

## **NexECMRtx**

### **EtherCAT Master for RTX**

### **用户手册**

Manual Rev.: 1.8

Revision Date: May, 2<sup>th</sup>, 2016

Part No:

## Revise note:

Date	Ver	Description
2013/1/10	1.0	First version release
2013/12/10	1.1	Add NEC_RtSetProcessDataPtrSource function
2014/1/23	1.2	Modify chapter 2.3.1 How to config RTX TCP/IP
2014/6/16	1.3	Add Chapter 7. NexECM Library (Win32API)
2014/11/19	1.4	Ch2.1 Add support platform Ch2.3 Modify NIC driver setup Ch3 Enhance NEXCAT Utility description
2015/5/15	1.5	Typo fix
2015/8/10	1.6	Add Ch6.6.2 NEC_RtGetSlaveInfomation() API Add Ch6.6.3 NEC_RtGetSlaveState() API
2015/12/11	1.7	Add Ch6.8.1 NEC_RtGetConfiguredAddress() API Add Ch6.8.2 NEC_RtGetAliasAddress() API Add Ch6.8.3 NEC_RtNEC_GetSlaveCoeProfileNum() API Add Ch7.9.1 NEC_GetConfiguredAddress() API Add Ch7.9.2 NEC_GetAliasAddress() API Add Ch7.9.3 NEC_GetSlaveCoeProfileNum () API
2016/5/6	1.8	Add Ch5.8.2 CoE – Emergency 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_RtEmgDataCount Add Ch6.7.12 NEC_RtEmgData 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_EmgDataCount Add Ch7.7.10 NEC_EmgData

# 目录

NexECMRtx .....	1
Revise note: .....	2
目录 .....	i
1. NexECMRtx 介绍 .....	1
1.1. NexECMRtx 功能特点 .....	2
2. NexECMRtx 安装说明 .....	4
2.1. 硬件需求 .....	4
2.2. 软件需求 .....	4
2.3. NexECMRtx 安装、开发流程 .....	4
2.3.1. 安装 RTX NIC 驱动程序 .....	5
2.3.2. 设定网络卡“C:\RtxEcNic.ini” .....	11
2.3.3. 开发 EtherCAT 主站 RTX 应用程序 .....	14
3. EtherCAT 公用程序说明 .....	19
3.1. NexCAT 简介 .....	19
3.1.1. 软件需求 .....	19
3.1.2. NexCAT 操作流程说明 .....	20
3.1.3. ESI 和 ENI 档案 .....	22
3.1.4. NexCAT 主页面操作 .....	23
3.1.5. 从站模块参数设定页面 (Set Slave Parameters) .....	25
3.1.6. EC-Master 参数设定页面 (Set Master Parameters) .....	30
3.1.7. ESI 档案管理窗体 (ESI List) .....	32
3.1.8. DIO 操作页面 .....	33
3.1.9. CoE-SDO 存取操作页面 .....	35
3.1.10. Process Image 参数存取操作页面 .....	36
3.1.11. Network Quality Monitor 网络通讯质量测试页面 .....	37

3.2. NexECMRtxStarup 启动程序说明.....	39
4. EtherCAT 技术简介 .....	41
4.1. EtherCAT 通讯协议概述 .....	42
4.2. 网络拓扑 .....	43
4.3. 高速效能 .....	44
4.4. 网络同步 .....	45
5. 编程原理 .....	47
5.1. 基本编程框架 .....	47
5.2. ENI 载入.....	48
5.3. 初始化 NexECMRtx .....	49
5.4. 启动主站通讯 .....	50
5.5. Callback 函式.....	51
5.5.1. 注册回调函数(Callback functions).....	51
5.5.2. 周期回调函数(Cyclic-Callback function).....	51
5.5.3. 事件回调函数(Event-Callback function).....	53
5.5.4. 错误事件回调函数(Error-Callback function).....	53
5.6. EtherCAT 状态机 .....	54
5.6.1. ESM 状态变更 .....	54
5.7. Process Data 存取 .....	56
5.7.1. ProcessData 运作机制.....	56
5.7.2. ProcessData 数据内容.....	57
5.8. Mailbox 通讯 .....	59
5.8.1. CoE -SDO 通讯 .....	59
5.8.2. CoE -Emergency .....	60
6. NexECMRtx 链接库 .....	61
6.1. RTX API 总览.....	61
6.2. 初始化相关函式 .....	63

6.2.1. NEC_RtGetVersion .....	63
6.2.2. NEC_RtRetVer .....	65
6.2.3. NEC_RtInitMaster .....	66
6.2.4. NEC_RtCloseMaster .....	67
6.2.5. NEC_RtSetParameter .....	68
6.2.6. NEC_RtGetParameter .....	68
6.2.7. NEC_RtRegisterClient.....	70
6.2.8. NEC_RtUnregisterClient.....	72
6.2.9. NEC_RtGetProcessDataPtr.....	73
6.2.10. NEC_RtSetProcessDataPtrSource .....	74
6.3. EC-Master 控制相关函式 .....	76
6.3.1. NEC_RtStartMaster .....	76
6.3.2. NEC_RtStopMaster .....	77
6.3.3. NEC_RtStopMasterCb .....	78
6.3.4. NEC_RtWaitMasterStop.....	79
6.3.5. NEC_RtGetMasterState .....	80
6.3.6. NEC_RtSetMasterState .....	80
6.3.7. NEC_RtChangeStateToOP .....	82
6.3.8. NEC_RtSetMasterStateWait .....	83
6.3.9. NEC_RtGetStateError.....	84
6.4. Process data 存取相关函式.....	85
6.4.1. NEC_RtSetProcessDataOutput.....	85
6.4.2. NEC_RtGetProcessDataOutput.....	85
6.4.3. NEC_RtGetProcessDataInput.....	85
6.4.4. NEC_RtSetSlaveProcessDataOutput .....	86
6.4.5. NEC_RtGetSlaveProcessDataOutput .....	86
6.4.6. NEC_RtGetSlaveProcessDataInput .....	86

6.5. Callback 函式.....	88
6.5.1. *NEC_RtCyclicCallback.....	88
6.5.2. *NEC_RtEventCallback.....	89
6.5.3. *NEC_RtErrorCallback.....	90
6.6. ENI 相关函式.....	91
6.6.1. NEC_RtGetSlaveCount .....	91
6.6.2. NEC_RtGetSlaveInfomation .....	92
6.6.3. NEC_RtGetSlaveState.....	94
6.7. CoE 通讯相关函式.....	95
6.7.1. NEC_RtStartSDODownload .....	95
6.7.2. NEC_RtStartSDOUpload.....	95
6.7.3. NEC_RtSDODownload.....	97
6.7.4. NEC_RtSDOUpload.....	97
6.7.5. NEC_RtSDODownloadEx .....	97
6.7.6. NEC_RtSDOUploadEx.....	97
6.7.7. NEC_RtStartGetODListCount .....	98
6.7.8. NEC_RtStartGetODList .....	100
6.7.9. NEC_RtStartGetObjDesc .....	102
6.7.10. NEC_RtStartGetEntryDesc .....	104
6.7.11. NEC_RtGetEmgDataCount .....	106
6.7.12. NEC_RtGetEmgData.....	107
6.8. 存取从站模块硬件信息相关函式 .....	109
6.8.1. NEC_RtGetConfiguredAddress.....	109
6.8.2. NEC_RtGetAliasAddress.....	110
6.8.3. NEC_RtGetSlaveCoeProfileNum .....	112
7. NexECM 链接库 (Win32 APIs) .....	113
7.1. Win32 API 总览 .....	114

7.2. RTX 系统相关控制 API.....	117
7.2.1. NEC_LoadRtxApp .....	117
7.2.2. NEC_KillRtxApp .....	118
7.3. NexECM 初始化相关函式 .....	119
7.3.1. NEC_GetVersion.....	119
7.3.2. NEC_StartDriver .....	120
7.3.3. NEC_StopDriver .....	121
7.4. NexECMRtx Runtime 控制相关函式.....	122
7.4.1. NEC_GetRtMasterId.....	122
7.4.2. NEC_GetRtMasterVersion.....	123
7.4.3. NEC_GetRtMasterPorcessId .....	124
7.4.4. NEC_ResetEcMaster.....	125
7.4.5. NEC_LoadNetworkConfig.....	126
7.4.6. NEC_StartNetwork.....	127
7.4.7. NEC_StartNetworkEx .....	128
7.4.8. NEC_StopNetwork .....	129
7.4.9. NEC_SetParameter .....	130
7.4.10. NEC_GetParameter.....	130
7.5. 网络状态存取相关函式 .....	132
7.5.1. NEC_GetSlaveCount.....	132
7.5.2. NEC_GetNetworkState .....	133
7.5.3. NEC_GetSlaveState .....	134
7.5.4. NEC_GetStateError .....	135
7.5.5. NEC_GetErrorMsg.....	136
7.6. DIO 控制相关函式 .....	137
7.6.1. NEC_SetDo .....	137
7.6.2. NEC_GetDo .....	138

7.6.3. NEC_GetDi.....	139
7.7. CoE 通讯相关函式 .....	140
7.7.1. NEC_SDODownloadEx.....	140
7.7.2. NEC_SDOUploadEx .....	140
7.7.3. NEC_SDODownload .....	140
7.7.4. NEC_SDOUpload .....	140
7.7.5. NEC_GetODListCount.....	142
7.7.6. NEC_GetODList .....	144
7.7.7. NEC_GetObjDesc.....	146
7.7.8. NEC_GetEntryDesc.....	148
7.7.9. NEC_GetEmgDataCount.....	150
7.7.10. NEC_GetEmgData .....	151
7.8. Process data 存取相关函式.....	153
7.8.1. NEC_RWProcessImage.....	153
7.8.2. NEC_GetProcessImageSize .....	154
7.8.3. NEC_RWSlaveProcessImage .....	155
7.9. 存取从站模块硬件信息相关函式 .....	156
7.9.1. NEC_GetConfiguredAddress .....	156
7.9.2. NEC_GetAliasAddress .....	157
7.9.3. NEC_GetSlaveCoeProfileNum.....	159



## 1. NexECMRtx 介绍

NexECMRtx 是一套建构于 RTX (Microsoft Windows 延伸实时子操作系统) 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。NexECMRtx 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 – NexCAT 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 NexECMRtx 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。

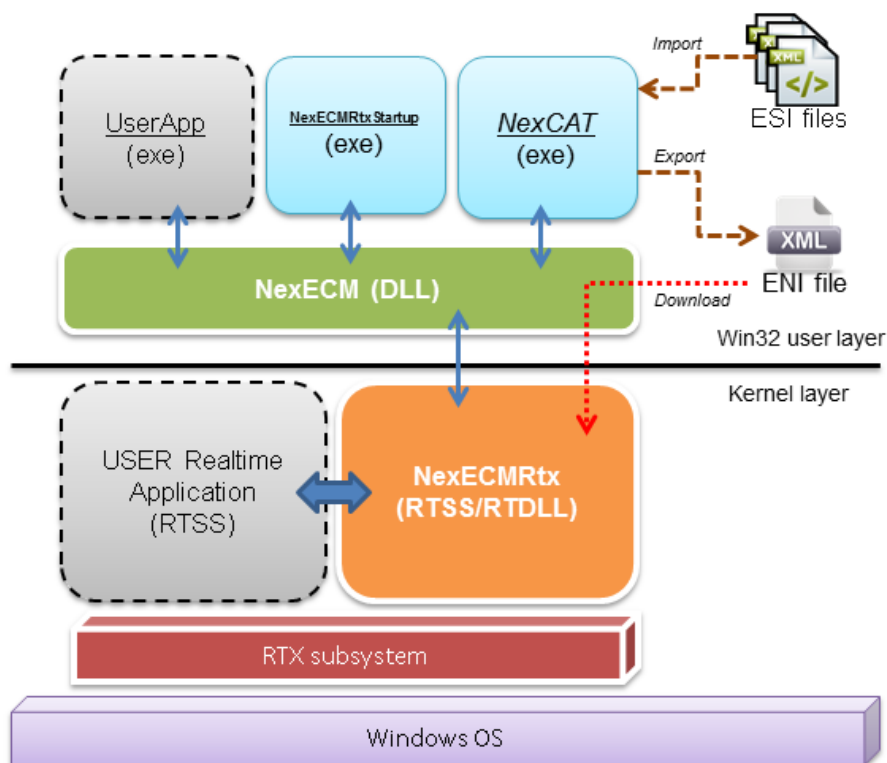


图 1.1: NEXCOM EtherCAT master solution 系统方块图

系统方块图个模块功能简述如下表:

软件模块	说明	参考章节
运作位置:RTX 子系统		
NexECMRtx.rtss	主站通讯 (EC-Master)核心 RTX 链接库	CH 5, CH 6
运作位置:Win32 用户层(UserMode)		
NexCAT.exe	EC-Master 公用程序	CH 3.1
NexECMRtxStartup.exe	EC-Master 启动辅助功能程序	CH 3.2
NexECM.dll	EC-Master Win32 沟通接口链接库	CH7

### 1.1. NexECMRtx 功能特点

根据 EtherCAT 标准文件编号 ETG.1500, NexECMRtx 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
<b>Basic Features</b>		
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
<b>Process Data Exchange</b>		
Cyclic PDO	Cyclic process data exchange	V
<b>Network Configuration</b>		
Reading ENI	Network Configuration taken from ENI file	V
Compare Network configuration	Compare configured and existing network configuration during boot-up	V
Explicit Device Identification	Identification used for Hot Connect and prevention against cable swapping	V
Station Alias Addressing	Support configured station alias in slave, i.e. enable 2nd Address and use it	V
Access to	Support routines to access EEPROM via ESC	V

EEPROM	register	
<b>Mailbox Support</b>		
Support Mailbox	Main functionality for mailbox transfer	V
Mailbox polling	Polling Mailbox state in slaves	V
<b>CAN application layer over EtherCAT (CoE)</b>		
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	△
<b>Ethernet over EtherCAT (EoE)</b>		
EoE	Ethernet over EtherCAT	△
<b>File over EtherCAT (FoE)</b>		
FoE	File over EtherCAT	△
<b>Servo over EtherCAT (SoE)</b>		
SoE	Servo over EtherCAT	△
<b>Distributed Clocks</b>		
DC	Support of Distributed Clock	V

## 2. NexECMRtx 安装说明

### 2.1. 硬件需求

标准 X86 PC / IPC，并具备一个以上以太网(Ethernet)通讯端口，NexECMRtx 已经在 Nexcom 平台进行验证及优化，请参考“NexECMRtx hardware support list.pdf”文件，此文件存放在 NexECMRtx 安装目录中

### 2.2. 软件需求

依照上述硬件规格，在系统上须预安装：

1. 操作系统要求 Microsoft Windows XP SP3 and Window 7 32 位操作系统
2. IntervalZero RTX 2012
3. Microsoft Visual studio 2005 ~2012
4. NEXCOM NexECMRtx 安装档案
5. .Net framework 4.0 (供 NexCAT.exe 公用程序)
6. Visual Basic Power Packs 3.0 (供 NexCAT.exe 公用程序)

- 安装 RTX 2012，你可以在官方网站中下载 RTX 2012，90 天试用版<sup>(\*)</sup>：

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

(\*) 购买正式版或申请试用版序号请洽 RTX 大中华区总代理-新汉计算机

Tel : +886-2-8226-7786 # ICS 业务处

- Net framework 4.0 官方免费下载：

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

- Visual Basic Power Packs 3.0 官方免费下载：


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

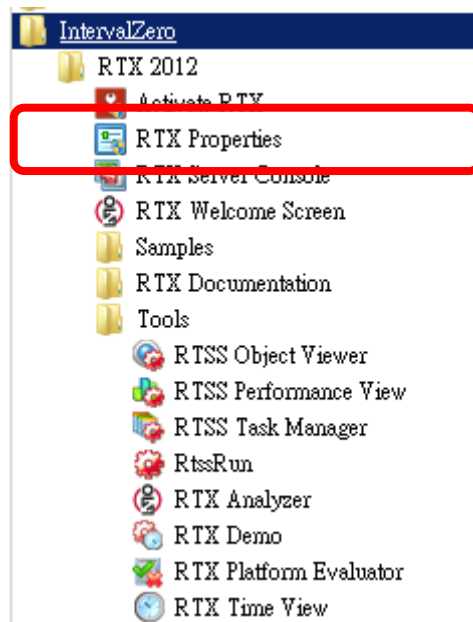
### 2.3. NexECMRtx 安装、开发流程

当您取得 IPC 硬件平台，并安装好 Windows 操作系统和 RTX 子系统，请依照下列步骤完成软件安装：

1. 安装 RTX NIC 驱动程序
2. 设定网络卡“C:\RtxEcNic.ini”
3. 使用 NexCAT 设定 EtherCAT 网络(请参考第三章)
4. 开发 EtherCAT 主站 RTX 应用程序

## 2.3.1. 安装 RTX NIC 驱动程序

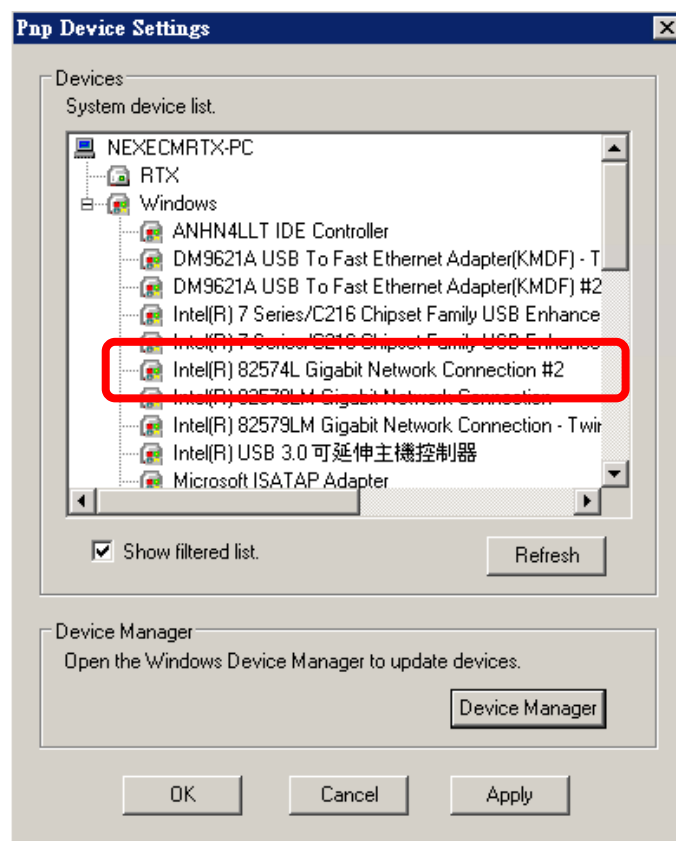
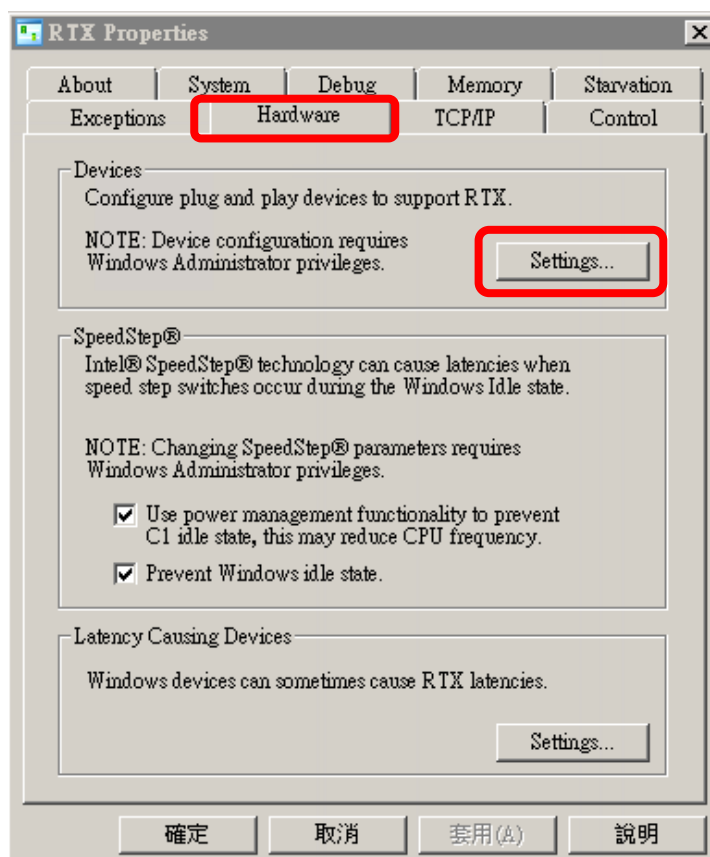
在开始菜单中(如下图)或桌面上  打开 RTX 工具程序:“RTX properties” 如下图:

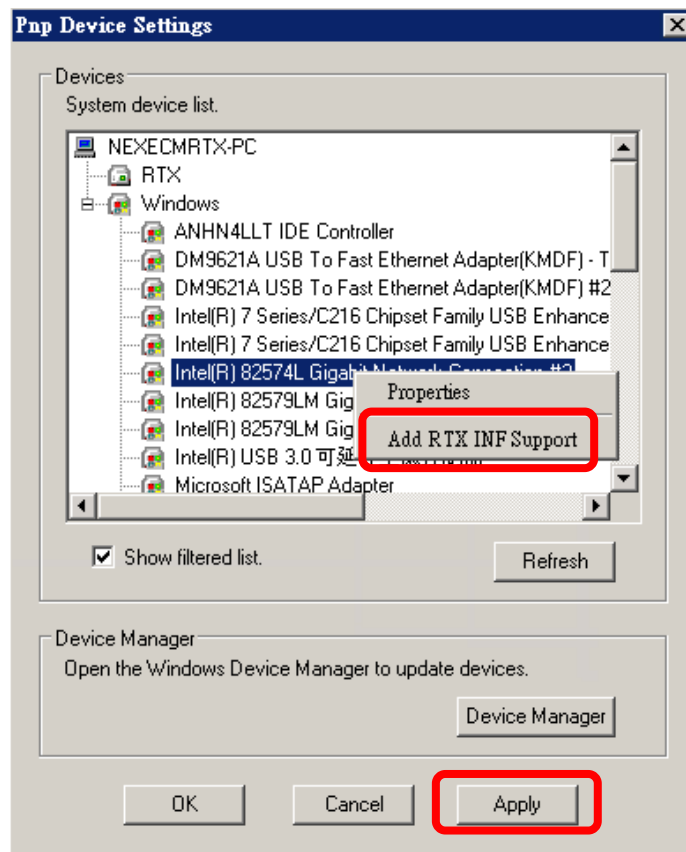


RTX properties tool

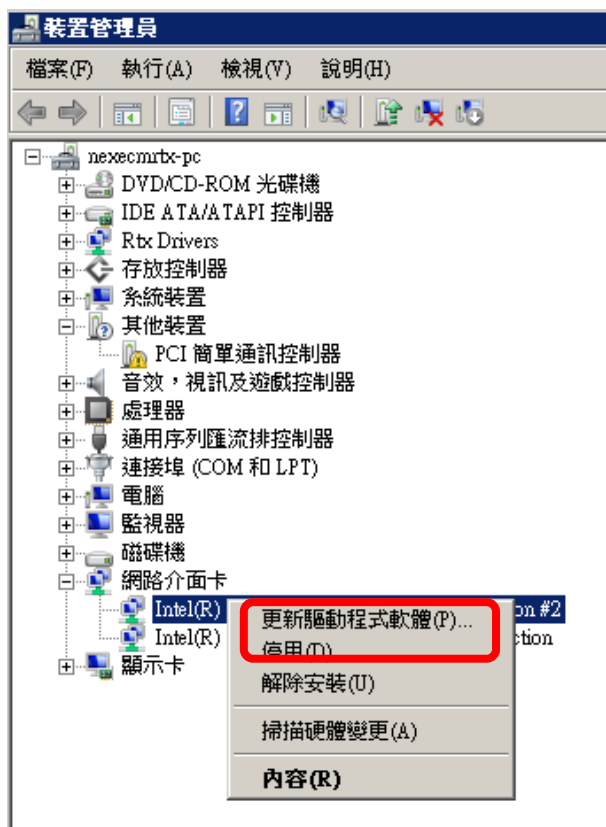
在 “Hardware” 页面(如下图)

1. 点选 Device/Setting 按钮弹出“Pnp Device Settings”页面
2. 在网络卡上右键选择 “Add RTX INF Support” 并点选 “Apply”

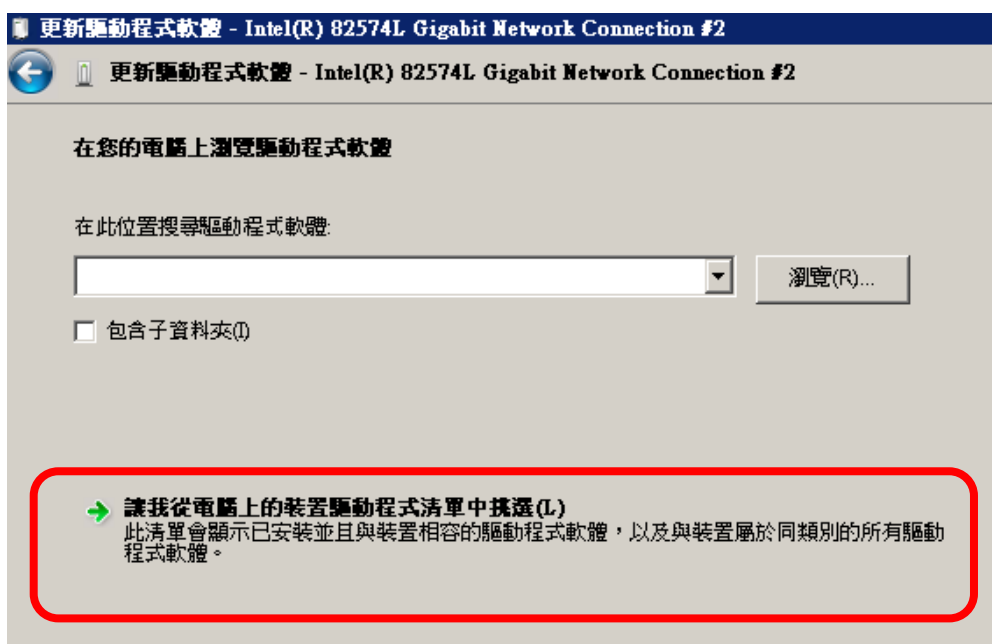
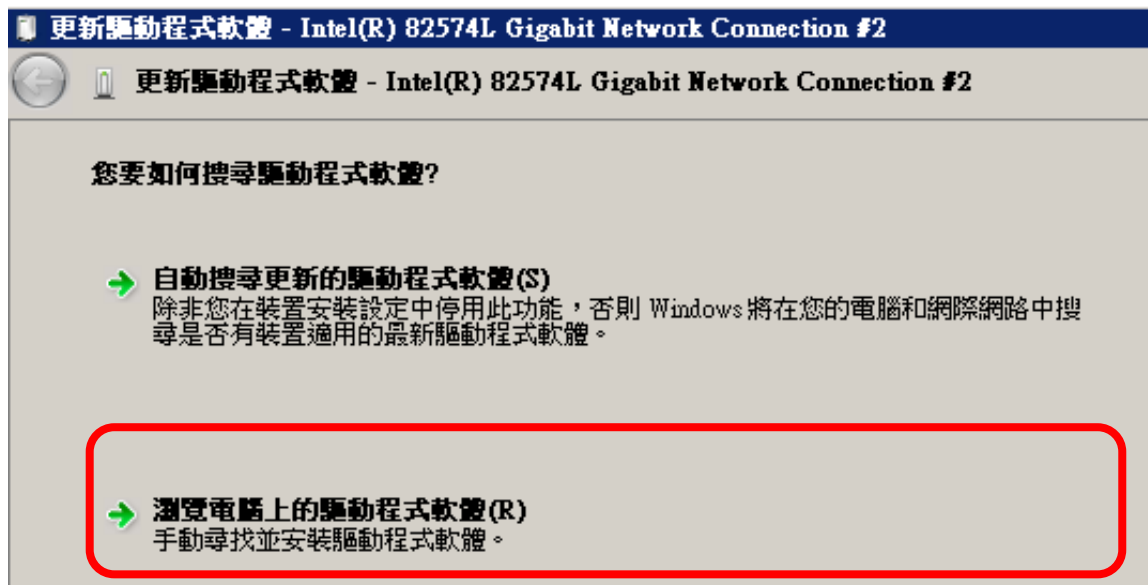




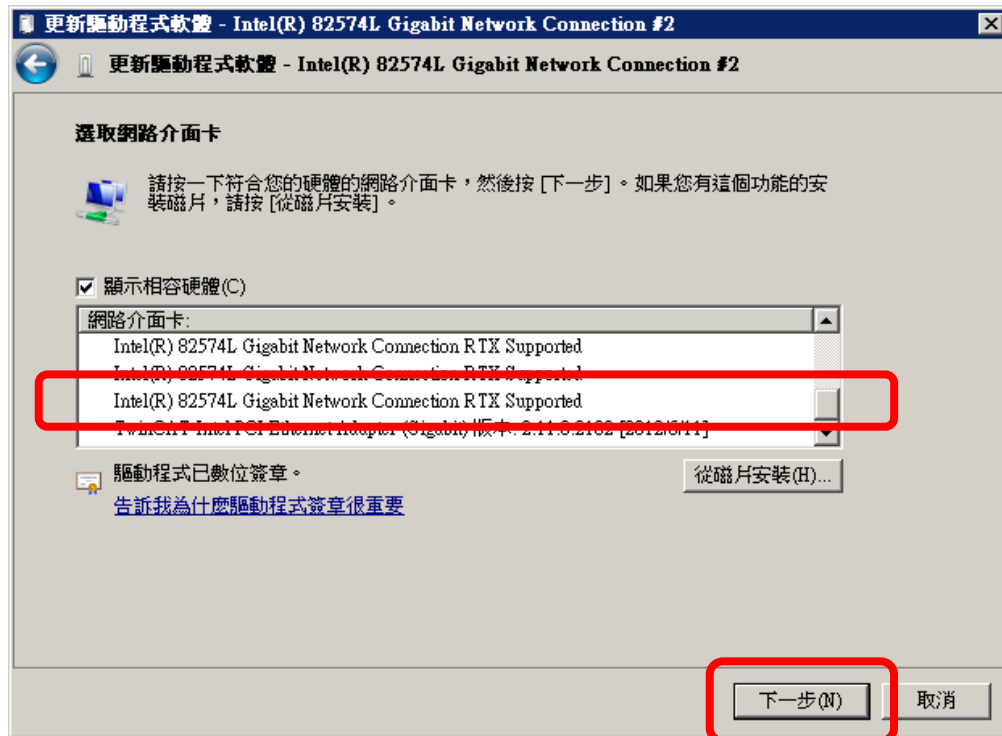
开启文件管理器，右键点选网络卡“更新驱动程序软件”



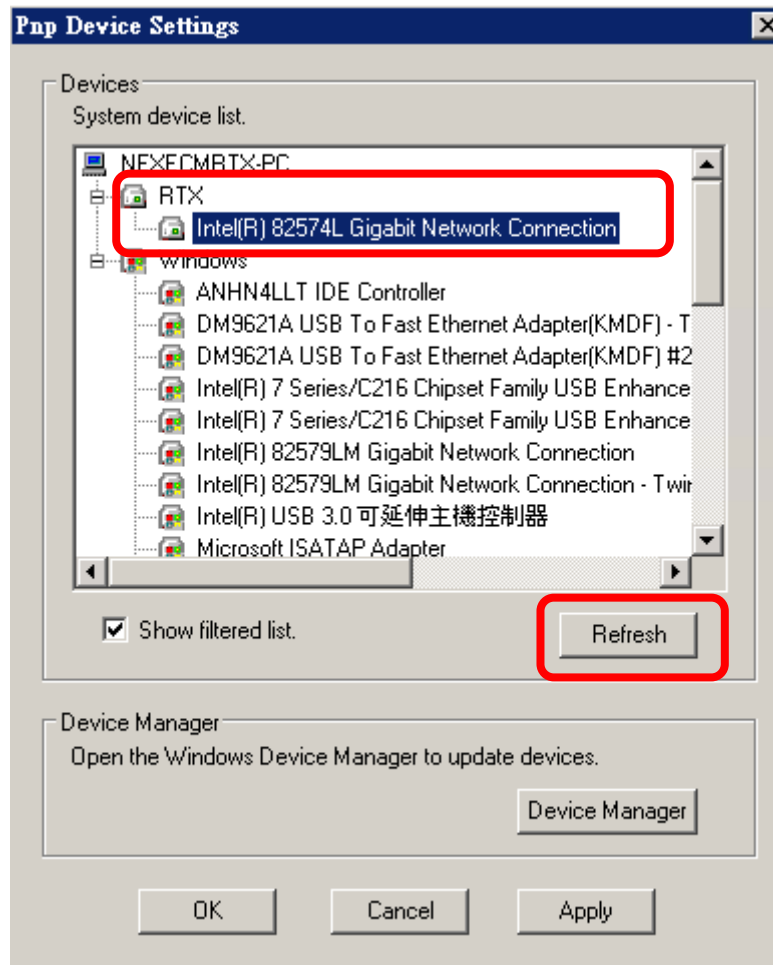
以下列步骤选取 RTX supported 网卡驱动程序







安裝完成后可在“Pnp Device Settings”页面确认 RTX 驱动程序是否正常安装



### 2.3.2. 设定网络卡“C:\RtxEcNic.ini”

使用记事本开启“C:\RtxEcNic.ini”档案如下



```

[Info]
NumOfInterfaces = 2

[Nic0]
BusNum=2
DevNum=0
FunNum=0

[Nic1]
BusNum=5
DevNum=0
FunNum=0
    
```

NumOfInterfaces: EtherCAT 网络卡数量

[Nic0]: 第一张网卡的位置

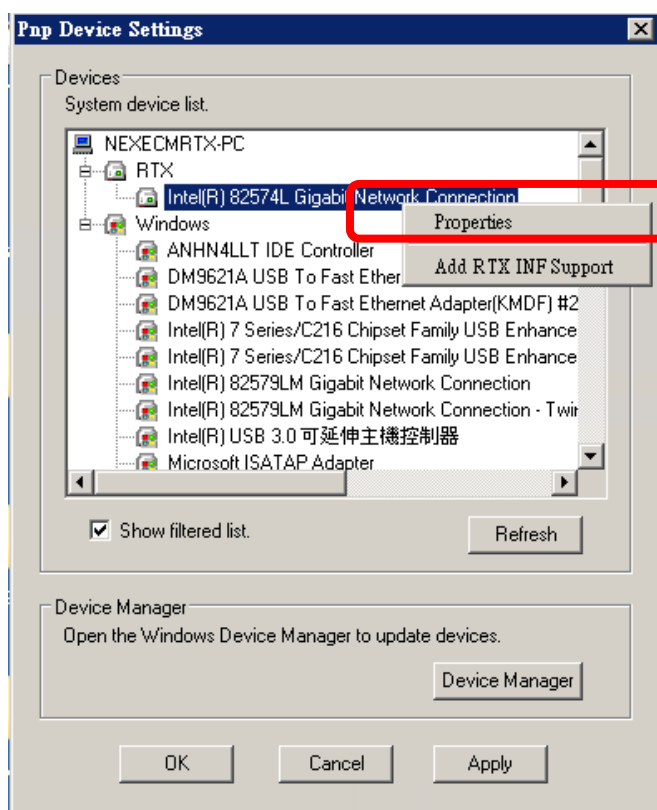
[Nic1]: 第二张网卡的位置

...以此类推

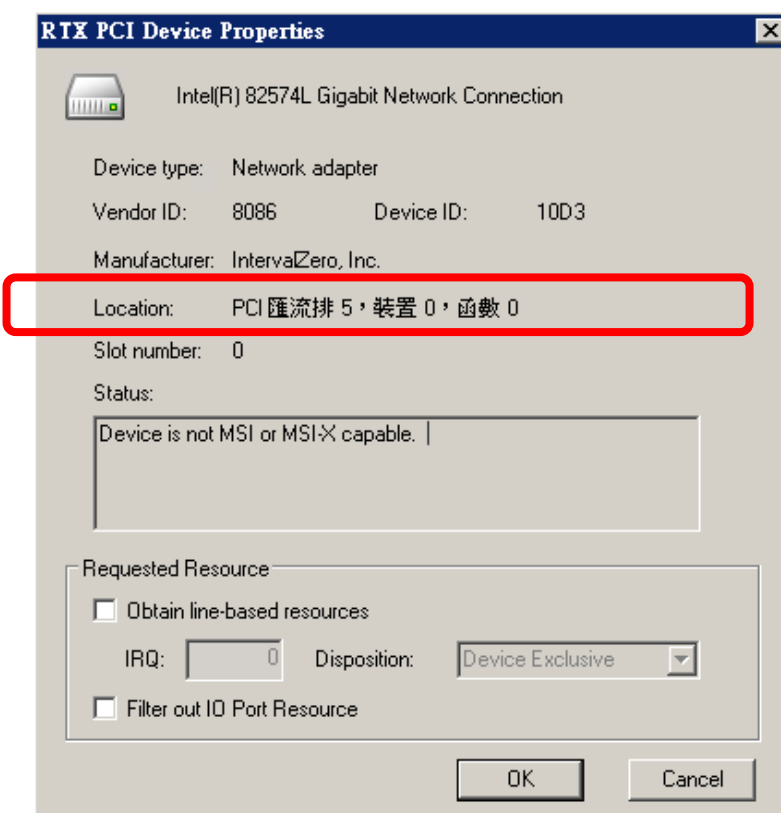
BusNum / DevNum/ FunNum PCI 位置信息可由下述步骤查询:

在 “Pnp Device Settings”页面, 右键网卡选择 “Properties”即弹出 “RTX PCI Device Properties” 页面, 将“Location” 位置纪录下来填入 INI 档案中。

**特别注意:** 此 INI 档案必须置于 “C:\RtxEcNic.ini” 路径



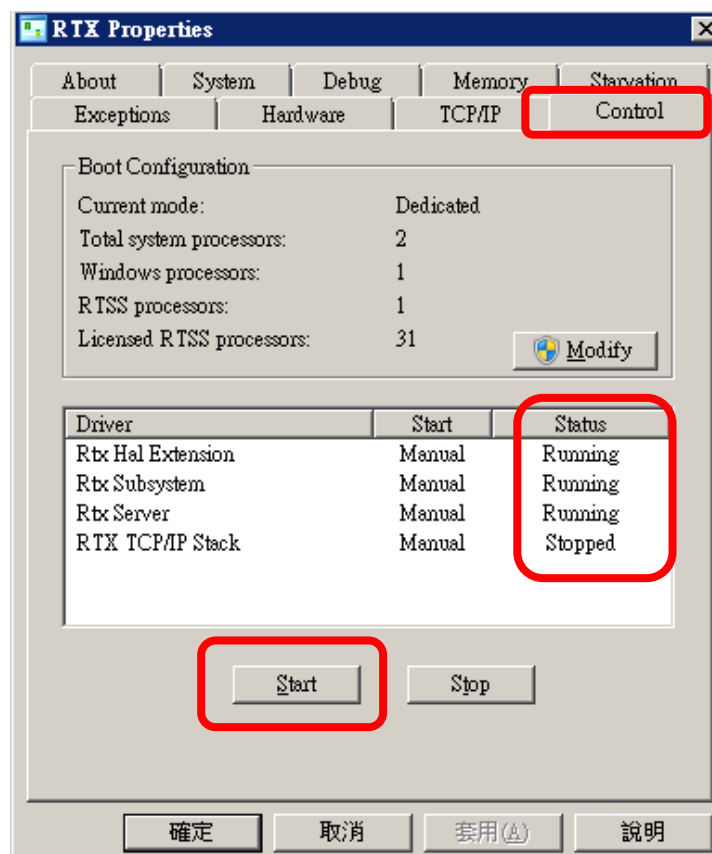
在“Pnp Device Settings”页面，右键网卡选择“Properties”



将“Location”位置纪录下来填入 INI 档案中

## 启动 RTX Subsystem

打开 RTX 工具程序: “RTX properties”，于“Control” 页面(如下图)，点选“Start” button 启动 RTX system，RTX 的 Driver 状态会切换到 “Running”状态。

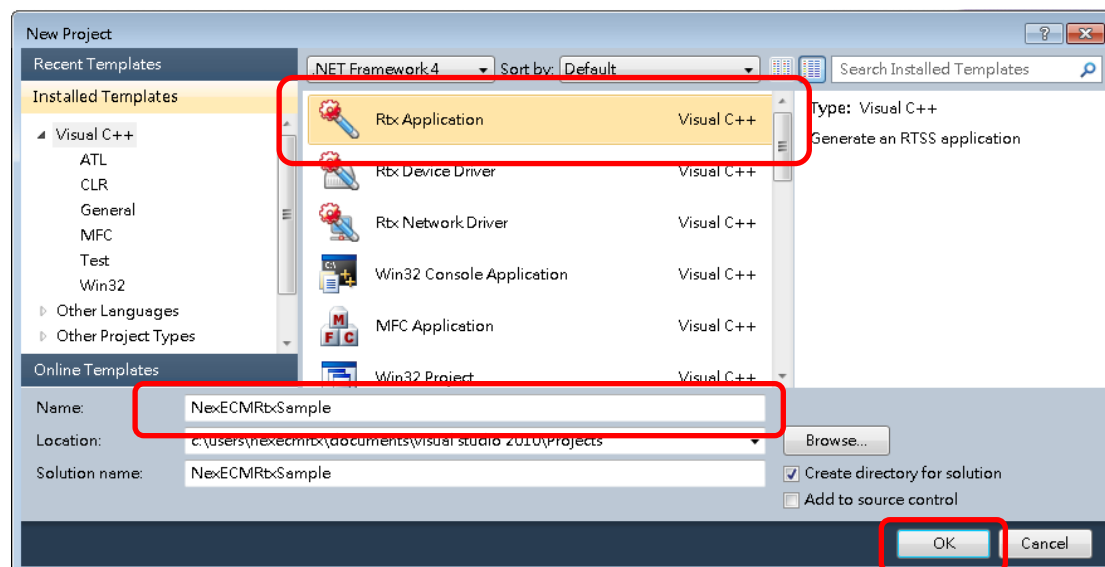


当上述步骤完成，NexECMRtx 的前置安装步骤已经全部完成，而上述步骤只需第一次安装 NexECMRtx 完成。后续无须再进行设定。

## 2.3.3. 开发 EtherCAT 主站 RTX 应用程序

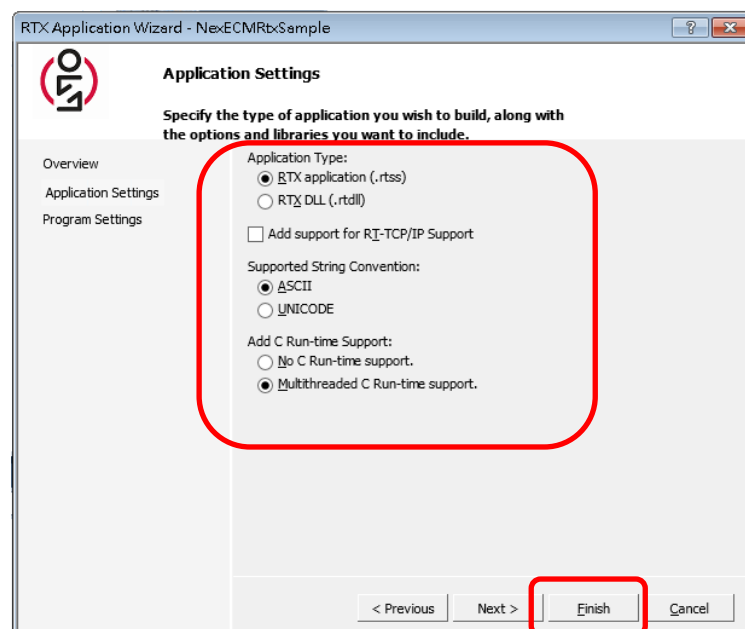
下面步骤说明如何使用 Visual studio 开发 EtherCAT 主站 RTX 应用程序，以 Visual studio 2010 为例：

### 1. 新增一专案(File/New)

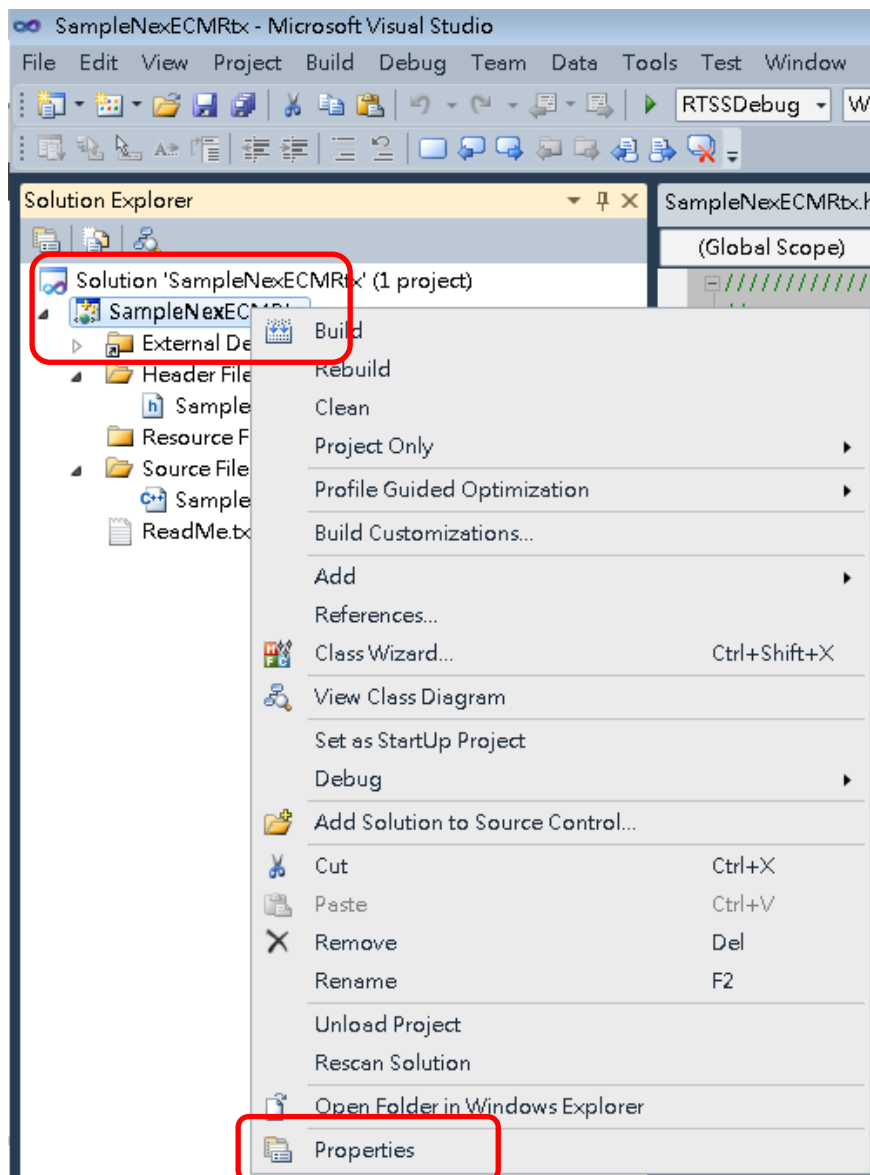


选择 RTX 项目，输入项目名称。

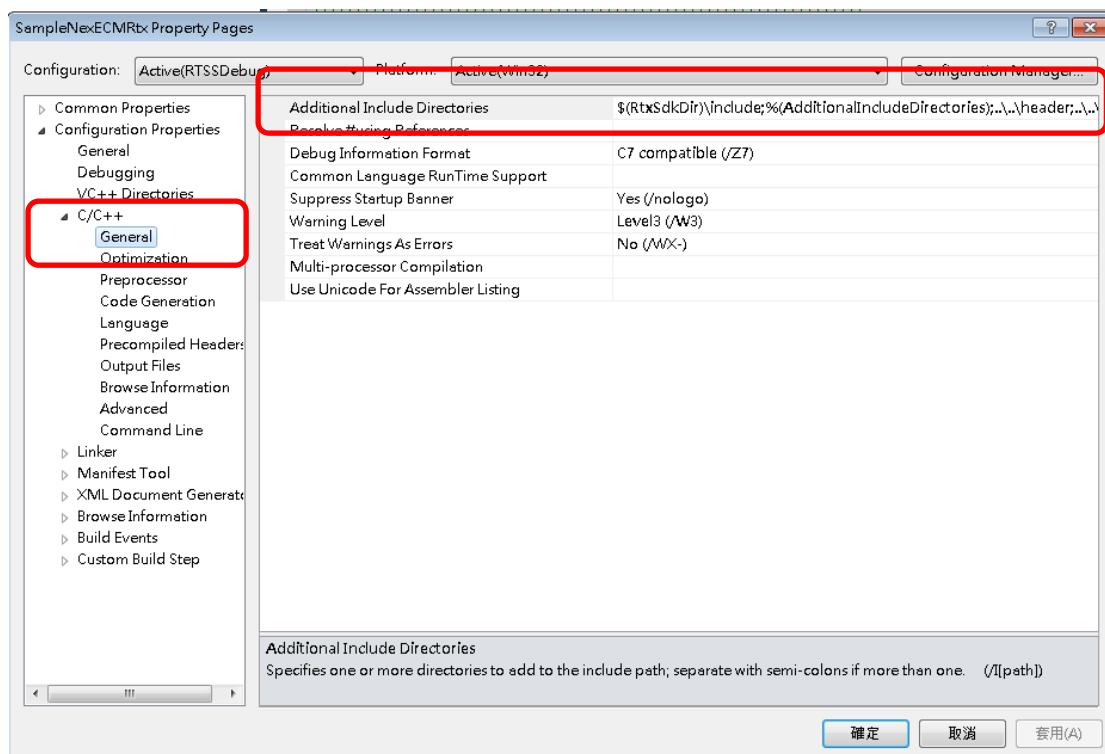
### 2. 设定 RTX 应用程序项目选项，如下图



3. 设定使用 NexECMRtx 函式库，在 Solution Explorer 中项目上右键点选 Properties

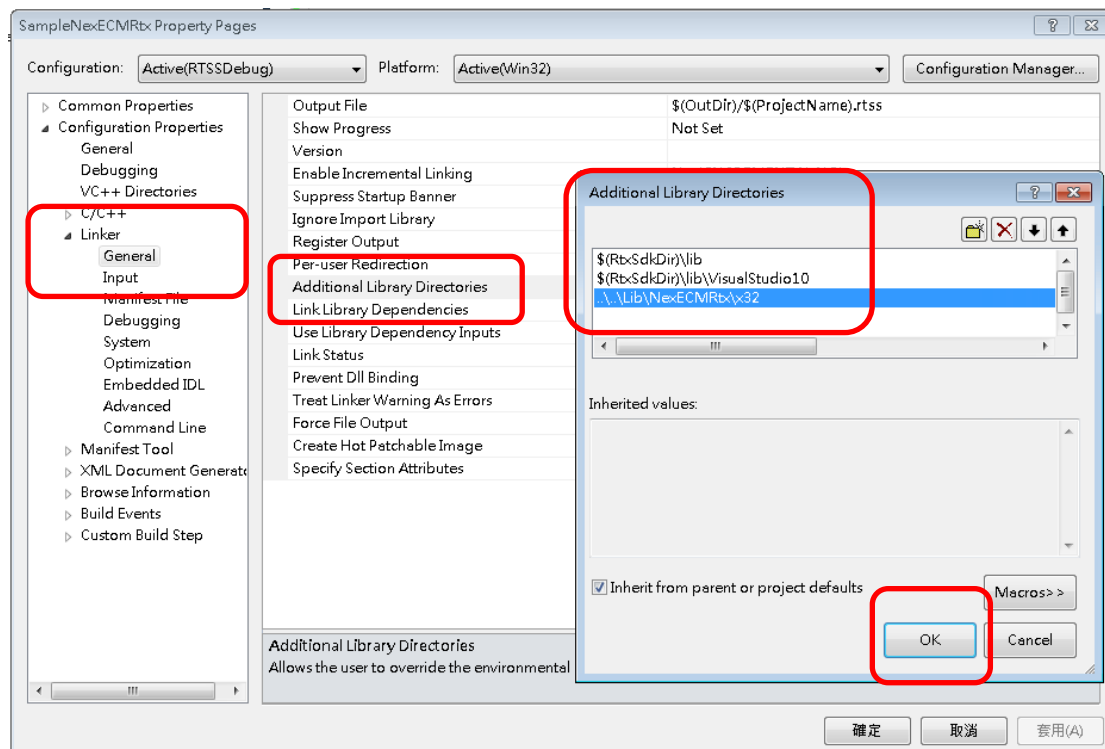


C/C++ \ General 中设定 NexECMRtx 的头文件路径，如下图：



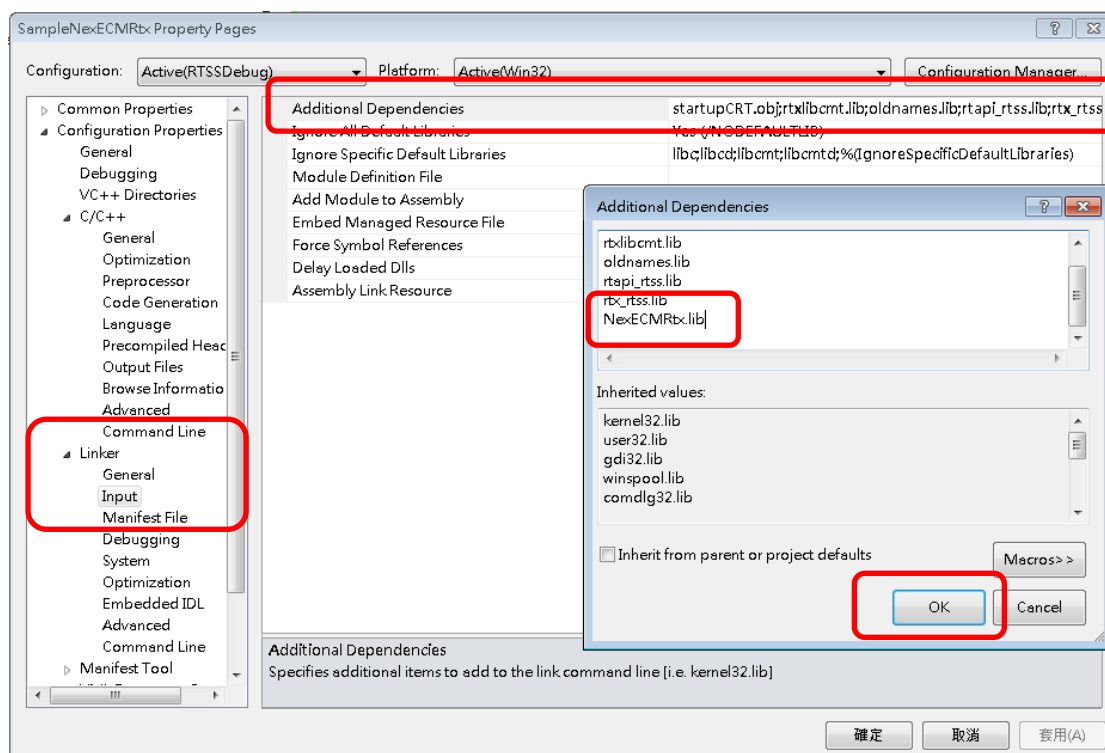
NexECMRtx 的头文件置于安装目录中。

Linker\General 设定 NexECMRtx.lib 的路径

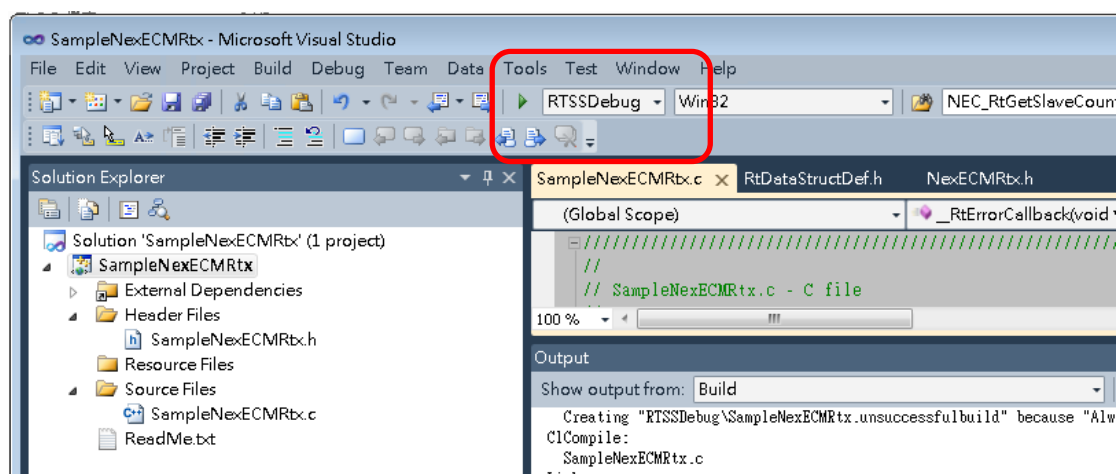




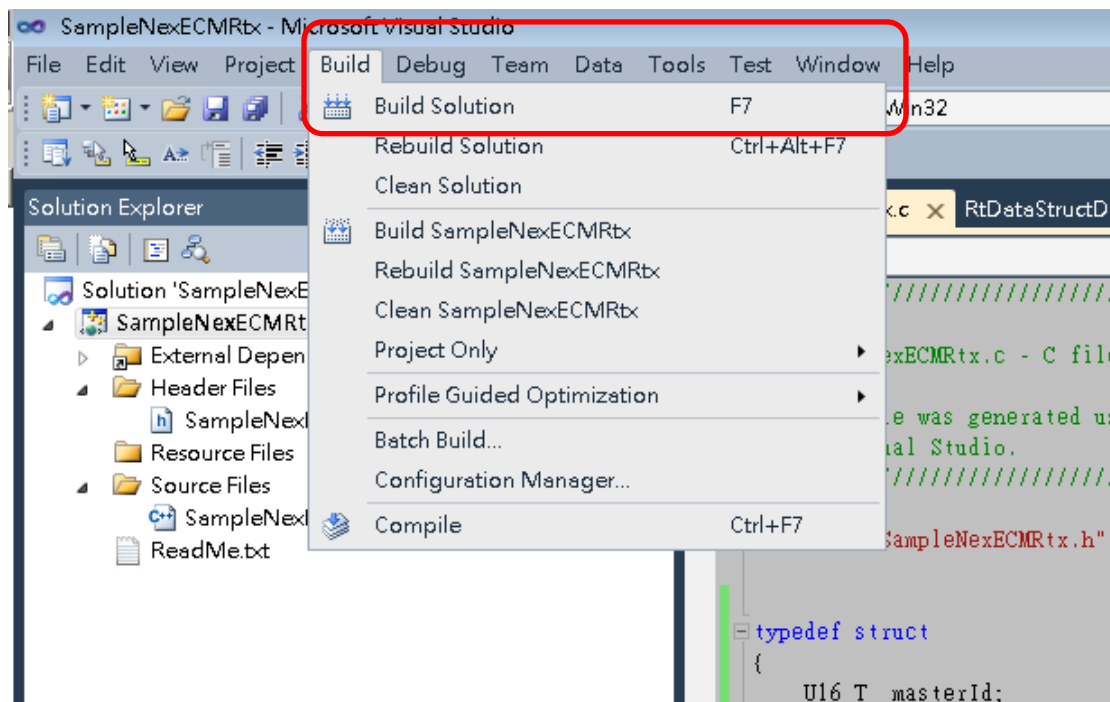
在 Linker\Input 中设定引入的函数库 NexECMRtx.lib



开发程序，编译程序代码，注意要选择 RTSSDebug 或 RTSSRelease



## 编译 RTX 应用程序



### 3. EtherCAT 公用程序说明

#### 3.1. NexCAT 简介

透过公用程序 NexCAT，可完成下列事项。

1. EtherCAT 网络装置扫描
2. 汇入 ESI 档案，输出 ENI 档案
3. CoE 从站模块(Slaves) PDO 映像(Mapping)
4. ProcessData 存取
5. CoE 从站模块(Slaves) SDO 通讯测试
6. EtherCAT 通讯质量监控测试
7. 从站模块(Slaves)功能性测试操作

##### 3.1.1. 软件需求

使用前请先确定 MS-Windows 是否已安装下列原件：

- Net framework 4.0 微软官方免费下载：

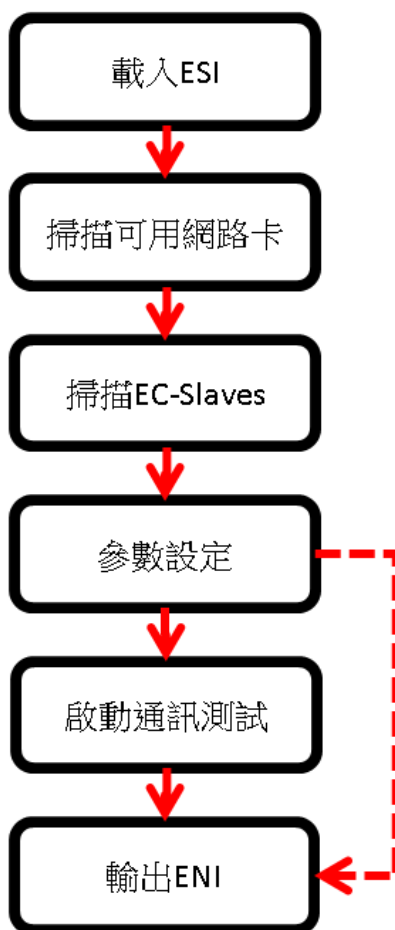
<http://www.microsoft.com/zh-tw/download/details.aspx?id=17718>

- Visual Basic Power Packs 3.0 微软官方免费下载：

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

## 3.1.2. NexCAT 操作流程说明

NexCAT 的基本操作流程如下图:



- **載入 ESI:**

由 NexCAT 于程序启动时完成汇入所有的档案，其路径为 NexCAT.exe 安装路径下的“ESI”子目录，其附文件名为\*.xml。

- **扫描可用网络卡:**

由 NexCAT 完成，程序可以自动判断在 Windows RTX 环境下，自动找到对应之网络卡之网络端口。关于 RTX 网络卡的安装设定请参考 Ch2.3 章节。

- **扫描 EC-Slaves:**

NexCAT 可以自动扫描用户所选择的网络端口，找到串连之 Slave 模块，当串连之模块无法由对应之 ESI 档内容找到对应之内容(比对 Vendor ID and Device ID)，该模块名称显示为未知 “Unknown”。将鼠标光标移置“Unknow”装置可以显示该模块的硬件信息(VendorID, DeviceID and RevisionNumber)

- **参数设定:**

NexCAT 依照 ESI 内容自动产生对应之 PDO 与 ProcessData 内存规划，并输出 ENI 档案。使用者可以利用内建之 PDO mapping 编辑器来进行规划。

- **启动通讯测试:**

规划完成之后可以直接启动所有的 Slave 模块，其状态由初始 INIT 至操作 Operation 状态。若其中有一个模块无法成功转至 Operation，主页面于区域 4、5 显示状态及讯息(参考下页图)。

- **输出 ENI:**

当测试完成各模块之动作皆正常时，使用者可以使用 Export ENI 档案的功能，输出至储存装置，倘若用户使用启动网络(Start Network)功能，系统自动输出当下的设定及网络拓扑至 ENI 文件。

(其默认路径为 : system root:\ENI\_NexCAT\_Export.xml)

### 3.1.3. ESI 和 ENI 档案

ESI (EtherCAT Slave Information)文件为硬件描述档案，其内容描述了硬件的信息及相关的参数及内存配置，其中包含该装置所支持的通讯协议。由提供 EtherCAT slave 模块的硬件厂商提供。

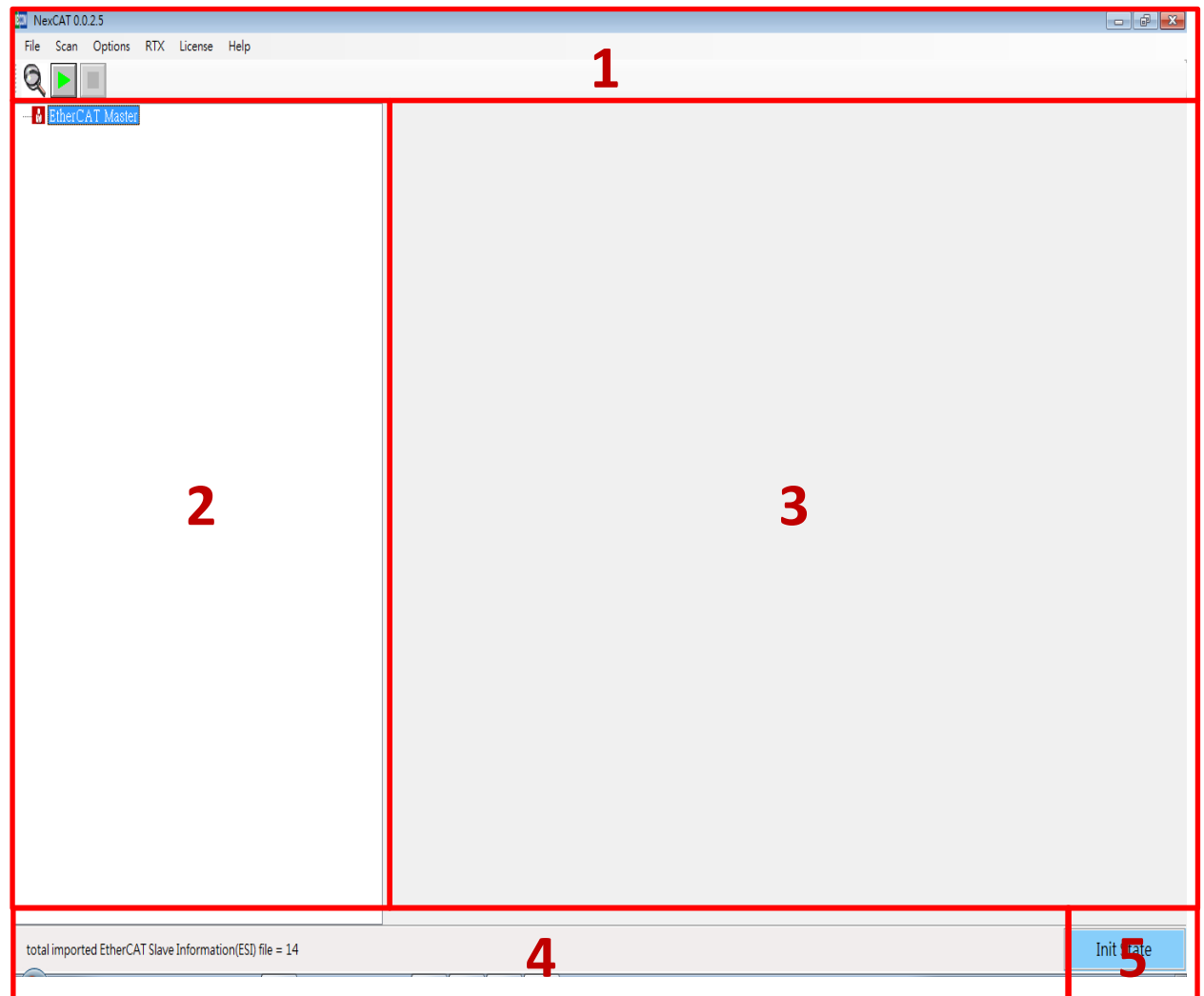
\*EtherCAT 公用程序对于模块之间的默认配置，系根据 ESI 的内容而来，因此强烈建议请勿更动 ESI 的内容，并确保所取得的 ESI 档案为该硬件的最新版本。

ENI(EtherCAT Network Information)为记载一个特定的 EtherCAT 网络拓扑及该网络拓扑下的 Process Data 内存配置和 EC-Slave 网络组态的描述档案，用户在完成 EC-Slave 模块串连配线之后，必须透过 NexCAT 公用程序，产生出 ENI 档案之后，EC-Master 便可依据其档案设定内容运行。




\*强烈建议请勿更动 ENI 的内容，若网络配置有异动请另外再输出更动后的 ENI 档案。

### 3.1.4. NexCAT 主页面操作

主页面共分成下列 5 个操作区域并分开说明于下列陈述:



- 区域 1:  
窗体的正上方显示软件版本信息。

Icon	说明
	Scan NIC:自动寻找可以用网络端口, 并显示于窗体上
	Start Network: 启动通讯并输出 ENI 档案到默认的路径(C:\)
	Stop Network:停止通讯自动停止所有 ECAT-Slaves

- 区域 2:  
显示整个网络拓扑(Topology)。完成扫描模块(Scan Slave)后所有在线的 EC-Slave 显示于此。倘若 EC-Slave 模块未能于所有 ESI 文件内找到对应的内容, 该模块显示名称为(Unknown)。此时请洽该模块供货商, 提供该模块之 ESI 文件, 并进行汇入之动作。



“Unknown”装置, 光标放置可显示 ID 信息

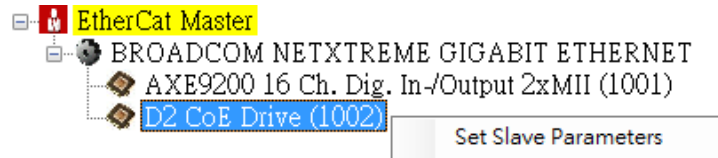
- 区域 3:  
设定参数选单之显示区。设定 Slave Parameters 及设定 Master Parameters. 页面显示区。
- 区域 4:  
讯息及错误代码显示区。
- 区域 5:  
Slave 模块状态显示区。目前共有 4 种状态显示模块状态:
  1. Initial(初始状态): 尚未启动网络时, 模块的初始状态。
  2. Error(错误状态): 启动网络时, Master 无法将所有 Slave 模块由 INIT 状态切换至 OP 操作状态。常见的错误为 ENI 档案与实际网络装置不符, ESI 版本与 Slave 装置版本不符等。
  3. Retry(重试状态): 当 ECAT master 的参数 “Link Error Mode” 若是设定为 “Auto re-connect” (自动重新连接), 当 OP 状态中的 Slave 模块遭遇断线时呈现 Retry 状态, Master 会针对断线模块进行重新联机的动作, 其它模块还是可照常运作, 此状态持续显示到该断讯模块重新连接工作正常为止。



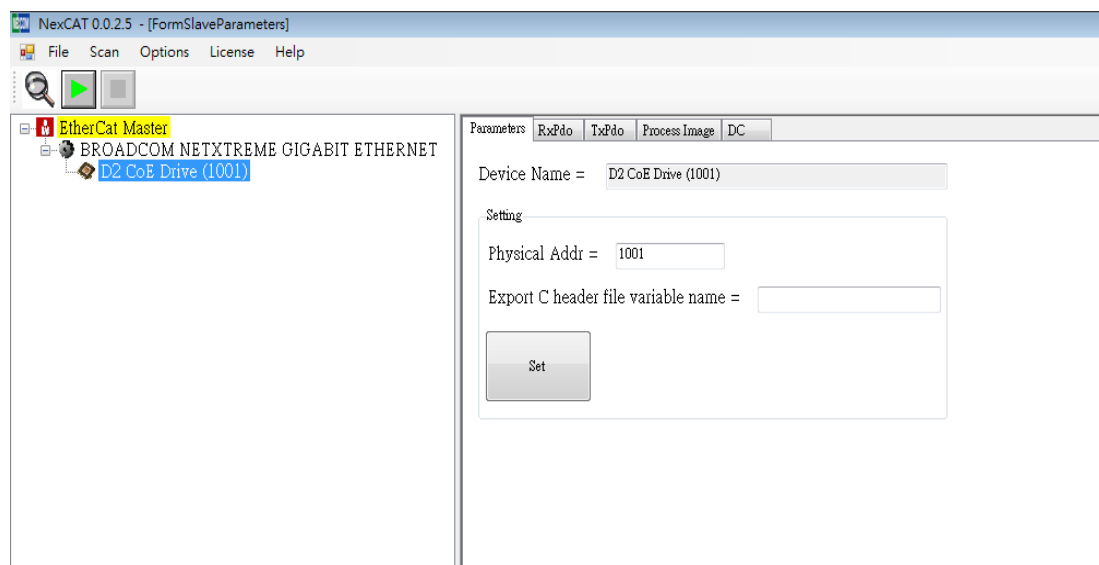
4. Running(工作状态):启动网络之后顺利转换至工作状态(OP)。

### 3.1.5. 从站模块参数设定页面 (Set Slave Parameters)

点选模块的名称，按下鼠标右键出现弹跳选单(pop-up menu)，选择 set slave parameters 即可。



模块设定页必须于启动网络(start network)之前才可以被使用，因为必须于启动网络之前，将所有参数设定完成之后，设定才会生效，若启动网络之后，所更改之设定值不会马上生效，必须重新启动网络之后才会生效。



## 1. 参数页 Parameters Page

Device Name =

Setting

Physical Addr =

Export C header file variable name =

**Device Name**: 显示目前设定页所设定之模块名称。

**Physical Addr**: 设定 Slave 模块之 node address (configured address)。

**Export C header file variable name**: Define 变量名称，此功能必须配合 Master Parameters 设定之 Export C file 功能一起使用，其功能为输出各模块之下内存配置(Process Image)之变量名称如下列所示: **请注意内容有无空格符**

#define \_Physical Addrsss (+ **variable name**)\_ObjectName [内存位置]

范例:

**Export C header file variable name** = “\_AXIS”

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

输出 C header file 请参考下小节说明

## 2. RxPdo & TxPdo Page

The screenshot shows the NexCAT 0.0.2.5 software interface. The main window is titled 'NexCAT 0.0.2.5 - [FormSlaveParameters]'. It features a menu bar (File, Scan, Options, License, Help) and a toolbar. On the left, there is a sidebar with a tree view showing 'EtherCat Master' and 'BROADCOM NETXTREME GIGABIT ETHERNET' with a sub-entry 'D2 CoE Drive (1001)'. The main area is divided into two panels: 'Parameters' and 'TxPdo'. The 'Parameters' panel contains a table with columns: RxPdo Name, Index(Hex), SM, Mandatory, and Fixed. The 'TxPdo' panel contains a table with columns: Entry Name, Index(Hex), Sub Index, Bit Len, and Data Type. The status bar at the bottom indicates 'found 1 slave module' and 'Init State'.

RxPdo Name	Index(Hex)	SM	Mandatory	Fixed
RxPDO 1	1600	2	-1	0
RxPDO 2	1601	-1	-1	0
RxPDO 3	1602	-1	-1	0

Entry Name	Index(Hex)	Sub Index	Bit Len	Data Type
Constantwert	6040	0	16	UINT
Target Position	607A	0	32	DINT
	0000	0	0	
	0000	0	0	
	0000	0	0	

表格字段说明:

**RxPdo(TxPdo) Name:** 预设的名称是由 ESI 的内容所提供, 使用者可以于字段内更改其名称后, 输出至 ENI 档。

**Index:** CoE 提供的参数值, 不建议使用者自行更改其设定值。

**SM:** 对映之 **Sync Manager** 号码, 可以由用户自行定义更改。

**Mandatory:** 定义是否为必要之参数。

**Fixed:** 定义该参数是否可以被使用者更改。

**Entry Name:** 其名称是由 CoE 所提供之名称, 可以更改名称, 输出至 ENI。

**Index:** CoE 提供的参数值, 不建议使用者自行更改其设定值。

**Sub Index:** CoE 提供的参数值, 不建议使用者自行更改其设定值。

**BitLen:** CoE 提供的参数值, 不建议使用者自行更改其设定值。

**Data Type:** CoE 提供的参数值, 不建议使用者自行更改其设定值。

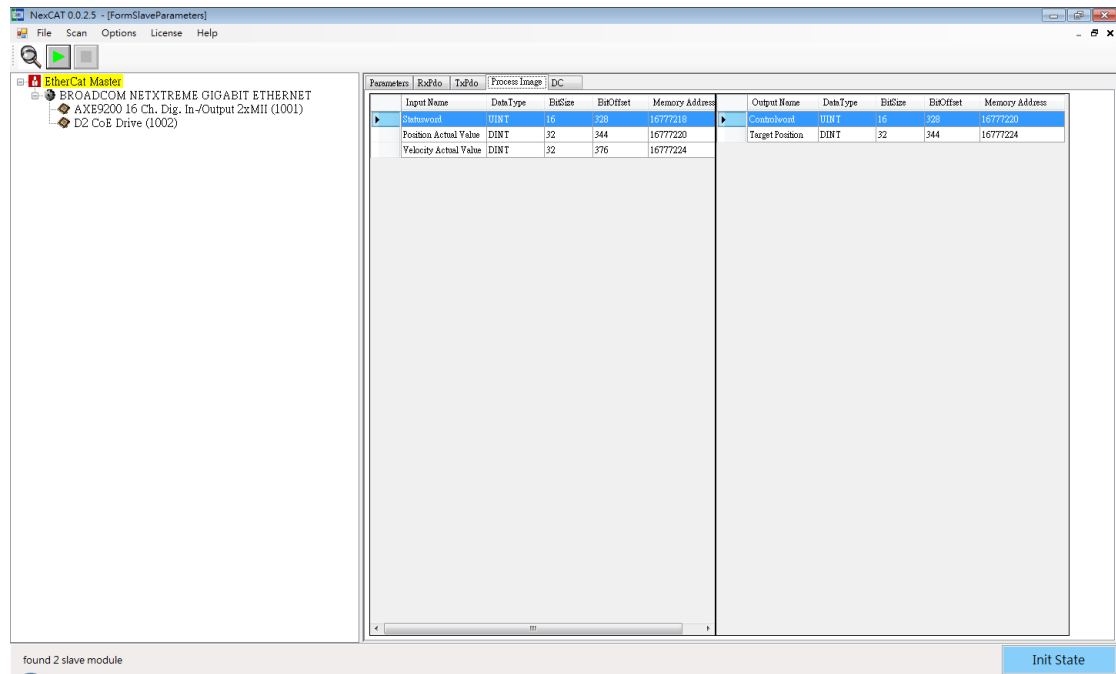
**Save Button:** 编辑完成之后, 必须要储存后才生效。

**Default Button:** 可回复成原始 ESI 的设定。

**Clear All Button:** 清空 PDO 的设定。

### 3. Process Image Page

使用者可以于 RxPdo 或 TxPdo 编辑页面，编辑完之后至此页面观查它对应之内存位置，编辑完之后必须要储存之后才会生效。



表格字段说明:

**Input(Output) Name:**依据 RxPdo 或 TxPdo 所设定的名称。

**BitSize:** 变量所存在之内存位置大小。

**BitOffset:** 依据 RxPdo 或 TxPdo 所设定。

**Memory Address:** 变量所存在之内存位置。

## 4. DC

设定 Slave 模块的 DC 模式，此页设定内容的默认值，系由各 Slave 的 ESI file 所提供之信息

The screenshot shows the 'DC' configuration tab in the EtherCAT Master software. It contains a 'Setting' section with the following values:

- Mode: DC
- Description: DC SYNC0
- DC SYNC Activation: 0x0300

Buttons for 'Apply To Other' and 'Set' are located at the bottom of the setting area.

### Mode (Description):

选择 DC 模式。若 Slave 支持 DC 模式，则默认为启动“DC”。当网络拓扑中有任一 Slave 启动 DC 功能，EtherCAT Master 将输出有 DC 信息(功能)的 ENI File；欲关闭 DC 功能，用户必须将所有模块的 DC 选项关闭，设定为 FreeRun mode，ENI 档的输出内容将没有 DC 的相关描述。

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

0x0000 – Disable SYNC0 & SYNC1 (Free Run)

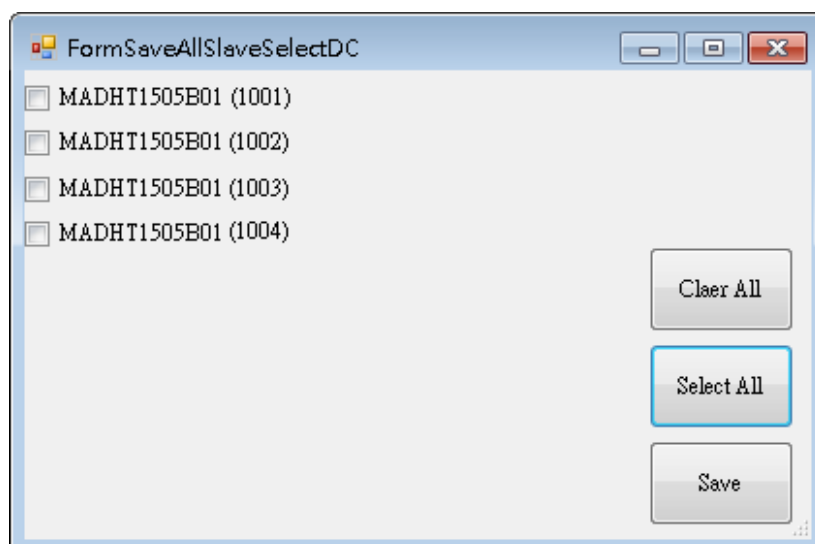
0x0300 – Activate SYNC0 (DC Sync)

0x0700 – Activate SYNC0 & SYNC1

此为进阶设定，选择 DC 模式时此栏会根据 ESI 内容显示模式对应之设定值，用于控制 Slave 内部 SYNC 中断讯号，一般依照默认值即可。

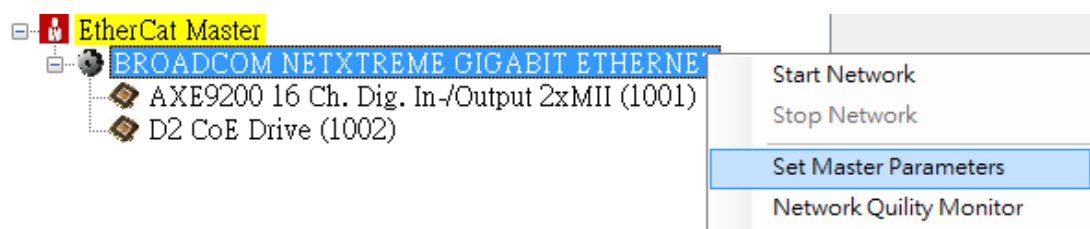
### Apply To Other:

可将此设定套用到网络拓扑上其他相同型号之 Slave。(如下图)



### 3.1.6. EC-Master 参数设定页面 (Set Master Parameters)

点选树形图上网络卡的名称，按下鼠标右键出现弹跳选单(pop-up menu)，选择 set master parameters。

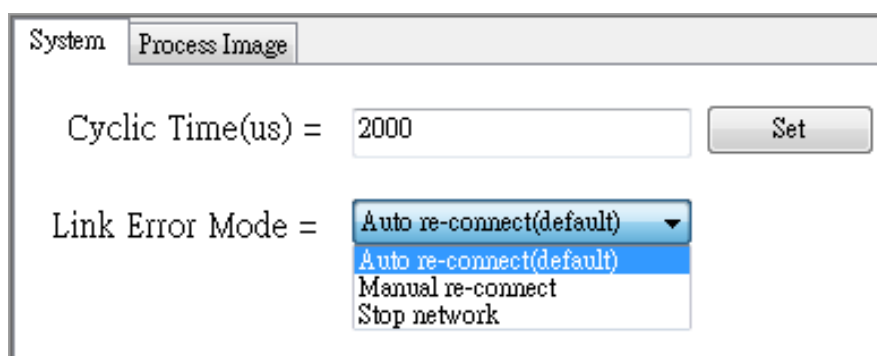


共有两个子页面：

1. System
2. ProcessImage

描述如下：

#### System: 系统设定页面



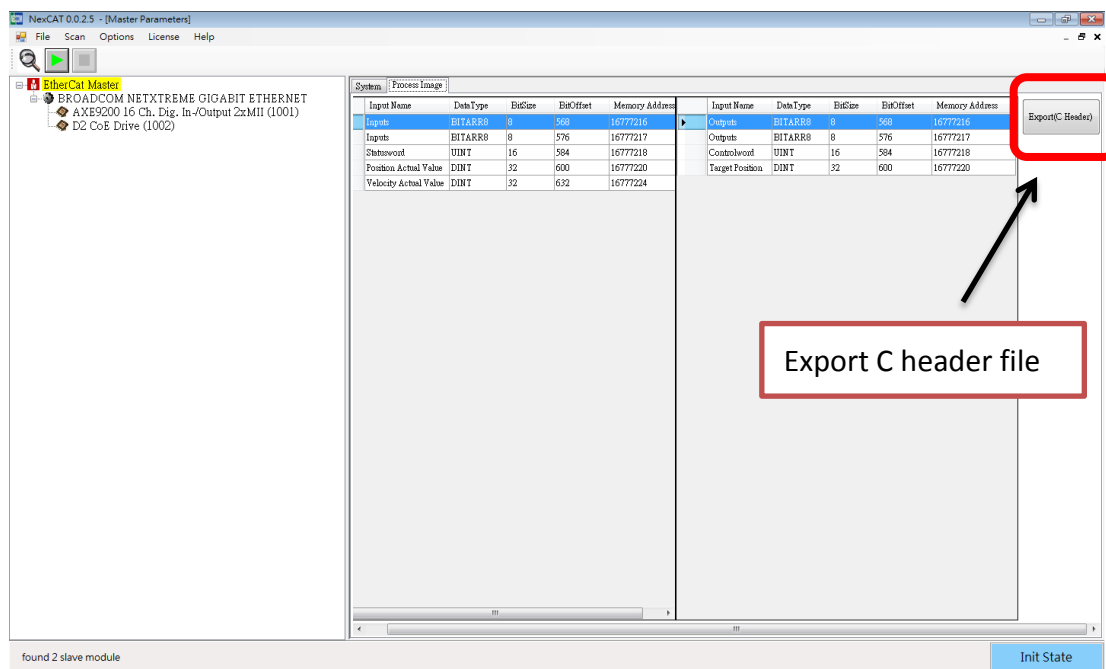
**Cyclic Time:** EC-Master 通讯周期，单位为 micro second(us)，指定系统的通讯

周期，与模块间数据的沟通时间或更新频率，最小数值不能超出系统限制，可透过呼叫 API 设定。请参考 6.2.5 NEC\_RtSetParameter()

**Link Error Mode:** 联机错误处理机制，启动网络(start network)之后，模块进入操作模式(Operation state)，当 Master 侦测联机中断时所进行的处理机制。共有三种模式可供选择，如下：  
此模式所对应之 API 请参考 6.2.5 NEC\_RtSetParameter()

- **Auto re-connect(default):** 当模块通讯中断，主页面区域 5 出现 Slave Retry 讯息，同时系统自动对失去通讯的模块重试联机，一直持续至联机成功为止，其它模块继续正常工作。
- **Manual re-connect:** 当模块通讯中断，其它模块继续正常工作，主页面区域 5 出现错误讯息，持续至下次启动网络联机成功为止。
- **Stop network:** 当模块通讯中断，EC-Master 将停止网络联机，主页面区域 5 出现错误讯息，持续至下次启动网络联机成功为止。

## ProcessImage 页面



- **Network process image map**  
与 3.1.5 之章节所描述的 ProcessImage page 内容格式相同，在此可以看到整个网络拓扑产生出之内存配置位置

- 输出 C header file for process image map

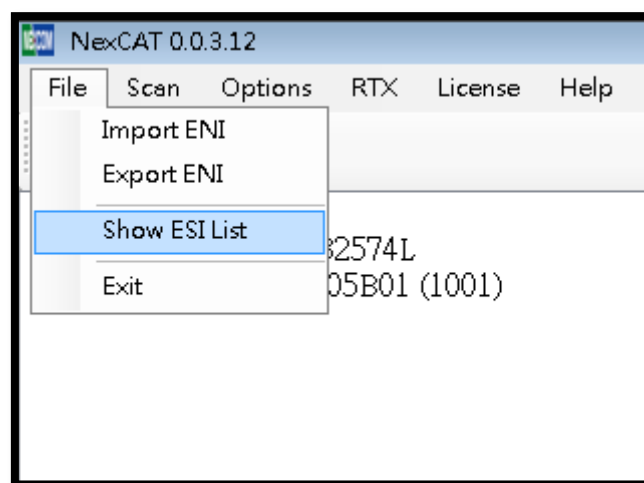
点击 “Export C Header File”按钮 可输出各从站模块之 PDO mapping 在 Process Image 中的偏移量(offset)定义，撰写程序时透过使用 define symbol 的方式对该内存进行存取，当 PDO 顺序改变时，只需从新输出此档案达成程序代码的易维护性及可读性。输出形式可参考 3.1.5 小节从站模块参数设定。

### 3.1.7. ESI 档案管理窗体 (ESI List)

使用 NexCAT 进行扫描网络时，可以得到线路上有多少从站装置并取得该装置的 Device ID，透过比对 ESI 档案来辨识该硬件装置(可参考 3.1.3 小节)。若用户取得一新的 ECAT slave 装置，必须将该 ESI 档案汇入到 NexCAT 中。

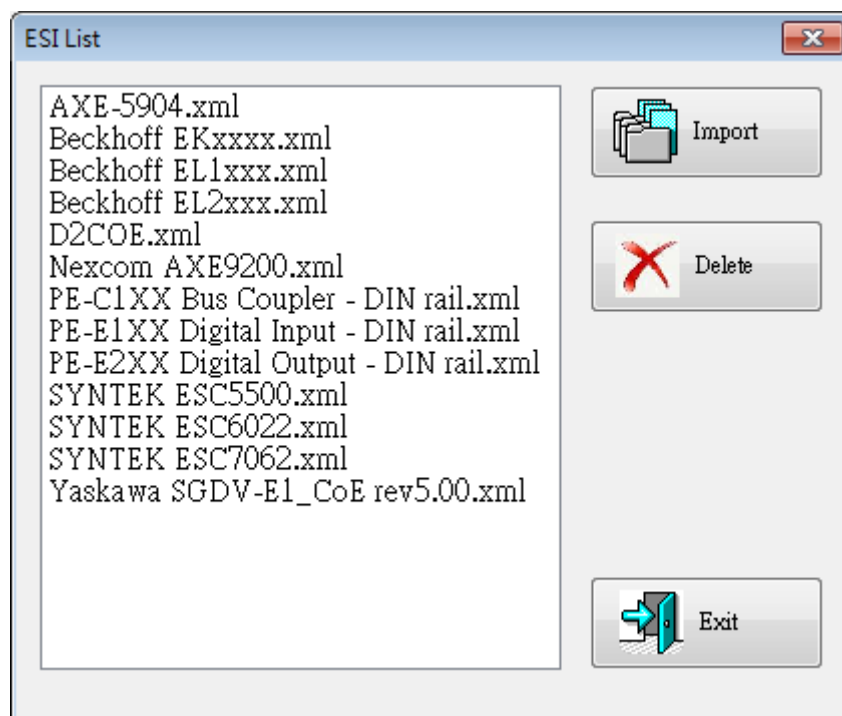
NexCAT 提供了 2 种方式来管理 ESI 档案：

1. 用户直接将 ESI 档案放置在 NexCAT 指定的目录下，从新启动 NexCAT，NexCAT 会于启动过程中加载文件夹中所有的 ESI 档。
2. 使用“ESI List” 档案管理窗体：当使用者欲新增/删除 ESI 文件，可透过此接口来操作。当用户编辑完成之后按下 Exit 系统自动重新整理 ESI 文件夹内之档案，直接重新扫描即可，不须重新启动程序。



开启 ESI List



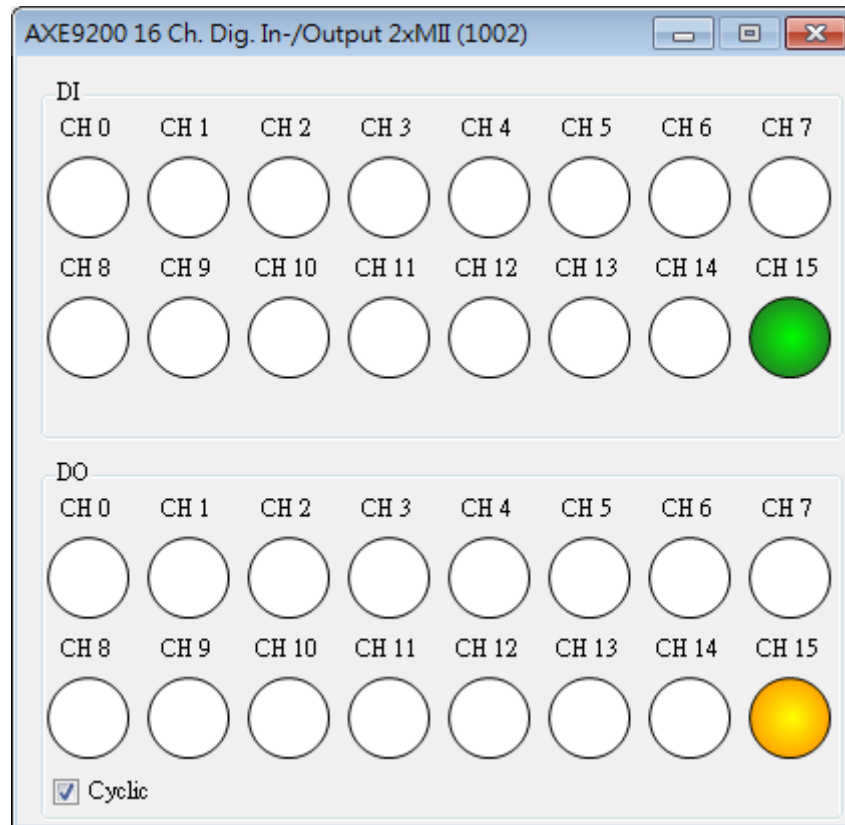


ESI List 画面

### 3.1.8. DIO 操作页面

用户于主页面之区域 2，点选须要操作之 DIO 模块，快速点二下鼠标左键则 DIO 操作窗体自动出现画面中央。程序会自动判定该模块为 DI、DO 或是 DIO 模块并自动计算其 IO 数量。于画面中自动呈现。

当鼠标光标移至 DO 的按钮处，用户可以手动压按 DO 按钮，即可操作 DO，亦或者，用户可以使用 Cyclic 功能，让 DO 模块自动运行启动跑马灯功能，该模块由第 0 个信道(Channel)开始，由小至大重复运行。当 Cyclic 的框选处被打勾后自动运行，勾选取消则程序自动停止停在运行时执行的最后一个通道。

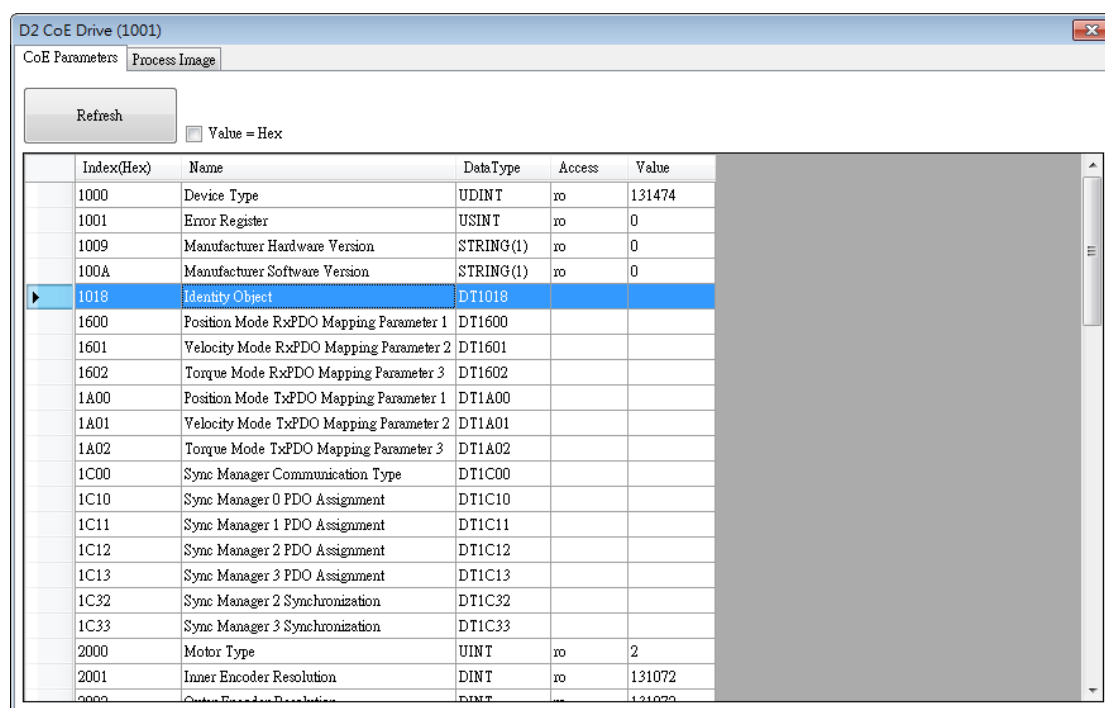


### 3.1.9. CoE-SDO 存取操作页面

用户于主页面之区域 2，点选须要操作具 CoE 功能之模块，快速点下鼠标左键则 CoE 操作窗体自动出现画面中央。程序会自动判定该模块为是否支持 CIA 402 之模块。

按下“Refresh”按钮后表格内之参数值将自动更新，使用者可自行选择以 10 进制表示或者以 16 进制表示，若有某参数为浮点数，则该参数不受进位制影响一律以浮点数表示之。

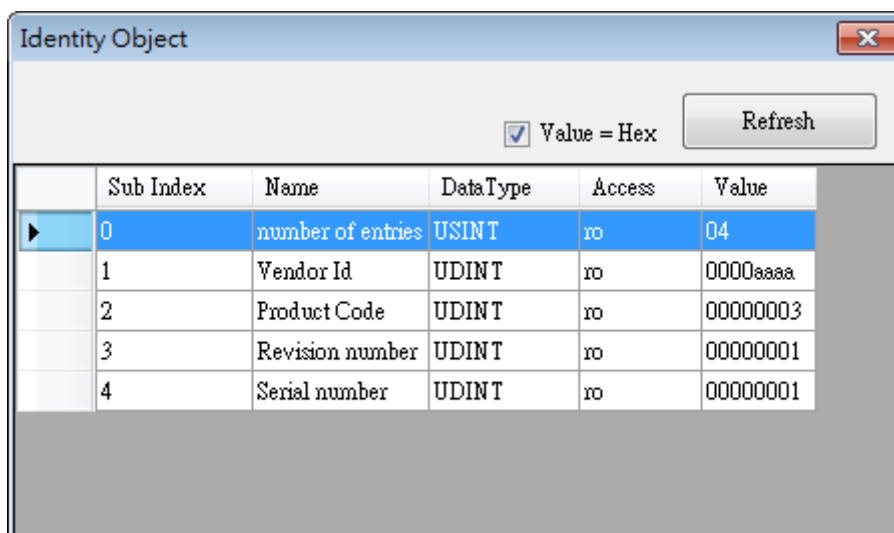
倘若用户欲改变参数值，可以使用鼠标，于 Value 的表格内快速点二次鼠标左键，可编辑该参数值，编辑完成后，按下 Enter 键或离开该表格即可成功写入，若写入失败或写入格式不符合规范的数据形态，则该参数值自动回复成为编辑前之状态。



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

CoE parameters

若参数的数据型态为 DataType 时，表示该参数包含了子参数(Sub Index)，使用者可以于想要存取之参数快速点鼠标左键二次，若程序判断其底下确实有子参数，会出现子窗口如下图所示，其值之读取与写入与之前章节所提及到的方式相同。



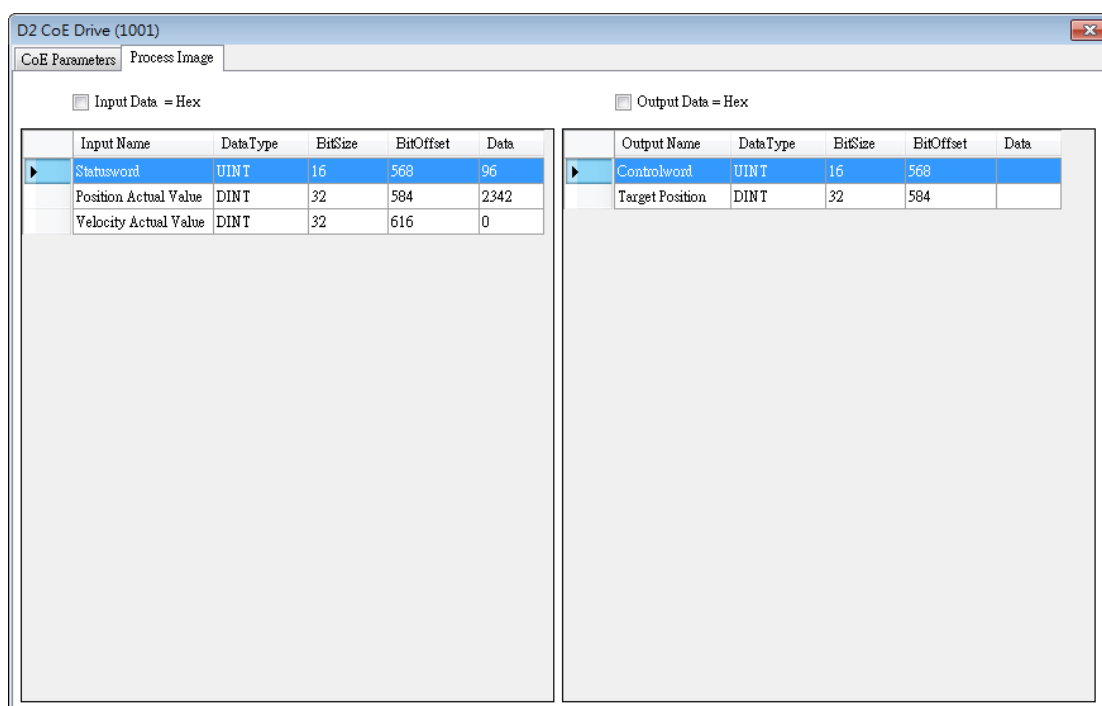
Identity Object

☒ Value = Hex    Refresh

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

Sub parameters

### 3.1.10. Process Image 参数存取操作页面



D2 CoE Drive (1001)

CoE Parameters    Process Image

☐ Input Data = Hex    ☐ Output Data = Hex

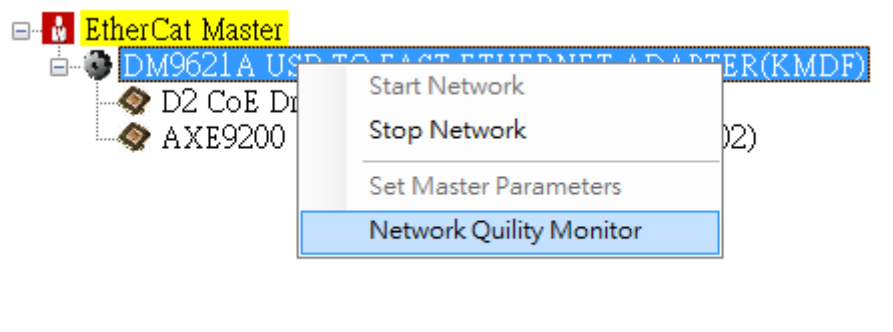
Input Name	Data Type	BitSize	BitOffset	Data
▶ Statusword	UINT	16	568	96
Position Actual Value	DINT	32	584	2342
Velocity Actual Value	DINT	32	616	0

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

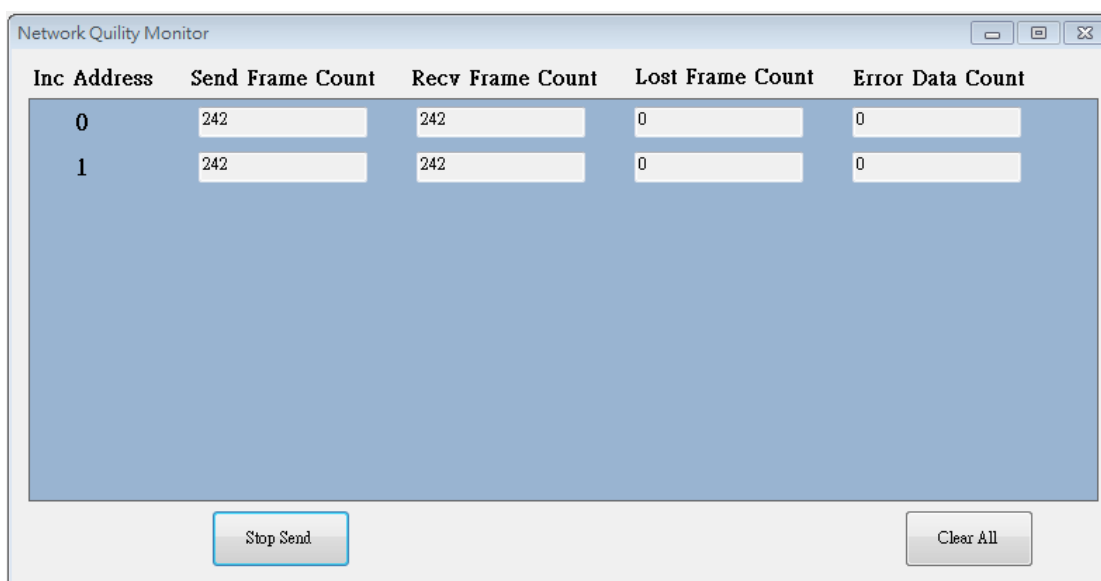
用户可于成功启动网络(start network)之后，存取 pdo(process data object)数据其显示方式，当 Input data(output data) = hex 的框选栏被勾选时，其数据于表格内所示之数值为 16 进制，反之为 10 进制的表示方式。

### 3.1.11. Network Quality Monitor 网络通讯质量测试页面

网络成功启用之后，用户可以开启网络通讯质量测试页面，进行 Master 端对于各模块的通讯封包测试。欲开启此页面，可以使用鼠标于主页面的区域 2 之 NIC 的节点上，按下鼠标右键后，将会出现一个弹跳窗口(pop-up menu)，可以点选 Network Quility Monitor 的选项后，出现网络质量测试页面。



NIC 的节点上，按下鼠标右键



Network Quality Monitor 页面

- Inc Address：模块之 Slave ID，系依照所扫描到的模块顺序排列之。
- Send Frame Count:系统对模块发送之测试封包数量，读取 Slave 模块的状态是否为“OP”状态，系统发封包的频率为 10 ms，实际的发送速度依照系统当时的效率。
- Recv Frame Count: Slave 模块响应的封包数量，正常的情况之下，每发送一个测试封包 Slave 模块需响应一个封包。

- Lost Frame Count: 遗失的封包，当发送出去封包，其模块无响应该封包的数量。
- Error Frame Count: 检查 Slave 响应之封包内容信息，若其状态不为“OP”状态则 Error Frame Count + 1。

其关系如下列式

Send Frame count = Recv Frame count + Lost Frame count

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

### 3.2. NexECMRtxStartup 启动程序说明

NexECMRtxStartup.exe 为提供你在使用 EtherCAT 主站(EC-Master)的便利性。此程序根据 NexECMRtxConfig.ini 的内容，提供以下主要的三个功能：

1. 载入 EtherCAT 主站(EC-Master) - NexECMRtx.rtss
2. 下载 ENI (EtherCAT Network Information) 网络信息文件
3. 加载用户 RTX 应用程序 (例如:UserRTXApp.rtss)

你可透过记事本或其它文字编辑软件修改 NexECMRtxStartup.ini 内容以符合你当前的档案放置情形；通常须修改 1.应用程序(Application)的位置，2.网络信息文件(ENI)的位置。你可在 “C:\Program Files\NEXCOM\NexECMRtx\tools” 此位置找到此 ini 档案，请参考以下图解说明。



NexECMRtxConfig.ini 的内容

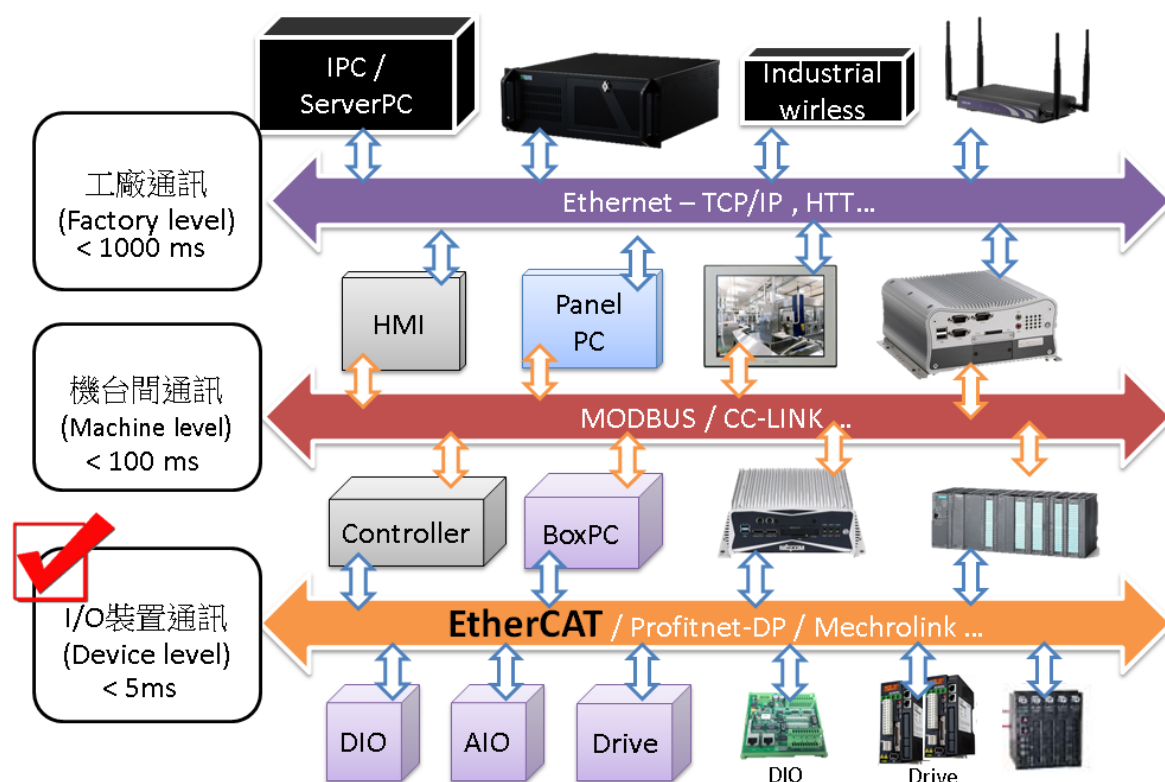
标识符	说明
<b>PATH_ENI</b>	
PATH:	网络信息文件(ENI)档案位置。 OPTION: 是否使用 ENI 中，网络卡的信息 0:使用 ENI 中的网络卡信息 1:不使用 ENI 信息，采用 Parameter 的设定
<b>PATH_NEXECMRtx_DRIVER</b>	
PATH:	NexECMRtx.rtss 档案位置。
<b>PATH_USER_APP (Option)</b>	
PATH:	填入你的 RTX 应用程序(*.rtss)的路径位置与文件名。



## 4. 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。

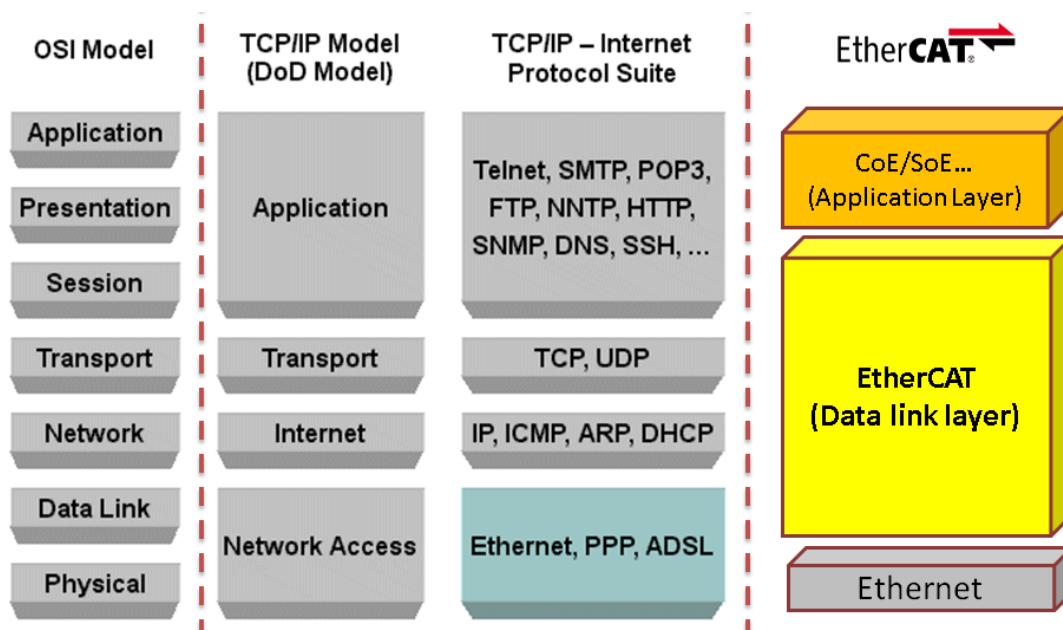


下面几个小节简单说明 EtherCAT 通讯技术及其特点。

## 4.1. EtherCAT 通讯协议概述

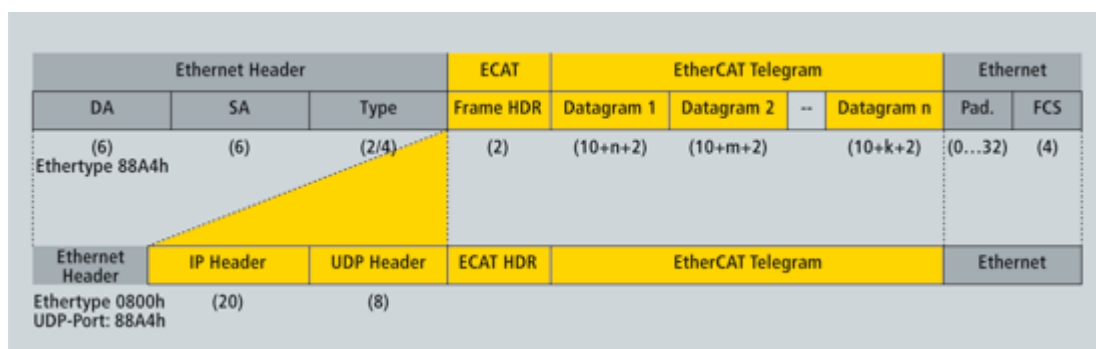
EtherCAT 是一个架构在 Ethernet 物理层(Physical Layer)上具实时性(Real-time)的通讯协议，目前由 ETG (EtherCAT Technology Group)组织来维护及推广。

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。

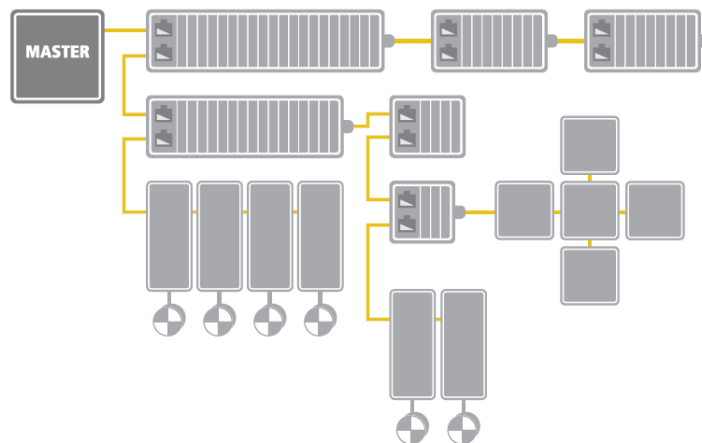
EtherCAT 的内文部分主要包含了 ECAT 的标头(Frame header)以及后面的报文 (Telegrams)。



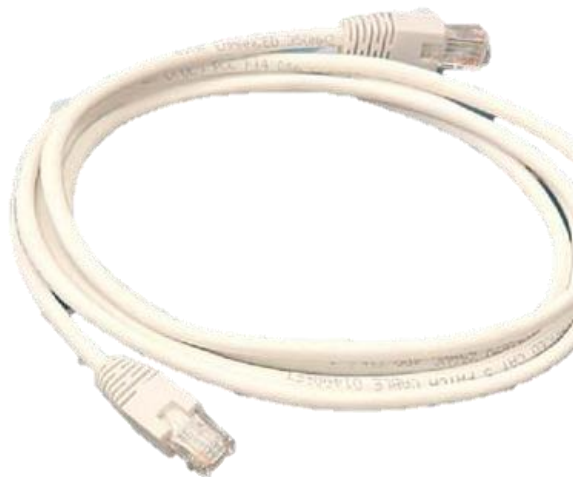
EtherCAT 网络封包格式 (数据源: <http://www.ethercat.org/>)

## 4.2. 网络拓扑

网络拓扑特性关系到现场布线的可能性。EtherCAT 几乎支持所有网络拓扑形式包含直线，树状及星状等拓扑，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有任何限制。



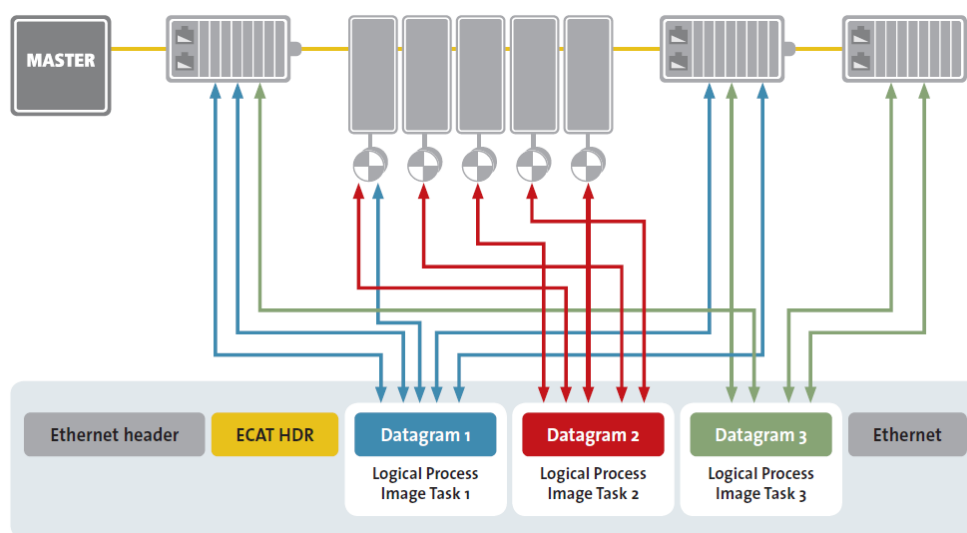
EtherCAT 网络拓扑, Line, Tree and Star 拓扑  
(数据源: <http://www.ethercat.org/>)



使用标准 Ethernet 网络线

## 4.3. 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：

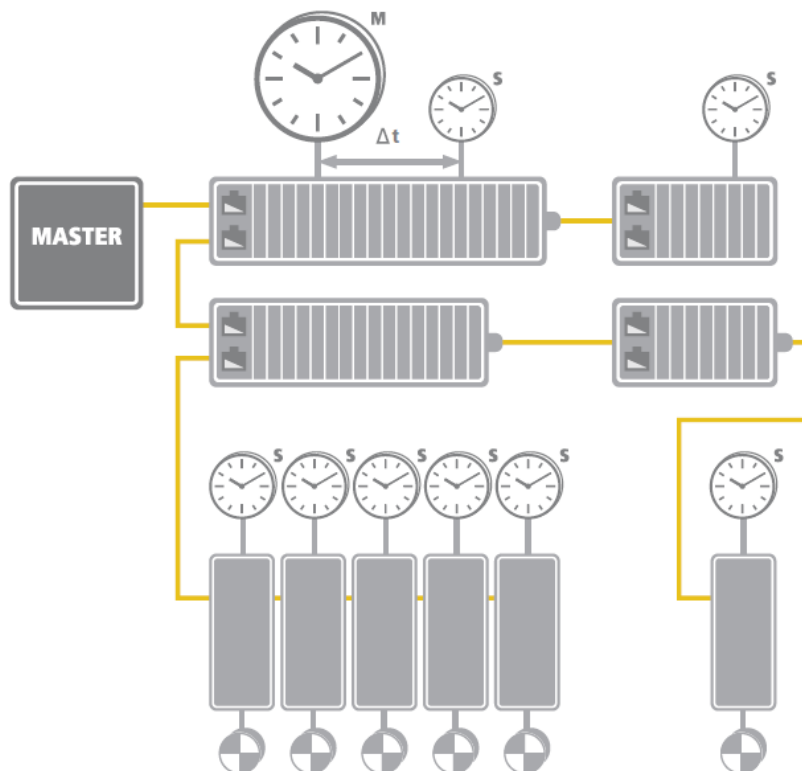
资料量	更新时间
256 点 I/O	11μs
1000 点 I/O	30μs
200 Channels 類比 I/O (16 位)	50μs ( =20kHz )
100 个伺服轴 (8 Bytes Input and output per axis)	100μs
1 个现场汇流排主网关 (Fieldbus Master-Gateway) (1486 字节输入与 1486 字节输出资料)	150μs

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。

#### 4.4. 网络同步

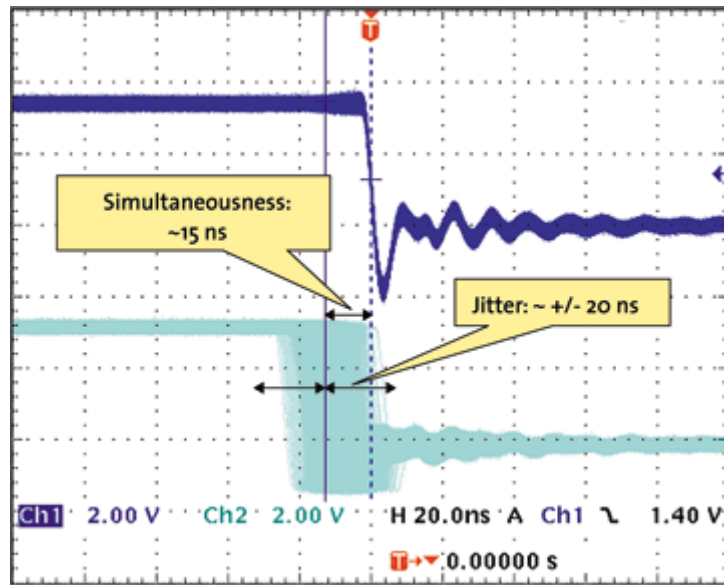
在设备自动化应用中对分布式网络的要求除了网络通讯需具备实时性(Real-time)外，其装置之间的同步性要求不可或缺，例如控制多轴伺服马达的直线或圆弧补间运动，龙门(Gantry)轴控制等。这些应用必须具备网络同步性才能确保应用的正确性。



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)，其时间最小间隔为 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<sup>(\*)</sup>，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 $\pm 20$  奈秒(nano-second)之内，如下图两个从站间 I/O 信号误差约为 20 ns。

<sup>(\*)</sup> 可参考 EtherCAT standard document ETG1000.4

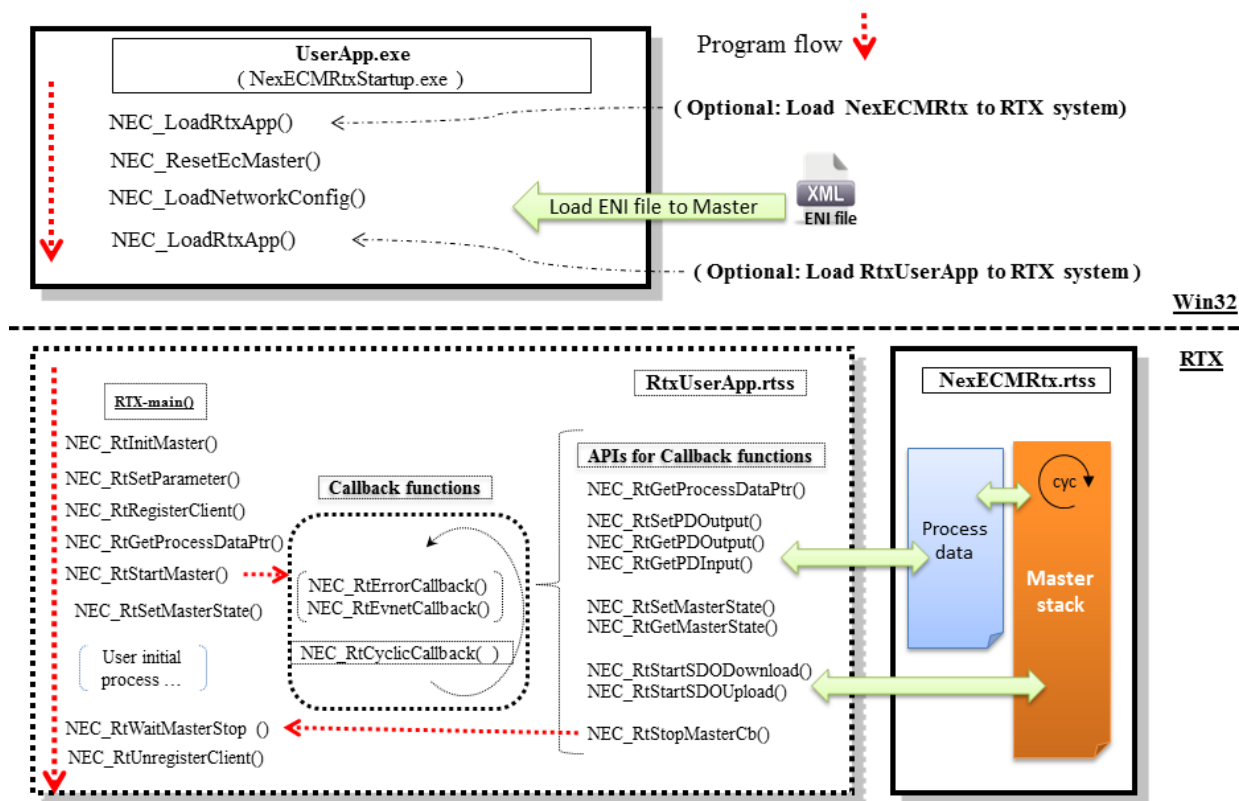


Sync 信号误差 (数据源: <http://www.ethercat.org/>)

## 5. 编程原理

### 5.1. 基本编程框架

下图说明 NexECMRtx 基本编程流程。



基本流程如下

步骤	说明	相关函数
您的 Windows 上层(UserMode)应用程序: ex. <u>UserApp.exe</u> (详细 API 说明请参考 CH.7)		
1	加载 NexECMRtx (EtherCAT 主站通讯层)	NEC_LoadRtxApp()
2	重置 NexECMRtx	NEC_ResetEcMaster()
3	加载 ENI 网络配置文件案	NEC_LoadNetworkConfig()
4	加载您的 RTX 应用程序 (如上图 RtxUserApp.rtss)	NEC_LoadRtxApp()
您的 RTX 应用程序主程序 main(): ex. <u>RtxUserApp.trss</u> (详细 API 说明请参考 CH.6)		
5	初始化 EC-Master	NEC_RtInitMaster()
6	设定参数, 如主站通讯周期(Cycle-time)等	NEC_RtSetParameter() NEC_RtGetParameter()
7	注册 Callback 函数	NEC_RtRegisterClient()



		NEC_RtUnregisterClient()
8	取得 ProcessImage 的内存指针 (Memory pointer)	NEC_RtGetProcessDataPtr() NEC_RtGetSlaveProcessDataPtr()
9	启动 EC-master 通讯	NEC_RtStartMaster()
10	将 Ec-Master 的状态切换至“OP”状态	NEC_RtSetMasterState()
11	RTX 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： 1. 让主程序进入睡眠等待 EC-master 通讯结束或 2. 您的其他(装置)控制应用程序	NEC_RtWaitMasterStop()
在周期 Callback 函式中 (详细 API 说明请参考 CH.6)		
12	你的控制程序进行 1. ProcessData 存取 2. CoE 通讯	NEC_RtSetPDOOutput() NEC_RtGetPDOOutput() NEC_RtGetPDInput() NEC_RtStartSDODownload() NEC_RtStartSDOUpload()
13	应用程序结束，停止 EC-Master 通讯	NEC_RtStopMasterCb()

上述可流程可参考 NexECMRtx 安装目录中的范例程序。

## 5.2. ENI 载入

ENI 网络档案(EtherCAT Network Information)须由您的 Win32 程序进行下载，下载前请先将 NexECMRtx.rtss 加载 RTX 系统中。加载方式分为下列 2 种：

1. 手动载入：鼠标左键双击“NexECMRtx.rtss”
2. 呼叫 NEC\_LoadRtxApp() API 载入

载入 NexECNRtx.rtss 后，呼叫 NEC\_LoadNetworkConfig() 下载 ENI 档。当 ENI 档案成功载入后，EC-Master 将保留这些加载的数据在其内部存储器中。当 EC-Master 启动通讯时会透过这些数据来设定 EtherCAT 从站(EC-Slaves)。启动过程中 EC-Master 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 NexCAT 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第三章。

完成加载 ENI 档案到 EC-Master 后, 你可以接着启动您的 RTX 应用程序。并在你的程序中对 NexECMRtx 进行通讯的控制及操作, 请继续参与后续小节的说明。

上述的步骤你也可以使用 *NexECMRtxStartup.exe* 工具程序来完成, 在程序文件夹中 INI 档案中设定:

1. ENI 档案的路径
2. NexECMRtx 的路径
3. 您的 RTX 应用程序路径

详细的功能设定请参考 3.2 章的说明。

设定好后, 执行该程序会自动完成上述加载 ENI 的动作, 并启动您的 RTX 应用程序。

### 5.3. 初始化 NexECMRtx

在您的 RTX 应用程序中, 呼叫 *NEC\_RtInitMaster()* 函数来初始化 EC-Master。此动作将不会重置您从 Win32 应用程序中下载的 ENI 的数据数据。

呼叫 *NEC\_RtSetParameter()* 函数来完成配置 EC-Master。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。而其他参数请参阅该函数的说明。

使用 *NEC\_RtRegisterClient()* 来注册一组个客户端的 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 EC-Master 呼叫
NEC_RtErrorCallback	当错误事件产生时会被 EC-Master 呼叫

这些 Callback 函数用于与 EC-Master 进行的同步通信与 ProcessData 数据交换。当 EC-Master 在状态机(State-Machine)<sup>(\*)</sup>更改为“SAFEOP”和“OP”的状态时，EC-Master 开始在每个通讯周期中将 ProcessData 数据更新传送到 EC-Slaves 并从 EC-Slave 将数据传回 ProcessData 中。

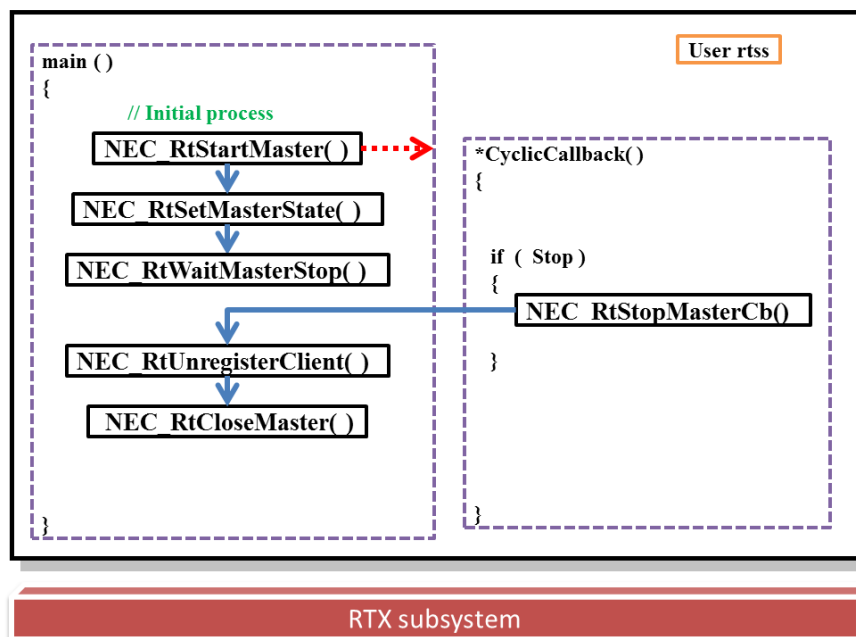
(\*)关于 EtherCAT 状态机，请参考 5.6 小节“EtherCAT 状态机”的说明。

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 函数中编程的基本原则是**避免**在 Callback 中撰写导致 Callback 停止或过于冗长的程序使 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 5.5 小节 Callback 函数。

## 5.4. 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()` 启动主站 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 EC-Master 的状态机会保持在“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，RTX 应用程序是一个类似主控制台(Console)编程的方式，RTX 程序有一标准的进入点 `main()`函数。当 `main()`函数返回时，该程序即结束并由 RTX 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 EC-Master 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。EC-Master 状态机将详述于 4.6 小节

## 5.5. Callback 函式

### 5.5.1. 注册回调函数(Callback functions)

NexECMRtx 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 EC-Master 通讯前，使用 *NEC\_RtRegisterClient()*将 Callback 函数注册 EC-Master 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。

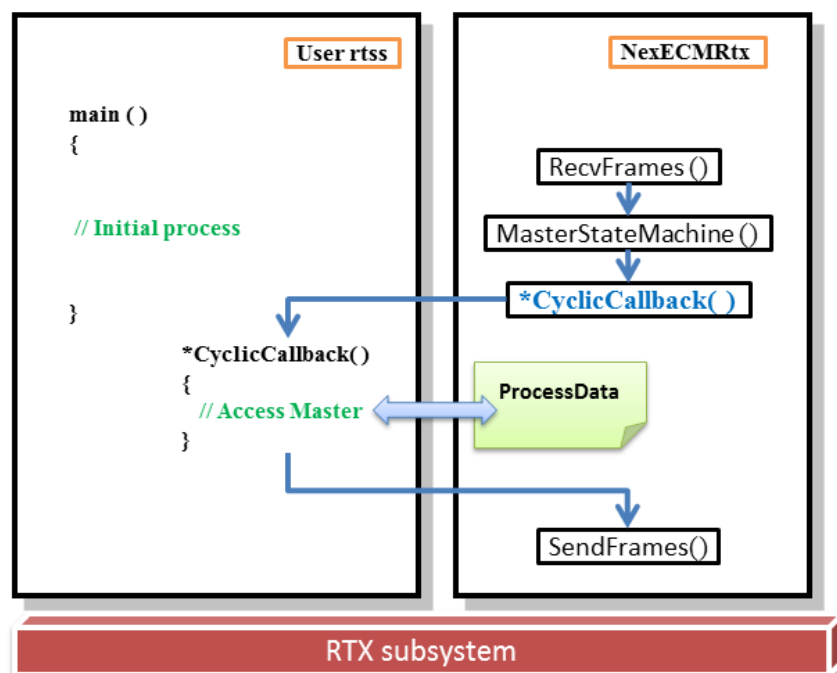
### 5.5.2. 周期回调函数(Cyclic-Callback function)

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 EC-Master 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 EC-Master 之间的数据交换管道。但 NexECMRtx 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 EC-Master 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 EC-Master 的周期通讯程序必须互相链接。

NexECMRtx 提供 Callback 函数的同步回调机制(synchronous callbacks)，使您的 Real-time 应用程序与 NexECMRtx 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 NexECMRtx 中，EC-Master 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。



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

当 EC-Master 内定的事件发生时，EC-Master 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。

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

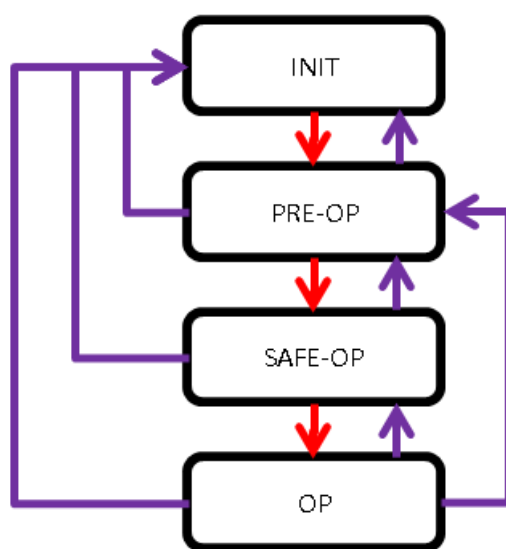
当 EC-Master 发生错误事件时，EC-Master 会同步呼叫此函数通知用户，同时传入错误代码(Error code)，使用者可以根据错误代码来处理对应的事件，或者忽略不处理。

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。

## 5.6. 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 EC-Master 网络组态和 EC-Slaves 模块相关设定，其设定的内容是根据 NexCAT 公用程序所输出的 ENI (EtherCAT Network Information) 档案。

### 5.6.1. ESM 状态变更

启动 EC-Master 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 EC-Master 状态，`NEC_RtSetMasterState()`函数只用于发出状态改变要求，并不会等待 EC-Master 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-Master 所提供的服务以及 EC-Slaves 可以接受的操作指令如下表所示：

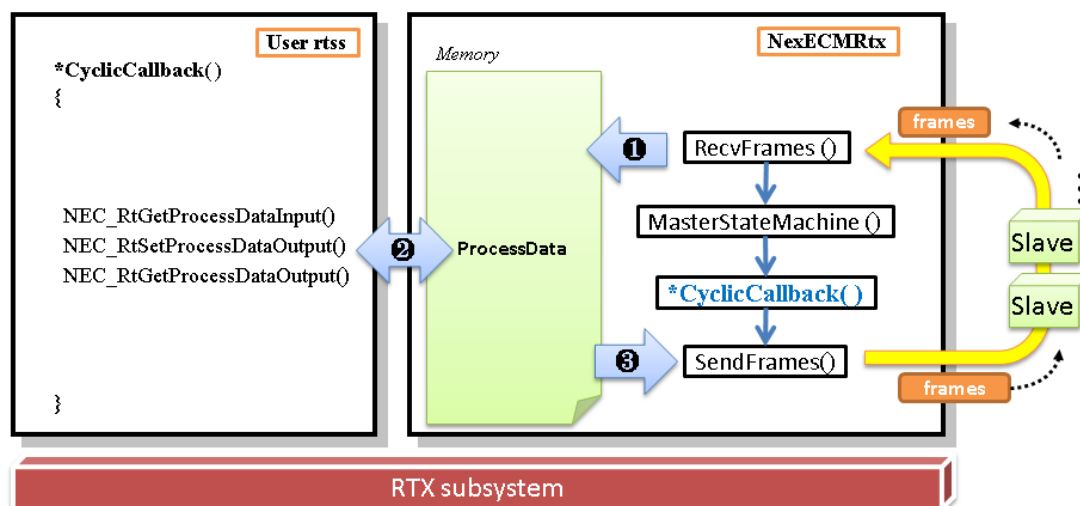
状态	EC-Master 服务	EC-Slave 服务
INIT	<ul style="list-style-type: none"> <li>➤ 周期回调函数启动</li> <li>➤ 无 ProcessData 通讯传输(To slaves)</li> <li>➤ 无 Mailbox 通讯传输(To slaves)</li> </ul>	<ul style="list-style-type: none"> <li>➤ EC-Master 不可控制</li> </ul>
PREOP	<ul style="list-style-type: none"> <li>➤ 周期回调函数启动</li> <li>➤ Mailbox 通讯传输启动(To slaves)</li> <li>-SDO 通讯启动</li> <li>➤ 无 ProcessData 通讯传输(To slaves)</li> </ul>	<ul style="list-style-type: none"> <li>➤ Mailbox 通讯启动</li> </ul>
SAFEOP	<ul style="list-style-type: none"> <li>➤ 周期回调函数启动</li> <li>➤ Mailbox 通讯传输启动(To slaves)</li> <li>-SDO 通讯启动</li> <li>➤ ProcessDataInput 通讯传输启动</li> <li>➤ 无 ProcessDataOutput 通讯传输</li> </ul>	<ul style="list-style-type: none"> <li>➤ Mailbox 通讯启动</li> <li>➤ 将 Input 数据更新到 ProcessDataInput</li> </ul>
OP	<ul style="list-style-type: none"> <li>➤ 周期回调函数启动</li> <li>➤ Mailbox 通讯传输启动(To slaves)</li> <li>-SDO 通讯启动</li> <li>➤ ProcessDataInput 通讯传输启动</li> <li>➤ ProcessDataOutput 通讯传输启动</li> </ul>	<ul style="list-style-type: none"> <li>➤ Mailbox 通讯启动</li> <li>➤ 将 Input 数据更新到 ProcessDataInput</li> <li>➤ 从 ProcessDataOutput 取得 Output 数据并输出</li> </ul>

## 5.7. Process Data 存取

### 5.7.1. ProcessData 运作机制

NexECMRtx 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 EC-Master 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();
```

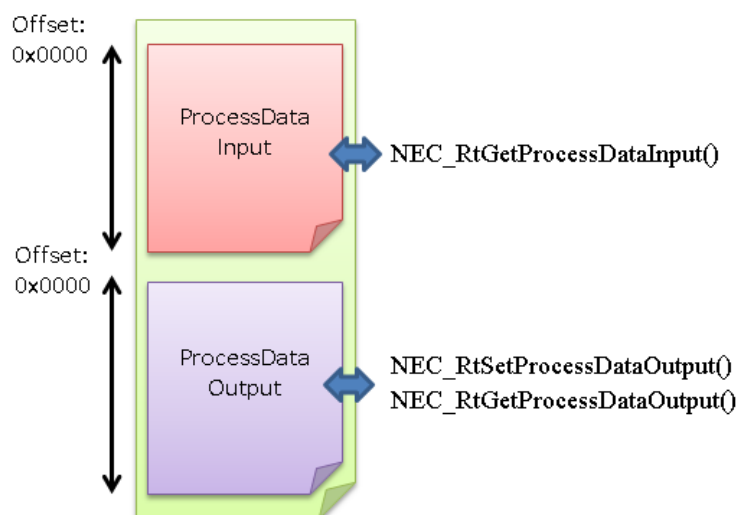
关于 Process Data 的存取时机，建议在 Callback 程序之中来存取，以确保数据全区块的同步完整性，若在 Callback 函数之外存取 ProcessData，数据的同步性只有 1 Byte 为单位。

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 EC-Master	Read Only
ProcessDataOutput	EC-Master 传送至 EC-Slaves	Read/Write

存取所对应的 API 如下图表示:

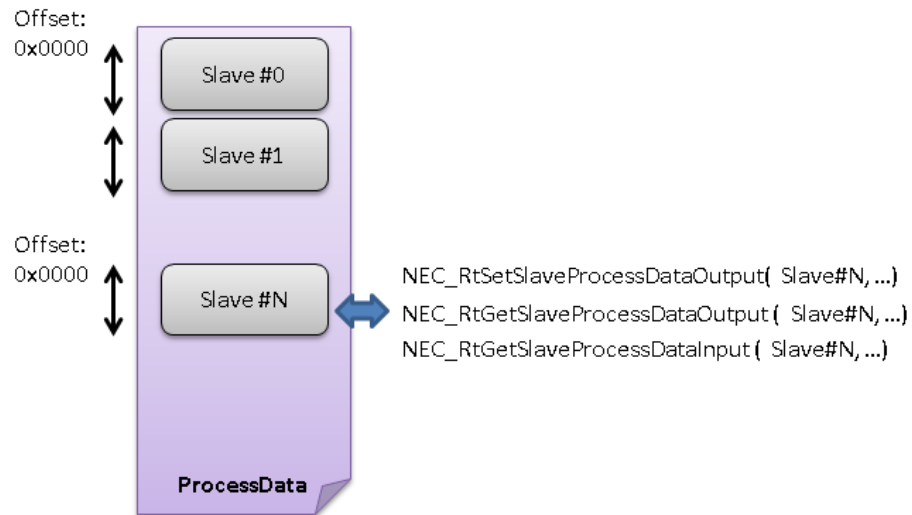


## 5.7.2. ProcessData 数据内容

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 ()`

## 5.8. Mailbox 通讯

### 5.8.1. 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 传输成功表示证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

NexECMRtx 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 6.7 小节说明。

### 5.8.2. CoE -Emergency

当 EC-Slaves 有错误发生时，会透过 CoE 通讯传递 Emergency 讯息至 EC-Master。EC-Master 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 EC-Master 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，EC-Master 接收来自多个 EC-Slaves 的 Emergency 讯息，此时回调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。

NexECMRtx 提供下列函数来完成读取 EC-Master 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 6.7 小节说明。

## 6. NexECMRtx 链接库

### 6.1. RTX API 总览

下表列出 NexECMRtx 函式库 API 的列表，API 定义于 NexECMRtx.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	取得 NexECMRtx 目前版本信息	B
NEC_RtRetVer	回传 NexECMRtx 目前版本信息	B
NEC_RtInitMaster	初始化 NexECMRtx	X
NEC_RtCloseMaster	关闭 NexECMRtx (EC-Master)	X
NEC_RtSetParameter	设置 EC-Master 参数	X
NEC_RtGetParameter	读取 EC-Master 参数	X
NEC_RtRegisterClient	注册 Callback clients 函数	X
NEC_RtUnregisterClient	取消 Callback client 的注册	X
NEC_RtGetProcessDataPtr	取得 EC-Mater 中 ProcessData 内存指针	B
NEC_RtSetProcessDataPtrSource	设置 ProcessData 内存的来源	X
EC-Master 控制相关函式		
NEC_RtStartMaster	启动 EC-Master 通讯	X
NEC_RtStopMaster	停止 EC-Master 周期通讯	X
NEC_RtStopMasterCb	发出 EC-Master 通讯中止要求	C
NEC_RtWaitMasterStop	等待 EC-Master 通讯中止要求并停止通讯	X
NEC_RtGetMasterState	取得 EC-Master 目前的状态	B
NEC_RtSetMasterState	发出 EC-Master 状态变更要求	B
NEC_RtChangeStateToOP	阻塞式变更 EC-Master 状态成为“OP”状态	X
NEC_RtSetMasterStateWait	阻塞式变更 EC-Master 状态	X
NEC_RtGetStateError	读取状态错误代码	B
Process data 存取相关函式		
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
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_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 EC-Master 中 Emergency 数据数量	B
NEC_RtGetEmgData	读取 EC-Master 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

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

## 6.2. 初始化相关函式

### 6.2.1. NEC\_RtGetVersion

取得 NexECMRtx 目前版本信息

**C/C++语法:**

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

**参数:**

U32\_T \*Version: Return the version of NexECMRtx.

**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()





### 6.2.2. NEC\_RtRetVer

回传 NexECMRtx 目前版本信息

**C/C++语法:**

```
U32_T NEC_RtRetVer();
```

**参数:**

<无参数>

**回传值:**

回传NexECMRtx目前版本信息。

**用法:**

没有限制。

**参阅::**

NEC\_RtGetVersion()

### 6.2.3. NEC\_RtInitMaster

初始化 NexECMRtx

**C/C++语法:**

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

**参数:**

U16\_T MasterId: 指定目标 EC-Master 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:**

使用其他 NexECMRtx 函式前，先调用此函数进行函市库内部初始化。

**注意!** 禁止于 Callback 函式中调用此函数

**参阅::**

NEC\_RtCloseMaster()

#### 6.2.4. NEC\_RtCloseMaster

关闭 NexECMRtx (EC-Master)

##### C/C++语法:

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

##### 参数:

U16\_T MasterId: 指定目标 EC-Master 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 NexECMRtx 内部资源。

**注意!** 禁止于 Callback 函数中调用此函数

##### 参阅::

NEC\_RtInitMaste()

## 6.2.5. NEC\_RtSetParameter

## 6.2.6. NEC\_RtGetParameter

这两个函数用于设置和读取 EC-Master 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: 指定目标 EC-Master 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 EC-Master 通讯之前, 设定 EC-Master 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 函数中调用此函数

参数代码	说明	参数值
NEC_PARA_S_ECM_CYCLETIMEUS	EC-Master 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 <i>NEC_LoadNetworkConfig()</i>	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: <b>LINKERR_AUTO:</b> 当 EC-Slave(s) 断线, EC-Master 自动侦测	LINKERR_AUTO (0) LINKERR_MANUAL (1)

	<p>断线的装置。当装置从新联机自动将 EC-Slaves 初始化并设定为“OP”状态</p> <p><b>LINKERR_MANUAL:</b></p> <p>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，不会从新初始化。</p> <p><b>LINKERR_STOP:</b></p> <p>只要一装置断线，EC-Master 将网络中断，并进入 ERROR 状态。</p>	LINKERR_STOP (2)
NEC_PARA_ECM_DC_CYC_TIME_MODE	<p>DC 时间设定模式:</p> <p>0: 根据 Master cycle time (预设)</p> <p>1: 根据 ENI 资料</p>	0~1

参阅:

NEC\_RtGetStateError(); NEC\_RtStartMaster()

### 6.2.7. NEC\_RtRegisterClient

注册 Callback clients 函数

**C/C++语法:**

```
RTN_ERR NEC_RtRegisterClient( U16_T MasterId, TClintParam *ClientParam );
```

**参数:**

U16\_T MasterId: 指定目标 EC-Master 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。

EC-Master 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 EC-Master 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 EC-Master 后当预设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 EC-Master 后当预设错误事件产生后会被呼叫。

U16\_T clientID: 保留 NexECMRtx 内部使用。请勿变更此参数。

(\*) 关于 Callback 函数的使用, 请参考 5.5 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 EC-Master 通讯之前调用此函数。若要从新注册或 EC-Master 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 函数中调用此函数

**参阅::**

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

### 6.2.8. NEC\_RtUnregisterClient

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: 指定目标 EC-Master 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 EC-Master 通讯结束后, 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, 请勿在 EC-Master 通讯过程中取销注册。

**注意!** 禁止于 Callback 函式中调用此函数

#### 参阅::

*NEC\_RtRegisterClient()*; *NEC\_RtWaitMasterStop()*



### 6.2.9. NEC\_RtGetProcessDataPtr

取得 EC-Mater 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: 指定目标 EC-Master 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 slave)内存指针, 设定为 0 忽略

U32\_T \*OutPDSIZEInByte:

回传 process data input (Slaves to master)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 5.5 callback 函式和 5.7 小节 ProcessData 存取。

#### 参阅::

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

### 6.2.10. 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: 指定目标 EC-Master 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:

EC-Master 内定是以内部存储器当作 ProcessData 来源可使用

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, 此函是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 EC-Master 停止通讯后, 用户的 RTX 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(Consistence), 相关说明请参考 5.5 callback 函式和 5.7 小节 ProcessData 存取。

参阅::

NEC\_RtSetSlaveProcessDataOutput (); NEC\_RtGetSlaveProcessDataOutput ();

NEC\_RtGetSlaveProcessDataInput (); NEC\_RtGetProcessDataPtr();

## 6.3. EC-Master 控制相关函式

### 6.3.1. NEC\_RtStartMaster

启动 EC-Master 通讯。

**C/C++语法:**

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

**参数:**

U16\_T MasterId: 指定目标 EC-Master 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 EC-Master 参数后启动通讯，启动成功后，EC-Master 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()*

### 6.3.2. NEC\_RtStopMaster

停止 EC-Master 周期通讯。

#### C/C++语法:

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

#### 参数:

U16\_T MasterId: 指定目标 EC-Master 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 函数中调用此函数

此函数用于停止 EC-Master 周期通讯。在停止通讯前须将 EC-Master 状态切换至“INIT”状态，可调用 *NEC\_RtSetMasterState()*。

若要在 Callback 函数中停止 EC-Master 请参考 *NEC\_RtStopMasterCb()*。

#### 参阅::

*NEC\_RtStartMaster()*; *NEC\_RtStopMasterCb()*

### 6.3.3. NEC\_RtStopMasterCb

发出 EC-Master 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: 指定目标 EC-Master 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()

#### 6.3.4. NEC\_RtWaitMasterStop

等待 EC-Master 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: 指定目标 EC-Master 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 EC-Master 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 RTX 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，RTX 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 EC-Master 讯号。

**注意!** 禁止于 Callback 函数中调用此函数

##### 参阅:

NEC\_RtStopMasterCb()

### 6.3.5. NEC\_RtGetMasterState

### 6.3.6. NEC\_RtSetMasterState

*NEC\_RtGetMasterState()* : 取得 EC-Master 目前的状态(状态机)

*NEC\_RtSetMasterState()* : 发出 EC-Master 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: 指定目标 EC-Master 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 EC-Master 通讯后, 使用 *NEC\_RtSetMasterState()* 函式改变目标 EC-Master 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 EC-Master 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。

另外也可使用阻塞式函数来改变 EC-Master 状态, 请参阅

*NEC\_RtSetMasterStateWait()*, *NEC\_RtChangeStateToOP()*



关于 EC-Master 状态机的说明请参考 5.6 小节 “EtherCAT 状态机” 的说明。

参阅:

`NEC_RtStartMaster(); NEC_RtSetMasterStateWait(), NEC_RtChangeStateToOP()`

### 6.3.7. NEC\_RtChangeStateToOP

阻塞式函数，切换 EC-Master 状态成为” OP” 状态

#### C/C++语法:

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

#### 参数:

U16\_T MasterId: 指定目标 EC-Master 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\_RtGetMasterState()*，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 EC-Master 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 EC-Master 通讯后用来改变目标 EC-Master 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 EC-Master 状态机的说明请参考 5.6 小节 ”EtherCAT 状态机”的说明。

#### 参阅:

*NEC\_RtStartMaster()*; *NEC\_RtSetMasterState ()*; *RtSetMasterStateWait()*,

### 6.3.8. NEC\_RtSetMasterStateWait

阻塞式变更 EC-Master 状态

#### C/C++语法:

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

#### 参数:

U16\_T MasterId: 指定目标 EC-Master 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\_RtGetMasterState()*, 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 EC-Master 状态。用法类似 *NEC\_RtSetMasterState()* 启动 EC-Master 通讯后用来改变目标 EC-Master 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 EC-Master 状态机的说明请参考 5.6 小节 “EtherCAT 状态机”的说明。

#### 参阅:

*NEC\_RtStartMaster(); NEC\_RtSetMasterState (); NEC\_RtChangeStateToOP (),*

### 6.3.9. NEC\_RtGetStateError

读取状态错误代码

#### C/C++语法:

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

#### 参数:

U16\_T MasterId: 指定目标 EC-Master 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 EC-Master 状态异常(如通讯断线), 状态会改变为 *ECM\_STA\_ERROR* 状态, EC-Master 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()

## 6.4. Process data 存取相关函式

### 6.4.1. NEC\_RtSetProcessDataOutput

### 6.4.2. NEC\_RtGetProcessDataOutput

### 6.4.3. NEC\_RtGetProcessDataInput

存取 EC-Master 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: 指定目标 EC-Master 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 5.7 小节 Process Data 存取

#### 参阅:

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

#### 6.4.4. NEC\_RtSetSlaveProcessDataOutput

#### 6.4.5. NEC\_RtGetSlaveProcessDataOutput

#### 6.4.6. NEC\_RtGetSlaveProcessDataInput

存取 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: 指定目标 EC-Master 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 5.7 小节 Process Data 存取

#### 参阅:

NEC\_RtSetProcessDataOutput(); NEC\_RtGetProcessDataOutput();

NEC\_RtGetProcessDataInput()

## 6.5. Callback 函式

### 6.5.1. \*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 EC-Master 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 5.5 小节“Callback 函式”的说明。

**参阅:**

```
NEC_RtRegisterClient();NEC_RtUnregisterClient();
```



## 6.5.2. \*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 EC-Master 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	EC-Master 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();

### 6.5.3. \*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 EC-Master 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()*;

## 6.6. ENI 相关函式

### 6.6.1. NEC\_RtGetSlaveCount

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: 指定目标 EC-Master 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 EC-Master 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>

### 6.6.2. NEC\_RtGetSlaveInformation

读取某个 EC-Slaves 的信息

#### C/C++语法:

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

#### 参数:

U16\_T MasterId: 指定目标 EC-Master 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 EC-Master 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>

### 6.6.3. NEC\_RtGetSlaveState

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: 指定目标 EC-Master 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:

本函数可一次读回多个 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>

## 6.7. CoE 通讯相关函式

### 6.7.1. NEC\_RtStartSDODownload

### 6.7.2. NEC\_RtStartSDOUpload

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: 指定目标 EC-Master 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()



**6.7.3. NEC\_RtSDODownload****6.7.4. NEC\_RtSDOUpload****6.7.5. NEC\_RtSDODownloadEx****6.7.6. 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: 指定目标 EC-Master 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 函数中调用此函数

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()

### 6.7.7. NEC\_RtStartGetODListCount

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

参数:

U16\_T MasterId: 指定目标 EC-Master 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();

### 6.7.8. NEC\_RtStartGetODList

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

参数:

U16\_T MasterId: 指定目标 EC-Master 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()

### 6.7.9. NEC\_RtStartGetObjDesc

读取 EC-Slave 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: 指定目标 EC-Master 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 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: 回传对象数据类型

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()

### 6.7.10. NEC\_RtStartGetEntryDesc

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: 指定目标 EC-Master 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;
} TCoEntryDesc;
```

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()

### 6.7.11. NEC\_RtGetEmgDataCount

读取 EC-Master 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: 指定目标 EC-Master 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 EC-Master 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()

### 6.7.12. NEC\_RtGetEmgData

读取 EC-Master 中 Emergency 数据

C/C++语法:

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

参数:

U16\_T MasterId: 指定目标 EC-Master 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 *pEmgMsgDataArr;
} 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 EC-Master 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()

## 6.8. 存取从站模块硬件信息相关函式

### 6.8.1. NEC\_RtGetConfiguredAddress

取得从站模块“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: 指定目标 EC-Master 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，用户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:

### 6.8.2. NEC\_RtGetAliasAddress

取得从站模块” 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: 指定目标 EC-Master 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 函数中调用此函数

此函数用于取得指定从站模块的 Configured Station Alias 信息。例如, 用户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:

### 6.8.3. NEC\_RtGetSlaveCoeProfileNum

取得从站模块”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: 指定目标 EC-Master 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 );
```

```
    }
```

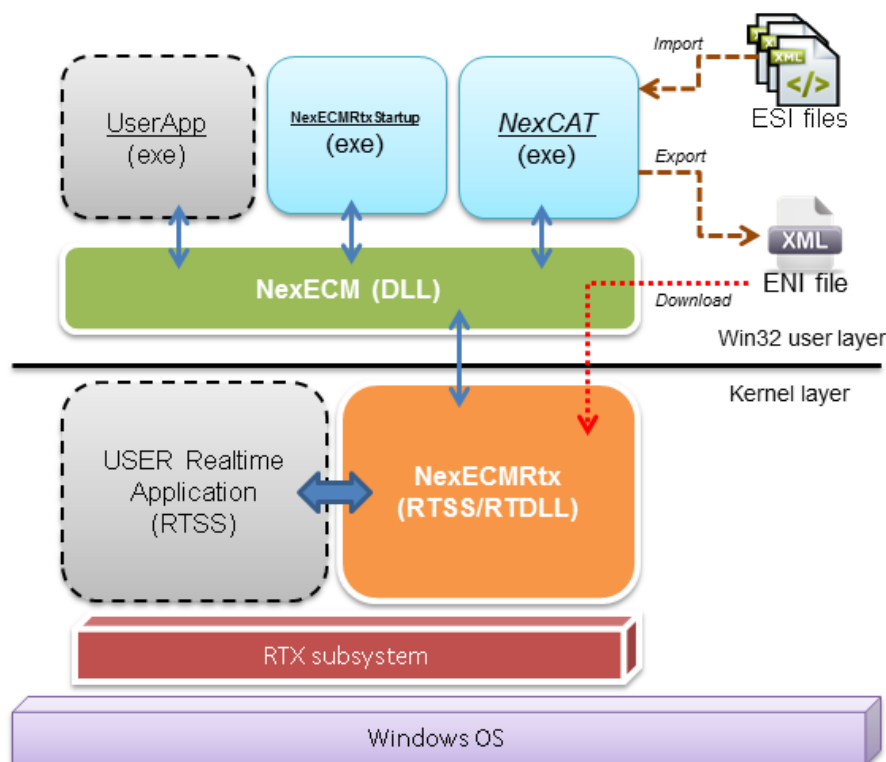
```
}
```

#### 参阅:



## 7. NexECM 链接库 (Win32 APIs)

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 NexECMRtx runtime。透过 NexECM Win32 函数库用户可以很快速的将 Win32 的应用程序和 NexECMRtx RTX 应用程序做整合。



NexECM 函数库主要提供下列功能:

1. 加载 / 移除 RTSS 应用程序至 RTX 系统中。
2. 加载 EtherCAT 网络描述文件(ENI)信息至 EC-Master(NexECMRtx)中
3. ProcessData 存取(透过 NexECMRtx)
4. CoE SDO 存取(透过 NexECMRtx)
5. CiA402 APIs, 请参考 CiA402 APIs 使用手册

## 7.1. 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	
RTX 系统相关控制 API		
NEC_LoadRtxApp	加载RTSS 应用程序	
NEC_KillRtxApp	移除RTSS应用程序	
NexECM 初始化相关函式		
NEC_GetVersion	读取NexECM 函式库 (NexECM.dll) 版本号码	
NEC_StartDriver	初始化NexECM函式库	
NEC_StopDriver	关闭NexECM函式库	
NexECMRtx Runtime 控制相关函式		
NEC_GetRtMasterId	自动侦测NexECMRtx并取得NexECMRtx所属的控制代号 (MasterID)	
NEC_GetRtMasterVersion	读取NexECMRtx 版本号码	
NEC_GetRtMasterProcessId	读取NexECMRtx Process ID (RTX Process ID)	
NEC_ResetEcMaster	重置 EC-Master (NexECMRtx)	
NEC_LoadNetworkConfig	载入ENI档案	
NEC_StartNetwork	启动EC-Master 通讯	
NEC_StartNetworkEx	启动EC-Master 通讯附带Option选项	
NEC_StopNetwork	停止EC-Master周期通讯	
NEC_SetParameter	设置 EC-Master 参数	
NEC_GetParameter	读取 EC-Master 参数	
网络状态存取相关函式		
NEC_GetSlaveCount	读取EC-Slaves 个数	
NEC_GetNetworkState	读取EC-Master当前状态	
NEC_GetSlaveState	读取EC-Slave当前状态	
NEC_GetStateError	读取EC-Master错误状态	
NEC_GetErrorMsg	读取EC-Master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	
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-Slave 单一对象的 Object Description 信息	
NEC_RtStartGetEntryDesc	读取 EC-Slave 的单一对象 Entry Description 信息	
NEC_RtGetEmgDataCount	读取 EC-Master 中 Emergency 数据数量	
NEC_RtGetEmgData	读取 EC-Master 中 Emergency 数据	
Process data 存取相关函数		
NEC_RWProcessImage	存取 EC-Master 的 ProcessData 资料	
NEC_GetProcessImageSize	取得 EC-Master 的 ProcessData 内存长度	
NEC_RWSlaveProcessImage	存取 EC-Slave 的 ProcessData 内存	
存取从站模块硬件信息相关函数		
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++ 数据类型定义于 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

## 7.2. RTX 系统相关控制 API

### 7.2.1. NEC\_LoadRtxApp

加载 RTSS 应用程序

**C/C++语法:**

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

**参数:**

const char\* RtxFileName: RTSS 档案路径 C-style 字符串

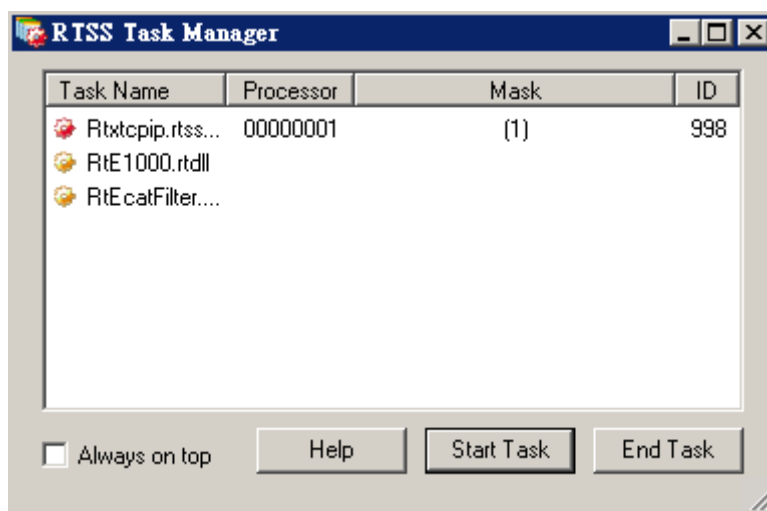
**回传值:**

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:**

可用本函数将 EC-Master runtime (NexECMRtx.rtss) 或任何 RTSS 应用程序加载到 RTX 系统中。可用 RTSS Task Manager (RTX 的工具程序)观察被加载的程序。



本函数调用和本函数 *NEC\_StartDriver()* 无关，可先于 *NEC\_StartDriver()*

**参阅:**

NEC\_GetRtMasterId (); NEC\_KillRtxApp (); NEC\_GetRtMasterPorcessId()

## 7.2.2. NEC\_KillRtxApp

移除 RTSS 应用程序

**C/C++语法:**

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

**参数:**

U32\_T ProcessId: RTSS's Process ID.

**回传值:**

回传错误代码。

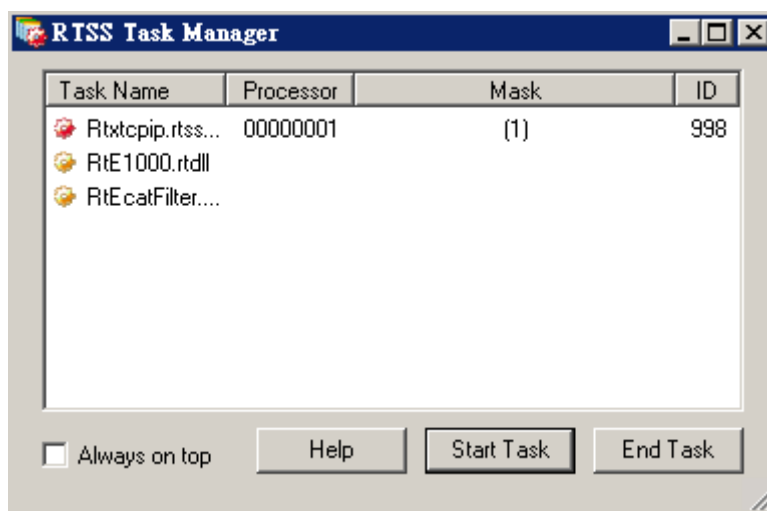
调用函数成功回传“ECERR\_SUCCESS” (0)，反之函数调用失败回传错误代码，错误代码定义于EcErrors.h头文件中。

**用法:**

可用本函数将 EC-Master runtime (NexECMRtx.rtss) 或任何 RTSS 应用程序从 RTX 系统中强制移除。ProcessID 可经由 RTX 工具程序: “Task Manager” (如下图)观察。

亦可用 RTSS Task Manager (RTX 的工具程序)确认 RTSS 的程序是否被移除。

NexECMRtx 的 ProcessID 可以用 NEC\_GetRtMasterPorcessId() 取得。



若使用本函数强制移除相关应用程序，可能造成系统不稳定，须注意使用。容易不稳定的状况可能来自移除应用程序的顺序，例如当应用程序直接存取 NexECMRtx 的内存时将 NexECMRtx 移除。

本函式调用和函数 NEC\_StartDriver()无关，可先于 NEC\_StartDriver()

**参阅:**

NEC\_LoadRtxApp(); NEC\_GetRtMasterPorcessId(); NEC\_GetRtMasterId();

## 7.3. NexECM 初始化相关函式

### 7.3.1. NEC\_GetVersion

读取 NexECM 函式库 (NexECM.dll) 版本号码

**C/C++语法:**

```
U32_T NEC_GetVersion();
```

**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:**

读取 NexECM 函式库 (NexECM.dll) 版本号码

**参阅:**

<无参阅>

### 7.3.2. 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 NexECMRtx 是否加载，因此必须确定 NexECMRtx 是否已经被加载到 RTX 系统中。载入 NexECMRtx 可参考 *NEC\_LoadRtxApp()*

**参阅:**

*NEC\_LoadRtxApp();NEC\_StopDriver();*



### 7.3.3. 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 ()

## 7.4. NexECMRtx Runtime 控制相关函式

### 7.4.1. NEC\_GetRtMasterId

侦测 NexECMRtx 是否已加载并取得 NexECMRtx 所属的控制代号(MasterID)

**C/C++语法:**

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

**参数:**

U16\_T \*pMasterId: 回传 NexECMRtx 的控制代号(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) NexECMRtx之前必须先使用本函数取得控制代号(MasterID)，若本函数还处错误代码(ECERR\_DRIVER\_NOT\_FOUND)可能表示RTX系统中尚未加载NexECMRtx。

**参阅:**

NEC\_LoadRtxApp(); NEC\_KillRtxApp (); NEC\_GetRtMasterPorcessId()

#### 7.4.2. NEC\_GetRtMasterVersion

读取 NexECMRtx 版本号码

##### C/C++语法:

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

##### 参数:

U16\_T MasterId: 指定目标 EC-Master 代号。ID 可由 *NEC\_GetRtMasterId()*取得

U32\_T \*pVersion: NexECMRtx 版本号码

##### 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:

读取 NexECMRtx (RTX EtherCAT master) 版本号码。同 *NEC\_RtGetVersion()*

##### 参阅:

## 7.4.3. NEC\_GetRtMasterPorcessId

读取 NexECMRtx Process ID (RTX Process ID)

**C/C++语法:**

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

**参数:**

U16\_T MasterId: 指定目标 EC-Master 代号。ID 可由 *NEC\_GetRtMasterId()*取得

U32\_T \*pProcessID: 回传 RTX Process ID

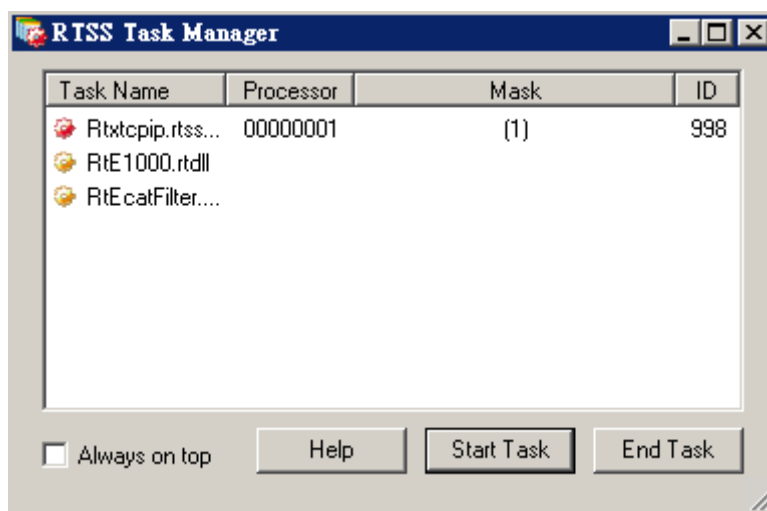
**回传值:**

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:**

RTX ProcessID 可经由工具程序 Task manager 观察。用本函数可取得 NexECMRtx 的 ProcessID. 可用于 *NEC\_KillRtxApp()*。



RTSS Task Manager

**参阅:**

*NEC\_KillRtxApp()*;

#### 7.4.4. NEC\_ResetEcMaster

重置 EC-Master (NexECMRtx)

**C/C++语法:**

```
RTN_ERR NEC_ResetEcMaster( U16_T MasterId );
```

**参数:**

U16\_T MasterId: 指定目标 EC-Master 代号。ID 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 EC-Master (NexECMRtx)。一般使用本函数的时机是在 *NEC\_GetRtMasterId()*取得 MasterID 后，载入 ENI 档案前。使用本函数先将 EC-Master 内部变量进行重置。若先前已经有下载过 ENI 档案，ENI 档案数据也会被清除。

**参阅:**

*NEC\_GetRtMasterId()*; *NEC\_LoadNetworkConfig()*

#### 7.4.5. NEC\_LoadNetworkConfig

载入 ENI 档案

##### 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 EC-Master 代号。ID 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，EC-Master 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:

此还是一般用在 *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 NexCAT 工具程序产生(请参阅第三章)

##### 参阅:

*NEC\_ResetEcMaster()*

#### 7.4.6. NEC\_StartNetwork

启动 EC-Master 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 EC-Master 代号。ID 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 EC-Master 参数后启动通讯, 启动成功后, EC-Master 会建立周期的通讯机制。呼叫 *NEC\_StopNetwork()*停止通讯 EC-Master 通讯。

##### 参阅:

*NEC\_GetRtMasterId()*; *NEC\_SetParameter()*; *NEC\_StopNetwork()*;  
*NEC\_StartNetworkEx()*

### 7.4.7. NEC\_StartNetworkEx

启动 EC-Master 通讯；此 API 同 NEC\_StartNetwork，多了一个设定参数。

#### C/C++语法:

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

#### 参数:

U16\_T MasterId: 指定目标 EC-Master 代号。ID 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，EC-Master 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 EC-Master 参数后启动通讯，启动成功后，EC-Master 会建立周期的通讯机制。呼叫 NEC\_StopNetwork 停止通讯 EC-Master 通讯。

#### 参阅:

NEC\_GetRtMasterId();NEC\_SetParameter();NEC\_StartNetwork();NEC\_StopNetwork()



#### 7.4.8. NEC\_StopNetwork

停止 EC-Master 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 EC-Master 代号。ID 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 EC-Master 周期通讯。停止通讯过程中, EC-Master 会切回 “INIT” 状态。

**参阅:**

NEC\_GetRtMasterId();NEC\_SetParameter();NEC\_StartNetwork();

NEC\_StartNetworkEx()

## 7.4.9. NEC\_SetParameter

## 7.4.10. NEC\_GetParameter

这两个函数用于设置和读取 EC-Master 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 EC-Master 代号。ID 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 EC-Master 通讯之前, 设定 EC-Master 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	EC-Master 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: <b>LINKERR_AUTO:</b> 当 EC-Slave(s) 断线, EC-Master 自动侦测断线的装置。当装置从新联机自动将 EC-Slaves 初始化并设定为“OP”状态 <b>LINKERR_MANUAL:</b> 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)

	<b>LINKERR_STOP:</b> 只要有一装置断线，EC-Master 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();

## 7.5. 网络状态存取相关函式

### 7.5.1. NEC\_GetSlaveCount

读取在线 EC-Slaves 装置数量

**C/C++语法:**

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

参数:

U16\_T MasterId: 指定目标 EC-Master 代号。ID 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 EC-Master 通讯后，使用此函数读取 EC-Slave 的个数。此数量基本上由 ENI 档案内容决定。

参阅:

*NEC\_GetRtMasterId()*

### 7.5.2. NEC\_GetNetworkState

读取网络联机状态

**C/C++语法:**

```
RTN_ERR NEC_GetNetworkState( U16_T MasterId, U16_T *State );
```

参数:

U16\_T MasterId: 指定目标 EC-Master 代号。ID 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
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 EC-Master 通讯之后，轮询网络当前的状态。

参阅:

NEC\_GetRtMasterId();

### 7.5.3. NEC\_GetSlaveState

读取网络联机状态

#### 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 EC-Master 代号。ID 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

U8\_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 EC-Master 通讯之后, 轮询网络上所有 EC-Slaves 的当前状态。

#### 参阅:

NEC\_GetRtMasterId();

#### 7.5.4. NEC\_GetStateError

读取 EC-Master 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 EC-Master 代号。ID 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 EC-Master 通讯之后，EC-Master 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 EC-Master 状态错误代码。

##### 参阅:

NEC\_GetRtMasterId();NEC\_GetErrorMsg()

### 7.5.5. NEC\_GetErrorMsg

读取 EC-Master 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 EC-Master 代号。ID 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 EC-Master 通讯之后，EC-Master 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 EC-Master 状态错误字符串。

#### 参阅:

NEC\_GetRtMasterId();NEC\_GetStateError()



## 7.6. DIO 控制相关函式

### 7.6.1. 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 EC-Master 代号。ID 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 EC-Master 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();

### 7.6.2. 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 EC-Master 代号。ID 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 EC-Master 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();

### 7.6.3. 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 EC-Master 代号。ID 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 EC-Master 通讯之后，读取 EC-Slave Digital input 输出；使用上需注意 SizeByte 与 DiData 长度。

#### 参阅:

NEC\_GetRtMasterId();

## 7.7. CoE 通讯相关函式

### 7.7.1. NEC\_SDODownloadEx

### 7.7.2. NEC\_SDOUploadEx

### 7.7.3. NEC\_SDODownload

### 7.7.4. 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 EC-Master 代号。ID 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();

### 7.7.5. 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 EC-Master 代号。ID 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 index;  
    U8_T subIndex;  
    U8_T ctrlFlag;  
    U8_T *dataPtr;  
    U16_T dataLenByte;  
    U16_T status;  
    I32_T abortCode;  
} TSDO;
```

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();

### 7.7.6. 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 EC-Master 代号。ID 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()

### 7.7.7. NEC\_GetObjDesc

读取 EC-Slave 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 EC-Master 代号。ID 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 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: 回传对象数据类型

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();

### 7.7.8. NEC\_GetEntryDesc

读取 EC-Slave 的单一对象进入 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 EC-Master 代号。ID 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 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();

### 7.7.9. NEC\_GetEmgDataCount

读取 EC-Master 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 EC-Master 代号。ID 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谱上 EC-Slaves 产生的 Emergency 数据数量。

**参阅:**

NEC\_GetODListCount();NEC\_GetODList();NEC\_GetObjDesc();

NEC\_GetEntryDesc();NEC\_GetEmgData();NEC\_GetRtMasterId();

### 7.7.10. NEC\_GetEmgData

读取 EC-Master 中 Emergency 讯息器中讯息

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

参数:

U16\_T MasterId: 指定目标 EC-Master 代号。ID 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 *pEmgMsgDataArr;
} 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 EC-Master 中的 Emergency 数据。

参阅:

NEC\_GetODListCount(); NEC\_GetODList(); NEC\_GetObjDesc();  
NEC\_GetEntryDesc(); NEC\_GetEmgData(); NEC\_GetRtMasterId();



## 7.8. Process data 存取相关函式

### 7.8.1. NEC\_RWProcessImage

存取 EC-Master 的 ProcessData 资料。

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 EC-Master 代号。ID 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: EC-Master 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 EC-Master (NexECMRtx) 内部的 Process Image (或称 ProcessData) 。ProcessData 原理及操作方式请参考 5.7 小节 Process Data 存取。

参阅:

NEC\_GetRtMasterId();NEC\_GetProcessImageSize();NEC\_RWSlaveProcessImage ()

## 7.8.2. NEC\_GetProcessImageSize

取得 EC-Master 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 EC-Master 代号。ID 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 记忆大小指针

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 EC-Master 中 ProcessImage 内存的大小，ProcessData 原理及操作方式请参考 5.7 小节 Process Data 存取。

参阅:

NEC\_GetRtMasterId();NEC\_RWProcessImage ();NEC\_RWSlaveProcessImage ()

### 7.8.3. NEC\_RWSlaveProcessImage

存取 EC-Slave 的 ProcessData 内存。

C/C++语法:

```
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 EC-Master 代号。ID 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 内存之中, ProcessData  
原理及操作方式请参考 5.7 小节 Process Data 存取。

参阅:

NEC\_GetRtMasterId(); NEC\_RWProcessImage (); NEC\_GetProcessImageSize ()

## 7.9. 存取从站模块硬件信息相关函式

### 7.9.1. 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: 指定目标 EC-Master 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，用户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:

### 7.9.2. 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: 指定目标 EC-Master 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, 用户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:

### 7.9.3. 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: 指定目标 EC-Master 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: