NexECM

EtherCAT Master

用户手册

Manual Rev.: 1.9

Revision Date: June, 1th, 2019

Part No:

NEXCOBOT Inc.
www.nexcobot.com

Revise note:

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

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

目录

NexECM 1

Revise note: 2

目录 i

1. NexECM 介绍 1

1.1. NexECM 功能特点 2

2. EtherCAT 技术简介 4

2.1. EtherCAT 通讯协议概述 5

2.2. 网络拓朴 6

2.3. 高速效能 7

2.4. 网络同步 8

3. 编程原理 10

3.1. 基本编程框架 10

3.2. ENI 载入 11

3.3. 初始化 NexECM 11

3.4. 启动主站通讯 12

3.5. Callback 函式 13

3.5.1. 注册回调函数(Callback functions) 13

3.5.2. 周期回调函数(Cyclic-Callback function) 13

3.5.3. 事件回调函数(Event-Callback function) 15

3.5.4. 错误事件回调函数(Error-Callback function) 15

3.6. EtherCAT 状态机 16

3.6.1. ESM 状态变更 16

3.7. Process Data 存取 18

3.7.1. ProcessData 运作机制 18

3.7.2. ProcessData 数据内容 19

3.8. Mailbox 通讯 21
3.8.1. CoE -SDO 通讯 21
3.8.2. CoE -Emergency 22
4. NexECM 实时(real-time)链接库 23
4.1. RT API 总览 23
4.2. 初始化相关函式 27
4.2.1. NEC_RtGetVersion 27
4.2.2. NEC_RtGetVersionEx 28
4.2.3. NEC_RtRetVer 29
4.2.4. NEC_RtCheckLicense 30
4.2.5. NEC_RtInitMaster 31
4.2.6. NEC_RtCloseMaster 32
4.2.7. NEC_RtSetParameter / NEC_RtGetParameter 33
4.2.8. NEC_RtRegisterClient 35
4.2.9. NEC_RtUnregisterClient 37
4.3. EC-Master 控制相关函式 38
4.3.1. NEC_RtStartMaster 38
4.3.2. NEC_RtStopMaster 39
4.3.3. NEC_RtStopMasterCb 40
4.3.4. NEC_RtWaitMasterStop 41
4.3.5. NEC_RtGetMasterState / NEC_RtSetMasterState 42
4.3.6. NEC_RtChangeStateToOP 44
4.3.7. NEC_RtSetMasterStateWait 45
4.3.8. NEC_RtGetStateError 46
4.4. Process data 存取相关函式 47
4.4.1. NEC_RtGetProcessDataPtr 47
4.4.2. NEC_RtSetProcessDataPtrSource 48

4.4.3. NEC_RtGetSlaveProcessDataPtr 50
4.4.4. NEC_RtSetProcessDataOutput / NEC_RtGetProcessDataOutput / NEC_RtGetProcessDataInput 52
4.4.5. NEC_RtSetSlaveProcessDataOutput / NEC_RtGetSlaveProcessDataOutput / NEC_RtGetSlaveProcessDataInput 53
4.4.6. NEC_RtSetSlaveProcessDataOutputEx / NEC_RtGetSlaveProcessDataOutputEx / NEC_RtGetSlaveProcessDataInputEx 54
4.5. Callback 函式 56
4.5.1. *NEC_RtCyclicCallback 56
4.5.2. *NEC_RtEventCallback 57
4.5.3. *NEC_RtErrorCallback 58
4.6. ENI 相关函式 59
4.6.1. NEC_RtLoadNetworkConfig 59
4.6.2. NEC_RtGetSlaveCount 60
4.6.3. NEC_RtGetSlaveInfomation 61
4.6.4. NEC_RtGetSlaveState 63
4.7. CoE 通讯相关函式 64
4.7.1. NEC_RtStartSDODownload / NEC_RtStartSDOUpload 64
4.7.2. NEC_RtSDODownload / NEC_RtSDOUpload / NEC_RtSDODownloadEx / NEC_RtSDOUploadEx 66
4.7.3. NEC_RtStartGetODListCount 67
4.7.4. NEC_RtStartGetODList 69
4.7.5. NEC_RtStartGetObjDesc 71
4.7.6. NEC_RtStartGetEntryDesc 73
4.7.7. NEC_RtGetEmgDataCount 75
4.7.8. NEC_RtGetEmgData 76
4.8. 存取从站模块硬件信息相关函式 78
4.8.1. NEC_RtGetConfiguredAddress 78

4.8.2. NEC_RtGetAliasAddress 79
4.8.3. NEC_RtGetSlaveCoeProfileNum 81
4.8.4. NEC_RtGetSlaveConfiguredAddressEni 82
4.8.5. NEC_RtGetSlaveTimeDiffWithRefSlave 83
4.8.6. NEC_RtGetSlaveCountEx 84
4.8.7. NEC_RtGetSlaveVendorId 85
4.8.8. NEC_RtGetSlaveProductCode 86
4.8.9. NEC_RtGetSlaveRevisionNo 87
4.9. 效能监控 88
4.9.1. NEC_RtGetCyclicTick 88
4.9.2. NEC_RtGetCyclicTime 89
4.9.3. NEC_RtGetLastCyclicLoad 90
4.9.4. NEC_RtIsCyclicOverLoad 91
4.9.5. NEC_RtResetCyclicOverLoad 92
4.9.6. NEC_RtGetCyclicOverLoadCount 93
4.9.7. NEC_RtClearProbe 94
4.9.8. NEC_RtGetMasterCycleConsumption 95
4.9.9. NEC_RtGetMasterActualCycleTime 96
5. NexECM 非实时(Non-real-time)函式库(Win32 Library) 97
5.1. Win32 API 总览 98
5.2. NexECM 初始化相关函式 101
5.2.1. NEC_GetVersion / NEC_GerVersionEx 101
5.2.2. NEC_StartDriver 102
5.2.3. NEC_StopDriver 103
5.2.4. NEC_LoadRtMaster / NEC_UnLoadRtMaster 104
5.2.5. NEC_LoadRtApp 105
5.3. NexECM Runtime 控制相关函式 106

5.3.1. NEC_GetRtMasterId 106
5.3.2. NEC_ResetEcMaster 107
5.3.3. NEC_LoadNetworkConfig 108
5.3.4. NEC_StartNetwork 109
5.3.5. NEC_StartNetworkEx 110
5.3.6. NEC_StopNetwork 111
5.3.7. NEC_SetParameter / NEC_GetParameter 112
5.3.8. NEC_GetMasterType 114
5.3.9. NEC_GetMasterCount 115
5.4. 网络状态存取相关函式 116
5.4.1. NEC_GetSlaveCount 116
5.4.2. NEC_GetNetworkState 117
5.4.3. NEC_GetSlaveState 118
5.4.4. NEC_GetStateError 119
5.4.5. NEC_GetErrorMsg 120
5.5. DIO 控制相关函式 121
5.5.1. NEC_SetDo 121
5.5.2. NEC_GetDo 122
5.5.3. NEC_GetDi 123
5.5.4. NEC_SetDo_s 124
5.6. CoE 通讯相关函式 125
5.6.1. NEC_SDODownloadEx / NEC_SDOUploadEx / NEC_SDODownload / NEC_SDOUpload 125
5.6.2. NEC_GetODListCount 127
5.6.3. NEC_GetODList 129
5.6.4. NEC_GetObjDesc 131
5.6.5. NEC_GetEntryDesc 133

5.6.6. NEC_GetEmgDataCount 135
5.6.7. NEC_GetEmgData 136
5.7. Process data 存取相关函式 138
5.7.1. NEC_RWProcessImage 138
5.7.2. NEC_GetProcessImageSize 139
5.7.3. NEC_RWSlaveProcessImage 140
5.7.4. NEC_RWProcessImageEx 141
5.7.5. NEC_RWSlaveProcessImageEx 142
5.8. 存取从站模块硬件信息相关函式 143
5.8.1. NEC_GetConfiguredAddress 143
5.8.2. NEC_GetAliasAddress 144
5.8.3. NEC_GetSlaveCoeProfileNum 146

1. NexECM 介绍

NexECM 是一套建构于微软 Windows 搭配实时系统(RTOS)的 EtherCAT 主站(EC-Master) 通讯层解决方案，透过此通讯层和 EtherCAT 从站(EC-Slave devices)进行实时通讯作业。NexECM 提供丰富高阶 C / C++ 应用程序开发接口 (API) 来编程操作。

除 EtherCAT 主站通讯层外，亦可透过图形化 EtherCAT 公用程序 – NexECM Studio 进行 EtherCAT 相关工业应用开发，其功能包含:

1. EtherCAT 网络装置扫描
2. 汇入 ESI 档案，输出 ENI 档案
3. 从站模块(Slaves)网络参数设定
4. EtherCAT 通讯质量监控测试
5. 从站模块(Slaves)功能性测试操作

其详细功能介绍请参考第三章。

下图为 NexECM 主站通讯解决方案系统方块图，虚线的方块代表用户的应用程序开发位置，箭头代表存取沟通的对象。

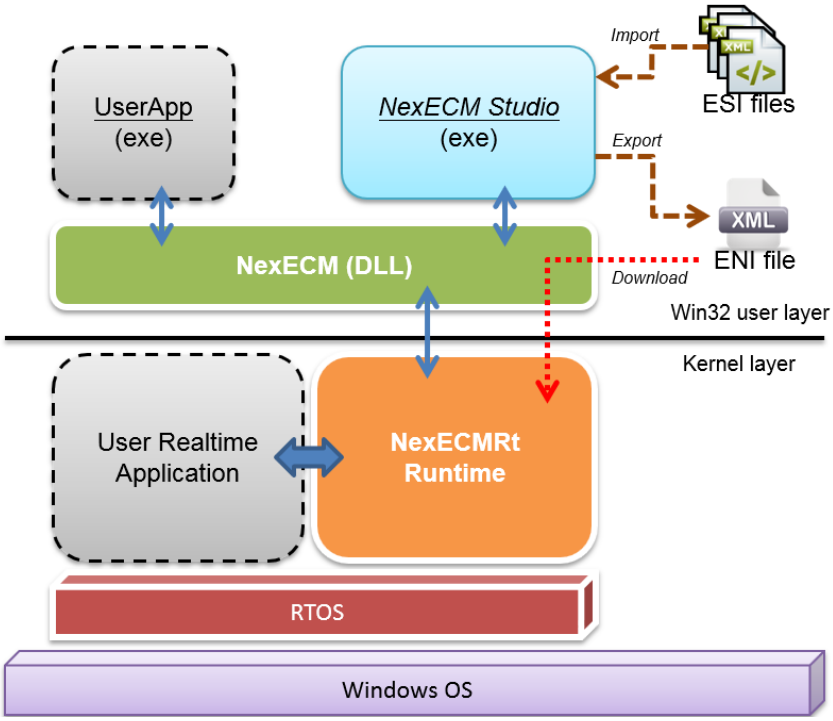


图 1.1: NEXCOBOT EtherCAT master solution 系统方块图

系统方块图个模块功能简述如下表:

软件模块	说明
运作位置:RTOS 环境	
NexECMRt Runtime	EtherCAT 主站堆栈，运行于 RTOS 环境 参考章节 CH2, CH3
运作位置:Win32 非实时环境	
NexECM.dll	Win32 函式库，透过函式库提供的 APIs，控制运行于 RTOS 环境下的 NexECMRt Runtime 参考章节 CH5
NexECM Studio.exe	NexECM 公用程序 参考文件:NexECM Studio 使用者文件

1.1. NexECM 功能特点

根据 EtherCAT 标准文件编号 ETG.1500，NexECM 支持 EtherCAT 主站功能如下表所列:

V: 已支援, △: 即将支持

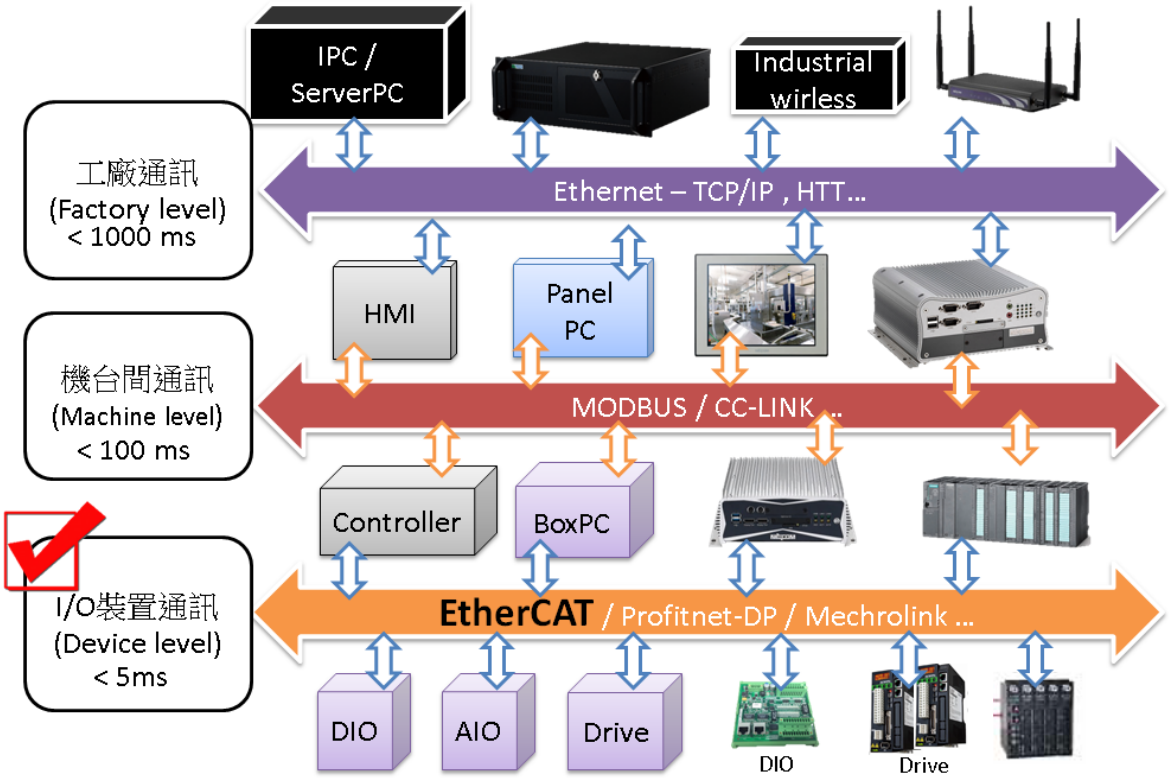
Feature name	Short description	NexECMRtx
Basic Features		
Service Commands	Support of all commands	V
IRQ field in datagram	Use IRQ information from Slave in datagram header	V
Slaves with Device Emulation	Support Slaves with and without application controller	V
EtherCAT State Machine	Support of ESM special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

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

2. EtherCAT 技术简介

随着通讯技术的进步，工业通讯现场总线(Fildbus)的技术已广泛的应用在工业控制系统之中。就网络技术而言，Ethernet 已成为最普遍，也是最广泛被应用在各种领域之上，其相关的硬件和软件技术也随技术的普及而高速的发展中。在如此大量地使用之下 Ethernet 已成为最先进且最具成本效益的一项通讯技术之一。

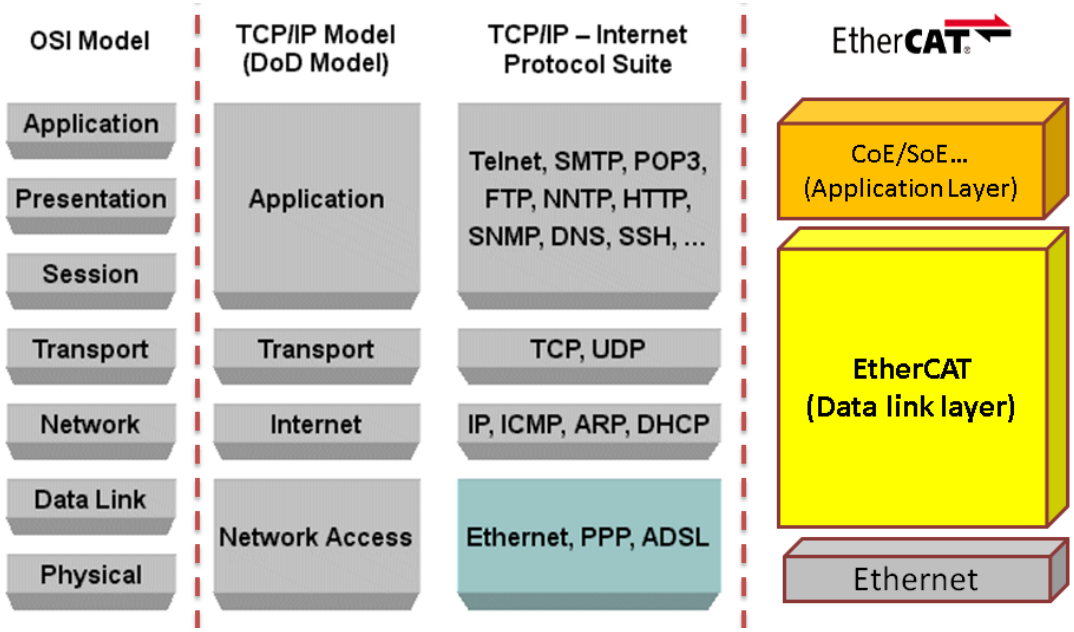
而 EtherCAT (Ethernet Control Automation Technology)是一项基于 Ethernet 被设计应用在自动化(特别是在机台自动化)的工业通讯技术，它挟带着 Ethernet 的各项优势(普遍，高速，低成本)，其相关的产品也以惊人的速度在成长。下图为一工业通讯系统示意图，EtherCAT 主要被应用在连接高速实时的 I/O 装置。



下面几个小节简单说明 EtherCAT 通讯技术及其特点。

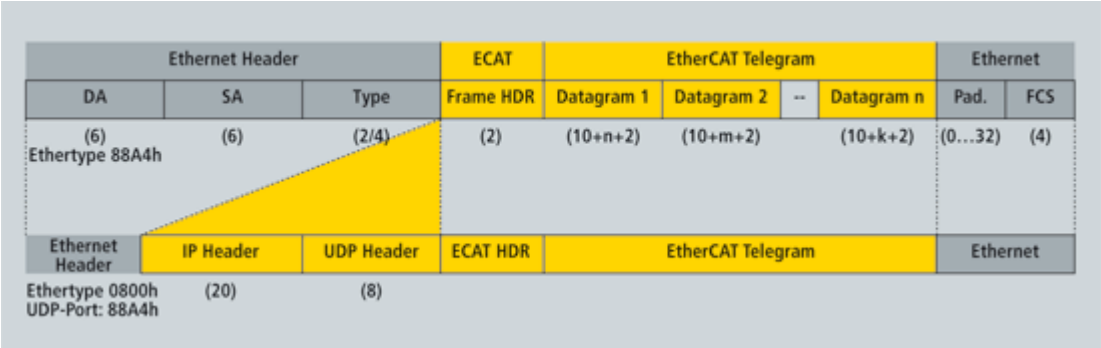
2.1. EtherCAT 通讯协议概述

EtherCAT 是一个架构在 Ethernet 物理层(Physical Layer)上具实时性(Real-time)的通讯协议，目前由 ETG (EtherCAT Technology Group)组织来维护及推广。
以 OSI 网络模型而言，EtherCAT 通讯协议主要定义于数据层(Data link Layer)和应用层(Application Layer)。



EtherCAT 的网络封包使用 IEEE 802.3 Ethernet 格式。EtherType 除了可使用标准 EtherCAT 封包之外(EtherType = 0x88A4)，亦可使用 UDP 格式(EtherType = 0x0800)，如下图。

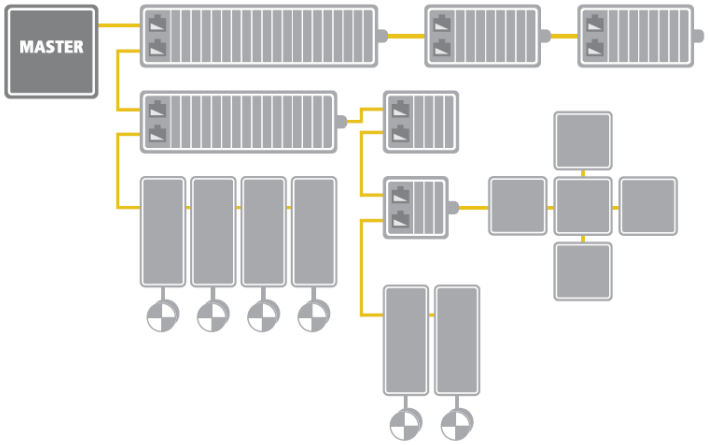
EtherCAT 的内文部分主要包含了 ECAT 的标头(Frame header)以及后面的报文(Telegrams)。



EtherCAT 网络封包格式 (数据源: http://www.ethercat.org/)

2.2. 网络拓朴

网络拓朴特性关系到现场布线的可能性。EtherCAT 几乎支持所有网络拓朴形式包含直线，树状及星状等拓朴，另外，使用标准 100BASE-TX 网络线两个装置之间最长的距离可达 100 米，由上述规格来说在设备自动化的应用领域而言几乎没任何限制。

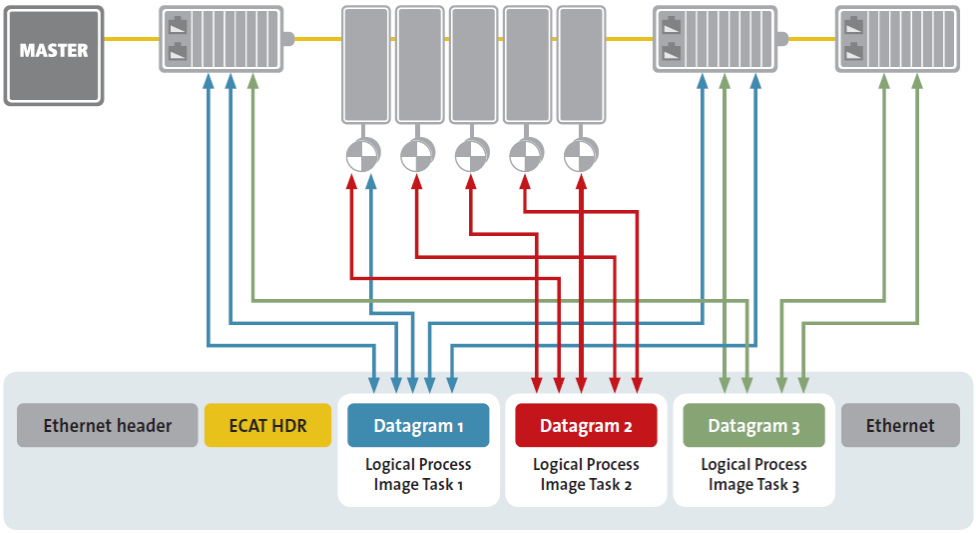


EtherCAT 网络拓朴, Line,Tree and Star 拓朴
(数据源: http://www.ethercat.org/)

使用标准 Ethernet 网络线

2.3. 高速效能

透过 EtherCAT 所定义的寻址方式，配合 EC-Slaves 上的硬件所实作的内存管理单元: Fieldbus memory manager unit (FMMU)，可完成经由传送一个网络封包可与在线所有装置(EC-Slaves)做同步的数据交换，例如 DIO 数据，AIO 数据和伺服马达位置数据等。如下示意图:



EtherCAT 网络封包与装置数据交换 (数据源: http://www.ethercat.org/)

EtherCAT 从站(EC-slaves)装置的部分，会搭载一 EtherCAT 专用通讯 IC (EtherCAT Slave IC,简称 ESC) 来完成，搭配上述 FMMU 技术，根据 ETG 的资料，1000 点的 I/O 资料更新只需约 30us 可完成交换。100 轴的伺服马达数据只需 100us。请参阅下表:

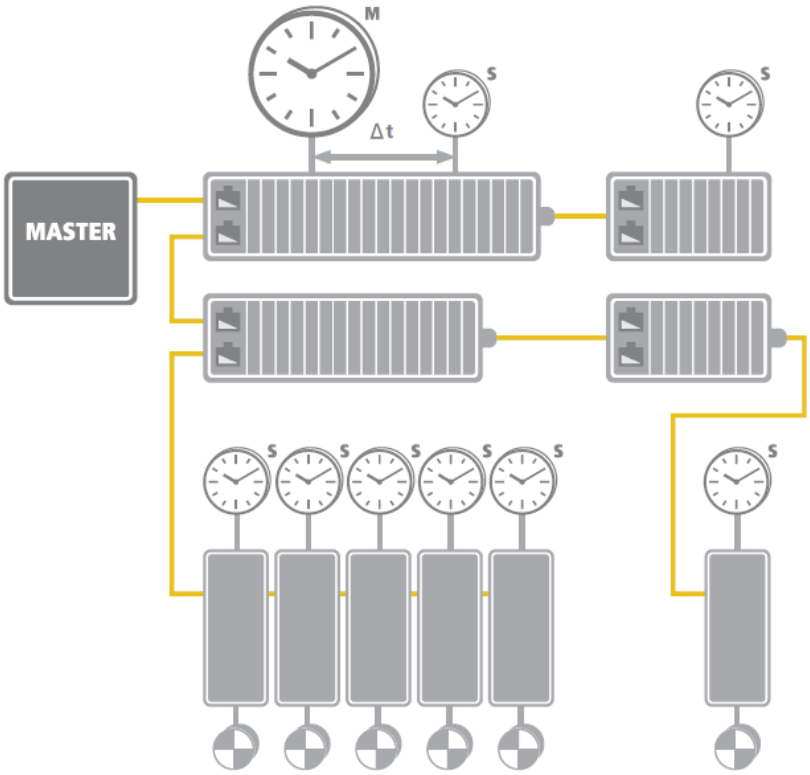
资料量	更新时间
256 点 I/O	11µs
1000 点 I/O	30µs
200 Channels 類比 I/O (16 位)	50µs (=20kHz)
100 个伺服轴 (8 Bytes Input and output per axis)	100µs
1 个现场汇流排主网关 (Fieldbus Master-Gateway) (1486 字节输入与 1486 字节输出资料)	150µs

EtherCAT 通讯数据量与时间关系 (数据源: http://www.ethercat.org/)

若以 100 轴 100us 的数据交换速度而言相当于控制带宽为 10KHz，此控制带宽用于速度控制回路或甚至扭矩(Torque)控制回路都已足够。

2.4. 网络同步

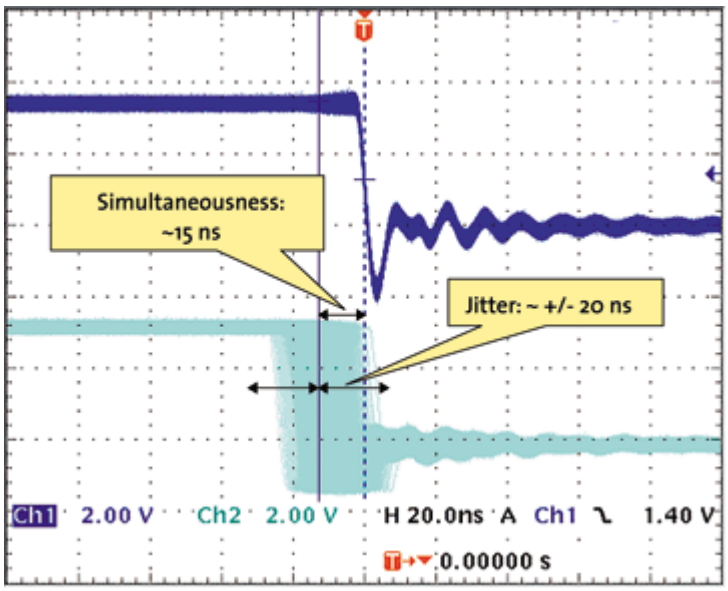
在设备自动化应用中对分布式网络的要求除了网络通讯需具备实时性(Real-time)外，其装置之间的同步性要求不可或缺，例如控制多轴伺服马达的直线或圆弧补间运动，龙门(Gantry)轴控制等。这些应用必须具备网络同步性才能确保应用的正确性。



EtherCAT 分散时钟技术示意图 (数据源: http://www.ethercat.org/)

EtherCAT 的网络同步机制是根据 IEEE-1588 精确时间同步协议，延伸定义出所谓分散时钟技术(Distributed-clock 简称 DC)。简单的来说在每个 EtherCAT 的从站装置上(ESC 内)都自行维护一个硬件时钟(Local Clock)，其时间最小间格为 1 ns 共 64 位，这个由 EC-Slave 自行维护的时间称之为”当地系统时钟” (Local system time)，在透过精确网络对时机制以及动态时间补偿机制后(*1)，EtherCAT 的 DC 技术可以保证所有的从站装置上的当地系统时钟(Local system time)的误差在+/- 20 奈秒(nano-second)之内，如下图两个从站间 I/O 信号误差约为 20 ns。

(*1) 可参考 EtherCAT standard document ETG1000.4

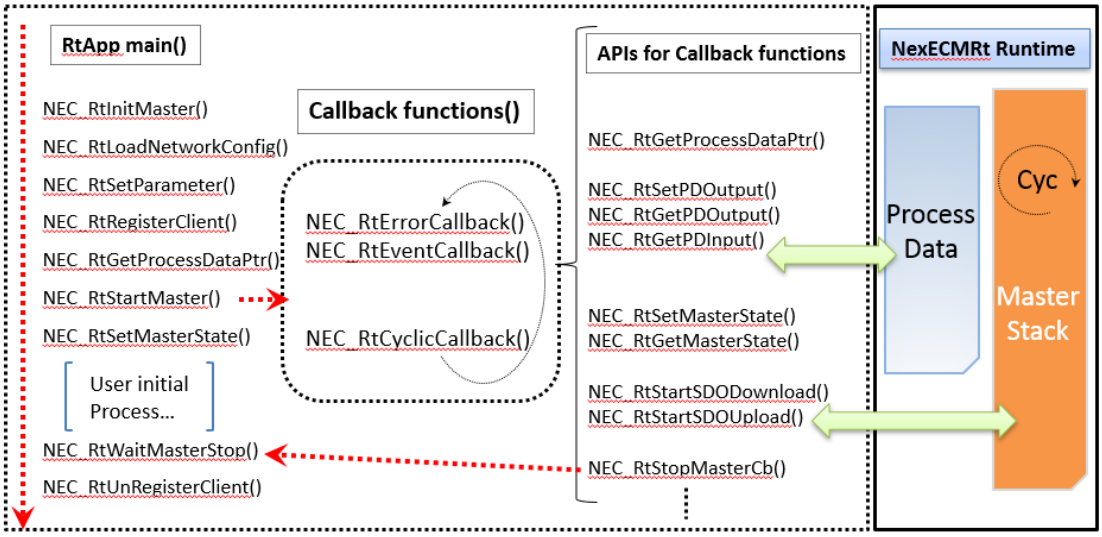


Sync 信号误差 (数据源: http://www.ethercat.org/)

3. 编程原理

3.1. 基本编程框架

下图说明 NexECM 基本编程流程。



基本流程如下

步骤	说明	相关函式
您的实时应用程序主程序 main(): ex. RtUserApp.rtss (详细 API 说明请参考 CH.4)		
1	初始化 NexECMRt Runtime	NEC_RtInitMaster()
2	加载 ENI 数据	NEC_RtLoadNetworkConfig()
3	设定参数，如主站通讯周期(Cycle-time)等	NEC_RtSetParameter() NEC_RtGetParameter()
4	注册 Callback 函式	NEC_RtRegisterClient() NEC_RtUnregisterClient()
5	取得 ProcessImage 的内存指针(Memory pointer)	NEC_RtGetProcessDataPtr() NEC_RtGetSlaveProcessDataPtr()
6	启动 EtherCAT 通讯	NEC_RtStartMaster()
7	将 EtherCAT 的状态切换至”**OP**”状态	NEC_RtSetMasterState()
8	RTOS 主程序为一 main()型态终端应用程序，当程序结束后即整个程序会被卸除，因此当完成上述程序后，可依照您的应用:	NEC_RtWaitMasterStop()

	1. 让主程序进入睡眠等待 EtherCAT 通讯结束或 2. 您的其他(装置)控制应用程序	
在周期 Callback 函式中 (详细 API 说明请参考 CH.4)		
9	你的控制程序进行 1. ProcessData 存取 2. CoE 通讯	NEC_RtSetPDOutput() NEC_RtGetPDOutput() NEC_RtGetPDInput() NEC_RtStartSDODownload() NEC_RtStartSDOUpload()
10	应用程序结束，停止 EtherCAT 通讯	NEC_RtStopMasterCb()

上述可流程可参考 NexECM 安装目录中的范例程序。

3.2. ENI 载入

ENI 网络档案(EtherCAT Network Information)除提供 NexECM runtime 如何初始化 EC-Slaves 的信息另包含 runtime 本身的设定参数。因此，进行 EtherCAT 通讯前，必须先加载 ENI 档案。启动通讯后，NexECM runtime 即会依照 ENI 数据，对 EC-Slaves 进行设定。启动过程中 NexECM runtime 会侦测 ENI 中的数据是否与目前实际网络接线配置是否相同，若有异常会停止通讯并发出错误事件(Error callback)，

你可以透过 NexECM Studio 公用程序来设定网络配置并产生 ENI 档案，其详细操作方式请参考 NexECM Studio 使用文件。

3.3. 初始化 NexECM

在您的实时应用程序中，呼叫 *NEC_RtInitMaster*()函数来初始化 NexECM runtime。此动作将不会重置已存在的 ENI 的数据数据。

呼叫 *NEC_RtSetParameter*()函数来完成配置 NexECM runtime。例如，参数代号 *NEC_PARA_S_ECM_CYCLETIMEUS* 是用来设置主站的通讯周期时间。而其它参数请参阅该函数的说明。

使用 *NEC_RtRegisterClient*()来注册一组客户端的 Callback 函数，其 Callback 的原型如下:

```
void NEC_RtCyclicCallback( void *UserDataPtr );
```

void NEC_RtEventCallback (void *UserDataPtr, U32_T EventCode);
void NEC_RtErrorCallback (void *UserDataPtr, U32_T ErrorCode);

Callback	说明
NEC_RtCyclicCallback	周期性 Callback 函数，实作控制程序
NEC_RtEventCallback	当预先定义的事件发生由 NexECM runtime 呼叫
NEC_RtErrorCallback	当错误事件产生时会被 NexECM runtime 呼叫

这些 Callback 函数用于与 NexECM runtime 进行的同步通信与 ProcessData 数据交换。当 EtheCAT 状态机状态 (State-Machine)(*)更改为“ SAFEOP ”和“OP”时，NexECM runtime 开始在每个通讯周期中将 ProcessData 数据更新传送至 EC-Slaves 并从 EC-Slave 将数据传回 ProcessData 中。

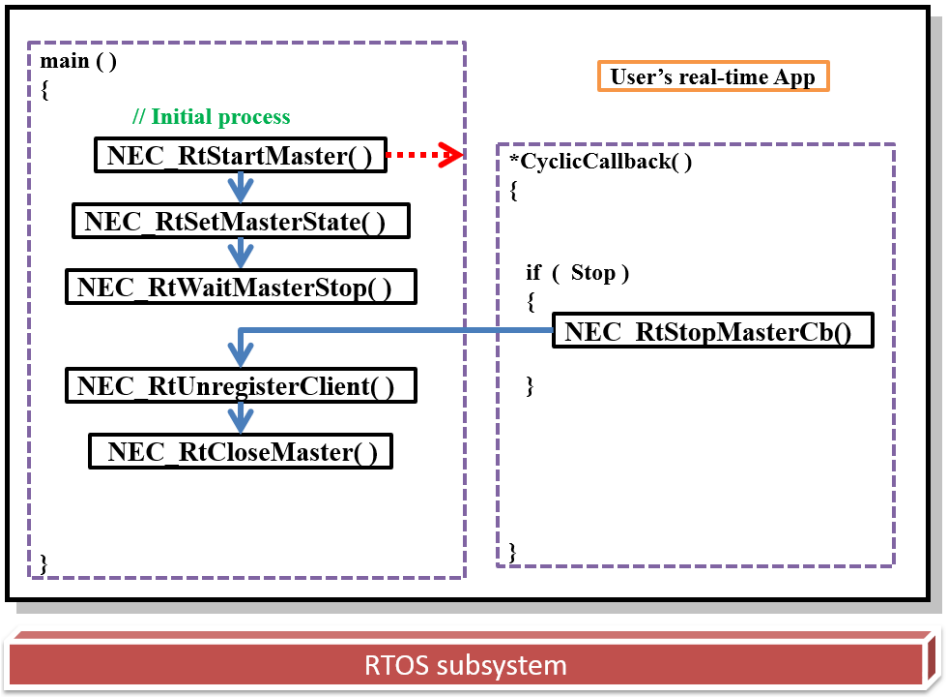
(*)关于 EtherCAT 状态机，请参考 3.6 小节”EtherCAT 状态机”的说明。

大多数的 EC-Slave 的通讯应用程序(例如运动控制算法等)将被放置在周期的 Callback 函数中，在 Callback 函数种编程的基本原则是**避免**在 Callback 中撰写导致 Callbak 停止或过于拢长的程序使 Callback 的运行时间超出通讯周期。因此尽量避免使用类似 Sleep(), RtSleep()或 while();, do while()的 Polling 等程序在 Callback 函数中。关于 Callback 的使用请参考 3.5 小节 Callback 函式。

3.4. 启动主站通讯

根据上一小节完成主站的初始化设定程序后，调用 *NEC_RtStartMaster*() 起动主站 EtherCAT 通讯，若成功的调用此函数，用户所注册的周期回调函数(Callback)会周期的执行。请注意，当启动通讯后 NexECM runtime 内的 EtherCAT 状态机会保持在”INIT” 状态，直到使用 *NEC_RtSetMasterState*()来送出状态变更要求。

一般的情况下，用户会将控制算法实现在回调函数(Callback function)之中。然而，实时应用程序是一个类似主控制台(Console)编程的方式，有一标准的进入点 *main*()函数。当 *main*()函数返回时，该程序即结束并由 RTOS 系统中卸除。为了防止主程序的结束，用户可使用 *NEC_RtWaitMasterStop*()来阻止该程序结束，使主程序进入休眠状态并等待 NexECM runtime 的停止信号。当用户要结束应用程序并停止通讯，可在回调函数中使用 *NEC_RtStopMasterCb*()来发送停止信号来唤醒被阻塞的 *NEC_RtWaitMasterStop*()使其返回。



关于回调函数(Callback function)的使用，请参考下一小节。EtherCAT 状态机将详述于 3.6 小节

3.5. Callback 函式

3.5.1. 注册回调函数(Callback functions)

NexECM 提供三种回调函数，

1. 周期回调函数 (Cyclic-Callback function)
2. 事件回调函数 (Event-Callback function)
3. 错误事件回调函数 (Error-Callback function)

启动 EtherCAT 通讯前，使用 *NEC_RtRegisterClient*()将 Callback 函数注册 NexECM runtime 之中。结束通讯后使用 *NEC_RtUnRegisterClient*()取消 Callback 函数的注册。必须注意避免在通讯过程中使用 *RtUnRegisterClient*()将 Callback 函数取消注册。

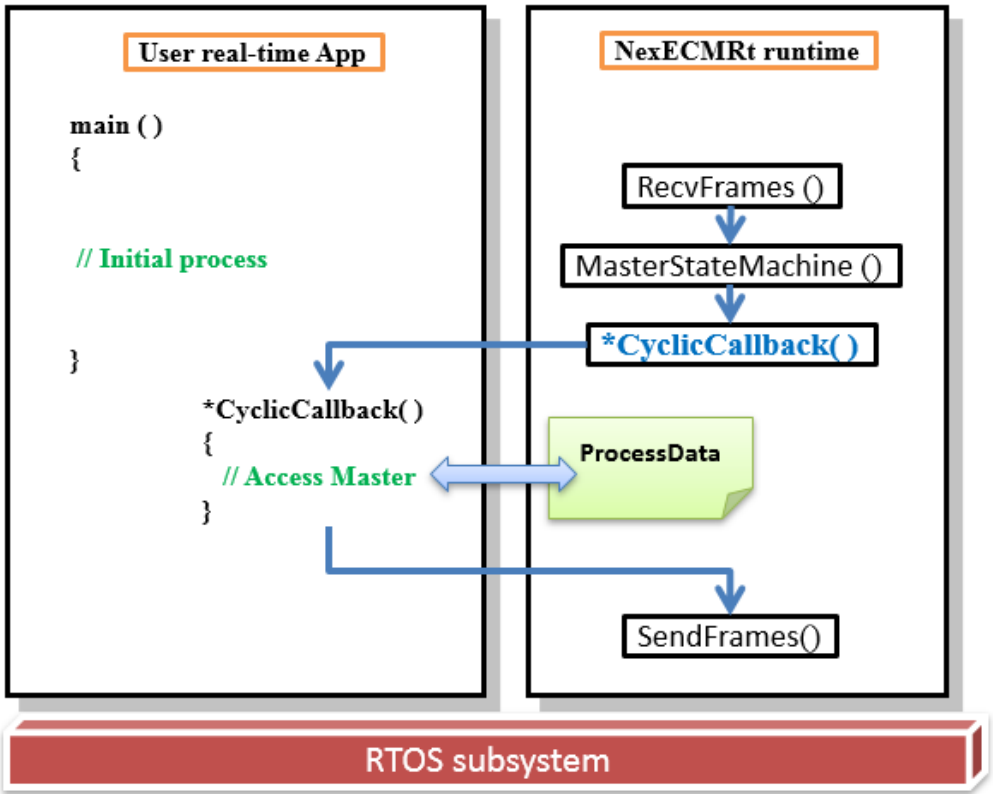
3.5.2. 周期回调函数(Cyclic-Callback function)

透过 *NEC_RtGetProcessDataPtr*()函数取得 ProcessData 内存的指针(Memory pointer)，在取得 ProcessData memory 指标后你可以任意的读写 NexECM runtime 中的 ProcessData，因此我们可以将这个内存视为一块 Share memory，使之成为

用户的程序和 NexECM runtime 之间的数据交换管道。但 NexECM runtime 的周期程序和用户的程序基本上可以视作两个独立的线程(Thread)，因此如何让用户的程序和 runtime 的程序之间能同步的存取 ProcessData，保持数据的一致性(Data Consistence)。

另外，在实时工业控制应用中，某些从站应用必须在每个通讯周期中取得主站的数据以完成实时工业控制行为。例如伺服马达的控制，必须在每个通讯周期中对数个伺服马达送出该马达的位置控制信息，来完成同步位置控制。因此，用户的实时控制程序必须与 runtime 的周期通讯程序必须互相链接。

NexECM 提供 Callback 函数的同步回呼机制(synchronous callbacks)，使您的实时应用程序与 NexECM 中的 ProcessData 存取同步。透过此机制来完成上述的两种状况，保证传递数据的一致性以及通讯周期的同步性。下图表示在 NexECM 中，NexECM runtime 的数据处理程序与用户所提供的 Callback 函数运行的关系流程图。



3.5.3. 事件回调函数(Event-Callback function)

当 NexECM runtime 内定的事件发生时，runtime 会同步呼叫此函数通知用户，同时传入事件代码(Event code)，使用者可以根据事件代码来处理对应的事件，或者忽略不处理。
如同周期回调函数，在函数内存取 ProcessData 可以确保数据的同步性。

3.5.4. 错误事件回调函数(Error-Callback function)

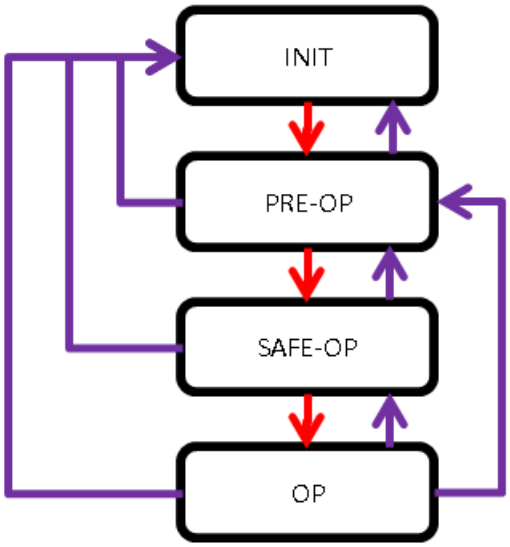
当 NexECM runtime 发生错误事件时，runtime 会同步呼叫此函数通知用户，同时传入错误代码(Error code)，使用者可以根据错误代码来处理对应的事件，或者忽略不处理。

如同周期回调函数，在函数内存取 ProcessData 可以确保数据的同步性。

3.6. EtherCAT 状态机

根据 EtherCAT 标准文件(ETG.1000.6)，EtherCAT 主站必须具备状态机(EtherCAT State Machine，简称 ESM)，来负责处理主站与各从站(EC-Slaves)之间从初始化状态到可操作状态的工作流程。其状态图如下图所示，包含四种状态:

1. INIT 状态
2. PREOP 状态
3. SAFEOP 状态
4. OP 状态



EtherCAT State Machine diagram

一般而言在 EtherCAT 的应用中，在进行工业控制流程之前，必须将状态机的状态从”INIT”状态变更为”OP”状态，此动作所隐含的意义在于初始化 NexECM runtime 网络组态和 EC-Slaves 模块相关设定，其设定的内容是根据 NexECM Studio 公用程序所输出的 ENI (EtherCAT Network Information) 档案。

3.6.1. ESM 状态变更

启动 EtherCAT 通讯后，*NEC_RtChangeStateToOP*()将状态切换至”OP”状态。若有特殊应用可使用 *NEC_RtSetMasterStateWait*()切换至不同状态。上述函数不应在 Callback 函数中使用。在 Callback 函数中应使用 *NEC_RtSetMasterState*()函数改变目标主站状态，*NEC_RtSetMasterState()*函数只用于发出状态改变要求，并不会等待的状态是否成功改变，须配合使用 *NEC_RtGetMasterState()*函数检查状态是否正确地完成变更。关于函数的用法请参考相关函数说明。

在各个状态下，EC-Maste 所提供的服务以及 EC-Slaves 可以接受的操作指令如下表所示:

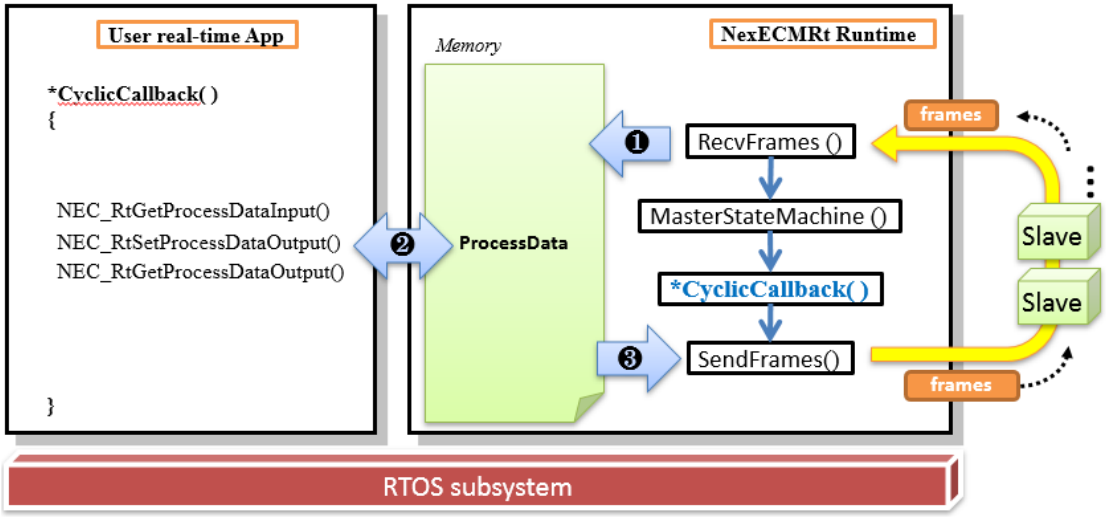
状态	EC-Master 服务	EC-Slave 服务
INIT	➢ **周期回调函数启动** ➢ 无 ProcessData 通讯传输(To slaves) ➢ 无 Mailbox 通讯传输(To slaves)	➢ EC-Master 不可控制
PREOP	➢ 周期回调函数启动 ➢ **Mailbox 通讯传输启动(To slaves)** -SDO 通讯启动 ➢ 无 ProcessData 通讯传输(To slaves)	➢ Mailbox 通讯启动
SAFEOP	➢ 周期回调函数启动 ➢ Mailbox 通讯传输启动(To slaves) -SDO 通讯启动 ➢ **ProcessDataInput 通讯传输启动** ➢ 无 ProcessDataOutput 通讯传输	➢ Mailbox 通讯启动 ➢ 将 Input 数据更新到 ProcessDataInput
OP	➢ 周期回调函数启动 ➢ Mailbox 通讯传输启动(To slaves) -SDO 通讯启动 ➢ ProcessDataInput 通讯传输启动 ➢ **ProcessDataOutput 通讯传输启动**	➢ Mailbox 通讯启动 ➢ 将 Input 数据更新到 ProcessDataInput ➢ 从 ProcessDataOutpt 取得 Output 数据并输出

3.7. Process Data 存取

3.7.1. ProcessData 运作机制

NexECM 在内部维护一块内存供 EtherCAT 通讯 ProcessData 使用，ProcessData 上的数据会根据通讯周期时间定期的传输及更新数据。如下图所示 NexECM Runtime 定期的执行:

1. 接收 EC-Slaves 数据，写入 ProcessData 内存中
2. EthreCAT 状态机处理
3. 呼叫周期 Callback 函数
4. 将 ProcessData 的数据传送数据到 EC-Slaves



可利用 *NEC_RtGetProcessDataPtr*()函数直接取得 ProcessData 的内存指针，用户的程序可直接访问此内部存储器。此方式其优点在于有较高的存取效率，但必须自行注意存取的范围以及存取的时机，当存取错误时可能会导致系统崩溃。另一较安全的存取方式是使用下列 API 方式存取 ProcessData，API 内部会检查存取的范围是否正确。

NEC_RtSetProcessDataOutput();

NEC_RtGetProcessDataOutput();

NEC_RtGetProcessDataInput();

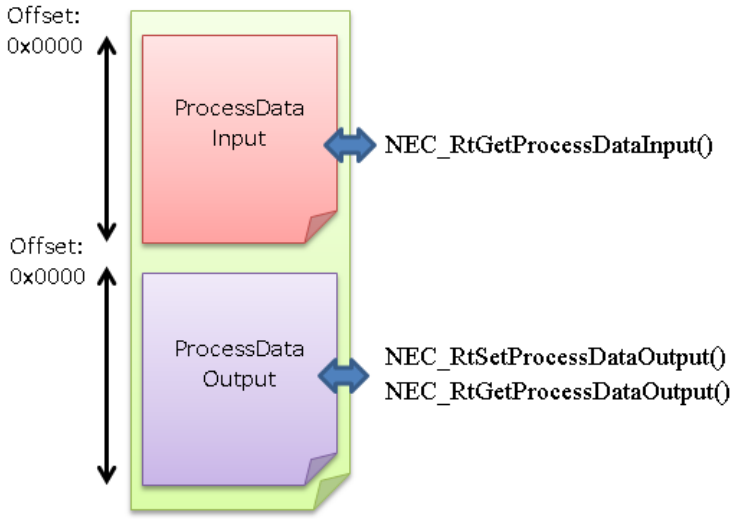
关于 Process Data 的存取时机，建议在 Callback 程序之中来存取，以确保数据全区块的同步完整性，若在 Callback 函数之外存取 ProcessData，数据的同步性只有 1 Byte 为单位。

ProcessData 在内部实际分为 ProcessDataInput 区域和 ProcessDataOutput 区域。其传递数据的方向如下表:

ProcessData	资料方向	Read/Write
ProcessDataInput	EC-Slaves 传送至 NexECM runtime	Read Only

ProcessDataOutput	NexECM runtime 传送至 EC-Slaves	Read/Write

存取所对应的 API 如下图表示:

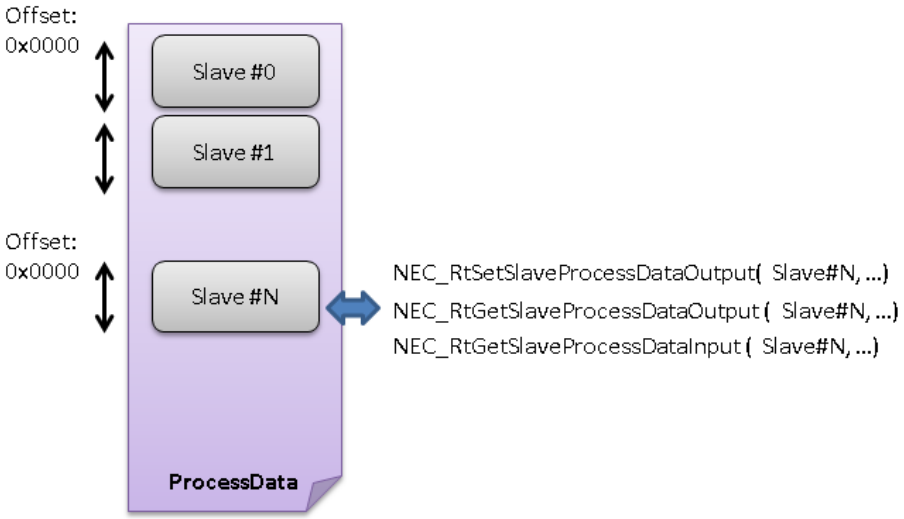


3.7.2. ProcessData 数据内容

ProcessData 内的数据内容根据不同的 EC-Slaves 模块所提供的内容有所差异。例如 DIO 类的 EC-Slave 模块其 ProcessData 为 Digital Input 的状态值或 Digital Output 的输出数据。而功能较多的 EC-Slave 模块例如伺服驱动器，其 ProcessData 的数据可以是”目标位置”(Target Position)，”马达实际位置” (Position actual value)等。

