NexECM

NEXCOBOT EtherCAT Master

User Manual

Manual Rev.: 1.9

Revision Date: June, 1th, 2019

Part No:

NEXCOBOT Inc.
www.nexcobot.com

Revise note:

Ver	Description
V1.9	2020/2/4 Add Ch4.4.3 NEC_RtGetSlaveProcessDataPtr Add Ch4.8.4 NEC_RtGetSlaveConfiguredAddressEni Add Ch4.4.5 NEC_RtGetSlaveTimeDiffWithRefSlave Add Ch4.4.6 NEC_RtGetSlaveCountEx Add Ch4.4.7 NEC_RtGetSlaveVendorId Add Ch4.4.8 NEC_RtGetSlaveProductCode Add Ch4.4.9 NEC_RtGetSlaveRevisionNo Add Ch4.9.1 NEC_RtGetCyclicTick Add Ch4.9.2 NEC_RtGetCyclicTime Add Ch4.9.3 NEC_RtGetLastCyclicLoad Add Ch4.9.4 NEC_RtIsCyclicOverLoad Add Ch4.9.5 NEC_RtResetCyclicOverLoad Add Ch4.9.6 NEC_RtGetCyclicOverLoadCount Add Ch4.9.7 NEC_RtClearProbe Add Ch4.9.8 NEC_RtGetMasterCycleConsumption Add Ch4.9.9 NEC_RtGetMasterActualCycleTime
V1.8	2019/6/1 Remove original chapters 2 and 3. Add Ch4.4.9 NEC_RtSetSlaveProcessDataOutputEx Add Ch4.4.10 NEC_RtGetSlaveProcessDataOutputEx Add Ch4.4.11 NEC_RtGetSlaveProcessDataInputEx Add Ch4.6.1 NEC_RtLoadNetworkConfig Add Ch5.2.5 NEC_LoadRtMaster Add Ch5.2.6 NEC_UnLoadRtMaster Add Ch5.2.7 NEC_LoadRtApp Add Ch5.3.9 NEC_GetMasterType Add Ch5.3.10 NEC_GetMasterCount Add Ch5.5.4 NEC_SetDo_s Add Ch5.7.4 NEC_RWProcessImagEx Add Ch5.7.5 NEC_RWSlaveProcessImageEx
V1.7	2016/5/2 Add Ch5.8.2 CoE-Emergency communication

	Add Ch6.7.7 NEC_RtStartGetODListCount Add Ch6.7.8 NEC_RtStartGetODList Add Ch6.7.9 NEC_RtStartGetObjDesc Add Ch6.7.10 NEC_RtStartGetEntryDesc Add Ch6.7.11 NEC_RtGetEmgDataCount Add Ch6.7.12 NEC_RtGetEmgData Add Ch7.7.5 NEC_GetODListCount Add Ch7.7.6 NEC_GetODList Add Ch7.7.7 NEC_GetObjDesc Add Ch7.7.8 NEC_GetEntryDesc Add Ch7.7.9 NEC_GetEmgDataCount Add Ch7.7.10 NEC_GetEmgData
V1.6	2015/12/11 Add Ch6.8.1 NEC_RtGetConfiguredAddress() API Add Ch6.8.2 NEC_RtGetAliasAddress() API Add Ch6.8.3 NEC_RtGetSlaveCoeProfileNum() API Add Ch7.9.1 NEC_GetConfiguredAddress() API Add Ch7.9.2 NEC_GetAliasAddress() API Add Ch7.9.3 NEC_GetSlaveCoeProfileNum() API
V1.5	2015/8/1 Add Ch6.6.2 NEC_RtGetSlaveInfomation() API Add Ch6.6.3 NEC_RtGetSlaveState() API
V1.4	2014/11/19 Ch2.1 Add support platform Ch2.3 Modify NIC driver setup Ch3 Enhance NEXCAT Utility description Ch6 & 7 Enhance and revise API descriptions
V1.3	2014/6/16 Add Chapter 7. NexECM Library (Win32API)
V1.2	2014/1/23 Modify chapter 2.3.1 How to config RTX TCP/IP
V1.1	2013/12/10 Add NEC_RtSetProcessdataPtrSource function

Contents

1. NexECM Introduction

NexECM is an EtherCAT Master Communication Protocol solution, it is based on RTOS running alongside with Micorsoft Windows to offer real-time communication between EtherCAT Master and EtherCAT slave devices. NexECM offers rich high level C / C++ API for rapid application development.

NexECM also provide a configuration utility: NexECM Studio, a graphic user interface tool for customer to edit parameters for EtherCAT communication between master and slave devices; its functions are as follow:
EtherCAT Slave devices scanning
Import ESI file,export ENI file
Configure EtherCAT slave devices
Monitoring the EtherCAT communication quality
Test functions for EtherCAT slave devices

Other detail functions, please refer to NexECM Studio User Manual.

Below is NEXCOBOT EtherCAT master system structure, the “UserApp” and “User Realtime Application” (dash line boxes) represent as user application, the arrows mean the communication relationship.

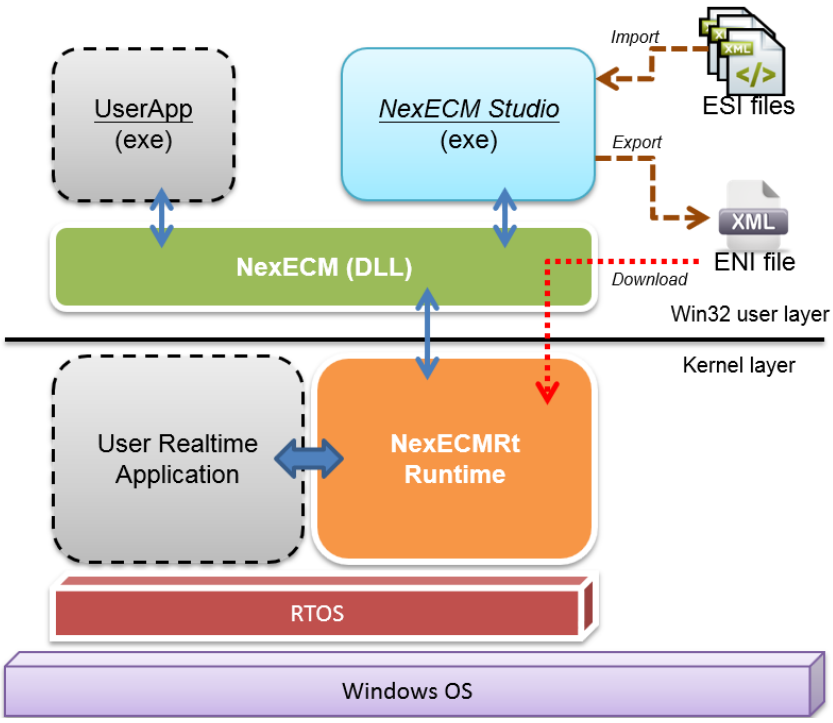


Figure 1.1: NEXCOM EtherCAT master solution system structure

The software module details as below table:

Module	Description
Operation at RTOS environment	
NexECMRt Runtime	EtherCAT Master stack running on RTOS Refer to: CH2, CH3 sections below
Operation at Win32 environment	
NexECM.dll	Win32 API libraries for controlling NeECMRt runtime Refer to:CH3 section below
NexECM Studio.exe	Graphic interface for configuring EtherCAT Master Refer to: NexECM Studio User Manual

1.1. NexECM Features

According to EtherCAT standard document: ETG.1500, NexECMRtx currently supports Master function, which is shown as in the list below

V: Ready, △: By Project Request

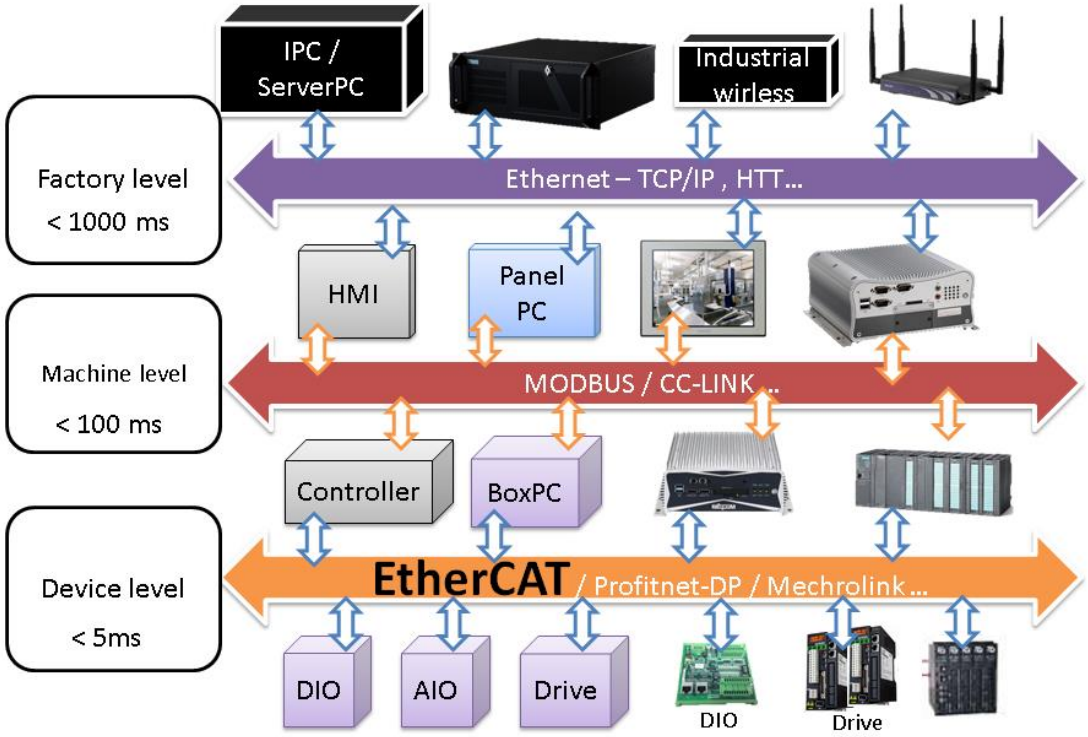
Feature name	Short description	NexECMRtx
Basic Features		
service Commands	Support of all commands	V
IRQ field in datagram	Use IRQ information from Slave in datagram header	V
Slaves with Device Emulation	Support Slaves with and without application controller	V
EtherCAT state Machine	Support of ESM special behavior	V
Error Handling	Checking of network or slave errors, e.g. Working Counter	V
Process data Exchange		
Cyclic PDO	Cyclic process data exchange	V
Network Configuration		
Reading ENI	Network Configuration taken from ENI file	V
Compare Network Configuration	Compare configured and existing network configuration during boot-up	V
Explicit Device Identification	Identification used for Hot Connect and prevention against cable swapping	V

Station Alias Addressing	Support configured station alias in slave, i.e. enable 2nd Address and use it	V
Access to EEPROM	Support functions to access EEPROM via ESC registerer	V
Mailbox Support		
Support Mailbox	Main functionality for mailbox transfer	V
Mailbox polling	Polling Mailbox state in slaves	V
CAN application layer over EtherCAT (CoE)		
SDO Up/Download	Normal and expedited transfer	V
Complete Access	Transfer the entire object (with all sub-indices) at Once	V
SDO Info service	services to read object dictionary	V
Emergency Message	Receive Emergency messages	V
Ethernet over EtherCAT (EoE)		
EoE	Ethernet over EtherCAT	△
File over EtherCAT (FoE)		
FoE	File over EtherCAT	△
Servo over EtherCAT (SoE)		
SoE	Servo over EtherCAT	△
Distributed Clocks		
DC	Support of Distributed Clock	V

2. EtherCAT Introduction

With the significant progress of communication technology, the Fildbus technology has been comprehensively applied in the field of industrial motion control. Ethernet has become the most common technology and has been widely applied in various fields related to internet technology. In addition, Ethernet hardware and software are developing fast with the wide application of this technology. As a result, Ethernet has become one of the most advanced and cost-efficient technology in the world today.

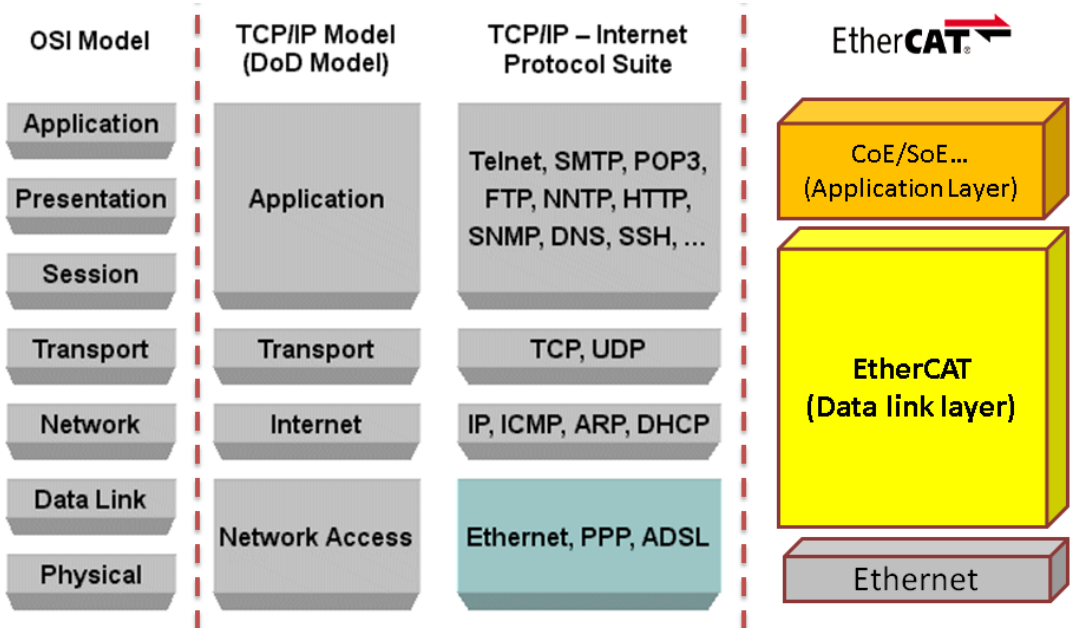
EtherCAT is designed on the base of Ethernet and is an industrial communication technology specifically designed for automation (especially machine automation). EtherCAT has the benefits of Ethernet (such as universality, high speed and low-cost) and its related products are developing in amazing speed. The diagram below illustrates an industrial communication system. It shows that EtherCAT is mainly used in the connection of high-speed and real-time I/O modules.



In the following sections the features of EtherCAT technology will be explained.

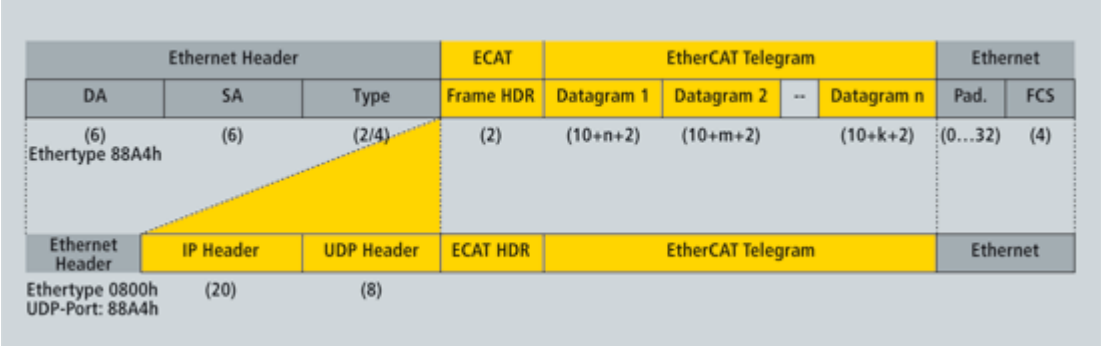
2.1. EtherCAT communication Protocol

EtherCAT is a kind of real-time communication protocol constructed on the base of Ethernet physical layer. This technology is currently maintained and promoted by EtherCAT Technology Group (ETG). As far as OSI internet model is concerned, EtherCAT is mainly defined in data link layer and Application layer.



The internet packet of EtherCAT uses IEEE802.3 Ethernet format. In addition to standard EtherCAT packet (EtherType = 0x88A4), EtherType can also use UDP format (EtherType = 0x0800), as shown in below diagram.

The content of EtherCAT is mainly composed of ECAT Frame header and Telegrams.

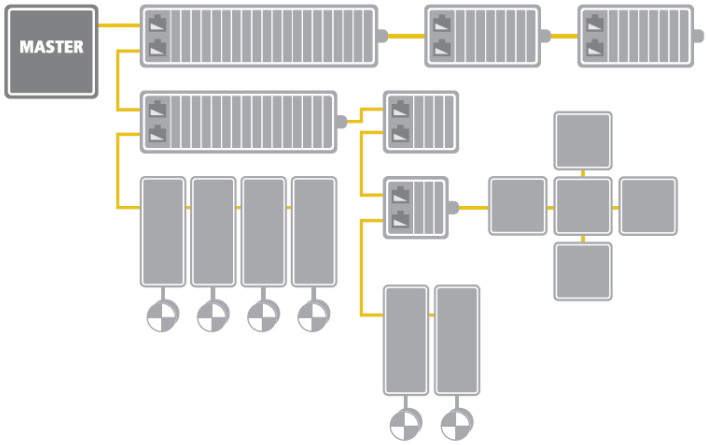


EtherCAT internet packet format

(Source of information: http://www.ethercat.org/)

2.2. Network Typology

The type of network topology decides the method to arrange the cables on field site. EtherCAT supports almost every kind of network topology, including line, tree and star. In addition, if we use standard 100BASE-TX internet cable, the distance between two devices can be as far as 100 meters. In conclusion, there is almost not any limit in the application of machine automation with the use of EtherCAT and standard 100BASE-TX network cable.

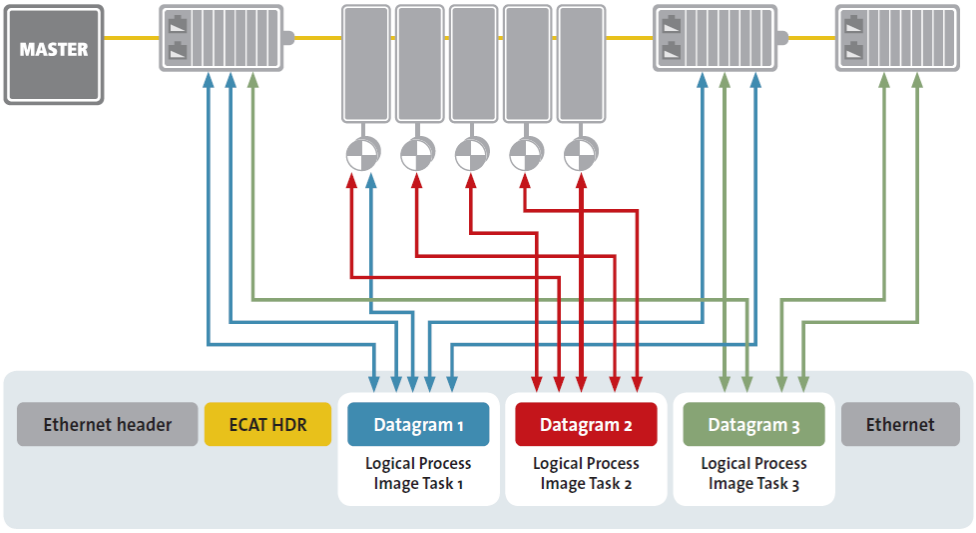


EtherCAT Network Topology, Line,Tree and Star Topology
(Source of information: http://www.ethercat.org/)

Using standard Ethernet network cable

2.3. High Speed Performance

Through addressing mode of EtherCAT and the memory control technology “Fieldbus Memory Management unit (FMMU)” performed by EC-Slaves hardware, we can exchange all data synchronically from all ECAT-Slaves on bus by just passing one single internet packet. The types of data include DIO, AIO and servo motor position etc. Please refer to the diagram as below.



