acontis technologies GmbH

SOFTWARE

EC-Master

EtherCAT® Master Stack

Feature Pack RAS

Edition: 2021-07-05

Content

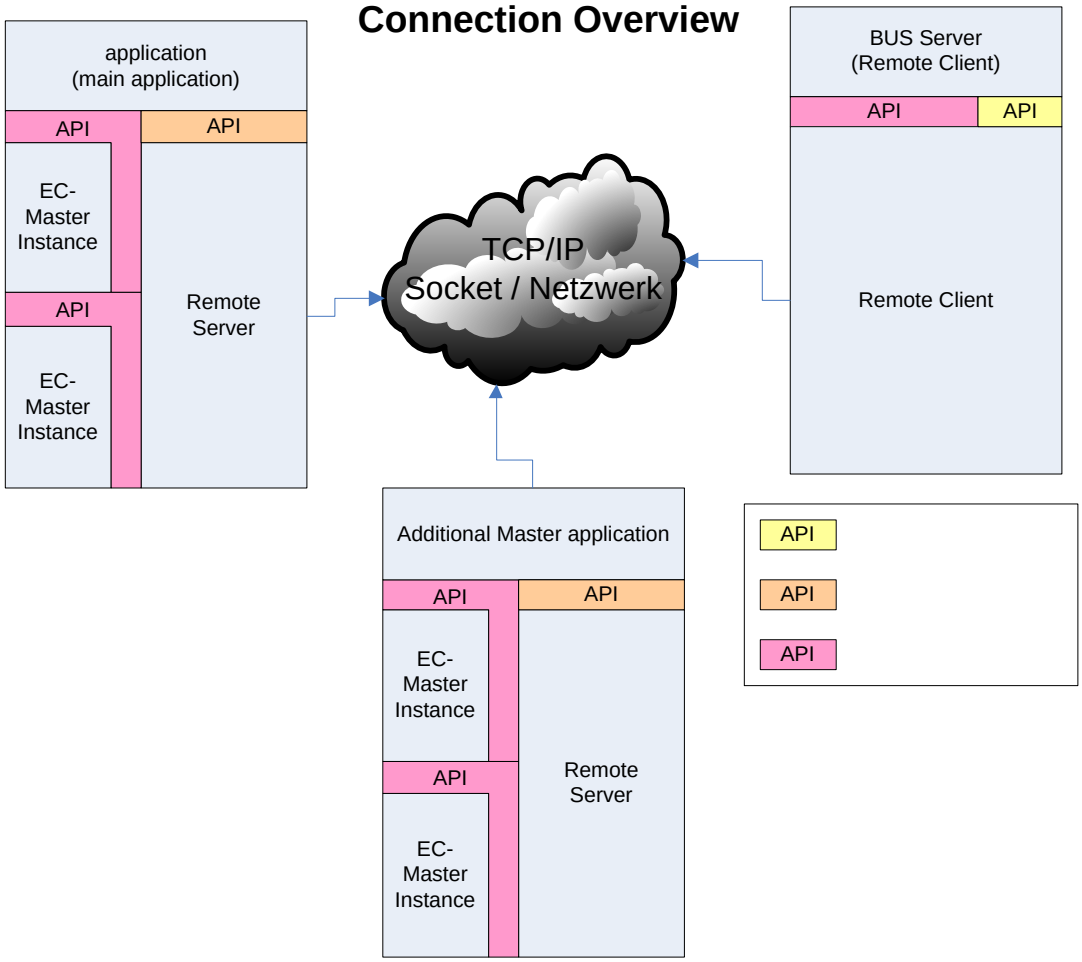
1 Introduction

1.1 What is Remote API?

Within a Windows CE™ System when a second process, e.g. an OPC Server, is likely to access data of the EtherCAT® Bus system or to perform Operations on the EtherCAT® stack, the Remote API provides an interface to do so. Within Windows CE™ two applications (EXE – Files) cannot access each other's memory therefore the Remote API works by aims of a TCP/IP connection, which allows on the other hand to access the Remote Interface also from a different Host system like Windows XP™ or similar.

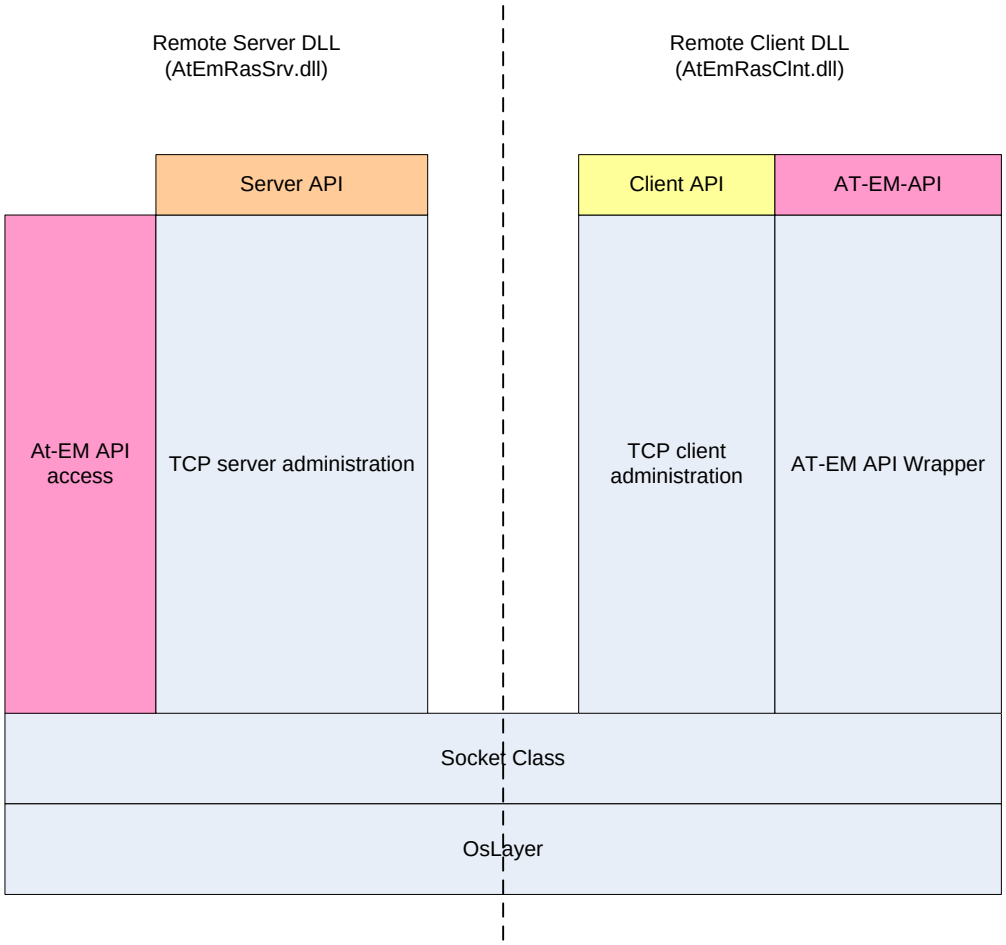
1.2 Remote API Structure

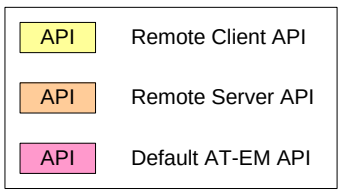
The Remote API works based on TCP/IP sockets, which is almost completely transparent to the calling application.



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

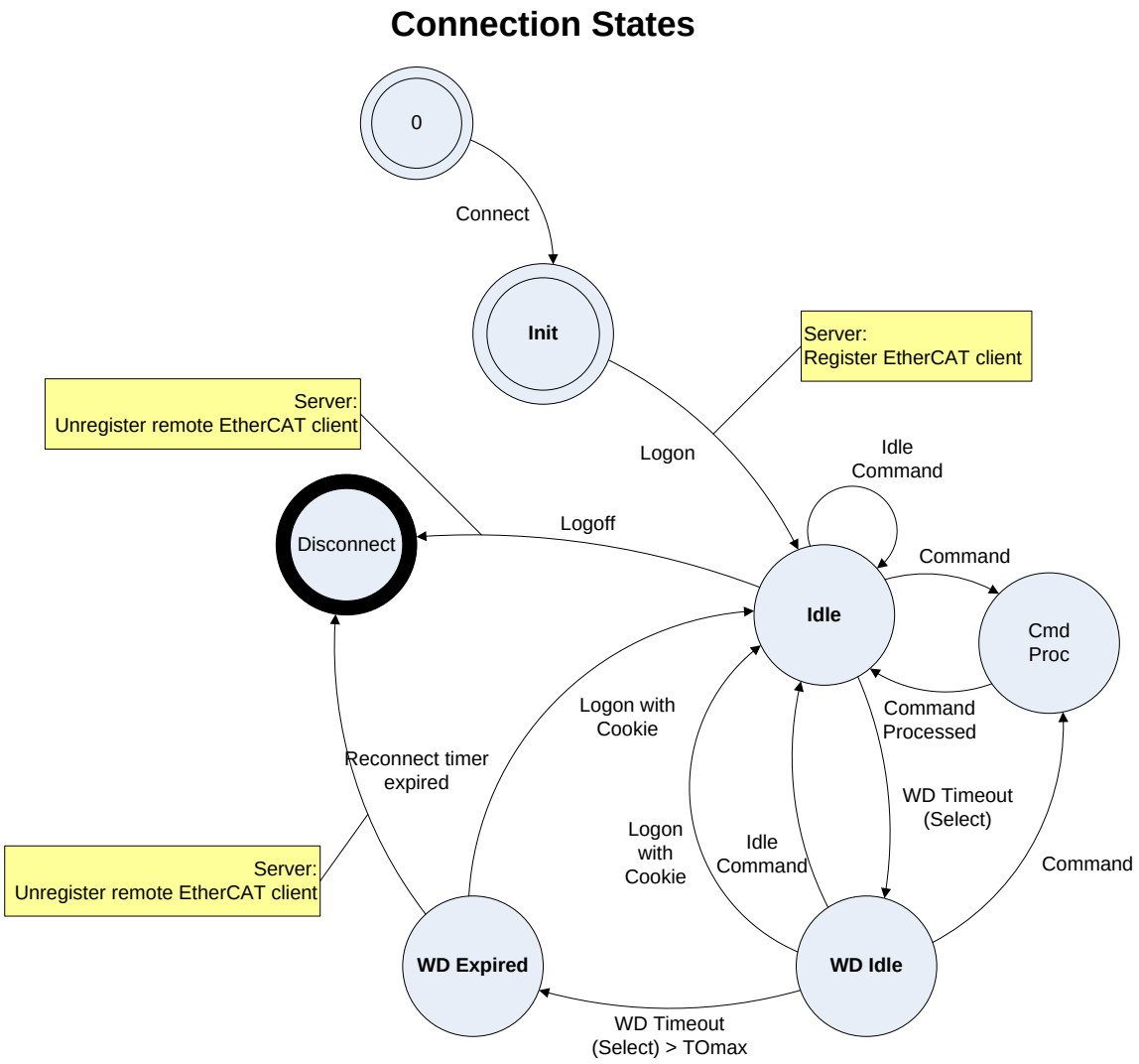

Remote Server DLL
(AtEmRasSrv.dll)
Remote Client DLL
(AtEmRasClnt.dll)
Server API
Client API
AT-EM-API
At-EM API access
TCP server administration
TCP client administration
AT-EM API Wrapper
Socket Class
OsLayer


API
Remote Client API
API
Remote Server API
API
Default AT-EM API

1.3 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 (→ see EC-Master Manual) 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

2.1 Server site (Remote API Server)

2.1.1 Overview

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

a) Link the Remote Server API Lib to the Project

b) Make the Remote API Server DLL available to the Runtime environment of your application.

c) Include the necessary curve up and shutdown calls to your master application

d) Compile

e) Run

2.1.2 Pseudo Example

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

```
#include <AtEmRasSrv.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.

2.1.3 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: AtemRasSrv.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 ATEMRAS_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

2.1.4 Additional API description

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

2.1.4.1 emRasSrvGetVersion

Get Version of Remote API Server Software.

```
EC_T_DWORD emRasSrvGetVersion(
    EC_T_VOID
);
```

Parameters

–

Return

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

Comment

-.

2.1.4.2 emRasSrvStart

Initializes and start remote API Server Instance.

```
EC_T_DWORD emRasSrvStart (
    ATEMRAS_T_SRVPARMS*     pParms,
    EC_T_PVOID*             ppHandle
);
```

Parameters

pParms

[in] Parameter definitions

ppHandle

[out] Server Instance handle

Return

EC_E_NOERROR or error code

Comment

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

ATEMRAS_T_SRVPARMS

Remote API Server initialization parameters.

```
typedef struct _ATEMRAS_T_SRVPARMS
{
        EC_T_DWORD          dwSignature;
        EC_T_DWORD          dwSize;
        EC_T_LOG_PARMS      LogParms;

        ATEMRAS_T_IPADDR    oAddr;
        EC_T_WORD           wPort;
        EC_T_WORD           wMaxClientCnt;
        EC_T_DWORD          dwCycleTime;
        EC_T_DWORD          dwWDTOLimit;

        EC_T_CPUSET         cpuAffinityMask;
        EC_T_DWORD          dwMasterPrio;
        EC_T_DWORD          dwClientPrio;

        EC_T_DWORD          dwConcNotifyAmount
        EC_T_DWORD          dwMbxNotifyAmount;
        EC_T_DWORD          dwMbxUsrNotifySize
        EC_T_PVOID          pvNotifCtxt;
        EC_PF_NOTIFY        pfNotification;
        EC_T_DWORD          dwCycErrInterval;
} ATEMRAS_T_SRVPARMS;
```

Description

dwSignature
: [in] Set to ATEMRASSRV_SIGNATURE

dwSize
: [in] Set to sizeof(ATEMRAS_T_SRVPARMS)

LogParms
: [in] Logging parameters

oAddr
: [in] IP Address to bind Remote API Server to (DWORD or 4Byte Array).

wPort
: [in] TCP Port to bind Remote API Server to.

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

dwCycleTime
: [in] Time in milliseconds which determines the timeout value of a poll cycle which either accepts a new connection or, if connection established, reads commands from the socket Interface. This is the maximum timeout data is processed asynchronuous when ready for read.

dwWDTOLimit
: [in] Amount of cycles (determined by dwCycleTime) before connection enters wdexpired state.

cpuAffinityMask
: [in] CPU affinity mask

dwMasterPrio
: [in] Priority of connection acceptor thread.

dwClientPrio
: [in] Priority of command receiver thread in an established connection state.

dwConcNotifyAmount
: [in] Amount of concurrently queueable Notifications (not Mailboxes).

dwMbxNotifyAmount
: [in] Amount of pre-allocated notification memory buffers used for mailbox notifications. The application can handle up to this amount of mailboxes simultaneously. For security reasons the actual used amount of mailboxes shall be slightly lower than dwMbxNotifyAmount.

dwMbxUsrNotifySize
: [in] User definable amount of bytes each mailbox notification buffer is enlarged off. This value should be at least the size of the largest transferred / used mailbox object.

pvNotifCtxt
: [in] Buffer to user defined data, which is passed to each call of Remote API Server Notification function.

pfNotification
: [in] Pointer to function which is called to notify change of state or errors.

dwCycErrInterval
: [in] Shortest amount of time in msec in between two cyclic error messages of the same kind are transferred to a remote client.

2.1.4.3 emRasSrvStop

Stop and de-initialize remote API Server Instance.

```
EC_T_DWORD emRasSrvStop(
    EC_T_PVOID pvHandle,
    EC_T_DWORD dwTimeout
);
```

Parameters

pvHandle

[in] Handle retrieved from emRasSrvStart

dwTimeout

[in] Timeout used to shut down all spawned threads. This timeout value is in msecs and multiplied internally by the amount of threads spawned.

Return

EC_E_NOERROR or error code

Comment

-

2.1.4.4 emrasNotify - xxx

Callback function called by Remote API Server in case of State changes or error situations.

```
EC_T_DWORD emrasNotify(
    EC_T_DWORD              dwCode,
    EC_T_NOTIFYPARMS*       pParms
);
```

Parameter

dwCode
[in] Notification code
pParms
[in] Notification code depending data

Return

EC_E_NOERROR or error code

Comment

EC_T_NOTIFYPARMS
Data structure filled with detailed information about the according notification.

```
typedef struct _EC_T_NOTIFYPARMS{
    EC_T_VOID*              pCallerData;
    EC_T_BYTE*              pbyInBuf;
    EC_T_DWORD              dwInBufSize;
    EC_T_BYTE*              pbyOutBuf;
    EC_T_DWORD              dwOutBufSize;
    EC_T_DWORD*             pdwNumOutData;
} EC_T_NOTIFYPARMS;
```

Description

pCallerData
[in] Client depending caller data parameter. This pointer is one of the parameters when the client registers with the master.
pbyInBuf
[in] Notification input parameters.
dwInBufSize
[in] Size of the input parameter buffer.
pbyOutBuf
[out] Notification output (result).
dwOutBufSize
[in] Size of the output buffer.
pdwNumOutData
[out] Actually used buffer size of the output buffer.

2.1.4.5 emrasNotify – ATEMRAS_NOTIFY_CONNECTION

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

Parameters

pbyInBuf
[in] Pointer to data of type ATEMRAS_T_CONNOTIFYDESC.
dwInBufSize
[in] sizeof(ATEMRAS_T_CONNOTIFYDESC)
pbyOutBuf
[in] NULL (not used).
dwOutBufSize
[in] 0 (not used).
pdwNumOutData
[out] NULL (not used).

Comment

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

```
typedef struct _ATEMRAS_T_CONNOTIFYDESC{
    EC_T_DWORD          dwCause;
    EC_T_DWORD          dwCookie;
} ATEMRAS_T_CONNOTIFYDESC;
```

Description
dwCause
[in] Cause of state connection state change which is one of:

EC_E_NOERROR :	new logon
EMRAS_E_LOGONCANCELLED:	error during logon
EMRAS_EVT_RECONNECT:	resume of former connection
EMRAS_EVT_RECONEXPIRED:	re-connect failed due to long term timeout
EMRAS_EVT_SERVERSTOPPED:	RAS Server shutdown, re-connects impossible
EC_E_INVALIDSTATE:	if accepted socket object is invalid

dwCookie
[in] Unique identification cookie of connection instance.

2.1.4.6 emrasNotify – ATEMRAS_NOTIFY_REGISTER

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

Parameters

pbyInBuf
[in] Pointer to data of type ATEMRAS_T_REGNOTIFYDESC.
dwInBufSize
[in] sizeof(ATEMRAS_T_REGNOTIFYDESC)
pbyOutBuf
[in] NULL (not used).
dwOutBufSize
[in] 0 (not used).
pdwNumOutData
[out] NULL (not used).

Comment

ATEMRAS_T_REGNOTIFYDESC

```
typedef struct _ATEMRAS_T_REGNOTIFYDESC{
    EC_T_DWORD          dwCookie;
    EC_T_DWORD          dwResult;
    EC_T_DWORD          dwInstanceId;
    EC_T_DWORD          dwClientId;
} ATEMRAS_T_REGNOTIFYDESC;
```

Description
dwCookie
[in] Unique identification cookie of connection instance.
dwResult
[in] Result of registration request.
dwInstanceId
[in] Master Instance client registered to.
dwClientId
[in] Client ID of registered client.

2.1.4.7 emrasNotify – ATEMRAS_NOTIFY_UNREGISTER

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

Parameters

pbyInBuf
[in] Pointer to data of type ATEMRAS_T_REGNOTIFYDESC.
dwInBufSize
[in] sizeof(ATEMRAS_T_REGNOTIFYDESC)
pbyOutBuf
[in] NULL (not used).
dwOutBufSize
[in] 0 (not used).
pdwNumOutData
[out] NULL (not used).

Comment

ATEMRAS_T_REGNOTIFYDESC

```
typedef struct _ATEMRAS_T_REGNOTIFYDESC{
    EC_T_DWORD          dwCookie;
    EC_T_DWORD          dwResult;
    EC_T_DWORD          dwInstanceId;
    EC_T_DWORD          dwClientId;
} ATEMRAS_T_REGNOTIFYDESC;
```

Description
dwCookie
[in] Unique identification cookie of connection instance.
dwResult
[in] Result of un - registration request.
dwInstanceId
[in] Master Instance client un - registered from.
dwClientId
[in] Client ID of un - registered client.

2.1.4.8 emrasNotify – ATEMRAS_NOTIFY_MARSHALERROR

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

Parameters

pbyInBuf
[in] Pointer to data of type ATEMRAS_T_MARSHALERRORDESC.
dwInBufSize
[in] sizeof(ATEMRAS_T_MARSHALERRORDESC)
pbyOutBuf
[in] NULL (not used).
dwOutBufSize
[in] 0 (not used).
pdwNumOutData
[out] NULL (not used).

Comment

ATEMRAS_T_MARSHALERRORDESC

```
typedef struct _ATEMRAS_T_MARSHALERRORDESC{
    EC_T_DWORD          dwCookie;
    EC_T_DWORD          dwCause;
    EC_T_DWORD          dwLenStatCmd;
    EC_T_DWORD          dwCommandCode;
} ATEMRAS_T_MARSHALERRORDESC;
```

Description
dwCookie
[in] Unique identification cookie of connection instance.
dwCause
[in] Error code.
dwLenStaCmd
[in] Length of faulty command.
dwCommandCode
[in] Command code of faulty command.

2.1.4.9 emrasNotify – ATEMRAS_NOTIFY_ACKERROR

Notification about an error during creation of ack / nack packet.

Parameters

pbyInBuf
[in] Pointer to EC_T_DWORD containing error code.
dwInBufSize
[in] sizeof(EC_T_DWORD)
pbyOutBuf
[in] NULL (not used).
dwOutBufSize
[in] 0 (not used).
pdwNumOutData
[out] NULL (not used).

Comment

–

2.1.4.10 emrasNotify – ATEMRAS_NOTIFY_NONOTIFYMEMORY

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

Parameters

pbyInBuf
[in] Pointer to EC_T_DWORD containing unique identification cookie of connection instance.
dwInBufSize
[in] sizeof(EC_T_DWORD)
pbyOutBuf
[in] NULL (not used).
dwOutBufSize
[in] 0 (not used).
pdwNumOutData
[out] NULL (not used).

Comment

–

2.1.4.11 emrasNotify – ATEMRAS_NOTIFY_STDNOTIFYMEMORYSMALL

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

Parameters

pbyInBuf
[in] Pointer to EC_T_DWORD containing unique identification cookie of connection instance.
dwInBufSize
[in] sizeof(EC_T_DWORD)
pbyOutBuf
[in] NULL (not used).
dwOutBufSize
[in] 0 (not used).
pdwNumOutData
[out] NULL (not used).

Comment

–

2.1.4.12 emrasNotify – ATEMRAS_NOTIFY_MBXNOTIFYMEMORYSMALL

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

Parameters

pbyInBuf
[in] Pointer to EC_T_DWORD containing unique identification cookie of connection instance.
dwInBufSize
[in] sizeof(EC_T_DWORD)
pbyOutBuf
[in] NULL (not used).
dwOutBufSize
[in] 0 (not used).
pdwNumOutData
[out] NULL (not used).

Comment

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.

2.1.5 Access control

The access control is used in order to restrict RAS client in calling API functions. The access control is deactivated by default. The global access level can be set to one of the following value (in order of access rights lowering) after activating the access control:

- “Full access” (default), all API functions may be executed by client, see also “opt-out” below.
- “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, see also “opt-in” below.

In order to configure the access control subsystem emRasSrvConfigAccessLevel is used, to activate or 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 the error code EMRAS_E_ACCESSLESS will be returned.

In case the configuration is missing, the error code EMRAS_E_ACCESS_NOT_FOUND will be returned.

The RAS server notifies the application whenever a blocked API is called.

2.1.5.1 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.

2.1.5.2 emRasSrvConfigAccessLevel

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

```
EC_T_DWORD emRasSrvConfigAccessLevel (
    EC_T_PVOID              pvHandle,
    ATEMRAS_T_SPOCCFG*      pCfgData,
    EC_T_DWORD              dwCfgDataCnt
    );
```

Parameters

pvHandle
[in] Handle to previously started server
pCfgData
[in] Pointer to Configuration data or EC_NULL for default configuration
dwCfgDataCnt
[in] Amount of configuration data entries

Return

EC_E_NOERROR or error code

Comment

pCfgData optionally defines the access control configuration in an array with each entry of type ATEMRAS_T_SPOCCFG, see chapter 2.1.5.1 “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.

ATEMRAS_T_SPOCCFG
The access control subsystem configuration structure.

```
typedef
{
    EC_T_DWORD          dwAccessLevel;
    EC_T_DWORD          dwOrdinal;
    EC_T_DWORD          dwIndex;
    EC_T_DWORD          dwSubIndex;
    EC_T_DWORD          dwReserved;
} ATEMRAS_T_SPOCCFG;
```

Description
dwAccessLevel
[in] access level for this API call, can be one of the following

ATEMRAS_ACCESS_LEVEL_ALLOW_ALL: all functions calls allowed, i.e. change of master state as well
ATEMRAS_ACCESS_LEVEL_READWRITE: functions with parameter change, i.e. set or download
ATEMRAS_ACCESS_LEVEL_READONLY: functions with no parameter change, i.e. get or upload
ATEMRAS_ACCESS_LEVEL_BLOCK_ALL: no functions calls allowed
ATEMRAS_ACCESS_LEVEL_EXCLUDED: never allowed, disregarding the global control level (opt-out)

dwOrdinal
[in] API call ID

ord_emInitMaster	201
ord_emDeinitMaster	202
ord_emStart	203
ord_emStop	204
ord_emIoControl	205
ord_emGetSlaveId	207
ord_emMbxTferCreate	208
ord_emMbxTferDelete	209
ord_emCoeSdoDownloadReq	210
ord_emCoeSdoUploadReq	211
ord_emCoeGetODList	212
ord_emCoeGetObjectDesc	213
ord_emCoeGetEntryDesc	214
ord_emGetSlaveProp	218
ord_emGetSlaveState	219
ord_emSetSlaveState	220
ord_emTferSingleRawCmd	221
ord_emGetSlaveIdAtPosition	225
ord_emGetNumConfiguredSlaves	226
ord_emConfigureMaster	227
ord_emSetMasterState	228
ord_emQueueRawCmd	229
ord_emCoeRxPdoTfer	230
ord_emExecJob	231
ord_emGetProcessData	234
ord_emSetProcessData	235
ord_emGetMasterState	236
ord_emFoeFileUpload	237
ord_emFoeFileDownload	238
ord_emFoeUpoadReq	239
ord_emFoeDownloadReq	240
ord_emCoeSdoDownload	241
ord_emCoeSdoUpload	242
ord_emGetNumConnectedSlaves	243

ord_emResetSlaveController	244
ord_emGetSlaveInfo	245
ord_emIsSlavePresent	246
ord_emAoeWriteReq	247
ord_emAoeReadReq	248
ord_emAoeWrite	249
ord_emAoeRead	250
ord_emAoeGetSlaveNetId	251
ord_emGetFixedAddr	252
ord_emGetSlaveProcVarInfoNumOf	253
ord_emGetSlaveProcVarInfo	254
ord_emFindProcVarByName	255
ord_emGetProcessDataBits	256
ord_emSetProcessDataBits	257
ord_emReloadSlaveEEPRom	258
ord_emReadSlaveEEPRom	259
ord_emWriteSlaveEEPRom	260
ord_emAssignSlaveEEPRom	261
ord_emSoeRead	262
ord_emSoeWrite	263
ord_emSoeAbortProcCmd	264
ord_emGetNumConnectedSlavesMain	265
ord_emGetNumConnectedSlavesRed	266
ord_emNotifyApp	267
ord_emAoeReadWriteReq	268
ord_emAoeReadWrite	269
ord_emGetCfgSlaveInfo	270
ord_emGetBusSlaveInfo	271
ord_emReadSlaveIdentification	272
ord_emSetSlaveDisabled	273
ord_emSetSlaveDisconnected	274
ord_emRescueScan	275
ord_emGetMasterInfo	276
ord_emConfigExtend	277
ord_emAoeWriteControl	278
ord_emSetSlavesDisabled	279
ord_emSetSlavesDisconnected	280
ord_emSetMbxProtocolsSerialize	281
ord_emBadConnectionsDetect	282
ord_emIsConfigured	283
ord_esConnectPorts	605
ord_esDisconnectPort	606
ord_esPowerSlave	607
ord_esSetErrorAtSlavePort	616
ord_esSetErrorGenerationAtSlavePort	617
ord_esResetErrorGenerationAtSlavePorts	618
ord_esSetLinkDownAtSlavePort	619
ord_esSetLinkDownGenerationAtSlavePort	620
ord_esResetLinkDownGenerationAtSlavePorts	621

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

dwReserved

[in] Reserved. Set to 0.

2.1.5.3 emRasSrvSetAccessControl

Activates or deactivates the access control subsystem.

```
EC_T_DWORD emRasSrvSetAccessControl (
        EC_T_PVOID          pvHandle,
        EC_T_BOOL           bActive);
```

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

Return

EC_E_NOERROR or error code

Comment

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

2.1.5.4 emRasSrvSetAccessLevel

Sets the current global access level.

```
EC_T_DWORD emRasSrvSetAccessLevel (
        EC_T_PVOID          pvHandle,
        EC_T_DWORD          dwAccessLevel);
```

Parameters

pvHandle
[in] Handle to previously started Server

dwAccessLevel
[in] New access level, see *ATEMRAS_ACCESS_LEVEL_*...

Return

EC_E_NOERROR or error code

Comment

The API calls with required access level lower than *dwAccessLevel* will be blocked from execution.

2.1.5.5 emRasSrvGetAccessLevel

Returns the current access level.

```
EC_T_DWORD emRasSrvGetAccessLevel (
        EC_T_PVOID          pvHandle,
        EC_T_DWORD*         pdwAccessLevel);
```

Parameters

pvHandle
[in] Handle to previously started Server

pdwAccessLevel
[out] Pointer to a buffer storing actual access level, see emRasSrvSetAccessLevel

Return

EC_E_NOERROR or error code

Comment

The memory pointed by *pdwAccessLevel* has to be allocated prior.

2.1.5.6 emRasSrvSetCallAccessLevel

Modifies the required access level for an API call.

```
EC_T_DWORD emRasSrvSetCallAccessLevel (
    EC_T_PVOID      pvHandle,
    EC_T_DWORD      dwOrdinal,
    EC_T_DWORD      dwIndex,
    EC_T_DWORD      dwSubIndex,
    EC_T_DWORD      dwAccessLevel);
```

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_*...

Return

EC_E_NOERROR or error code

Comment

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

2.2 Remote site (Remote API Client)

2.2.1 Overview

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

a) Link the Remote Client API lib to the Project
b) Make the Remote API Client DLL available to the Runtime environment of your application.
c) Include the necessary connect and disconnect calls to your client application.
d) Compile
e) Run

2.2.2 Pseudo Example

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

```
#include <AtEmRasClnt.h>
.
.
emRasClntInit(…);      /* initialize Remote API Client Module */
.
.
/* do not call ecatInitMaster(…) */
.
emRasClntAddConnection(…);    /* connect to Remote API Server */
.
.
/* access EC-Master API */
.
.
emRasClntRemoveConnection(…); /* disconnect from Remote API Server */
.
.
/* do not call ecatDeinitMaster(…) */
.
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>.

2.2.3 Additional API description

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

2.2.3.1 emRasClntGetVersion

Get Version of Remote API Server Software.

```
EC_T_DWORD emRasClntGetVersion(
    EC_T_VOID
);
```

Parameters

–

Return

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

Comment

-.

2.2.3.2 emRasClntInit

Initializes remote API client instance.

```
EC_T_DWORD emRasClntInit (
    ATEMRAS_T_CLNTPARMS* pParms
);
```

Parameters

oParms

[in] Parameter definitions

Return

EC_E_NOERROR or error code

Comment

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

ATEMRAS_T_CLNTPARMS

Remote API Client initialization parameters.

```
typedef struct _ATEMRAS_T_CLNTPARMS
{
    EC_T_DWORD          dwSignature;
    EC_T_DWORD          dwSize;
    EC_T_LOG_PARMS      LogParms;

    EC_T_CPUSET         cpuAffinityMask;
    EC_T_DWORD          dwAdmPrio;
    EC_T_DWORD          dwAdmStackSize;

    EC_T_PVOID          pvNotifCtxt;
    EC_PF_NOTIFY        pfNotification;
} ATEMRAS_T_CLNTPARMS;
```

Description

dwSignature
[in] Set to ATEMRASCLNT_SIGNATURE

dwSize
[in] Set to sizeof(ATEMRAS_T_CLNTPARMS)

LogParms
[in] Logging parameters

cpuAffinityMask
[in] CPU affinity mask

dwAdmPrio
[in] Priority of Administrative task

dwAdmStackSize
[in] Stack size of Administrative task

pvNotifCtxt
[in] Buffer to user defined data, which is passed to each call of Remote API Server Notification function.

pfNotification
[in] Pointer to function which is called to notify change of state or errors .

dwCycErrInterval
[in] Shortest amount of time in msec in between two cyclic error messages of the same kind are transferred to a remote client.

2.2.3.3 emRasClntClose

Disconnect all client connections and de-initialize Remote API Client instance.

```
EC_T_DWORD emRasClntClose (
    EC_T_DWORD dwTimeout
);
```

Parameters

dwTimeout

[in] Timeout used to shut down all spawned threads. This timeout value is in msecs and multiplied internally by the amount of threads spawned.

Return

EC_E_NOERROR or error code

Comment

-

2.2.3.4 emRasClntAddConnection

Establish connection to a remote server.

```
EC_T_DWORD emRasClntAddConnection (
    ATEMRAS_T_CLNTCONDESC*    pConDesc,
    EC_T_DWORD*               pdwIdMask
);
```

Parameters

pConDesc

[in] Pointer to Parameter structure for connection establishment

pdwIdMask

[out] Instance ID which has to be or'ed to any multi – instance API call of EC-Master API when used on a remote client. This Id identifies the connection to the remote host (which could be more than one at a time)

Return

EC_E_NOERROR or error code

Comment

ATEMRAS_T_CLNTCONDESC
Remote API Client connection parameters.

```
typedef struct _ATEMRAS_T_CLNTCONDESC{
    ATEMRAS_T_IPADDR        oAddr;
    EC_T_WORD               wPort;
    EC_T_DWORD              dwWatchDog;
    EC_T_DWORD              dwCycleTime;
    EC_T_DWORD              dwWDTOLimit;
    EC_T_DWORD              dwRecvPrio;
    EC_T_DWORD              dwStackSize;
    EC_T_DWORD              dwPktAdminSize;
    EC_T_VOID*              pvConHandle;
} ATEMRAS_T_CLNTCONDESC;
```

Description

oAddr
: [in] IP Address of remote API server (DWORD or 4 Byte Array).

wPort
: [in] Port of remoite API server.

dwWatchDog
: [in] Timeout in msecs. After this amount of time a idle packet is send in case no other command was send by remote client or remote server.

dwCycleTime
: [in] Time in milliseconds which determines the timeout value of a poll cycle which reads commands from the socket Interface. This is the maximum timeout data is processed asynchronuous when ready for read.

dwWDTOLimit
: [in] Amount of cycles (determined by dwCycleTime) before connection enters wdexpired state (currently unused).

dwRecvPrio
: [in] Priority of command receiver thread (should be higher than application's threads because notifications from Remote Server are raised from here).

dwStackSize
: [in] Size of stack of command receiver thread.

dwPktAdminSize
: [in] Granularity of client connection send queue. Multiples of this amount entries are allocated, when send queue runs out of buffers.

pvConHandle
: [out] Pointer which carries connection handle after successful return. This handle is used for emRasClntRemoveConnection.

2.2.3.5 emRasClntRemoveConnection

Tear down an existing connection to a remote server.

```
EC_T_DWORD emRasClntRemoveConnection (
    EC_T_VOID*                  pvConHandle,
    EC_T_DWORD                  dwTimeout
);
```

Parameters

pvConHandle

[in] Connection handle received from emRasClntAddConnection.

dwTimeout

[in] Timeout to wait for pending threads to shut down.

Return

EC_E_NOERROR or error code

Comment

-

2.2.3.6 emRasGetConnectionInfo

Get Version of Remote API Server Software.

```
EC_T_DWORD emRasGetConnectionInfo (
        EC_T_PVOID                          pvConHandle,
        struct _EC_T_RAS_CONNECTION_INFO*   pConInfo
        );
```

Parameters

pvConHandle

[in] Connection handle

pConInfo

[in] Pointer to buffer to store connection info

```
typedef struct _EC_T_RAS_CONNECTION_INFO
{
        EC_T_DWORD          dwAccessLevel;
} EC_T_RAS_CONNECTION_INFO;
```

Description

dwAccessLevel

[out] actual access level, see *ATEMRAS_ACCESS_LEVEL_*....

Return

EC_E_NOERROR or error code

Comment

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

2.3 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.

2.3.1 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);

2.3.2 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:

2.3.3 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);