

NexECMRtx

EtherCAT Master for RTX

User Manual

Manual Rev.: 1.7

Revision Date: May, 2th, 2016

Part No:

Revise note:

Date	Ver	Description
2013/1/10	1.0	First version release
2013/12/10	1.1	Add NEC_RtSetProcessdataPtrSource function
2014/1/23	1.2	Modify chapter 2.3.1 How to config RTX TCP/IP
2014/6/16	1.3	Add Chapter 7. NexECM Library (Win32API)
2014/11/19	1.4	Ch2.1 Add support platform Ch2.3 Modify NIC driver setup Ch3 Enhance NEXCAT Utility description Ch6 & 7 Enhance and revise API descriptions
2015/8/10	1.5	Add Ch6.6.2 NEC_RtGetSlaveInfomation() API Add Ch6.6.3 NEC_RtGetSlaveState() API
2015/12/11	1.6	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
2016/5/2	1.7	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

Contents

NexECMRtx	1
Revise note:	2
Contents	i
1. NexECMRtx Introduction	1
1.1. NexECMRtx Features	2
2. NexECMRtx Installation	4
2.1. Hardware Requirements	4
2.2. Software Requirements	4
2.3. NexECMRtx Installation and Development Flow	4
2.3.1. Install RTX NIC Driver	5
2.3.2. Configure INI file "C:\RtxEcNic.ini"	9
2.3.3. Develop EtherCAT Master RTX Application	13
3. EtherCAT Utilities	18
3.1. NexCAT Introduction	18
3.1.1. Software Requirements	18
3.1.2. NexCAT Operation Flow	19
3.1.3. ESI & ENI	20
3.1.4. NexCAT Main Page	22
3.1.5. Set Slave Parameters	24
3.1.6. Set Master Parameters	29
3.1.7. ESI List (ESI File management)	31
3.1.8. DIO User interface	32
3.1.9. CoE-SDO Operation page	34
3.1.10. Process Image Parameters Operation page	35
3.1.11. Network Quality Monitor	36

3.2.	NexECMRtxStartup	38
4	EtherCAT Introduction.....	39
4.1	EtherCAT communication Protocol	40
4.2	Network Typology.....	41
4.3	High Speed Performance	42
4.4	Network Synchronization	43
5	Principles of Programming	45
5.1	Basic Programming Framework.....	45
5.2	ENI Load.....	46
5.3	Initialization of NexECMRtx	47
5.4	Start Master communication.....	48
5.5	Callback Functions	49
5.5.1	Register Callback function	49
5.5.2	Cyclic Callback Function	49
5.5.3	Event Callback Function	50
5.5.4	Error Callback Function	51
5.6	EtherCAT state Machine	52
5.6.1	ESM state Change	52
5.7	Process Data Access.....	54
5.7.1	ProcessData Operation mechanism.....	54
5.7.2	ProcessData Data Content	55
5.8	Mailbox communication.....	57
5.8.1	CoE -SDO communication	57
5.8.2	CoE -Emergency communication	58
6	NexECMRtx Library.....	59
6.1	RTX API Overview	59
6.2	Initial Related APIs.....	62

6.2.1	NEC_RtGetVersion	62
6.2.2	NEC_RtRetVer	63
6.2.3	NEC_RtInitMaster	64
6.2.4	NEC_RtCloseMaster	65
6.2.5	NEC_RtSetParameter	66
6.2.6	NEC_RtGetParameter	66
6.2.7	NEC_RtregistererClient	68
6.2.8	NEC_RtUnregistererClient	70
6.2.9	NEC_RtGetProcessdataPtr	71
6.2.10	NEC_RtSetProcessdataPtrSource	72
6.3	EC-Master Control Related APIs	74
6.3.1	NEC_RtStartMaster	74
6.3.2	NEC_RtStopMaster	75
6.3.3	NEC_RtStopMasterCb	76
6.3.4	NEC_RtWaitMasterStop	77
6.3.5	NEC_RtGetMasterstate	78
6.3.6	NEC_RtSetMasterstate	78
6.3.7	NEC_RtChangestateToOP	80
6.3.8	NEC_RtSetMasterstateWait	81
6.3.9	NEC_RtGetstateError	83
6.4	ProcessData Access Related APIs	84
6.4.1	NEC_RtSetProcessdataOutput	84
6.4.2	NEC_RtGetProcessdataOutput	84
6.4.3	NEC_RtGetProcessdataInput	84
6.4.4	NEC_RtSetSlaveProcessdataOutput	86
6.4.5	NEC_RtGetSlaveProcessdataOutput	86
6.4.6	NEC_RtGetSlaveProcessdataInput	86

6.5	Callback APIs	88
6.5.1	*NEC_RtCyclicCallback	88
6.5.2	*NEC_RtEventCallback	89
6.5.3	*NEC_RtErrorCallback	90
6.6	ENI Related APIs	91
6.6.1	NEC_RtGetSlaveCount	91
6.6.2	NEC_RtGetSlaveInformation	92
6.6.3	NEC_RtGetSlaveState	94
6.7	CoE Communication Related APIs	96
6.7.1	NEC_RtStartSDODownload	96
6.7.2	NEC_RtStartSDOUpload	96
6.7.3	NEC_RtSDODownload	98
6.7.4	NEC_RtSDOUpload	98
6.7.5	NEC_RtSDODownloadEx	98
6.7.6	NEC_RtSDOUploadEx	98
6.7.7	NEC_RtStartGetODListCount	100
6.7.8	NEC_RtStartGetODList	102
6.7.9	NEC_RtStartGetObjDesc	104
6.7.10	NEC_RtStartGetEntryDesc	106
6.7.11	NEC_RtGetEmgDataCount	108
6.7.12	NEC_RtGetEmgData	109
6.8	Slave Hardware Information Access APIs	110
6.8.1	NEC_RtGetConfiguredAddress	110
6.8.2	NEC_RtGetAliasAddress	111
6.8.3	NEC_RtGetSlaveCoeProfileNum	113
7	NexECM Library (Win32 APIs)	115
7.1	Win32 API Summary	116

7.2	RTX System Control APIs.....	119
7.2.1	NEC_LoadRtxApp	119
7.2.2	NEC_KillRtxApp	120
7.3	NexECM initial related APIs	122
7.3.1	NEC_GetVersion.....	122
7.3.2	NEC_StartDriver	123
7.3.3	NEC_StopDriver	124
7.4	NexECMRtx Runtime Control Related APIs	125
7.4.1	NEC_GetRtMasterId.....	125
7.4.2	NEC_GetRtMasterVersion.....	126
7.4.3	NEC_GetRtMasterPorcessId	127
7.4.4	NEC_ResetEcMaster.....	128
7.4.5	NEC_LoadNetworkConfig.....	129
7.4.6	NEC_StartNetwork.....	130
7.4.7	NEC_StartNetworkEx	131
7.4.8	NEC_StopNetwork	133
7.4.9	NEC_SetParameter	134
7.4.10	NEC_GetParameter.....	134
7.5	Network status APIs.....	136
7.5.1	NEC_GetSlaveCount.....	136
7.5.2	NEC_GetNetworkstate.....	137
7.5.3	NEC_GetSlavestate	138
7.5.4	NEC_GetstateError.....	139
7.5.5	NEC_GetErrorMsg.....	140
7.6	Digital I/O control APIs	141
7.6.1	NEC_SetDo	141
7.6.2	NEC_GetDo	142

7.6.3	NEC_GetDi.....	143
7.7	CoE SDO communication APIs	144
7.7.1	NEC_SDODownloadEx.....	144
7.7.2	NEC_SDOUploadEx	144
7.7.3	NEC_SDODownload	144
7.7.4	NEC_SDOUpload	144
7.7.5	NEC_GetODListCount.....	146
7.7.6	NEC_GetODList	148
7.7.7	NEC_GetObjDesc.....	150
7.7.8	NEC_GetEntryDesc.....	152
7.7.9	NEC_GetEmgDataCount.....	154
7.7.10	NEC_GetEmgData	155
7.8	ProcessData Access APIs.....	156
7.8.1	NEC_RWProcessImage.....	156
7.8.2	NEC_GetProcessImageSize	157
7.8.3	NEC_RWSlaveProcessImage	158
7.9	Slave Hardware Information Access APIs	159
7.9.1	NEC_GetConfiguredAddress	159
7.9.2	NEC_GetAliasAddress	160
7.9.3	NEC_GetSlaveCoeProfileNum	162

1. NexECMRtx Introduction

NexECMRtx is an EtherCAT Master Communication Protocol solution, it is based on IntervalZero RTX (RTX is a real-time extension on Microsoft Windows) to offer real-time communication between EtherCAT Master and EtherCAT slave devices. NexECMRtx offers rich high level C / C++ API for rapid application development.

NexECMRtx also provide a configuration utility: NexCAT, a graphic user interface tool for customer to edit parameters for EtherCAT communication between master and slave devices; its functions are as follow:

1. EtherCAT Slave devices scanning
2. Import ESI file, export ENI file
3. Configure EtherCAT slave devices
4. Monitoring the EtherCAT communication quality
5. Test functions for EtherCAT slave devices

Other detail functions, please refer to Chapter 3

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

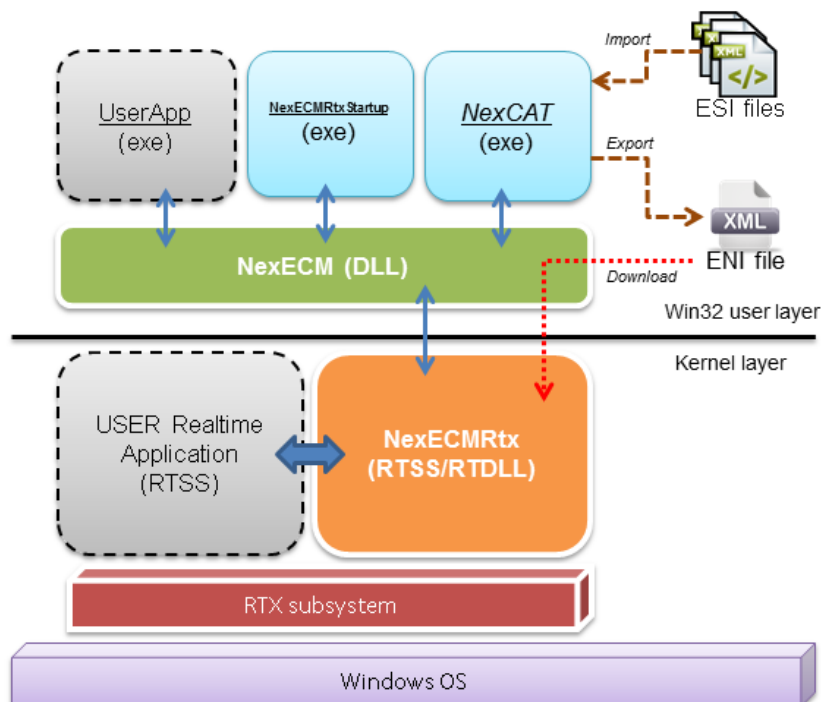


Figure 1.1: NEXCOM EtherCAT master solution system structure

The software module details as below table:

Module	Description	Reference
Operation at RTX environment		
NexECMRtx.rtss	EtherCAT Master runtime stack	CH 5, CH 6
Operation at Win32 environment		
NexCAT.exe	EtherCAT Master configuration utility	CH 3.1
NexECMRtxStartup.exe	EtherCAT Master startup utility	CH 3.2
NexECM.dll	EtherCAT Master Win32 API libraries	CH7

1.1. NexECMRtx 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	Support functions to access EEPROM via ESC	V

EEPROM	registerer	
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. NexECMRtx Installation

2.1. Hardware Requirements

NexECMRtx is tested and optimized for NEXCOM Industrial Computing Platforms.
You could check the document: “NexECMRtx hardware support list.pdf” in installation folder

2.2. Software Requirements

Before beginning the installation, check the following:

1. Operation System: Microsoft Windows XP SP3 or Window 7 32bit
2. IntervalZero RTX 2012
3. Microsoft Visual Studio 2005 ~2012
4. NEXCOM NexECMRtx installation file
5. Microsoft .Net framework 4.0 (For Configure Utility: NexCAT.exe)
6. Visual Basic Power Packs 3.0 (For Configure Utility: NexCAT.exe)

For RTX 2012, you can download 90 days trial version before purchasing^(*):

<http://www.intervalzero.com/rtx-downloads/>

(*)Buy official version or trial version serial number; please contact - NEXCOM International Co., Ltd.

Tel : +886-2-8226-7786 # ICS sales

- Microsoft .Net framework 4.0 Official free download:

<http://www.microsoft.com/zh-tw/download/details.aspx?id=17718>

- Visual Basic Power Packs 3.0 Official free download:

<http://download.microsoft.com/download/1/2/A/12AA9B28-4F67-42C3-9319-684E8AD6F0AE/VisualBasicPowerPacks3Setup.exe>


2.3. NexECMRtx Installation and Development Flow

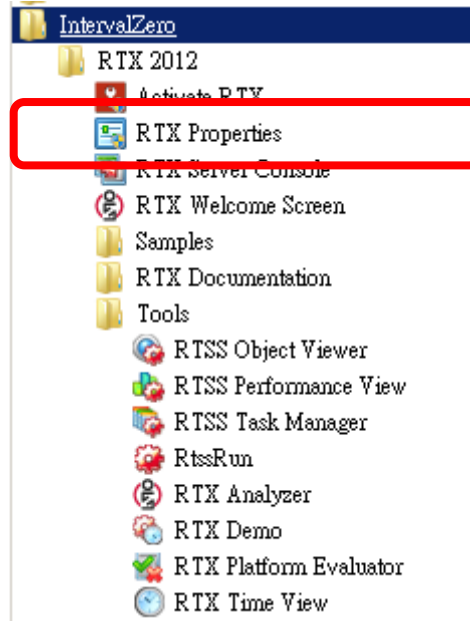
After you get IPC hardware platform, install Windows operation system and the RTX subsystem, please follow these steps to complete the software installation:

1. Install RTX network interface card drivers
2. Configure INI file “C:\RtxEcNic.ini”

3. Use NexCAT.exe to setup EtherCAT network (please refer to Chapter 3)
4. Develop EtherCAT Master RTX application

2.3.1. Install RTX NIC Driver

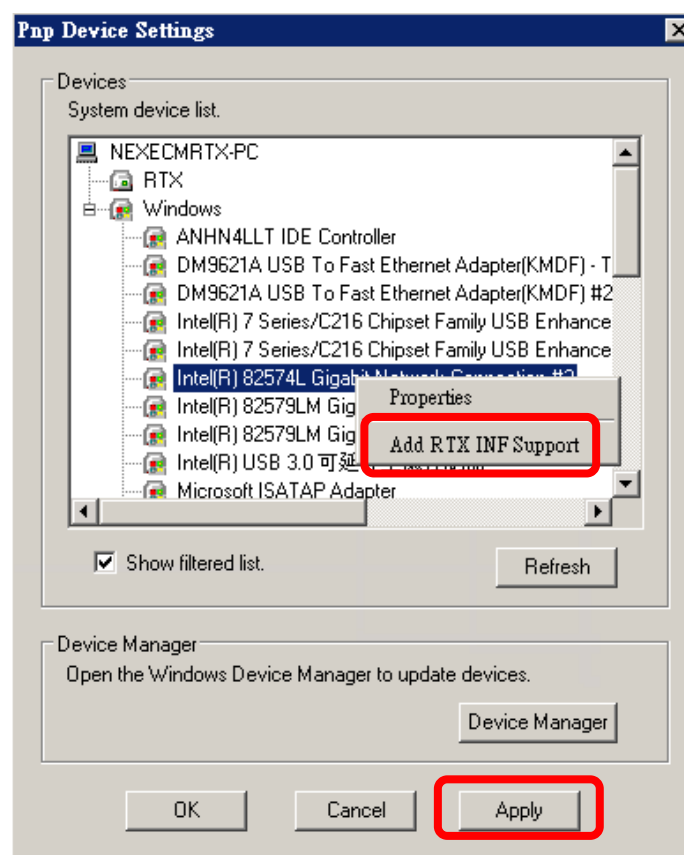
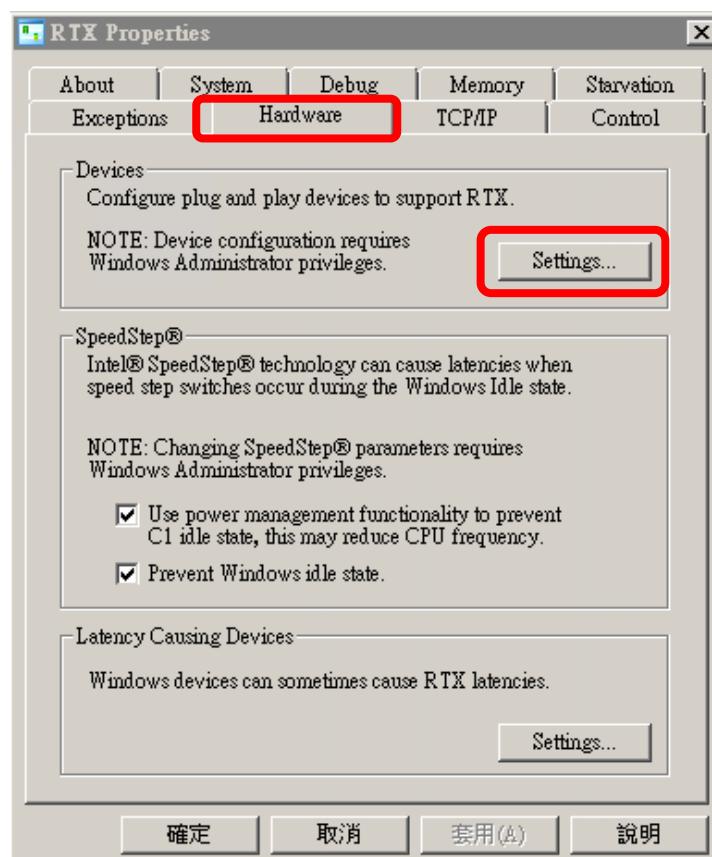
Open the Start menu on (as shown below) or  desktop RTX Utility: "RTX properties" as shown below:



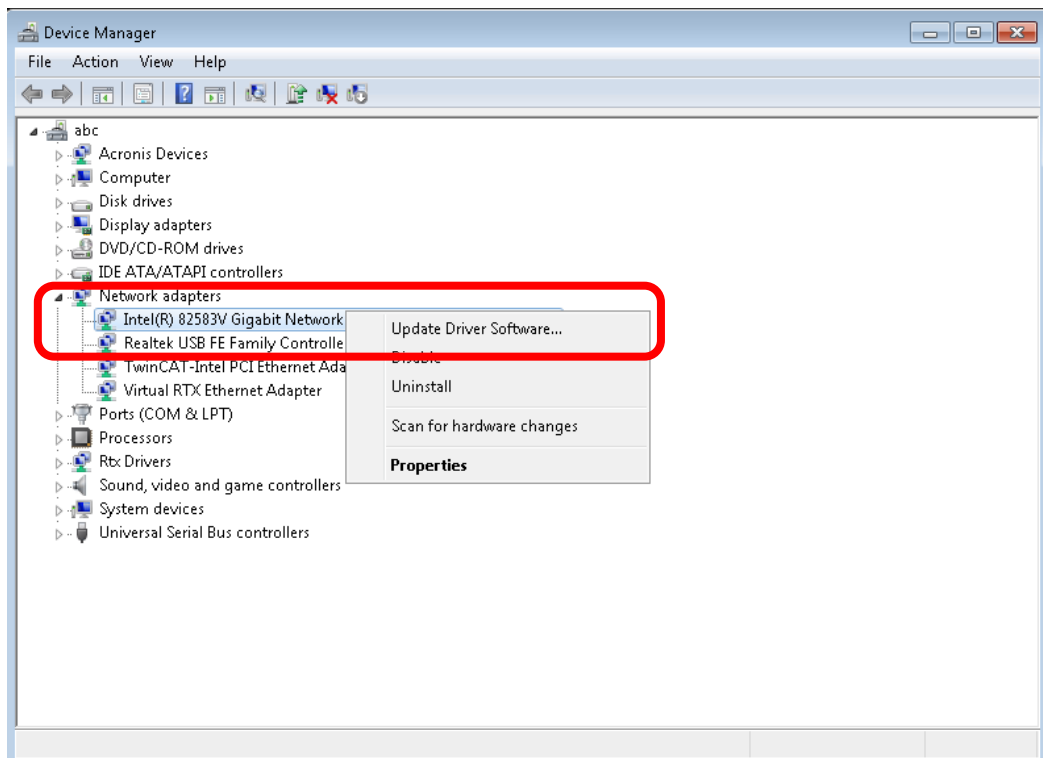
RTX 2012 Tool Tree

In "Hardware" tab (as shown below)