这些数据排放在 ProcessData 上的位置(Offset)根据模块串接的顺序和其所提供数据的长度的大小所决定，如下图所示，每个 EC-Slaves 在 ProcessData 之中会有各自的内存区块。

若要同步输出数据到不同模块上，可在 Callback 函数中存取这些内存即可达到此效果。



可以使用下列函数存取 Slave 各自的 ProcessData 内存区域

NEC_RtSetSlaveProcessDataOutput ();

NEC_RtGetSlaveProcessDataOutput ();

NEC_RtGetSlaveProcessDataInput ()

3.8. Mailbox 通讯

3.8.1. CoE -SDO 通讯

CANOpen 技术在工业自动化应用中已经是一相当普遍且成熟的技术，为了和既有控制技术结合，EtherCAT 提供 CANOpen over EtherCAT (CoE)来实现。在 CANOpen 中主要定义了两种传输方式:

1. PDO (Process data object)，实时同步数据传输
2. SDO (Service data object)，异步数据传输

PDO 的数据会直接映像到 EtherCAT 的 ProcessData 区域，而 SDO 部分则使用 EtherCAT 的 Mailbox 机制达成。
根据 CANOpen 的 SDO 传输可区分为

1. SDO download: 主站将数据传输到从站
2. SDO upload: 资料由从站上传到主站
3. SDO information: 取得 ObjectDictionary List

SDO 特性如下:

1. SDO 传输采用交握(Handshake)的方式
2. 需要数个通讯周期才能完成一次 SDO 传输
3. 当 SDO 传输成功表示数据有正确的交换到另一端
4. 一般用在非实时性的参数设定

NexECM 提供下列函数来完成 SDO 的通讯传输:

函数名称	说明	T
NEC_RtStartSDODownload	送出 SDO download 命令数据(Master to slave)	B
NEC_RtStartSDOUpload	送出 SDO upload 命令数据(Slave to Master)	B
NEC_RtSDODownload	执行 SDO download (Structure)	X
NEC_RtSDOUpload	执行 SDO upload (Structure)	X
NEC_RtSDODownloadEx	执行 SDO download (Native data type)	X
NEC_RtSDOUploadEx	执行 SDO upload (Native data type)	X

API 详细的使用方式请参考 3.7 小节说明。

3.8.2. CoE -Emergency

当 EC-Slaves 有错误发生时，会透过 CoE 通讯传递 Emergency 讯息至 NexECM runtime。NexECM runtime 收到此 Emergency 后，会储存至内部的 Emergency 讯息容器中，若用户有注册事件回调函数，则事件回调函数会被 NexECM runtime 呼叫。Emergency 引发事件回调函数的规则如下：

1. 当有新的 Emergency 讯息新增至讯息容器时，事件回调函数会被引用。
2. 同一时间点，NexECM runtime 接收来自多个 EC-Slaves 的 Emergency 讯息，此时回呼事件函数仅会被呼叫一次。
3. 若无新的事件发生，则虽然 Emergency 讯息容器含有未读走的讯息，事件回调函数不会再次被呼叫。

若 Emergency 讯息容器已满，再来的讯息会覆盖掉最旧的讯息。

NexECM 提供下列函数来完成读取内部中的 Emergency 讯息，

函数名称	说明	T
NEC_RtEmgDataCount	读取 Emergency 讯息容器中讯息数据数量	B
NEC_RtEmgData	读取 Emergency 讯息容器中讯息数据	B

API 详细的使用方式请参考 3.7 小节说明。

4. NexECM 实时(real-time)链接库

4.1. RT API 总览

下表列出 NexECM 实时函式库 API 的列表，API 定义于 NexECMRt.h 头文件之中。

“T 字段(Type)”代表该函式可被呼叫的位置

C :只能在 Callback 函式中呼叫

X :不能再 Callback 函式中呼叫

B :没有限制

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

函数名称	说明	T
初始化相关函式		
NEC_RtGetVersion NEC_RtGetVersionEx NEC_RtRetVer	取得 NexECM 目前版本信息	B
NEC_RtCheckLicense	检查软件授权状态	X
NEC_RtInitMaster	初始化主站	X
NEC_RtCloseMaster	关闭主站	X
NEC_RtSetParameter	设置主站参数	X
NEC_RtGetParameter	读取主站参数	X
NEC_RtRegisterClient	注册 Callback clients 函数	X
NEC_RtUnregisterClient	取消 Callback client 函数注册	X
EC-Master 控制相关函式		
NEC_RtStartMaster	启动 EtherCAT 通讯	X
NEC_RtStopMaster	停止 EtherCAT 周期通讯	X
NEC_RtStopMasterCb	发出 EtherCAT 通讯中止要求	C
NEC_RtWaitMasterStop	等待 EtherCAT 通讯中止要求并停止通讯	X
NEC_RtGetMasterState	取得 EtherCAT 目前的状态	B
NEC_RtSetMasterState	发出 EtherCAT 状态变更要求	B
NEC_RtChangeStateToOP	阻塞式变更 EtherCAT 状态为”OP”状态	X
NEC_RtSetMasterStateWait	阻塞式变更 EtherCAT 状态	X
NEC_RtGetStateError	读取状态错误代码	B
Process data 存取相关函式		
NEC_RtGetProcessDataPtr	取得 ProcessData 内存位置	B

NEC_RtSetProcessDataPtrSource	设定外部空间给 ProcessData	B
NEC_RtSetProcessDataOutput	写入 ProcessDataOutput 资料	B
NEC_RtGetProcessDataOutput	读取 ProcessDataOutput 数据	B
NEC_RtGetProcessDataInput	读取 ProcessDataInput 数据	B
NEC_RtSetSlaveProcessDataOutput	写入 EC-Slave 的 ProcessDataOutput 资料	B
NEC_RtGetSlaveProcessDataOutput	读取 EC-Slave 的 ProcessDataOutput 数据	B
NEC_RtGetSlaveProcessDataInput	读取 EC-Slave 的 ProcessDataInput 数据	B
NEC_RtSetSlaveProcessDataOutputEx	写入 EC-Slave 的 ProcessDataOutput 数据，支持位型式	B
NEC_RtGetSlaveProcessDataOutputEx	读取 EC-Slave 的 ProcessDataOutput 数据，支持位型式	B
NEC_RtGetSlaveProcessDataInputEx	读取 EC-Slave 的 ProcessDataInput 数据，支持位型式	B
Callback 函式（函式原型）		
*NEC_RtCyclicCallback	周期回调函数	C
*NEC_RtEventCallback	事件回调函数	C
*NEC_RtErrorCallback	错误事件回调函数	C
ENI 相关函式		
NEC_RtLoadNetworkConfig	加载 ENI 信息	X
NEC_RtGetSlaveCount	读取 EC-Slaves 个数	B
NEC_RtGetSlaveInfomation	读取某个EC-Slaves 的信息	B
NEC_RtGetSlaveState	读取多个EC-Slaves 的状态	B
CoE 通讯相关函式		
NEC_RtStartSDODownload	送出 SDO download 命令数据 (Master to slave)	B
NEC_RtStartSDOUpload	送出 SDO upload 命令数据(Slave to Master)	B
NEC_RtSDODownload	执行 SDO download (Structure)	X
NEC_RtSDOUpload	执行 SDO upload (Structure)	X
NEC_RtSDODownloadEx	执行 SDO download (Native data type)	X

NEC_RtSDOUploadEx	执行 SDO upload (Native data type)	X
NEC_RtStartGetODListCount	读取 EC-Slave 五个对象字典(Object Dictionary)种类各别数量	B
NEC_RtStartGetODList	读取 EC-Slaves 的各种类的对象字典(Object Dictionary)索引值(index)	B
NEC_RtStartGetObjDesc	读取 EC-Slavea 单一对象的 Object Description 信息	B
NEC_RtStartGetEntryDesc	读取 EC-Slavea 的单一对象 Entry Description 信息	B
NEC_RtGetEmgDataCount	读取目标主站中 Emergency 数据数量	B
NEC_RtGetEmgData	读取目标主站中 Emergency 数据	B
存取从站模块硬件信息相关函式		
NEC_RtGetConfiguredAddress	取得从站 Configured Station Address	X
NEC_RtGetAliasAddress	取得从站 Configured Station Alias	X
NEC_RtGetSlaveCoeProfileNum	取得从站 ProfileNum 信息	B
周期时钟信息与监测相关函式		
NEC_RtGetCyclicTick	取得主站周期时钟 Tick	B
NEC_RtGetCyclicTime	取得主站周期时钟时间	B
NEC_RtGetLastCyclicLoad	取得主站最后一次周期程序所花费时间	B
NEC_RtIsCyclicOverLoad	检查主站周期程序是否曾经逾时	B
NEC_RtResetCyclicOverLoad	重置主站周期程序逾时状态	B
NEC_RtGetCyclicOverLoadCount	取得主站周期程序逾时统计	B

API 所使用的 C/C++数据型态定义于 nex_type.h 中，说明如下表:

型别	C/C++ 原型	说明	大小 byte	范围
BOOL_T	int	布尔型别	4	0:False, 1:True
U8_T	unsigned char	无号整数	1	0 ~ 255
U16_T	unsigned short	无号整数	2	0 ~ 65535
U32_T	unsigned int	无号整数	4	0 ~ 4294967295
U64_T	unsigned __int64	无号整数	8	0 ~ 18446744073709551615
I8_T	char	有号整数	1	-128 ~ 127
I16_T	short	有号整数	2	-32768 ~ 32767

I32_T	int	有号整数	4	-2147483648 ~ 2147483647
I64_T	__int64	有号整数	8	-9223372036854775808 ~ 9223372036854775807
F32_T	float	浮点数	4	IEEE-754，有效小数后 7 位
F64_T	double	双精浮点数	8	IEEE-754，有效小数后 15 位
RTN_ERR	int	错误代码	4	-2147483648 ~ 2147483647

4.2. 初始化相关函式

4.2.1. NEC_RtGetVersion

取得 NexECM 目前版本信息

C/C++语法:

RTN_ERR NEC_RtGetVersion(U32_T *Version);

参数:

U32_T *Version:

回传 NexECM 版本信息

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

没有限制。

参阅:

NEC_RtRetVer()

4.2.2. NEC_RtGetVersionEx

取得 NexECM 目前版本信息

C/C++语法:

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

参数:

_opt_null_ I32_T *PRetMajor:

回传 Major 信息

_opt_null_ I32_T *PRetMinor:

回传 Manior 信息

_opt_null_ I32_T *PRetStage:

回传 Stage 信息

_opt_null_ I32_T *PRetBuild:

回传 Build 信息

回传值:

回传完整的NexECM版本信息

用法:

没有限制。

参阅:

NEC_RtRetVer()

4.2.3. NEC_RtRetVer

回传 NexECM 目前版本信息

C/C++语法:
U32_T NEC_RtRetVer();

参数:
<无参数>

回传值:
回传NexECM目前版本信息。

用法:
没有限制。

参阅:
NEC_RtGetVersion()

4.2.4. NEC_RtCheckLicense

回传 NexECM 产品授权序号状态

C/C++语法:

RTN_ERR NEC_RtCheckLicense();

参数:

<无参数>

回传值:

软件授权序号有效时，返回0，反之，返回其它值

用法:

没有限制。

参阅:

4.2.5. NEC_RtInitMaster

初始化目标主站

C/C++语法:

RTN_ERR NEC_RtInitMaster(U16_T MasterId);

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

使用其它 NexECM RT 函式前，先调用此函数进行函式库内部初始化。

注意! 禁止于 Callback 函式中调用此函数

参阅::

NEC_RtCloseMaster()

4.2.6. NEC_RtCloseMaster

关闭目标主站

C/C++语法:

RTN_ERR NEC_RtCloseMaster(U16_T MasterId);

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

应用程序结束前，调用此函数释放 NexECM runtime 内部资源。

注意! 禁止于 Callback 函式中调用此函数

参阅::

NEC_RtInitMaster()

4.2.7. NEC_RtSetParameter / NEC_RtGetParameter

这两个函数用于设置和读取目标主站参数。

C/C++语法:

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);

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0

U16_T ParaNum: 指定参数代码，请参考用法:

I32_T ParaData: 指定参数值，请参考用法:

I32_T *ParaData: 回传参数值，请参考用法:

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

用于启动 EtherCAT 通讯之前，设定 EtherCAT 通讯使用的参数，其参数列表如下。调用 *NEC_RtInitMaster*()不会影响此区参数设定。

注意! 禁止于 Callback 函式中调用此函数

参数代码	说明	参数值
NEC_PARA_S_ECM_CYCLETIMEUS	通讯周期，单位 micro-second	250 ~ 1000000
NEC_PARA_S_NIC_INDEX	指定要使用的 NIC Port，注意, Load ENI 时可能会修改此参数，请参考 CH7.4.5 *NEC_LoadNetworkConfig*()	0 ~ Max NIC port
NEC_PARA_ECM_RECV_TIMEOUT_US	定义 EtherCAT 网络封包回传 timeout 时间，单位 micro-second。一般使用默认值即可。	250 ~ 2000000
NEC_PARA_ECM_LINK_ERR_MODE	网络断线行为模式: **LINKERR_AUTO**:	LINKERR_AUTO (0) LINKERR_MANUAL (1)

	当 EC-Slave(s) 断线，目标主站自动侦测断线的装置。当装置重新联机自动将 EC-Slaves 初始化并设定为”OP”状态 **LINKERR_MANUAL:** 当某装置断线，其状态会停留在 ERROR 状态。当断线状置恢复联机，不会重新初始化。 **LINKERR_STOP:** 只要有一装置断线，目标主站将网络中断，并进入 ERROR 状态。	LINKERR_STOP (2)
NEC_PARA_ECM_DC_CYC_TIME_MODE	DC 时间设定模式: 0: 根据 Master cycle time (预设) 1: 根据 ENI 资料	0~1

参阅:

NEC_RtGetStateError(); NEC_RtStartMaster()

4.2.8. NEC_RtRegisterClient

注册 Callback clients 函数至目标主站

C/C++语法:

RTN_ERR NEC_RtRegisterClient(U16_T MasterId, TClintParam *ClientParam);

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0

TClintParam *ClientParam: Callback client 资料

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

U32_T version: 传入前设定版本，使用 *NEC_RtRetVer*()

void *userDataPtr:

传入用户自定义的数据结构指针(Pointer)，可设 0 表示不传入。主站会记住，并在呼叫 Callback 函式时将该指标传入。

NEC_RtCyclicCallback cyclicCallback:

传入 Callback 函数指针。可设定为 0 表示不使用。此 Callback 函数在启动 EtherCAT 通讯后会被周期性的呼叫。

NEC_RtEventCallback eventCallback:

传入 Callback 函数指针。可设定为 0 表示不使用。此 Callback 函数在启动 EtherCAT 通讯后当默认事件产生后会被呼叫。

NEC_RtErrorCallback errorCallback:

传入 Callback 函数指针。可设定为 0 表示不使用。此 Callback 函数在启动通讯后当默认错误事件产生后会被呼叫。

U16_T clientID: 保留 NexECM runtime 内部使用。请勿变更此参数。

(*) 关于 Callback 函数的使用，请参考 3.5 小节的说明。

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

必须在启动 EtherCAT 通讯之前调用此函数。 若要从新注册或主站通讯结束后，必须呼叫 *NEC_RtUnregisterClient()* 取消原先的 Callback client 注册。

注意! 禁止于 Callback 函式中调用此函数

参阅::

NEC_RtStartMaster (); NEC_RtUnregisterClient();NEC_RtStopMaster();

4.2.9. NEC_RtUnregisterClient

从目标主站取消 Callback client 的注册

C/C++语法:

RTN_ERR NEC_RtUnregisterClient(U16_T MasterId, TClintParam *ClientParam);

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0

TClintParam *ClientParam: 请参阅 *NEC_RtRegisterClient*() 函数说明。

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

若要重新注册或结束应用程序前，必须呼叫 *NEC_RtUnregisterClient()* 取消原先的 Callback client 注册。一般情况下，请在 *NEC_RtWaitMasterStop*()之后取消注册。

注意! 禁止于 Callback 函式中调用此函数

参阅:

NEC_RtRegisterClient(); NEC_RtWaitMasterStop()

4.3. EC-Master 控制相关函式

4.3.1. NEC_RtStartMaster

启动目标主站 EtherCAT 通讯。

C/C++语法:

RTN_ERR NEC_RtStartMaster(U16_T MasterId);

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

注意! 禁止于 Callback 函式中调用此函数

此函数的使用时机在于使用 *NEC_RtSetParameter*() 设定目标主站参数后启动通讯，启动成功后，主站会建立周期的通讯机制。若有注册 Callback 函数 *NEC_RtRegisterClient*()，Callback 函数则在启动成功后即开始周期性的运作。停止通讯的方法:

1.在 Callback 函数中呼叫 *NEC_RtStopMasterCb*()或

2.非 Callback 函数中使用 *NEC_RtStopMaster*()

参阅:

NEC_RtSetParameter(); NEC_RtRegisterClient(); NEC_RtStopMaster(); NEC_RtStopMasterCb();**NEC_RtCyclicCallback()*

4.3.2. NEC_RtStopMaster

停止目标主站周期通讯。

C/C++语法:

RTN_ERR NEC_RtStopMaster(U16_T MasterId);

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

注意! 禁止于 Callback 函式中调用此函数

此函数用于停止目标主站 EtherCAT 通讯。在停止通讯前须将 EtherCAT 状态切换至”INIT”状态，可调用 *NEC_RtSetMasterState*()。

若要在 Callback 函数中停止 EtherCAT 通讯请参考 *NEC_RtStopMasterCb()*。

参阅::

NEC_RtStartMaster();NEC_RtStopMasterCb()

4.3.3. NEC_RtStopMasterCb

发出目标主站通讯停止要求，使用于 Callback 函数之中。

C/C++语法:

RTN_ERR NEC_RtStopMasterCb(U16_T MasterId);

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

当用户欲在 Callback 函数中停止 EtherCAT 主站通信。

成功呼叫此函数后并未立即停止通讯。它发出了一个信号来唤醒阻塞函数-*NEC_RtWaitMasterStop*()。通讯会在 *NEC_RtWaitMasterStop*()成功返回后停止，过程中 EtherCAT 主站的状态将变为“INIT”状态。

参阅:

NEC_RtStartMaster();NEC_RtStopMasterCb();NEC_RtWaitMasterStop()

4.3.4. NEC_RtWaitMasterStop

等待目标主站通讯中止要求并停止通讯，阻塞式(Blocked)函数。

C/C++语法:

RTN_ERR NEC_RtWaitMasterStop(U16_T MasterId);

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

主功能有二:

1. 让主程序进入睡眠等待
2. 收到停止讯号后，将 EtherCAT 状态机切换至”INIT”状态。

多数的情况下，用户主要将工业控制程序实作在周期的 Callback()函数之中，然而 RTOS 的应用程序为 Console 主程序的架构，主程序的进入点为 main()函数，当 main()函数返回后，RTOS 的应用程序即结束。为避免应用程序结束，用户可以呼叫此函数来等待 Callback()函数中的 *NEC_RtStopMasterCb()*所发出的停止 EtherCAT 通讯要求。

注意! 禁止于 Callback 函式中调用此函数

参阅:

NEC_RtStopMasterCb()

4.3.5. NEC_RtGetMasterState / NEC_RtSetMasterState

NEC_RtGetMasterState() : 取得目标主站目前 EtherCAT 状态(状态机)
NEC_RtSetMasterState() : 发出目标主站 EtherCAT 状态变更要求

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

参数:
U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0
U16_T *State: 回传目前状态。请参考下列定义:

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

U16_T State : 设定目标定义. 可设定范围请参考下列定义:

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

回传值:
回传错误代码。
调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:
启动 EtherCAT 通讯后，使用 *NEC_RtSetMasterState*()函式改变目标主站 EtherCAT 状态，*NEC_RtSetMasterState*()函数可在 callback 函式中呼叫，只用于发出状态改变要求，并不会等待 EtherCAT 的状态是否成功改变。*NEC_RtGetMasterState*()函数用于检查状态是否正确地完成变更。另外也可使用阻塞式函数来改变 EtherCAT 状态，请参阅 *NEC_RtSetMasterStateWait*(), *NEC_RtChangeStateToOP*()

关于 EtherCAT 状态机的说明请参考 3.6 小节 ”EtherCAT 状态机” 的说明。

参阅:

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

4.3.6. NEC_RtChangeStateToOP

阻塞式函数，切换目标主站的 EtherCAT 状态为”OP”状态

C/C++语法:

RTN_ERR NEC_RtChangeStateToOP (U16_T MasterId, I32_T TimeoutMs);

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0

I32_T TimeoutMs: 逾时等待时间，单位 millisecond。

设定 0 其效果等同 *NEC_RtSetMasterState*()，设定-1 表示永不逾时。

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

大部分的应用程序是在 EtherCAT 状态为: ”OP”状态中进行。可使用此函数来切换至”OP”状态。用法类似 *NEC_RtSetMasterState*() 启动 EtherCAT 通讯后用来改变目标主站的 EtherCAT 状态。不同之处在于可设定逾时等待，函数成功返回代表状态已成功改变至目标状态。

注意! 禁止于 Callback 函式中调用此函数

关于 EtherCAT 状态机的说明请参考 3.6 小节 ”EtherCAT 状态机”的说明。

参阅:

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

4.3.7. NEC_RtSetMasterStateWait

阻塞式变更目标主站的 EtherCAT 状态

C/C++语法:

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

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0

I32_T TimeoutMs: 逾时等待时间，单位 millisecond。

设定 0 其效果等同 *NEC_RtSetMasterState*()，设定-1 表示永不逾时。

U16_T State : 设定目标定义. 请参考下列定义:

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

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

使用此函数来切换目标主站 EtherCAT 状态。用法类似 *NEC_RtSetMasterState*() 启动 EtherCAT 通讯后用来改变目标主站状态。不同之处在于可设定逾时等待，函数成功返回代表状态已成功改变至目标状态。

注意! 禁止于 Callback 函式中调用此函数

关于 EtherCAT 状态机的说明请参考 3.6 小节 ”EtherCAT 状态机”的说明。

参阅:

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

4.3.8. NEC_RtGetStateError

读取目标主站状态错误代码

C/C++语法:

RTN_ERR NEC_RtGetStateError(U16_T MasterId, I32_T *Code);

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0

I32_T *Code: 回传错误代码，错误代码定义于 EcErrors.h 头文件中。

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

当目标主站的 EtherCAT 状态异常(如通讯断线)，状态会改变为 *ECM_STA_ERROR* 状态，目标主站会记录状态异常(错误代码)，可使用此函数读取状态异常纪录，用于除错。

参阅:

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

4.4. Process data 存取相关函式

4.4.1. NEC_RtGetProcessDataPtr

取得目标主站中 ProcessData 内存指针(Pointer)

C/C++语法:

RTN_ERR NEC_RtGetProcessDataPtr(U16_T MasterId, U8_T **InputProcessDataPtr, U32_T *InPDSizeInByte, U8_T **OutputProcessDataPtr, U32_T *OutPDSizeInByte);

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0

U8_T **InputProcessDataPtr:

回传 process data input (Slaves to master)内存指针，设定为 0 忽略

U32_T *InPDSizeInByte:

回传 process data input (Slaves to master)内存指针可存取长度，设定为 0 忽略

U8_T **OutputProcessDataPtr:

回传 process data output (Master to slaves)内存指针，设定为 0 忽略

U32_T *OutPDSizeInByte:

回传 process data output (Master to slaves)内存指针可存取长度，设定为 0 忽略

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

为提高 ProcessData 数据交换效率，使用此功能来获得内存指针，始用者程序可直接访问内部存储器。但须注意，用户必须非常小心地访问此块内存，当访问冲突可能会导致系统崩溃。直接存取 ProcessData 数据必须在 Callback 函式中进行，才能确保数据的一致性(Consistance)，相关说明请参考 3.5 callback 函式和 3.7 小节 ProcessData 存取。

参阅::

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

4.4.2. NEC_RtSetProcessDataPtrSource

设置 ProcessData 内存的来源

C/C++语法:

RTN_ERR NEC_RtSetProcessDataPtrSource(U16_T MasterId, void *InputProcessDataPtrSource, U32_T InPDSizeInByte, void *OutputProcessDataPtrSource, U32_T OutPDSizeInByte);

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0

void *InputProcessDataPtrSource:

设定 Process data input 内存由外部提供， 若设定 0 表示使用内部存储器

U32_T InPDSizeInByte:

Process data input 内存的大小，单位 Byte

void *OutputProcessDataPtrSource:

设定 Process data output 内存由外部提供， 若设定 0 表示使用内部存储器

U32_T OutPDSizeInByte:

Process data output 内存的大小，单位 Byte

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

NexECM runtime 内定是以内部存储器当作 ProcessData 来源，使用者可透过 *NEC_RtGetProcessDataPtr*()取得内部存储器指标加以存取。另一种方式为采用 *NEC_RtSetProcessDataPtrSource*()设定外部内存当作 ProcessData 的使用空间，此函式必须在启动通讯前使用，启动通讯后不能修改。当通讯停止后，ProcessData 自动恢复使用内部存储器方式，因此若需再此启动通讯，必须重新呼叫此函数从新设定。

注意! 当目标主站停止通讯后，用户的**实时应用程序卸除之前**必须呼叫 *NEC_RtSetProcessDataPtrSource*() 还原为内部存储器，以避免内存存取错误。

直接存取 ProcessData 数据必须在 Callback 函式中进行，才能确保数据的一致性(Consistance)，相关说明请参考 3.5 callback 函式和 3.7 小节 ProcessData 存取。

参阅::

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

4.4.3. NEC_RtGetSlaveProcessDataPtr

取得目标主站中指定 Slave 的 ProcessData 内存指针(Pointer)

C/C++语法:

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

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0

U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号

U8_T **InputProcessDataPtr:

回传 process data input (Slaves to master)内存指针，设定为 0 忽略

U32_T *InPDSizeInByte:

回传 process data input (Slaves to master)内存指针可存取长度，设定为 0 忽略

U8_T **OutputProcessDataPtr:

回传 process data output (Master to slaves)内存指针，设定为 0 忽略

U32_T *OutPDSizeInByte:

回传 process data output (Master to slaves)内存指针可存取长度，设定为 0 忽略

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

为提高 ProcessData 数据交换效率，使用此功能来获得内存指针，始用者程序可直接访问内部存储器。但须注意，用户必须非常小心地访问此块内存，当访问冲突可能会导致系统崩溃。直接存取 ProcessData 数据必须在 Callback 函式中进行，才能确保数据的一致性(Consistance)，相关说明请参考 3.5 callback 函式和 3.7 小节 ProcessData 存取。

参阅::

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

4.4.4. NEC_RtSetProcessDataOutput / NEC_RtGetProcessDataOutput / NEC_RtGetProcessDataInput

存取目标主站中的 ProcessData 内存

C/C++语法:

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);

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0
U32_T PDOAddr: ProcessData 内存偏移位置，单位 Byte
U32_T LengthOfData: 数据存取大小，单位 Byte
U8_T *SetData: 写入数据指针
U8_T *RetData: 读取数据指针

回传值:

回传错误代码。
调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

此函数只能用在 Callback 函数之中，用来读取或写入数据到 ProcessData 之中。
若在 Callback 函数外使用，则不保证读取或写入数据的一致性。
关于 ProcessData 存取，请参考 3.7 小节 Process Data 存取

参阅:

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

4.4.5. NEC_RtSetSlaveProcessDataOutput / NEC_RtGetSlaveProcessDataOutput / NEC_RtGetSlaveProcessDataInput

存取目标主站中 EC-Slave 的 ProcessData 资料。

C/C++语法:

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);

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0
U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号
U16_T Offset: 该 EC-Slave ProcessData 内存偏移量，从 0 开始，单位 Byte
U8_T *Data: 数据指针
U32_T Size: 数据长度，单位 Byte

回传值:

回传错误代码。
调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

此函数只能用在 Callback 函数之中，用来读取或写入数据到 EC-Slave 所占的 ProcessData 内存之中。若在 Callback 函数外使用，则不保证读取或写入数据的一致性。关于 ProcessData 存取，请参考 3.7 小节 Process Data 存取

参阅:

NEC_RtSetProcessDataOutput(); NEC_RtGetProcessDataOutput(); NEC_RtGetProcessDataInput()

4.4.6. NEC_RtSetSlaveProcessDataOutputEx / NEC_RtGetSlaveProcessDataOutputEx / NEC_RtGetSlaveProcessDataInputEx

存取目标主站 EC-Slave 的 ProcessData 资料。

C/C++语法:

C/C++Syntax :

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

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

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

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0

RwPdoData_T *PRwPdoData/*PRetRwPdoData

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

U32_T offsetBit: 指定要存取的位偏移

U32_T bitSize: 指定要存取的位数量

U8_T data[MAX_RW_PDO_DATA_LEN]: 数据储存空间

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

此函数只能用在 Callback 函数之中，用来读取或写入数据到 EC-Slave 所占的 ProcessData 内存之中。若在 Callback 函数外使用，则不保证读取或写入数据的一致性。关于 ProcessData 存取，请参考 3.7 小节 Process Data 存取

参阅:

NEC_RtSetProcessDataOutput(); NEC_RtGetProcessDataOutput();

NEC_RtGetProcessDataInput()

4.5. Callback 函式

4.5.1. *NEC_RtCyclicCallback

周期回调函数

C/C++语法:
void (*NEC_RtCyclicCallback)(void *UserDataPtr);

参数:
void *UserDataPtr: 传入用户所定义的数据指针

回传值:
void 无回传值

用法:
启动通讯前必须使用 *NEC_RtRegisterClient*()将此 Callback 函数注册到 NexECM runtime 之中。启动通讯后此函数周期性的被调用，并传入用户的数据指针。结束通讯后使用 *NEC_RtUnregisterClient*()取消 Callback 函数的注册。避免在通讯过程中取消注册。

(*) 关于 Callback 函数的使用，请参考 3.5 小节”Callback 函式”的说明。

参阅:
NEC_RtRegisterClient();NEC_RtUnregisterClient();

4.5.2. *NEC_RtEventCallback

事件回调函数

C/C++语法:

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

参数:

void *UserDataPtr: 传入用户的数据指针

U32_T EventCode: 传入事件代码，请参考用法说明。

回传值:

void 无回传值

用法:

启动通讯前必须使用 *NEC_RtRegisterClient*()将此 Callback 函数注册到 NexECM runtime 之中。启动通讯后当下表示件发生时，此 Callback 函数会被呼叫，用户可在 Callback 函数中撰写事件处理程序。

(*)事件代号定义在 RtDataStructDef.h 之中

事件代号	说明
EVENT_ECM_STATE_CHANGE	主站状态改变事件
EVENT_ECM_CNECT_FINISH	所有 EC-Slaves 的状态“Operational” state.

结束通讯后使用 *NEC_RtUnregisterClient*()取消 Callback 函数的注册。避免在通讯过程中取消注册。

(*) 关于 Callback 函数的使用，请参考 5.5 小节”Callback 函式”的说明。

参阅:

NEC_RtRegisterClient();NEC_RtUnregisterClient();

4.5.3. *NEC_RtErrorCallback

错误事件回调函数

C/C++语法:
void (*NEC_RtErrorCallback) (void *UserDataPtr, I32_T ErrorCode);

参数:
void *UserDataPtr: 传入用户的数据指针
I32_T ErrorCode: 错误代码，定义于 `EcErrors.h` 头文件中

回传值:
void 无回传值

用法:
启动通讯前必须使用 *NEC_RtRegisterClient*()将此 Callback 函数注册到 NexECM runtime 之中。启动通讯后当错误事件产生，此 Callback 函数会被呼叫，用户可在 Callback 函数中撰写错误事件处理程序。

结束通讯后使用 *NEC_RtUnregisterClient*()取消 Callback 函数的注册。避免在通讯过程中取消注册。

(*) 关于 Callback 函数的使用，请参考 5.5 小节”Callback 函式”的说明。

参阅:
NEC_RtRegisterClient(); NEC_RtUnregisterClient();

4.6. ENI 相关函式

4.6.1. NEC_RtLoadNetworkConfig

加载 ENI 档案数据至目标主站

C/C++语法:

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

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0
const char *ConfigurationFile: ENI 档案路径 C-style 字符串
U32_T Option: 加载选项。

Bit 号	31 ~ 1	0
说明	保留(设定为 0)	网络卡(NIC)选择 0:使用 ENI 设定 1:使用内部参数

当 Bit 0 设定为 1 时，目标主站所使用的网络卡(NIC)由参数设定，请参考 *RtSetParameter*().