EtherCAT: Exchange of internet packet and data
(Source of information: http://www.ethercat.org/)

The EtherCAT slaves are usually equipped with EtherCAT Slave IC (so called ESC). With the FMMU technology, the exchange of 1000 unit I/O data can be finished within only 30us. Data for 100 axis servo motor can be exchanged only within 100 us. Please refer to the table as below. (According to ETG data)

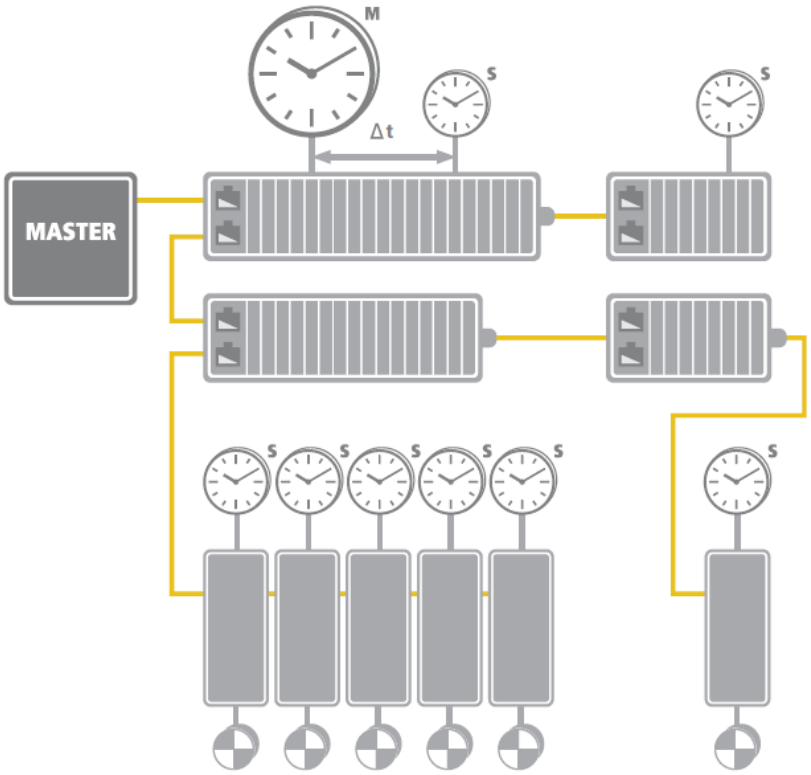
Process Data	Update Time
256 distributed digital I/O	11µs
1000 distributed digital I/O	30µs
200 channels analog I/O (16 Bits)	50µs (=20kHz)
100 servo axis (8 Bytes Input and output per axis)	100µs
1 Fieldbus Master-Gateway (1486 Bytes Input and 1486 Bytes output data)	150µs

EtherCAT Performance overview
(Source of information: http://www.ethercat.org/)

100us data exchange speed for axis control loop is equal to the control bandwidth 10 kHz. It is sufficient for doing speed close loop control and even for torque control.

2.4. Network Synchronization

For the application of automation, it's necessary that the "real-time" data transmission on the network communication. Moreover, "synchronization" is also a must among all devices (axes). Cases for example are the CNC machine of performing a line or arc interpolation for multiple servo axes and the control of gantry system for 2 axes etc.. For these kinds of application, synchronization among all devices is necessary to make sure accurate performance.

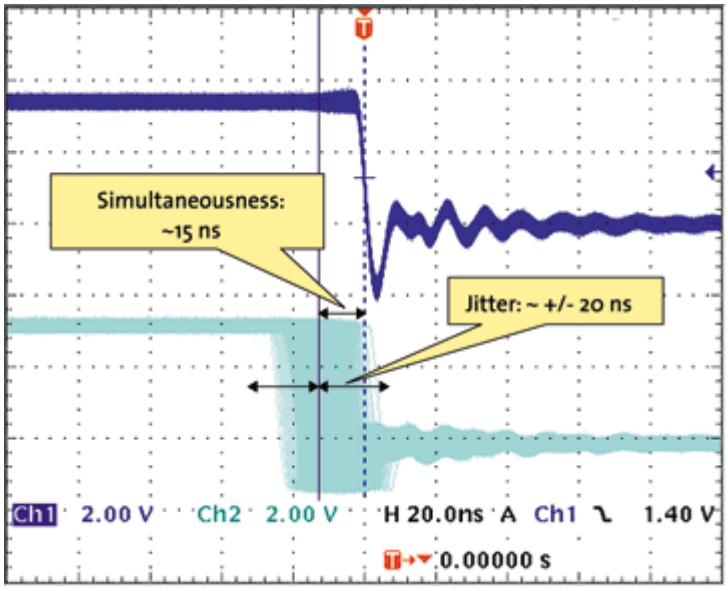


EtherCAT: Illustration of Distributed Clock (DC)
(Source of information: http://www.ethercat.org/)

The synchronization mechanism of EtherCAT is based on IEEE-1588 Precision Clock Synchronization Protocol and extends the definition to so-called Distributed-clock (DC). To put it simple, every EtherCAT ESC maintains a hardware based clock and the minimum time interval is 1 nano-second (64 bits in total). The time maintained by EC-Slave is called Local system time.

With accurate internet time synchronization mechanism and dynamic time compensation mechanism (*1), EtherCAT DC technology can guarantee that the time difference among every EC-Slave local system time is within +/- 20 nano-seconds. Following diagram is a scope view of two slave devices ouput digital signals. We can see that the time difference between the I/O signal from two EC-Slaves is around 20 nano-seconds.

(*1) Please refer to EtherCAT standard document ETG1000.4

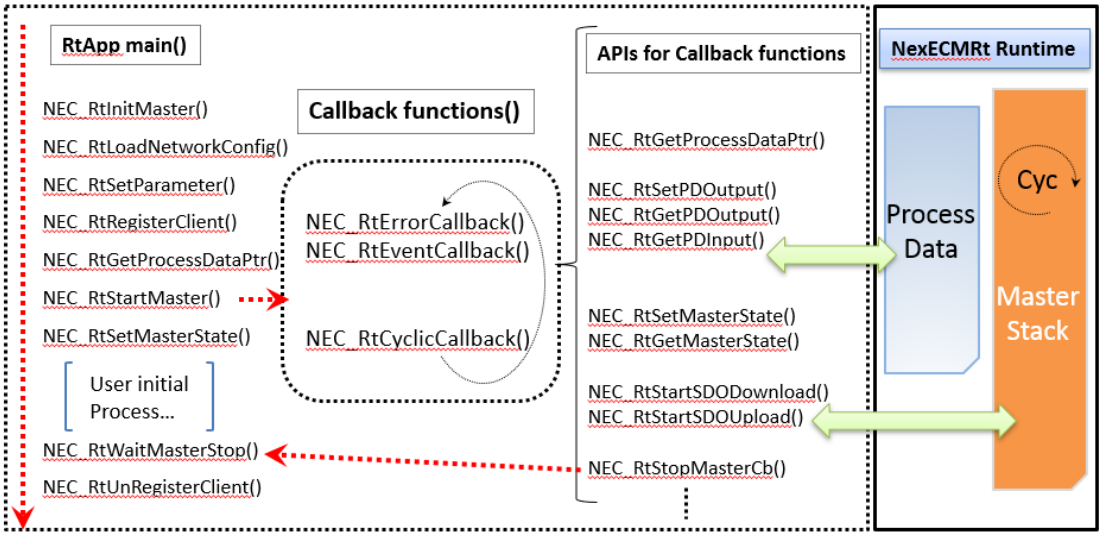


Synchronicity and Simultaneousness: Scope view of
two distributed devices with 300 nodes and 120 m of cable between them
(Source of information: http://www.ethercat.org/)

3. Principles of Programming

3.1. Basic Programming Framework

Below explains the NexECM basic programming framework.



Basic Flow:

Steps	Description	Related API
Your real-time Application main(): Ex. RtUserApp (Details API Description please refer to CH.4)		
1	Initial target EC-Master	NEC_RtInitMaster()
2	Load ENI	NEC_RtLoadNetworkConfig()
3	Set Parameters (ex: cycle-time…etc)	NEC_RtSetParameter() NEC_RtGetParameter()
4	Registerer Callback function function	NEC_RtregistererClient() NEC_RtUnregistererClient()
5	Get ProcessImage Memory pointer	NEC_RtGetProcessdataPtr() NEC_RtGetSlaveProcessdataPtr()
6	Start EtherCAT communication	NEC_RtStartMaster()
7	Switch EtherCAT's state to "**OP**" state	NEC_RtSetMasterstate()
8	Hang the main() thread. A main() type application, when all process is done, it will be unloaded. So, the main() thread need be hanged until the procedure	NEC_RtWaitMasterStop()

	provided by user (maybe in callback function) is done.	
In the period of Callback function (Details API Description please refer to CH.4)		
9	Your control program ProcessData access CoE communication	NEC_RtSetPDOutput() NEC_RtGetPDOutput() NEC_RtGetPDInput() NEC_RtStartSDODownload() NEC_RtStartSDOUpload()
10	End the application, stop EtherCAT communication	NEC_RtStopMasterCb()

The above process, you can refer to NexECM real-time sample program in installation directory.

3.2. ENI Load

ENI (EtherCAT Network Information) provides NexECM runtime the information of how to initialize those EC-Slave online and some parameters for NexECM runtime self. So before starting the the EtherCAT communication, the ENI must be loaded. Then, when start the communication, NexECM runtime will use those data to initial all of EC-Slaves online. If NexECM runtime detects the difference between ENI and current network topology, it will stop communication and trigger an error callback event (If you have registed an error callback function, it will be called).
You can use NexECM Studio to configure parameters for NexECM runtime and EC-Slaves and reproduce the ENI file, for detailed operation, please refer to NexECM Studio User Manual.

3.3. Initialization of NexECM

In your real-time application, call *NEC_RtInitMaster*() API to Initiate NexECM runtime. This action will not reset your ENI data from NexECM runtime.

Call *NEC_RtSetParameter*() API to complete the setting for NexECM runtime. Ex, Parameters: *NEC_PARA_S_ECM_CYCLETIMEUS* is used to set cyclic time of NexECM runtime.

Use *NEC_RtregistererClient*() to register Callback functions ,the Callback type:

void NEC_RtCyclicCallback(void *UserdataPtr);
void NEC_RtEventCallback (void *UserdataPtr, U32_T EventCode);
void NEC_RtErrorCallback (void *UserdataPtr, U32_T ErrorCode);

Callback	Description
NEC_RtCyclicCallback	Cyclical Callback function, implement control procedures
NEC_RtEventCallback	When a pre-defined event occurs, call by NexECM runtime
NEC_RtErrorCallback	When a pre-defined error event occurs, call by NexECM runtime

These Callback functions will be used to synchronize and exchange ProcessData(PDO) data with NexECM runtime. When EtherCAT's state(*) change to "SAFEOP" or "OP" in NexECM runtime, runtime will start ProcessData(PDO) data exchange with EC-Slaves at each cyclic communication.
(*)About the EtherCAT state machine, please refer to CH 3.6

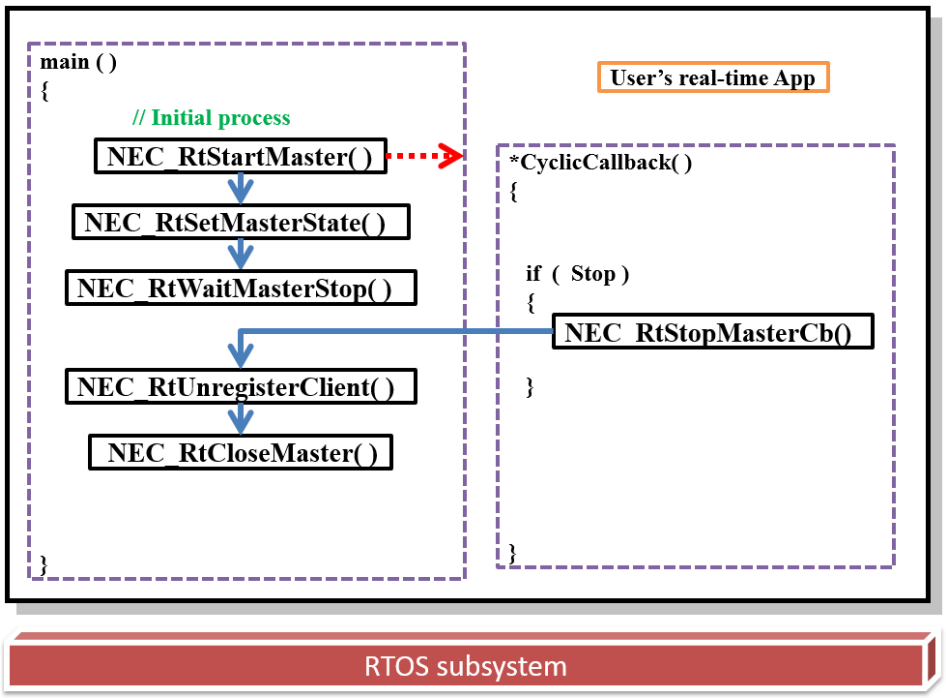
The most of users' application procedure will be put in the Callback functions. The basic programming principles of Callback function are to prevent from stoping or long executing time exceeded communication cycle. So, time-consuming API should be prohibited in Callback function. To know more about Callback function, please refer to Section 3.5.

3.4. Start Master communication

According to previous section, all settings for starting EtherCAT communication should be finished, then start the communication by calling *NEC_RtStartMaster()*. If the call is successul, the EherCAT communication will be cyclic execution and furthermore the Callback function is called cyclically by NexECM runtime. Please note that when the communication starts, the state of EtherCAT in NexECM runtime will keep in "INIT" state, until use *NEC_RtSetMasterstate*() to send request for state change.

Under normal circumstances, the user will implement control algorithms in the Callback function. However, real-time application is a console type program which there is a *main*() entery in. When the *main*() returns, the program ends and unloaded by real-time system. Then, the Callback function also will become invalid.

To prevent the program return, users can use *NEC_RtWaitMasterStop* (), so that the main thread of the program goes to suspend state and wait for EC-Master's stop signal.



About Callback function, please refer to CH3.5
About EtherCAT state machine, please refer to CH3.6

3.5. Callback Functions

3.5.1. Register Callback Function

NexECM offers three type of Callback functions:

Cyclic callback function
Event Callback function
Error Callback function

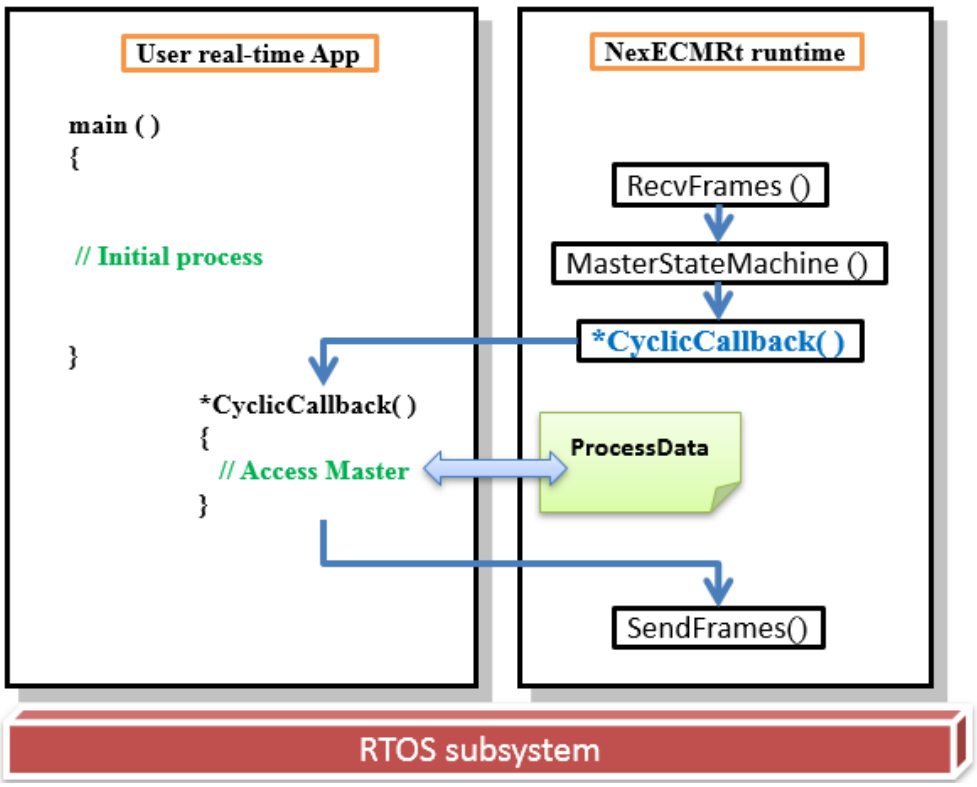
Before starting EtherCAT communication, use *NEC_RtRegisterClient*() to register Callback function in NexECM runtime. And use *NEC_RtUnregisterClient*() to release callback function from NexECM runtime, before the end of the real-time program. Please do not use *RtUnregisterClient*() to release callback function during communication.

3.5.2. Cyclic Callback Function

Use *NEC_RtGetProcessDataPtr*() API to get ProcessData memory pointer. After getting it, you can read/write freely, so we can treat this as a share memory, make it as a data exchange tunnel between user application and NexECM runtime. The cycle processes of NexECM runtime and user application are two independent running threads, so it is important to use share memory to keep the data consistent.

Furthermore, in the real-time industrial control applications, some applications must send data from EC-Master to all slave devices in each cycle to meet the real-time requirement. Such as control of the servo motor, the EC-Master must send positions to several servo motors to complete the precise control. Therefore, the user real-time control program must link to NexECM runtime's cycle communication program.

NexECM offers synchronous callback function, making your real-time applications and the ProcessData access in NexECM synchronize with each other, through this mechanism to ensure the synchronization at consistency of delivery data and the communication cycle. The following diagram shows the flow chart of relationship between both of NexECM data processing and user's Callback procedure.



3.5.3. Event Callback Function

When the predefined events happen, NexECM runtime will inform all users to use this function and pass the event code (Event code) synchronized. The user can handle the corresponding event according to the event code, or ignore the processing.

Same as Cyclic callback function, inside the function to access ProcessData can guarantee all data is synchronized.

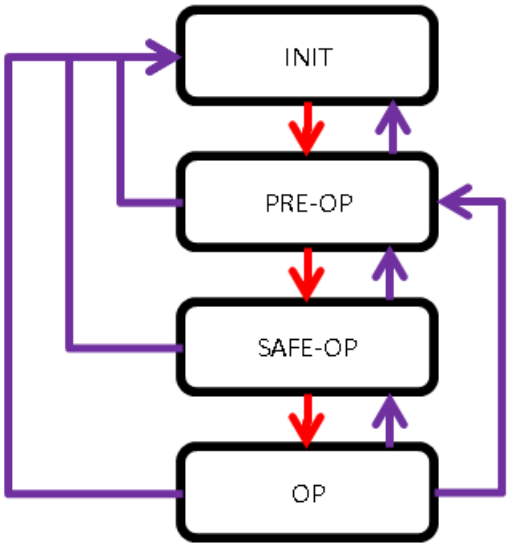
3.5.4. Error Callback Function

When the predefined errors happen, NexECM runtime will inform all users to use this function and pass the error code (Error code) synchronized. The user can handle the corresponding error according to the error code, or ignore the processing.
Same as Cyclic callback function, inside the function to access Processdata can guarantee all data synchronized.

3.6. EtherCAT state Machine

According to EtherCAT standard file (ETG.1000.6), EC-Master must have the state Machine (EtherCAT state Machine, called ESM), to be responsible for processing between the main station and each slave (EC-Slaves) to Operation state from Initial state. Their workflow diagram is shown as below, containing four kinds of state:

INIT state
PREOP state
SAFEOP state
OP state



EtherCAT state Machine diagram

Generally in EtherCAT applications, the EtherCAT state Machine must be changed from "INIT" state to "OP" state. The function for this action is to Initialize EC-Master network configuration and EC-Slave device setting. The setting are in accordance with ENI (EtherCAT Network Information) file that generated from NexECM Studio utility.

3.6.1. ESM state Change

Start EtherCAT communication and use *NEC_RtChangestateToOP*() to change the state to ”OP” state. For special application, you can use *NEC_RtSetMasterstateWait*() to switch to different state . However, this API is not allowed in Callback function. In Callback function, you must use *NEC_RtSetMasterstate*(). *NEC_RtSetMasterstate*() is only to send a request to change the state and doesn’t wait for state change. We

suggest you use *NEC_RtGetMasterstate()* to check if the state has been changed successful.

In below table, list what service is provided at each state in NexECM runtime and EC-Slave.

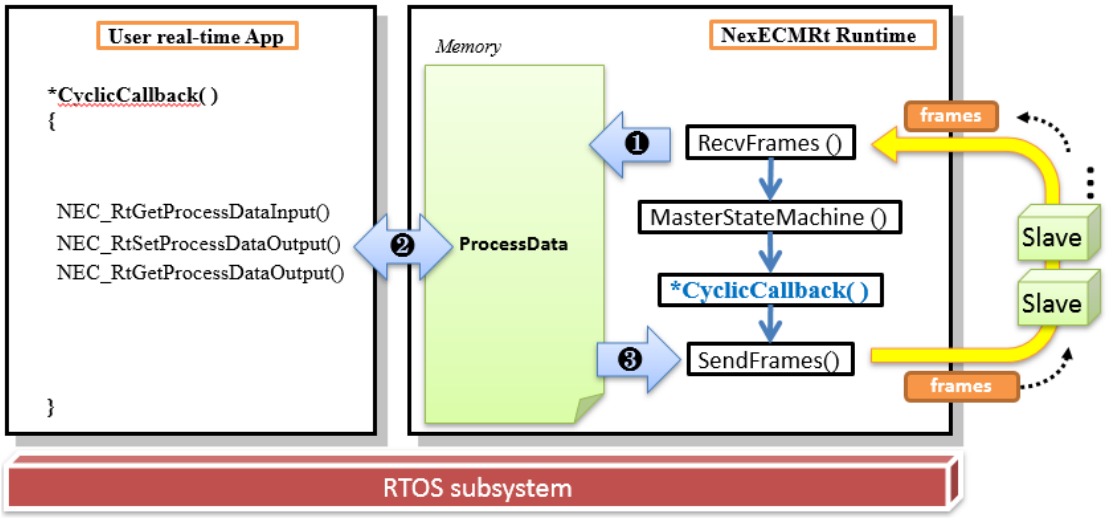
state	EC-Master service	EC-Slave service
INIT	**Cyclic callback function start** No ProcessData communication (To slaves) No Mailbox communication (To slaves)	EC-Master not ready
PREOP	Cyclic callback function start **Mailbox communication start (To slaves)** -SDO communication start No ProcessData communication (To slaves)	Mailbox communication start
SAFEOP	Cyclic callback function start Mailbox communication start (To slaves) -SDO communication start **ProcessData Input communication start** No ProcessData Output communication	Mailbox communication start Update the Input data to ProcessData Input
OP	Cyclic callback function start Mailbox communication start (To slaves) -SDO communication start ProcessData Input communication start **ProcessData Output communication start**	Mailbox communication start Update the Input data to ProcessData Input Get ProcessdataOutpt, Output data and transfer

3.7. Process Data Access

3.7.1. ProcessData Operation mechanism

NexECM will maintain an internal memory for EtherCAT communication ProcessData, The data will update and refresh every communication cycle. Every cycle the NexECM Runtime will execute following steps:

Receive EC-Slaves data, write to ProcessData memory
EthreCAT state machine process
Execute CyclicCallback function
Send ProcessData to EC-Slaves



User application can get the internal memory pointer by usging *NEC_RtGetProcessdataPtr*(), then can directly access this memory. The benefit is high efficiency of accessing, but user need to take care of the range and timing, because the error may cause corruption. Another way is using following API to access ProcessData, API will check the available range before accessing to prevent corruption.

NEC_RtSetProcessdataOutput();
NEC_RtGetProcessdataOutput();
NEC_RtGetProcessdataInput();

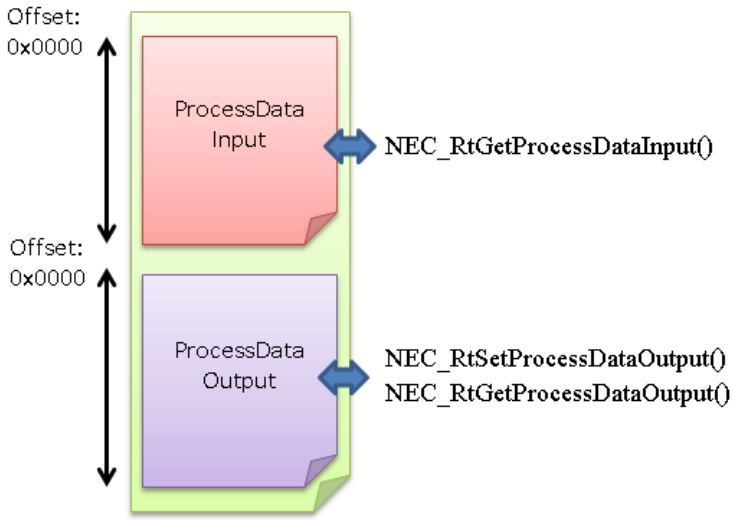
.

About the timing to access ProcessData, it is recommend accessing ProcessData during the callback procedure in order to ensure the consistence of data. If accessing ProcessData outside callback function, data synchronization is only 1 Byte unit.

Actually, the ProcessData area is divided into ProcessDataInput and ProcessDataOutput regionals. Its transfer data direction are listed in the following table:

Processdata	data direction	Read/Write
ProcessdataInput	EC-Slaves transfer to NexECM runtime	Read Only
ProcessdataOutput	NexECM runtime transfer to EC-Slaves	Read/Write

The related API for accessing ProcessData:

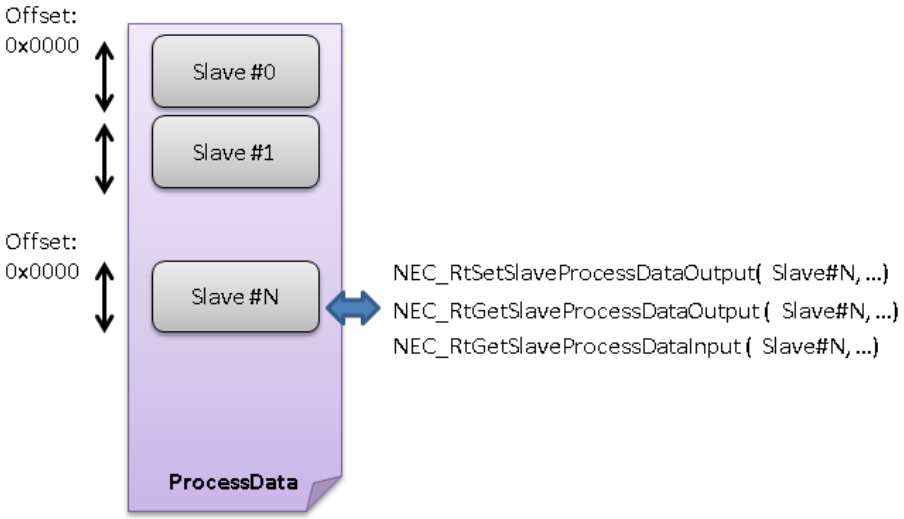


3.7.2. ProcessData Data Content

The content of ProcessData is different for different slave devices. Ex, in category: DIO, its ProcessData is about “digital Input value” or “digital output value” for EC-Slave device. In category: Server Motor, its ProcessData may be “target position” & “actual position value “… ect.

Each slave occupied a block of ProcessData. The base offset of each slave is based on the connection order and the block data size. Show as below diagram.

If need to output data to different slave devices, you can access these moemoy in callback function.



Below are the available APIs for Slave ProcessData memory access:

NEC_RtSetSlaveProcessdataOutput ();
NEC_RtGetSlaveProcessdataOutput ();
NEC_RtGetSlaveProcessdataInput ()

3.8. Mailbox communication

3.8.1. CoE -SDO communication

CANOpen technology in industrial automation applications has been a fairly common and mature technology. For combining with existing control technology, EtherCAT offers CANOpen over EtherCAT (CoE). CANOpen mainly defined in two ways to exchange data:

PDO (Process data object), for real-time data communication
SDO (service data object), for non-realtime data communication

PDO data will map to ProcessData of EtherCAT; About ProcessData please refer CH5.7
SDO communication is achieved by mailbox mechanism of EtherCAT.

Accoring to CANOpen SDO has following types:
SDO download: data download from EC-Master to Slave
SDO upload: data upload from Slave to Master
SDO information: Get object dictionary list

NexECM offer following APIs for SDO communication.

library	Description	T
NEC_RtStartSDODownload	Send SDO download Command data(Master to slave)	B
NEC_RtStartSDOUpload	Send SDO uploadCommand data(Slave to Master)	B
NEC_RtSDODownload	Run SDO download (Structure)	X
NEC_RtSDOUpload	Run SDO upload (Structure)	X
NEC_RtSDODownloadEx	Run SDO download (Native data type)	X
NEC_RtSDOUploadEx	Run SDO upload (Native data type)	X

As to how to use API, please refer to section 3.7.

3.8.2. CoE -Emergency communication

When error occurs in the EC-Slave, emergency message is produced in EC-Slave and delivered to NexECM runtime from EC-Slaves through CoE communication. This message will be stored to internal message queue. Then, NexECM runtime calls the event callback if user registered it. The rules of NexECM runtime calling the event callback as below:

When a new emergency message adds to the message queue, NexECM runtime will call the event callback.
At one cycle time, the event callback will be called only once, even more than one emergency message added to the queue in the same time.
No new emergency message adding to the queue no calling the event callback, even there are many emergency messages in the queue.

If the emergency queue is full, the new message will replace the older message.

NexECM offer following APIs for reading back Emergency message.

library	Description	T
NEC_RtEmgDataCount	Get the amount of emergency message	B
NEC_RtEmgData	Get the emergency message	B

As to how to use API, please refer to section 4.7.

4. NexECM RT-Library

4.1. RT API Overview

This is all NexECM rea-time Libraries API list, APIs are defined at NexECMRt.h. All of APIs listed in NexECMRt.h are only available in RTOS environment.

T(Type) column: This field represent that where the API can be used in:

C: API can only be used in Callback function

X: API can't be used in Callback function

B: No limitation

(**T**: Type ➔ C: Callback only, X: Not for callback, B:Both)

library	Description	T
Initial Related APIs		
NEC_RtGetVersion NEC_RtGetVersionEx NEC_RtRetVer	Get the current version of NexECM	B
NEC_RtCheckLicense	Get the license state	B
NEC_RtInitMaster	Initial target Master	X
NEC_RtCloseMaster	Close target Master	X
NEC_RtSetParameter	Set target Master Parameters	X
NEC_RtGetParameter	Get target EC-Master Parameters	X
NEC_RtRegisterClient	register Callback client function	X
NEC_RtUnRegisterClient	release Callback client register	X
EC-Master control Related APIs		
NEC_RtStartMaster	Start EtherCAT communication	X
NEC_RtStopMaster	Stop EtherCAT cycle communication	X
NEC_RtStopMasterCb	Send EtherCAT communication stop request in Callback client	C
NEC_RtWaitMasterStop	Wait EtherCAT communication stop request & Stop communication	X
NEC_RtGetMasterState	Get EtherCAT current state	B
NEC_RtSetMasterState	Send EtherCAT change state request	B
NEC_RtChangeStateToOP	Blocking call, send EtherCAT state change request & wait until "OP" state	X

NEC_RtSetMasterStateWait	Blocking call, send EtherCAT state change request with assigned state and & until reaching state assigned.	X
NEC_RtGetStateError	Get state Error Code	B
ProcessData Access Related APIs		
NEC_RtGetProcessDataPtr	Get the memory pointer point to ProcessData in EC-Mater	B
NEC_RtSetProcessDataPtrSource	Set ProcessData Memory source	B
NEC_RtSetProcessDataOutput	Write ProcessDataOutput data	B
NEC_RtGetProcessDataOutput	Get ProcessDataOutput data	B
NEC_RtGetProcessDataInput	Get ProcessDataInput data	B
NEC_RtSetSlaveProcessDataOutput	Write EC-Slave ProcessDataOutput data	B
NEC_RtGetSlaveProcessDataOutput	Get EC-Slave ProcessDataOutput data	B
NEC_RtGetSlaveProcessDataInput	Get EC-Slave ProcessDataInput data	B
NEC_RtSetSlaveProcessDataOutputEx	Set EC-Slave ProcessDataOutput data by bit format	B
NEC_RtGetSlaveProcessDataOutputEx	Get EC-Slave ProcessDataOutput data by bit format	B
NEC_RtGetSlaveProcessDataInputEx	Get EC-Slave ProcessDataInput data by bit format	B
Callback APIs (APIs Prototype)		
*NEC_RtCyclicCallback	Cyclic callback function	C
*NEC_RtEventCallback	Event Call back function	C
*NEC_RtErrorCallback	Error Call back function	C
ENI Related APIs		
NEC_RtGetSlaveCount	Get EC-Slaves Numbers	B
NEC_RtGetSlaveInfomation	Get the EC-Slave information	B
NEC_RtGetSlaveState	Get state of EC-Slaves	B
NEC_RtLoadNetworkConfig	Load ENI file to target EC-Master	X
CoE Communication Related APIs		
NEC_RtStartSDODownload	Send SDO download Command data(Master to slave)	B
NEC_RtStartSDOUpload	Send SDO upload Command	B

	data(Slave to Master)	
NEC_RtSDODownload	Run SDO download (Structure)	X
NEC_RtSDOUpload	Run SDO upload (Structure)	X
NEC_RtSDODownloadEx	Run SDO download (Native data type)	X
NEC_RtSDOUploadEx	Run SDO upload (Native data type)	X
NEC_RtStartGetODListCount	Get numbers of index list of each OD type	B
NEC_RtStartGetODList	Get list of indexes of one OD type	B
NEC_RtStartGetObjDesc	Get index's object description	B
NEC_RtStartGetEntryDesc	Get index's Entry description	B
NEC_RtGetEmgDataCount	Get number of emergency data	B
NEC_RtGetEmgData	Get emergency data	B
Slave Hardware Information Access APIs		
NEC_RtGetConfiguredAddress	Read back Slave's Configured Station Address	X
NEC_RtGetAliasAddress	Read back Slave's Configured Station Alias	X
NEC_RtGetSlaveCoeProfileNum	Get the Slave's CoeProfileNum	B
Cyclic timer information and diagnosis		
NEC_RtGetCyclicTick	Get the tick of target EC-Master system time	
NEC_RtGetCyclicTime	Get the target EC-Master system time	
NEC_RtGetLastCyclicLoad	Get the lastest loading of cyclic process	
NEC_RtIsCyclicOverLoad	check whether the over cycle is ever occurred in cyclic process	
NEC_RtResetCyclicOverLoad	Reset the over cycle time state	
NEC_RtGetCyclicOverLoadCount	Get the numbers of over cycle of cyclic process	

API C/C++ data type defined at nex_type.h:

Type	C/C++	Description	Byte	Range
BOOL_T	Int	Boolean	4	0:False, 1:True
U8_T	unsigned char	Character	1	0 ~ 255
U16_T	unsigned short	Character	2	0 ~ 65535

U32_T	unsigned int	Character	4	0 ~ 4294967295
U64_T	unsigned __int64	Character	8	0 ~ 18446744073709551615
I8_T	Char	Integer	1	-128 ~ 127
I16_T	Short	Integer	2	-32768 ~ 32767
I32_T	Int	Integer	4	-2147483648 ~ 2147483647
I64_T	__int64	Integer	8	-9223372036854775808 ~ 9223372036854775807
F32_T	Float	Float	4	IEEE-754, 7 decimal
F64_T	Double	Double Float	8	IEEE-754, 15 decimal
RTN_ERR	Int	Error Code	4	-2147483648 ~ 2147483647

4.2. Initial Related APIs

4.2.1. NEC_RtGetVersion

Return NexECM version

C/C++ Syntax :

RTN_ERR NEC_RtGetVersion(U32_T *Version);

Parameters:

U32_T *Version:

Return the complete information of version of NexECM

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

No Limit.

Reference:

NEC_RtRetVer()

4.2.2. NEC_RtGetVersionEx

Return NexECM version

C/C++ Syntax :

RTN_ERR NEC_RtGetVersionEx(_opt_null_ I32_T *PRetMajor, _opt_null_ I32_T *PRetMinor, _opt_null_ I32_T *PRetStage, _opt_null_ I32_T *PRetBuild);

Parameters:

_opt_null_ I32_T *PRetMajor:

Return the Major cod of version.

_opt_null_ I32_T *PRetMinor:

Return the Minor code of version

_opt_null_ I32_T *PRetStage:

Return the Stage code of version

_opt_null_ I32_T *PRetBuild:

Return the Build code of version

Return Value:

Return the complete information of version of NexECM

Usage:

No Limit.

Reference:

NEC_RtRetVer()

4.2.3. NEC_RtRetVer

Return NexECM current version

C/C++Syntax :

U32_T NEC_RtRetVer();

Parameters:

<No Parameters>

Return Value:

Return the complete information of version of NexECM

Usage:

No Limit.

Reference:

NEC_RtGetVersion()

4.2.4. NEC_RtCheckLicense

Return the state of license for NexECM.

C/C++Syntax :
RTN_ERR NEC_RtCheckLicense();

Parameters:
<No Parameters>

Return Value:
It will return zero if a valid license is existed on the system. Or other values will be return.

Usage:
No Limit.

Reference:

4.2.5. NEC_RtInitMaster

Initialize target EC-Master

C/C++Syntax :

RTN_ERR NEC_RtInitMaster(U16_T MasterId);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

Call this API to do initialization before using other NexECM APIs.

Note! Never call this API in Callback functions.

Reference:

NEC_RtCloseMaster()

4.2.6. NEC_RtCloseMaster

Close target EC-Master

C/C++Syntax :

RTN_ERR NEC_RtCloseMaster(U16_T MasterId);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

Call this API to release internal resource of NexECM runtime before program end.

Note! Never call this API in Callback functions.

Reference:

NEC_RtInitMaste()

4.2.7. NEC_RtGetParameter / NEC_RtSetParameter

These two APIs are used to set and get target EC-Master parameters.

C/C++Syntax :

RTN_ERR NEC_RtSetParameter(U16_T MasterId, U16_T ParaNum, I32_T Paradata);

RTN_ERR NEC_RtGetParameter(U16_T MasterId, U16_T ParaNum, I32_T *Paradata);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0

U16_T ParaNum:

Assign Parameters Code, please refer to Usage:

I32_T Paradata:

Assign Parameters value, please refer to Usage:

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

Before Start EtherCAT communication, set the parameters for EtherCAT communication, parameters list as following. Call *NEC_RtInitMaster*() will not effect this parameters set .

Note! Never call this API in Callback functions.

Parameters Code	Description	Parameters Value
NEC_PARA_S_ECM_CYCLETIMEUS	Communication cycle Unit: micro-second	250 ~ 1000000
NEC_PARA_S_NIC_INDEX	Assign a NIC Port to Master. This parameter could be modify when load ENI information. Please refer to CH5.4.5	0 ~ Max NIC port

	NEC_LoadNetworkConfig()	
NEC_PARA_ECM_RECV_TIMEOUT_US	Define EtherCAT network packet return timeout. Unit: micro-second. Usually you can use the default value.	250 ~ 2000000
NEC_PARA_ECM_LINK_ERR_MODE	Internel disconnected behavior: **LINKERR_AUTO**: When the EC-Slave devices disconnected is detected, target EC-Master will polling disconnection devices until them back to connection and switch those devices back to “OP” state. **LINKERR_MANUAL:** When a device is disconnected, its state will change to ERROR state and remain in this state even reconnected is done. **LINKERR_STOP:** As long as there is a device disconnected, target EC-Master will stop network, and enters ERROR state.	LINKERR_AUTO (0) LINKERR_MANUAL (1) LINKERR_STOP (2)
NEC_PARA_ECM_DC_CYC_TIME_MODE	DC Time Set Mode: 0: Use Master cycle time (Default) 1: Use ENI data	0~1

Reference:

NEC_RtGetStateError(); NEC_RtStartMaster()

4.2.8. NEC_RtRegisterClient

Register callback client functions to target Master.

C/C++Syntax :

RTN_ERR NEC_RtRegisterClient(U16_T MasterId, TClintParam *ClientParam);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

TClintParam *ClientParam: Callback client data structure:

```
typedef struct
{
    U32_T version;
    void  *userDataPtr;
    NEC_RtCyclicCallback cyclicCallback;
    NEC_RtEventCallback  eventCallback;
    NEC_RtErrorCallback  errorCallback;
    U16_T  clientID;
} TClintParam;
```

U32_T version:

Previous Set version, use *NEC_RtRetVer*()

void *userdataPtr:

Pass user-defined data structure pointer. Master will remember and pass the pointer to Callback functions. Set NULL (0) to ignore.

NEC_RtCyclicCallback cyclicCallback:

Pass “cyclic callback function” pointer. Set to “0” mean will not use it. After start communication, this function will be called periodically.

NEC_RtEventCallback event Callback:

Pass “event callback function” pointer. Set to “0” mean will not use it. After start communication, this function will be called when predefined event happens.

NEC_RtErrorCallback errorCallback:

Pass “error allback function” pointer. Set to “0” mean will not use it. After start communication, this function will be called when predefined error happens.

U16_T clientID:

Reserved for NexECM runtime internall use, don’t change these Parameters.

(*) The operation theory about callback function please refer toCH 3.5

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

Call this API before start EtherCAT communication. If you want to do a re-registeration, you must stop the communication first and call *NEC_RtUnregisterClient() to* unregister existing callback client.

Note! Never call this API in Callback functions.

Reference:

NEC_RtStartMaster (); NEC_RtUnRegisterClient();NEC_RtStopMaster();

4.2.9. NEC_RtUnRegisterClient

Unregister callback client functions from target Master

C/C++Syntax :

RTN_ERR NEC_RtUnRegisterClient(U16_T MasterId, TClintParam *ClientParam);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

TclintParam *ClientParam:

Reference *NEC_RtRegisterClient*() API description.

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

Using *NEC_RtRegisterClient*() to register callback functions. When end of the application call *NEC_RtUnRegisterClient() to* unregister existing callback client. Generally please un-register client after *NEC_RtWaitMasterStop*().

Note! Never call this API in Callback functions.

Reference:

NEC_RtRegisterClient(); NEC_RtWaitMasterStop()

4.3. EC-Master Control Related APIs

4.3.1. NEC_RtStartMaster

Target EC-Master starts EtherCAT communication.

C/C++Syntax :

RTN_ERR NEC_RtStartMaster(U16_T MasterId);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

After confige the parameters of Master by using *NEC_RtSetParameter*(), Using this API to start the communication. After communication started, a cyclic process will be active and the callback function which registered by *NEC_RtRegistererClient*() will be executed periodically. The “state” of the Target EC-Master will stay at “INIT” state before you change it.

Two ways for stoping communication:

Call NEC_RtStopMasterCb() in Callback function, or

Use NEC_RtStopMaster()

Note! Never call this API in Callback functions.

Reference:

NEC_RtSetParameter();NEC_RtRegistererClient();NEC_RtStopMaster();

NEC_RtStopMasterCb();**NEC_RtCyclicCallback()*

4.3.2. NEC_RtStopMaster

Target Master stops cycle communication.

C/C++Syntax :

RTN_ERR NEC_RtStopMaster(U16_T MasterId);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API is ued to stop EtherCAT cycle communication. You should call *NEC_RtSetMasterState*() to change assigned EC-Master’s state to ”INIT” state before stop EtherCAT communication.

If you want to stop communication during callback function, please refer to *NEC_RtStopMasterCb()*.

Note! Never call this API in Callback functions.

Reference:

NEC_RtStartMaster();NEC_RtStopMasterCb();NEC_RtSetMasterstate()

4.3.3. NEC_RtStopMasterCb

Send target Master communication stop request, used in callback function.

C/C++Syntax :

RTN_ERR NEC_RtStopMasterCb(U16_T MasterId);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

Users want to stop EtherCAT communication in callback function.

When this API is called, it send a signal to wake up the block API – *NEC_RtWaitMasterStop*() and immediately returns. The *NEC_RtWaitMasterStop*() will be waked up and execute the stopping process. After successful stop, the state of EC-Master assigned will change to “INIT” state.

Reference:

NEC_RtStartMaster();NEC_RtStopMasterCb();NEC_RtWaitMasterStop()

4.3.4. NEC_RtWaitMasterStop

Wait for target Master communication stop request and stop communication, It is a blocked type API.

C/C++Syntax :
RTN_ERR NEC_RtWaitMasterStop(U16_T MasterId);

Parameters:
U16_T MasterId:
Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

Return Value:
Return Error Code.
If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:
There are two purposes for this API:
Let main program goes to sleep, this API will be blocked.
After receiving a stop signal, the Target EC-Master state machine is switched to “INIT” state and this API will weakup and return.

Under most circumstances, the user will put the control procedures into the cyclic callback function. However, an RTOS application is a console type program in which there is a *main*() entry. When the *main*() thread returns, the program is ended and the cyclic callback function also becomes invalid. So, to avoid the application end, user can use this API to suspend the main thread and wait forstop signal trigged by *NEC_RtStopMasterCb*() in callback function.

Note! Never call this API in callback function.

Reference:
NEC_RtStopMasterCb()

4.3.5. NEC_RtSetMasterState / NEC_RtGetMasterState

NEC_RtGetMasterState() : Get target EC-Master current EtherCAT state
NEC_RtSetMasterState() : Send target EC-Master EtherCAT state change request

C/C++Syntax :
RTN_ERR NEC_RtGetMasterState(U16_T MasterId, U16_T *State);
RTN_ERR NEC_RtSetMasterState(U16_T MasterId, U16_T State);

Parameters:
U16_T MasterId:
Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.
U16_T *State:
Return current state. Please refers to following:

ECM_STA_INIT	(1)
ECM_STA_PREOP	(2)
ECM_STA_SAFEOP	(3)
ECM_STA_OPERATION	(4)
ECM_STA_ERROR	(6)

U16_T State:
Set Target state & state range. Please refer to following:

ECM_STA_INIT	(1)
ECM_STA_PREOP	(2)
ECM_STA_SAFEOP	(3)
ECM_STA_OPERATION	(4)

Return Value:
Return Error Code.
If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

After start EtherCAT communication, use *NEC_Rt**Set**MasterState*() to switch EtherCAT state to assigned state. *NEC_Rt**Set**MasterState*() can be used in callback function, it is just send a state change request and will not wait for state change. Using *NEC_Rt**Get**MasterState*() API to check state transition is needed.

Another way is use block API to switch `EtherCAT` state, refers to Reference*NEC_RtSetMasterStateWait*() and *NEC_RtChangestateToOP*()

More about EtherCAT's state machine description please refer to CH3.6 EtherCAT state machine.

Reference:

NEC_RtStartMaster(); *NEC_RtSetMasterStateWait*(), *NEC_RtChangeStateToOP*()

4.3.6. NEC_RtChangeStateToOP

This is a blocked type API. To switch EtherCAT state to “OP “state

C/C++Syntax :

RTN_ERR NEC_RtChangeStateToOP (U16_T MasterId, I32_T TimeoutMs);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

I32_T TimeoutMs:

Timeout, Unit: millisecond.

Set 0 equal to use *NEC_RtSetMasterState*(),

Set -1 mean never Timeout.

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

All EtherCAT slave devices need to be switch to “OP” State when operation. This API is convenient to switch state to “OP” directly. Usage is similar to *NEC_RtSetMasterState*() after start communication to change the EtherCAT state of target Master to target state. The major difference between two functions is that this API can be set a timeout time and this API will be blocked until state switch to “OP” state.

Note! Never call this API in Callback function.

About EtherCAT’s State machine, please refer to CH 3.6 EtherCAT state machine.

Reference:

NEC_RtStartMaster();NEC_RtSetMasterState (); *RtSetMasterStateWait*()

4.3.7. NEC_RtSetMasterStateWait

Blocking change target Master's EtherCAT state

C/C++Syntax :

RTN_ERR NEC_RtSetMasterStateWait(U16_T MasterId, U16_T State, I32_T TimeoutMs);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

I32_T TimeoutMs:

Timeout, Unit: millisecond.

Set 0 equal to use *NEC_RtGetMasterState*(),

Set -1 mean never Timeout.

U16_T State: Set Target state. Please refer to following:

```
#define ECM_STA_INIT        (1)
#define ECM_STA_PREOP       (2)
#define ECM_STA_SAFEOP      (3)
#define ECM_STA_OPERATION   (4)
```

Return Value:

Return Error Code.

If calling of the API is successful, "ECERR_SUCCESS" (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

Use this to switch the target EC-Master state. Usage is similar to NEC_RtSetMasterState() after the start EtherCAT communication to change the EtherCAT state of target EC-Master to target state. Major difference between two funcitons is that this API can be set timeout time. When API success returns mean the state has success changed to target state.

Note! Never call this API in Callback function.

About EtherCAT's state machine, please refer to CH 3.6 EtherCAT state machine.

Reference:

NEC_RtStartMaster();NEC_RtSetMasterState ();*NEC_RtChangeStateToOP* (),

4.3.8. NEC_RtGetStateError

Get the state error code of target Master

C/C++Syntax:

RTN_ERR NEC_RtGetStateError(U16_T MasterId, I32_T *Code);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

I32_T *Code:

Return error code, error Code defined at EcErrors.h header file.

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

When the EtherCAT state of target EC-Master is abnormal (ex: Link broken), state will change to ECM_STA_ERROR state, target Master will record state error code, and you can use this API get the state error code for debugging

Reference:

NEC_RtSetMasterState (); *RtSetMasterStateWait*(),NEC_RtGetMasterState()

4.4. ProcessData Access Related APIs

4.4.1. NEC_RtGetProcessDataPtr

Get EC-Mater assigned internal ProcessData Memory pointer

C/C++Syntax :

RTN_ERR NEC_RtGetProcessDataPtr(U16_T MasterId, U8_T **InputProcessdataPtr, U32_T *InPDSizeInByte, U8_T **OutputProcessdataPtr, U32_T *OutPDSizeInByte);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

U8_T **InputProcessdataPtr:

Return ProcessDataInput(Slaves to master) memory pointer. Set to 0 for by pass.

U32_T *InPDSizeInByte:

Return ProcessDataInput(Slaves to master) memory pointer for length access. Set to 0 for by pass.

U8_T **OutputProcessdataPtr:

Return ProcessDataOnput(master to slave) memory pointer. Set to 0 for by pass

U32_T *OutPDSizeInByte:

Return ProcessDataOnput(master to slave) memory pointer length access. Set to 0 for by pass.

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

To improve ProcessData data exchange efficiency, use this function to get memory pointer. User application can access internal memory but must be very carefully to avoid the system crash. You must do ProcessData access during the Callback functions to keep the data consistence. Related information please refers to CH3.5 callback function and CH3.7 Process data access.

Reference:

NEC_RtSetSlaveProcessDataOutput();NEC_RtGetSlaveProcessDataOutput();
NEC_RtGetSlaveProcessDataInput ();NEC_RtSetProcessDataPtrSource()

4.4.2. NEC_RtSetProcessDataPtrSource

Set ProcessdataMemory source

C/C++Syntax :

```
RTN_ERR NEC_RtSetProcessdataPtrSource( U16_T MasterId
                                     , void *InputProcessdataPtrSource
                                     , U32_T InPDSizeInByte
                                     , void *OutputProcessdataPtrSource
                                     , U32_T OutPDSizeInByte );
```

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

void *InputProcessdataPtrSource:

Set ProcessDataInput memory from outside, if set to 0, mean use internal memory memory

U32_T InPDSizeInByte:

ProcessDataInput memory size, unit is Byte

void *OutputProcessdataPtrSource:

Set ProcessDataOutput memory from outside, if set to 0, mean use internal memory.

U32_T OutPDSizeInByte:

ProcessDataOutput memory size, unit is Byte

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

The default ProcessData memory source is from NexECM runtime’s internal memory. Using *NEC_RtGetProcessDataPtr*() to get the pointer. Another way is using *NEC_RtSetProcessdataPtrSource*() to assign a external memory space as ProcessData memory (Provide the memory space by user). This API must be used before start communication. When communication stoping, ProcessData will back to use internal

memory automatically. So it needs to be assigned again when your re-start the communication.

Note! When stop communication, user must call *NEC_RtSetProcessdataPtrSource*() to assign the memory source back to internal memory before **real-time application unloaded.**

To ensure data consistency, to access the ProcessData during callback functions is recommend. Related information please refers to CH3.5 callback function and CH3.7 Process data access.

Reference:

NEC_RtSetSlaveProcessDataOutput();NEC_RtGetSlaveProcessDataOutput();
NEC_RtGetSlaveProcessDataInput (); NEC_RtGetProcessDataPtr();

4.4.3. NEC_RtGetSlaveProcessDataPtr

Get EC-Mater assigned internal ProcessData Memory pointer by Slave

C/C++Syntax :

RTN_ERR FNTYPE NEC_RtGetSlaveProcessDataPtr(U16_T MasterId, U16_T SlaveAddr, U8_T **InputProcessDataPtr, U32_T *InPDSizeInByte, U8_T **OutputProcessDataPtr, U32_T *OutPDSizeInByte)

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

U16_T SlaveAddr:

Assign target EC-Slave index, increasing number starting from 0.

U8_T **InputProcessDataPtr:

Return ProcessDataInput(Slaves to master) memory pointer. Set to 0 for by pass.

U32_T *InPDSizeInByte:

Return ProcessDataInput(Slaves to master) memory pointer for length access. Set to 0 for by pass.

U8_T **OutputProcessDataPtr:

Return ProcessDataOnput(master to slave) memory pointer. Set to 0 for by pass

U32_T *OutPDSizeInByte:

Return ProcessDataOnput(master to slave) memory pointer length access. Set to 0 for by pass.

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

To improve ProcessData data exchange efficiency, use this function to get memory pointer. User application can access internal memory but must be very carefully to avoid the system crash. You must do ProcessData access during the Callback functions to keep the data consistence. Related information please refers to CH3.5

callback function and CH3.7 Process data access.

Reference:

NEC_RtSetSlaveProcessDataOutput();NEC_RtGetSlaveProcessDataOutput();
NEC_RtGetSlaveProcessDataInput ();NEC_RtSetProcessDataPtrSource()

4.4.4. NEC_RtGetProcessDataInput

Access target EC-Master ProcessData (Process Image) memory

C/C++Syntax :

RTN_ERR NEC_RtGetProcessDataInput(U16_T MasterId, U32_T PDIAddr, U32_T LengthOfdata, U8_T *Retdata);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

U32_T PDOAddr:

ProcessData memory Offset, Unit: Byte

U32_T LengthOfdata:

Data access size, Unit: Byte

U8_T *Retdata:

A memory pointer point to the data for reading from ProcessData

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API can be used only in callback function being used to read or write data to ProcessData.

If not use in callback function it doesn’t guarantee the consistency of reading or writing data.

About Processdata access, please refer to CH3.7 Process Data Access.

Reference:

NEC_RtGetProcessDataPtr();NEC_RtSetSlaveProcessDataOutput();
NEC_RtGetSlaveProcessDataOutput(); NEC_RtGetSlaveProcessDataInput()

4.4.5. NEC_RtSetProcessDataOutput

Access target EC-Master ProcessData (Process Image) memory

C/C++Syntax :

RTN_ERR NEC_RtSetProcessDataOutput(U16_T MasterId, U32_T PDOAddr, U32_T LengthOfdata, U8_T *Setdata);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

U32_T PDOAddr:

ProcessData memory Offset, Unit: Byte

U32_T LengthOfdata:

Data access size, Unit: Byte

U8_T *Setdata:

A memory pointer point to the data for writing to ProcessData

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API can be used only in callback function being used to read or write data to ProcessData.

If not use in callback function it doesn’t guarantee the consistency of reading or writing data.

About Processdata access, please refer to CH3.7 Process Data Access.

Reference:

NEC_RtGetProcessDataPtr();NEC_RtSetSlaveProcessDataOutput(); NEC_RtGetSlaveProcessDataOutput(); NEC_RtGetSlaveProcessDataInput()

4.4.6. NEC_RtGetProcessDataOutput

Access target EC-Master ProcessData (Process Image) memory

C/C++Syntax :

RTN_ERR NEC_RtGetProcessdDtaOutput(U16_T MasterId, U32_T PDOAddr, U32_T LengthOfdata, U8_T *Retdata);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

U32_T PDOAddr:

ProcessData memory Offset, Unit: Byte

U32_T LengthOfdata:

Data access size, Unit: Byte

U8_T *Retdata:

A memory pointer point to the data for reading from ProcessData

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API can be used only in callback function being used to read or write data to ProcessData.

If not use in callback function it doesn’t guarantee the consistency of reading or writing data.

About Processdata access, please refer to CH3.7 Process Data Access.

Reference:

NEC_RtGetProcessDataPtr();NEC_RtSetSlaveProcessDataOutput();
NEC_RtGetSlaveProcessDataOutput(); NEC_RtGetSlaveProcessDataInput()

4.4.7. NEC_RtGetSlaveProcessDataInput

Read or Write EC-Slave ProcessData data of target EC-Master.

C/C++Syntax :

RTN_ERR NEC_RtGetSlaveProcessDataInput(U16_T MasterId, U16_T SlaveAddr, U16_T Offset, U8_T *PRetData, U32_T Size);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

U16_T SlaveAddr:

Assign target EC-Slave index, increasing number starting from 0

U16_T Offset:

EC-Slave ProcessData memory Offset, number starting from 0, Unit: Byte

U8_T * PRetData:

Memory pointer for read or write

U32_T Size:

DataLength, Unit: Byte

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API can be used only callback function being used to read or write data to ProcessData memory which area is occupied by an EC-Slave. If use this API outside callback function, there is no guarantee consistency reading or writing data.

About Processdata access, please refer to CH3.7 Process Data Access.

Reference:

NEC_RtSetProcessDataOutput(); NEC_RtGetProcessDataOutput(); NEC_RtGetProcessDataInput()

4.4.8. NEC_RtSetSlaveProcessDataOutput

Read or Write EC-Slave ProcessData data of target EC-Master.

C/C++Syntax :

RTN_ERR NEC_RtSetSlaveProcessDataOutput(U16_T MasterId, U16_T SlaveAddr, U16_T Offset, U8_T *Data, U32_T Size);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

U16_T SlaveAddr:

Assign target EC-Slave index, increasing number starting from 0

U16_T Offset:

EC-Slave ProcessData memory Offset, number starting from 0, Unit: Byte

U8_T * Data:

Memory pointer for read or write

U32_T Size:

DataLength, Unit: Byte

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API can be used only callback function being used to read or write data to ProcessData memory which area is occupied by an EC-Slave. If use this API outside callback function, there is no guarantee consistency reading or writing data.

About Processdata access, please refer to CH3.7 Process Data Access.

Reference:

NEC_RtSetProcessDataOutput(); NEC_RtGetProcessDataOutput(); NEC_RtGetProcessDataInput()

4.4.9. NEC_RtGetSlaveProcessDataOutput

Read or Write EC-Slave ProcessData data of target EC-Master.

C/C++Syntax :

RTN_ERR NEC_RtGetSlaveProcessDataOutput(U16_T MasterId, U16_T SlaveAddr, U16_T Offset, U8_T *PRetData, U32_T Size);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

U16_T SlaveAddr:

Assign target EC-Slave index, increasing number starting from 0

U16_T Offset:

EC-Slave ProcessData memory Offset, number starting from 0, Unit: Byte

U8_T *PRetData :

Memory pointer for read or write

U32_T Size:

DataLength, Unit: Byte

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API can be used only callback function being used to read or write data to ProcessData memory which area is occupied by an EC-Slave. If use this API outside callback function, there is no guarantee consistency reading or writing data.

About Processdata access, please refer to CH3.7 Process Data Access.

Reference:

NEC_RtSetProcessDataOutput(); NEC_RtGetProcessDataOutput();
NEC_RtGetProcessDataInput()

4.4.10. NEC_RtGetSlaveProcessDataInputEx

Read or Write EC-Slave ProcessData data by bit format.

C/C++Syntax :

RTN_ERR NEC_RtGetSlaveProcessDataInputEx(U16_T MasterId, U16_T SlaveAddr, RwPdoData_T *PRetRwPdoData);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

RwPdoData_T *PRetRwPdoData:

```
typedef struct
{
    U32_T offsetBit;
    U32_T bitSize;
    U8_T  data[MAX_RW_PDO_DATA_LEN];
} RwPdoData_T;
```

U32_T offsetBit:

Assigned offset bits to read/wirte.

U32_T bitSize:

Assigned the size of bits to read/write.

U8_T data[MAX_RW_PDO_DATA_LEN]:

A buffer for data access.

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API can be used only callback function being used to read or write data to ProcessData memory which area is occupied by an EC-Slave. If use this API outside callback function, there is no guarantee consistency reading or writing data.

About Processdata access, please refer to CH3.7 Process Data Access.

Reference:

NEC_RtSetProcessDataOutputEx(); NEC_RtGetProcessDataOutputEx();
NEC_RtGetProcessDataInputEx()

4.4.11. NEC_RtSetSlaveProcessDataOutputEx

Read or Write EC-Slave ProcessData data by bit format.

C/C++Syntax :

RTN_ERR NEC_RtSetSlaveProcessDataOutputEx(U16_T MasterId, U16_T SlaveAddr, const RwPdoData_T *PRwPdoData);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

RwPdoData_T *PRwPdoData:

```
typedef struct
{
    U32_T offsetBit;
    U32_T bitSize;
    U8_T  data[MAX_RW_PDO_DATA_LEN];
} RwPdoData_T;
```

U32_T offsetBit:

Assigned offset bits to read/wirte.

U32_T bitSize:

Assigned the size of bits to read/write.

U8_T data[MAX_RW_PDO_DATA_LEN]:

A buffer for data access.

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API can be used only callback function being used to read or write data to ProcessData memory which area is occupied by an EC-Slave. If use this API outside callback function, there is no guarantee consistency reading or writing data.

About Processdata access, please refer to CH3.7 Process Data Access.

Reference:

NEC_RtSetProcessDataOutputEx(); NEC_RtGetProcessDataOutputEx();
NEC_RtGetProcessDataInputEx()

4.4.12. NEC_RtGetSlaveProcessDataOutputEx

Read or Write EC-Slave ProcessData data by bit format.

C/C++Syntax :

RTN_ERR NEC_RtGetSlaveProcessDataOutputEx(U16_T MasterId, U16_T SlaveAddr, RwPdoData_T *PRetRwPdoData);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

RwPdoData_T *PRetRwPdoData:

```
typedef struct
{
    U32_T offsetBit;
    U32_T bitSize;
    U8_T  data[MAX_RW_PDO_DATA_LEN];
} RwPdoData_T;
```

U32_T offsetBit:

Assigned offset bits to read/wirte.

U32_T bitSize:

Assigned the size of bits to read/write.

U8_T data[MAX_RW_PDO_DATA_LEN]:

A buffer for data access.

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API can be used only callback function being used to read or write data to ProcessData memory which area is occupied by an EC-Slave. If use this API outside callback function, there is no guarantee consistency reading or writing data.

About Processdata access, please refer to CH3.7 Process Data Access.

Reference:

NEC_RtSetProcessDataOutputEx(); NEC_RtGetProcessDataOutputEx();
NEC_RtGetProcessDataInputEx()

4.5. Callback APIs

4.5.1. NEC_RtCyclicCallback

Cyclic callback function

C/C++Syntax :

void (*NEC_RtCyclicCallback)(void *UserdataPtr);

Parameters:

void *UserdataPtr:

A pointer point to user defined data (or data structure)

Return Value:

void no return value

Usage:

Before start communication, please use *NEC_RtRegisterClient*() to register this callback function to NexECM runtime. Then this function will be called cyclically and pass pointer which point to user data.

After stop communication, please use *NEC_RtUnregisterClient*() to release the register of Callback function. Avoid unregistering this callback during communication.

(*) More about Callback library please refer to CH3.5 Callback functions.

Reference:

NEC_RtRegistererClient(); NEC_RtUnregisterClient();

4.5.2. NEC_RtEventCallback

Event Callback function

C/C++Syntax :

void (*NEC_RtEventCallback) (void *UserdataPtr, U32_T EventCode);

Parameters:

void *UserdataPtr:

A pointer point to user defined data (or data structure)

U32_T EventCode:

Pass event code, please refer to usage description.

Return Value:

No return value

Usage:

Use *NEC_RtRegisterClient*() to register callback function to NexECM runtime before start communication. The event callback function will be triggered when following events happened. Users can write event handler in callback function.

(*)Event code defined in RtdataStructDef.h

Event Code	Description
EVENT_ECM_STATE_CHANGE	EC-Master state Change Event
EVENT_ECM_CNECT_FINISH	All EC-Slaves's state change to "Operational" state.

After stop communication, please use *NEC_RtUnregisterClient*() to release the register of Callback function. Avoid unregistering this callback during communication.

(*) More about Callback library please refer to CH 3.5 Callback functions.

Reference:

NEC_RtRegisterClient();NEC_RtUnRegisterClient();

4.5.3. NEC_RtErrorCallback

Error callback function prototype

C/C++Syntax :

void (*NEC_RtErrorCallback) (void *UserdataPtr, I32_T ErrorCode);

Parameters:

void *UserdataPtr:

A pointer point to user defined data (or data structure)

I32_T ErrorCode:

Error code, defined at EcErrors.h header file

Return Value:

No return value

Usage:

Use *NEC_RtRegisterClient*() to register callback function to NexECM runtime before start communication. Error callback function will be triggerd If predefined error event happen. Users can write event handler in callback function.

After stop communication, please use *NEC_RtUnregisterClient*() to release the register of Callback function. Avoid unregistering this callback during communication.

(*) More about Callback library please refer to CH3.5 Callback functions.

Reference:

NEC_RtRegisterClient(); NEC_RtUnregisterClient();

4.6. ENI Related APIs

4.6.1. NEC_RtLoadNetworkConfig

Load ENI to target EC-Master

C/C++Syntax :

RTN_ERR NEC_RtLoadNetworkConfig(U16_T MasterId, const char *ConfigurationFile, U32_T Option);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

const char *ConfigurationFile:

ENI file path C-style string

U32_T Option:

Load option.

Bit number	31 ~ 1	0
Description	Reserve (Set to “0”)	Networking Output Port (NIC) selection 0: Use ENI Setting 1: Use internal Parameters

When Bit 0 Set to 1, refer to *RtSetParameter*() set Networking output port.

Return Value:

Return Error Code.

If calling of the library is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API is used to load EtherCAT network information (ENI) XML file. The ENI File must meet ETG.2100 specification. ENI file can be generated by NexECM Studio (see NexECM Studio User Manual)

Reference:

<No Reference>

4.6.2. NEC_RtGetSlaveCount

Get EC-Slaves numbers from target EC-Master

C/C++Syntax :

RTN_ERR NEC_RtGetSlaveCount(U16_T MasterId, U16_T *Count);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

U16_T *Count:

Return total EC-Slaves numbers

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

When ENI file is downloaded to target EC-Master, use this API to get EC-Slave numbers. This information come from ENI file.

Reference:

<No Reference>

4.6.3. NEC_RtGetSlaveInfomation

Get the EC-Slave information from target EC-Master

C/C++Syntax :

RTN_ERR NEC_RtGetSlaveInfomation(U16_T MasterId, U16_T SlaveAddr, SLAVE_INFO *pSlaveInfo);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

U16_T SlaveAddr:

Assign target EC-Slave index. Increasing number starting from 0

SLAVE_INFO *pSlaveInfo: Return slave information, the data structure as follow:

```
Define in RtDataStructDef.h

typedef struct
{
    U16_T autoIncAddr;
    U16_T configAddr;
    U32_T vendorId;
    U32_T productCode;
    U32_T revisionNo;
    U32_T serialNo;
    U16_T DL_Status;
    U16_T res; //Reserved set 0
} SLAVE_INFO;
```

U16_T autoIncAddr: EtherCAT auto incremental address

U16_T configAddr: EtherCAT config address

U32_T vendorId: EtherCAT slave vendor ID

U32_T productCode: EtherCAT slave product code

U32_T revisionNo: EtherCAT slave revision number

U32_T serialNo: EtherCAT slave serial number

U16_T DL_Status: ESC register value (offset: 0x0110)

Note: When slave's state is transfer from INIT to PREOP, DL_status is updated.

U16_T res: Reserved for internal used.

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

When ENI file is downloaded to target EC-Master, use this API to get EC-Slave’s information. This information comes from ENI file (Except DL_status).

Reference:

<No Reference>

4.6.4. NEC_RtGetSlaveState

Get state of EC-Slaves from target EC-Master

C/C++Syntax :

RTN_ERR NEC_RtGetSlaveState(U16_T MasterId, U16_T SlaveIndex, U8_T *StateArr, U16_T *ArrLen);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

U16_T SlaveIndex: Assign target EC-Slave index. Increasing number starting from 0. Starting index of slave's state.

U8_T *StateArr: Array point, To return an array of slaves' state. State definition as follow:

Define	value	Description
STATE_STOPPED	0	Slave in INIT, PreOP, SafeOP state
STATE_OPERATIONAL	1	Slave in OP state
STATE_ERROR	2	Slave is in error state
STATE_SLAVE_RETRY	3	Slave is in re-connect state

U16_T *ArrLen:

As Input: Size of StareArr

As Output: Return actual array size. If array size small than actual size, it return array size.

Return Value:

Return Error Code.

If calling of the API is successful, "ECERR_SUCCESS" (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API can retrieve more than one slave's state at a time. For example:

```
U16_T SlaveIndex = 0;    // From first slave (0)
U8_T StateArr[6];        // If you have 6 slaves on line.
U16_T ArrLen = 6;
NEC_RtGetSlaveState( MasterId, SlaveIndex, StateArr, &ArrLen );
//It return slave 0 ~ slave5's state
```

Reference:

<No Reference>

4.7. CoE Communication Related APIs

4.7.1. NEC_RtStartSDOUpload / NEC_RtStartSDODownload

Non blocked type APIs for accessing CoE object of EC-Slave from target EC-Master

C/C++Syntax :

RTN_ERR NEC_RtStartSDODownload(U16_T MasterId, U16_T SlaveAddr, TSDO *Hsdo);

RTN_ERR NEC_RtStartSDOUpload(U16_T MasterId, U16_T SlaveAddr, TSDO *Hsdo);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

U16_T SlaveAddr:

Assign target EC-Slave index, increasing number starting from 0

TSDO *Hsdo:

Pointer point to SDO sturture

```
typedef struct
{
    U16_T index;
    U8_T subIndex;
    U8_T ctrlFlag;
    U8_T  *dataPtr;
    U16_T  dataLenByte;
    U16_T  status;
    I32_T  abortCode;
} TSDO;
```

U16_T index:	CANOpen Object index
U8_T subIndex:	CANOpen Object sub-index
U8_T ctrlFlag:	Reserved, Set to 0
U8_T *dataPtr:	Download or Upload data pointer
U16_T dataLenByte:	dataLength
U16_T status:	SDO state
SDO_INIT	(0): In SDO communication
SDO_SUCCESS	(1): SDO communication finish, SDO data transfer completely.
SDO_FAIL	(2): SDO communication fail, Return Error code by EC-Master
SDO_ABORT	(3):SDO communication fail, Return Abort code from EC-Slave

I32_T abortCode:
SDO abort code or EC-Master error code

Return Value:
Return Error Code.
If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:
This API is used to send a SDO Download or SDO Upload request to target EC-Master; The function just send the request and return immediately. This API can be used in callback function.
SDO command usually need several communication cycle times, user need to poll the SDO state “*TSDO .status* equal *SDO_SUCCESS*” if the SDO request is finished.

Reference:
NEC_RtSDODownload(); NEC_RtSDOUpload(); NEC_RtSDODownloadEx();
NEC_RtSDOUploadEx()

4.7.2. NEC_RtSDOUploadEx / NEC_RtSDODownload / NEC_RtSDOUpload/ NEC_RtSDODownloadEx

Blocked type APIs for accessing CoE object of EC-Slave from target EC-Master

C/C++Syntax :

```
RTN_ERR NEC_RtSDODownload( U16_T MasterId, U16_T SlaveAddr, TSDO *Hsdo );
RTN_ERR NEC_RtSDOUpload( U16_T MasterId, U16_T SlaveAddr, TSDO *Hsdo );
RTN_ERR NEC_RtSDODownloadEx( U16_T MasterId, U16_T SlaveAddr
        , U16_T Index, U8_T SubIndex , U8_T CtrlFlag
        , U32_T dataLenByte, U8_T *dataPtr, I32_T *AbortCode );
RTN_ERR NEC_RtSDOUploadEx( U16_T MasterId, U16_T SlaveAddr
        , U16_T Index, U8_T SubIndex, U8_T CtrlFlag
        , U32_T dataLenByte, U8_T *dataPtr, I32_T *AbortCode );
```

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

U16_T SlaveAddr:

Assign target EC-Slave index. Increasing number starting from 0

TSDO *Hsdo:

Pointer point to SDO data sturture. Refer to 6.7.1 *NEC_RtStartSDODownload*() Parameters Description

U16_T index: CANOpen Object index

U8_T subIndex: CANOpen Object subIndex

U8_T ctrlFlag: Reserved. Pease Set to 0

U8_T *dataPtr: data pointer for Downloador or Upload

U16_T dataLenByte: data Length

U16_T status: SDO state

SDO_INIT (0): In SDO communication

SDO_SUCCESS (1): SDO communication finish, SDO data transfer completely.

SDO_FAIL (2): SDO communication fail, Return Error code by EC-Master

SDO_ABORT (3):SDO communication fail, Return Abort code from EC-Slave

I32_T abortCode:

SDO abort code or EC-Master error code

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

The API used to complete an SDO Download or SDO Upload of request, when the API success returns, SDO has been completed. Due to SDO command needs several communication cycles, therefore this API **CANNOT** be used in callback function to avoid procedural deadlock.

Note! Never call this API in Callback function.

Reference:

NEC_RtStartSDODownload();NEC_RtStartSDOUpload()

4.7.3. NEC_RtStartGetODListCount

This function is used to get those amounts of each class of object dictionary of specific EC-Slave from target EC-Master

C/C++Syntax :

RTN_ERR FNTYPE NEC_RtStartGetODListCount(U16_T MasterId, U16_T SlaveAddr, TCoEODListCount *pCoeOdListCount);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

U16_T SlaveAddr:

Assign target EC-Slave index. Increasing number starting from 0

TCoEODListCount * pCoeOdListCount:

Pointer points to TCoEODListCount data sturture.

TCoEODListCount data structure.

U16_T numOfAllObj; // [i] Number of entries in the list with all objects.
U16_T numOfRxPdoObj; // [o] RxPDO mapable objects.
U16_T numOfTxPdoObj; // [o] TxPDO mapable objects.
U16_T numOf BackupObj; // [o] Objects to be stored for device replacement.
U16_T numOfStartObj; // [o] Startup parameter objects.
U16_T status; // [o] SDO state.
SDO_INIT (0): In SDO communication
SDO_SUCCESS (1): SDO communication finish, SDO data transfer completely.
SDO_FAIL (2): SDO communication fail, Return Error code by EC-Master
SDO_ABORT (3):SDO communication fail, Return Abort code from EC-Slave
I32_T abortCode; //[o] Abort code or EC-Master error code

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API is used to send SDO information to get those amounts of each class of object dictionary of specific EC-Slave; the function just sends the request and return immediately. This API can be used in callback function.

SDO command usually need several communication cycle time, user need to poll the SDO state until "TCoEODListCount.*status* equal not to *SDO_INIT*".

Reference:

NEC_RtStartGetODList();NEC_RtStartGetObjDesc();NEC_RtStartGetEntryDesc();
NEC_RtGetEmgDataCount();NEC_RtGetEmgData();

4.7.4. NEC_RtStartGetODList

This function is used to get list of object dictionary indexes of specific class from target EC-Slave.

C/C++Syntax :

RTN_ERR FNTYPE NEC_RtStartGetODList(U16_T MasterId, U16_T SlaveAddr, TCoEODList *pCoeOdList);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

U16_T SlaveAddr:

Assign target EC-Slave index. Increasing number starting from 0

TCoEODList * pCoeOdList:

Pointer points to TCoEODList data sturture.

TCoEODList data structure.

```
U16_T listType;          // [i] assign list type.
U16_T lenOfList;         // [i/o] assign number of plistData array.
U16_T *plistData;        // [i/o] data pointer.
U16_T status;            // [o] SDO state.
SDO_INIT                   (0): In SDO communication
SDO_SUCCESS                (1): SDO communication finish, SDO data transfer
completely.
SDO_FAIL                   (2): SDO communication fail, Return Error code by
EC-Master
SDO_ABORT      (3):SDO communication fail, Return Abort code from EC-Slave
I32_T abortCode;         // [o] Abort code or EC-Master error code.
```

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API is used to send SDO information to get list of object dictionary indexes of specific class from target EC-Slave; the function just sends the request and return immediately. This API can be used in callback function.
SDO command usually need several communication cycle time, user need to poll the SDO state until “TCoEODList.*status* equal not to *SDO_INIT*”.

Reference:
NEC_RtStartGetODListCount();NEC_RtStartGetObjDesc();NEC_RtStartGetEntryDesc();
NEC_RtGetEmgDataCount();NEC_RtGetEmgData();

4.7.5. NEC_RtStartGetObjDesc

This function returns an object description of specific index belong to slave's OD

C/C++Syntax :

RTN_ERR FNTYPE NEC_RtStartGetObjDesc(U16_T MasterId, U16_T SlaveAddr, TCoEObjDesc *pCoeObjDesc);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

U16_T SlaveAddr:

Assign target EC-Slave index. Increasing number starting from 0

TCoEObjDesc * pCoeObjDesc:

Pointer points to TCoEObjDesc data sturture.

TCoEObjDesc data structure

```
U16_T index;                // [i] assign index.
U16_T dataType;             // [o] return object's data type.
U8_T maxNumOfSubIndex;      // [o] return object's maximum of sub index.
U8_T objectCode;            // [o] return objects' RW properties.
U16_T sizeOfName;           // [i/o] assign length of pName array.
U8_T *pName;                // [i/o] data pointer.
U16_T status;               // [o] SDO state.
SDO_INIT                    (0): In SDO communication
SDO_SUCCESS                 (1): SDO communication finish, SDO data transfer
completely.
SDO_FAIL                    (2): SDO communication fail, Return Error code by
EC-Master
SDO_ABORT     (3):SDO communication fail, Return Abort code from EC-Slave
I32_T abortCode;            // [o] Abort code or EC-Master error code
```

Return Value:

Return Error Code.

If calling of the API is successful, "ECERR_SUCCESS" (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API is used to send SDO information to get an object description to EC-Slaves; the function just sends the request and return immediately. This API can be used in callback function.

SDO command usually need several communication cycle time, user need to poll the SDO state until “TCoEObjDesc.*status* equal not to *SDO_INIT*”.

Reference:

NEC_RtStartGetODListCount();NEC_RtStartGetODList();NEC_RtStartGetEntryDesc();
NEC_RtGetEmgDataCount();NEC_RtGetEmgData();

4.7.6. NEC_RtStartGetEntryDesc

This function returns a description of a single object dictionary Entry.

C/C++Syntax :

RTN_ERR FNTYPE NEC_RtStartGetEntryDesc(U16_T MasterId, U16_T SlaveAddr, TCoEEntryDesc *pCoeEntryDesc);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

U16_T SlaveAddr:

Assign target EC-Slave index. Increasing number starting from 0

TCoEEntryDesc * pCoeEntryDesc:

Pointer points to TCoEEntryDesc data sturture.

TCoEEntryDesc data structure

```
U16_T index;              // [i] assign index.
U8_T subIndex;            // [i] assign sub index.
U8_T valueInfo;           // [i] assign values in the response.
U16_T dataType;           // [o]data type of object.
U16_T bitLength;          // [o] bit length of object.
U16_T objectAccess;       // [o] access and mapping attributes.
U16_T sizeOfData;         // [i/o] assign length of pName array.
U8_T *pData;              // [i/o] data pointer.
U16_T status;             // [o] SDO state.
SDO_INIT                    (0): In SDO communication
SDO_SUCCESS                 (1): SDO communication finish, SDO data transfer
completely.
SDO_FAIL                    (2): SDO communication fail, Return Error code by
EC-Master
SDO_ABORT      (3):SDO communication fail, Return Abort code from EC-Slave
I32_T abortCode;            // [o] Abort code or EC-Master error code.
```

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API is used to send SDO information to get an description of sigle object Entry to EC-Slaves; the function just sends the request and return immediately. This API can be used in callback function.

SDO command usually need several communication cycle time, user need to poll the SDO state until “TCoEEntryDesc.*status* equal not to *SDO_INIT*”.

Reference:

NEC_RtStartGetODListCount();NEC_RtStartGetODList();NEC_RtStartGetObjDesc();
NEC_RtGetEmgDataCount();NEC_RtGetEmgData();

4.7.7. NEC_RtGetEmgDataCount

This function returns the number of emergency message in the target EC-Master.

C/C++Syntax :

RTN_ERR FNTYPE NEC_RtGetEmgDataCount(U16_T MasterId, U16_T *pEmgDataCount);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

U16_T *pEmgDataCount:

Pointer points to data.

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This function returns the number of emergency message in the Target EC-Master. This API can be used in callback function.

Reference:

NEC_RtStartGetODListCount();NEC_RtStartGetODList();NEC_RtStartGetObjDesc();NEC_RtStartGetEntryDesc();NEC_RtGetEmgData();

4.7.8. NEC_RtGetEmgData

This function is used to get emergency message from target EC-Master.

C/C++Syntax :

RTN_ERR FNTYPE NEC_RtGetEmgData(U16_T MasterId, TEmgData *pEmgData);

Parameters:

U16_T MasterId:

Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

U16_T * pEmgData:

Pointer points to TEmgData data structure.

TEmgData data structure

```
U16_T lenOfData;                // [i/o] assign length of data pointer.
U16_T res ;                     // reserved.
U16_T *pSlaveAddrDataArr;       // [i/o] assign values in the response.
TCoEEmgMsg *pEmgMsgDataArr;     // [i/o] TCoEEmgMsg data structure pointer.
```

TCoEEmgMsg data structure

```
U16_T errorCode;    // [o] Emergency error code.
U8_T errorRegister; // [o] Emergency error register.
U8_T data[5];       // [o] Emergency data.
```

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This function is used to get the emergency message from target EC-Master. This API can be used in callback function.

Reference:

NEC_RtStartGetODListCount();NEC_RtStartGetODList();NEC_RtStartGetObjDesc();
NEC_RtStartGetEntryDesc();NEC_RtGetEmgDataCount();

4.8. Slave Hardware Information Access APIs

4.8.1. NEC_RtGetConfiguredAddress

Get slave's "Configured Station Address" from target EC-Master

C/C++Syntax :

RTN_ERR FNTYPE NEC_RtGetConfiguredAddress(U16_T MasterId, U16_T SlaveAddr, U16_T *pConfigAddr);

Parameters:

U16_T MasterId: Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

U16_T SlaveAddr: Assign target EC-Slave index. Increasing number starting from 0

U16_T * pConfigAddr: A pointer to return slave's *Configured* Address.

Return Value:

Return Error Code.

If calling of the API is successful, "ECERR_SUCCESS" (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API can get the slave's configured address by the slave's physical location on the network. For example: user can read back the configured addess of first slave by following code.

```
U16_T SlaveAddr = 0;      // The first slave
U16_T ConfigAddr = 0;     // A variable for storing the configured address

NEC_RtGetConfiguredAddress ( MasterId,    SlaveAddr,    &ConfigAddr );
```

Note! Never call this API in Callback function.

Reference:

<No Reference>

4.8.2. NEC_RtGetAliasAddress

Get slave's "Configured Station Alias" from target EC-Master

C/C++Syntax :

RTN_ERR FNTYPE NEC_RtGetAliasAddress(U16_T MasterId, U16_T SlaveAddr, U16_T *pAliasAddr)

Parameters:

U16_T MasterId: Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

U16_T SlaveAddr: Assign target EC-Slave index. Increasing number starting from 0

U16_T *pAliasAddr: A pointer to return slave's *Alias* address.

Return Value:

Return Error Code.

If calling of the API is successful, "ECERR_SUCCESS" (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API can get the slave's *Alias* address by the slave's physical location on the network. If one slave has *Alias* address on the network, you can refer to following code to get the slave's physical location.

```
U16_T I;
U16_T SlaveAddr;
U16_T RetAddr = 0;
U16_T SlaveCnt = 0;        // variable, for storing the total slave's on the network.
U16_T const AliasAddr = 0x0021; // A const value for searching.

NEC_RtGetSlaveCount(MasterId, &SlaveCnt)
for( I = 0; I < SlaveCnt; ++I )
{
        NEC_ RtGetAliasAddress(MasterId, I, &RetAddr);
        if( AliasAddr == RetAddr )
        {
            SlaveAddr = I; //find out!
```

```
                Break;
            }
}
```

Note! Never call this API in Callback function.

Reference:

<No Reference>

4.8.3. NEC_RtGetSlaveCoeProfileNum

Get slave's "CoeProfileNum" from target EC-Master

C/C++Syntax :

RTN_ERR FNTYPE NEC_RtGetSlaveCoeProfileNum (U16_T MasterId, U16_T SlaveAddr, U32_T * pCoeProfileNum)

Parameters:

U16_T MasterId: Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

U16_T SlaveAddr: Assign target EC-Slave index. Increasing number starting from 0

U32_T *pCoeProfileNum: A pointer to return slave's CoeProfileNum.

Return Value:

Return Error Code.

If calling of the API is successful, "ECERR_SUCCESS" (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API can get the slave's *CoeProfileNum* by the slave's physical location on the network. User can refer to following code to find out which slave is drive (CoeProfileNum = 402) on the network.

```
U16_T i;
U16_T SlaveCnt = 0;
U32_T CoeProfileNum

NEC_RtGetSlaveCount(MasterId, &SlaveCnt)
for( i = 0; I < SlaveCnt; ++i )
{
        NEC_RtGetSlaveCoeProfileNum (MasterId, i, &CoeProfileNum);
        if(CoeProfileNum == 402 )
        {
            RtPrintf("Slave:%d is a drive\n", I );
        }
}
```

Reference:

<No Reference>

4.8.4. NEC_RtGetSlaveConfiguredAddressEni

Get slave's "Config Address" from target EC-Master

C/C++Syntax :

RTN_ERR FNTYPE NEC_RtGetSlaveConfiguredAddressEni(U16_T MasterId, U16_T SlaveAddr, U16_T *PRetConfigAddr)

Parameters:

U16_T MasterId: Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

U16_T SlaveAddr: Assign target EC-Slave index. Increasing number starting from 0

U32_T *PRetConfigAddr: A pointer to return slave's Configure Address.

Return Value:

Return Error Code.

If calling of the API is successful, "ECERR_SUCCESS" (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API can get the slave's Configure Address .

Reference:

<No Reference>

4.8.5. NEC_RtGetSlaveTimeDiffWithRefSlave

Get slave's "time different" from target EC-Master

C/C++Syntax :

RTN_ERR FNTYPE NEC_RtGetSlaveTimeDiffWithRefSlave(U16_T MasterId, U16_T SlaveAddr, I32_T *PRetTimeDiffNs); // CANNOT use in callback

Parameters:

U16_T MasterId: Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

U16_T SlaveAddr: Assign target EC-Slave index. Increasing number starting from 0

U32_T * PRetTimeDiffNs: A pointer to return slave's time different.

Return Value:

Return Error Code.

If calling of the API is successful, "ECERR_SUCCESS" (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API can get the slave's time different.(register: 0x092c)

Reference:

<No Reference>

4.8.6. NEC_RtGetSlaveCountEx

Get slave count from target EC-Master

C/C++Syntax :
RTN_ERR FNTYPE NEC_RtGetSlaveCountEx(U16_T MasterId, U16_T *PRetCount);
// CANNOT use in callback

Parameters:
U16_T MasterId: Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.
U32_T * PRetCount: A pointer to return slave count.

Return Value:
Return Error Code.
If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:
This API can get the slave count.

Reference:
<No Reference>

4.8.7. NEC_RtGetSlaveVendorId

Get slave's "VendorId" from target EC-Master

C/C++Syntax :

RTN_ERR FNTYPE NEC_RtGetSlaveVendorId(U16_T MasterId, U16_T SlaveAddr, U32_T *PRetVendorId); // CANNOT use in callback

Parameters:

U16_T MasterId: Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

U16_T SlaveAddr: Assign target EC-Slave index. Increasing number starting from 0

U32_T * PRetVendorId: A pointer to return slave's VendorId.

Return Value:

Return Error Code.

If calling of the API is successful, "ECERR_SUCCESS" (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API can get the slave's VendorId.

Reference:

<No Reference>

4.8.8. NEC_RtGetSlaveProductCode

Get slave's "Product Code" from target EC-Master

C/C++Syntax :

RTN_ERR FNTYPE NEC_RtGetSlaveProductCode(U16_T MasterId, U16_T SlaveAddr, U32_T *PRetProductCode); // CANNOT use in callback

Parameters:

U16_T MasterId: Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

U16_T SlaveAddr: Assign target EC-Slave index. Increasing number starting from 0

U32_T * PRetProductCode: A pointer to return slave's Product Code.

Return Value:

Return Error Code.

If calling of the API is successful, "ECERR_SUCCESS" (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API can get the slave's Product Code.

Reference:

<No Reference>

4.8.9. NEC_RtGetSlaveRevisionNo

Get slave's "Revision Number" from target EC-Master

C/C++Syntax :

RTN_ERR FNTYPE NEC_RtGetSlaveRevisionNo(U16_T MasterId, U16_T SlaveAddr, U32_T *PRetRevisionNo); // CANNOT use in callback

Parameters:

U16_T MasterId: Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

U16_T SlaveAddr: Assign target EC-Slave index. Increasing number starting from 0

U32_T * PRetRevisionNo: A pointer to return slave's Revision Number

Return Value:

Return Error Code.

If calling of the API is successful, "ECERR_SUCCESS" (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API can get the slave's Revision Number.

Reference:

<No Reference>

4.9. Time Monitoring

4.9.1. NEC_RtGetCyclicTick

Get running count of master cyclic loop from target EC-Master

C/C++Syntax :

RTN_ERR FNTYPE NEC_RtGetCyclicTick(U16_T MasterId, NECTimerTick_T *PRetTimerTicks);

Parameters:

U16_T MasterId: Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

```
typedef struct
{
    U64_T ticks; // count from every cyclic
} NECTimerTick_T;
```

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

Get running count of master cyclic loop from target EC-Master

Reference:

<No Reference>

4.9.2. NEC_RtGetCyclicTime

Get running time of master cyclic loop from target EC-Master

C/C++Syntax :

RTN_ERR FNTYPE NEC_RtGetCyclicTime(U16_T MasterId, NECTimerTime_T *PRetTimerTime);

Parameters:

U16_T MasterId: Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

```
typedef struct
{
    U32_T ms;       // millisecond 0 to 999
    U32_T sec;      // seconds of minutes from 0 to 61
    U32_T min;      // minutes of hour from 0 to 59
    U32_T hour;     // hours of day from 0 to 24
    U32_T day;      // every 24 hours a day 0 to n
} NECTimerTime_T;
```

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

Get running time of master cyclic loop from target EC-Master

Reference:

<No Reference>

4.9.3. NEC_RtGetLastCyclicLoad

Get running time of cyclic callback function from target EC-Master

C/C++Syntax :
RTN_ERR FNTYPE NEC_RtGetLastCyclicLoad(U16_T MasterId, U32_T *PRetLoadingUs);

Parameters:
U16_T MasterId: Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.
U16_T SlaveAddr: Assign target EC-Slave index. Increasing number starting from 0
U32_T * PRetRevisionNo: A pointer to return slave's Revision Number

Return Value:
Return Error Code.
If calling of the API is successful, "ECERR_SUCCESS" (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:
Get running time of cyclic callback function from target EC-Master.It means overload when running time of cyclic callback is greater then master cycle time.

Reference:
<No Reference>

4.9.4. NEC_RtIsCyclicOverLoad

Return whether master is overload or not.

C/C++Syntax :

BOOL_T FNTYPE NEC_RtIsCyclicOverLoad(U16_T MasterId);

Parameters:

U16_T MasterId: Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

Return Value:

TRUE: master overload

FALSE: master overload

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

Return whether master is overload or not.

Reference:

<No Reference>

4.9.5. NEC_RtResetCyclicOverLoad

Reset overLoad counter

C/C++Syntax :
RTN_ERR FNTYPE NEC_RtResetCyclicOverLoad(U16_T MasterId);

Parameters:
U16_T MasterId: Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

Return Value:
Return Error Code.
If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:
Reset overLoad counter.

Reference:
<No Reference>

4.9.6. NEC_RtGetCyclicOverLoadCount

Get value of overload counter from target EC-Master

C/C++Syntax :
RTN_ERR FNTYPE NEC_RtGetCyclicOverLoadCount(U16_T MasterId, U32_T *PRetOverLoadCnt);

Parameters:
U16_T MasterId: Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.
U32_T * PRetOverLoadCnt: value of overload counter

Return Value:
Return Error Code.
If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:
Get value of overload counter from target EC-Master

Reference:
<No Reference>

4.9.7. NEC_RtClearProbe

Clear all value about time monitoring.

C/C++Syntax :
RTN_ERR FNTYPE NEC_RtClearProbe(U16_T MasterId);

Parameters:
U16_T MasterId: Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

Return Value:
Return Error Code.
If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:
Clear all value about time monitoring.

Reference:
<No Reference>

4.9.8. NEC_RtGetMasterCycleConsumption

Get running time about Callback function from target EC-Master

C/C++Syntax :

RTN_ERR FNTYPE NEC_RtGetMasterCycleConsumption(U16_T MasterId, U32_T *PRetTotalCnt, U32_T *PRetMaxConsumptionTime_US, U32_T *PRetMinConsumptionTime_US, U32_T *PRetAvgConsumptionTime_NS);

Parameters:

U16_T MasterId: Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

U32_T *PRetTotalCnt: running time of Cyclic Callback

U32_T *PRetMaxConsumptionTime_US: Max running time of Cyclic Callback

U32_T *PRetMinConsumptionTime_US: Min running time of Cyclic Callback

U32_T *PRetAvgConsumptionTime_NS: Avg running time of Cyclic Callback

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

Get running time about Callback function from target EC-Master

Reference:

<No Reference>

4.9.9. NEC_RtGetMasterActualCycleTime

Get running time about master cycle from target EC-Master

C/C++Syntax :

RTN_ERR FNTYPE NEC_RtGetMasterActualCycleTime(U16_T MasterId, U32_T *PRetTotalCnt, U32_T *PRetMaxCycleTime_US, U32_T *PRetMinCycleTime_US, U32_T *PRetAvgCycleTime_NS);

Parameters:

U16_T MasterId: Assign control ID of Master in NexECM runtime. Single EC-Master, set to 0.

U32_T *PRetTotalCnt: running count of master cycle

U32_T *PRetMaxCycleTime_US: Max Cycle time of master

U32_T *PRetMinCycleTime_US: Min Cycle time of master

U32_T *PRetAvgCycleTime_NS: Avg Cycle time of master

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

Get running time about master cycle from target EC-Master

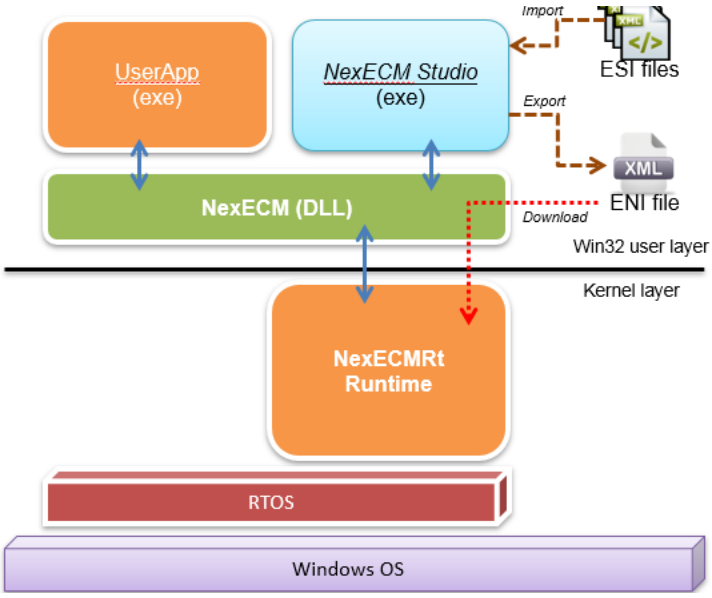
Reference:

<No Reference>

5. NexECM Non-RT Library (Win32 Library)

Win32 user application (as shown in UserApp.exe) can be linked through NexECM Dynamic Library (Win32 Dynamicaly linked library, DLL), to control NexECM runtime running in RTOS.

User that doesn't need real-time characteristic in their application can just use NexECM Win32 library to control the EC-Slave. Such a user doesn't need the RTOS SDK to develop a real-time program running in RTOS.



NexECM non-real-time libraries main functions:

Load / Remove real-time application to / from RTOS
Load EtherCAT Network Information(ENI) to EC-Master
ProcessData access (Through NexECM runtime)
CoE SDO access (Through NexECM runtime)
CiA402 APIs, refer to CiA402 APIs user manual

5.1. Win32 API Summary

Below table lists all NexECM Win32 library APIs. All APIs are defined in NexECM.h.

Library Name	Description	
System Related APIs		
NEC_GetVersion NEC_GetVersionEx	Get version of NexECM	
NEC_StartDriver	Initial NexECM Libraries	
NEC_StopDriver	Close NexECM Libraries	
NEC_LoadRtMaster	Load NexECMRt runtime	
NEC_UnLoadRtMaster	Unload NexECMRt runtime	
NEC_LoadRtApp	Load a RTOS Application	
NexECM Runtime control Related APIs		
NEC_GetRtMasterId	Auto detect in NexECMRt runtime and return the control ID of target EC-Master.	
NEC_ResetEcMaster	Reset target EC-Master	
NEC_LoadNetworkConfig	LoadENI File to target EC-Master	
NEC_StartNetwork	Start EtherCAT communication	
NEC_StartNetworkEx	Start EtherCAT communication (with “Option”)	
NEC_StopNetwork	Stop EtherCAT communication	
NEC_SetParameter	Set target EC-Master parameters	
NEC_GetParameter	Get target EC-Master parameter	
NEC_GetMasterCount	Get the number of EC-Master in the system	
NEC_GetMasterType	Get the type of target EC-Master	
Network status APIs		
NEC_GetSlaveCount	Get counts of EC-Slaves	
NEC_GetNetworkState	Get network state of target EC-Master	
NEC_GetSlaveState	Get network state of EC-Slave	
NEC_GetStateError	Get error code from target EC-Master	
NEC_GetErrorMsg	Get error message from target EC-Master	
Digital I/O control APIs		
NEC_SetDo	Set EC-Slave digital output	

NEC_GetDo	Get status of EC-Slave digital output	
NEC_GetDi	Get status of EC-Slave digital input	
NEC_SetDo_s	Set EC-Slave digital output and wait	
CoE SDO communication APIs		
NEC_SDODownload	Execute a SDO download (Structure)	
NEC_SDOUpload	Execute a SDO upload (Structure)	
NEC_SDODownloadEx	Send a SDO download request (Master to Slave)	
NEC_SDOUploadEx	Send a SDO upload request (Slave to Master)	
NEC_GetODListCount	Get numbers of index list of each OD type	
NEC_GetODList	Get list of indexes of one OD type	
NEC_GetObjDesc	Get index's object description	
NEC_GetEntryDesc	Get index's Entry description	
NEC_EmgDataCount	Get number of emergency data	
NEC_GetEmgData	Get emergency data	
ProcessData Access APIs		
NEC_RWProcessImage	Access target EC-Master ProcessData	
NEC_GetProcessImageSize	Get length of target EC-Master ProcessData	
NEC_RWSlaveProcessImage	Access EC-Slave's ProcessData	
NEC_RWProcessImageEx	Access target EC-Master's ProcessData by bit format with global offset	
NEC_RWSlaveProcessImages	Access target EC-Master's ProcessData by bit format with slave's local offset	
Slave Hardware Information Access APIs		
NEC_GetConfiguredAddress	Read back Slave's Configured Station Address	
NEC_GetAliasAddress	Read back Slave's Configured Station Alias	
NEC_GetSlaveCoeProfileNum	Get Slave's CoeProfileNum	

API C/C++ data type defined at nex_type.h:

Type	C/C++	Description	Byte	Range
BOOL_T	Int	Boolean	4	0:False, 1:True
U8_T	unsigned char	Character	1	0 ~ 255

U16_T	unsigned short	Character	2	0 ~ 65535
U32_T	unsigned int	Character	4	0 ~ 4294967295
U64_T	unsigned __int64	Character	8	0 ~ 18446744073709551615
I8_T	Char	Integer	1	-128 ~ 127
I16_T	Short	Integer	2	-32768 ~ 32767
I32_T	Int	Integer	4	-2147483648 ~ 2147483647
I64_T	__int64	Integer	8	-9223372036854775808 ~ 9223372036854775807
F32_T	Float	Float	4	IEEE-754, 7 decimal
F64_T	Double	Double Float	8	IEEE-754, 15 decimal
RTN_ERR	Int	Error Code	4	-2147483648 ~ 2147483647

5.2. System Related APIs

5.2.1. NEC_GetVersion

Get version of NexECM runtime.

C/C++Syntax :

U32_T NEC_GetVersion();

Parameters:

<No Parameters>

Return Value:

Return the complete information of version of NexECM runtime.

Usage:

This API is used to get the version of NexECM runtime.

Reference:

<No Reference>

5.2.2. NEC_GetVersionEx

Get version of NexECM runtime.

C/C++Syntax :

U32_T NEC_GetVersionEx(_opt_null_ I32_T *PRetMajor
, _opt_null_ I32_T *PRetMinor
, _opt_null_ I32_T *PRetStage
, _opt_null_ I32_T *PRetBuild);

Parameters:

_opt_null_ I32_T *PRetMajor:
Pointer to I32_T for returing Major code of version

_opt_null_ I32_T *PRetMinor:
Pointer to I32_T for returing Minor code of version

_opt_null_ I32_T *PRetStage:
Pointer to I32_T for returning Stage code of version

_opt_null_ I32_T *PRetBuild:
Pointer to I32_T for returning Build code of version

Return Value:

Return the complete information of version of NexECM

Usage:

This API is used to get the version of NexECM runtime.

Reference:

<No Reference>

5.2.3. NEC_StartDriver

Initial NexECM Win32 library

C/C++Syntax :
RTN_ERR NEC_StartDriver();

Parameters:
<No Parameters>

Return Value:
Return Error Code.
If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:
This API is used to initial the library. Using all of APIs in NexECM Non-RT library must be after this API is called. User can load NexECM runtime by using NEC_LoadRtMaster(). About loading NexECM runtime please refer to *NEC_LoadRtApp*() section.

Reference:
NEC_LoadRtApp();NEC_StopDriver();

5.2.4. NEC_StopDriver

Close NexECM Win32 library

C/C++Syntax :
void NEC_StopDriver();

Parameters:
<No Parameters>

Return Value:
Return Error Code.
If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:
Generally this library is used to release the system resource before Win32 Application is closed.

Reference:
NEC_StartDriver ()

5.2.5. NEC_LoadRtMaster / NEC_UnloadRtMaster

Load/unload the NexECMRt runtime into/from RTOS.

C/C++Syntax :
RTN_ERR NEC_LoadRtMaster()

RTN_ERR NEC_UnLoadRtMaster()

Parameters:
<No Parameters>

Return Value:
Return Error Code.
If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Reference:
<No Reference>

5.2.6. NEC_LoadRtApp

Load real-time application into RTOS

C/C++Syntax :

RTN_ERR NEC_LoadRtApp(U16_T RtAppType, const char *PPathOfRtApp);

Parameters:

U16_T RtAppType:

Assign specific type of real-time

Real-time type	Value
RT_APP_TYPE_RTX	1
RT_APP_TYPE_INTIME	2

const char *PPathOfRtApp:

real-time application file path C-style string

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Reference:

<No Reference>

5.3. NexECM Runtime Control Related APIs

5.3.1. NEC_GetRtMasterId

To detect whether NexECMRt runtime is loaded or not and get it's control ID (MasterID)

C/C++Syntax :

RTN_ERR NEC_GetRtMasterId(U16_T * PRetMasterId);

Parameters:

U16_T * PRetMasterId:

Return ID used to control NexECMRt runtime.

Return Value:

Return Error Code.

If calling of the API is successful, "ECERR_SUCCESS" (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

Before using NexECM Win32 library to access (control) the NexECMRt runtime, You must use this API to get the target EC-Master's ID in NexECMRt runtime. If this API returns error: "ECERR_DRIVER_NOT_FOUND". It means that no NexECMRt runtime is running in RTOS.

Reference:

NEC_LoadRtMaster();NEC_LoadRtApp();NEC_UnLoadRtMaster();

5.3.2. NEC_ResetEcMaster

Reset target EC-Master in NexECMRt runtime

C/C++Syntax :

RTN_ERR NEC_ResetEcMaster(U16_T MasterId);

Parameters:

U16_T MasterId:

Assign target EC-Master ID. ID can be get from *NEC_GetRtMasterId*()

Return Value:

Return Error Code.

If calling of the library is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API is used to reset target EC-Master in NexECMRt runtime. Usually be called after *NEC_GetRtMasterId*() and before Load ENI file *NEC_LoadNetworkConfig*(). Use this API will clear all of the internal varialbes including ENI information.

Reference:

NEC_GetRtMasterId(); NEC_LoadNetworkConfig()

5.3.3. NEC_LoadNetworkConfig

Load EtherCAT Network Information (ENI) file to the target EC-Master

C/C++Syntax :

RTN_ERR NEC_LoadNetworkConfig(U16_T MasterId, const char *ConfigurationFile, U32_T Option);

Parameters:

U16_T MasterId:

Assign target EC-Master ID which return from *NEC_GetRtMasterId*()

const char *ConfigurationFile:

ENI file path C-style string

U32_T Option:

Load option.

Bit number	31 ~ 1	0
Description	Reserve (Set to “0”)	Networking Output Port (NIC) selection 0: Use ENI Setting 1: Use internal Parameters

When Bit 0 Set to 1, refer to *RtSetParameter*() set Networking output port.

Return Value:

Return Error Code.

If calling of the library is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API usually is used after *ResetEcMaster*(). This API is used to load EtherCAT network information (ENI) XML file. The ENI File must meet ETG.2100 specification. ENI file can be generated by NexECM Studio. (refer to NexECM Studio User Manual)

Reference:

NEC_ResetEcMaster()

5.3.4. NEC_StartNetwork

Start EtherCAT communication

C/C++Syntax :

RTN_ERR NEC_ StartNetwork (U16_T MasterId, const char *ConfigurationFile, I32_T TimeoutMs);

Parameters:

U16_T MasterId:

Assign target EC-Master ID which return from *NEC_GetRtMasterId*()

const char *ConfigurationFile:

ENI file path C-style string

I32_T TimeoutMs :

Waiting time is out, unit is millisecond

Return Value:

Return Error Code.

If calling of the library is successful, “ECERR_SUCCESS”(0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

After the parameters of target EC-Master are set by *NEC_SetParameter*(). You can use this API to start communication. After the communication is started, a mechanism of cycle communication will be built. You can call *NEC_StopNetwork*() to stop communication.

Reference:

NEC_GetRtMasterId();NEC_SetParameter();NEC_StopNetwork();
NEC_StartNetworkEx()

5.3.5. NEC_StartNetworkEx

Start EtherCAT communication; This API is almost the same with *NEC_StartNetwork*(). The only difference is that it has one more parameter compared with *NEC_StartNetwork*().

C/C++Syntax :

RTN_ERR NEC_StartNetwork(U16_T MasterId, U16_T MasterId, const char *ConfigurationFile, U32_T Option, I32_T TimeoutMs);

Parameters

U16_T MasterId:

Assign target EC-Master ID. ID can be get from *NEC_GetRtMasterId*()

const char *ConfigurationFile:

ENI file path C-style string

U32_T Option:

Options to start。

Bit number	31 ~ 1	0
Description	Reserved (Original setting is 0)	Options for Network Output port: 0:Use ENI setting 1: Use internal parameters

When Bit 0 Set to 1, refer to *RtSetParameter*() set Networking output port.

I32_T TimeoutMs :

Waiting time is out, unit is millisecond

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

After the parameters of target EC-Master are set, you can use this API to start communication. After the communication is started, a mechanism of cycle communication will be built. You can call *NEC_StopNetwork*() to stop communication.

Reference:

NEC_GetRtMasterId();NEC_SetParameter();NEC_StartNetwork();NEC_StopNetwork()

5.3.6. NEC_StopNetwork

To stop EtherCAT communication

C/C++Syntax :
RTN_ERR NEC_StopNetwork(U16_T MasterId, I32_T TimeoutMs);

Parameters
U16_T MasterId:
Assign target EC-Master ID. ID can be get from *NEC_GetRtMasterId*()
I32_T TimeoutMs:
Waiting time is out, unit is millisecond

Return Value:
Return Error Code.
If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:
This API is used to stop EtherCAT cycle communication. During the process to stop the communication, target EC-Master will be switched back to “INIT” state.

Reference:
NEC_GetRtMasterId();NEC_SetParameter();NEC_StartNetwork();
NEC_StartNetworkEx()

5.3.7. NEC_GetParameter / NEC_SetParameter

The two APIs are used to set/get to/from parameters from target EC-Master

C/C++Syntax:

RTN_ERR NEC_SetParameter(U16_T MasterId, U16_T ParaNum, I32_T Paradata);
RTN_ERR NEC_GetParameter(U16_T MasterId, U16_T ParaNum, I32_T *Paradata);

Parameters:

U16_T MasterId:

Assign target EC-Master ID. ID can be get from *NEC_GetRtMasterId*()

U16_T ParaNum:

Define the parameter code name, please refer to usage below

I32_T Paradata:

Define the parameter value，please refer to usage below

I32_T *Paradata:

Return the parameter value，please refer to usage below

Return Value:

Return Error Code.

If calling of the library is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API is used to set the parameters for communication before EtherCAT communication is started. Parameters are listed as below:

Parameter Code	Description	Parameter Value
NEC_PARA_S_ECM_CYCLETIMEUS	Communication cycle,unit is micro-second	250 ~ 1000000
NEC_PARA_S_NIC_INDEX	Set NIC Port number	0 ~ Max NIC port
NEC_PARA_ECM_RECV_TIMEOUT_US	Define EtherCAT Network packet return timeout, unit is micro-second. Generally we can just use default value.	250 ~ 2000000
NEC_PARA_ECM_LINK_ERR_MODE	Behavior when network broken: **LINKERR_AUTO**: When EC-Slave(s) is disconnected, EC-Master detects the location of the	LINKERR_AUTO (0) LINKERR_MANUAL (1) LINKERR_STOP (2)

	disconnection automatically. When the device is connected again, EC-Slaves will be initialized and set to state “OP” **LINKERR_MANUAL:** When a device is disconnected, it will stay in state ERROR. When the device is connected again, it will not be initialized. **LINKERR_STOP:** As long as disconnection happens in any device, EC-Master stop the entire network and get into the state ERROR	
NEC_PARA_ECM_DC_CYC_TIME_MODE	DC Time setting model: 0: According to Master cycle time (Default) 1: According to ENI data	0~1

Reference:

NEC_GetRtMasterId();

5.3.8. NEC_GetMasterType

To get the EC-Master type with specific MasterId

C/C++Syntax :

U16_T NEC_GetMasterType(U16_T MasterId, U16_T *PRetType);

Parameters:

U16_T MasterId:

Assign target EC-Master ID

U16_T *PRetType:

A pointer to returing the type of Master. The define of returing value as below:

Real-time type	Value
MASTER_TYPE_RTX	1
MASTER_TYPE_INtime	2

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

Before using this API, user need to call NEC_StartDriver first. Next use NEC_LoadRtMaster to load NexECMRt runtime into RTOS. Finally, use NEC_GetMasterCount to get the total instances of EC-Master. Finally, use this API to get the type of EC-Master.

Reference:

NEC_LoadRtMaster();NEC_UnLoadRtMaster();NEC_StartDriver();
NEC_GetMasterCount();

5.3.9. NEC_GetMasterCount

To get the total instances of EC-Master in your system.

C/C++Syntax :

RTN_ERR FNTYPE NEC_GetMasterCount(U16_T *PRetMasterCount);

Parameters:

U16_T * PRetMasterCount:

Return total instances of EC-Master in system

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

Before using API, user need to call NEC_StartDriver() first. Next use NEC_LoadRtMaster to load NexECMRt runtime into RTOS. Finally, use this API to get the total instances of EC-Master in system.

Reference:

NEC_LoadRtMaster();NEC_UnLoadRtMaster();NEC_StartDriver();

5.4. Network status APIs

5.4.1. NEC_GetSlaveCount

Get the number of EC-Slaves from target EC-Master

C/C++Syntax:

RTN_ERR NEC_RtGetSlaveCount(U16_T MasterId, U16_T *Count);

Parameters:

U16_T MasterId:

Assign target EC-Master ID. ID can be get from *NEC_GetRtMasterId*()

U16_T *Count:

Return EC-Slave number.

Return Value:

Return Error Code.

If calling of the APIccessful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

Use this API to get the number of EC-Slaves in target EC-Master.

Reference:

NEC_GetRtMasterId()

5.4.2. NEC_GetNetworkstate

To get the network connection state

C/C++ Syntax:

RTN_ERR NEC_GetNetworkstate(U16_T MasterId, U16_T *state);

Parameters:

U16_T MasterId:

Assign target EC-Master ID. ID can be get from *NEC_GetRtMasterId*()

U16_T * state:

Return current network status. Value definition as below:

Value	Meaning
0 (STATE_STOPPED)	Networking is in stopped
1 (STATE_OPERATIONAL)	Networking is in operation state
2 (STATE_ERROR)	Networking / slaves errors, and stopped
3 (STATE_SLAVE_RETRY)	Networking / slaves errors, and try to re-connect

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API is used to get the state of the Network after EtherCAT communication is started.

Reference:

NEC_GetRtMasterId();

5.4.3. NEC_GetSlavestate

To get the state of EC-Slaves

C/C++Syntax:

RTN_ERR NEC_GetSlavestate(U16_T MasterId, U16_T SlaveIndex U8_T *stateArr, U16_T *ArrLen)

Parameters:

U16_T MasterId:

Assign target EC-Master ID. ID can be get from *NEC_GetRtMasterId*()

U16_T SlaveIndex:

Define the EC-Slave position to start

U8_T stateArr:

Return the state array of slave devices

U8_T ArrLen:

Input: Define the array size of “stateArr”

Output: Return the numbers of EC-Slaves which are successfully accessed.

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, if calling of the library fails, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

The API is used to read the state of all EC-Slaves on the Network, after the EtherCAT communication is started.

References:

NEC_GetRtMasterId();

5.4.4. NEC_GetStateError

To get state error string from target EC-Master

C/C++Syntax:

RTN_ERR FNTYPE NEC_GetstateError(U16_T MasterId, I32_T *Code);

Parameters:

U16_T MasterId:

Assign target EC-Master ID. ID can be get from *NEC_GetRtMasterId*()

I32_T *Code:

Return the state error pointer. To know the definition of the error code, please check EcErrors.h header file.

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

The API is used to get target EC-Master state error code when state is in “ERROR” state.

Reference:

NEC_GetRtMasterId();NEC_GetErrorMsg()

5.4.5. NEC_GetErrorMsg

To get state error string from target EC-Master

C/C++ Syntax

RTN_ERR NEC_GetErrorMsg(U16_T MasterId, char *ErrMsg_128_len);

Parameters:

U16_T MasterId:

Assign target EC-Master ID. ID can be get from *NEC_GetRtMasterId*()

char *ErrMsg_128_len:

Return state error string.

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

The API is used to get target EC-Master state error string when state is in “ERROR” state.

References:

NEC_GetRtMasterId();NEC_GetStateError()

5.5. Digital I/O control APIs

5.5.1. NEC_SetDo

To set the EC-Slave Digutal output

C/C++Syntax:

RTN_ERR NEC_SetDo(U16_T MasterId, U16_T SlaveAddr, U16_T Offset, U16_T SizeByte, const U8_T *DoData)

Parameters:

U16_T MasterId:

Assign target EC-Master ID. ID can be get from *NEC_GetRtMasterId*()

U16_T SlaveAddr:

Assign target EC-Slave address. The number will be added in order from 0

U16_T Offset:

Offset of the slave ProcessData memory (output)

U16_T SizeByte:

Define the length DoData, unit byte.

const U8_T *DoData:

Define the digital output data. (DO channel is mapped to each bits)

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

The API is used to set EC-Slave Digital output after the EtherCAT communication is started. Please notice the length(SizeByte) of DoData must meet the DO size of target slave device.

Reference:

NEC_GetRtMasterId();

5.5.2. NEC_GetDo

To read the EC-Slave Digutal output

C/C++Syntax:

RTN_ERR NEC_GetDo(U16_T MasterId, U16_T SlaveAddr, U16_T Offset, U16_T SizeByte, U8_T *Dodata)

Parameters:

U16_T MasterId:

Assign target EC-Master ID. ID can be get from *NEC_GetRtMasterId*()

U16_T SlaveAddr:

Assign target EC-Slave address. The number will be added in order from 0

U16_T Offset:

Offset of the slave ProcessData memory (output)

U16_T SizeByte:

Define the length DoData, unit byte.

U8_T *Dodata:

Return of digital output data (Array); DO channel is mapped to each bits

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

The API is used to get the digital output status of the target EC-Slave after the EtherCAT communication is started. Please notice the length(SizeByte) of DoData must meet the DO size of target slave device.

Reference:

NEC_GetRtMasterId();

5.5.3. NEC_GetDi

To read the EC-Slave digutal input status

C/C++Syntax:

RTN_ERR NEC_GetDi(U16_T MasterId, U16_T SlaveAddr, U16_T Offset, U16_T SizeByte, U8_T *DiData);

Parameters:

U16_T MasterId:

Assign target EC-Master ID. ID can be get from *NEC_GetRtMasterId*()

U16_T SlaveAddr:

Assign target EC-Slave address. The number will be added in order from 0

U16_T Offset:

Offset of the slave ProcessData memory (input)

U16_T SizeByte:

Define the DiData's length, unit byte

U8_T *Didata:

Return of digital input data (Array); DI channel is mapped to each bits

Return Value:

Return Error Code.

If calling of the API is successful, "ECERR_SUCCESS" (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

The API is used to access the input of EC-Slave Digital input after the EtherCAT communication is started. Please notice the length of SizeByte and Didata when you are using it.

Reference:

NEC_GetRtMasterId();

5.5.4. NEC_SetDo_s

To set the EC-Slave Digutal output and wait until EC-master accepted.

C/C++Syntax:

RTN_ERR NEC_SetDo_s(U16_T MasterId, U16_T SlaveAddr, U16_T Offset, U16_T SizeByte, const U8_T *DoData);

Parameters:

U16_T MasterId:

Assign target EC-Master ID. ID can be get from *NEC_GetRtMasterId*()

U16_T SlaveAddr:

Assign target EC-Slave address. The number will be added in order from 0

U16_T Offset:

Offset of the slave ProcessData memory (output)

U16_T SizeByte:

Define the length DoData, unit byte.

const U8_T *DoData:

Define the digital output data. (DO channel is mapped to each bits)

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

The API is used to set EC-Slave Digital output after the EtherCAT communication is started. Please notice the length(SizeByte) of DoData must meet the DO size of target slave device.

Reference:

NEC_GetRtMasterId();

5.6. CoE SDO communication APIs

5.6.1. NEC_SDOUpload / NEC_SDODownloadEx / NEC_SDOUploadEx / NEC_SDODownload

C/C++Syntax:

RTN_ERR NEC_SDODownloadEx(U16_T MasterId, U16_T SlaveAddr
, U16_T Index, U8_T SubIndex , U8_T CtrlFlag
, U32_T dataLenByte, U8_T *dataPtr, I32_T *AbortCode);
RTN_ERR NEC_SDOUploadEx(U16_T MasterId, U16_T SlaveAddr
, U16_T Index, U8_T SubIndex, U8_T CtrlFlag
, U32_T dataLenByte, U8_T *dataPtr, I32_T *AbortCode);
RTN_ERR NEC_SDODownload(U16_T MasterId, U16_T SlaveAddr, TSDO *HSdo);
RTN_ERR NEC_SDOUpload(U16_T MasterId, U16_T SlaveAddr, TSDO *HSdo);

Paremeters:

U16_T MasterId:

Assign target EC-Master ID. ID can be get from *NEC_GetRtMasterId*().

U16_T SlaveAddr:

Assign target EC-Slave address. The number will be added in order from 0

TSDO *HSdo:

A pointer point to SDO data structure:

```
typedef struct
{
    U16_T Index;
    U8_T SubIndex;
    U8_T ctrlFlag;
    U8_T  *dataPtr;
    U16_T  dataLenByte;
    U16_T  status;
    I32_T  abortCode;
} TSDO;
```

U16_T Index:

CANOpen Object Index

U8_T SubIndex:

CANOpen Object SubIndex

U8_T ctrlFlag:

Reserved, please set it to 0

U8_T *dataPtr:

Pointer point to the data to be downloaded or uploaded

U16_T dataLenByte:

Data length, unit byte.

U16_T status:

Reserved

I32_T abortCode:

SDO abort code or EC-Master error code

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

The API is used to complete the request of SDO Download or SDO Upload. If the API returns successfully, this means that the SDO transmission has been completed.

Reference:

NEC_GetRtMasterId();

5.6.2. NEC_GetODListCount

This function is used to get the number of object dictionary with are available for the different CoE list types.

C/C++Syntax:

RTN_ERR FNTYPE NEC_GetODListCount(U16_T MasterId, U16_T SlaveAddr, TCoEODListCount *pCoeOdListCount);

Paremeters:

U16_T MasterId:

Assign target EC-Master ID. ID can be get from *NEC_GetRtMasterId*().

U16_T SlaveAddr:

Assign target EC-Slave address. The number will be added in order from 0

TCoEODListCount * pCoeOdListCount:

A pointer points to TCoEODListCount data structure.

TCoEODListCount data structure:

```
U16_T numOfAllObj;        // [i] Number of entries in the list with all objects.
U16_T numOfRxPdoObj;      // [o] RxPDO mapable objects.
U16_T numOfTxPdoObj;      // [o] TxPDO mapable objects.
U16_T numOf BackupObj;    // [o] Objects to be stored for device replacement.
U16_T numOfStartObj;      // [o] Startup parameter objects.
U16_T status;             // [o] Reserved.
I32_T abortCode;          // [o] Abort code or EC-Master error code
```

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API is used to send SDO information to get the number of object dictionary with are available for the different CoE list types to EC-Slaves. If the API returns successfully, this means that the SDO transmission has been completed.

Reference:

NEC_GetODList();NEC_GetObjDesc();NEC_GetEntryDesc();
NEC_GetEmgDataCount();NEC_GetEmgData();NEC_GetRtMasterId();

5.6.3. NEC_GetODList

This function is used to get list of object dictionary indexes which belong to this CoE list type assigned by user.

C/C++Syntax:

RTN_ERR FNTYPE NEC_GetODList(U16_T MasterId, U16_T SlaveAddr, TCoEODList *pCoeOdList);

Paremeters:

U16_T MasterId:

Assign target EC-Master ID. ID can be get from *NEC_GetRtMasterId*().

U16_T SlaveAddr:

Assign target EC-Slave address. The number will be added in order from 0

TCoEODList * pCoeOdList:

A pointer points to TCoEODList data structure.

TCoEODList data structure:

```
U16_T listType;        // [i] assign list type.
U16_T lenOfList;       // [i/o] assign number of plistData array.
U16_T *plistData;      // [i/o] data pointer.
U16_T status;          // [o] SDO state
U16_T status;          // [o] Reserved.
I32_T abortCode;       // [o] Abort code or EC-Master error code
```

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API is used to send SDO information to get list of object dictionary indexes which belong to this CoE list type assigned to EC-Slaves. If the API returns successfully, this means that the SDO transmission has been completed.

Reference:

NEC_GetODListCount();NEC_GetObjDesc();NEC_GetEntryDesc();

NEC_GetEmgDataCount();NEC_GetEmgData();NEC_GetRtMasterId();

5.6.4. NEC_GetObjDesc

This function returns an object description of index assigned belong to slave's OD.

C/C++Syntax:

RTN_ERR FNTYPE NEC_GetObjDesc(U16_T MasterId, U16_T SlaveAddr, TCoEObjDesc *pCoeObjDesc);

Paremeters:

U16_T MasterId:

Assign target EC-Master ID. ID can be get from *NEC_GetRtMasterId*().

U16_T SlaveAddr:

Assign target EC-Slave address. The number will be added in order from 0

TCoEObjDesc * pCoeObjDesc:

A pointer points to TCoEObjDesc data structure.

TCoEObjDesc data structure:

```
U16_T index;                // [i] assign index.
U16_T dataType;             // [o] return object's data type.
U8_T maxNumOfSubIndex;      // [o] return object's maximum of sub index.
U8_T objectCode;            // [o] return objects' RW properties.
U16_T sizeOfName;           // [i/o] assign length of pName array.
U8_T *pName;                // [i/o] data pointer.
U16_T status;               // [o] Reserved.
I32_T abortCode;            // [o] Abort code or EC-Master error code
```

Return Value:

Return Error Code.

If calling of the API is successful, "ECERR_SUCCESS" (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API is used to send SDO information to get an object description to EC-Slaves. If the API returns successfully, this means that the SDO transmission has been completed.

Reference:

NEC_GetODListCount();NEC_GetODList();NEC_GetEntryDesc();
NEC_GetEmgDataCount();NEC_GetEmgData();NEC_GetRtMasterId();

5.6.5. NEC_GetEntryDesc

This function returns a description of a single object dictionary Entry.

C/C++Syntax:

RTN_ERR FNTYPE NEC_GetEntryDesc(U16_T MasterId, U16_T SlaveAddr, TCoEEntryDesc *pCoeEntryDesc);

Paremeters:

U16_T MasterId:

Assign target EC-Master ID. ID can be get from *NEC_GetRtMasterId*().

U16_T SlaveAddr:

Assign target EC-Slave address. The number will be added in order from 0

TCoEEntryDesc * pCoeEntryDesc:

A pointer points to TCoEEntryDesc data structure.

TCoEEntryDesc data structure:

```
U16_T index;          // [i] assign index.
U8_T subIndex;        // [i] assign sub index.
U8_T valueInfo;       // [i] assign values in the response.
U16_T dataType;       // [o]data type of object.
U16_T bitLength;      // [o] bit length of object.
U16_T objectAccess;   // [o] access and mapping attributes.
U16_T sizeOfData;     // [i/o] assign length of pName array.
U8_T *pData;          // [i/o] data pointer.
U16_T status;         // [o] Reserved.
I32_T abortCode;      // [o] Abort code or EC-Master error code
```

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API is used to send SDO information to get a description of sigle object Entry to EC-Slaves. If the API returns successfully, this means that the SDO transmission has been completed.

Reference:

NEC_GetODListCount();NEC_GetODList();NEC_GetObjDesc();

NEC_GetEmgDataCount();NEC_GetEmgData();NEC_GetRtMasterId();

5.6.6. NEC_GetEmgDataCount

This function returns the number of emergency message in the EC-Master

C/C++Syntax:

RTN_ERR FNTYPE NEC_GetEmgDataCount(U16_T MasterId, U16_T *pEmgDataCount);

Paremeters:

U16_T MasterId:

Assign target EC-Master ID. ID can be get from *NEC_GetRtMasterId*().

U16_T *pEmgDataCount:

Pointer points to data.

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This function returns the number of emergency message in the target EC-Master.

Reference:

NEC_GetODListCount();NEC_GetODList();NEC_GetObjDesc();
NEC_GetEntryDesc();NEC_GetEmgData();NEC_GetRtMasterId();

5.6.7. NEC_GetEmgData

This function is used to get emergency message from EC-Master

C/C++Syntax:

RTN_ERR FNTYPE NEC_GetEmgData(U16_T MasterId, TEmgData *pEmgData);

Paremeters:

U16_T MasterId:

Assign target EC-Master ID. ID can be get from *NEC_GetRtMasterId*().

TEmgData _T * pEmgData:

Pointer points to TEmgData data structure.

TEmgData data structure

```
U16_T lenOfData;                // [i/o] assign length of data pointer.
U16_T res ;                     // reserved.
U16_T *pSlaveAddrDataArr;       // [i/o] assign values in the response.
TCoEEmgMsg *pEmgMsgDataArr;     // [i/o] TCoEEmgMsg data structure pointer.
```

TCoEEmgMsg data structure

```
U16_T errorCode;     // [o] Emergency error code.
U8_T errorRegister;  // [o] Emergency error register.
U8_T data[5];        // [o] Emergency data.
```

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This function is used to get the emergency message from target EC-Master.

Reference:

NEC_GetODListCount();NEC_GetODList();NEC_GetObjDesc();
NEC_GetEntryDesc();NEC_GetEmgDataCount();NEC_GetRtMasterId();

5.7. ProcessData Access APIs

5.7.1. NEC_RWProcessImage

To access the EC-Maste ProcessData (Process image)

C/C++Syntax:

RTN_ERR NEC_ NEC_RWProcessImage(U16_T MasterId, U16_T RW, U16_T Offset, U8_T *data, U16_T Size);

Parameters:

U16_T MasterId:

Assign target EC-Master ID. ID can be get from *NEC_GetRtMasterId*()

U16_T RW:

Define the read/write action

READ_PI (0) : Read ProcessImage (PDI)

WRITE_PI (1) : Write ProcessImage (PDO)

READ_PDO (2) : Read ProcessImage (PDO)

U16_T Offset:

Offset of the EC-Slave Processdata memory, starts from 0, unis is Byte

U8_T *data:

Data pointer

U16_T Size:

Size of the data, unit is Byte

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This function can be used to read and write the Process Image (ProcessData) which inside the target EC-Master. ProcessData principle and operation, please refer to section 3.7 Process Data Access.

Reference:

NEC_GetRtMasterId();NEC_GetProcessImageSize();NEC_RWSlaveProcessImage ()

5.7.2. NEC_GetProcessImageSize

To get length of target EC-Master's image (ProcessData)

C/C++Syntax:

RTN_ERR NEC_GetProcessImageSize(U16_T MasterId, U16_T *SizeOfInputByte, U16_T *SizeOfOutputByte);

Parameters:

U16_T MasterId:

Assign target EC-Master ID. ID can be get from *NEC_GetRtMasterId*()

U16_T *SizeOfInputByte:

Return Processdata Input memory size

U16_T *SizeOfOutputByte:

Return Procedata Output memory size

Return Value:

Return Error Code.

If calling of the API is successful, "ECERR_SUCCESS" (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This function can be used to read the ProcessImage (ProcessData) memory's size of target EC-Master. ProcessData principle and operation, please refer to section 3.7 Process Data Access.

Reference:

NEC_GetRtMasterId();NEC_RWProcessImage ();NEC_RWSlaveProcessImage ()

5.7.3. NEC_RWSlaveProcessImage

To read the memory of EC-Slave

C/C++Syntax

NEC_RWSlaveProcessImage(U16_T MasterId, U16_T SlaveAddr, U16_T RW, U16_T Offset, U8_T *data, U16_T Size);

Parameters:

U16_T MasterId:

Assign target EC-Master ID. ID can be get from *NEC_GetRtMasterId*()

U16_T SlaveAddr:

Assign target EC-Slave code name. The number will be added in order from 0

U16_T RW:

Define read/write action

READ_PI (0) : Read ProcessImage (PDI)

WRITE_PI (1) : Write ProcessImage (PDO)

READ_PDO (2) : Read ProcessImage (PDO)

U16_T Offset:

Offset of the EC-Slave Processdata memory, starts from 0, unis is Byte

U8_T *data:

data pointer

U16_T Size:

Size of the data, unit is Byte

Return Value:

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This function is used to read or write the data to the slave's ProcessData memory which is occupied by the slave device. ProcessData principle and operation, please refer to section 5.7 Process Data Access.

Reference:

NEC_GetRtMasterId();NEC_RWProcessImage ();NEC_GetProcessImageSize ()

5.7.4. NEC_RWProcessImageEx

To access the EC-Maste ProcessData (Process image) with RwPdoData_T structure

C/C++Syntax:

RTN_ERR NEC_RWProcessImageEx(U16_T MasterId, U16_T RW, RwPdoData_T *PRwPdoData);

Parameters:

U16_T MasterId:

Assign target EC-Master ID. ID can be get from *NEC_GetRtMasterId*()

U16_T RW:

Define the read/write action

READ_PI (0) : Read ProcessImage (PDI)

WRITE_PI (1) : Write ProcessImage (PDO)

READ_PDO (2) : Read ProcessImage (PDO)

RwPdoData_T *PRwPdoData:

```
typedef struct
{
    U32_T offsetBit;
    U32_T bitSize;
    U8_T  data[MAX_RW_PDO_DATA_LEN];
} RwPdoData_T;
```

U32_T offsetBit:

Assigned offset bits to wirte.

U32_T bitSize:

Assigned the size of bits to write.

U8_T data[MAX_RW_PDO_DATA_LEN]:

A buffer for data access.

Return Value:

Return Error Code.

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This function can be used to read and write the Process Image (ProcessData) which inside the target EC-Master. ProcessData principle and operation, please refer to

section 3.7 Process Data Access.

Reference:

NEC_GetRtMasterId();NEC_GetProcessImageSize();NEC_RWSlaveProcessImage ()

5.7.5. NEC_RWSlaveProcessImageEx

To read the memory of EC-Slave with RwPdoData_T structure

C/C++Syntax

RTN_ERR NEC_RWSlaveProcessImageEx(U16_T MasterId, U16_T SlaveAddr, U16_T RW, RwPdoData_T *PRwPdoData);

Parameters:

U16_T MasterId:

Assign target EC-Master ID. ID can be get from *NEC_GetRtMasterId*()

U16_T SlaveAddr:

Assign target EC-Slave code name. The number will be added in order from 0

U16_T RW:

Define read/write action

READ_PI (0) : Read ProcessImage (PDI)

WRITE_PI (1) : Write ProcessImage (PDO)

READ_PDO (2) : Read ProcessImage (PDO)

RwPdoData_T *PRwPdoData:

```
typedef struct
{
    U32_T offsetBit;
    U32_T bitSize;
    U8_T  data[MAX_RW_PDO_DATA_LEN];
} RwPdoData_T;
```

U32_T offsetBit:

Assigned offset bits to read.

U32_T bitSize:

Assigned the size of bits to read.

U8_T data[MAX_RW_PDO_DATA_LEN]:

A buffer for data access.

Return Value:

If calling of the API is successful, “ECERR_SUCCESS” (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This function is used to read or write the data to the slave's ProcessData memory

which is occupied by the slave device. ProcessData principle and operation, please refer to section 5.7 Process Data Access.

Reference:

NEC_GetRtMasterId();NEC_RWProcessImage ();NEC_GetProcessImageSize ()

5.8. Slave Hardware Information Access APIs

5.8.1. NEC_GetConfiguredAddress

Get slave's "Configured Station Address "

C/C++Syntax:

RTN_ERR FNTYPE NEC_GetConfiguredAddress(U16_T MasterId, U16_T SlaveAddr, U16_T *pConfigAddr);

Parameters:

U16_T MasterId:

Assign target EC-Master ID. ID can be get from *NEC_GetRtMasterId*()

U16_T SlaveAddr:

Assign target EC-Slave address. The number will be added in order from 0

U16_T *pConfigAddr:

A pointer to return slave's *Configured* Address.

Return Value:

Return Error Code.

If calling of the API is successful, "ECERR_SUCCESS" (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API can get the slave's configured address by the slave's physical location on the network. For example: user can read back the *configured* addess of first slave by following code.

```
U16_T SlaveAddr = 0;      // The first slave
U16_T ConfigAddr = 0;     // A variable for storing the configured address

NEC_GetConfiguredAddress ( MasterId,    SlaveAddr,    &ConfigAddr );
```

Reference:

5.8.2. NEC_GetAliasAddress

Get slave's "Configured Station Alias"

C/C++Syntax:

RTN_ERR FNTYPE NEC_GetAliasAddress (U16_T MasterId, U16_T SlaveAddr, U16_T *pAliasAddr);

Parameters:

U16_T MasterId:

Assign target EC-Master ID. ID can be get from NEC_GetRtMasterId()

U16_T SlaveAddr:

Assign target EC-Slave address. The number will be added in order from 0

U16_T * pAliasAddr:

A pointer to return slave's *Alias* Address.

Return Value:

Return Error Code.

If calling of the API is successful, "ECERR_SUCCESS" (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API can get the slave's *Alias* address by the slave's physical location on the network. If one slave has *Alias* address on the network, you can refer to following code to get the slave's physical location.

```
U16_T i;
U16_T SlaveAddr;
U16_T RetAddr = 0;
U16_T SlaveCnt = 0;
U16_T const AliasAddr = 0x0021; // A const value for searching.

NEC_GetSlaveCount(MasterId, &SlaveCnt)
for( i = 0; i < SlaveCnt; ++i )
{
        NEC_ GetAliasAddress(MasterId, i, &RetAddr);
        if( AliasAddr == RetAddr )
```

```
        {
            SlaveAddr = i; //find out!
            break;
        }
    }
```

Reference:

5.8.3. NEC_GetSlaveCoeProfileNum

Get slave's "CoeProfileNum"

C/C++Syntax :

RTN_ERR FNTYPE NEC_GetSlaveCoeProfileNum(U16_T MasterId, U16_T SlaveAddr, U32_T * pCoeProfileNum)

Parameters:

U16_T MasterId:

Assign target EC-Master ID. ID can be get from NEC_GetRtMasterId()

U16_T SlaveAddr:

Assign target EC-Slave address. The number will be added in order from 0

U32_T *pCoeProfileNum:

A pointer to return slave's CoeProfileNum.

Return Value:

Return Error Code.

If calling of the API is successful, "ECERR_SUCCESS" (0) will be returned. On the contrary, an error code will be returned. To know the definition of the error code, please check EcErrors.h header file.

Usage:

This API can get the slave's *CoeProfileNum* by the slave's physical location on the network. User can refer to following code to find out which slave is drive (CoeProfileNum = 402) on the network.

```
U16_T i;
U16_T SlaveCnt = 0;
U32_T CoeProfileNum

NEC_GetSlaveCount(MasterId, &SlaveCnt)
for( i = 0; I < SlaveCnt; ++i )
{
        NEC_GetSlaveCoeProfileNum (MasterId, i, &CoeProfileNum);
        if(CoeProfileNum == 402 )
        {
            Printf("Slave:%d is a drive\n", I );
```

```
        }
}
```

Reference:

<No Reference>