acontis technologies GmbH

SOFTWARE

EC-Master

Feature Pack RAS

Version 3.2

Edition: September 23, 2025

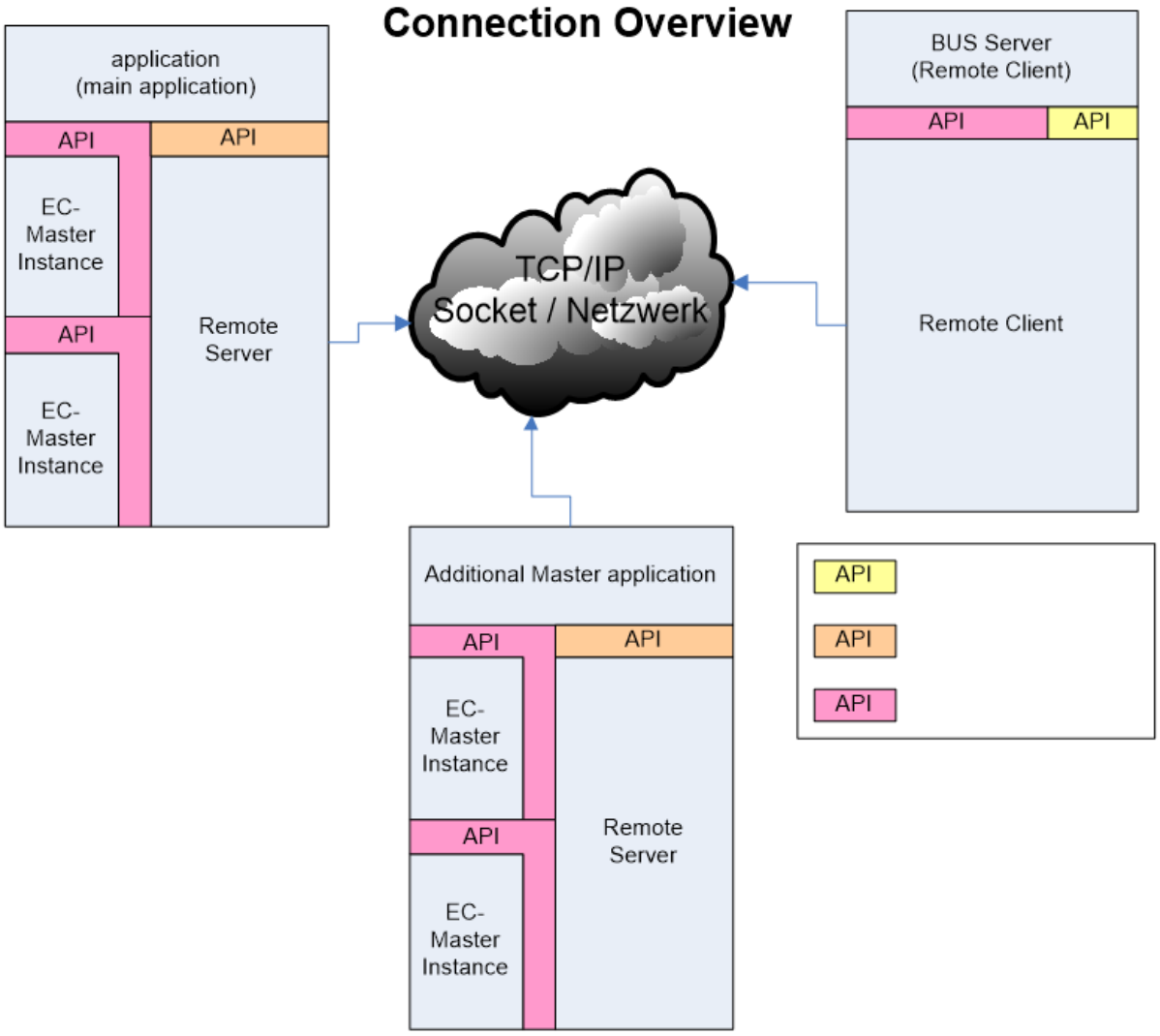
Contents

1 Introduction

In a Linux system, when a secondary process, such as an OPC Server, needs to access EtherCAT® bus data or perform operations on the EtherCAT® stack, the Remote API provides an interface to facilitate this access. Since two applications in Linux cannot directly access each other's memory, the Remote API utilizes a TCP/IP connection. This approach not only enables inter-process communication within the same device but also allows access to the Remote Interface from other host systems, such as Windows or similar platforms.

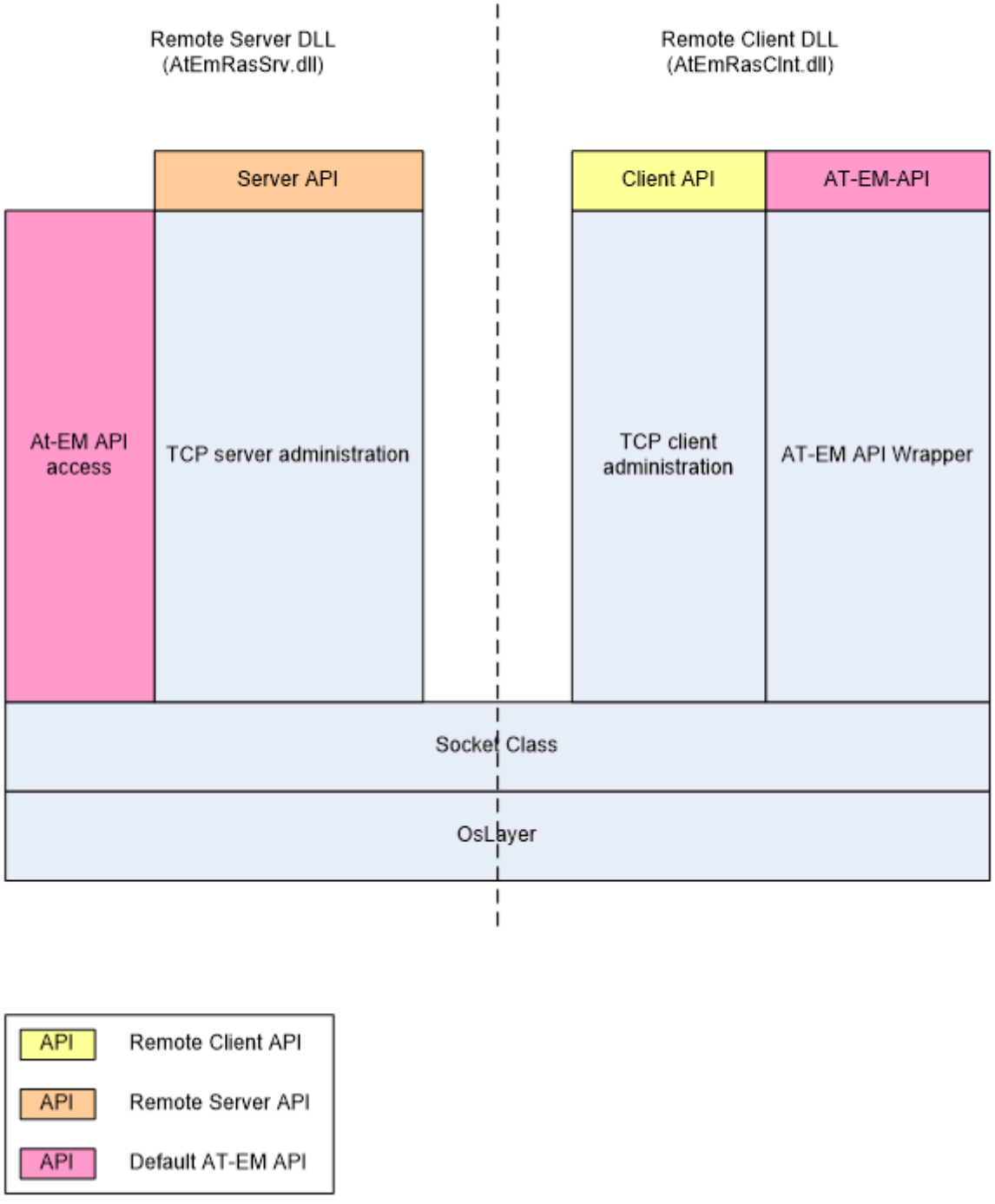
1.1 Remote API Structure

The Remote API works based on TCP/IP sockets, which is almost completely transparent to the calling application.



All a remote application has to take care for is, to initialize the remote API DLL which contains the abstraction of the connection to the Remote Server. After initializing the connection all calls (which are supported remotely) may be used as usual with a "local" master stack. Of course the master stack itself has to run with the additional Remote API Server library which has to be set - up to accept remote clients.

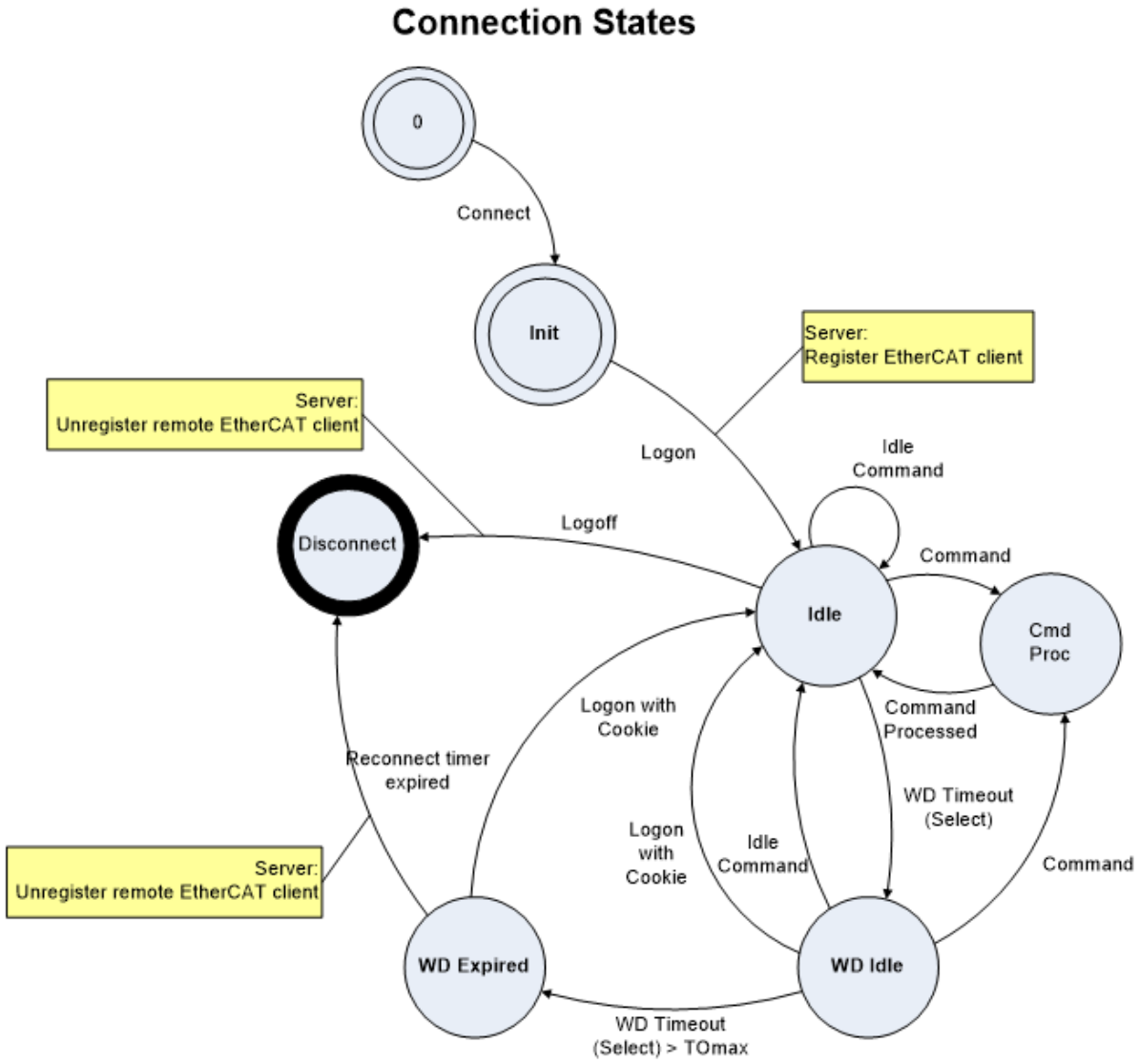
Code Module


Remote Server DLL
(AtEmRasSrv.dll)
Remote Client DLL
(AtEmRasClnt.dll)
Server API
Client API
AT-EM-API
At-EM API access
TCP server administration
TCP client administration
AT-EM API Wrapper
Socket Class
OsLayer
API
Remote Client API
API
Remote Server API
API
Default AT-EM API

1.2 Connection states

In some cases, it is necessary to take care about the internal structure of the Remote API connection, while using a Remote connection. The Remote API library supports a reconnect to the Master Stack in case of a temporary disconnection (e.g. line break). This recoverage of a connection may take place within a well-defined time if it does not, the connection is established newly and an error is notified which has to be taken in account by the fact that the Registered Client has to be Re-Registered and all used Mailbox objects have to be created again after such a reconnect attempt. This is necessary due to the fact, the Remote API Server tries to keep the Master Stack free from unused Memory to provide the highest possible availability and a minimum influence on the Real-Time Application used with the Master Stack (e.g. the PLC Runtime system).

The different states used within the connection's life are shown in below figure. This illustration is only shown to give a clue about the things happening within the Remote API "Layer". All the programmer of a remote application has to take care for is, if the reconnect to the Remote API Server fails because the reconnection timeout has expired, all volatile objects described before have to be re - created.



2 Software Integration - Server site (Remote API Server)

The Remote API Server is included to the master stack using application by following steps:

1. Link the Remote Server API Lib to the Project
2. Make the Remote API Server DLL available to the Runtime environment of your application.
3. Include the necessary curve up and shutdown calls to your master application
4. Compile
5. Run

2.1 Pseudo Example

A master application which includes remote API hosting needs to call following steps:

```
#include "EcRasServer.h"
.
.
emRasSrvStart(…);    /* initialize Remote API Server Module, which starts the␣
↪connection
* acceptor implicitly*/
.
.
/* EC-Master API remote access provided */
.
.
emRasSrvStop(…);     /* de-initialize Remote API Server Module, closes all␣
↪connections */
```

For closer details find a Remote API Server example project <AtemDemoServer> with your installed Examples.

2.2 Remote API Server integration example (RTOS32 and RTOS32Win)

This section shall help you to integrate the Remote API Server into your RTOS32 and RTOS32Win application. We demonstrate step by step the integration using the example of the <AtemDemo> project. It will be assumed that EC-Master for On Time RTOS32 is installed and you can execute the demo application on you target system. The following steps are necessary to integrate the Remote API Server.

1. Open your <AtemDemo> project and add the files `NetRTOS32Init.cpp` and `NetRTOS32Init.h` to your project. These files are located in: `(%Programfiles%)\EC-Master-RTOS-32\SDK\INC\RTOS-32`
2. Add the following libraries into you project settings: EcMasterRasServer.lib; rtip.lib. For RTOS32Win please add also netvmf.lib to use the RTOS32Win shared memory network interface for IP communication.
3. Uncomment the define #define ECMASTERRAS_SERVER in ATEMDemo.h
4. Adjust the added NetRTOS32Init.h

- For RTOS32 please adjust TargetIP, NetMask, DefaultGateway and DNSServer. Furthermore set the DEVICE_ID to one of the supported network adapters.
- For RTOS32Win you can use the VMF-network interface. In this case netvmf.lib should be added to your project and the #define DEVICE_ID should be set to RTVMF_DEVICE.

- Compile and run the demo

2.3 Additional API description

Following calls are necessary to initialize, de-initialize and observe the Remote API Server functionality.

2.3.1 emRasSrvGetVersion

EC_T_DWORD EC_NAMESPACE::**emRasSrvGetVersion**(EC_T_VOID)
Get Version of Remote API Server Software.

Returns
EC_T_DWORD containing the Version description of the Remote API Server in Format: MMmmssbb: MM Major version byte, mm Minor version byte, ss Servicepack nr byte, bb Build number

2.3.2 emRasSrvStart

EC_T_DWORD EC_NAMESPACE::**emRasSrvStart**(
ECMASTERRAS_T_SRVPARMS *pParms,
EC_T_PVOID *ppHandle
)
Initializes and start remote API Server Instance.

The Remote API Server will be initialized and started by calling this function.

Parameters

- **pParms** – [in] Server start-up parameters
- **ppHandle** – [out] Handle to opened instance, used for ctrl access

Returns
EC_E_NOERROR or error code

struct **ECMASTERRAS_T_SRVPARMS**
RAS Server init parameters.

Public Members

EC_T_DWORD **dwSignature**
[in] Set to ECMASTERRASSERVER_SIGNATURE

EC_T_DWORD **dwSize**
[in] Set to sizeof(ECMASTERRAS_T_SRVPARMS)

EC_T_LOG_PARMS **LogParms**
[in] Logging parameters

EC_T_IPADDR **oAddr**
[in] Remote Access Server (RAS) listen IP address

EC_T_WORD **wPort**
[in] Remote Access Server (RAS) listen port

EC_T_WORD **wMaxClientCnt**
[in] Max. clients in parallel (0: unlimited)

EC_T_DWORD **dwCycleTime**
[in] Cycle Time of RAS Network access (acceptor, worker)

EC_T_DWORD **dwCommunicationTimeout**
[in] timeout before automatically closing connection

EC_T_CPUSET **oAcceptorThreadCpuAffinityMask**
[in] Acceptor Thread CPU affinity mask

EC_T_DWORD **dwAcceptorThreadPrio**
[in] Acceptor Thread Priority

EC_T_DWORD **dwAcceptorThreadStackSize**
[in] Acceptor Thread Stack Size

EC_T_CPUSET **oClientWorkerThreadCpuAffinityMask**
[in] Client Worker Thread CPU affinity mask

EC_T_DWORD **dwClientWorkerThreadPrio**
[in] Client Worker Thread Priority

EC_T_DWORD **dwClientWorkerThreadStackSize**
[in] Client Worker Thread Stack Size

EC_T_DWORD **dwMaxQueuedNotificationCnt**
[in] Amount of concurrently queue able Notifications

EC_T_DWORD **dwMaxParallelMbxTferCnt**
[in] Amount of concurrent active mailbox transfers

EC_PF_NOTIFY **pfnRasNotify**
[in] Function pointer called to notify error and status information generated by Remote API Layer

EC_T_VOID ***pvRasNotifyCtxt**
[in] Notification context returned while calling pfNotification

EC_T_DWORD **dwCycErrInterval**
[in] Interval which allows cyclic Notifications

EC_T_DWORD **dwMaxQueuedNotificationSize**
[in] Size of concurrent active mailbox transfers

EC_PF_CHECK_TOKEN **pfCheckToken**
[in] Function pointer called to check token

EC_T_VOID ***pvCheckTokenContext**
[in] Check token context

2.3.3 emRasSrvStop

EC_T_DWORD EC_NAMESPACE::**emRasSrvStop** (
EC_T_PVOID pvHandle,
EC_T_DWORD dwTimeout
)

Stop and de-initialize remote API Server Instance.

Parameters

- **pvHandle** – [in] Handle to previously started Server
- **dwTimeout** – [in] Timeout [ms] used to shut down all spawned threads, it's multiplied internally by the amount of threads spawned.

Returns

EC_E_NOERROR or error code

2.3.4 emrasNotify - xxx

struct **EC_T_NOTIFYPARMS**

Detailed information about the according notification (EC_NOTIFY_…)

Public Members

EC_T_VOID ***pCallerData**

[in] Parameter arbitrarily defined by the application at client registration with the EtherCAT stack

EC_T_BYTE ***pbyInBuf**

[in] Notification input parameters

EC_T_DWORD **dwInBufSize**

[in] Size of the input parameter buffer

EC_T_BYTE ***pbyOutBuf**

[out] Notification output (result)

EC_T_DWORD **dwOutBufSize**

[in] Size of the output buffer

EC_T_DWORD ***pdwNumOutData**

[out] Actually used buffer size of the output buffer

2.3.5 emrasNotify - ECMASTERRAS_NOTIFY_CONNECTION

Notification about a change in the Remote API's state.

emrasNotify - ECMASTERRAS_NOTIFY_CONNECTION

Parameter

- `pbyInBuf`: [in] Pointer to data of type ECMASTERRAS_T_CONNOTIFYDESC.
- `dwInBufSize`: [in] sizeof(ECMASTERRAS_T_CONNOTIFYDESC)

- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

Data structure containing the new Remote API state and the cause of state change.

struct **ECMASTERRAS_T_CONNOTIFYDESC**

Public Members

EC_T_DWORD **dwCause**
[in] Cause of state connection state change

EC_T_DWORD **dwCookie**
[in] Unique identification cookie of connection instance.

2.3.6 emrasNotify - ECMASTERRAS_NOTIFY_REGISTER

Notification about a connected application registered a client to the master stack.

emrasNotify - ECMASTERRAS_NOTIFY_REGISTER

Parameter

- `pbyInBuf`: [in] Pointer to data of type ECMASTERRAS_T_REGNOTIFYDESC.
- `dwInBufSize`: [in] sizeof(ECMASTERRAS_T_REGNOTIFYDESC)
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

struct **ECMASTERRAS_T_REGNOTIFYDESC**

Public Members

EC_T_DWORD **dwCookie**
[in] Unique identification cookie of connection instance

EC_T_DWORD **dwResult**
[in] Result of registration request

EC_T_DWORD **dwInstanceId**
[in] Master Instance client registered to

EC_T_DWORD **dwClientId**
[in] Client ID of registered client

2.3.7 emrasNotify - ECMASTERRAS_NOTIFY_UNREGISTER

Notification about a connected application un-registered a client from the master stack.

emrasNotify - ECMASTERRAS_NOTIFY_UNREGISTER

Parameter

- `pbyInBuf`: [in] Pointer to data of type ECMASTERRAS_T_REGNOTIFYDESC.
- `dwInBufSize`: [in] sizeof(ECMASTERRAS_T_REGNOTIFYDESC)
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

struct *ECMASTERRAS_T_REGNOTIFYDESC*

2.3.8 emrasNotify - ECMASTERRAS_NOTIFY_MARSHALERROR

Notification about an error during marshaling in Remote API Server connection layer.

emrasNotify - ECMASTERRAS_NOTIFY_MARSHALERROR

Parameter

- `pbyInBuf`: [in] Pointer to data of type ECMASTERRAS_T_MARSHALERRORDESC.
- `dwInBufSize`: [in] sizeof(ECMASTERRAS_T_MARSHALERRORDESC)
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

struct **ECMASTERRAS_T_MARSHALERRORDESC**

Public Members

EC_T_DWORD **dwCookie**
: [in] Unique identification cookie of connection instance

EC_T_DWORD **dwCause**
: [in] Cause of the command marshalling error

EC_T_DWORD **dwLenStatCmd**
: [in] Length faulty command

EC_T_DWORD **dwCommandCode**
: [in] Command code of faulty command

2.3.9 emrasNotify - ECMASTERRAS_NOTIFY_ACKERROR

Notification about an error during creation of ACK / NACK packet.

emrasNotify - ECMASTERRAS_NOTIFY_ACKERROR

Parameter

- `pbyInBuf`: [in] Pointer to EC_T_DWORD containing error code.
- `dwInBufSize`: [in] sizeof(EC_T_DWORD)
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

2.3.10 emrasNotify - ECMASTERRAS_NOTIFY_NONOTIFYMEMORY

Notification raised, when no empty buffers for notifications are available in pre-allocated notification store. This points to a configuration error.

emrasNotify - ECMASTERRAS_NOTIFY_NONOTIFYMEMORY

Parameter

- `pbyInBuf`: [in] Pointer to EC_T_DWORD containing unique identification cookie of connection instance.
- `dwInBufSize`: [in] sizeof(EC_T_DWORD)
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

2.3.11 emrasNotify - ECMASTERRAS_NOTIFY_STDNOTIFYMEMORYSMALL

Notification raised, when buffer size for standard notifications available in pre-allocated notification store are too small to carry a specific notification. This points to a configuration error.

emrasNotify - ECMASTERRAS_NOTIFY_STDNOTIFYMEMORYSMALL

Parameter

- `pbyInBuf`: [in] Pointer to EC_T_DWORD containing unique identification cookie of connection instance.
- `dwInBufSize`: [in] sizeof(EC_T_DWORD)
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

2.3.12 emrasNotify - ECMASTERRAS_NOTIFY_MBXNOTIFYMEMORYSMALL

Notification raised, when buffer size for Mailbox notifications available in pre-allocated notification store are too small to carry a specific notification. This points to a configuration error.

emrasNotify - ECMASTERRAS_NOTIFY_MBXNOTIFYMEMORYSMALL

Parameter

- `pbyInBuf`: [in] Pointer to EC_T_DWORD containing unique identification cookie of connection instance.
- `dwInBufSize`: [in] sizeof(EC_T_DWORD)
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

This is a serious error. If this error is raised, Mailbox Transfer objects may have been become out of sync and therefore no more valid usable. Mailbox notifications should be dimensioned correctly see `emRasSrvStart()`.

2.4 Access control

The access control is used in order to restrict RAS client in calling API functions. The RAS server set actual access level according to application logic. The API calls with required access level lower than actual RAS server access level are blocked for execution; the RAS client will be notified about it. There are following access levels (in order of access rights lowering):

- "Full access", all API functions may be executed by client. This is the default level after initialization of access control or if the RAS server did not initialize the access control subsystem as well.
- "Read/write access", lower assess level as "full access". Recommended for modifying API calls.
- "Read only access", lower access level as "read/write access", all modifying API calls are blocked.
- "Block all", all API calls are forbidden for execution.

In order to configure the access control subsystem `emRasSrvConfigAccessLevel()` is used, to active/deactivate the access control call `emRasSrvSetAccessControl()`, to alter the access level required for each API call separately `emRasSrvModifyCallAccessLevel()` is used.

If the required access level of the current API call is lower than actual RAS access level an error code `EMRAS_E_ACCESSLESS` will be returned.

2.4.1 Configuration

Default:
It is possible to omit the configuration data (using *EC_NULL*) in order to apply the default configuration.

Opt-in:
The application can define for each API the lowest value of the global access level that includes the given API. All non-configured APIs are excluded below global access level "Full access".

Opt-out:
The application can define for each API to be completely excluded, even at global access level "Full access".

Mixing modes:
"Opt-Out" (access level "Excluded") and "Opt-in" (non-configured APIs) can be combined. "Opt-Out" is checked before "Opt-in".

Because the first matching configuration data entry specifies the access level of the corresponding API, there might be irrelevant configuration data entries that the RAS server does not detect. This is by intention as it enables the application to e.g. include *RASSPOCCFGINITDEFAULT* at the end of the configuration in order to minimally modify the default configuration. Entries that are more concrete must be given before less concrete entries, see PARAMETER_IGNORE below.

"Opt-Out" can be combined with *RASSPOCCFGINITDEFAULT* in order to support global access levels below "Full access" in conjunction with completely blocking discreet APIs.

2.4.2 emRasSrvConfigAccessLevel

EC_T_DWORD `EC_NAMESPACE::emRasSrvConfigAccessLevel` (
EC_T_PVOID pvHandle,
ECMASTERRAS_T_SPOCCFG *pCfgData,
EC_T_DWORD dwCfgDataCnt
)

Configures and activates the access control subsystem and sets the global control level to full access.

pCfgData optionally defines the access control configuration in an array with each entry of type ATEMRAS_T_SPOCCFG, see chapter "Configuration". The parameter dwCfgDataCnt specifies how many configuration data entries are provided at pCfgData. The access control subsystem creates its own copy of the input configuration structure, so the buffer at pCfgData can be destroyed after this call.

Parameters

- **`pvHandle`** – [in] Handle to previously started Server
- **`pCfgData`** – [in] Pointer to Configuration data or EC_NULL for default configuration
- **`dwCfgDataCnt`** – [in] Amount of entries in array pointed by pConfigData

Returns
EC_E_NOERROR or error code

The access control subsystem will be initialized by calling this function. The access control configuration is defined in an array with each entry of type *ECMASTERRAS_T_SPOCCFG*. The parameter dwLen specifies the overall length of configuration data pointed by pbyData. In order to use default configuration the parameter pbyData has to be set to EC_NULL, in this case the parameter dwLen will be ignored.

The access control subsystem creates its own copy of the input configuration structure, so the pbyData can be destroyed after this call. After initialization this configuration can be altered with `emRasSrvCallAccessLevel()`. After initialization the access control is active and control level is "full access", to change use `emRasSrvSetAccessControl()` and `emRasSrvSetAccessLevel()` respectively.

struct **`ECMASTERRAS_T_SPOCCFG`**

Public Members

EC_T_DWORD **`dwAccessLevel`**
[in] see ECMASTERRAS_ACCESS_LEVEL_…, e.g. ECMASTERRAS_ACCESS_LEVEL_READWRITE

EC_T_DWORD **`dwOrdinal`**
[in] see ord_…, e.g. ord_emSetMasterState

EC_T_DWORD **`dwIndex`**
[in] set to PARAMETER_IGNORE, if not needed

EC_T_DWORD **dwSubIndex**
[in] set to PARAMETER_IGNORE, if not needed

enum **ECMASTERRAS_T_ORDINAL**
Values:

enumerator **ord_emInitMaster**

enumerator **ord_emDeinitMaster**

enumerator **ord_emStart**

enumerator **ord_emStop**

enumerator **ord_emIoControl**

enumerator **ord_emGetSlaveId**

enumerator **ord_emMbxTferCreate**

enumerator **ord_emMbxTferDelete**

enumerator **ord_emCoeSdoDownloadReq**

enumerator **ord_emCoeSdoUploadReq**

enumerator **ord_emCoeGetODListReq**

enumerator **ord_emCoeGetObjectDescReq**

enumerator **ord_emCoeGetEntryDescReq**

enumerator **ord_emGetSlaveProp**

enumerator **ord_emGetSlaveState**

enumerator **ord_emSetSlaveState**

enumerator **ord_emTferSingleRawCmd**

enumerator **ord_emGetSlaveIdAtPosition**

enumerator **ord_emGetNumConfiguredSlaves**

enumerator **ord_emConfigureNetwork**

enumerator **ord_emSetMasterState**

enumerator **ord_emQueueRawCmd**

enumerator **ord_emCoeRxPdoTfer**

enumerator **ord_emExecJob**

enumerator **ord_emGetProcessData**

enumerator **ord_emSetProcessData**

enumerator **ord_emGetMasterState**

enumerator **ord_emFoeFileUpload**

enumerator **ord_emFoeFileDownload**

enumerator **ord_emFoeUpoadReq**

enumerator **ord_emFoeDownloadReq**

enumerator **ord_emCoeSdoDownload**

enumerator **ord_emCoeSdoUpload**

enumerator **ord_emGetNumConnectedSlaves**

enumerator **ord_emResetSlaveController**

enumerator **ord_emGetSlaveInfo**

enumerator **ord_emIsSlavePresent**

enumerator **ord_emAoeWriteReq**

enumerator **ord_emAoeReadReq**

enumerator **ord_emAoeWrite**

enumerator **ord_emAoeRead**

enumerator **ord_emAoeGetSlaveNetId**

enumerator **ord_emGetFixedAddr**

enumerator **ord_emGetSlaveProcVarInfoNumOf**

enumerator **ord_emGetSlaveProcVarInfo**

enumerator **ord_emFindProcVarByName**

enumerator **ord_emGetProcessDataBits**

enumerator **ord_emSetProcessDataBits**

enumerator **ord_emReloadSlaveEEPRom**

enumerator **ord_emReadSlaveEEPRom**

enumerator **ord_emWriteSlaveEEPRom**

enumerator **ord_emAssignSlaveEEPRom**

enumerator **ord_emSoeRead**

enumerator **ord_emSoeWrite**

enumerator **ord_emSoeAbortProcCmd**

enumerator **ord_emGetNumConnectedSlavesMain**

enumerator **ord_emGetNumConnectedSlavesRed**

enumerator **ord_emNotifyApp**

enumerator **ord_emAoeReadWriteReq**

enumerator **ord_emAoeReadWrite**

enumerator **ord_emGetCfgSlaveInfo**

enumerator **ord_emGetBusSlaveInfo**

enumerator **ord_emReadSlaveIdentification**

enumerator **ord_emSetSlaveDisabled**

enumerator **ord_emSetSlaveDisconnected**

enumerator **ord_emRescueScan**

enumerator **ord_emGetMasterInfo**

enumerator **ord_emConfigExtend**

enumerator **ord_emAoeWriteControl**

enumerator **ord_emSetSlavesDisabled**

enumerator **ord_emSetSlavesDisconnected**

enumerator **ord_emSetMbxProtocolsSerialize**

enumerator **ord_emBadConnectionsDetect**

enumerator **ord_emIsConfigured**

enumerator **ord_emPerfMeasReset**

enumerator **ord_emPerfMeasGetRaw**

enumerator **ord_emPerfMeasGetInfo**

enumerator **ord_emPerfMeasGetNumOf**

enumerator **ord_emSelfTestScan**

enumerator **ord_emScanBus**

enumerator **ord_emGetMemoryUsage**

enumerator **ord_emGetCfgSlaveSmInfo**

enumerator **ord_esConnectPorts**

enumerator **ord_esDisconnectPort**

enumerator **ord_esPowerSlave**

enumerator **ord_esSetErrorAtSlavePort**

enumerator **ord_esSetErrorGenerationAtSlavePort**

enumerator **ord_esResetErrorGenerationAtSlavePorts**

enumerator **ord_esSetLinkDownAtSlavePort**

enumerator **ord_esSetLinkDownGenerationAtSlavePort**

enumerator **ord_esResetLinkDownGenerationAtSlavePorts**

enumerator **ord_emGetMasterStateEx**

enumerator **ord_esSendSlaveCoeEmergency**

enumerator **ord_esVoeSend**

enumerator **ord_esGetSimSlaveInfo**

enumerator **ord_esSetSimSlaveState**

enumerator **ord_emSetMasterStateReq**

enumerator **ord_emSetSlaveStateReq**

2.4.3 emRasSrvSetAccessControl

EC_T_DWORD EC_NAMESPACE::**emRasSrvSetAccessControl**(
EC_T_PVOID pvHandle,
EC_T_BOOL bActive
)

Activates or deactivates the access control subsystem.

In case access control subsystem is deactivated, the current access level will be switched to full access.

Parameters

- **pvHandle** – [in] Handle to previously started Server
- **bActive** – [in] New state of access control, EC_TRUE = access control active, EC_FALSE = access control is not active

Returns
EC_E_NOERROR or error code

2.4.4 emRasSrvSetAccessLevel

EC_T_DWORD EC_NAMESPACE::**emRasSrvSetAccessLevel** (
EC_T_PVOID pvHandle,
EC_T_DWORD dwAccessLevel
)

Sets the current global access level.

A configuration entry to modify has to exist, otherwise EMRAS_E_ACCESS_NOT_FOUND error code will be returned.

Parameters

- **pvHandle** – [in] Handle to previously started Server
- **dwAccessLevel** – [in] New Access level

Returns
EC_E_NOERROR or error code

2.4.5 emRasSrvGetAccessLevel

EC_T_DWORD EC_NAMESPACE::**emRasSrvGetAccessLevel** (
EC_T_PVOID pvHandle,
EC_T_DWORD *pdwAccessLevel
)

Returns the current access level.

The memory pointed by pdwAccessLevel has to be allocated prior.

Parameters

- **pvHandle** – [in] Handle to previously started Server
- **pdwAccessLevel** – [out] Pointer to a buffer storing actual access level, see emRasSrvSetAccessLevel

Returns
EC_E_NOERROR or error code

2.4.6 emRasSrvSetCallAccessLevel

EC_T_DWORD EC_NAMESPACE::**emRasSrvSetCallAccessLevel** (
EC_T_PVOID pvHandle,
EC_T_DWORD dwOrdinal,
EC_T_DWORD dwIndex,
EC_T_DWORD dwSubIndex,
EC_T_DWORD dwAccessLevel
)

Modifies the required access level for an API call.

A configuration entry to modify has to exist, otherwise EMRAS_E_ACCESS_NOT_FOUND error code will be returned.

Parameters

- **pvHandle** – [in] Handle to previously started Server
- **dwOrdinal** – [in] API call ID, see ord_…
- **dwIndex** – [in] extra parameter, used for distinguishing different configuration entries of the same API call. Should be set to PARAMETER_IGNORE, if not needed
- **dwSubIndex** – [in] extra parameter, used for distinguishing different configuration entries of the same API call and the same index. Should be set to PARAMETER_IGNORE, if not needed
- **dwAccessLevel** – [in] New access level required for this API call, see ATEMRAS_ACCESS_LEVEL_…

Returns

EC_E_NOERROR or error code

3 Software Integration - Remote site (Remote API Client)

The Remote API Client is included to the master stack using application by following steps:

1. Link the Remote Client API lib to the Project
2. Make the Remote API Client DLL available to the Runtime environment of your application.
3. Include the necessary connect and disconnect calls to your client application.
4. Compile
5. Run

3.1 Example

Here is an example for the Remote API Client:

First, a connection has to be added:

```
/* Connect to Remote API Server */
ECMASTERRAS_T_CLNTCONDESC oRasClientConDesc;
OsMemset(&oRasClientConDesc, 0, sizeof(ECMASTERRAS_T_CLNTCONDESC));

oRasClientConDesc.oAddr.dwAddr = OsInetAddr("127.0.0.1");
oRasClientConDesc.wPort = ECMASTERRAS_DEFAULT_PORT;

oRasClientConDesc.dwCycleTime = ECMASTERRAS_CYCLE_TIME;
oRasClientConDesc.dwWDTOLimit = (ECMASTERRAS_MAX_WATCHDOG_TIMEOUT / ECMASTERRAS_
↪CYCLE_TIME);
oRasClientConDesc.dwRecvPrio = REMOTE_RECV_THREAD_PRIO;

oRasClientConDesc.dwPktAdminSize = 20;
oRasClientConDesc.dwWatchDog = 500;

dwRes = emRasClntAddConnection(&oRasClientConDesc, &dwRasClientId);
if (dwRes != EC_E_NOERROR)
{
    dwRetVal = dwRes;
    goto Exit;
}
dwRasClientId = dwRasClientId | dwMasterId;
```

Then, optionally the master state can be requested:

```
/* Get master state over RAS */
EC_T_STATE eMasterState = emGetMasterState(dwRasClientId);
EcLogMsg(EC_LOG_LEVEL_INFO, (pEcLogContext, EC_LOG_LEVEL_INFO, "Master state is %s\
↪n", ecatStateToStr(eMasterState)));
```

Then, the RAS client has to be initialized:

```
ECMASTERRAS_T_CLNTPARMS oRemoteApiParms;
OsMemset(&oRemoteApiParms, 0, sizeof(ECMASTERRAS_T_CLNTPARMS));

oRemoteApiParms.dwSignature = ECMASTERRASCLIENT_SIGNATURE;
oRemoteApiParms.dwSize = sizeof(ECMASTERRAS_T_CLNTPARMS);

OsMemcpy(&oRemoteApiParms.LogParms, pEcLogParms, sizeof(EC_T_LOG_PARMS));

EC_CPUSET_ZERO(oRemoteApiParms.cpuAffinityMask);
oRemoteApiParms.dwAdmStackSize = 8192; /* 8k */

oRemoteApiParms.pvNotifCtxt = pvMyAppNotifyContext;
oRemoteApiParms.pfNotification = myAppRasNotifyCallback;

dwRes = emRasClntInit(&oRemoteApiParms);
if (dwRes != EC_E_NOERROR)
{
    dwRetVal = dwRes;
    goto Exit;
}
```

An application which uses the remote API to access a master stack needs to call following steps:

```
#include "EcRasClient.h"
.
.
emRasClntInit(…);   /* initialize Remote API Client Module */
.
.
/* do not call emInitMaster(…) */
.
emRasClntAddConnection(…);  /* connect to Remote API Server */
.
.
/* access EC-Master API */
.
.
emRasClntRemoveConnection(…);       /* disconnect from Remote API Server */
.
.
/* do not call emDeinitMaster(…) */
.
emRasClntClose(…);  /* de-initialize Remote API Client Module, closes all␣
↪connections */
```

For closer details find a Remote API Client example project <AtemDemo> with your installed Examples. To get the Remote API Client example use the Buildspec <AtemRasDbg> or <AtemRasRel>.

3.2 Additional API description

Following calls are necessary to initialize, de-initialize and observe the Remote API Client functionality.

3.2.1 emRasClntGetVersion

EC_T_DWORD **emRasClntGetVersion**(EC_T_VOID)
Get the RAS Client version.

Returns
Version as EC_T_DWORD

3.2.2 emRasClntInit

EC_T_DWORD **emRasClntInit**(*ECMASTERRAS_T_CLNTPARMS* *pParms)
Initialize the RAS Client stack.

The Remote API client will be initialized by calling this function.

Parameters
pParms – [in] Init parameters

Returns
EC_E_NOERROR or error code

struct **ECMASTERRAS_T_CLNTPARMS**
RAS Client init parameters.

Public Members

EC_T_DWORD **dwSignature**
[in] Set to ECMASTERRASCLIENT_SIGNATURE

EC_T_DWORD **dwSize**
[in] Set to sizeof(ECMASTERRAS_T_CLNTPARMS)

EC_T_CPUSET **cpuAffinityMask**
[in] CPU affinity mask

EC_T_DWORD **dwAdmPrio**
[in] Priority of Administrative task

EC_T_DWORD **dwAdmStackSize**
[in] Stack size of Administrative task

EC_T_PVOID **pvNotifCtxt**
[in] Notification context returned while calling pfNotification

EC_PF_NOTIFY **pfNotification**
[in] Function pointer called to notify error and status information generated by Remote API Layer

EC_T_DWORD **dwCommThreadStackSize**
[in] Communication Thread Stack Size

3.2.3 emRasClntClose

EC_T_DWORD **emRasClntClose** (EC_T_DWORD dwTimeout)
De-initialize the RAS Client stack (closing all connections)

Parameters
dwTimeout – [in] Timeout in milliseconds

Returns
EC_E_NOERROR or error code

3.2.4 emRasClntAddConnection

EC_T_DWORD **emRasClntAddConnection** (
ECMASTERRAS_T_CLNTCONDESC *pConDesc,
EC_T_DWORD *pdwIdMask
)
Connect to RAS Server.

Parameters

- **pConDesc** – [in] Connection parameters
- **pdwIdMask** – [out] Instance ID mask returned by stack needed for API calls

Returns
EC_E_NOERROR or error code

struct **ECMASTERRAS_T_CLNTCONDESC**
RAS Client connection parameters.

Public Members

EC_T_IPADDR **oAddr**
[in] Remote Access Server (RAS) connect IP address

EC_T_WORD **wPort**
[in] Remote Access Server (RAS) connect port

EC_T_DWORD **dwWatchDog**
[in] Watchdog interval when to send IDL packets

EC_T_DWORD **dwCycleTime**
[in] Cycle Time for Recv Polling

EC_T_DWORD **dwWDTOLimit**
[in] Amount of cycles without receiving commands (idles) before Entering state wdexpired

EC_T_CPUSET **cpuAffinityMask**
[in] CPU affinity mask

EC_T_DWORD **dwRecvPrio**
[in] Thread Priority of RAS Receive Thread

EC_T_DWORD **dwPktAdminSize**
[in] Slave Packet Administrative List Size

EC_T_PVOID **pvConHandle**
[in/out] Connection Handle needed for disconnect

EC_T_DWORD **dwInstanceID**
[in] OEM Master / Simulator Instance. Only used if OEM Key given.

EC_T_UINT64 **qwOemKey**
[in] OEM Key

3.2.5 emRasClntRemoveConnection

EC_T_DWORD **emRasClntRemoveConnection** (
EC_T_VOID *pvConHandle,
EC_T_DWORD dwTimeout
)
Disconnect from RAS Server.

Parameters

- **pvConHandle** – [in] Connection handle received from emRasClntAddConnection
- **dwTimeout** – [in] Timeout in milliseconds to wait for pending threads to terminate

Returns
EC_E_NOERROR or error code

3.2.6 emRasGetConnectionInfo

EC_T_DWORD **emRasGetConnectionInfo** (
EC_T_PVOID pvConHandle,
EC_T_RAS_CONNECTION_INFO *pConInfo
)
Get connection information (SPOC)

The memory for buffer has to be allocated prior to this function call.

Parameters

- **pvConHandle** – [in] Connection handle
- **pConInfo** – [in] Pointer to buffer to store connection info

Returns
EC_E_NOERROR or error code

3.3 API calls supported

This chapter lists the API calls supported via Remote API and their restrictions (if exist). Syntax description of each call may be found in EC-Master Manual.

3.4 Fully supported calls

- EC_T_DWORD **emStart**(EC_T_DWORD dwInstanceID, EC_T_DWORD dwTimeout);
- EC_T_DWORD **emStop**(EC_T_DWORD dwInstanceID, EC_T_DWORD dwTimeout);
- EC_T_DWORD **emGetSlaveId**(EC_T_DWORD dwInstanceID, EC_T_WORD wStationAddress);
- EC_T_DWORD **emGetSlaveIdAtPosition**(EC_T_DWORD dwInstanceID, EC_T_WORD wAutoIncAddress);
- EC_T_BOOL **emGetSlaveProp**(EC_T_DWORD dwInstanceID, EC_T_DWORD dwSlaveId, EC_T_SLAVE_PROP* pSlaveProp);
- EC_T_DWORD **emGetSlaveState**(EC_T_DWORD dwInstanceID, EC_T_DWORD dwSlaveId, EC_T_WORD* pwCurrDevState, EC_T_WORD* pwReqDevState, EC_T_DWORD dwTimeout);
- EC_T_DWORD **emSetSlaveState**(EC_T_DWORD dwInstanceID, EC_T_DWORD dwSlaveId, EC_T_WORD wNewReqDevState, EC_T_DWORD dwTimeout);
- EC_T_DWORD **emTferSingleRawCmd**(EC_T_DWORD dwInstanceID, EC_T_DWORD dwSlaveId, EC_T_BYTE byCmd, EC_T_DWORD dwMemoryAddress, EC_T_VOID* pvData, EC_T_WORD wLen, EC_T_DWORD dwTimeout);
- EC_T_DWORD **emQueueRawCmd**(EC_T_DWORD dwInstanceID, EC_T_DWORD dwSlaveId, EC_T_WORD wInvokeId, EC_T_BYTE byCmd, EC_T_DWORD dwMemoryAddress, EC_T_VOID* pvData, EC_T_WORD wLen);
- EC_T_DWORD **emGetNumConfiguredSlaves**(EC_T_DWORD dwInstanceID);
- EC_T_MBXTFER* **emMbxTferCreate**(EC_T_DWORD dwInstanceID, EC_T_MBXTFER_DESC* pMbxTferDesc);
- EC_T_VOID **emMbxTferDelete**(EC_T_DWORD dwInstanceID, EC_T_MBXTFER* pMbxTfer);
- EC_T_DWORD **emCoeSdoDownloadReq**(EC_T_DWORD dwInstanceID, EC_T_MBXTFER* pMbxTfer, EC_T_DWORD dwSlaveId, EC_T_WORD wObIndex, EC_T_BYTE byObSubIndex, EC_T_DWORD dwTimeout);
- EC_T_DWORD **emCoeSdoDownload**(EC_T_DWORD dwInstanceID, EC_T_DWORD dwSlaveId, EC_T_WORD wObIndex, EC_T_BYTE byObSubIndex, EC_T_BYTE* pbyData, EC_T_DWORD dwDataLen, EC_T_DWORD dwTimeout);
- EC_T_DWORD **emCoeSdoUploadReq**(EC_T_DWORD dwInstanceID, EC_T_MBXTFER* pMbxTfer, EC_T_DWORD dwSlaveId, EC_T_WORD wObIndex, EC_T_BYTE byObSubIndex, EC_T_DWORD dwTimeout);
- EC_T_DWORD **emCoeSdoUpload**(EC_T_DWORD dwInstanceID, EC_T_DWORD dwSlaveId, EC_T_WORD wObIndex, EC_T_BYTE byObSubIndex, EC_T_BYTE* pbyData, EC_T_DWORD dwDataLen, EC_T_DWORD* pdwOutDataLen, EC_T_DWORD dwTimeout);
- EC_T_DWORD **emCoeGetODList**(EC_T_DWORD dwInstanceID, EC_T_MBXTFER* pMbxTfer, EC_T_DWORD dwSlaveId, EC_T_COE_ODLIST_TYPE eListType, EC_T_DWORD dwTimeout);
- EC_T_DWORD **emCoeGetObjectDesc**(EC_T_DWORD dwInstanceID, EC_T_MBXTFER* pMbxTfer, EC_T_DWORD dwSlaveId, EC_T_WORD wObIndex, EC_T_DWORD dwTimeout);
- EC_T_DWORD **emCoeGetEntryDesc**(EC_T_DWORD dwInstanceID, EC_T_MBXTFER* pMbxTfer, EC_T_DWORD dwSlaveId, EC_T_WORD wObIndex, EC_T_BYTE byObSubIndex, EC_T_BYTE byValueInfo, EC_T_DWORD dwTimeout);

- EC_T_DWORD **emSetMasterState**(EC_T_DWORD dwInstanceID, EC_T_DWORD dwTimeout, EC_T_STATE eReqState);
- EC_T_STATE **emGetMasterState**(EC_T_DWORD dwInstanceID);
- EC_T_DWORD **emFoeFileUpload**(EC_T_DWORD dwInstanceID , EC_T_DWORD dwSlaveId, EC_T_CHAR* szFileName, EC_T_DWORD dwFileNameLen, EC_T_BYTE* pbyData, EC_T_DWORD dwDataLen, EC_T_DWORD* pdwOutDataLen, EC_T_DWORD dwPassWd, EC_T_DWORD dwTimeout);
- EC_T_DWORD **emFoeFileDownload**(EC_T_DWORD dwInstanceID, EC_T_DWORD dwSlaveId, EC_T_CHAR* szFileName, EC_T_DWORD dwFileNameLen, EC_T_BYTE* pbyData, EC_T_DWORD dwDataLen, EC_T_DWORD dwPassWd, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatStart**(EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatStop**(EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatGetSlaveId**(EC_T_WORD wStationAddress);
- EC_T_DWORD **ecatGetSlaveIdAtPosition**(EC_T_WORD wAutoIncAddress);
- EC_T_BOOL **ecatGetSlaveProp**(EC_T_DWORD dwSlaveId, EC_T_SLAVE_PROP* pSlaveProp);
- EC_T_DWORD **ecatGetSlaveState**(EC_T_DWORD dwSlaveId, EC_T_WORD* pwCurrDevState, EC_T_WORD* pwReqDevState, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatSetSlaveState**(EC_T_DWORD dwSlaveId, EC_T_WORD wNewReqDevState, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatTferSingleRawCmd**(EC_T_DWORD dwSlaveId, EC_T_BYTE byCmd, EC_T_DWORD dwMemoryAddress, EC_T_VOID* pvData, EC_T_WORD wLen, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatQueueRawCmd**(EC_T_DWORD dwSlaveId, EC_T_WORD wInvokeId, EC_T_BYTE byCmd, EC_T_DWORD dwMemoryAddress, EC_T_VOID* pvData, EC_T_WORD wLen);
- EC_T_DWORD **ecatGetNumConfiguredSlaves**( );
- EC_T_MBXTFER* **ecatMbxTferCreate**(EC_T_MBXTFER_DESC* pMbxTferDesc);
- EC_T_VOID **ecatMbxTferDelete**(EC_T_MBXTFER* pMbxTfer);
- EC_T_DWORD **ecatCoeSdoDownloadReq**(EC_T_MBXTFER* pMbxTfer, EC_T_DWORD dwSlaveId, EC_T_WORD wObIndex, EC_T_BYTE byObSubIndex, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatCoeSdoDownload**(EC_T_DWORD dwSlaveId, EC_T_WORD wObIndex, EC_T_BYTE byObSubIndex, EC_T_BYTE* pbyData, EC_T_DWORD dwDataLen, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatCoeSdoUploadReq**(EC_T_MBXTFER* pMbxTfer, EC_T_DWORD dwSlaveId, EC_T_WORD wObIndex, EC_T_BYTE byObSubIndex, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatCoeSdoUpload**(EC_T_DWORD dwSlaveId, EC_T_WORD wObIndex, EC_T_BYTE byObSubIndex, EC_T_BYTE* pbyData, EC_T_DWORD dwDataLen, EC_T_DWORD* pdwOutDataLen, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatCoeGetODList**(EC_T_MBXTFER* pMbxTfer, EC_T_DWORD dwSlaveId, EC_T_COE_ODLIST_TYPE eListType, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatCoeGetObjectDesc**(EC_T_MBXTFER* pMbxTfer, EC_T_DWORD dwSlaveId, EC_T_WORD wObIndex, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatCoeGetEntryDesc**(EC_T_MBXTFER* pMbxTfer, EC_T_DWORD dwSlaveId, EC_T_WORD wObIndex, EC_T_BYTE byObSubIndex, EC_T_BYTE byValueInfo, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatSetMasterState**(EC_T_DWORD dwTimeout, EC_T_STATE eReqState);

- EC_T_STATE **ecatGetMasterState**(EC_T_VOID);
- EC_T_DWORD **ecatFoeFileUpload**(EC_T_DWORD dwSlaveId, EC_T_CHAR* szFileName, EC_T_DWORD dwFileNameLen, EC_T_BYTE* pbyData, EC_T_DWORD dwDataLen, EC_T_DWORD* pdwOutDataLen, EC_T_DWORD dwPassWd, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatFoeFileDownload**(EC_T_DWORD dwSlaveId, EC_T_CHAR* szFileName, EC_T_DWORD dwFileNameLen, EC_T_BYTE* pbyData, EC_T_DWORD dwDataLen, EC_T_DWORD dwPassWd, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatGetSlaveInpVarInfoNumOf**(EC_T_BOOL bFixedAddress EC_T_WORD wSlaveAddress, EC_T_WORD* pwSlaveInpVarInfoNumOf);
- EC_T_DWORD **emGetSlaveOutpVarInfoNumOf**(EC_T_DWORD dwInstanceID, EC_T_BOOL bFixedAddress, EC_T_WORD wSlaveAddress, EC_T_WORD* pwSlaveOutpVarInfoNumOf);
- EC_T_DWORD **emGetSlaveInpVarInfo**(EC_T_DWORD dwInstanceID, EC_T_BOOL bFixedAddress, EC_T_WORD wSlaveAddress, EC_T_WORD wNumOfVarsToRead, EC_T_PROCESS_VAR_INFO* pSlaveProcVarInfoEntries, EC_T_WORD* pwReadEntries);
- EC_T_DWORD **emGetSlaveOutpVarInfo**(EC_T_DWORD dwInstanceID, EC_T_BOOL bFixedAddress, EC_T_WORD wSlaveAddress, EC_T_WORD wNumOfVarsToRead, EC_T_PROCESS_VAR_INFO* pSlaveProcVarInfoEntries, EC_T_WORD* pwReadEntries);
- EC_T_DWORD **emFindOutpVarByName** (EC_T_DWORD dwInstanceID, EC_T_CHAR* szVariableName, EC_T_PROCESS_VAR_INFO* pSlaveOutpVarInfo);
- EC_T_DWORD **emFindInpVarByName** (EC_T_DWORD dwInstanceID, EC_T_CHAR* szVariableName, EC_T_PROCESS_VAR_INFO* pSlaveOutpVarInfo);
- EC_T_DWORD **ecatIsSlavePresent**(EC_T_DWORD dwSlaveId, EC_T_BOOL* pbPresence);
- EC_T_DWORD **ecatResetSlaveController**(EC_T_BOOL bFixedAddressing, EC_T_WORD wSlaveAddress, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatGetNumConnectedSlaves**(EC_T_VOID)
- EC_T_DWORD **ecatSetProcessData**(EC_T_BOOL bOutputData, EC_T_DWORD wOffset, EC_T_BYTE* pbyData, EC_T_DWORD dwLength, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatGetProcessData**(EC_T_BOOL bOutputData, EC_T_DWORD dwOffset, EC_T_BYTE* pbyData, EC_T_DWORD dwLength, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatReadSlaveRegister**(EC_T_BOOL bFixedAddressing, EC_T_WORD wSlaveAddress, EC_T_WORD wRegisterOffset, EC_T_VOID* pvData, EC_T_WORD wLen, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatWriteSlaveRegister**(EC_T_BOOL bFixedAddressing, EC_T_WORD wSlaveAddress, EC_T_WORD wRegisterOffset, EC_T_VOID* pvData, EC_T_WORD wLen, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatSoeWrite**(EC_T_DWORD dwSlaveId, EC_T_BYTE byDriveNo, EC_T_BYTE byElementFlags, EC_T_WORD wIDN, EC_T_BYTE* pbyData, EC_T_DWORD dwDataLen, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatSoeRead**(EC_T_DWORD dwSlaveId, EC_T_BYTE byDriveNo, EC_T_BYTE byElementFlags, EC_T_WORD wIDN, EC_T_BYTE* pbyData, EC_T_DWORD dwDataLen, EC_T_DWORD* pdwOutDataLen, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatSoeAbortProcCmd**(EC_T_DWORD dwSlaveId, EC_T_BYTE byDriveNo, EC_T_BYTE byElementFlags, EC_T_WORD wIDN, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatReadSlaveEEPRom**(EC_T_BOOL bFixedAddressing, EC_T_WORD wSlaveAddress, EC_T_WORD wEEPRomStartOffset, EC_T_WORD* pwReadData, EC_T_DWORD dwReadLen, EC_T_DWORD* pdwNumOutData, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatWriteSlaveEEPRom**(EC_T_BOOL bFixedAddressing, EC_T_WORD wSlaveAddress, EC_T_WORD wEEPRomStartOffset, EC_T_WORD* pwWriteData, EC_T_DWORD dwWriteLen, EC_T_DWORD dwTimeout);

- EC_T_DWORD **ecatReloadSlaveEEPRom**(EC_T_BOOL bFixedAddressing, EC_T_WORD wSlaveAddress, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatResetSlaveController**(EC_T_BOOL bFixedAddressing, EC_T_WORD wSlaveAddress, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatAssignSlaveEEPRom**(EC_T_BOOL bFixedAddressing, EC_T_WORD wSlaveAddress, EC_T_BOOL bSlavePDIAccessEnable, EC_T_BOOL bForceAssign, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatRegisterClient**(EC_PF_NOTIFY pfnNotify, EC_T_VOID* pCallerData, EC_T_REGISTERRESULTS* pRegResults);
- EC_T_DWORD **ecatUnregisterClient**(EC_T_DWORD dwClntId);

3.5 Restricted supported calls

- EC_T_DWORD **emIoControl**(EC_T_DWORD dwInstanceID, EC_T_DWORD dwCode, EC_T_IOCTLPARMS* pParms);
  - Supported:
    - EC_IOCTL_REGISTERCLIENT:
    - EC_IOCTL_UNREGISTERCLIENT:
    - EC_IOCTL_ISLINK_CONNECTED:
    - EC_IOCTL_SET_CYC_ERROR_NOTIFY_MASK:
    - EC_IOCTL_GET_PDMEMORYSIZE
    - EC_IOCTL_SLAVE_LINKMESSAGES:
    - EC_IOCTL_DC_SLV_SYNC_STATUS_GET:
    - EC_IOCTL_DC_SLV_SYNC_DEVLIMIT_SET:
    - EC_IOCTL_DC_SLV_SYNC_DEVLIMIT_GET:
    - EC_IOCTL_SB_RESTART:
    - EC_IOCTL_SB_STATUS_GET:
    - EC_IOCTL_SB_SET_BUSCNF_VERIFY:
    - EC_IOCTL_SB_SET_BUSCNF_VERIFY_PROP:
    - EC_IOCTL_SB_BUSCNF_GETSLAVE_INFO:
    - EC_IOCTL_SB_BUSCNF_GETSLAVE_INFO_EEP:
    - EC_IOCTL_SB_ENABLE:
  - Not Supported:
    - EC_IOCTL_RESET_SLAVE:
    - EC_IOCTL_FORCE_BROADCAST_DESTINATION:
    - EC_IOCTL_SET_FRAME_LOSS_SIMULATION:
    - EC_IOCTL_SET_RXFRAME_LOSS_SIMULATION:
    - EC_IOCTL_SET_TXFRAME_LOSS_SIMULATION:
    - EC_IOCTL_SET_SOFT_ASSERTIONS:
    - EC_IOCTL_SET_HARD_ASSERTIONS:
    - EC_IOCTL_LINKLAYER_DBG_MSG:
    - EC_IOCTL_SET_COE_DBG_LEVEL:

        - EC_IOCTL_GET_CYCLIC_CONFIG_INFO:
        - EC_IOCTL_REGISTER_PDMEMORYPROVIDER:
        - EC_IOCTL_REG_DC_SLV_SYNC_NTFY:
        - EC_IOCTL_UNREG_DC_SLV_SYNC_NTFY:
        - EC_IOCTL_DCM_REGISTER_TIMESTAMP:
        - EC_IOCTL_DCM_UNREGISTER_TIMESTAMP:
        - EC_IOCTL_RED_SET_LINK:
        - EC_IOCTL_SLV_ALIAS_ENABLE:
        - EC_IOCTL_SB_BUSCNF_GETSLAVE_INFO_EX:
- EC_T_DWORD **emConfigureMaster**(EC_T_DWORD dwInstanceID, EC_T_CNF_TYPE eCnfType, EC_T_PBYTE pbyCnfData, EC_T_DWORD dwCnfDataLen);
  - Supported : eCnfType = eCnfType_Data
  - Not supported: eCnfType = eCnfType_Filename
- EC_T_DWORD **ecatIoControl**(EC_T_DWORD dwCode, EC_T_IOCTLPARMS* pParms);
  - Supported:
    - EC_IOCTL_GETSTATE:
    - EC_IOCTL_REGISTERCLIENT:
    - EC_IOCTL_UNREGISTERCLIENT:
    - EC_IOCTL_SET_CYC_ERROR_NOTIFY_MASK:
    - EC_IOCTL_ISLINK_CONNECTED:
    - EC_IOCTL_SET_PHYS_MBX_POLLING_PERIOD:
    - EC_IOCTL_SET_SLAVE_STATE_UPDATE_TIMEOUT:
    - EC_IOCTL_RESET_SLAVE:
    - EC_IOCTL_UPDATE_ALL_SLAVE_STATE:
    - EC_IOCTL_GET_PDMEMORYSIZE:
    - EC_IOCTL_FORCE_BROADCAST_DESTINATION:
    - EC_IOCTL_SLAVE_LINKMESSAGES:
    - EC_IOCTL_SET_FRAME_LOSS_SIMULATION:
    - EC_IOCTL_SET_RXFRAME_LOSS_SIMULATION:
    - EC_IOCTL_SET_TXFRAME_LOSS_SIMULATION:
    - EC_IOCTL_SET_SOFT_ASSERTIONS:
    - EC_IOCTL_SET_HARD_ASSERTIONS:
    - EC_IOCTL_DC_ENABLE:
    - EC_IOCTL_DC_DISABLE:
    - EC_IOCTL_DC_SLV_SYNC_STATUS_GET:
    - EC_IOCTL_DC_SLV_SYNC_DEVLIMIT_SET:
    - EC_IOCTL_DC_SLV_SYNC_DEVLIMIT_GET:
    - EC_IOCTL_DC_SLV_SYNC_RESTART:
    - EC_IOCTL_DC_SLV_SYNC_SETTLETIME_SET:

        - EC_IOCTL_DC_SLV_SYNC_SETTLETIME_GET:
        - EC_IOCTL_DC_SLAVESYNCDISABLE:
        - EC_IOCTL_SB_RESTART:
        - EC_IOCTL_SB_STATUS_GET:
        - EC_IOCTL_SB_SET_BUSCNF_VERIFY:
        - EC_IOCTL_SB_SET_BUSCNF_VERIFY_PROP:
        - EC_IOCTL_SB_BUSCNF_GETSLAVE_INFO:
        - EC_IOCTL_SB_BUSCNF_GETSLAVE_INFO_EEP:
        - EC_IOCTL_SB_ENABLE:
        - EC_IOCTL_SB_BUSCNF_GETSLAVE_INFO_EX:
        - EC_IOCTL_SLV_ALIAS_ENABLE:
    - Not Supported:
        - EC_IOCTL_LINKLAYER_DBG_MSG:
        - EC_IOCTL_SET_COE_DBG_LEVEL:
        - EC_IOCTL_GET_CYCLIC_CONFIG_INFO:
        - EC_IOCTL_REGISTER_PDMEMORYPROVIDER:
        - EC_IOCTL_REG_DC_SLV_SYNC_NTFY:
        - EC_IOCTL_UNREG_DC_SLV_SYNC_NTFY:
        - EC_IOCTL_REG_DC_MAST_SYNC_NTFY:
        - EC_IOCTL_UNREG_DC_MAST_SYNC_NTFY:
        - EC_IOCTL_DC_SYSTIME_ADD_OFFSET:
        - EC_IOCTL_DC_PDM_CYCLES_SET:
        - EC_IOCTL_DC_PDM_CYCLES_GET:
        - EC_IOCTL_DC_CONFIGURE_BURST:
        - EC_IOCTL_DCM_REGISTER_TIMESTAMP:
        - EC_IOCTL_DCM_UNREGISTER_TIMESTAMP:
        - EC_IOCTL_RED_SET_LINK:
- EC_T_DWORD **ecatConfigureMaster**(EC_T_CNF_TYPE eCnfType, EC_T_PBYTE pbyCnfData, EC_T_DWORD dwCnfDataLen);
    - Supported : eCnfType = eCnfType_Data
    - Not supported: eCnfType = eCnfType_Filename
- EC_T_DWORD **ecatNotify**(EC_T_DWORD dwCode, EC_T_NOTIFYPARMS* pParms);
    - Supported:
        - EC_NOTIFY_STATECHANGED:
        - EC_NOTIFY_CYCCMD_WKC_ERROR:
        - EC_NOTIFY_MASTER_INITCMD_WKC_ERROR:
        - EC_NOTIFY_SLAVE_INITCMD_WKC_ERROR:
        - EC_NOTIFY_COE_MBXRCV_WKC_ERROR:
        - EC_NOTIFY_COE_MBXSND_WKC_ERROR:

    - EC_NOTIFY_SLAVE_NOT_ADDRESSABLE:
    - EC_NOTIFY_FRAME_RESPONSE_ERROR:
    - EC_NOTIFY_SLAVE_INITCMD_RESPONSE_ERROR:
    - EC_NOTIFY_MBSLAVE_INITCMD_TIMEOUT:
    - EC_NOTIFY_MASTER_INITCMD_RESPONSE_ERROR:
    - EC_NOTIFY_CMD_MISSING:
    - EC_NOTIFY_NOT_ALL_DEVICES_OPERATIONAL:
    - EC_NOTIFY_STATUS_SLAVE_ERROR:
    - EC_NOTIFY_SLAVE_ERROR_STATUS_INFO:
    - EC_NOTIFY_ETH_LINK_NOT_CONNECTED:
    - EC_NOTIFY_RED_LINEBRK:
    - EC_NOTIFY_ETH_LINK_CONNECTED:
    - EC_NOTIFY_SB_STATUS:
    - EC_NOTIFY_RAWCMD_DONE:
    - EC_NOTIFY_MBOXRCV:
  - Not Supported:
    - EC_NOTIFY_CYCCMD_TIMEOUT:
    - EC_NOTIFY_DC_STATUS:
    - EC_NOTIFY_DC_SLV_SYNC:
    - EC_NOTIFY_DC_MAST_SYNC:
    - EC_NOTIFY_DC_MAST_SYNC_CYC:
    - EC_NOTIFY_DCL_STATUS:
    - EC_NOTIFY_DCL_SLV_LATCH_EVT:
    - EC_NOTIFY_DCL_SLV_TIMER_READ:
    - EC_NOTIFY_COE_TX_PDO:
    - EC_NOTIFY_EOE_MBXRCV_WKC_ERROR:
    - EC_NOTIFY_FOE_MBXRCV_WKC_ERROR:
    - EC_NOTIFY_EOE_MBXSND_WKC_ERROR:
    - EC_NOTIFY_FOE_MBXSND_WKC_ERROR:

3.6 Not supported calls

- EC_T_DWORD **emCoeRxPdoTfer**(EC_T_DWORD dwInstanceID, EC_T_MBXTFER* pMbxTfer, EC_T_DWORD dwSlaveId, EC_T_DWORD dwNumber, EC_T_DWORD dwTimeout);
- EC_T_DWORD **emExecJob**(EC_T_DWORD dwInstanceID, EC_T_USER_JOB eUserJob, EC_T_PVOID pvParam);
- EC_T_DWORD **ecatCoeRxPdoTfer**(EC_T_MBXTFER* pMbxTfer, EC_T_DWORD dwSlaveId, EC_T_DWORD dwNumber, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatExecJob**(EC_T_USER_JOB eUserJob, EC_T_PVOID pvParam);

- EC_T_DWORD **ecatEthDbgMsg**(EC_T_BYTE byEthTypeByte0, EC_T_BYTE byEthTypeByte1, EC_T_CHAR* szMsg);
- EC_T_DWORD **ecatDcConfigure**(EC_T_DC_CONFIGURE* pDcConfigure);
- EC_T_DWORD **ecatBlockNode**(EC_T_SB_MISMATCH_DESC, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatOpenBlockedPorts**(EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatForceTopologyChange**(EC_T_DWORD dwInstanceID);
- EC_T_DWORD **ecatDcDisable**();
- EC_T_DWORD **ecatDcDisable**();
- EC_T_DWORD **ecatSetSlavePortState**(EC_T_DWORD dwSlaveID, EC_T_WORD wPort, EC_T_BOOL bClose, EC_T_BOOL bForce, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatHCGetSlaveIdsOfGroup**(EC_T_DWORD dwGroupIndex, EC_T_DWORD* adwSlaveId, EC_T_DWORD dwMaxNumSlaveIds);
- EC_T_DWORD **ecatHCGetNumGroupMembers**(EC_T_DWORD dwGroupIndex);
- EC_T_DWORD **ecatHCAcceptTopoChange**(EC_T_VOID);
- EC_T_DWORD **ecatReloadSlaveEEPRom**(EC_T_BOOL bFixedAddressing, EC_T_WORD wSlaveAddress, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatWriteSlaveEEPRom**(EC_T_BOOL bFixedAddressing, EC_T_WORD wSlaveAddress, EC_T_WORD wEEPRomStartOffset, EC_T_WORD* pwWriteData, EC_T_DWORD dwWriteLen, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatReadSlaveEEPRom**(EC_T_BOOL bFixedAddressing, EC_T_WORD wSlaveAddress, EC_T_WORD wEEPRomStartOffset,EC_T_WORD* pwReadData,EC_T_DWORD dwReadLen, EC_T_DWORD* pdwNumOutData, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatEoeRegisterEndpoint**(EC_T_CHAR* szEoEDrvIdent, EC_T_VOID* pLinkDrvDesc);
- EC_T_DWORD **ecatSoeAbortProcCmd**(EC_T_DWORD dwSlaveId,EC_T_BYTE byDriveNo, EC_T_BYTE byElementFlags, EC_T_WORD wIDN,EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatSoeRead**(EC_T_DWORD dwSlaveId,EC_T_BYTE byDriveNo,EC_T_BYTE byElementFlags,EC_T_WORD wIDN, EC_T_BYTE* byData,EC_T_DWORD dwDataLen, EC_T_DWORD* pdwOutDataLen, EC_T_DWORD dwTimeout);
- EC_T_DWORD **ecatSoeWrite**(EC_T_DWORD dwSlaveId, EC_T_BYTE byDriveNo, EC_T_BYTE byElementFlags, EC_T_WORD wIDN, EC_T_BYTE* pbyData, EC_T_DWORD dwDataLen, EC_T_DWORD dwTimeout);