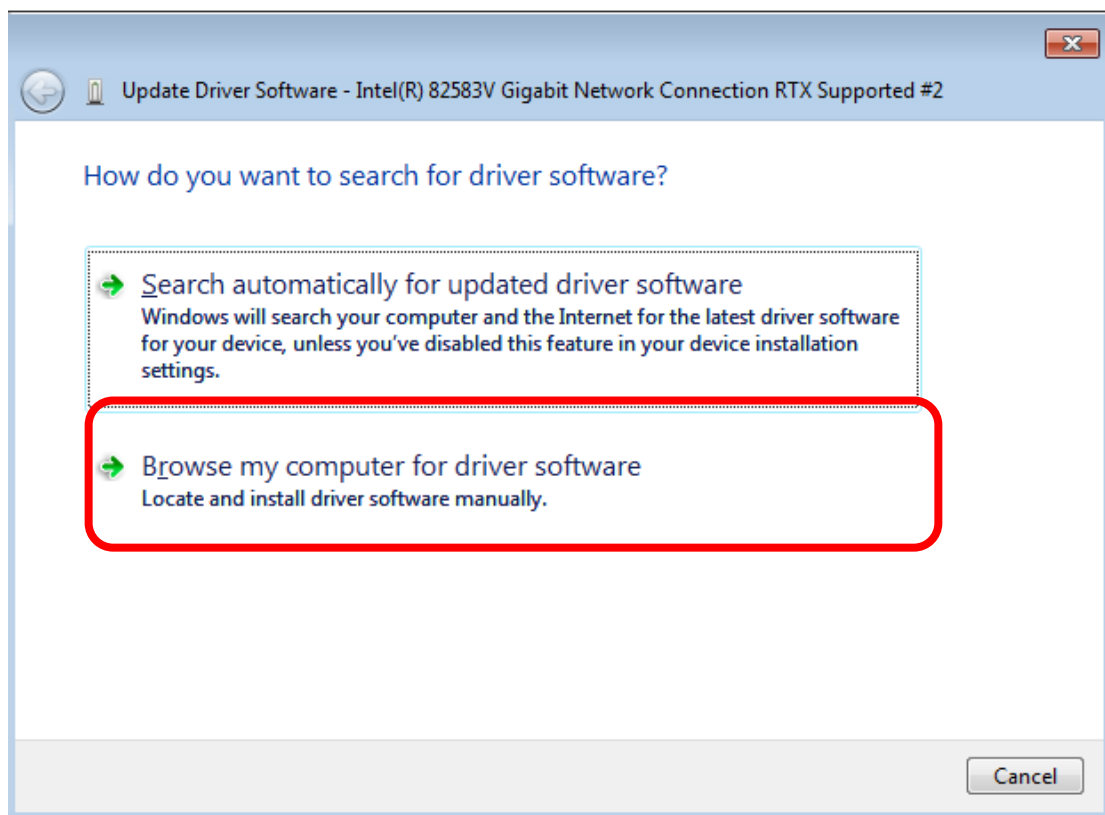
1. Click Device/Setting button and popup "Pnp Device Settings" page
2. Right click on NIC device and select "Add RTX INF Support" then click "Apply" button



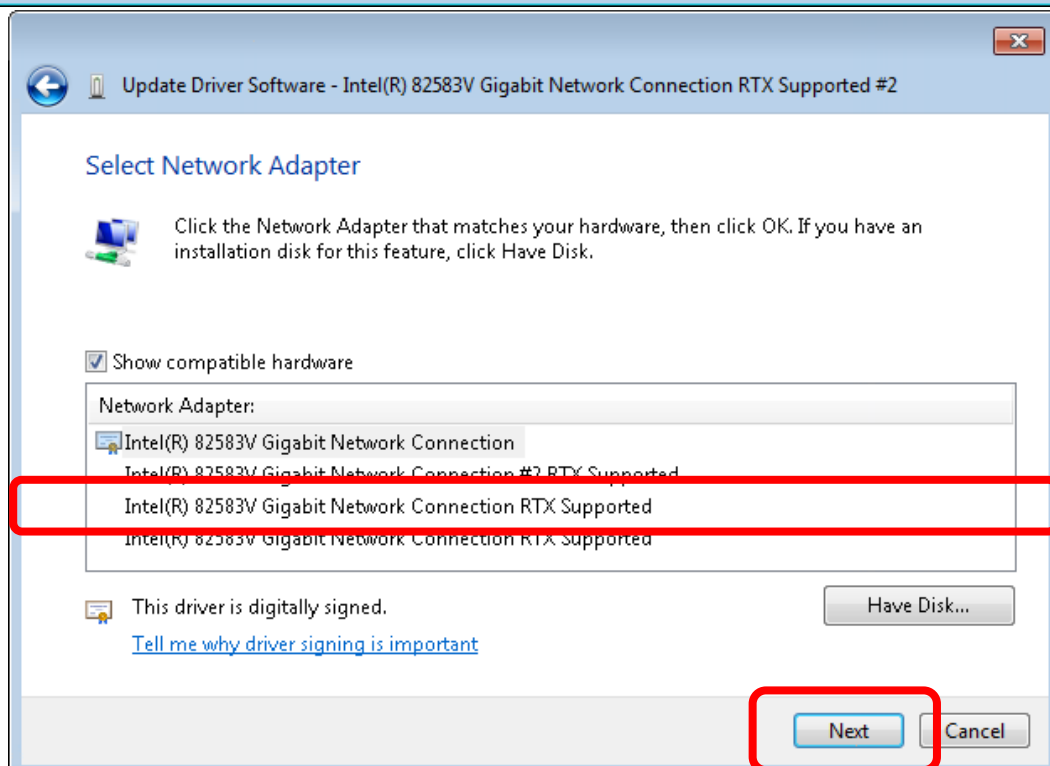
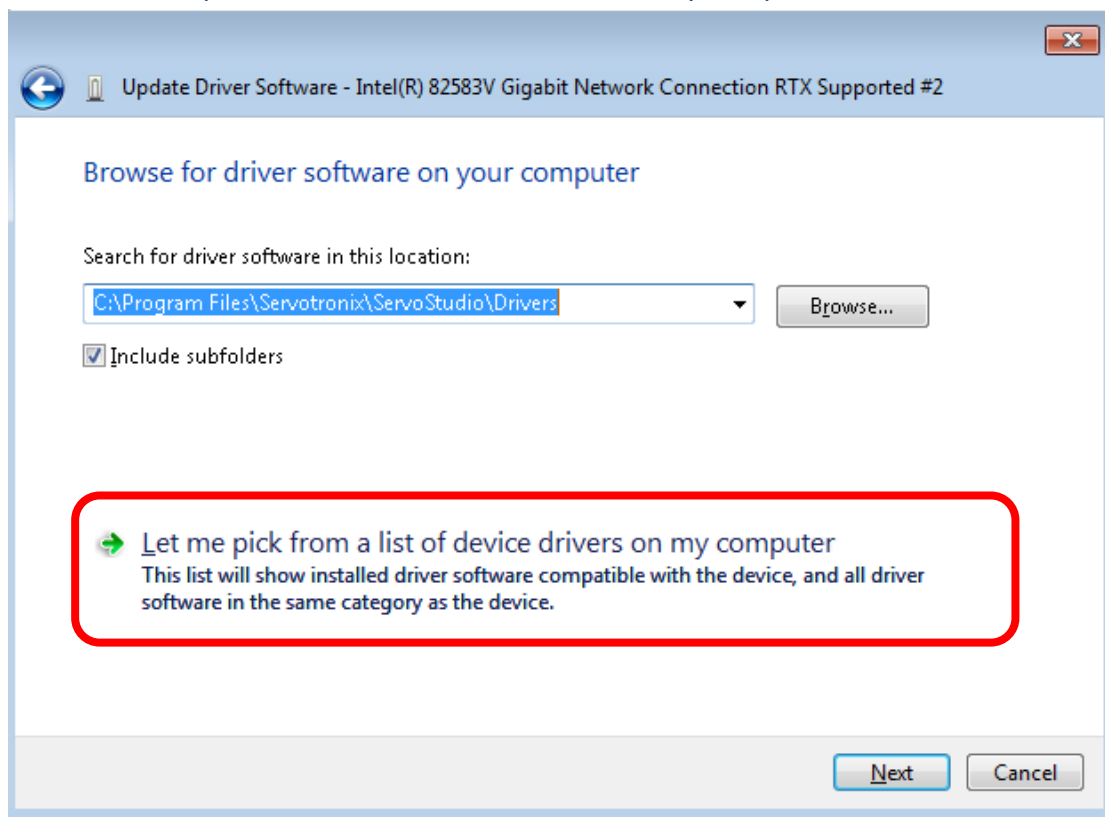
Open “Device Manager”, right click on NIC driver and select “Update Driver Software”



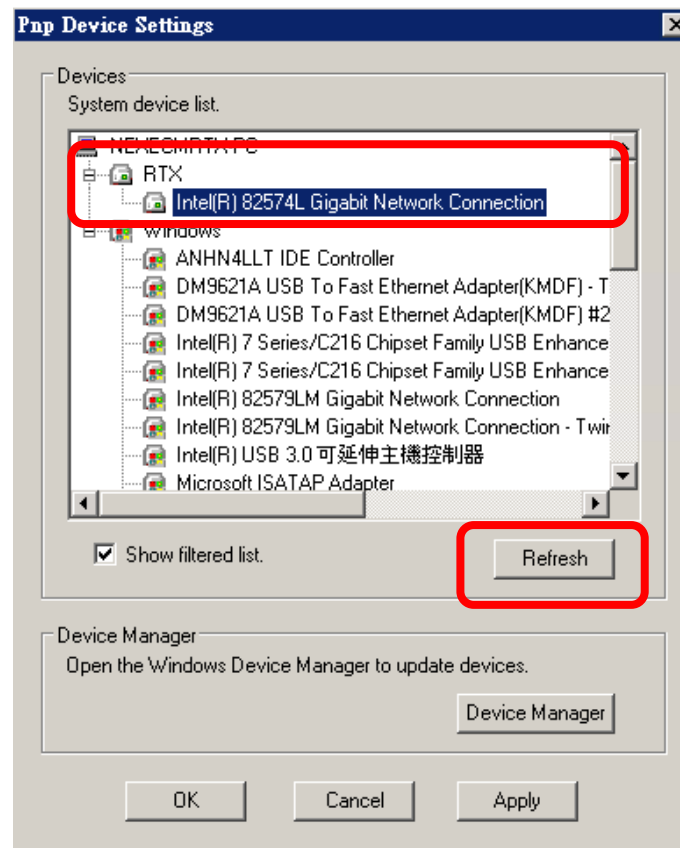
According to following figures to select the RTX supported network device driver:



Select "Let me pick from a list of device drivers on my computer"

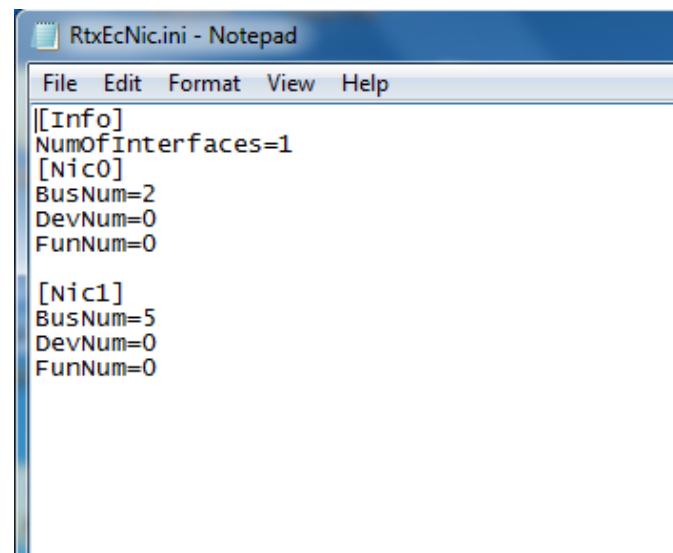


After installation is complete, you could confirm the RTX driver is installed correctly in "Pnp Device Settings" page.



2.3.2. Configure INI file "C:\RtxEcNic.ini"

Open the INI file "C:\RtxEcNic.ini" by text editor as follow:



NumOfInterfaces: The total number of EtherCAT network interface card

[Nic0] : The first network card information

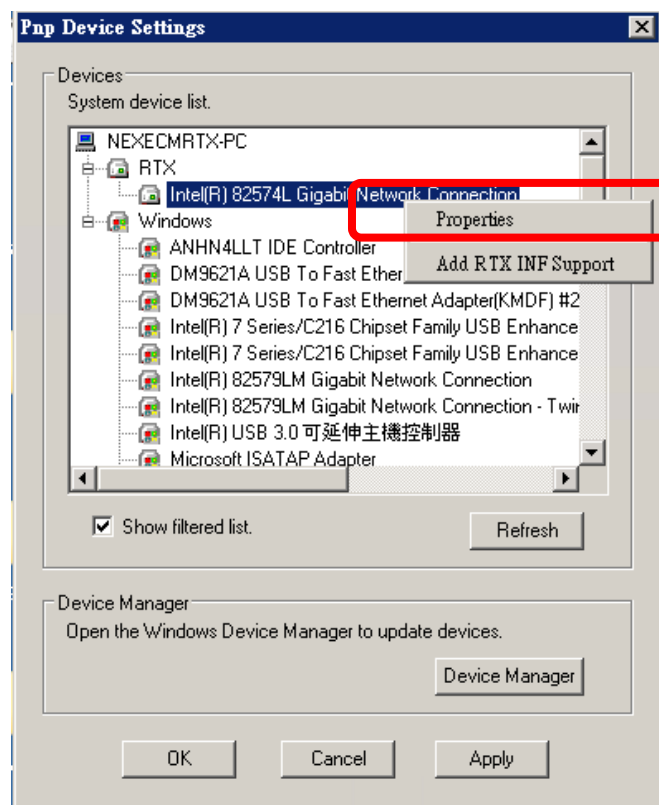
[Nic1]: Second network card information

...and so on

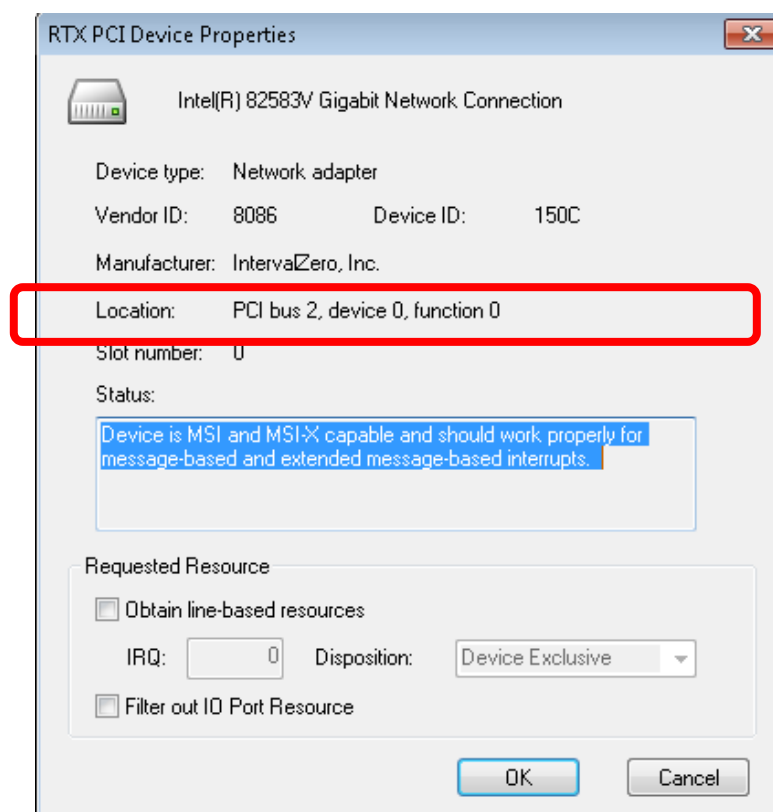
The value of “BusNum” , “DevNum” and “FunNum” – PCI location can be check via following steps:

Right click on network card in “Pnp Device Settings” page and select “properties” and popup “Pnp Device Settings” page. The “Location” information can be found.

Caution: This INI file must be placed in "C: \ RtxEcNic.ini" path



Right click on network card and select “properties”

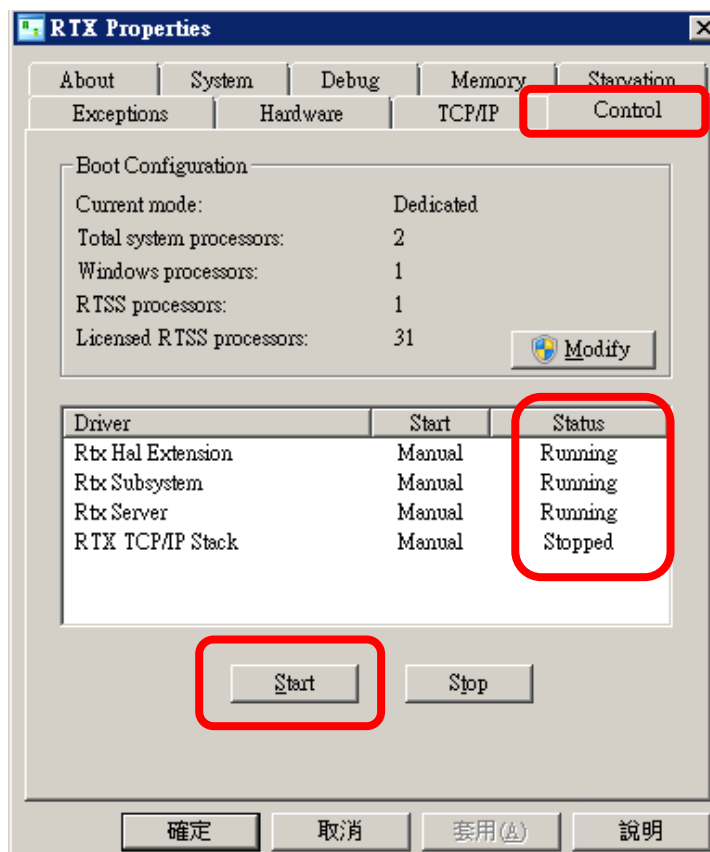


The “Location” information can be found for INI file

Start RTX Subsystem

Open RTX tools program: "RTX properties"

in the "Control" page (see below), click the "Start" button to start RTX system then RTX's Driver status will switch to "Running" state.



When the above steps are completed, NexECMRtx pre-installation steps have been completed. Above steps only need when first installation NexECMRtx.

2.3.3. Develop EtherCAT Master RTX Application

The following steps describe how to use Visual studio to develop EtherCAT Master RTX applications. (We use Visual studio 2010 as an example)

1. Add a new project (File/New), select RTX Application, importing the project name.

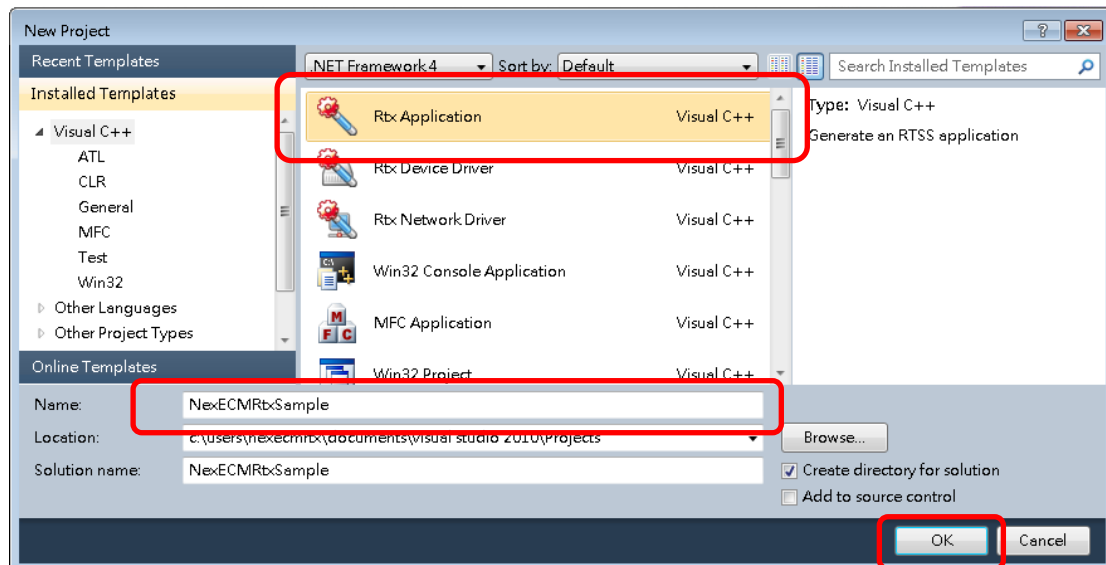


Figure 2-6: New Project

2. Setup RTX Application settings, see below

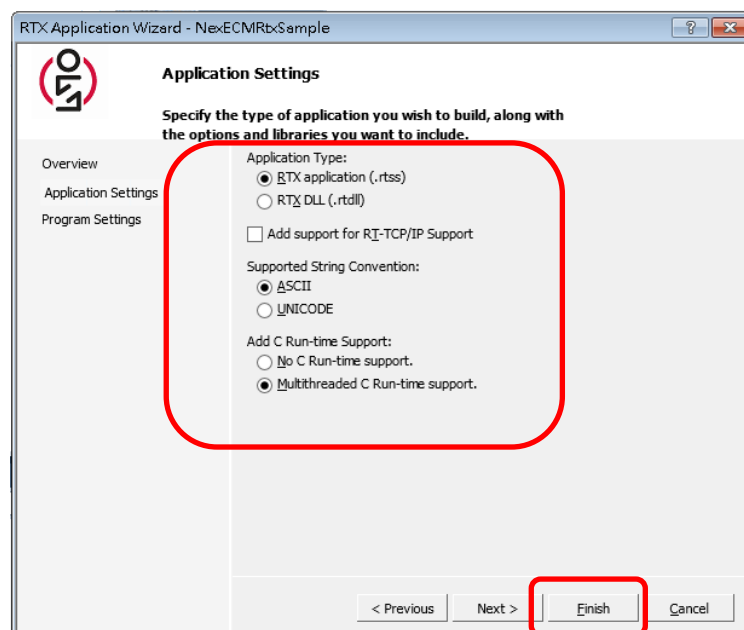


Figure 2-7: RTX Application settings

3. Use NexECMRtx API libraries; right click the properties in Solution Explorer.

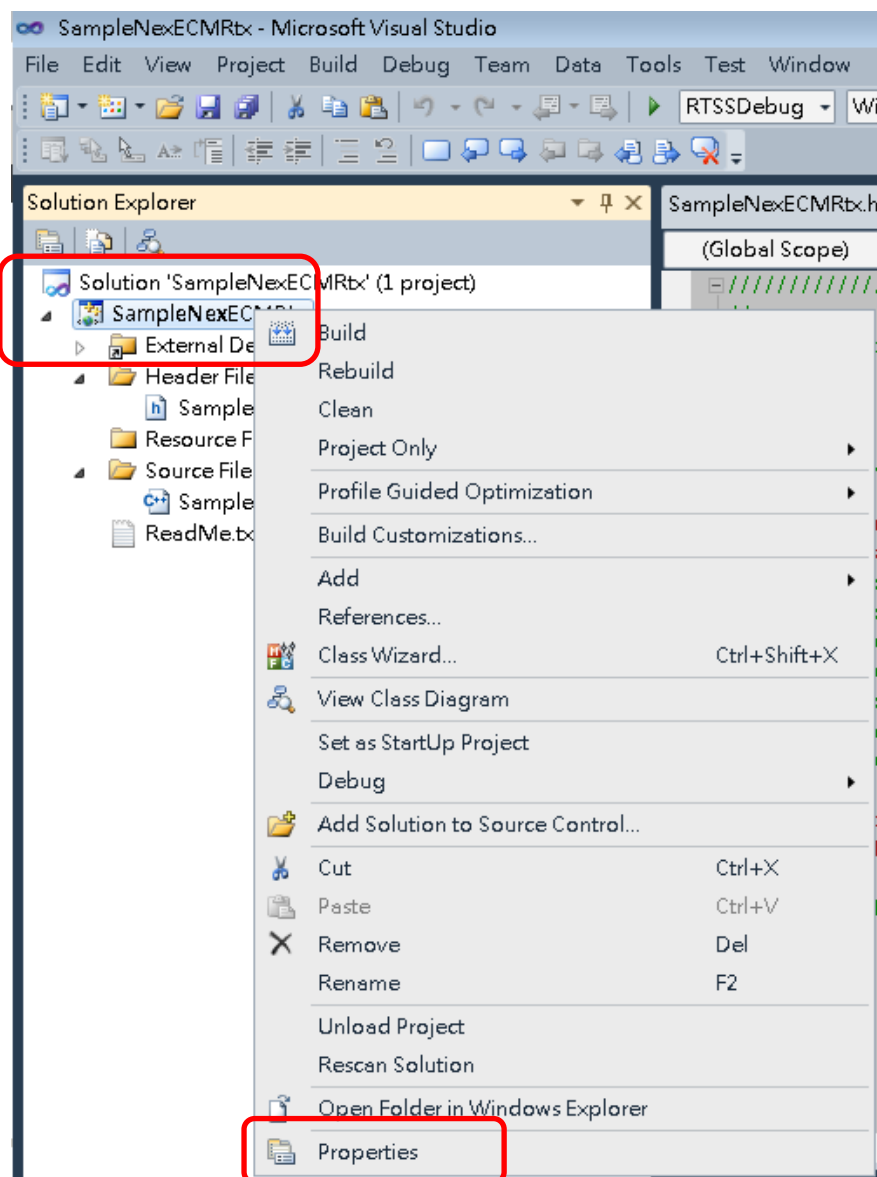


Figure 2-8: Solution Explorer

In C/C++ \ General, set “NexECMRtx” as the “Additional Include Directories”

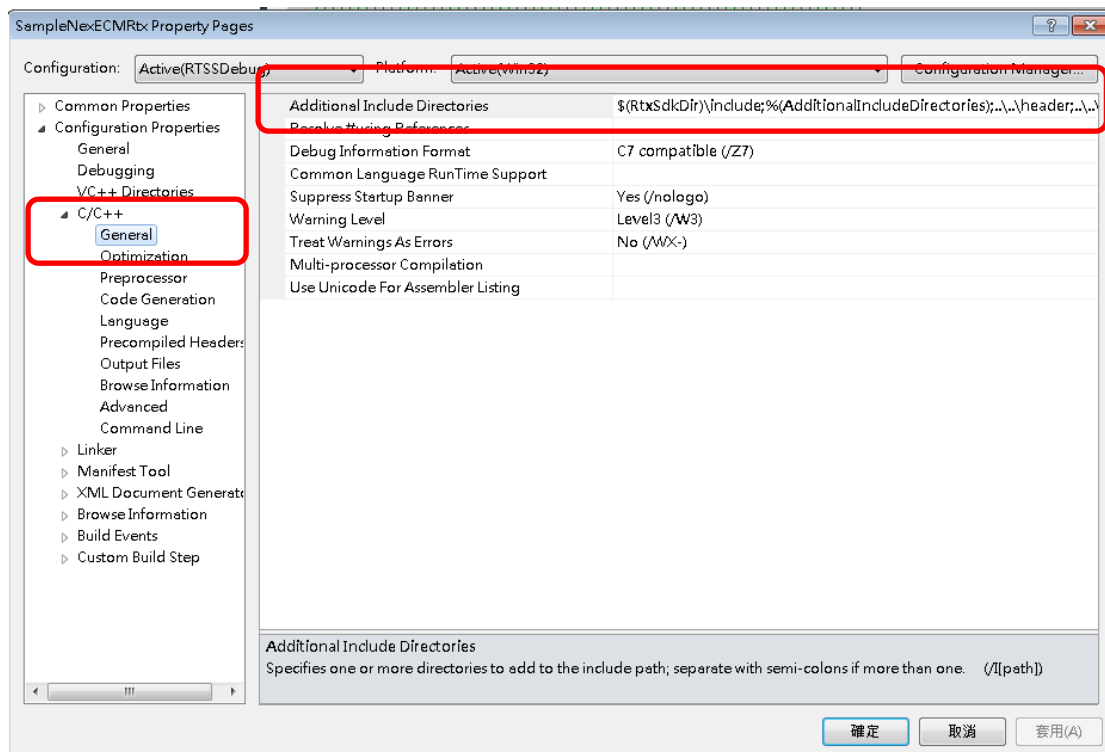


Figure 2-9: NexECMRtx Property page

In Linker\General, set path for “NexECMRtx.lib”

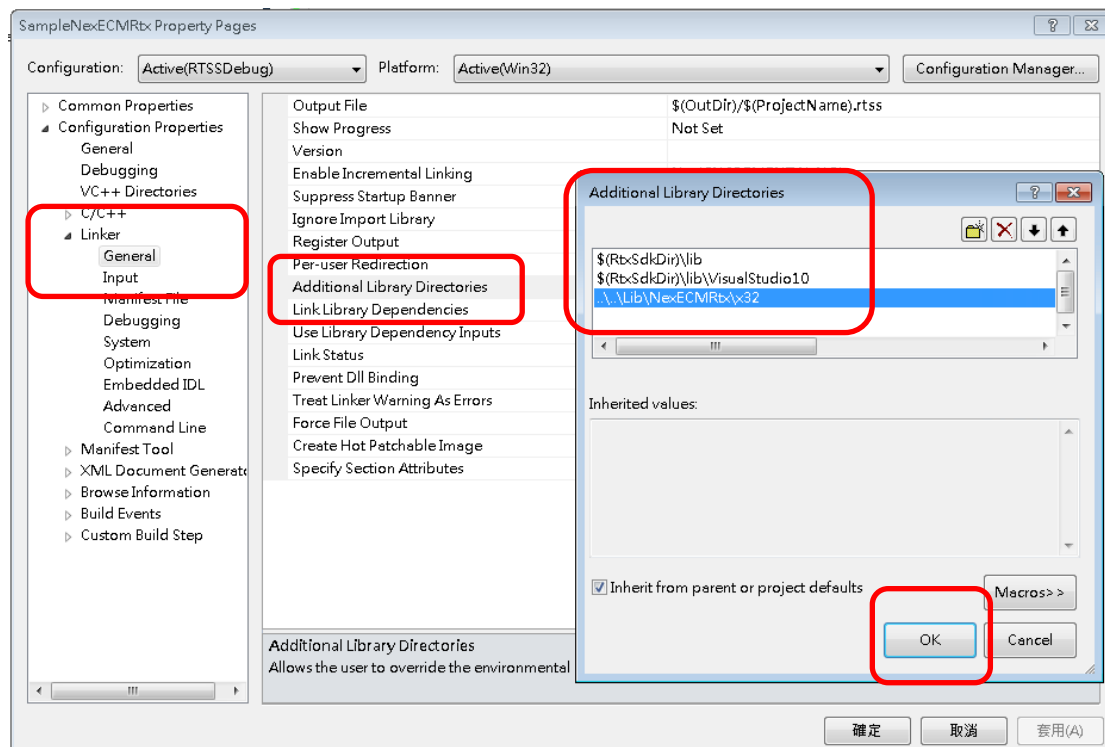


Figure 2-10: NexECMRtx Property page

In Linker\Input, set path for “NexECMRtx.lib”

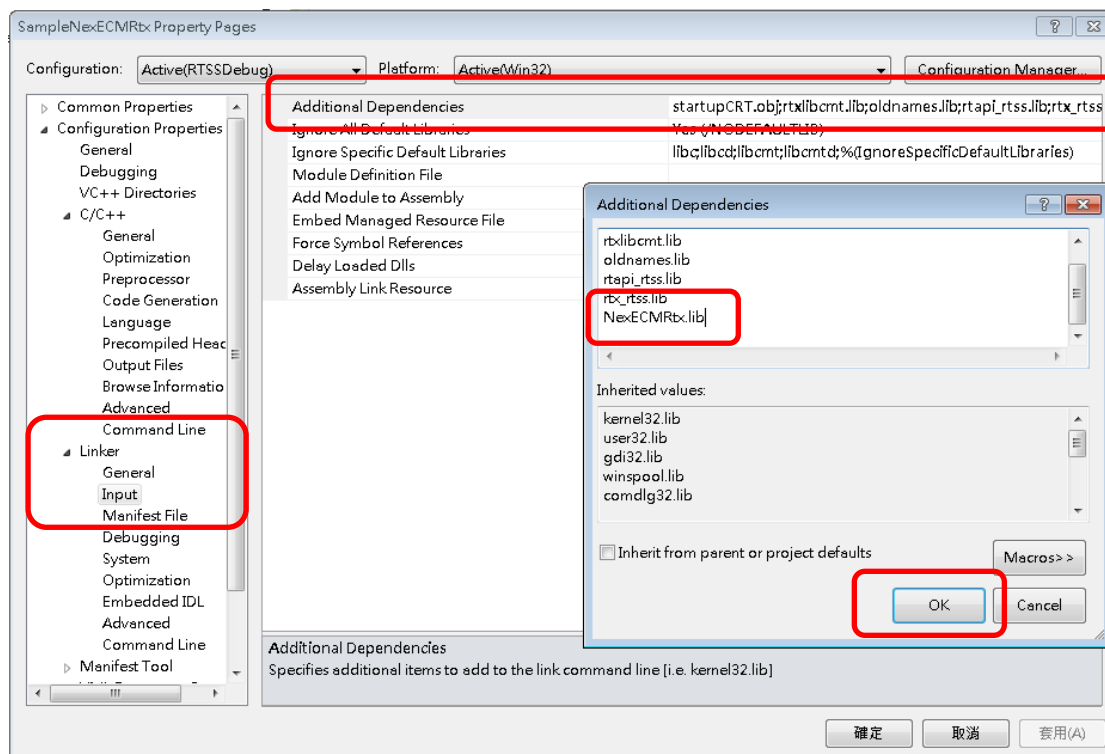


Figure 2-11: NexECMRtx Property page

Be noted when compiling, must select “RTSSDebug” or “RTSSRelease”

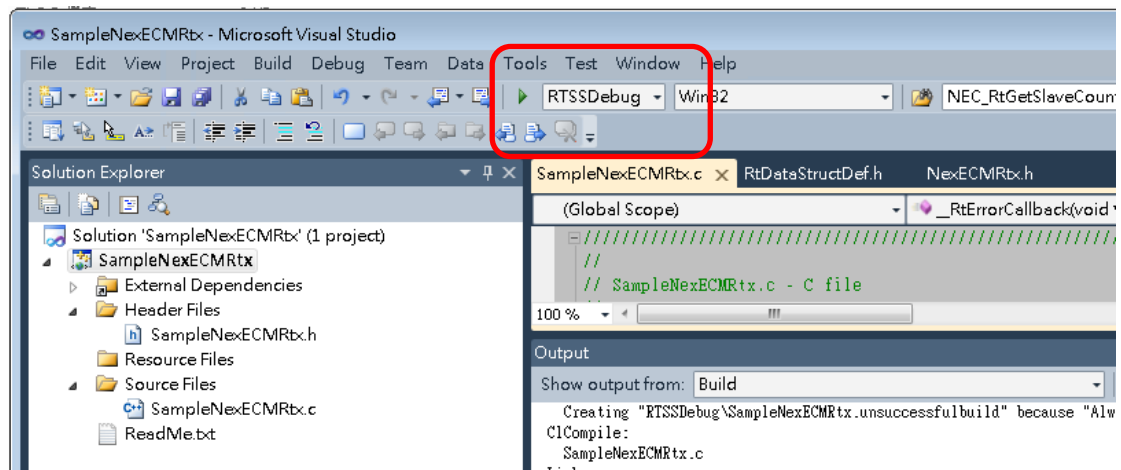


Figure 2-12: RTSSDebug

Build RTX application

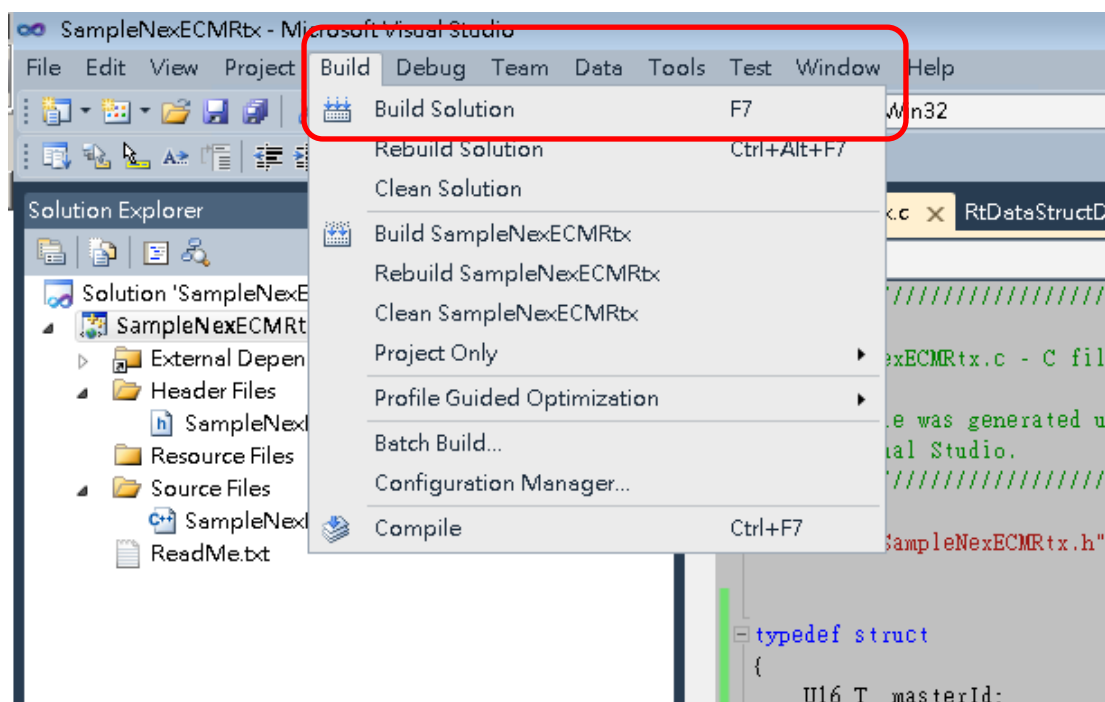


Figure 2-13: Build Solution

3. EtherCAT Utilities

3.1. NexCAT Introduction

You can complete the following with NexCAT master utility:

1. EtherCAT slave device scan
2. Import ESI file, export ENI file
3. Edit CoE slave devices PDO Mapping
4. ProcessData access
5. CoE slave devices SDO communication test
6. Monitoring the EtherCAT communication quality
7. Slave devices operation test

3.1.1. Software Requirements

Microsoft .Net framework 4.0 & Visual Basic Power Packs 3.0 are must. Please make sure they are properly installed.

- Microsoft .Net framework 4.0 Official free download:

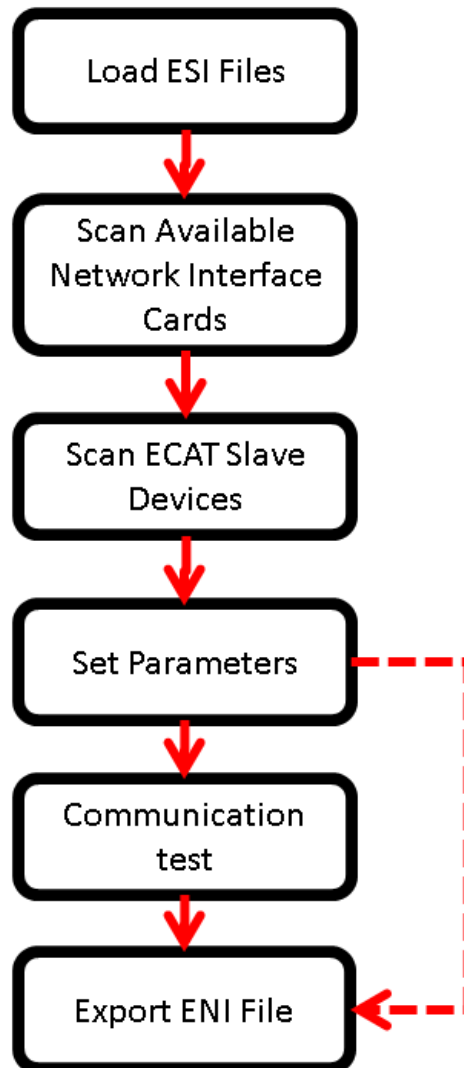
<http://www.microsoft.com/zh-tw/download/details.aspx?id=17718>

- Visual Basic Power Packs 3.0 Official free download:

<http://download.microsoft.com/download/1/2/A/12AA9B28-4F67-42C3-9319-684E8AD6F0AE/VisualBasicPowerPacks3Setup.exe>

3.1.2. NexCAT Operation Flow

The basic operation flow of NexCAT is as follow:



ESI: A XML file to describe the EtherCAT Slave devices Information,

ENI: A XML file to describe the EtherCAT Network Information

- **Load ESI Files:**

When NexCAT program starts, it will automatically import all the files in the folder which location is "\\ESI" of NexCAT installation path, the file name extension is * .xml

- **Scan Available Network Interface Cards:**

NexCAT can detect RTX environment and automatically find the all available network interface cards. About RTX network card driver installation please refer

to CH2.3.

- **Scan ECAT Slave Devices:**

NexCAT scans the ECAT slave devices on the selected network port.

If a device has no matched ESI file (VendorID, DeviceID not matched), it will be defined as “Unknown”. Move the mouse cursor to "Unknown" device will pop up the hardware information (VendorID, DeviceID and RevisionNumber).

- **Set Parameters:**

NexCAT generates the plan of PDO and ProcessData memory according to ESI files, then export to ENI file automatically. Users can also use the NexCAT built-in PDO mapping editor to customize their own plan, and then export the final setting to ENI file.

- **Communication Test:**

User can start all the EC-Slave devices directly; the status will be changed from initial state (INIT) to operation state (OP). If there is a slave device which can not be transferred to operation state successfully, you can find the status and messages from the main page's area 4 and area 5 (In following figure).

- **Export ENI file:**

If the tests on each devices show normal, the user can use the function “Export ENI”, to export the ENI file to the storage device. Actually, when you use the “Start Network” feature, the system automatically exports the current settings and network topology to ENI file.

(The default path is C:\ENI_NexCAT_Export.xml)

3.1.3. ESI & ENI

ESI (EtherCAT Slave Information) is a hardware description file, which describes the information and related parameters and memory configuration, which including the device support protocol. EtherCAT slave device hardware vendor will offer this file. The ESI file is used by NexCAT (NEXCOM EtherCAT configuration tool) to generate the EtherCAT Network Information file (ENI).

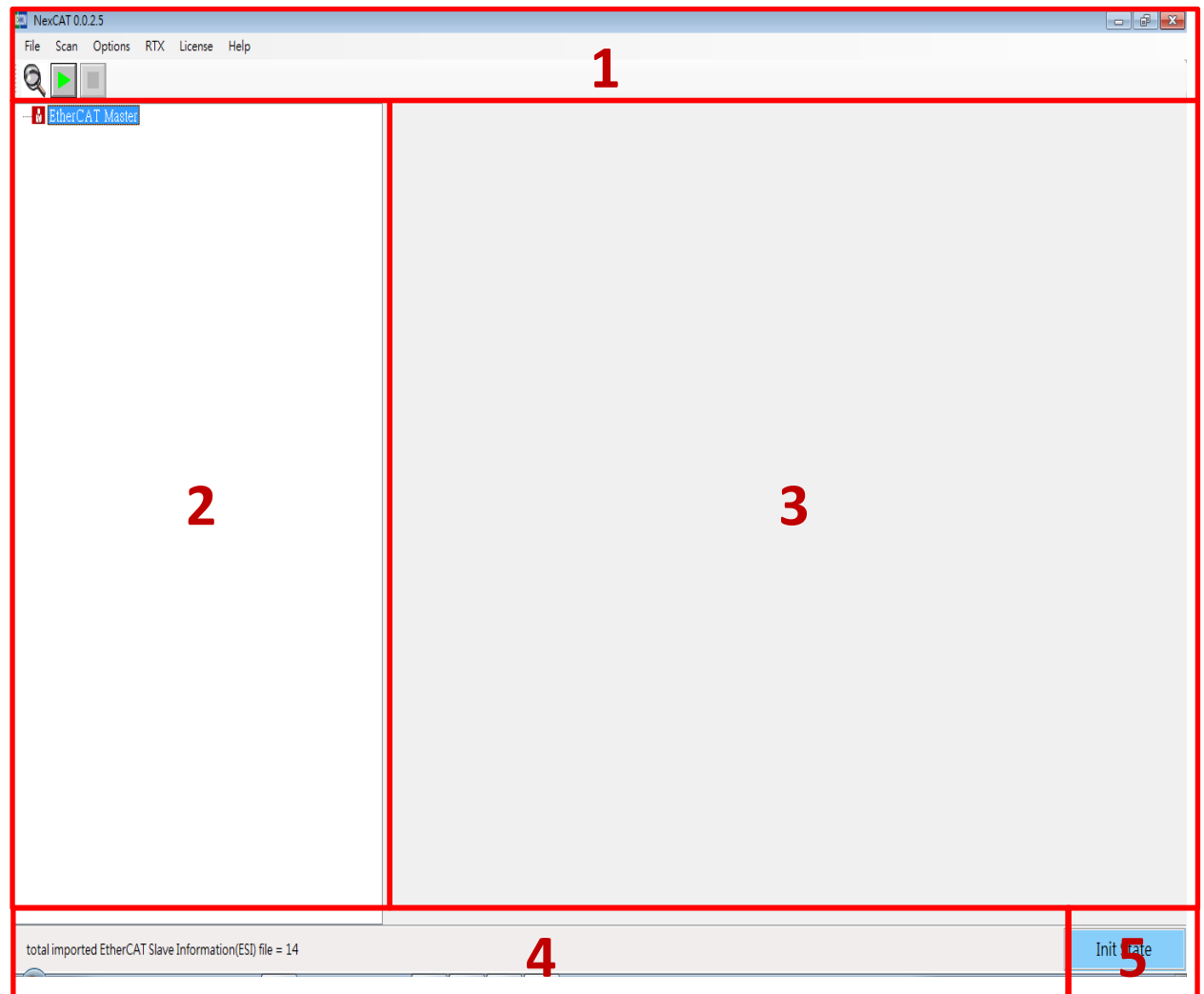
* EtherCAT environment configuration will use ENI as the default value. We strongly suggest not modify it and check with slave devices vendor for latest ENI file.

ENI (EtherCAT Network Information) contains the necessary settings to configure an EtherCAT network. The XML-based file contains general information about the master and the configurations of every slave device connected to the master. Furthermore, the file defines the EtherCAT commands for initialization that should be sent during a specific transition (e.g. from Init to Pre-Operational) according to the EtherCAT state machine. The file can be created by a “NexCAT” (NEXCOM EtherCAT configuration tool) and can be loaded by an EtherCAT master.

*We strongly recommend not changes the contents of ENI, if the network is configured with an additional transaction, please use NexCAT to output a new ENI file after changes




3.1.4 NexCAT Main Page

NexCAT Main Page are divided into 5 areas, we will explain it in the following:



- **Area 1:**

Show the software name and version, ex: NexCAT 0025.

Icon	Description
	Scan NIC: Find the available network interface and displayed on the form
	Start Network: Start communication and export ENI file to the default path (C:\)
	Stop Network: Stop all EtherCAT slave devices communication

- **Area 2:**

Show the entire network topology & all on-line EtherCAT slave devices. If the EtherCAT slave device fails to be scanned and shows “Unknown”, in this, please updates the ESI file of the slave device by contacting the slave device supplier and imports it again.



“Unknown” device , Popup info when cursor move onto the item.

- **Area 3:**

Show the menu of parameters. You can set the slave device parameters and master parameters here.

- **Area 4:**

Show message and error code.

- **Area 5:**

Show the status of EtherCAT slave devices. Currently we have 4 statuses:

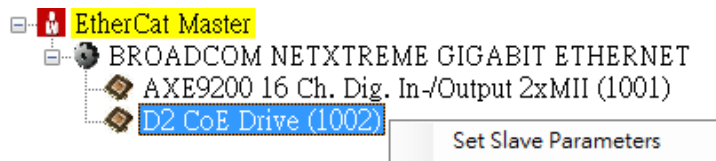
1. **Initial:** No communication all slave devices are in initial state.
2. **Error:** Start communication but slave devices cannot be switched to OP state. Common errors are: ENI file does not match with the actual network topology; ESI version does not match with the slave device version and so on.
3. **Retry:** When the parameter “Link Error Mode” of ECAT master is set to “Auto re-connect” (Refer to 3.1.6) and slave device is in “OP” state but face of link problem, The master show “Retry” status and trying to re-connect the

disconnected slaves until back to work. Master will try to reconnect those lost link modules, other modules can operate as usual, this status is displayed continuously until all slaves back to “OP” state.

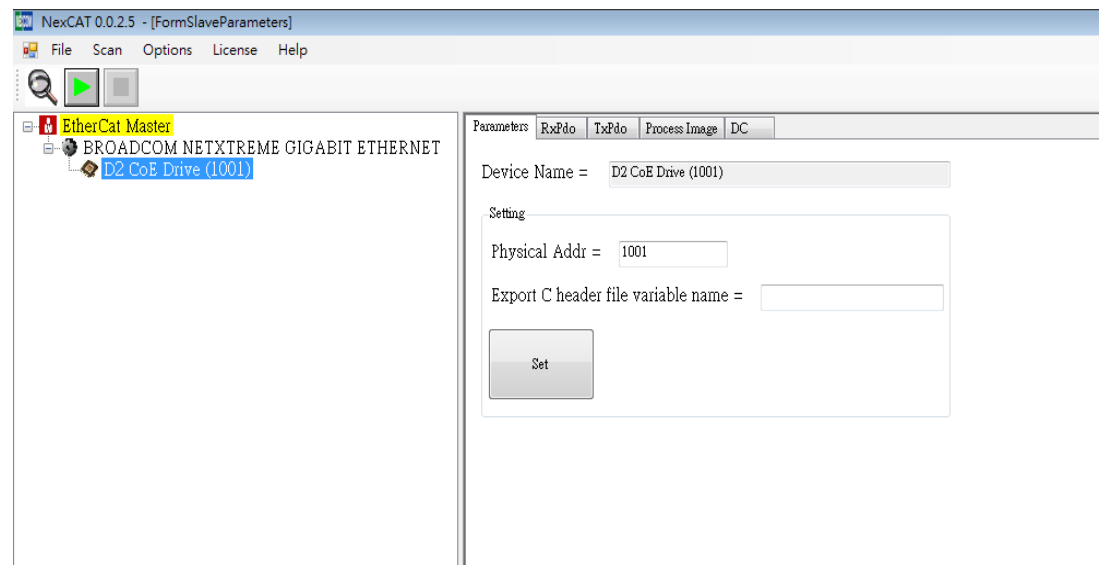
4. **Running:** Network is connected and all slave devices are in “OP” state.

3.1.5 Set Slave Parameters

Select the slave device and right click to show a pop-up menu, choose “set slave parameters”



Slave device setting page must be used before start of the Network, because all the parameters changed are valid only before start of Network. If user changed the setting after the start of Network, the re-start network is must.



1. Parameters Tab

Device Name =

Setting

Physical Addr =

Export C header file variable name =

Device Name: Show the name of current selected slave device.

Physical Addr: Define the node address (configured address) for a slave device.

Export C header file variable name: Export the process image for each slave, it must be used with function “Export C file” of Master Parameters setting (Refer to CH 3.1.6).

```
#define _Physical Addrsss (+ variable name)_ObjectName    [ProcessData offset]
```

Example:

Export C header file variable name = “_AXIS”

Export C header will be:

```
#define _1001_AXIS_Statusword          16777216
#define _1001_AXIS_PositionActualValue 16777218
#define _1001_AXIS_VelocityActualValue 16777222
#define _1001_AXIS_Controlword         16777216
#define _1001_AXIS_TargetPosition      16777218
```

2. RxPdo & TxPdo Tab

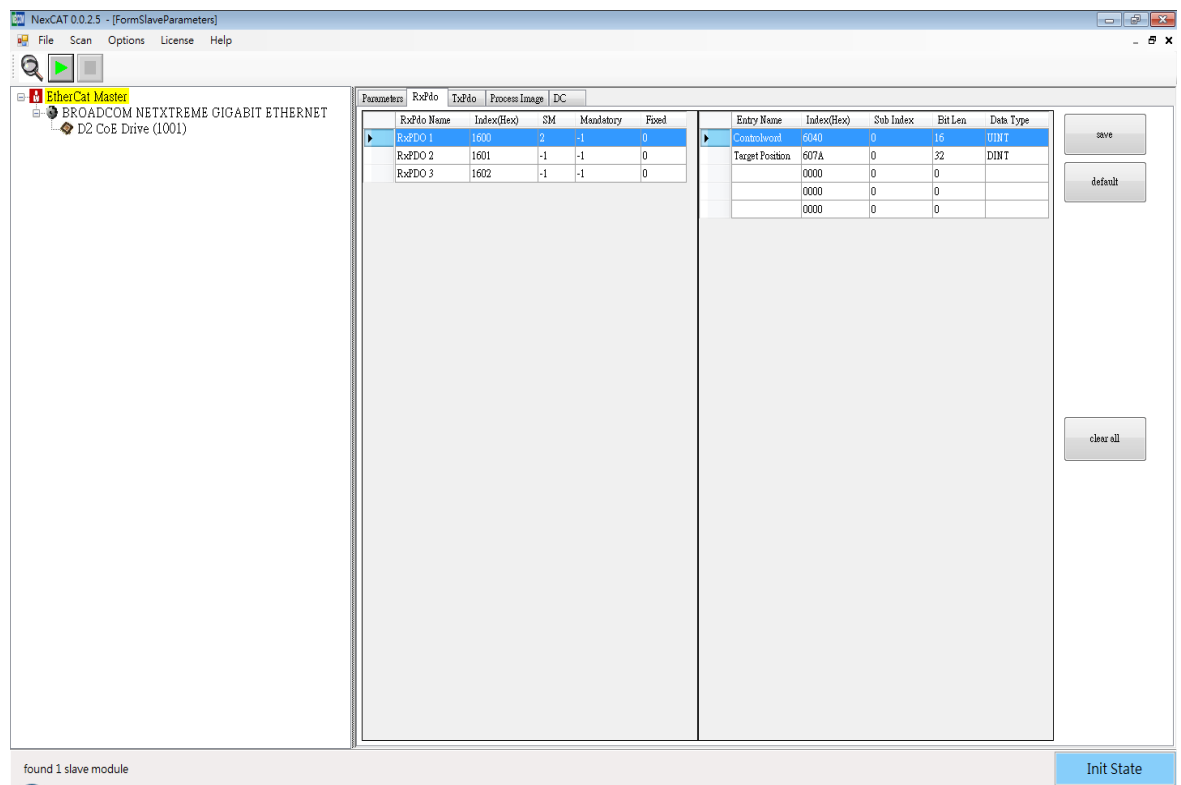


Table description:

- RxPdo(TxPdo) Name: Default name is from ESI file, user can change and then export to ENI
- Index: Parameters from CoE. Changes are not recommended
- SM: Number of Sync Manager, user can change
- Mandatory: Define the necessary parameters
- Fixed: Defines the parameter whether the user can change
- Entry Name: From CoE, user can change, export to ENI
- Indicator: Parameter from CoE. Changes are not recommended
- Sub Indicator: Parameter from CoE. Changes are not recommended
- BitLen: Parameter from CoE. Changes are not recommended
- DataType: Parameter from CoE. Changes are not recommended
- Save Button: Save after the editing
- Default Button: Back to default ESI setting
- Clear All Button: Clear PDO setting

3. Process Image Tab

User can edit on “RxPdo” or “TxPdo” tab. After editing, you can check the correspondent memory address in this tab. The edited setting will be valid after you click on the “save” button.

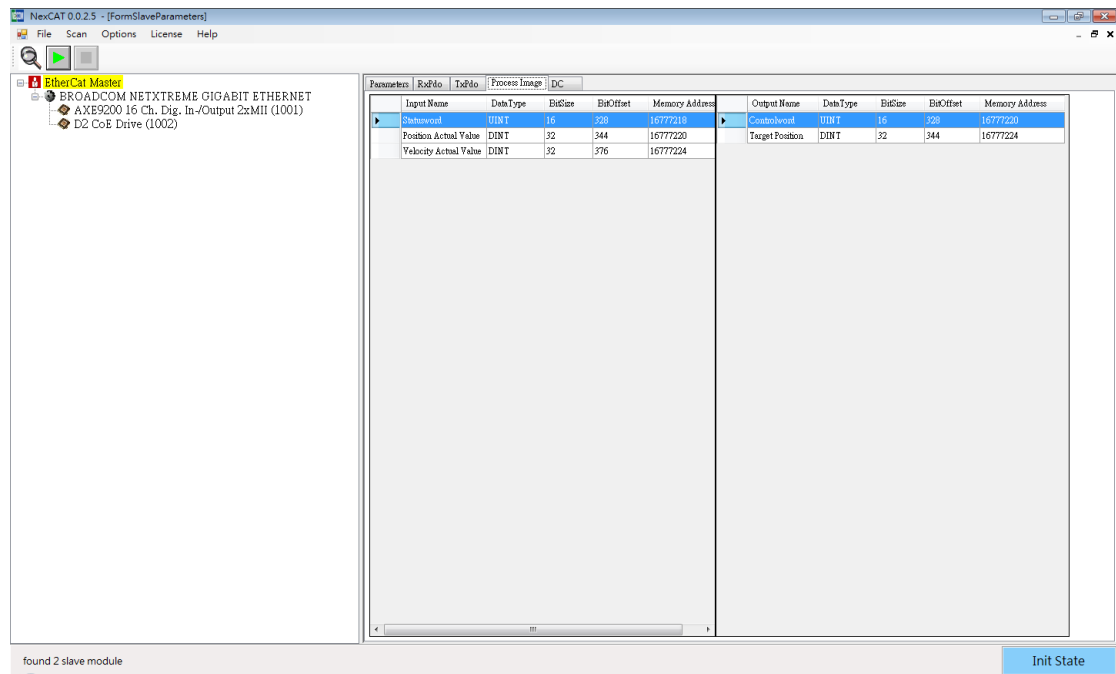


Table description:

- Input(Output) Name: It use the name on “RxPdo” or “TxPdo” tab
- BitSize: Variables Memory size
- BitOffset: Variable offset (base on setting in “RxPdo” or “TxPdo”)
- Memory Address: Variables Memory Address

4. DC tab

This tab is used to set DC mode. Default DC settings of each slave are from its ESI file.

Parameters	RxPdo	TxPdo	Process Image	DC
Setting				
Mode	=	DC		
Description	=	DC SYNC0		
DC SYNC Activation	=	0x0300		
		Apply To Other		Set

Mode (Description) :

Select the DC mode. If the slave support DC mode, the default is to enable "DC" sync mode. As long as (a) slave(s) device's DC mode be selected in network, EtherCAT Master will have a DC output information (function) of ENI File; To turn off the DC function from network, the user must be set all slaves as "free run" mode.

DC SYNC Activation : (ESC Register 0x0980~0x0981)

0x0000 –Disable SYNC0 & SYNC1 (Free Run)

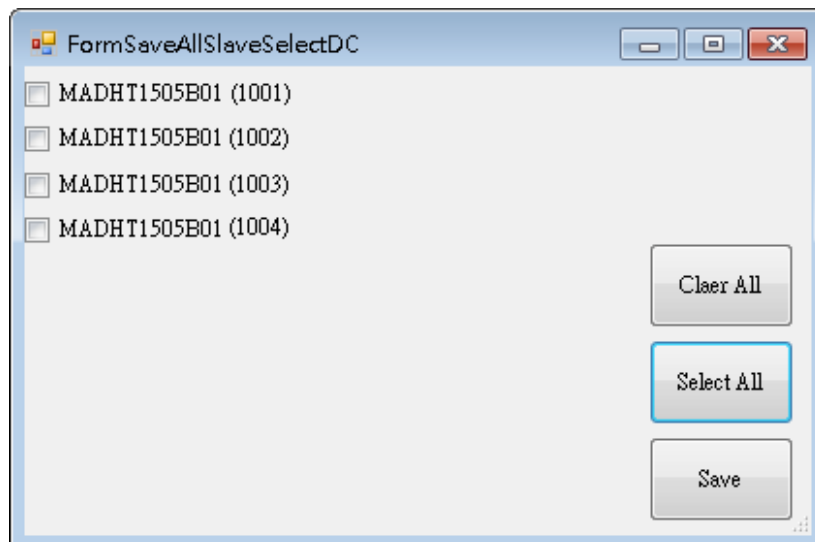
0x0300 – Activate SYNC0 (DC Sync)

0x0700 – Activate SYNC0 & SYNC1

This is the advanced setting. This column will be displayed according to ESI file when you select the DC mode. It is used to control DC SYNC signal output. Generally let it default.

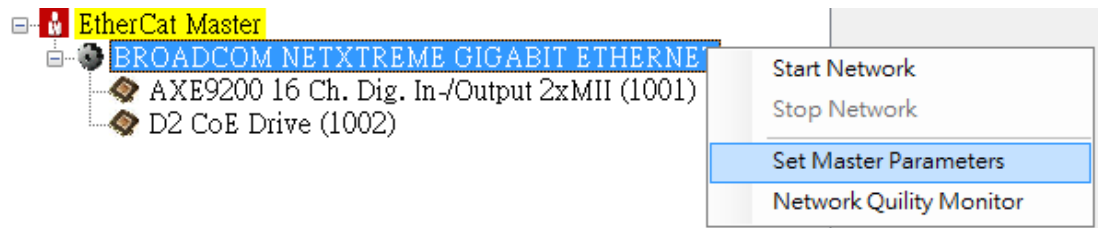
Apply To Other :

Apply current slave device's settings to other slaves. Click the button pop up following dialog.



3.1.6 Set Master Parameters

Select the device and right click to show a pop-up menu, choose “set master parameters”

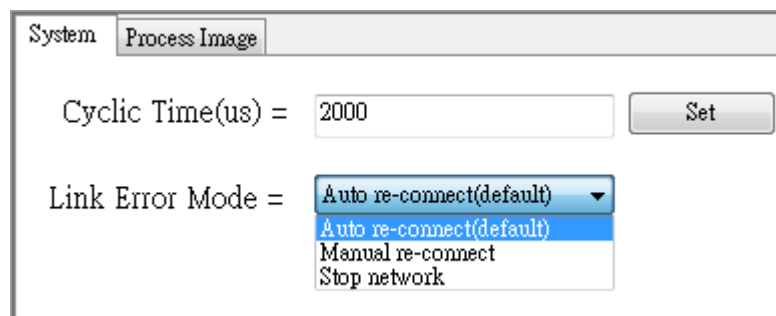


There are 2 tabs:

1. System
2. ProcessImage

Are described as below:

System tab



The Cyclic Time: Used to set the system performance. The values are communication time or refresh frequency between EC-Master and EC-Slave devices. The minimum value can't be larger than system limit value. This also can be set by calling API. Unit is micro second (us).

Link Error Mode: Behavior when link error. After network is starting, slave devices will be in “Operation” state. There are three modes when EC-Master detects the link error:

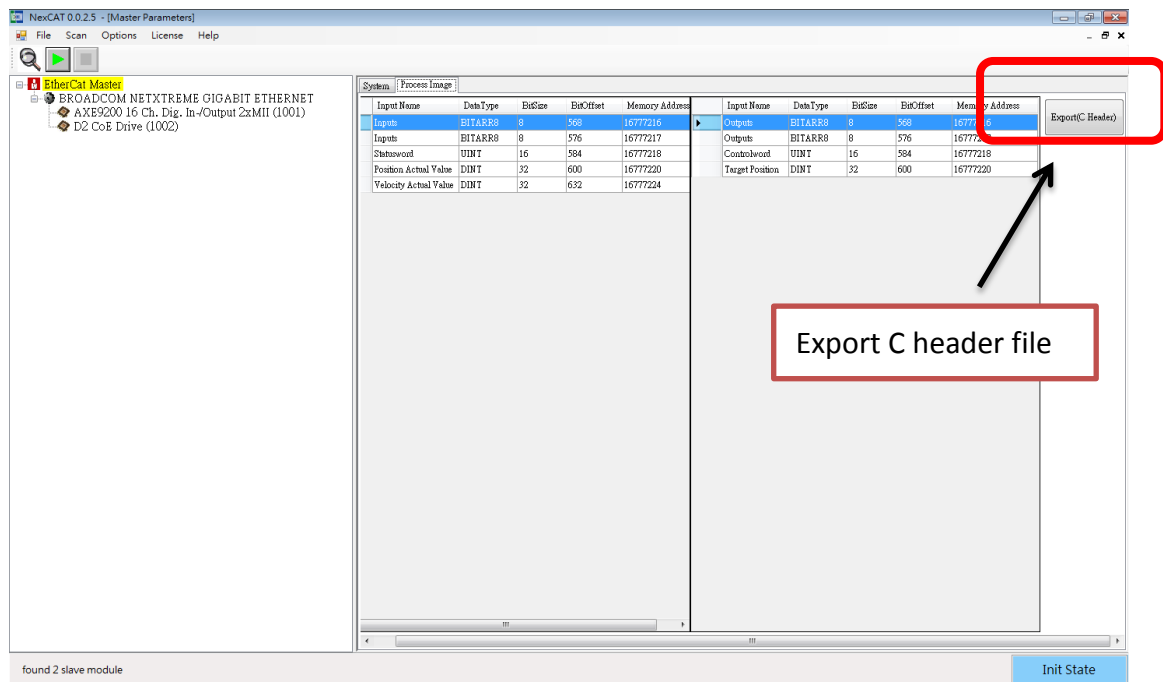
User also can set the mode by calling API, please refer to ch6.2.5 `NEC_RtSetParameter()`.

- Auto re-connect (default): When a slave device loses communication, the main page “Area 5” will appear “Slave Retry” message, while the system

continues to retry to re-connect automatically until the connection succeeds. Other slave device continues to work at the same time.

- **Manual re-connect:** When a slave device loses communication, other slave device continue to work normally, the main page "Area 5" will appear "Error message", continue to the next time when a network connection is successful.
- **Stop network:** When a slave device loses communication, EC-Master will stop the network. Finally, the main page "Area 5" will appear "Error message"

Process Image tab



- **Network process image map**
The format is the same with the processImage of a slave described in section 3.1.5, But here you can see the memory allocation for the entire network topology, or use "Export C Header File" function to output variables of each slave device. You also can write your own program when the memory is accessed directly through the API.
- **Export C header file for process image map**
Click "Export C Header File" button

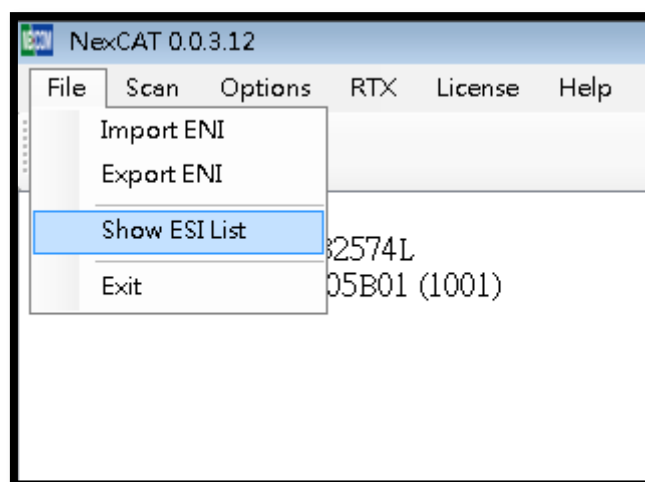
PDO memory mapping offset can be output as a C header (*.h), It is easy maintain your code using the define symbol when PDO mapping has changed. Output symbol format please refer to CH 3.1.5.

3.1.7 ESI List (ESI File management)

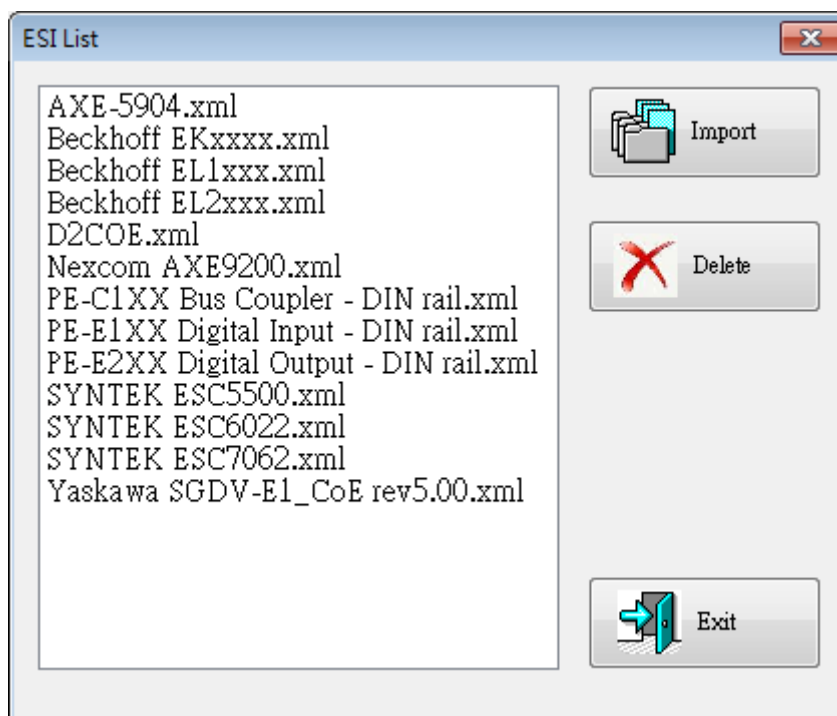
When using NexCAT to scan the devices, you can get how many slave devices and obtain the hardware information (Ex.Device ID etc). Through comparing the information, NexCAT will get which ESI belongs to. (About ESI file please refer to section 3.1.3). If the user gets a new ECAT slave device, he must import the ESI of the device.

2 methods to manage the ESI files:

1. Add / Remove the ESI file to the specified folder directly. When you add new ESI file, you need restart the NexCAT.
2. Use "ESI list" page to import / delete ESI files. The action of import & delete is applied immediately. No need to restart the NexCAT.



Show ESI list

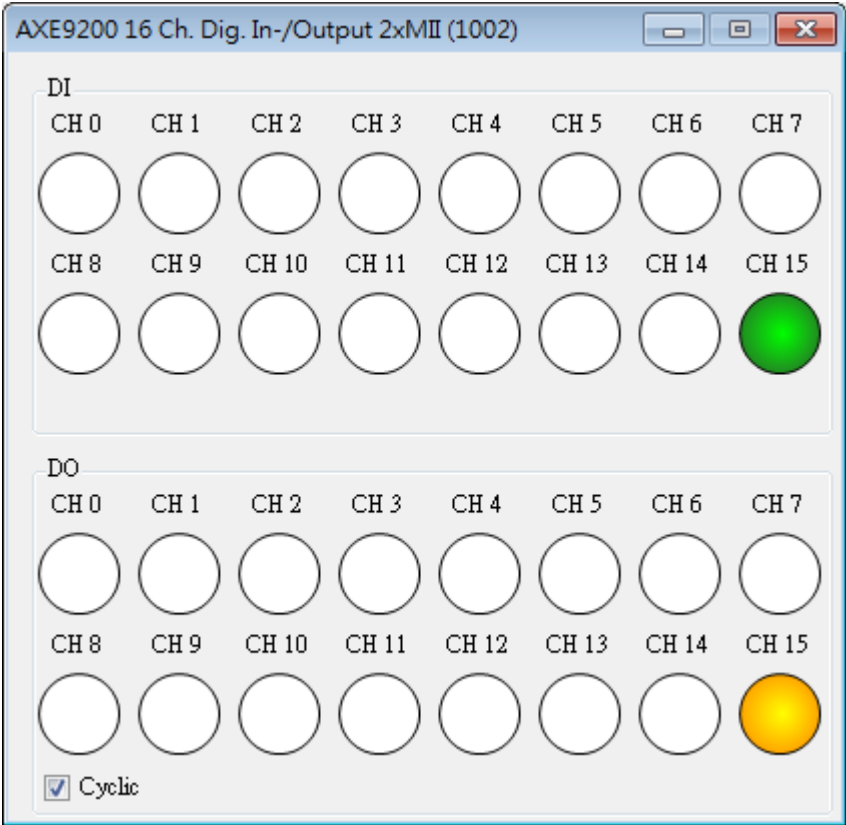


ESI List page

3.1.8 DIO User interface

In Area 2, double-click the selected DIO slave device which you want to testing, the DIO operation menu will appear. NexCAT will determine the device for DI, DO or DIO devices and automatically calculate the number of IO.

When the mouse cursor at the DO button, the user can manually press DO button to operate DO, or user can use the "Cyclic function", let DO slave device to run automatically to start Marquee features starting from small (0) to large, and repeated run. After you check the box for Cyclic, it operates automatically. After the check is canceled, the program stops at the last channel being executed in operation.

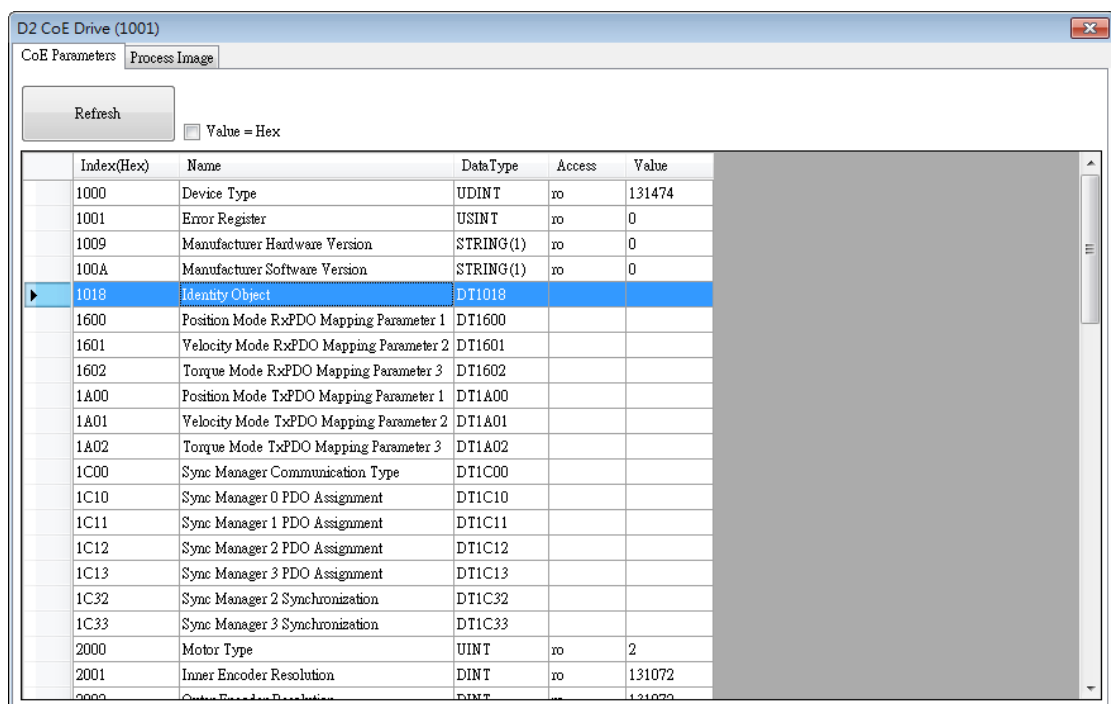


3.1.9 CoE-SDO Operation page

In Area 2, double-click the selected CoE slave device which you want to testing, the CoE operation menu will appear. NexCAT will automatically determine whether the slave device supports CIA 402.

Press the Refresh button and this will update parameter values automatically, the user can choose to represent decimal or hexadecimal display format. If a parameter is float, then the parameter from binary system will display in float.

If the user wants to change parameters value, you can use the mouse, click the left mouse button twice quickly, you may edit the parameters value. After editing is completed, press the Enter key or leave the table then it can be successfully written. If the write fails or does not meet the standard written format data form, the parameter values automatically go back to the state before editing.

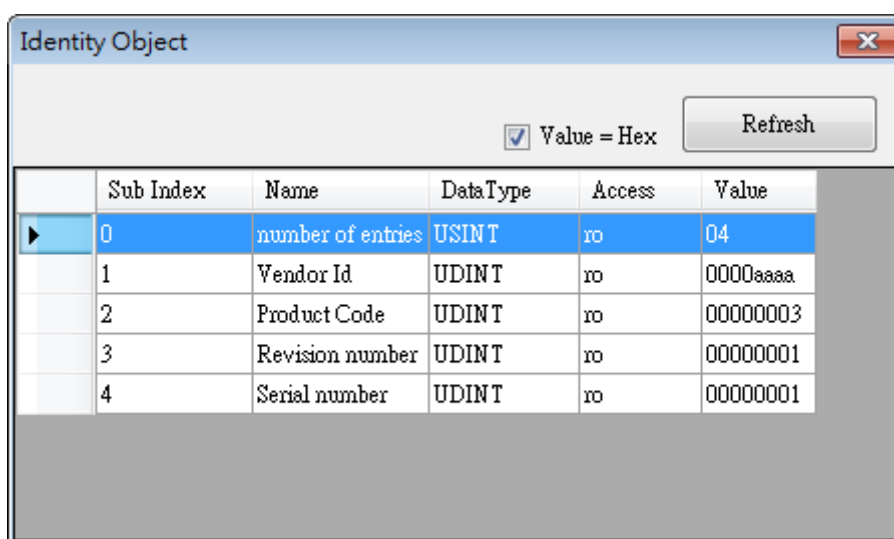


Index(Hex)	Name	DataType	Access	Value
1000	Device Type	UDINT	ro	131474
1001	Error Register	USINT	ro	0
1009	Manufacturer Hardware Version	STRING(1)	ro	0
100A	Manufacturer Software Version	STRING(1)	ro	0
1018	Identity Object	DT1018		
1600	Position Mode RxPDO Mapping Parameter 1	DT1600		
1601	Velocity Mode RxPDO Mapping Parameter 2	DT1601		
1602	Torque Mode RxPDO Mapping Parameter 3	DT1602		
1A00	Position Mode TxPDO Mapping Parameter 1	DT1A00		
1A01	Velocity Mode TxPDO Mapping Parameter 2	DT1A01		
1A02	Torque Mode TxPDO Mapping Parameter 3	DT1A02		
1C00	Sync Manager Communication Type	DT1C00		
1C10	Sync Manager 0 PDO Assignment	DT1C10		
1C11	Sync Manager 1 PDO Assignment	DT1C11		
1C12	Sync Manager 2 PDO Assignment	DT1C12		
1C13	Sync Manager 3 PDO Assignment	DT1C13		
1C32	Sync Manager 2 Synchronization	DT1C32		
1C33	Sync Manager 3 Synchronization	DT1C33		
2000	Motor Type	UINT	ro	2
2001	Inner Encoder Resolution	DINT	ro	131072
2002	Outer Encoder Resolution	DINT	ro	131072

CoE parameters

If the parameter of data type is “dataType”, it indicates that the parameters contain sub parameters (Sub index), the user may want to access in the parameters by double-click the mouse, determine if the program does have a sub parameters (Sub Indicator), there will be a child window shown below. It is

the same to read and write as mentioned in previous chapter.

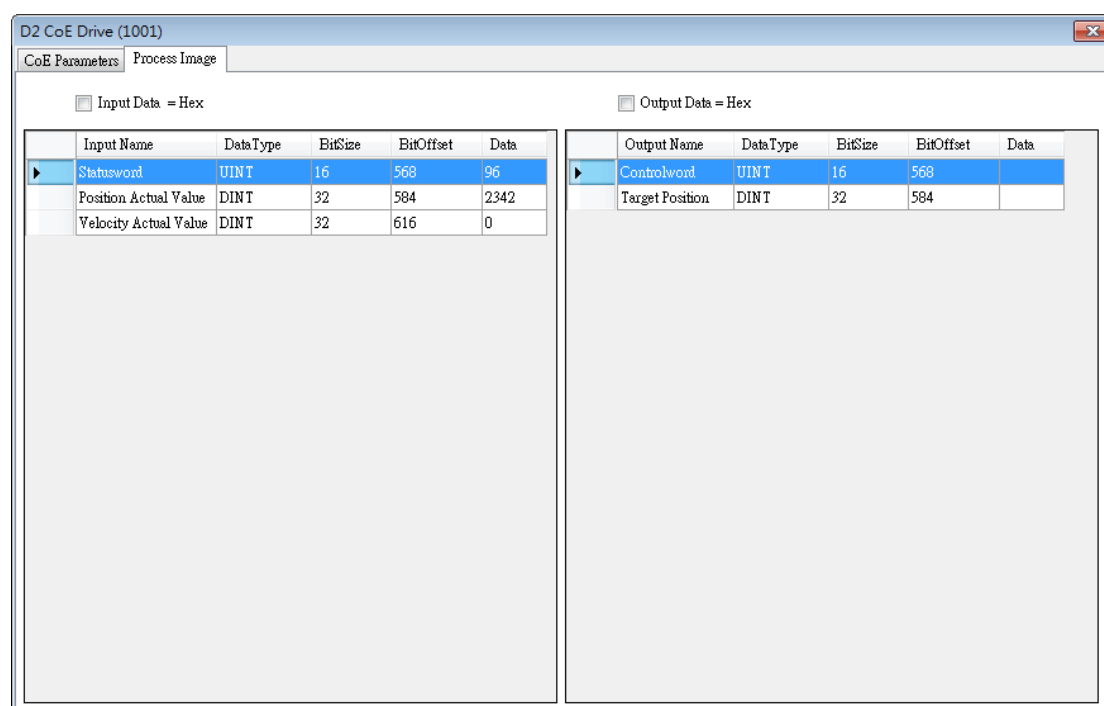


The screenshot shows a window titled "Identity Object" with a "Value = Hex" checkbox and a "Refresh" button. Below is a table with 6 columns: Sub Index, Name, DataType, Access, and Value.

	Sub Index	Name	DataType	Access	Value
▶	0	number of entries	USINT	ro	04
	1	Vendor Id	UDINT	ro	0000aaaa
	2	Product Code	UDINT	ro	00000003
	3	Revision number	UDINT	ro	00000001
	4	Serial number	UDINT	ro	00000001

Sub parameters

3.1.10 Process Image Parameters Operation page



The screenshot shows a window titled "D2 CoE Drive (1001)" with tabs for "CoE Parameters" and "Process Image". The "Process Image" tab is active, showing two tables: "Input Data" and "Output Data". Both tables have checkboxes for "Input Data = Hex" and "Output Data = Hex".

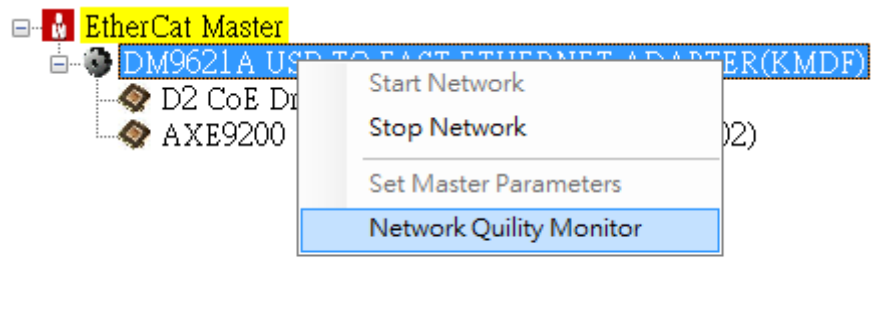
Input Name	DataType	BitSize	BitOffset	Data
▶ Statusword	UINT	16	568	96
Position Actual Value	DINT	32	584	2342
Velocity Actual Value	DINT	32	616	0

Output Name	DataType	BitSize	BitOffset	Data
▶ Controlword	UINT	16	568	
Target Position	DINT	32	584	

Users can access PDO (process data object) data after starting network. When the check box "input data (output data) = hex" is checked, the data in the table is display as hexadecimal format.

3.1.11 Network Quality Monitor

Users can open network communication quality test page after starting network. Do Master to each slave device communication packet test. To show this page, you can right click the mouse on the node of network card(NIC) in NexCAT area 2 and select the “Network Quility Monitor” and Network quality test page will appear.



Right click on NIC node.



Network Quality Monitor page

- Inc Address : The Slave ID will follow the order of the scanned, zero based.
- Send Frame Count: The numbers of test frames are sent to slave device, check if the slave devices are in “OP” state. The frequency of the send frames to 10 ms.
- Recv Frame Count: The number of response frames. Normally, Both Send Frame Count and Recv Frame Count should be consistent.
- Lost Frame Count: Lost frames

- Error Frame Count: The return frames data content does not belong to the slave device and state != OP

Their relationship are as in the following

Send Frame count = Recv Frame count + Lost Frame count

Recv Frame count = Normal Frame (state == OP) + Error Data Frame count.

3.2. NexECMRtxStarup

“NexECMRtxStartup.exe” provides the convenience while you're using EtherCAT Master. Based on “NexECMRtxConfig.ini”, we offer 3 major functions:

1. Load EtherCAT Master - NexECMRtx.rtss
2. Download ENI file (EtherCAT Network Information)
3. Load user's RTX appliation (ex:UserRTXApp.rtss)

You can modify NexECMRtxStartup.ini content by “Notepad” or text editing software to meet your current files placed circumstances. Usually you need to modify 1. “Application path” 2.”Network information file (ENI) path”. You can find the ini files "C \ Program Files \ NEXCOM \ NexECMRtx \ tools". Please refer to the following illustration.



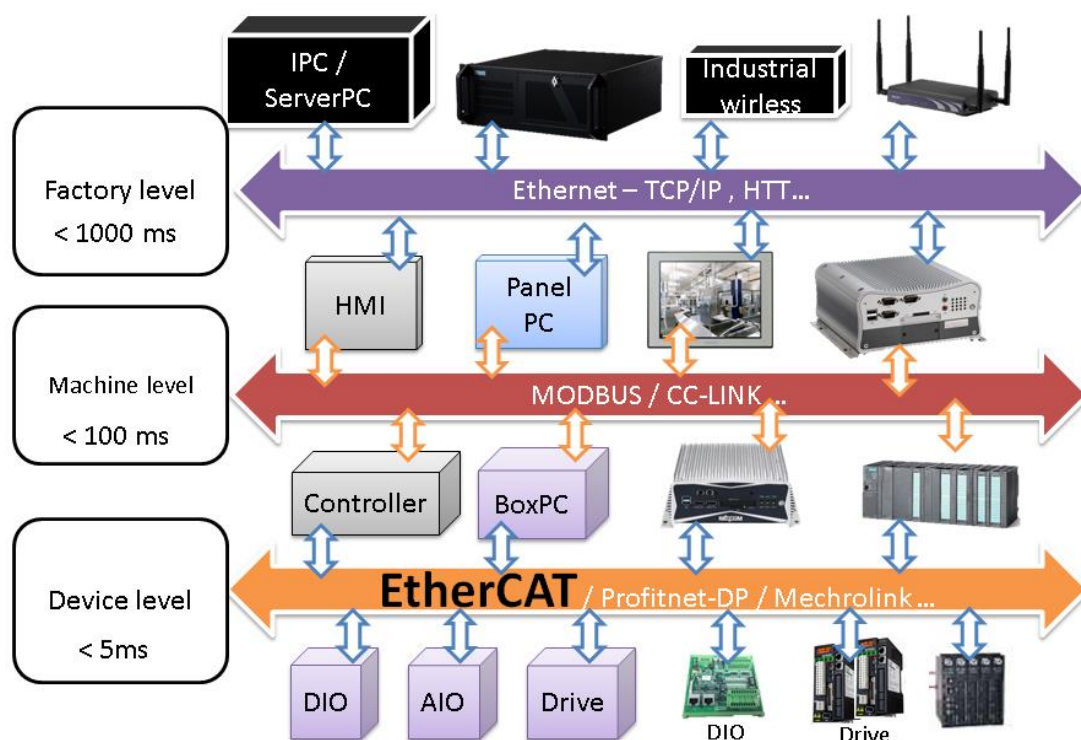
NexECMRtxConfig.ini content

	Description
PATH_ENI	
PATH:	Network Information File (ENI) path
OPTION:	Check the network interface card information by using ENI file
	0: Use ENI file
	1: No use ENI file, use Parameter setting
PATH_NEXECMRtx_DRIVER	
PATH:	NexECMRtx.rtss file ptah
PATH_USER_APP (Option)	
PATH:	Fill your RTX application (*.rtss) path and file name

4 EtherCAT Introduction

With the significant progress of communication technology, the Fieldbus 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.

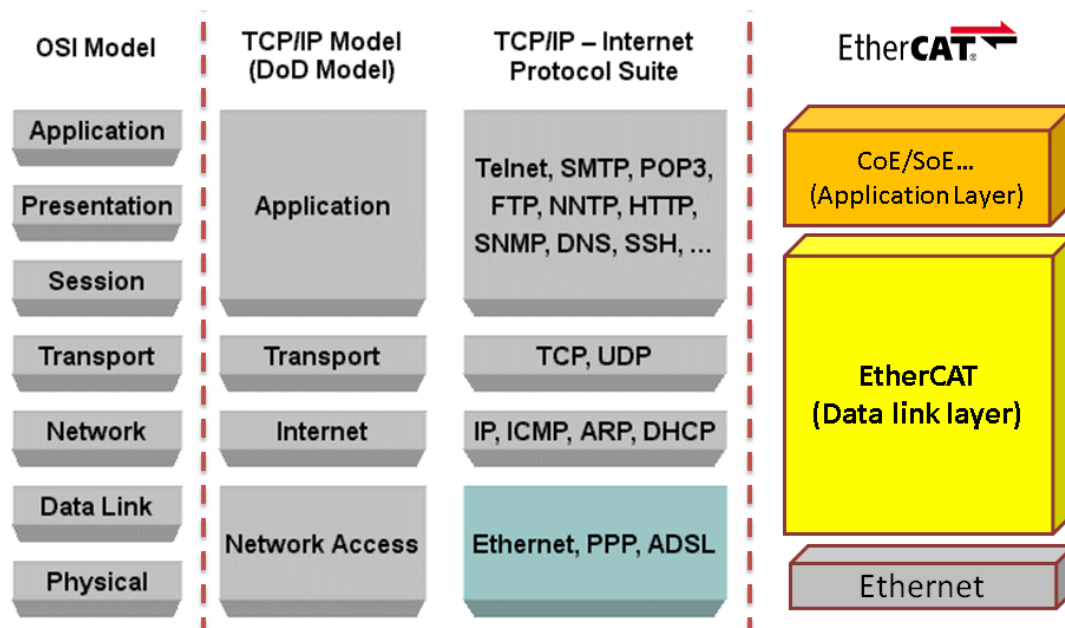
EtherCAT is designed on the base of Ethernet and is an industrial communication technology specifically designed for automation (especially machine automation). EtherCAT has the benefits of Ethernet (such as universality, high speed and low-cost) and its related products are developing in amazing speed. 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.

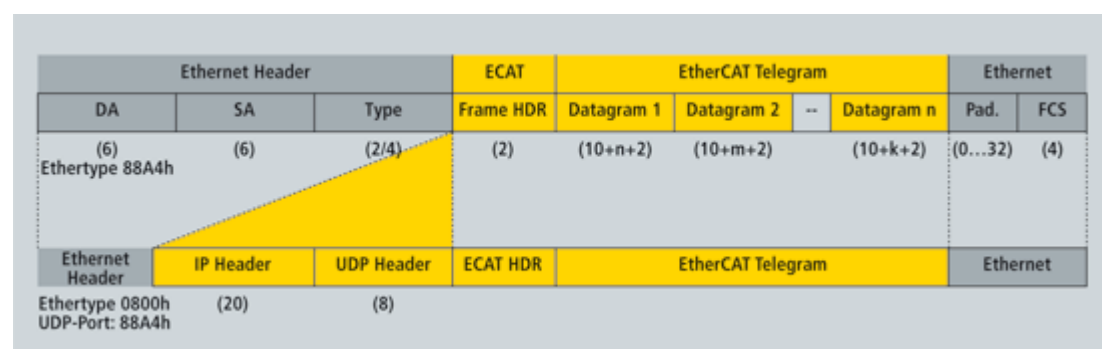
4.1 EtherCAT communication Protocol

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.

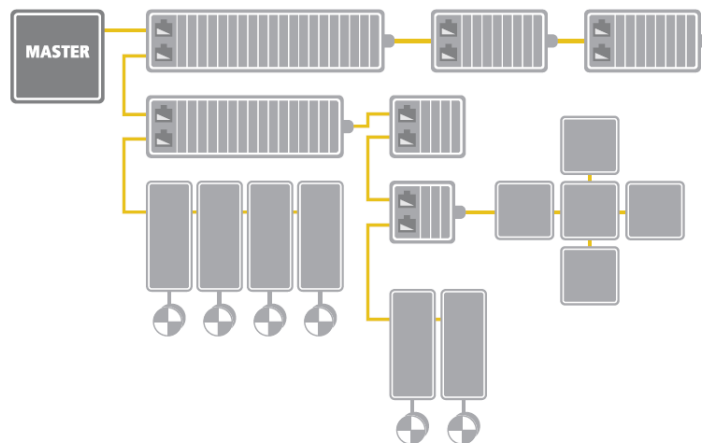


EtherCAT internet packet format

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

4.2 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.



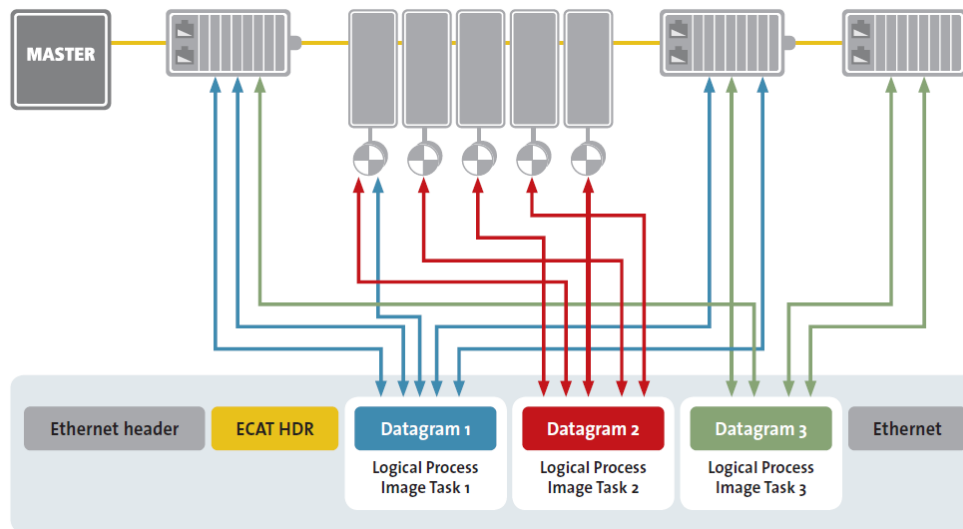
EtherCAT Network Topology, Line, Tree and Star Topology
(Source of information: <http://www.ethercat.org/>)



Using standard Ethernet network cable

4.3 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)

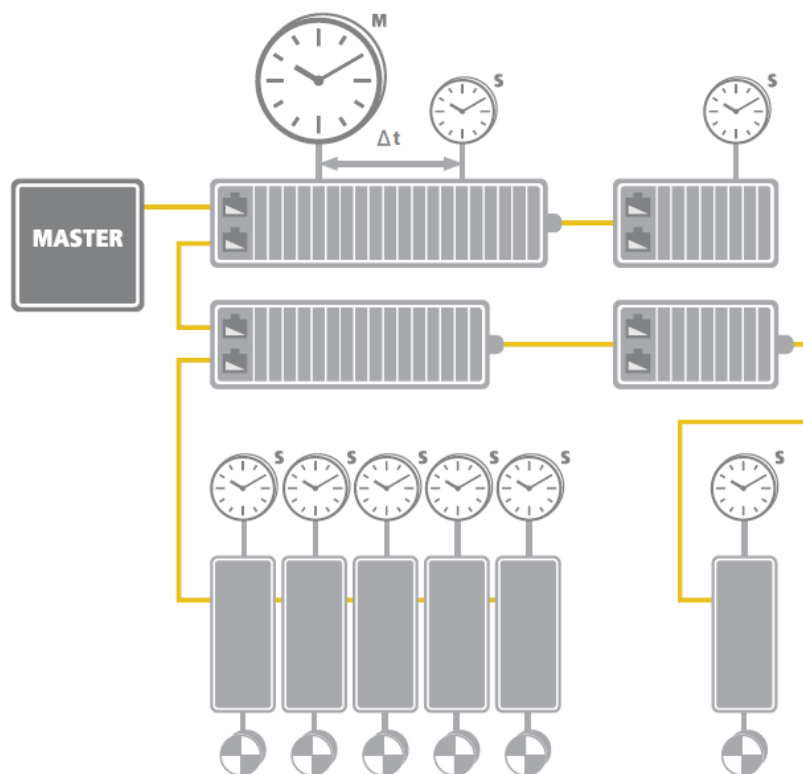
Process Data	Update Time
256 distributed digital I/O	11μs
1000 distributed digital I/O	30μs
200 channels analog I/O (16 Bits)	50μs (=20kHz)
100 servo axis (8 Bytes Input and output per axis)	100μs
1 Fieldbus Master-Gateway (1486 Bytes Input and 1486 Bytes output data)	150μs

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.

4.4 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.



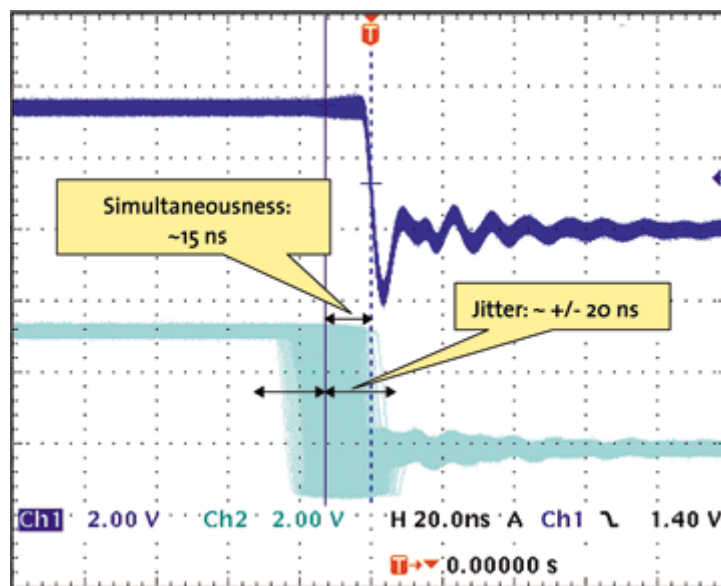
EtherCAT: Illustration of Distributed Clock (DC)

(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 output 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

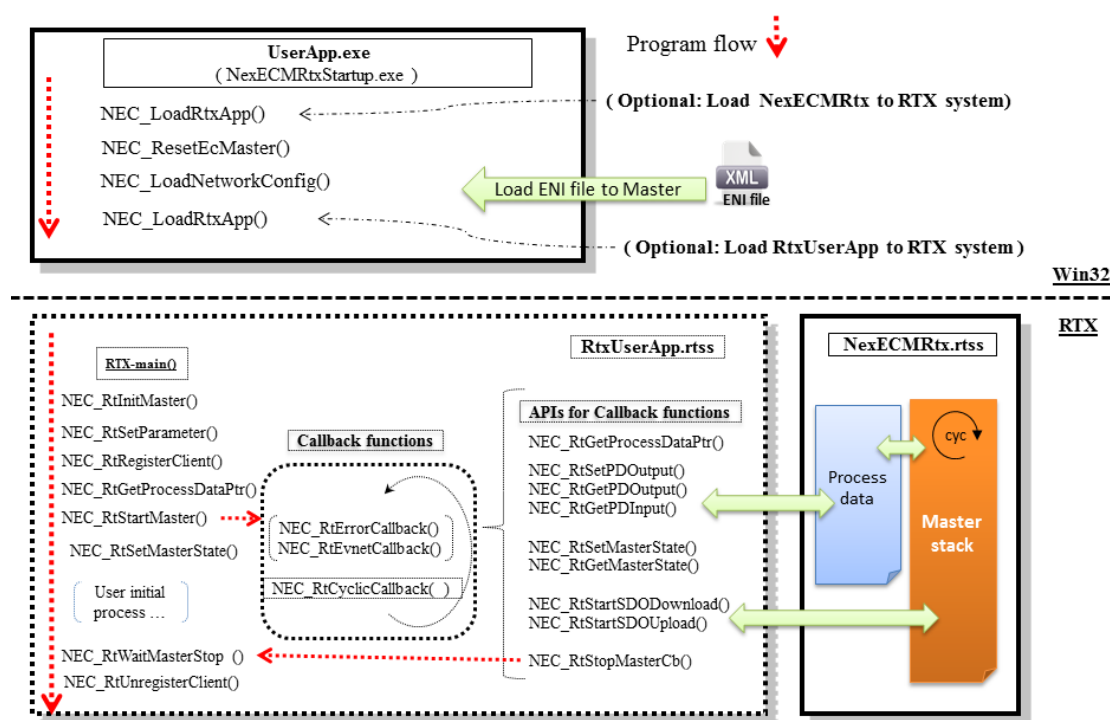


Synchronicity and Simultaneousness: Scope view of two distributed devices with 300 nodes and 120 m of cable between them
(Source of information: <http://www.ethercat.org/>)

5 Principles of Programming

5.1 Basic Programming Framework

Below explains the NexECMRtx basic programming framework



Basic Flow:

Steps	Description	Related API
Your Windows Application: Ex. <u>UserApp.exe</u> (Details API Description please refer to CH.7)		
1	Load NexECMRtx runtime stack	NEC_LoadRtxApp()
2	Reset NexECMRtx runtime stack	NEC_ResetEcMaster()
3	Load ENI	NEC_LoadNetworkConfig()
4	Load your RTX application (ex: RtxUserApp.rtss)	NEC_LoadRtxApp()
Your RTX Application main(): Ex. <u>RtxUserApp.rtss</u> (Details API Description please refer to CH.6)		
5	Initial EC-Master	NEC_RtInitMaster()
6	Set Parameters (ex: cycle-time...etc)	NEC_RtSetParameter() NEC_RtGetParameter()
7	Registerer Callback function function	NEC_RtregistererClient() NEC_RtUnregistererClient()

8	Get ProcessImage Memory pointer	NEC_RtGetProcessdataPtr() NEC_RtGetSlaveProcessdataPtr()
9	Start EC-Master communication	NEC_RtStartMaster()
10	Switch EC-Master to "OP" state	NEC_RtSetMasterstate()
11	Hang the main() thread. RTX is a main() type application, when all process is done, it will be unloaded. So, the main() thread need be hanged until user's procedure(maybe in callback function) is done.	NEC_RtWaitMasterStop()
In the period of Callback function (Details API Description please refer to CH.6)		
12	Your control program 1. ProcessData access 2. CoE communication	NEC_RtSetPDOOutput() NEC_RtGetPDOOutput() NEC_RtGetPDInput() NEC_RtStartSDODownload() NEC_RtStartSDOUpload()
13	End the application, stop EC-Master communication	NEC_RtStopMasterCb()

The above process, you can refer to NexECMRtx sample program in installation directory.

5.2 ENI Load

ENI (EtherCAT Network Information) must be loaded from your Win32 application. Before you load the ENI, the NexECMRtx.rtss must be load first. There are two ways for loading NexECMRtx.rtss:

1. Manual Load: Double click mouse left button on "NexECMRtx.rtss"
2. Call *NEC_LoadRtxApp()* API to Load

After strat NexECNRtx.rtss, call *NEC_LoadNetworkConfig()* to download the ENI file to the EC-Master. Then, when start the communication, EC-Master will use those data to initial EtherCAT slave devices online. If EC-Master detects the difference between ENI and current network topology, it will stop communication and trigger an error callback event (If you have registred an error callback function, it will be called).

You can use NexCAT utility to configure parameters for EC-Master and EC-Slave and reproduce the ENI file, for detailed operation, please refer to Chapter 3.

After loading ENI file to EC-Master, you can start your RTX Application and does communication control and operation in NexECMRtx, please refer to below sections.

“*NexECMRtxStartup.exe*” is an assistant tool for doing the above steps. You can find it at %InstallPath%/NEXCOM/NexECMRtx/tools/x32/NexECMRtxStartup.exe.

The tool acquires the files location from NexECMRtxConfig.ini. You also can find it at the path mentioned above. Then, edit the paths below by your files location.

1. ENI file path
2. NexECMRtx path
3. Your RTX application path

Details set functions, please refer to CH 3.2

Finally, run the NexECMRtxStartup.exe. ENI will be loaded automatically and your RTX application will be started.

5.3 Initialization of NexECMRtx

In your RTX application, call *NEC_RtInitMaster()* API to Initiate EC-Master. This action will not reset your ENI data from EC-Master.

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

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, by EC-Master Call
NEC_RtErrorCallback	When an error event occurs , by EC-Master Call

These Callback functions will be used to synchronize and exchange ProcessData(PDO)

data with EC-Master. When EC-Masrter's state^(*) change to "SAFEOP" or "OP" , EC-Master will start ProcessData(PDO) data exchange with EC-Slave at each cyclic communication.

(*)About the EtherCAT state machine, please refer to CH 5.6

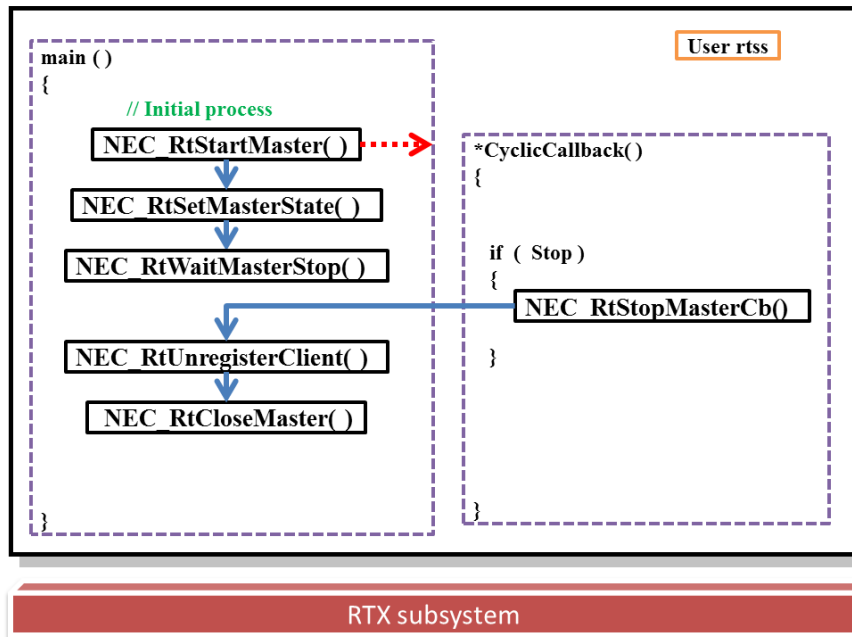
The most of users' application procedure will be put in the Callback functions. The basic programming principles of Callback function are to prevent from stoping or long executing time exceeded communication cycle. So, time-consuming API should be prohibited in Callback function. To know more about Callback function, please refer to Section 5.5.

5.4 Start Master communication

According to previous section, all settings for starting EC-Master should be finished, then start the EtherCAT 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 EC-Master. Please note that when the EC-Master communication starts, EC-Master 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, RTX application is a console type program which there is a *main()* entry in. When the *main()* returns, the program ends and unloaded by RTX 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 CH 5.5

About EtherCAT state machine, please refer to CH5.6

5.5 Callback Functions

5.5.1 Register Callback function

NexECMRtx offers three Callback functions:

1. Cyclic callback function
2. Event Callback function
3. Error Callback function

Before starting EC-Master communication, use `NEC_RtregistererClient()` to register Callback function in EC-Master. After communication is finished, use `NEC_RtUnregistererClient()` to release callback function. Please do not use `RtUnregistererClient()` to release callback function during communication.

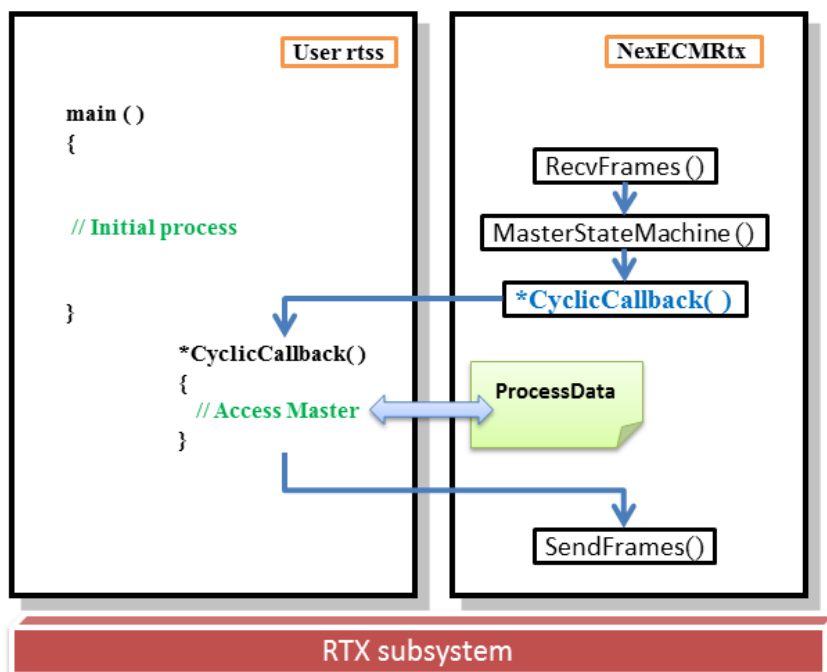
5.5.2 Cyclic Callback Function

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 EC-Master. The cycle

processes of NexECMRtx 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 have to 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 EC-Master's cycle communication program.

NexECMRtx offers synchronous callback function, making your Real-time applications and the ProcessData access in NexECMRtx 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 relationship between NexECMRtx: EC-Master data processing program with the user rtss flow chart provided by Callback function.



5.5.3 Event Callback Function

When the EC-Master predefined events happen, EC-Master 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.

5.5.4 Error Callback Function

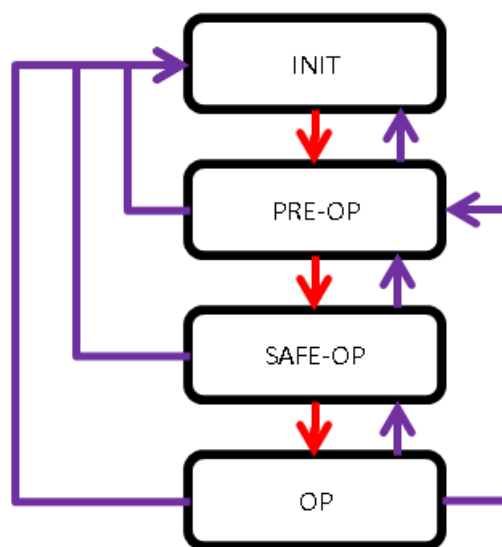
When the EC-Master predefined errors happen, EC-Master will inform all users to use this function and pass the error code (Error code) synchronized. The user can handle the corresponding error according to the error code, or ignore the processing.

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

5.6 EtherCAT state Machine

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

1. INIT state
2. PREOP state
3. SAFEOP state
4. 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 NexCAT utility.

5.6.1 ESM state Change

Start EC-Master 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 it does not check if the state of EC-Master is changed successfully or not. 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 EC-Master and EC-Slave.

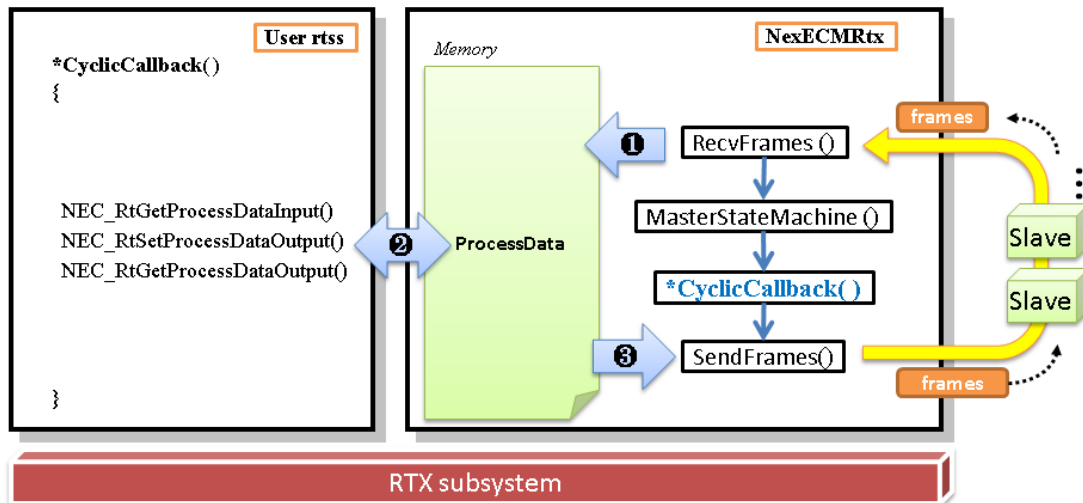
state	EC-Master service	EC-Slave service
INIT	<ul style="list-style-type: none"> ➤ Cyclic callback function start ➤ No ProcessData communication (To slaves) ➤ No Mailbox communication (To slaves) 	<ul style="list-style-type: none"> ➤ EC-Master not ready
PREOP	<ul style="list-style-type: none"> ➤ Cyclic callback function start ➤ Mailbox communication start (To slaves) -SDO communication start ➤ No ProcessData communication (To slaves) 	<ul style="list-style-type: none"> ➤ Mailbox communication start
SAFEOP	<ul style="list-style-type: none"> ➤ Cyclic callback function start ➤ Mailbox communication start (To slaves) -SDO communication start ➤ ProcessData Input communication start ➤ No ProcessData Output communication 	<ul style="list-style-type: none"> ➤ Mailbox communication start ➤ Update the Input data to ProcessData Input
OP	<ul style="list-style-type: none"> ➤ Cyclic callback function start ➤ Mailbox communication start (To slaves) -SDO communication start ➤ ProcessData Input communication start ➤ ProcessData Output communication start 	<ul style="list-style-type: none"> ➤ Mailbox communication start ➤ Update the Input data to ProcessData Input ➤ Get ProcessdataOutpt, Output data and transfer

5.7 Process Data Access

5.7.1 ProcessData Operation mechanism

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

1. Receive EC-Slaves data, write to ProcessData memory
2. EthreCAT state machine process
3. Execute CyclicCallback function
4. Send ProcessData to EC-Slaves



User application can get the internal memory pointer by using *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();

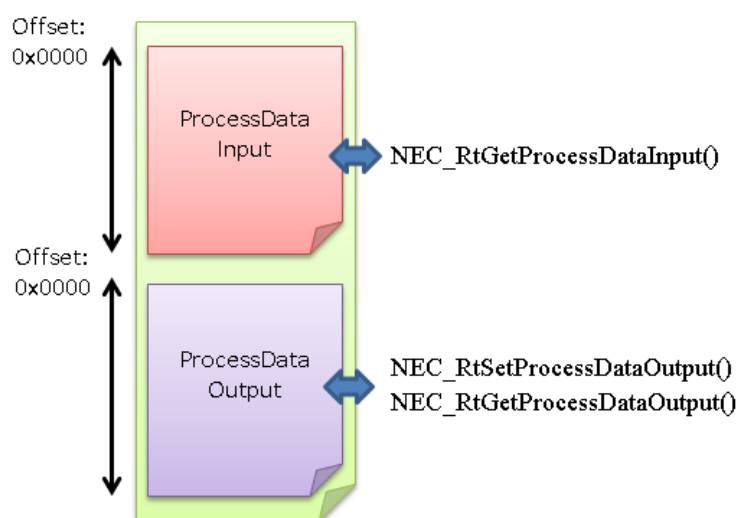
```

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 regional and ProcessDataOutput Internally, its transfer data direction are listed in the following table:

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

The related API for accessing ProcessData:

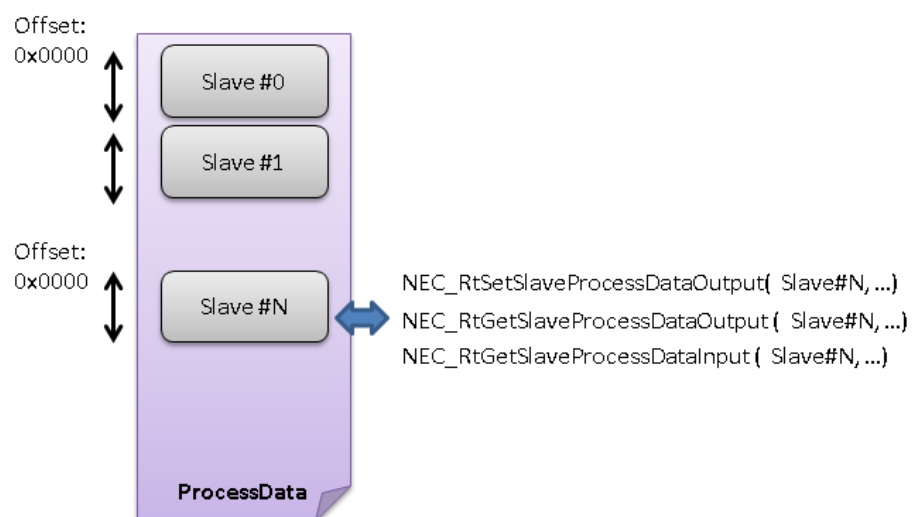


5.7.2 ProcessData Data Content

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 ()
```


5.8 Mailbox communication

5.8.1 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:

1. PDO (Process data object), for real-time data communication
2. 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.

According to CANOpen SDO has following types:

1. SDO download: data download from Master to Slave
2. SDO upload: data upload from Slave to Master
3. SDO information: Get object dictionary list

NexECMRtx 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 5.7.

5.8.2 CoE -Emergency communication

When error occurs in the EC-Slave, emergency message is produced in EC-Slave and delivered to EC-Master from EC-Slaves through CoE communication. EC-Master receives this message and stores it to internal message queue. Then, EC-Master calls the event callback if user registered it to the EC-Master. The rules of Master calling the event callback as below:

1. When EC-Master adds a new emergency message to the message queue, EC-Master will call the event callback.
2. At one cycle time, EC-Master calls event callback only once, even more than one emergency message added to the queue in the same time.
3. 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.

NexECMRtx offer following APIs for SDO communication.

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 6.7.

6 NexECMRtx Library

6.1 RTX API Overview

This is all NexECMRtx Libraries API list, APIs are defined at NexECMRtx.h.

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	Get NexECMRtx Current version information	B
NEC_RtRetVer	Return NexECMRtx Current version information	B
NEC_RtInitMaster	Initial NexECMRtx	X
NEC_RtCloseMaster	Close NexECMRtx (EC-Master)	X
NEC_RtSetParameter	Set EC-Master Parameters	X
NEC_RtGetParameter	Get EC-Master Parameters	X
NEC_RtregistererClient	register Callback clients function	X
NEC_RtUnregistererClient	release Callback client register	X
NEC_RtGetProcessdataPtr	Get the memory pointer point to ProcessData in EC-Mater	B
NEC_RtSetProcessdataPtrSource	Set ProcessData Memory source	X
EC-Master control Related APIs		
NEC_RtStartMaster	Start EC-Master communication	X
NEC_RtStopMaster	Stop EC-Master cycle communication	X
NEC_RtStopMasterCb	Send EC-Master communication stop request	C
NEC_RtWaitMasterStop	Wait EC-Master communication stop request & Stop communication	X
NEC_RtGetMasterstate	Get EC-Master Current state	B
NEC_RtSetMasterstate	Send EC-Master change state request	B
NEC_RtChangestateToOP	Blocking change EC-Master to "OP" state	X
NEC_RtSetMasterstateWait	Blocking change EC-Master state	X
NEC_RtGetstateError	Get state Error Code	B
ProcessData Access Related APIs		

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	
NEC_RtGetSlaveProcessdataOutput	Get EC-Slave ProcessDataOutput data	
NEC_RtGetSlaveProcessdataInput	Get EC-Slave ProcessDataInput data	
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
CoE Communication Related APIs		
NEC_RtStartSDODownload	Send SDO download Command data(Master to slave)	B
NEC_RtStartSDOUpload	Send SDO upload Command 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
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

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

6.2 Initial Related APIs

6.2.1 NEC_RtGetVersion

Return NexECMRtx version

C/C++ Syntax :

```
RTN_ERR NEC_RtGetVersion( U32_T *Version );
```

Parameters:

U32_T *Version:

Return the version of NexECMRtx.

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()

6.2.2 NEC_RtRetVer

Return NexECMRtx current version

C/C++Syntax :

```
U32_T NEC_RtRetVer();
```

Parameters:

<No Parameters>

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 API fails, 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_RtGetVersion()

6.2.3 NEC_RtInitMaster

Initialize NexECMRtx

C/C++Syntax :

```
RTN_ERR NEC_RtInitMaster( U16_T MasterId );
```

Parameters:

U16_T MasterId:

Assign target EC-Master ID, single EC-Master, please 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 NexECMRtx APIs.

Note! Never call this API in Callback functions.

Reference:

NEC_RtCloseMaster()

6.2.4 NEC_RtCloseMaster

Close NexECMRtx (EC-Master)

C/C++Syntax :

```
RTN_ERR NEC_RtCloseMaster( U16_T MasterId );
```

Parameters:

U16_T MasterId:

Assign target EC-Master ID, single EC-Master, please 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 NexECMRtx before program end.

Note! Never call this API in Callback functions.

Reference:

NEC_RtInitMaste()

6.2.5 NEC_RtSetParameter

6.2.6 NEC_RtGetParameter

These two APIs are used to set and get 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 target EC-Master ID, single EC-Master, please 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 EC-Master communication, set the parameters for EC-Master 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	EC-Master communication cycle Unit: micro-second	250 ~ 1000000
NEC_PARA_S_NIC_INDEX	Assign a NIC Port to EC-Master. This parameter could be modify when load ENI information. Please refer to CH7.4.5	0 ~ Max NIC port

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

Reference:

NEC_RtGetstateError(); NEC_RtStartMaster()

6.2.7 NEC_RtregistererClient

Register callback client functions.

C/C++Syntax :

```
RTN_ERR NEC_RtregistererClient( U16_T MasterId, TcIntParam *ClientParam );
```

Parameters:

U16_T MasterId:

Assign target EC-Master ID, single EC-Master, please set to 0

TcIntParam *ClientParam: Callback client data structure:

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

U32_T version:

Previous Set version, use *NEC_RtRetVer()*

void *userdataPtr:

Pass user-defined data structure pointer. EC-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 EC-Master, 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 EC-Master, 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 EC-Master, this function will be called when predefined error happens.

U16_T clientID:

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

(*) The operation theory about callback function please refer toCH 5.5

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 EC-Master communication. If you want to do a re-registration, you must stop the communication first and call *NEC_RtUnregistererClient()* to unregister existing callback client.

Note! Never call this API in Callback functions.

Reference:

NEC_RtStartMaster (); NEC_RtUnregistererClient();NEC_RtStopMaster();

6.2.8 NEC_RtUnregistererClient

Unregister callback client

C/C++Syntax :

```
RTN_ERR NEC_RtUnregistererClient( U16_T MasterId, TcIntParam *ClientParam );
```

Parameters:

U16_T MasterId:

Assign target EC-Master ID, single EC-Master, please set to 0

TcIntParam *ClientParam:

Reference *NEC_RtregistererClient()* 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_RtregistererClient()* to register callback functions. When end of the application call *NEC_RtUnregistererClient()* to unregister existing callback client. Generally please un-register client after *NEC_RtWaitMasterStop()*.

Note! Never call this API in Callback functions.

Reference:

NEC_RtregistererClient(); *NEC_RtWaitMasterStop()*

6.2.9 NEC_RtGetProcessdataPtr

Get EC-Mater 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 target EC-Master ID, single EC-Master, please 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 CH 5.5 callback function and Ch5.7 Process data access.

Reference:

```
NEC_RtSetSlaveProcessdataOutput();NEC_RtGetSlaveProcessdataOutput();
NEC_RtGetSlaveProcessdataInput ();NEC_RtSetProcessdataPtrSource()
```

6.2.10 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 target EC-Master ID, single EC-Master, please set to 0

void *InputProcessdataPtrSource:

Set ProcessDataInput memory from outside, if set to 0, mean use internal 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 EC-Master 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 EC-Master Stop communication, User must call *NEC_RtSetProcessdataPtrSource()* to assign the memory source back to internal memory before **RTX application unloaded**.

To ensure data consistency, to access the ProcessData during callback functions is recommend. Related information please refers to CH 5.5 callback function and Ch5.7 Process data access.

Reference:

```
NEC_RtSetSlaveProcessdataOutput();NEC_RtGetSlaveProcessdataOutput();  
NEC_RtGetSlaveProcessdataInput (); NEC_RtGetProcessdataPtr();
```

6.3 EC-Master Control Related APIs

6.3.1 NEC_RtStartMaster

Start EC-Master communication.

C/C++Syntax :

```
RTN_ERR NEC_RtStartMaster( U16_T MasterId );
```

Parameters:

U16_T MasterId:

Assign target EC-Master ID, single EC-Master, please 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 configure the parameters of EC-Master by using *NEC_RtSetParameter()*, Using this API to start the communication. After master started, a cyclic process will be active and the callback function which registered by *NEC_RtregistererClient()* will be executed periodically. The "state" of the EC-Master will stay at "INIT" state before you change it.

Two ways for stopping communication:

1. Call *NEC_RtStopMasterCb()* in Callback function, or
2. Use *NEC_RtStopMaster()*

Note! Never call this API in Callback functions.

Reference:

```
NEC_RtSetParameter();NEC_RtregistererClient();NEC_RtStopMaster();
NEC_RtStopMasterCb();*NEC_RtCyclicCallback()
```

6.3.2 NEC_RtStopMaster

Stop EC-Mastercycle communication.

C/C++Syntax :

```
RTN_ERR NEC_RtStopMaster( U16_T MasterId );
```

Parameters:

U16_T MasterId:

Assign target EC-Master ID, single EC-Master, please 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 used to stop EC-Master cycle communication. You should call *NEC_RtSetMasterstate()* to change EC-Master's state to "INIT" state before stop EC-Master communication.

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

Note! Never call this API in Callback functions.

Reference:

NEC_RtStartMaster();NEC_RtStopMasterCb();NEC_RtSetMasterstate()

6.3.3 NEC_RtStopMasterCb

Send EC-Master communication stop request, used in callback function.

C/C++Syntax :

```
RTN_ERR NEC_RtStopMasterCb( U16_T MasterId );
```

Parameters:

U16_T MasterId:

Assign target EC-Master ID, single EC-Master, please 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 EC-Master 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 will change to "INIT" state.

Reference:

NEC_RtStartMaster();NEC_RtStopMasterCb();NEC_RtWaitMasterStop()

6.3.4 NEC_RtWaitMasterStop

Wait for EC-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 target EC-Master ID, single EC-Master, please 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:

1. Let main program goes to sleep, this API will be blocked.
2. After receiving a stop signal, the EC-Master state machine is switched to "INIT" state and this API will wakeup and return.

Under most circumstances, the user will put the control procedures into the cyclic callback function. However, an RTX 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 a stop signal triggered by *NEC_RtStopMasterCb()* in callback function.

Note! Never call this API in callback function.

Reference:

NEC_RtStopMasterCb()

6.3.5 NEC_RtGetMasterstate

6.3.6 NEC_RtSetMasterstate

NEC_RtGetMasterstate() : Get EC-Master current state (state machine)

NEC_RtSetMasterstate() : Send EC-Master 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 target EC-Master IO, single EC-Master, please 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 EC-Master communication, use *NEC_RtSetMasterstate()* to switch EC-Master state. *NEC_RtSetMasterstate()* can be used in callback function, it is just send a state change request, it will not wait EC-Master state change successful or not. Using *NEC_RtGetMasterstate()* API to check state transition is needed.

Another way is use block API to switch EC-Master state, refers to Reference *NEC_RtSetMasterstateWait()* and *NEC_RtChangestateToOP()*

More about EC-Master state machine description please refer to CH 5.6 EtherCAT state machine.

Reference:

NEC_RtStartMaster(); *NEC_RtSetMasterstateWait()*, *NEC_RtChangestateToOP()*

6.3.7 NEC_RtChangestateToOP

This is a blocked type API. To switch EC-Master's state to "OP" state

C/C++Syntax :

```
RTN_ERR NEC_RtChangestateToOP ( U16_T MasterId, I32_T TimeoutMs );
```

Parameters:

U16_T MasterId:

Assign target EC-Master ID, single EC-Master, please 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 the start communication to change the EC-Master 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 EC-Masterstate machine, please refer to CH 5.6 EtherCAT state machine.

Reference:

NEC_RtStartMaster(); *NEC_RtSetMasterstate ()*; *RtSetMasterstateWait()*

6.3.8 NEC_RtSetMasterstateWait

Blocking change EC-Master state

C/C++Syntax :

```
RTN_ERR NEC_RtSetMasterstateWait( U16_T MasterId, U16_T state, I32_T
TimeoutMs );
```

Parameters:

U16_T MasterId:

Assign target EC-Master ID, single EC-Master, please 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 EC-Master state. Usage is similar to *NEC_RtSetMasterstate()* after the start EC-Master communication to change the EC-Master target state. Major difference between two functions is that this API can be set timeout time. When API success returns mean the master's state has success changed to target state.

Note! Never call this API in Callback function.

About EC-Masterstate machine, please refer to CH 5.6 EtherCAT state machine.

Reference:

`NEC_RtStartMaster();NEC_RtSetMasterstate ();NEC_RtChangestateToOP (),`

6.3.9 NEC_RtGetstateError

Get state error code

C/C++Syntax:

```
RTN_ERR NEC_RtGetstateError( U16_T MasterId, I32_T *Code );
```

Parameters:

U16_T MasterId:

Assign target EC-Master ID, single EC-Master, please 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 EC-Master state abnormal (ex: Link broken), state will change to ECM_STA_ERROR state, EC-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()

6.4 ProcessData Access Related APIs

6.4.1 NEC_RtSetProcessdataOutput

6.4.2 NEC_RtGetProcessdataOutput

6.4.3 NEC_RtGetProcessdataInput

Access 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 );
```

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

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

Parameters:

U16_T MasterId:

Assign target EC-Master ID, single EC-Master, please 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

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 Ch5.7 Process Data Access.

Reference:

```
NEC_RtGetProcessdataPtr(); NEC_RtSetSlaveProcessdataOutput();  
NEC_RtGetSlaveProcessdataOutput(); NEC_RtGetSlaveProcessdataInput()
```

6.4.4 NEC_RtSetSlaveProcessdataOutput

6.4.5 NEC_RtGetSlaveProcessdataOutput

6.4.6 NEC_RtGetSlaveProcessdataInput

Read or Write EC-Slave ProcessData data.

C/C++Syntax :

```
RTN_ERR NEC_RtSetSlaveProcessdataOutput( U16_T MasterId, U16_T SlaveAddr,
U16_T Offset, U8_T *data, U32_T Size );
```

```
RTN_ERR NEC_RtGetSlaveProcessdataOutput( U16_T MasterId, U16_T SlaveAddr,
U16_T Offset, U8_T *data, U32_T Size );
```

```
RTN_ERR NEC_RtGetSlaveProcessdataInput( U16_T MasterId, U16_T SlaveAddr,
U16_T Offset, U8_T *data, U32_T Size );
```

Parameters:

U16_T MasterId:

Assign target EC-Master ID, single EC-Master, please 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 Ch5.7 Process Data Access.

Reference:

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

6.5 Callback APIs

6.5.1 *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_RtregistererClient()* to register this callback function to EC-Master. Then this function will be called cyclically and pass pointer which point to user data.

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

(*) More about Callback library please refer to CH 5.5 Callback functions.

Reference:

NEC_RtregistererClient(); *NEC_RtUnregistererClient()*;

6.5.2 *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_RtregistererClient()* to register callback function to EC-Master 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-Masterstate Change Event
EVENT_ECM_CNECT_FINISH	All EC-Slaves's state change to "Operational" state.

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

(*) More about Callback library please refer to CH 5.5 Callback functions.

Reference:

NEC_RtregistererClient();NEC_RtUnregistererClient();

6.5.3 *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_RtregistererClient()* to register callback function to EC-Master before start communication. Error callback function will be triggered if predefined error event happens. Users can write event handler in callback function.

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

(*) More about Callback library please refer to CH 5.5 Callback functions.

Reference:

NEC_RtregistererClient(); *NEC_RtUnregistererClient()*;

6.6 ENI Related APIs

6.6.1 NEC_RtGetSlaveCount

Get EC-Slaves numbers

C/C++Syntax :

```
RTN_ERR NEC_RtGetSlaveCount(U16_T MasterId, U16_T *Count );
```

Parameters:

U16_T MasterId:

Assign target EC-Master ID, single EC-Master, please 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 EC-Master, use this API to get EC-Slave numbers. This information come from ENI file.

Reference:

<No Reference>

6.6.2 NEC_RtGetSlaveInfomation

Get the EC-Slave information

C/C++Syntax :

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

Parameters:

U16_T MasterId:

Assign target EC-Master ID, single EC-Master, please 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 EC-Master, use this API to get EC-Slave's information. This information comes from ENI file (Except DL_status).

Reference:

<No Reference>

6.6.3 NEC_RtGetSlaveState

Get state of EC-Slaves

C/C++Syntax :

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

Parameters:

U16_T MasterId:

Assign target EC-Master ID, single EC-Master, please 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 StateArr

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>

6.7 CoE Communication Related APIs

6.7.1 NEC_RtStartSDODownload

6.7.2 NEC_RtStartSDOUpload

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 target EC-Master ID, single EC-Master, please 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 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()

6.7.3 NEC_RtSDODownload

6.7.4 NEC_RtSDOUpload

6.7.5 NEC_RtSDODownloadEx

6.7.6 NEC_RtSDOUploadEx

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 target EC-Master ID, single EC-Master, please 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 Downloader 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()

6.7.7 NEC_RtStartGetODListCount

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_RtStartGetODListCount( U16_T MasterId, U16_T SlaveAddr,
TCOEODListCount *pCoeOdListCount );
```

Parameters:

U16_T MasterId:

Assign target EC-Master ID, single EC-Master, please 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 the number of object dictionary with are available for the different CoE list types 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 "TCoEODListCount.*status* equal not to *SDO_INIT*".

Reference:

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

6.7.8 NEC_RtStartGetODList

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_RtStartGetODList( U16_T MasterId, U16_T SlaveAddr,
TCOEODList *pCoeOdList );
```

Parameters:

U16_T MasterId:

Assign target EC-Master ID, single EC-Master, please 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 which belong to this CoE list type assigned 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 "TCoEODList.status equal not to *SDO_INIT*".

Reference:

```
NEC_RtStartGetODListCount();NEC_RtStartGetObjDesc();NEC_RtStartGetEntryDesc();  
NEC_RtGetEmgDataCount();NEC_RtGetEmgData();
```

6.7.9 NEC_RtStartGetObjDesc

This function returns an object description of index assigned 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 target EC-Master ID, single EC-Master, please 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();
```

6.7.10 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 target EC-Master ID, single EC-Master, please 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();
```

6.7.11 NEC_RtGetEmgDataCount

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

C/C++Syntax :

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

Parameters:

U16_T MasterId:

Assign target EC-Master ID, single EC-Master, please 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 EC-Master. This API can be used in callback function.

Reference:

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

6.7.12 NEC_RtGetEmgData

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

C/C++Syntax :

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

Parameters:

U16_T MasterId:

Assign target EC-Master ID, single EC-Master, please 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 EC-Master. This API can be used in callback function.

Reference:

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

6.8 Slave Hardware Information Access APIs

6.8.1 NEC_RtGetConfiguredAddress

Get slave's "Configured Station Address"

C/C++Syntax :

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

Parameters:

U16_T MasterId: Assign target EC-Master ID, single EC-Master, please 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 address 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>

6.8.2 NEC_RtGetAliasAddress

Get slave's "Configured Station Alias"

C/C++Syntax :

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

Parameters:

U16_T MasterId: Assign target EC-Master ID, single EC-Master, please 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>

6.8.3 NEC_RtGetSlaveCoeProfileNum

Get slave's "CoeProfileNum"

C/C++Syntax :

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

Parameters:

U16_T MasterId: Assign target EC-Master ID, single EC-Master, please 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 );
```

```
    }
```

```
}
```

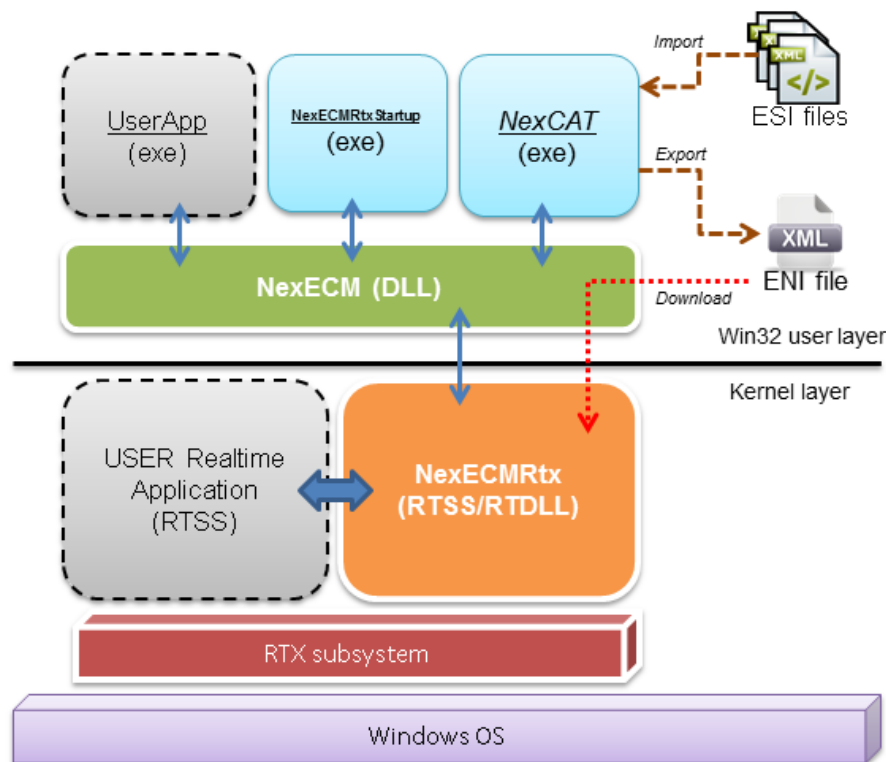


Reference:

<No Reference>

7 NexECM Library (Win32 APIs)

Win32 user application (as shown in UserApp.exe) can be linked through NexECM Dynamic Library (Win32 Dynamically linked library, DLL), to control NexECMRtx runtime. Through NexECM Win32 library users can very quickly integrate Win32 applications and NexECMRtx RTX applications.



NexECM Libraries main functions:

1. Load / Remove RTSS application to / from RTX sub-system.
2. Load EtherCAT Network Information(ENI) to EC-Master
3. ProcessData access (Through NexECMRtx)
4. CoE SDO access (Through NexECMRtx)
5. CiA402 APIs, refer to CiA402 APIs user manual

7.1 Win32 API Summary

This table lists all NexECM Libraries API. All API are defined at NexECM.h.

Library Name	Description	
RTX System Control APIs		
NEC_LoadRtxApp	Load a RTSS Application	
NEC_KillRtxApp	Remove RTSS Application	
NexECM initial related APIs		
NEC_GetVersion	Get NexECM Libraries (NexECM.dll) version	
NEC_StartDriver	Initial NexECM Libraries	
NEC_StopDriver	Close NexECM Libraries	
NexECMRtx Runtime Control Related APIs		
NEC_GetRtMasterId	Auto detect NexECMRtx and Get NexECMRtx Code (MasterID)	
NEC_GetRtMasterVersion	Get NexECMRtx Version	
NEC_GetRtMasterPorcessId	Get NexECMRtx Process ID (RTX Process ID)	
NEC_ResetEcMaster	Reset EC-Master (NexECMRtx)	
NEC_LoadNetworkConfig	LoadENI File	
NEC_StartNetwork	Start EC-Master communication	
NEC_StartNetworkEx	Start EC-Master communication (with "Option")	
NEC_StopNetwork	Stop EC-Mastercommunication	
NEC_SetParameter	Set EC-Master's parameters	
NEC_GetParameter	Get EC-Master's parameter	
Network status APIs		
NEC_GetSlaveCount	Get counts of EC-Slaves	
NEC_GetNetworkState	Get network state of EC-Master	
NEC_GetSlaveState	Get network state of EC-Slave	
NEC_GetStateError	Get error code when EC-Master has state error	
NEC_GetErrorMsg	Get error message when EC-Master state error	
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	

CoE SDO communication APIs		
NEC_SDODownloadEx	Send a SDO download request (Master to slave)	
NEC_SDOUploadEx	Send a SDO upload request (Slave to Master)	
NEC_SDODownload	Execute a SDO download (Structure)	
NEC_SDOUpload	Execute a SDO upload (Structure)	
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 EC-Master's ProcessData	
NEC_GetProcessImageSize	Get length of EC-Master's ProcessData	
NEC_RWSlaveProcessImage	Access EC-Slave's ProcessData	
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

7.2 RTX System Control APIs

7.2.1 NEC_LoadRtxApp

Load a RTSS Application into RTX subsystem

C/C++Syntax :

```
RTN_ERR NEC_LoadRtxApp( const char* RtxFileName );
```

Parameters:

const char* RtxFileName:

RTSS File path C-style string

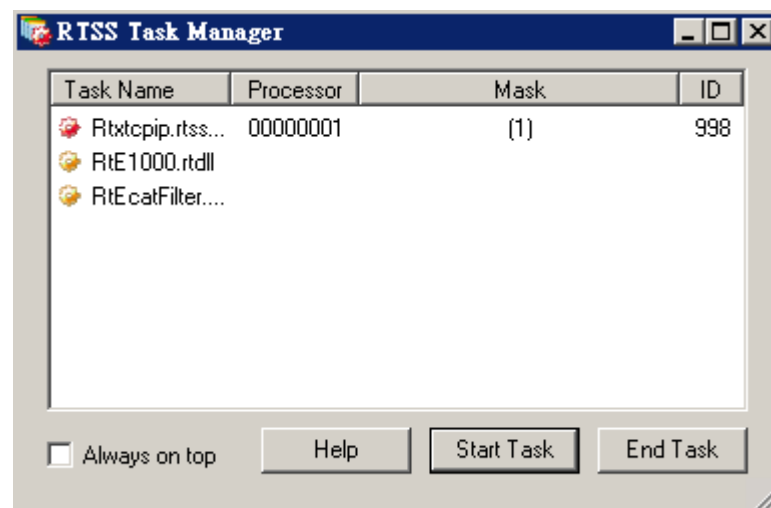
Return Value:

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.

Usage:

This API is used to load EC-Master runtime (NexECMRtx.rtss) or any RTSS application to RTX system. User can use RTSS Task Manager (RTX tool) to observe the program loaded.



Reference:

NEC_GetRtMasterId (); NEC_KillRtxApp (); NEC_GetRtMasterPorcessId()

7.2.2 NEC_KillRtxApp

Kill a RTSS Application

C/C++Syntax :

```
RTN_ERR NEC_KillRtxApp( U32_T ProcessId );
```

Parameters:

U32_T ProcessId:

RTSS's Process ID.

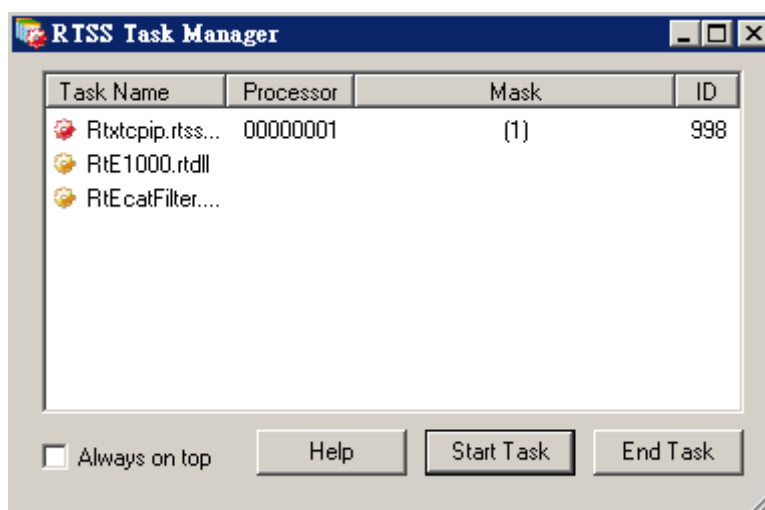
Return Value:

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 kill an RTSS process. With this API, EC-Master runtime (NexECMRtx.rtss) or any RTSS application can be forced removed. The ProcessID can be observed from RTX utility: "RTSS Task Manager" (see below). User can also use the utility to confirm whether the RTSS procedure is removed successfully. Using *NEC_GetRtMasterPorcessId()* to get the Process ID of NexECMRtx.



Force remove RTX application may result in the risk of system instability. It should be noted that the unstable situation may come from the order of removing application

using this API. For example, remove NexECMRtx when user's application is accessing it's memory.

NEC_KillRtxApp() may not release the system resources completely used by the application.

Reference:

`NEC_LoadRtxApp(); NEC_GetRtMasterPorcessId(); NEC_GetRtMasterId();`

7.3 NexECM initial related APIs

7.3.1 NEC_GetVersion

Get version of NexECM (NexECM.dll)

C/C++Syntax :

```
U32_T NEC_GetVersion();
```

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:

Return NexECM.dll's version.

Reference:

<No Reference>

7.3.2 NEC_StartDriver

Initial NexECMLibraries

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 APIs in NexECM library must after this API is called. This API has also detected whether the NexECMRtx is loaded. Load NexECMRtx before using this API. About load NexECMRtx please refer to *NEC_LoadRtxApp()*.

Reference:

NEC_LoadRtxApp();NEC_StopDriver();

7.3.3 NEC_StopDriver

Close NexECMLibraries

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 ()

7.4 NexECMRtx Runtime Control Related APIs

7.4.1 NEC_GetRtMasterId

To detect NexECMRtx is loaded or not and get it's control code (MasterID)

C/C++Syntax :

```
RTN_ERR NEC_GetRtMasterId( U16_T *pMasterId );
```

Parameters:

U16_T *pMasterId:

Return ID used to control NexECMRtx.

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 Win32 NexECM to access (control) the NexECMRtx, You must use this API to get the master's ID to reference the NexECMRtx. If this API returns error: "ECERR_DRIVER_NOT_FOUND". It means that no NexECMRtx is loaded in RTX system.

Reference:

NEC_LoadRtxApp(); NEC_KillRtxApp (); NEC_GetRtMasterPorcessId()

7.4.2 NEC_GetRtMasterVersion

Get NexECMRtx Version

C/C++Syntax :

```
RTN_ERR NEC_GetRtMasterVersion( U16_T MasterId, U32_T *pVersion );
```

Parameters:

U16_T MasterId:

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

U32_T *pVersion:

Return nexECMRtx Version

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:

Get NexECMRtx (RTX EtherCAT master) Version. Same as *NEC_RtGetVersion()*

Reference:

N/A

7.4.3 NEC_GetRtMasterPorcessId

Get NexECMRtx Process ID (RTX Process ID)

C/C++Syntax :

```
RTN_ERR NEC_GetRtMasterPorcessId( U16_T MasterId, U32_T *pProcessID );
```

Parameters:

U16_T MasterId:

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

U32_T *pProcessID:

Return RTX Process ID

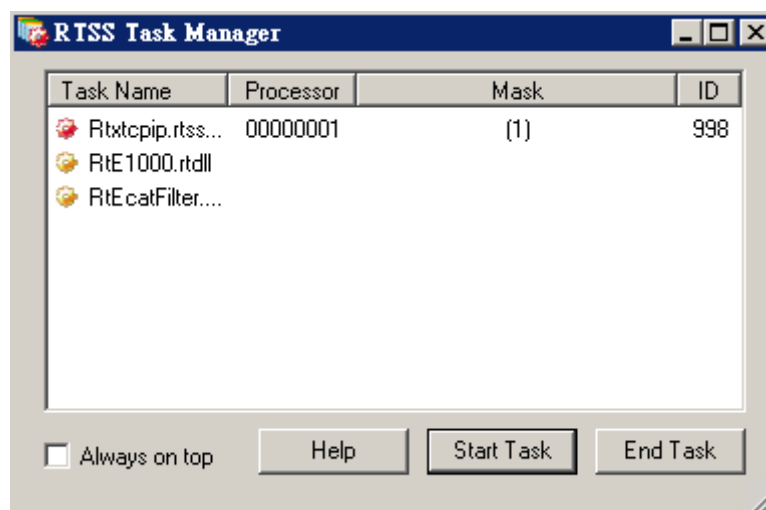
Return Value:

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:

RTX Process ID can be observed through the utility "RTSS Task Manager". Using this API can get NexECMRtx the ProcessID. Available for NEC_KillRtxApp().



RTSS Task Manager

Reference:

NEC_KillRtxApp();

7.4.4 NEC_ResetEcMaster

Reset EC-Master (NexECMRtx)

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 EC-Master(NexECMRtx). Usually be called after *NEC_GetRtMasterId()* and before Load ENI file *NEC_LoadNetworkConfig()*. Use this API the EC-Master internal variables were reset. If there has been previously downloaded ENI File, ENI data will be cleared from EC-Master.

Reference:

NEC_GetRtMasterId(); *NEC_LoadNetworkConfig()*

7.4.5 NEC_LoadNetworkConfig

Load EtherCAT Network Information (ENI) file to the 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 form *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 NexCAT utility (see Reference Ch3)

Reference:

NEC_ResetEcMaster()

7.4.6 NEC_StartNetwork

Start EC-Master 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 route 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 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 of EC-Master.

Reference:

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

7.4.7 NEC_StartNetworkEx

Start EC-Master 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 route 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 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

of EC-Master.

Reference:

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

7.4.8 NEC_StopNetwork

To stop EC-Master 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 EC-Mastercycle communication. During the process to stop the communication, EC-Master will be switched back to "INIT" state.

Reference:

```
NEC_GetRtMasterId();NEC_SetParameter();NEC_StartNetwork();  
NEC_StartNetworkEx()
```

7.4.9 NEC_SetParameter

7.4.10 NEC_GetParameter

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

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 EC-Master communication before EC-Master communication is started. Parameters are listed as below:

Parameter Code	Description	Parameter Value
NEC_PARA_S_ECM_CYCLETIMEUS	EC-Master 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:	LINKERR_AUTO (0) LINKERR_MANUAL

	<p>When EC-Slave(s) is disconnected, EC-Master detects the location of the disconnection automatically. When the device is connected again, EC-Slaves will be initialized and set to state "OP"</p> <p>LINKERR_MANUAL:</p> <p>When a device is disconnected, it will stay in state ERROR. When the device is connected again, it will not be initialized.</p> <p>LINKERR_STOP:</p> <p>As long as disconnection happens in any device, EC-Master stop the entire network and get into the state ERROR</p>	<p>(1)</p> <p>LINKERR_STOP (2)</p>
<p>NEC_PARA_ECM_DC_CYC_TIME_MOD</p> <p>E</p>	<p>DC Time setting model:</p> <p>0: According to Master cycle time (Default)</p> <p>1: According to ENI data</p>	<p>0~1</p>

Reference:

NEC_GetRtMasterId();

7.5 Network status APIs

7.5.1 NEC_GetSlaveCount

Get the number of EC-Slaves.

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 API 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 API to get the number of EC-Slaves after EC-Master communication is started. Basically, counts of slave is decided by ENI file.

Reference:

NEC_GetRtMasterId()

7.5.2 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:

- | | |
|-------------------|--|
| STATE_STOPPED | (0) : Networking is stopped. |
| STATE_OPERATIONAL | (1) : Networking is in operation state.(EtherCAT in |
| STATE_ERROR | (2) : Networking / slaves errors, and stopped. |
| STATE_SLAVE_RETRY | (3) : 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 EC-Master communication is started.

Reference:

```
NEC_GetRtMasterId();
```

7.5.3 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 EC-Master communication is started.

References:

NEC_GetRtMasterId();

7.5.4 NEC_GetstateError

To get EC-Master state error string

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 EC-Master state error code when EC-Master state is in "ERROR" state.

Reference:

NEC_GetRtMasterId(); NEC_GetErrorMsg()

7.5.5 NEC_GetErrorMsg

To get EC-Master state error string

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 EC-Master state error string when EC-Master state is in "ERROR" state.

References:

NEC_GetRtMasterId(); NEC_GetStateError()

7.6 Digital I/O control APIs

7.6.1 NEC_SetDo

To set the EC-Slave Digital 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 EC-Master communication is started. Please notice the length(SizeByte) of DoData must meet the DO size of target slave device.

Reference:

NEC_GetRtMasterId();

7.6.2 NEC_GetDo

To read the EC-Slave Digital 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 EC-Master communication is started. Please notice the length(SizeByte) of DoData must meet the DO size of target slave device.

Reference:

NEC_GetRtMasterId();

7.6.3 NEC_GetDi

To read the EC-Slave digital 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 EC-Master communication is started. Please notice the length of SizeByte and Didata when you are using it.

Reference:

NEC_GetRtMasterId();

7.7 CoE SDO communication APIs

7.7.1 NEC_SDODownloadEx

7.7.2 NEC_SDOUploadEx

7.7.3 NEC_SDODownload

7.7.4 NEC_SDOUpload

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.

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();

7.7.5 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 numOfBackupObj;   // [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();
```

7.7.6 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 );
```

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

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();
```

7.7.7 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();
```

7.7.8 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();
```

7.7.9 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 EC-Master.

Reference:

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

7.7.10 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 );
```

Parameters:

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 EC-Master.

Reference:

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

7.8 ProcessData Access APIs

7.8.1 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 EC-Master (NexECMRtx). ProcessData principle and operation, please refer to section 5.7 Process Data Access.

Reference:

NEC_GetRtMasterId();NEC_GetProcessImageSize();NEC_RWSlaveProcessImage ()

7.8.2 NEC_GetProcessImageSize

To get length of 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 EC-Master. ProcessData principle and operation, please refer to section 5.7 Process Data Access.

Reference:

NEC_GetRtMasterId(); NEC_RWProcessImage (); NEC_RWSlaveProcessImage ()

7.8.3 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, unit 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 ()

7.9 Slave Hardware Information Access APIs

7.9.1 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* address 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:

7.9.2 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:

7.9.3 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>