回传值:

回传错误代码。
调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

此函数一般用在 *ResetEcMaster*()之后，本函式用于加载目前 EtherCAT 网络组态信息(ENI) XML 档案。ENI 档案必须符合 ETG.2100 规范。ENI 档案可由 NexECM Studio 工具程序产生(请参阅 NexECM Studio 使用手册)

参阅:

NEC_ResetEcMaster()

4.6.2. NEC_RtGetSlaveCount

从目标主站读取 EC-Slaves 个数

C/C++语法:

RTN_ERR NEC_RtGetSlaveCount(U16_T MasterId, U16_T *Count);

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0

U16_T *Count: 回传总 EC-Slaves 数量

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

当 ENI 下载至目标主站后，使用此函数读取 EC-Slave 的个数，此信息来自 ENI 的内容。

参阅:

<无参阅>

4.6.3. NEC_RtGetSlaveInfomation

从目标主站，读取某个 EC-Slaves 的信息

C/C++语法:

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

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0

U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号

SLAVE_INFO *pSlaveInfo: 回传 Slave 结构信息，数据结构定义如下:

```
结构定义于：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)
     注意：当Slave的状态由INIT转换至PREOP时DL_status的值才会被更
新
    U16_T res: 保留内部使用
```

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

当 ENI 下载至目标主站后，使用此函数读取 EC-Slave 的相关信息，此信息来自 ENI 的内容(DL_Status 除外)。

参阅:

<无参阅>

4.6.4. NEC_RtGetSlaveState

从目标主站，读取多个 EC-Slaves 的状态

C/C++语法:

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

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0

U16_T SlaveIndex: 指定目标 EC-Slave 代号，从 0 开始依序增号，起始的 Slave 代号

U8_T *StateArr: 回传的状态数组: 状态值定义如下:

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:传入的状态数组大小, 回传实际有效的数组大小

如果传入的数组大小 < (小于) 实际 Slave 的个数，会以传入的大小为主。

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

本函式可一次读回多个 slave 的状态，例如:

```
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  );
```

参阅:

<无参阅>

4.7. CoE 通讯相关函式

4.7.1. NEC_RtStartSDODownload / NEC_RtStartSDOUpload

从目标主站，要求存取 EC-Slaves CoE 参数

C/C++语法:

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

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

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0

U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号

TSDO *HSdo: SDO 结构数据指针

```
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, 请设定为 0

U8_T *dataPtr: Download 或 Upload 的数据指针

U16_T dataLenByte: 数据长度

U16_T status: SDO 传输的状态

- SDO_INIT (0) // SDO 通讯中
- SDO_SUCCESS (1) // SDO 通讯完成
- SDO_FAIL (2) // SDO 通讯失败, EC-Master 回传 Error code
- SDO_ABORT (3) // SDO 通讯失败, EC-Slave 回传 Abort code

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

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代

码，错误代码定义于EcErrors.h头文件中。

用法:

本函数用于送出一个 SDO Download 或 SDO Upload 的要求，送出后函数立即返回。此函数适用于 Callback 函数之中。

SDO 命令一般需要数个通讯周期的时间来完成，使用者必须检查 SDO 的传输状态(TSDO.status)是否由 SDO_INIT(0)值改变成非(0)SDO_INIT 值。

参阅:

NEC_RtSDODownload(); NEC_RtSDOUpload( ); NEC_RtSDODownloadEx(); NEC_RtSDOUploadEx()

4.7.2. NEC_RtSDODownload / NEC_RtSDOUpload / NEC_RtSDODownloadEx / NEC_RtSDOUploadEx

存取 EC-Slaves CoE 参数

C/C++语法:

```
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 );
```

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0

U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号

TSDO *HSdo: SDO 结构数据指针。请参阅 6.7.1 *NEC_RtStartSDODownload()*参数说明

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

注意! 禁止于 Callback 函式中调用此函数

本函数用于完成一个 SDO Download 或 SDO Upload 的要求，当函数成功返回表示 SDO 传输已经完成。由于 SDO 命令一般需要数个通讯周期的时间来完成，此函数禁止使用于 Callback 函数之中，避免程序死结(Dead-Locked)

参阅:

NEC_RtStartSDODownload();NEC_RtStartSDOUpload()

4.7.3. NEC_RtStartGetODListCount

读取目标主站中，特定 EC-Slave 五个对象字典(Object Dictionary)种类个别数量

C/C++语法:

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

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0
U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号
TCoEODListCount * pCoeOdListCount: 结构数据指针

TCoEODListCount 数据结构

```
typedef struct
{
    U16_T numOfAllObj;
    U16_T numOfRxPdoObj;
    U16_T numOfTxPdoObj;
    U16_T numOfBackupObj;
    U16_T numOfStartObj;
    U16_T status;
    I32_T abortCode;
} TCoEODListCount;
```

U16_T numOfAllObj: 所有对象字典数量.
U16_T numOfRxPdoObj: 可映像的 RxPDO 字典数量.
U16_T numOfTxPdoObj: 可映像的 TxPDO 字典数量.
U16_T numOfBackupObj: 可储存的字典数量.
U16_T numOfStartObj: 开机加载的字典数量.
U16_T status: SDO 传输的状态

- SDO_INIT (0) // SDO 通讯中
- SDO_SUCCESS (1) // SDO 通讯完成
- SDO_FAIL (2) // SDO 通讯失败, EC-Master 回传 Error code
- SDO_ABORT (3) // SDO 通讯失败, EC-Slave 回传 Abort code

I32_T abortCode: SDO abort code 或 EC-Master error code.

回传值:

回传错误代码。
调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代

码，错误代码定义于EcErrors.h头文件中。

用法:

本函数用于送出一个 SDO information，取得 EC-Slave 五个对象字典种类个别数量，送出后函数立即返回。此函数适用于 Callback 函数之中。
SDO information 命令一般需要数个通讯周期的时间来完成，使用者必须检查 SDO 的传输状态(TCoEODListCount.status)是否由 SDO_INIT(0)值改变成非(0)SDO_INIT 值。

参阅:

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

4.7.4. NEC_RtStartGetODList

读取目标主站中，特定 EC-Slaves 的各种类的对象字典(Object Dictionary)索引值(index)

C/C++语法:

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

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0
U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号
TCoEODList * pCoeOdList: 结构数据指针

TCoEODList 数据结构

```
typedef struct
{
    U16_T listType;
    U16_T lenOfList;
    U16_T *plistData;
    U16_T status;
    I32_T abortCode;
} TCoEODList;
```

U16_T listType: 指定对象字典种类.
U16_T lenOfList: 指定回传数据指针数组长度.
U16_T *plistData: 指定回传数据指针.
U16_T status: SDO 传输的状态.

- SDO_INIT (0) // SDO 通讯中
- SDO_SUCCESS (1) // SDO 通讯完成
- SDO_FAIL (2) // SDO 通讯失败, EC-Master 回传 Error code
- SDO_ABORT (3) // SDO 通讯失败, EC-Slave 回传 Abort code

I32_T abortCode: SDO abort code 或 EC-Master error code.

回传值:

回传错误代码。
调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

本函数用于送出一个 SDO information，取得指定对象字典种类所有对象索引值(Index)送出后函数立即返回。此函数适用于 Callback 函数之中。
SDO information 命令一般需要数个通讯周期的时间来完成，使用者必须检查 SDO 的传输状态(TCoEODList.status)是否由 SDO_INIT(0)值改变成非(0)SDO_INIT 值。

参阅:

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

4.7.5. NEC_RtStartGetObjDesc

读取目标主站中，特定 EC-Slavea 单一对象的 Object Description 信息

C/C++语法:

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

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0

U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号

TCoEObjDesc *pCoeObjDesc: 结构数据指针

TCoEObjDesc数据结构

```
typedef struct
{
    U16_T index;
    U16_T dataType;
    U8_T maxNumOfSubIndex;
    U8_T objectCode;
    U16_T sizeOfName;
    U8_T *pName;
    U16_T reserved;
    U16_T status;
    I32_T abortCode;
} TCoEObjDesc;
```

U16_T index: 指定对象索引值

U16_T dataType: 回传对象数据型别

U8_T maxNumOfSubIndex: 回传对象最大子对象

U8_T objectCode: 回传对象数据形式

U16_T sizeOfName: 指定回传数据指针数组长度

U8_T *pName: 指定回传数据指针

U16_T reserved: 保留

U16_T status: SDO 传输的状态

- SDO_INIT (0) // SDO 通讯中
- SDO_SUCCESS (1) // SDO 通讯完成
- SDO_FAIL (2) // SDO 通讯失败, EC-Master 回传 Error code
- SDO_ABORT (3) // SDO 通讯失败, EC-Slave 回传 Abort code

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

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

本函数用于送出一个 SDO information，取得指定对象 Object Description 信息，送出后函数立即返回。此函数适用于 Callback 函数之中。

SDO information 命令一般需要数个通讯周期的时间来完成，使用者必须检查 SDO 的传输状态(TCoEObjDesc.status)是否由 SDO_INIT(0)值改变成非(0)SDO_INIT 值。

参阅:

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

4.7.6. NEC_RtStartGetEntryDesc

读取目标主站中，特定读取 EC-Slave 的单一对象 Entry Description 信息

C/C++语法:

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

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0
U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号
TCoEEntryDesc * pCoeEntryDesc: 结构数据指针

TCoEEntryDesc 数据结构

```
typedef struct
{
    U16_T index;
    U8_T subIndex;
    U8_T valueInfo;
    U16_T dataType;
    U16_T bitLength;
    U16_T objectAccess;
    U16_T sizeOfData
    U8_T *pData;
    U16_T status;
    I32_T abortCode;
} TCoEEntryDesc;
```

U16_T index: 指定对象索引值
U8_T subIndex: 指定对象子索引值
U8_T valueInfo: 指定回传信息(一般设 0)
U16_T dataType: 回传对象数据型别
U16_T bitLength: 回传对象数据长度
U8_T objectAccess: 回传物存取属性
U16_T sizeOfData: 指定回传数据指针数组长度
U8_T *pData: 指定回传数据指针
U16_T status: SDO 传输的状态

- SDO_INIT (0) // SDO 通讯中
- SDO_SUCCESS (1) // SDO 通讯完成
- SDO_FAIL (2) // SDO 通讯失败, EC-Master 回传 Error code
- SDO_ABORT (3) // SDO 通讯失败, EC-Slave 回传 Abort code

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

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

本函数用于送出一个 SDO information，取得指定对象 Entry Description 信息，送出后函数立即返回。此函数适用于 Callback 函数之中。

SDO information 命令一般需要数个通讯周期的时间来完成，使用者必须检查 SDO 的传输状态(TCoEEntryDesc.status)是否由 SDO_INIT(0)值改变成非(0)SDO_INIT 值。

参阅:

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

4.7.7. NEC_RtGetEmgDataCount

读取目标主站中 Emergency 数据数量

C/C++语法:

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

参数:
U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0
U16_T *pEmgDataCount: 数据回传指针

回传值:
回传错误代码。
调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:
本函数用于取得目标主站 Emergency 数据数量。此函数适用于 Callback 函数之中。

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

4.7.8. NEC_RtGetEmgData

读取目标主站中 Emergency 数据

C/C++语法:

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

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0

TEmgData _T *pEmgData: 数据回传结构指针

TEmgData 数据结构

```
typedef struct
{
    U16_T lenOfData;
    U16_T res;
    U16_T *pSlaveAddrDataArr;
    TCoEEmgMsg *pEmgMsgDatArr;
} TEmgData;;
```

U16_T lenOfData: 指定数据回传数组指针长度.

U16_T res : 保留

U16_T *pSlaveAddrDataArr: 数据回传结构指针

TCoEEmgMsg *pEmgMsgDataArr: 数据回传结构指针

TCoEEmgMsg 数据结构

```
typedef struct
{
    U16_T errorCode;
    U8_T errorRegister;
    U8_T data[5];
} TCoEEmgMsg;
```

U16_T errorCode: 回传错误码

U8_T errorRegister: 回传错误缓存器

U8_T data[5]: 回传数据数组

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

本函数用于读取目标主站中的 Emergency 数据。此函数适用于 Callback 函数之中。

参阅:

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

4.8. 存取从站模块硬件信息相关函式

4.8.1. NEC_RtGetConfiguredAddress

从目标主站，取得目标从站模块”Configured Station Address”

C/C++语法:

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

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0
U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号
U16_T *pConfigAddr: 写入数据指针

回传值:

回传错误代码。
调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

注意! 禁止于 Callback 函式中调用此函数

此函数用于取得指定从站模块的 Configured Station Address 信息。例如，使用者可经由底下程序代码，得到从站模块代号 0 其 Configured Station Address

```
U16_T SlaveAddr = 0;     // The first slave
U16_T ConfigAddr = 0;    // A variable for storing the configured address
NEC_RtGetConfiguredAddress ( MasterId,    SlaveAddr,    &ConfigAddr );
```

参阅:

4.8.2. NEC_RtGetAliasAddress

从目标主站，取得目标从站模块”Configured Station Alias”

C/C++语法:

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

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0
U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号
U16_T *pAliasAddr: 写入数据指针

回传值:

回传错误代码。
调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:

注意! 禁止于 Callback 函式中调用此函数

此函数用于取得指定从站模块的 Configured Station Alias 信息。例如，使用者可经由底下程序代码，得到 Configured Station Alias 为 0x0021 其模块代号

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

```
        }
    }
```

参阅:

4.8.3. NEC_RtGetSlaveCoeProfileNum

从目标主站，取得目标从站模块”CoeProfileNum”信息

C/C++语法:

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

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0
U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号
U32_T * pCoeProfileNum: 写入数据指针

回传值:

回传错误代码。
调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:

此函数用于取得指定从站模块的 CoeProfileNum 信息。例如，用户可经由底下程序代码，得到网络上那些模块为驱动器(CoeProfileNum 等于 402)

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

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

参阅:

4.8.4. NEC_RtGetSlaveConfiguredAddressEni

从目标主站，取得目标从站模块“Config Address “信息

C/C++语法:

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

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0
U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号
U32_T * PRetConfigAddr: 返回 Slave Configure Address 信息

回传值:

回传错误代码。
调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:

此函数用于取得指定从站模块的 Configure Address 信息。

参阅:

4.8.5. NEC_RtGetSlaveTimeDiffWithRefSlave

从目标主站，取得目标从站模块“Config Address “信息

C/C++语法:

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

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0
U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号
U32_T * PRetTimeDiffNs: 返回 time different 信息

回传值:

回传错误代码。
调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:

此函数用于取得指定从站模块的 time different 信息。 (register: 0x092c)

参阅:

4.8.6. NEC_RtGetSlaveCountEx

从目标主站，取得当前从站模块数量信息

C/C++语法:

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

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0
U32_T * PRetCount: 返回 Slave 数量信息

回传值:

回传错误代码。
调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:

此函数用于取得从站模块数量信息。

参阅:

4.8.7. NEC_RtGetSlaveVendorId

从目标主站，取得当前从站的 VendorId 信息

C/C++语法:

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

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0
U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号
U32_T * PRetVendorId: 返回 Slave VendorId 信息

回传值:

回传错误代码。
调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:

此函数用于取得指定从站模块的 VendorId 信息。

参阅:

4.8.8. NEC_RtGetSlaveProductCode

从目标主站，取得当前从站的 Product Code 信息

C/C++语法:

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

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0
U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号
U32_T * PRetProductCode: 返回 ProductCode 信息

回传值:

回传错误代码。
调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:

此函数用于取得指定从站模块的 Product Code 信息。

参阅:

4.8.9. NEC_RtGetSlaveRevisionNo

从目标主站，取得当前从站的 Revision Number 信息

C/C++语法:

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

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0
U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号
U32_T * PRetRevisionNo: 返回 Slave Revision Number 信息

回传值:

回传错误代码。
调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:

此函数用于取得指定从站模块的 Revision Number 信息。

参阅:

4.9. 效能监控

4.9.1. NEC_RtGetCyclicTick

当前主站 cyclic 运行次数

C/C++语法:

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

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0

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

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

此函数用于取得指定主站cyclic运行次数。

参阅:

4.9.2. NEC_RtGetCyclicTime

当前主站 cyclic 运行时间

C/C++语法:

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

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0

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

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

此函数用于取得指定主站cyclic运行时间。

参阅:

4.9.3. NEC_RtGetLastCyclicLoad

取得当前主站 cyclic callback 运行时间(us)

C/C++语法:

RTN_ERR FNTYPE NEC_RtGetLastCyclicLoad(U16_T MasterId, U32_T *PRetLoadingUs);

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0

U32_T *PRetLoadingUs: cyclic callback 运行时间

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

此函数用于取得指定主站cyclic callback运行时间，注意该时间若超过主站周期时间，表示over load。

参阅:

4.9.4. NEC_RtIsCyclicOverLoad

从目标主站，取得当前主站运行是否 OverLoad(Callback 运行时间 > 主站 Cycle time)

C/C++语法:

BOOL_T FNTYPE NEC_RtIsCyclicOverLoad(U16_T MasterId);

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0

回传值:

TRUE: 主站 OverLoad

FALSE: 主站无 OverLoad

回传其他错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

此函数用于取得当前主站运行是否 OverLoad(Callback 运行时间 > 主站 Cycle time)

参阅:

4.9.5. NEC_RtResetCyclicOverLoad

重设目标主站 OverLoad 状态

C/C++语法:

RTN_ERR FNTYPE NEC_RtResetCyclicOverLoad(U16_T MasterId);

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

此函数用于重设指定主站OverLoad状态。

参阅:

4.9.6. NEC_RtGetCyclicOverLoadCount

取得当前主站 OverLoad 发生次数

C/C++语法:

RTN_ERR FNTYPE NEC_RtGetCyclicOverLoadCount(U16_T MasterId, U32_T *PRetOverLoadCnt);

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0

U32_T *PRetOverLoadCnt: OverLoad 发生次数

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

此函数用于取得当前主站 OverLoad 发生次数。

参阅:

4.9.7. NEC_RtClearProbe

初始所有目标主站时间量测参数

C/C++语法:

RTN_ERR FNTYPE NEC_RtClearProbe(U16_T MasterId);

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

初始所有主站时间量测参数。

参阅:

4.9.8. NEC_RtGetMasterCycleConsumption

从目标主站，取得 Cyclic Callback 运行信息

C/C++语法:

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

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0
U32_T *PRetTotalCnt: Cyclic Callback 运行次数
U32_T *PRetMaxConsumptionTime_US: Cyclic Callback 最大运行时间
U32_T *PRetMinConsumptionTime_US: Cyclic Callback 最小运行时间
U32_T *PRetAvgConsumptionTime_NS: Cyclic Callback 平均运行时间

回传值:

回传错误代码。
调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

从目标主站，取得 Cyclic Callback 运行信息

参阅:

4.9.9. NEC_RtGetMasterActualCycleTime

从目标主站，取得主站周期运行信息

C/C++语法:

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

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0
U32_T *PRetTotalCnt: 主站周期运行次数
U32_T *PRetMaxCycleTime_US: 主站周期最大运行时间
U32_T *PRetMinCycleTime_US: 主站周期最小运行时间
U32_T *PRetAvgCycleTime_NS: 主站周期平均运行时间

回传值:

回传错误代码。
调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

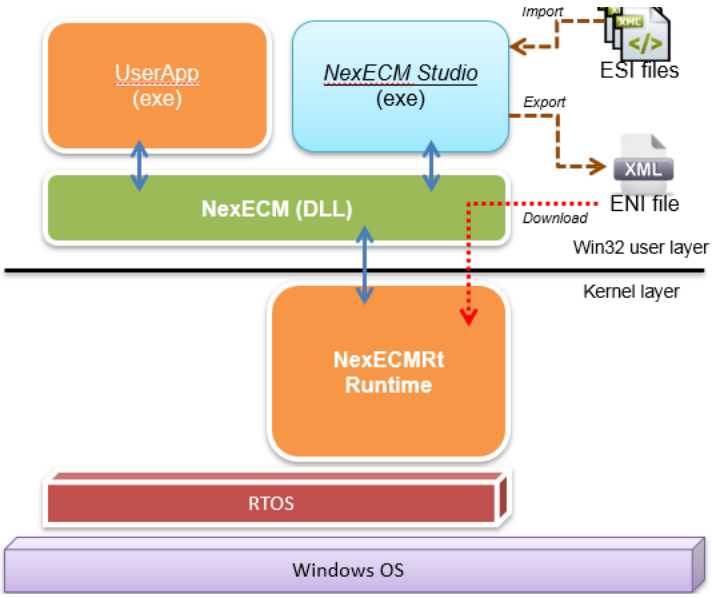
用法:

从目标主站，取得主站周期运行信息

参阅:

5. NexECM 非实时(Non-real-time)函式库(Win32 Library)

用户的 Win32 应用程序(如下图中 UserApp.exe)可透过 NexECM 动态链接库(Win32 Dynamicaly linked library, DLL), 来控制执行于 RTOS 环境下的 NexECM runtime。用户的程序若不需具有实时的特性，可单纯透过 NexECM 非实时函式库控制 EC-Slave，不需 RTOS 的 SDK 用以开发运行于实时环境的程序。



NexECM 函式库主要提供下列功能:

1. 加载实时程序至 RTOS 环境
2. 载入(移除)NexECM runtime 至(从)RTOS 环境
3. ProcessData 存取(透过 NexECM runtime)
4. CoE SDO 存取(透过 NexECM runtime)
5. CiA402 APIs，请参考 CiA402 APIs 使用手册

5.1. Win32 API 总览

下表列出 NexECM 函式库 API 的列表，API 定义于 NexECM.h 头文件之中。

函数名称	说明	
系统相关控制 API		
NEC_GetVersion NEC_GetVersionEx	读取NexECM 函式库版本号码	
NEC_StartDriver	初始化NexECM函式库	
NEC_StopDriver	关闭NexECM函式库	
NEC_LoadRtMaster	加载NexECMRt runtime至RTOS环境	
NEC_UnLoadRtMaster	从RTOS环境卸除NexECMRt runtime	
NEC_LoadRtApp	加载实时程序至RTOS环境	
NexECM Runtime 控制相关函式		
NEC_GetRtMasterId	自动侦测NexECMRt runtime并取得目标主站代号	
NEC_ResetEcMaster	重置目标主站	
NEC_LoadNetworkConfig	载入ENI档案	
NEC_StartNetwork	启动EtherCAT通讯	
NEC_StartNetworkEx	启动EtherCAT通讯附带Option选项	
NEC_StopNetwork	停止EtherCAT周期通讯	
NEC_SetParameter	设置目标主站参数	
NEC_GetParameter	读取目标主站参数	
NEC_GetMasterCount	取得系统主站数量	
NEC_GetMasterType	取得目标主站的种类	
网络状态存取相关函式		
NEC_GetSlaveCount	读取EC-Slaves 个数	
NEC_GetNetworkState	读取主站当前状态	
NEC_GetSlaveState	读取主站当前状态	
NEC_GetStateError	读取主站错误状态	
NEC_GetErrorMsg	读取主站错误讯息字符串	
DIO 控制相关函式		
NEC_SetDo	设定EC-Slave输出值	
NEC_GetDo	取得EC-Slave输出设定值	
NEC_GetDi	读取EC-Slave输入值	
NEC_SetDo_s	设定Ec-Slave输出值，并等待完成	

CoE 通讯相关函式		
NEC_RtStartSDODownload	送出 SDO download 命令数据(Master to slave)	
NEC_RtStartSDOUpload	送出 SDO upload 命令数据(Slave to Master)	
NEC_SDODownloadEx	送出 SDO download 命令数据(Master to slave)	
NEC_SDOUploadEx	送出 SDO upload 命令数据(Slave to Master)	
NEC_SDODownload	执行 SDO download (Structure)	
NEC_SDOUpload	执行 SDO upload (Structure)	
NEC_RtStartGetODListCount	读取 EC-Slave 五个对象字典(Object Dictionary)种类各别数量	
NEC_RtStartGetODList	读取 EC-Slaves 的各种类的对象字典(Object Dictionary)索引值(index)	
NEC_RtStartGetObjDesc	读取 EC-Slavea 单一对象的 Object Description 信息	
NEC_RtStartGetEntryDesc	读取 EC-Slavea 的单一对象 Entry Description 信息	
NEC_RtGetEmgDataCount	读取目标主站中 Emergency 数据数量	
NEC_RtGetEmgData	读取目标主站中 Emergency 数据	
Process data 存取相关函式		
NEC_RWProcessImage	存取目标主站的 ProcessData 数据	
NEC_GetProcessImageSize	取得目标主站的 ProcessData 内存长度	
NEC_RWSlaveProcessImage	存取 EC-Slave 的 ProcessData 内存	
NEC_RWProcessImageEx	存取目标主站的 ProcessData 数据，支持位型式	
NEC_RWSlaveProcessImageEx	存取 EC-Slave 的 ProcessData 内存，支持位型式	
存取从站模块硬件信息相关函式		
NEC_GetConfiguredAddress	取得从站 Configured Station Address	
NEC_GetAliasAddress	取得从站 Configured Station Alias	
NEC_GetSlaveCoeProfileNum	取得从站 CoeProfileNum	

API 所使用的 C/C++数据型态定义于 nex_type.h 中，说明如下表:

型别	C/C++ 原型	说明	大小 byte	范围
BOOL_T	int	布尔型别	4	0:False, 1:True

U8_T	unsigned char	无号整数	1	0 ~ 255
U16_T	unsigned short	无号整数	2	0 ~ 65535
U32_T	unsigned int	无号整数	4	0 ~ 4294967295
U64_T	unsigned __int64	无号整数	8	0 ~ 18446744073709551615
I8_T	char	有号整数	1	-128 ~ 127
I16_T	short	有号整数	2	-32768 ~ 32767
I32_T	int	有号整数	4	-2147483648 ~ 2147483647
I64_T	__int64	有号整数	8	-9223372036854775808 ~ 9223372036854775807
F32_T	float	浮点数	4	IEEE-754，有效小数后 7 位
F64_T	double	双精浮点数	8	IEEE-754，有效小数后 15 位
RTN_ERR	int	错误代码	4	-2147483648 ~ 2147483647

5.2. NexECM 初始化相关函式

5.2.1. NEC_GetVersion / NEC_GerVersionEx

读取 NexECM 函式库 (NexECM.dll) 版本号码

C/C++语法:

```
U32_T NEC_GetVersion();

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

参数:

_opt_null_ I32_T *PRetMajor:

指针变量，回传版本 Major 信息

_opt_null_ I32_T *PRetMinor:

指针变量，回传版本 Minor 信息

_opt_null_ I32_T *PRetStage:

指针变量，回传版本 State 信息

_opt_null_ I32_T *PRetBuild:

指针变量，回传版本 Build 信息

回传值:

回传完整NexECM版本号码

用法:

读取 NexECM 函式库版本号码

参阅:

<无参阅>

5.2.2. NEC_StartDriver

初始化 NexECM 函式库

C/C++语法:
RTN_ERR NEC_StartDriver();

参数:
<无参数>

回传值:
回传错误代码。
调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:
在使用 NexECM 函式库中所有函式(API)前，需先呼叫此函数进行初始化。本函数亦同时侦测 NexECMRt runtime 是否加载，因此必须确定 NexECMRt runtime 是否已经被加载到 RTOS 系统中。载入 NexECMRt runtime 可参考 *NEC_LoadRtMaster* ()

参阅:
NEC_LoadRtMaster();NEC_StopDriver();

5.2.3. NEC_StopDriver

关闭 NexECM 函式库

C/C++语法:
void NEC_StopDriver();

参数:
<无参数>

回传值:
回传错误代码。
调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:
通常在 Win32 应用程序结束前呼叫本函数来释放系统资源。

参阅:
NEC_StartDriver()

5.2.4. NEC_LoadRtMaster / NEC_UnLoadRtMaster

加载/卸除 NexECM runtime 至/从 RTOS 环境

C/C++语法:
RTN_ERR NEC_LoadRtMaster()

RTN_ERR NEC_UnLoadRtMaster()

参数:
<无参数>

回传值:
回传错误代码。
调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:
通常在 Win32 应用程序呼叫其它含式时，呼叫 NEC_LoadRtMaster 加载 NexECM runtime 至 RTOS。在 Win32 应用成式结束前，呼叫 NEC_UnLoadRtMaster 自 RTOS 环境卸除 NexECMRt runtime。

参阅:
<无>

5.2.5. NEC_LoadRtApp

加载实时程序至 RTOS 环境下

C/C++语法:

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

参数:

U16_T RtAppType:指定实时应用程序种类

实时程序种类	Value
RT_APP_TYPE_RTX	1
RT_APP_TYPE_INtime	2

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

用户可透过此函式将实时应用程序加载至 RTOS 环境下。

参阅:

<无>

5.3. NexECM Runtime 控制相关函式

5.3.1. NEC_GetRtMasterId

侦测 NexECMRt runtime 是否已加载并取得目标主站所属的控制代号

C/C++语法:

RTN_ERR NEC_GetRtMasterId(U16_T *PRetMasterId);

参数:

U16_T *PRetMasterId: 回传目标主站控制代号(MasterID)

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

Win32 NexECM在操作(或控制) NexECMRt runtime之前必须先使用本函数取得主站控制代号(MasterID)，若本函数返回错误代码(ECERR_DRIVER_NOT_FOUND)表示NexECMRt runtime尚未加载至RTOS系统中。

参阅:

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

5.3.2. NEC_ResetEcMaster

重置目标主站

C/C++语法:

RTN_ERR NEC_ResetEcMaster(U16_T MasterId);

参数:

U16_T MasterId: 指定目标主站代号。代号可由 *NEC_GetRtMasterId*()取得

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

本函数用来重置目标主站。一般使用本函数的时机是在 *NEC_GetRtMasterId*()取得目标主站代号后，载入 ENI 档案前。本函数将清除目标主站内的所有变量包含 ENI 信息。

参阅:

NEC_GetRtMasterId((); NEC_LoadNetworkConfig()

5.3.3. NEC_LoadNetworkConfig

加载 ENI 档案至目标主站

C/C++语法:

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

参数:

U16_T MasterId: 指定目标主站代号；代号可由 NEC_GetRtMasterId()取得

const char *ConfigurationFile: ENI 档案路径 C-style 字符串

U32_T Option: 加载选项。

Bit 号	31 ~ 1	0
说明	保留(设定为 0)	网络卡(NIC)选择 0:使用 ENI 设定 1:使用内部参数

当 Bit 0 设定为 1 时，目标主站所使用的网络卡(NIC)由参数设定，请参考 *RtSetParameter*().

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

此函数一般用在 *ResetEcMaster*()之后，本函式用于加载目前 EtherCAT 网络组态信息(ENI) XML 档案。ENI 档案必须符合 ETG.2100 规范。ENI 档案可由 NexECM Studio 工具程序产生(请参阅 NexECM Studio User Manual)

参阅:

NEC_ResetEcMaster()

5.3.4. NEC_StartNetwork

启动 EtherCAT 通讯

C/C++语法:

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

参数:

U16_T MasterId: 指定目标主站代号；代号可由 NEC_GetRtMasterId()取得
const char *ConfigurationFile: ENI 档案路径 C-style 字符串
I32_T TimeoutMs : 逾时等待时间，单位 millisecond

回传值:

回传错误代码。
调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

此函数的使用时机在于使用 *NEC_SetParameter*() 设定目标主站参数后启动通讯，启动成功后，目标主站会建立周期的通讯机制。呼叫 *NEC_StopNetwork*()停止通讯 EtherCAT 通讯。

参阅:

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

5.3.5. NEC_StartNetworkEx

启动 EtherCAT 通讯；此 API 同 NEC_StartNetwork，多了一个设定参数。

C/C++语法:

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

参数:

U16_T MasterId: 指定目标主站代号；代号可由 NEC_GetRtMasterId()取得

const char *ConfigurationFile: ENI 档案路径 C-style 字符串

U32_T Option: 启动选项。

Bit 号	31 ~ 1	0
说明	保留(设定为 0)	网络卡(NIC)选择 0:使用 ENI 设定 1:使用内部参数

当 Bit 0 设定为 1 时，目标主站所使用的网络卡(NIC)由参数设定，请参考 *RtSetParameter*().

I32_T TimeoutMs : 逾时等待时间，单位 millisecond

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

此函数的使用时机在于使用 NEC_SetParameter() 设定目标主站参数后启动通讯，启动成功后，目标主站会建立周期的通讯机制。呼叫 NEC_StopNetwork 停止通讯 EtherCAT 通讯。

参阅:

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

5.3.6. NEC_StopNetwork

停止 EtherCAT 通讯

C/C++语法:

RTN_ERR NEC_StopNetwork(U16_T MasterId, I32_T TimeoutMs);

参数:

U16_T MasterId: 指定目标主站代号；代号可由 NEC_GetRtMasterId()取得

I32_T TimeoutMs : 逾时等待时间，单位 millisecond

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

此函数用于停止 EtherCAT 周期通讯。停止通讯过程中，目标主站会切回 ”INIT” 状态。

参阅:

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

NEC_StartNetworkEx()

5.3.7. NEC_SetParameter / NEC_GetParameter

这两个函数用于设置和读取目标主站参数。

C/C++语法:

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);

参数:

U16_T MasterId: 指定目标主站代号；代号可由 NEC_GetRtMasterId()取得
U16_T ParaNum: 指定参数代码，请参考**用法**:
I32_T ParaData: 指定参数值，请参考**用法**:
I32_T *ParaData: 回传参数值，请参考**用法**:

回传值:

回传错误代码。
调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

用于启动 EtherCAT 通讯之前，设定目标主站通讯使用的参数，其参数列表如下。

参数代码	说明	参数值
NEC_PARA_S_ECM_CYCLETIMEUS	通讯周期，单位 micro-second	250 ~ 1000000
NEC_PARA_S_NIC_INDEX	设定 NIC Port 号码	0 ~ Max NIC port
NEC_PARA_ECM_RECV_TIMEOUT_US	定义 EtherCAT 网络封包回传 timeout 时间，单位 micro-second。一般使用默认值即可。	250 ~ 2000000
NEC_PARA_ECM_LINK_ERR_MODE	网络断线行为模式: **LINKERR_AUTO**: 当 EC-Slave(s) 断线，目标主站自动侦测断线的装置。当装置重新联机自动将 EC-Slaves 初始化并设定为”OP”状态 **LINKERR_MANUAL:** 当某装置断线，其状态会停留在 ERROR 状态。当断线装置恢复联机(实体联机)，该装置将不会被重新初始化。	LINKERR_AUTO (0) LINKERR_MANUAL (1) LINKERR_STOP (2)

	LINKERR_STOP: 只要有一装置断线，目标主站将网络中断，并进入 ERROR 状态。	
NEC_PARA_ECM_DC_CYC_TIME_MODE	DC 时间设定模式: 0: 根据 Master cycle time (预设) 1: 根据 ENI 资料	0~1

参阅:

NEC_GetRtMasterId();

5.3.8. NEC_GetMasterType

取得目标主站种类

C/C++语法:

U16_T NEC_GetMasterType(U16_T MasterId, U16_T *PRetType);

参数:

U16_T MasterId: 指定目标主站代号

U16_T *PRetType: 指针变量，回传目标主站种类，支持种类如下表:

NexECM Runtime 种类	Value
MASTER_TYPE_RTX	1
MASTER_TYPE_INtime	2

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

此函数用以取得目标主站类别，通常在载入 NexECMRt runtime 后呼叫。

参阅:

NEC_LoadRtMaster();

5.3.9. NEC_GetMasterCount

取得系统目前主站数量

C/C++语法:

RTN_ERR FNTYPE NEC_GetMasterCount(U16_T *PRetMasterCount);

参数:

U16_T *PRetMasterCount: 指针数变，回传系统内主站数量

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

此函数用以取得系统内所有主站数量，通常用于加载 NexECMRt runtime 后呼叫。

参阅:

NEC_LoadRtMaster();

5.4. 网络状态存取相关函式

5.4.1. NEC_GetSlaveCount

读取目标主站 EC-Slaves 装置数量

C/C++语法:

RTN_ERR NEC_GetSlaveCount(U16_T MasterId, U16_T *Count);

参数:

U16_T MasterId: 指定目标主站代号；代号可由 NEC_GetRtMasterId()取得

U16_T *Count: 回传 EC-Slaves 数量指标

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:

当开启加载 ENI 信息后，使用此函数读取 EC-Slave 的个数。

参阅:

NEC_GetRtMasterId()

5.4.2. NEC_GetNetworkState

读取目标主站网络联机状态

C/C++语法:

RTN_ERR NEC_GetNetworkState(U16_T MasterId, U16_T *State);

参数:

U16_T MasterId: 指定目标主站代号；代号可由 NEC_GetRtMasterId()取得

U16_T * State: 回传网络当前状态指针

请参考下列定义:

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

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:

用于启动 EtherCAT 通讯之后，轮询网络当前的状态。

参阅:

NEC_GetRtMasterId();

5.4.3. NEC_GetSlaveState

读取目标主站内，特定 Slave 联机状态

C/C++语法:

RTN_ERR NEC_GetSlaveState(U16_T MasterId, U16_T SlaveIndex, U8_T *StateArr, U16_T *ArrLen)

参数:

U16_T MasterId: 指定目标主站代号；代号可由 NEC_GetRtMasterId()取得

U16_T SlaveIndex: 指定起始的 EC-Slave 位置

U8_T *StateArr: 回传 Slave 状态值数组

U16_T *ArrLen:

Input: 指定读取 EC-Slaves 的数量，StateArr 数组大小。

Output: 返回实际读取 EC-Slaves 的数量

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:

用于启动 EtherCAT 通讯之后，轮询网络上所有 EC-Slaves 的当前状态。

参阅:

NEC_GetRtMasterId();

5.4.4. NEC_GetStateError

读取目标主站状态错误代码

C/C++语法:

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

参数:

U16_T MasterId: 指定目标主站代号；代号可由 NEC_GetRtMasterId()取得

I32_T *Code: 回传状态错误指标，错误代码定义于 EcErrors.h 头文件中

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:

用于启动 EtherCAT 通讯之后，目标主站会将状态由”INIT”切换到”OP”状态，若发生错误，则状态会被设定为”ERROR”状态，此时可使用此 API 读取目标主站状态错误代码。

参阅:

NEC_GetRtMasterId();NEC_GetErrorMsg()

5.4.5. NEC_GetErrorMsg

读取目标主站状态错误字符串

C/C++语法:

RTN_ERR NEC_GetErrorMsg(U16_T MasterId, char *ErrMsg_128_len);

参数:

U16_T MasterId: 指定目标主站代号；代号可由 NEC_GetRtMasterId()取得

char *ErrMsg_128_len: 回传状态字符串指针

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:

用于启动 EtherCAT 通讯之后，目标主站会将状态由”INIT”切换到”OP”状态，若发生错误，则状态会被设定为”ERROR”状态，此时可使用此 API 读取目标主站状态错误字符串。

参阅:

NEC_GetRtMasterId();NEC_GetStateError()

5.5. DIO 控制相关函式

5.5.1. NEC_SetDo

设定 EC-Slave Digital output 输出

C/C++语法:
RTN_ERR NEC_SetDo(U16_T MasterId, U16_T SlaveAddr, U16_T Offset, U16_T SizeByte, const U8_T *DoData)

参数:
U16_T MasterId: 指定目标主站代号；代号可由 NEC_GetRtMasterId()取得
U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号
U16_T Offset: Slave ProcessData (Output)内存位移值
U16_T SizeByte: 设定 DO 数据长度，单位 Byte
const U8_T *DoData: 指定输出数据常数指针

回传值:
回传错误代码。
调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:
用于启动 EtherCAT 通讯之后，设定 EC-Slave Digital output 输出；使用上需注意 SizeByte 与 DoData 长度。

参阅:
NEC_GetRtMasterId();

5.5.2. NEC_GetDo

读取 EC-Slave Digital output 输出

C/C++语法:

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

参数:

U16_T MasterId: 指定目标主站代号；代号可由 NEC_GetRtMasterId()取得

U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号

U16_T Offset: 内存位移值

U16_T SizeByte: 指定读取长度

U8_T *DoData: 指定读取数据指针

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:

用于启动 EtherCAT 通讯之后，读取 EC-Slave Digital output 输出；使用上需注意 SizeByte 与 DoData 长度。

参阅:

NEC_GetRtMasterId();

5.5.3. NEC_GetDi

读取 EC-Slave Digital input 输入

C/C++语法:

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

参数:

U16_T MasterId: 指定目标主站代号；代号可由 NEC_GetRtMasterId()取得

U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号

U16_T Offset: 内存位移值

U16_T SizeByte: 指定读取长度

U8_T *DiData: 指定读取数据指针

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:

用于启动 EtherCAT 通讯之后，读取 EC-Slave Digital input 输入；使用上需注意 SizeByte 与 DiData 长度。

参阅:

NEC_GetRtMasterId();

5.5.4. NEC_SetDo_s

设定 EC-Slave Digital output 输出，等待生效后返回

C/C++语法:

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

参数:

U16_T MasterId: 指定目标主站代号；代号可由 NEC_GetRtMasterId()取得

U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号

U16_T Offset: Slave ProcessData (Output)内存位移值

U16_T SizeByte: 设定 DO 数据长度，单位 Byte

const U8_T *DoData: 指定输出数据常数指针

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:

用于启动 EtherCAT 通讯之后，设定 EC-Slave Digital output 输出并等待，设定数值生效后返回；使用上需注意 SizeByte 与 DoData 长度。

参阅:

NEC_GetRtMasterId();

5.6. CoE 通讯相关函式

5.6.1. NEC_SDODownloadEx / NEC_SDOUploadEx / NEC_SDODownload / NEC_SDOUpload

C/C++语法:

```
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 );
```

参数:

U16_T MasterId: 指定目标主站代号；代号可由 NEC_GetRtMasterId()取得

U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号

TSDO *HSdo: SDO 结构数据指针。

TSDO 数据结构

```
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, 请设定为 0

U8_T *dataPtr: Download 或 Upload 的数据指针

U16_T dataLenByte: 数据长度

U16_T status: Reserved

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

回传值:

回传错误代码。
调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:
本函数用于完成一个 SDO Download 或 SDO Upload 的要求，当函数成功返回表示 SDO 传输已经完成。

参阅:
NEC_GetRtMasterId();

5.6.2. NEC_GetODListCount

读取 EC-Slave 五个对象字典(Object Dictionary)种类个别数量

C/C++语法:

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

参数:

U16_T MasterId: 指定目标主站代号；代号可由 NEC_GetRtMasterId()取得
U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号
TCoEODListCount * pCoeOdListCount: 结构数据指针

TCoEODListCount 数据结构

```
typedef struct
{
    U16_T numOfAllObj;
    U16_T numOfRxPdoObj;
    U16_T numOfTxPdoObj;
    U16_T numOfBackupObj;
    U16_T numOfStartObj;
    U16_T status;
    I32_T abortCode;
} TCoEODListCount;
```

U16_T numOfAllObj: 所有对象字典数量
U16_T numOfRxPdoObj: 可映像的 RxPDO 字典数量
U16_T numOfTxPdoObj: 可映像的 TxPDO 字典数量
U16_T numOfBackupObj: 可储存的字典数量
U16_T numOfStartObj: 开机加载的字典数量
U16_T status: Reserved
I32_T abortCode: SDO abort code 或 EC-Master error code

回传值:

回传错误代码。
调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:

本函数用于送出一个 SDO information，取得 EC-Slave 五个对象字典种类个别数量，当函数成功返回表示 SDO information 传输已经完成。

参阅:

NEC_GetODList();NEC_GetObjDesc();NEC_GetEntryDesc();

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

5.6.3. NEC_GetODList

读取 EC-Slaves 的各种类的对象字典(Object Dictionary)索引值(index)

C/C++语法:

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

参数:

U16_T MasterId: 指定目标主站代号；代号可由 NEC_GetRtMasterId()取得

U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号

TCoEODList * pCoeOdList: 结构数据指针

TCoEODList 数据结构

```
typedef struct
{
    U16_T listType;
    U16_T lenOfList;
    U16_T *plistData;
    U16_T status;
    I32_T abortCode;
} TCoEODList;
```

U16_T listType: 指定对象字典种类

U16_T lenOfList: 指定回传数据指针数组长度

U16_T *plistData: 指定回传数据指针

U16_T status: Reserved

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

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:

本函数用于送出一个 SDO information，取得指定对象字典种类所有对象索引值，当函数成功返回表示 SDO information 传输已经完成。

参阅:

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

NEC_EmgDataCount();NEC_EmgData()

5.6.4. NEC_GetObjDesc

读取 EC-Slavea 的单一对象 Object Description 信息

C/C++语法:

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

参数:

U16_T MasterId: 指定目标主站代号；代号可由 NEC_GetRtMasterId()取得

U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号

TCoEObjDesc *pCoeObjDesc: 结构数据指针

TCoEObjDesc 数据结构

```
typedef struct
{
    U16_T index;
    U16_T dataType;
    U8_T maxNumOfSubIndex;
    U8_T objectCode;
    U16_T sizeOfName;
    U8_T *pName;
    U16_T reserved;
    U16_T status;
    I32_T abortCode;
} TCoEObjDesc;
```

U16_T index: 指定对象索引值

U16_T dataType: 回传对象数据型别

U8_T maxNumOfSubIndex: 回传对象最大子对象

U8_T objectCode: 回传对象数据形式

U16_T sizeOfName: 指定回传数据指针数组长度

U8_T *pName: 指定回传数据指针

U16_T reserved: 保留

U16_T status: Reserved

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

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:

本函数用于送出一个 SDO information，取得指定对象 Object Description 信息，当函数成功返回表示 SDO information 传输已经完成。

参阅:

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

5.6.5. NEC_GetEntryDesc

读取 EC-Slave 的单一对象 Entry Description 信息

C/C++语法:

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

参数:

U16_T MasterId: 指定目标主站代号；代号可由 NEC_GetRtMasterId()取得
U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号
TCoEEntryDesc *pCoeEntryDesc: 结构数据指针

TCoEEntryDesc 数据结构

```
typedef struct
{
    U16_T index;
    U8_T subIndex;
    U8_T valueInfo;
    U16_T dataType;
    U16_T bitLength;
    U16_T objectAccess;
    U16_T sizeOfData
    U8_T *pData;
    U16_T status;
    I32_T abortCode;
} TCoEEntryDesc;
```

U16_T index: 指定对象索引值
U8_T subIndex: 指定对象子索引值
U8_T valueInfo: 指定回传信息(一般设 0)
U16_T dataType: 回传对象数据型别
U16_T bitLength: 回传对象数据长度
U8_T objectAccess: 回传物存取属性
U16_T sizeOfData: 指定回传数据指针数组长度
U8_T *pData: 指定回传数据指针
U16_T status: Reserved
I32_T abortCode: SDO abort code 或 EC-Master error code

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:

本函数用于送出一个 SDO information，取得指定对象 Entry Description 信息，当函数成功返回表示 SDO information 传输已经完成。

参阅:

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

5.6.6. NEC_GetEmgDataCount

读取目标主站中 Emergency 讯息容器中讯息数量

C/C++语法:

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

参数:

U16_T MasterId: 指定目标主站代号；代号可由 NEC_GetRtMasterId()取得

U16_T *pEmgDataCount: 数据回传指针

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:

本函数用于取得网络拓朴上 EC-Slaves 产生的 Emergency 数据数量。

参阅:

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

NEC_GetEntryDesc();NEC_GetEmgData();NEC_GetRtMasterId();

5.6.7. NEC_GetEmgData

读取目标主站中 Emergency 讯息容器中讯息

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

参数:

U16_T MasterId: 指定目标主站代号；代号可由 NEC_GetRtMasterId()取得

TEmgData _T *pEmgData: 数据回传结构指针

TEmgData 数据结构

```
typedef struct
{
    U16_T lenOfData;
    U16_T res;
    U16_T *pSlaveAddrDataArr;
    TCoEEmgMsg *pEmgMsgDatArr;
} TEmgData;;
```

U16_T lenOfData: 指定数据回传数组指针长度.

U16_T res : 保留

U16_T *pSlaveAddrDataArr: 数据回传结构指针

TCoEEmgMsg *pEmgMsgDataArr: 数据回传结构指针

TCoEEmgMsg 数据结构

```
typedef struct
{
    U16_T errorCode;
    U8_T errorRegister;
    U8_T data[5];
} TCoEEmgMsg;
```

U16_T errorCode: 回传错误码

U8_T errorRegister: 回传错误缓存器

U8_T data[5]: 回传数据数组

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS”（0），反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

用法:

本函数用于读取目标主站中的 Emergency 数据。

参阅:

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

5.7. Process data 存取相关函式

5.7.1. NEC_RWProcessImage

存取目标主站的 ProcessData 数据。

C/C++语法:
RTN_ERR NEC_ NEC_RWProcessImage(U16_T MasterId, U16_T RW, U16_T Offset, U8_T *Data, U16_T Size);

参数:
U16_T MasterId: 指定目标主站代号；代号可由 NEC_GetRtMasterId()取得
U16_T RW:写入/读取指定
READ_PI (0) : Read ProcessImage (PDI)
WRITE_PI (1) : Write ProcessImage (PDO)
READ_PDO (2) : Read ProcessImage (PDO)
U16_T Offset: 目标主站 ProcessData 内存偏移量，从 0 开始，单位 Byte
U8_T *Data: 数据指针
U16_T Size: 数据长度，单位 Byte

回传值:
回传错误代码。
调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:
本函式可用来读/写目标主站内部的 Process Image (或称 ProcessData) 。ProcessData 原理及操作方式请参考 3.7 小节 Process Data 存取。

参阅:
NEC_GetRtMasterId();NEC_GetProcessImageSize();NEC_RWSlaveProcessImage ()

5.7.2. NEC_GetProcessImageSize

取得目标主站的 ProcessData 内存长度。

C/C++语法:
RTN_ERR NEC_GetProcessImageSize(U16_T MasterId, U16_T *SizeOfInputByte, U16_T *SizeOfOutputByte);

参数:
U16_T MasterId: 指定目标主站代号；代号可由 NEC_GetRtMasterId()取得
U16_T *SizeOfInputByte: 回传 ProcessData Input 内存大小指针
U16_T *SizeOfOutputByte: 回传 ProceData Output 内存大小指针

回传值:
回传错误代码。
调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:
本函数可用来读取目标主站中 ProcessImage 内存的大小，ProcessData 原理及操作方式请参考 3.7 小节 Process Data 存取。

参阅:
NEC_GetRtMasterId();NEC_RWProcessImage ();NEC_RWSlaveProcessImage ()

5.7.3. NEC_RWSlaveProcessImage

存取 EC-Slave 的 ProcessData 内存。

C/C++语法:
RTN_ERR NEC_RWSlaveProcessImage(U16_T MasterId, U16_T SlaveAddr, U16_T RW, U16_T Offset, U8_T *Data, U16_T Size);

参数:
U16_T MasterId: 指定目标主站代号；代号可由 NEC_GetRtMasterId()取得
U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号
U16_T RW:写入/读取指定
READ_PI (0) : Read ProcessImage (PDI)
WRITE_PI (1) : Write ProcessImage (PDO)
READ_PDO (2) : Read ProcessImage (PDO)
U16_T Offset: EC-Slave ProcessData 内存偏移量，从 0 开始，单位 Byte
U8_T *Data: 数据指针
U16_T Size: 数据长度，单位 Byte

回传值:
回传错误代码。
调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:
本函数用来读取或写入数据到 EC-Slave 所占的 ProcessData 内存，ProcessData 原理及操作方式请参考 3.7 小节 Process Data 存取。

参阅:
NEC_GetRtMasterId();NEC_RWProcessImage ();NEC_GetProcessImageSize ()

5.7.4. NEC_RWProcessImageEx

存取目标主站的 ProcessData 数据，支持位型式

C/C++语法:

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

参数:

U16_T MasterId: 指定目标主站代号；代号可由 NEC_GetRtMasterId()取得

U16_T RW:写入/读取指定

READ_PI (0) : Read ProcessImage (PDI)

WRITE_PI (1) : Write ProcessImage (PDO)

READ_PDO (2) : Read ProcessImage (PDO)

RwPdoData_T *PRwPdoData:

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

U32_T offsetBit: 指定存取的位偏移量

U32_T bitSize: 指定存取的位数量

U8_T data[MAX_RW_PDO_DATA_LEN]: 数据储存空间

回传值:

回传错误代码。

调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:

本函式可用来读/写目标主站内部的 Process Image (或称 ProcessData) 。ProcessData 原理及操作方式请参考 3.7 小节 Process Data 存取。

参阅:

NEC_GetRtMasterId();NEC_GetProcessImageSize();NEC_RWSlaveProcessImageEx()

5.7.5. NEC_RWSlaveProcessImageEx

存取 EC-Slave 的 ProcessData 内存。

C/C++语法:
RTN_ERR NEC_RWSlaveProcessImageEx(U16_T MasterId, U16_T SlaveAddr, U16_T RW, RwPdoData_T *PRwPdoData);

参数:
U16_T MasterId: 指定目标主站代号；代号可由 NEC_GetRtMasterId()取得
U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号
U16_T RW:写入/读取指定

READ_PI (0) : Read ProcessImage (PDI)
WRITE_PI (1) : Write ProcessImage (PDO)
READ_PDO (2) : Read ProcessImage (PDO)

RwPdoData_T *PRwPdoData:

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

U32_T offsetBit: 指定存取的位偏移量
U32_T bitSize: 指定存取的位数量
U8_T data[MAX_RW_PDO_DATA_LEN]: 数据储存空间

回传值:
回传错误代码。
调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:
本函数用来读取或写入数据到 EC-Slave 所占的 ProcessData 内存，ProcessData 原理及操作方式请参考 3.7 小节 Process Data 存取。

参阅:
NEC_GetRtMasterId();NEC_RWProcessImageEx();NEC_GetProcessImageSize ()

5.8. 存取从站模块硬件信息相关函式

5.8.1. NEC_GetConfiguredAddress

取得从站模块”Configured Station Address”

C/C++语法:

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

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0
U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号
U16_T *pConfigAddr: 写入数据指针

回传值:

回传错误代码。
调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:

此函数用于取得指定从站模块的 Configured Station Address 信息。例如，使用者可经由底下程序代码，得到从站模块代号 0 其 Configured Station Address

```
U16_T SlaveAddr = 0;     // The first slave
U16_T ConfigAddr = 0;    // A variable for storing the configured address
NEC_GetConfiguredAddress ( MasterId,    SlaveAddr,    &ConfigAddr );
```

参阅:

5.8.2. NEC_GetAliasAddress

取得从站模块”Configured Station Alias”

C/C++语法:

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

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0
U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号
U16_T *pAliasAddr: 写入数据指针

回传值:

回传错误代码。
调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:

此函数用于取得指定从站模块的 Configured Station Alias 信息。例如，使用者可经由底下程序代码，得到 Configured Station Alias 为 0x0021 其模块代号

```
U16_T i;
U16_T SlaveAddr, i;
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;
    }
}
```

参阅:

5.8.3. NEC_GetSlaveCoeProfileNum

取得从站模块”CoeProfileNum”信息

C/C++语法:

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

参数:

U16_T MasterId: 目标主站的代号，单一 EC-Master 请设为 0
U16_T SlaveAddr: 指定目标 EC-Slave 代号，从 0 开始依序增号
U32_T *pCoeProfileNum: 写入数据指针

回传值:

回传错误代码。
调用函数成功回传“ECERR_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于 EcErrors.h 头文件中。

用法:

此函数用于取得指定从站模块的 CoeProfileNum 信息。例如，用户可经由底下程序代码，得到网络上那些模块为驱动器(CoeProfileNum 等于 402)

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

NEC_GetSlaveCount(MasterId, &SlaveCnt)
for( i = 0; i < SlaveCnt; ++i )
{
    NEC_GetSlaveCoeProfileNum (MasterId, i, &CoeProfileNum);
    if( CoeProfileNum == 402)
    {
        Printf(“SlaveAddr:%d is a drive.\n”, i );
    }
}
```

参阅: