – [in] Port to open or close. Can be ESC_PORT_A, ESC_PORT_B, ESC_PORT_C, ESC_PORT_D
- **bClose** – [in] EC_TRUE: close port, EC_FALSE: open port
- **bForce** – [in] EC_TRUE: port will be closed or open, EC_FALSE: port will be set in AutoClose mode
- **dwTimeout** – [in] Timeout [ms]

Returns

- EC_E_NOERROR if successful
- EC_E_INVALIDSTATE if master isn't initialized
- EC_E_INVALIDPARM if dwInstanceID is out of range
- EC_E_SLAVE_NOT_PRESENT if slave not present
- EC_E_NOTFOUND if no slave matching dwSlaveId can be found
- EC_E_MASTER_RED_STATE_INACTIVE if Master Redundancy is configured and master is inactive

6.9 emBlockNode

static EC_T_DWORD **ecatBlockNode** (
    EC_T_SB_MISMATCH_DESC *pMisMatch,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD **emBlockNode** (
    EC_T_DWORD dwInstanceID,
    EC_T_SB_MISMATCH_DESC *pMisMatch,
    EC_T_DWORD dwTimeout
)

Blocks a slave on a specific port.

If an invalid slave node is connected, which happens bus topology scan to fail, the previous port, where this node is connected to can be shut down with this call. This allows a Hot-Connect system to be not disturbed, if unknown nodes are connected. If this function will be executed on a Hot-Connect member (a slave which is part of a hot connect group) the complete hot connect group will be excluded from the bus. This function may only be called from within the JobTask's context with parameter dwTimeout set to EC_NOWAIT.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pMisMatch** – [in] Pointer to *EC_T_SB_MISMATCH_DESC* carrying mismatch descriptor.
- **dwTimeout** – [in] Timeout [ms]. The function will block at most for this time.

Returns

EC_E_NOERROR or error code

6.10 emOpenBlockedPorts

static EC_T_DWORD **ecatOpenBlockedPorts** (EC_T_DWORD dwTimeout)

EC_T_DWORD **emOpenBlockedPorts** (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwTimeout
)

Opens all blocked ports.

This call allows re-opening all blocked ports to check whether mismatch cause is removed from bus. This function may only be called from within the JobTask's context with parameter dwTimeout set to EC_NOWAIT.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwTimeout** – [in] Timeout [ms]. The function will block at most for this time.

Returns

EC_E_NOERROR or error code

6.11 emNotify - EC_NOTIFY_HC_TOPOCHGDONE

This notification is triggered when the Hot-Connect instances have fully processed a topology change. This means more precisely when the slaves have reached the current bus state.

emNotify - EC_NOTIFY_HC_TOPOCHGDONE

Parameter

- `pbyInBuf`: [in] Pointer to EC_T_DWORD (EC_E_NOERROR on success, Error code otherwise)
- `dwInBufSize`: [in] sizeof(EC_T_DWORD)
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

6.12 emNotify - EC_NOTIFY_HC_DETECTADDGROUPS

This notification is triggered when the Hot-Connect group detection is completed after adding the slaves of a group.

emNotify - EC_NOTIFY_HC_DETECTADDGROUPS

Parameter

- `pbyInBuf`: [in] pointer to notification descriptor EC_T_HC_DETECTALLGROUP_NTFY_DESC
- `dwInBufSize`: [in] sizeof(EC_T_HC_DETECTALLGROUP_NTFY_DESC)
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0

- `pdwNumOutData`: [out] Should be set to EC_NULL

struct **EC_T_HC_DETECTALLGROUP_NTFY_DESC**

Public Members

EC_T_DWORD **dwResultCode**
Result of Group detection

EC_T_DWORD **dwGroupCount**
Total number of Groups

EC_T_DWORD **dwGroupsPresent**
Number of connected groups

EC_T_DWORD **dwGroupMask**
Bitmask of first 32 Groups. 1 = present, 0 = absent

EC_T_DWORD **adwGroupMask**[100]
Bitmask of first 3200 Groups.

6.13 emNotify - EC_NOTIFY_HC_PROBEALLGROUPS

This notification is triggered when the Hot-Connect group detection is completed after a group disappears.

emNotify - EC_NOTIFY_HC_PROBEALLGROUPS

Parameter

- `pbyInBuf`: [in] pointer to notification descriptor EC_T_HC_DETECTALLGROUP_NTFY_DESC
- `dwInBufSize`: [in] sizeof(EC_T_HC_DETECTALLGROUP_NTFY_DESC)
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

struct *EC_T_HC_DETECTALLGROUP_NTFY_DESC*

EC_T_DWORD *dwResultCode*
EC_T_DWORD *dwGroupCount*
EC_T_DWORD *dwGroupsPresent*
EC_T_DWORD *dwGroupMask*
EC_T_DWORD *adwGroupMask*[100]

6.14 emNotify - EC_NOTIFY_SB_STATUS

Scan bus status notification.

This notification is enabled by default.

emNotify - EC_NOTIFY_SB_STATUS

Parameter

- `pbyInBuf`: [in] Pointer to EC_T_SB_STATUS_NTFY_DESC
- `dwInBufSize`: [in] Size of the input buffer provided at pbyInBuf in bytes.
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

struct **EC_T_SB_STATUS_NTFY_DESC**

Public Members

EC_T_DWORD **dwResultCode**
: [in] EC_E_NOERROR: success EC_E_NOTREADY: no bus scan executed EC_E_BUSCONFIG_MISMATCH: bus configuration mismatch Result of scanbus

EC_T_DWORD **dwSlaveCount**
: [in] number of slaves connected to the bus

6.15 emNotify - EC_NOTIFY_SB_MISMATCH

This notification will be initiated if scan bus detects mismatch of connected slaves and configuration, due to unexpected slaves or missing mandatory slaves.

emNotify - EC_NOTIFY_SB_MISMATCH

Parameter

- `pbyInBuf`: [in] Pointer to EC_T_SB_MISMATCH_DESC
- `dwInBufSize`: [in] Size of the input buffer provided at pbyInBuf in bytes.
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

This notification is enabled by default. In case of permanent frame loss no slaves can be found although the slaves are connected.

struct **EC_T_SB_MISMATCH_DESC**

Public Members

EC_T_WORD **wPrevFixedAddress**
[in] Previous slave station address

EC_T_WORD **wPrevPort**
[in] Previous slave station address

EC_T_WORD **wPrevAIncAddress**
[in] Previous slave auto-increment address

EC_T_WORD **wBusAIncAddress**
[in] Unexpected slave (bus) auto-inc address

EC_T_DWORD **dwBusVendorId**
[in] Unexpected slave (bus) vendor ID

EC_T_DWORD **dwBusProdCode**
[in] Unexpected slave (bus) product code

EC_T_DWORD **dwBusRevisionNo**
[in] Unexpected slave (bus) revision number

EC_T_DWORD **dwBusSerialNo**
[in] Unexpected slave (bus) serial number

EC_T_WORD **wBusFixedAddress**
[in] Unexpected slave (bus) station address

EC_T_BOOL **bIdentificationError**
[in] Identification command sent to slave but failed

EC_T_WORD **wIdentificationAdo**
[in] Identification register

EC_T_WORD **wIdentificationVal**
[in] last identification value read from slave according to the last used identification method

EC_T_WORD **wIdentificationValExpected**
[in] Identification expected value

EC_T_WORD **wCfgFixedAddress**
[in] Missing slave (config) station Address

EC_T_WORD **wCfgAIncAddress**
[in] Missing slave (config) Auto-Increment Address

EC_T_DWORD **dwCfgVendorId**
[in] Missing slave (config) Vendor ID

EC_T_DWORD **dwCfgProdCode**
[in] Missing slave (config) Product code

EC_T_DWORD **dwCfgRevisionNo**
[in] Missing slave (config) Revision Number

EC_T_DWORD **dwCfgSerialNo**
[in] Missing slave (config) Serial Number

6.16 emNotify - EC_NOTIFY_SLAVE_PRESENCE

This notification is given, if slave appears or disappears from the network.

This notification is enabled by default.

emNotify - EC_NOTIFY_SLAVE_PRESENCE

Parameter

- `pbyInBuf`: [in] Pointer to EC_T_SLAVE_PRESENCE_NTFY_DESC
- `dwInBufSize`: [in] Size of the input buffer provided at pbyInBuf in bytes.
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

Disconnecting the slave from the network, powering it off or a bad connection can produce this notification.

struct **EC_T_SLAVE_PRESENCE_NTFY_DESC**

Public Members

EC_T_WORD **wStationAddress**
Slave station address

EC_T_BYTE **bPresent**
EC_TRUE: present , EC_FALSE: absent