acontis technologies GmbH

SOFTWARE

EC-Master

Feature Pack UDP-Mailbox-Gateway

Version 3.2

Edition: September 23, 2025

Contents

1 Introduction

The Mailbox Gateway is a functionality of the EtherCAT Master to access EtherCAT devices within an EtherCAT network from an application outside the EtherCAT network.

It is following the Function Guideline *ETG.8200 EtherCAT Mailbox Gateway*.

All Mailbox protocols defined in the EtherCAT specification can be used in general, i.e. CoE, FoE, VoE, SoE.

There is no error handling specified for the Mailbox Gateway functionality. A request to a non-existing slave device can lead to no response from the master.

Running the Mailbox Gateway Server on the top of EC-Master makes possible to use for example the Beckhoff TwinSafe Loader to apply the configuration to the safety devices.

To reduce development effort the Mailbox Gateway Server is provided by RAS library, but has to be licensed separately.

2 Programmers Guide

2.1 emMbxGatewaySrvStart

EC_T_DWORD **emMbxGatewaySrvStart** (
    EC_T_MBX_GATEWAY_SRV_PARMS *pParms,
    EC_T_VOID **ppvHandle
)

Starts the Mailbox Gateway Server. Must be called after emConfigureMaster().

Parameters

- **pParms** – [in] Init parameters
- **ppvHandle** – [out] Instance handle

Returns
EC_E_NOERROR or error code

struct **EC_T_MBX_GATEWAY_SRV_PARMS**
Mailbox Gateway Server init parameters.

Public Members

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

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

EC_T_LOG_PARMS **LogParms**
[in] Logging parameters

EC_T_IPADDR **oAddr**
[in] Server bind address. The server only listens to this network. A value of 0 binds to every available network

EC_T_WORD **wPort**
[in] Server bind port. The server will listen to this port

EC_T_DWORD **dwInstanceID**
[in] Master Instance ID

EC_T_DWORD **dwMaxQueuedFrames**
[in] Maximum number of queued frames

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

EC_T_DWORD **dwThreadPriority**
[in] Priority of connection acceptor thread

2.2 emMbxGatewaySrvStop

EC_T_DWORD **`emMbxGatewaySrvStop`** (EC_T_VOID *pvHandle, EC_T_DWORD dwTimeout)

Stops the Mailbox Gateway Server. Must be called before emDeinitMaster() or calling emConfigureMaster() again.

Parameters

- **`pvHandle`** – [in] Instance handle from *emMbxGatewaySrvStart()*
- **`dwTimeout`** – [in] Timeout [ms]. The function will block at most for this time.

Returns

EC_E_NOERROR or error code

EC_E_INVALIDSTATE if EC-Master is already de-initialized.

2.3 emMbxGatewaySrvGetVersion

EC_T_DWORD **`emMbxGatewaySrvGetVersion`** (EC_T_VOID)

Get the Mailbox Gateway Server version.

Returns

Version as EC_T_DWORD

2.4 emMbxGatewayClntInit

EC_T_DWORD **`emMbxGatewayClntInit`** (*EC_T_MBX_GATEWAY_CLNT_PARMS* *pParms)

Initialize the Mailbox Gateway Client stack.

Parameters

`pParms` – [in] Init parameters

Returns

EC_E_NOERROR or error code

struct **`EC_T_MBX_GATEWAY_CLNT_PARMS`**

Mailbox Gateway Client init parameters.

Public Members

EC_T_DWORD **`dwSignature`**

[in] Set to EC_MBX_GATEWAY_CLNT_SIGNATURE

EC_T_DWORD **`dwSize`**

[in] Set to sizeof(EC_T_MBX_GATEWAY_CLNT_PARMS)

EC_T_LOG_PARMS **`LogParms`**

[in] Logging parameters

2.5 emMbxGatewayClntDeinit

EC_T_DWORD **emMbxGatewayClntDeinit** (EC_T_VOID)

De-initialize the Mailbox Gateway Client stack (closing all connections)

Returns

EC_E_NOERROR or error code

2.6 emMbxGatewayClntGetVersion

EC_T_DWORD **emMbxGatewayClntGetVersion** (EC_T_VOID)

Get the Mailbox Gateway Client version.

Returns

Version as EC_T_DWORD

2.7 emMbxGatewayClntAddConnection

EC_T_DWORD **emMbxGatewayClntAddConnection** (
EC_T_MBX_GATEWAY_CLNT_CONDESC *pConDesc,
EC_T_DWORD *pdwConnectionId
)

Connect to a mailbox gateway server.

Parameters

- **pConDesc** – [in] Connection parameters
- **pdwConnectionId** – [out] Connection ID

Returns

EC_E_NOERROR or error code

struct **EC_T_MBX_GATEWAY_CLNT_CONDESC**

Mailbox Gateway Client connection parameters.

Public Members

EC_T_LOG_PARMS **LogParms**

[in] Logging parameters

EC_T_IPADDR **oAddr**

[in] Server IP address

EC_T_WORD **wPort**

[in] Server Port

EC_T_WORD **wReserved**

[in] Reserved

EC_T_VOID ***pvConHandle**

[out] Connection handle. Deprecated

2.8 emMbxGatewayClntRemoveConnection

EC_T_DWORD **`emMbxGatewayClntRemoveConnection`** (EC_T_DWORD dwConnectionId)

Close the connection to a mailbox gateway server.

Parameters

`dwConnectionId` – [in] Connection ID

Returns

EC_E_NOERROR or error code

2.9 emMbxGatewayClntCoeSdoUpload

EC_T_DWORD **`emMbxGatewayClntCoeSdoUpload`** (
EC_T_DWORD dwConnectionId,
EC_T_WORD wAddress,
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 dwFlags
)

Performs a CoE SDO upload from an EtherCAT slave device to the Mailbox Gateway Client.

Parameters

- **`dwConnectionId`** – [in] Connection ID
- **`wAddress`** – [in] Station address of slave
- **`wObIndex`** – [in] Object Index
- **`byObSubIndex`** – [in] Object SubIndex. If Complete Access only 0 or 1 allowed
- **`pbyData`** – [in] Data buffer for upload
- **`dwDataLen`** – [in] Length of pbyData in bytes
- **`pdwOutDataLen`** – [out] Pointer returning size of data uploaded from slave
- **`dwTimeout`** – [in] Timeout [ms]. The function will block at most for this time.
- **`dwFlags`** – [in] Mailbox Flags. Bit 0: set if Complete Access (EC_MAILBOX_FLAG_SDO_COMPLETE).

Returns

EC_E_NOERROR or error code

2.10 emMbxGatewayClntCoeSdoDownload

```
EC_T_DWORD emMbxGatewayClntCoeSdoDownload(
        EC_T_DWORD dwConnectionId,
        EC_T_WORD wAddress,
        EC_T_WORD wObIndex,
        EC_T_BYTE byObSubIndex,
        EC_T_BYTE *pbyData,
        EC_T_DWORD dwDataLen,
        EC_T_DWORD dwTimeout,
        EC_T_DWORD dwFlags
)
```

Performs a CoE SDO download from a Mailbox Gateway Client to the EtherCAT slave device.

Parameters

- **dwConnectionId** – [in] Connection ID
- **wAddress** – [in] Station address of slave
- **wObIndex** – [in] Object Index
- **byObSubIndex** – [in] Object SubIndex. If Complete Access only 0 or 1 allowed
- **pbyData** – [in] Data buffer to be transferred
- **dwDataLen** – [in] Length of pbyData in bytes
- **dwTimeout** – [in] Timeout [ms]. The function will block at most for this time.
- **dwFlags** – [in] Mailbox Flags. Bit 0: set if Complete Access (EC_MAILBOX_FLAG_SDO_COMPLETE).

Returns

EC_E_NOERROR or error code

3 Examples

3.1 Server

Start the mailbox gateway server after calling `emConfigureMaster()` and stop prior to `emDeinitMaster()`.

```
/* emInitMaster(..) */
/* emConfigureMaster(..) */

EC_T_VOID* pvMbxGatewaySrvHandle = EC_NULL;
EC_T_MBX_GATEWAY_SRV_PARMS MbxGatewaySrvParms;
OsMemset(&MbxGatewaySrvParms, 0, sizeof(EC_T_MBX_GATEWAY_SRV_PARMS));
MbxGatewaySrvParms.dwSignature = EC_MBX_GATEWAY_SRV_SIGNATURE;
MbxGatewaySrvParms.dwSize = sizeof(EC_T_MBX_GATEWAY_SRV_PARMS);
MbxGatewaySrvParms.oAddr.dwAddr = 0; /* bind to every available network*/
MbxGatewaySrvParms.wPort = EC_MBX_GATEWAY_DEFAULT_PORT;
MbxGatewaySrvParms.dwInstanceID = INSTANCE_MASTER_DEFAULT;
MbxGatewaySrvParms.dwMaxQueuedFrames = 10;
EC_CPUSET_ZERO(MbxGatewaySrvParms.cpuAffinityMask);
MbxGatewaySrvParms.dwThreadPriority = MAIN_THREAD_PRIO;

dwRes = emMbxGatewaySrvStart(&MbxGatewaySrvParms, &pvMbxGatewaySrvHandle);

/* main loop */

dwRes = emMbxGatewaySrvStop(pvMbxGatewaySrvHandle,2000);

/* emDeInitMaster(..) */
```

3.2 Client

```
EC_T_DWORD dwMbxGatewayConId = 0;

/* initialize mailbox gateway */
{
    EC_T_MBX_GATEWAY_CLNT_PARMS MbxGatewayClientParms;
    OsMemset(&MbxGatewayClientParms, 0, sizeof(EC_T_MBX_GATEWAY_CLNT_PARMS));
    MbxGatewayClientParms.dwSignature = EC_MBX_GATEWAY_CLNT_SIGNATURE;
    MbxGatewayClientParms.dwSize = sizeof(EC_T_MBX_GATEWAY_CLNT_PARMS);
    OsMemcpy(&MbxGatewayClientParms.LogParms, G_pEcLogParms, sizeof(EC_T_LOG_
↪PARMS));
    dwRes = emMbxGatewayClntInit(&MbxGatewayClientParms);
}

/* add a connection to mailbox gateway */
{
    EC_T_MBX_GATEWAY_CLNT_CONDESC MbxGatewayClientConnection;
    OsMemset(&MbxGatewayClientConnection, 0, sizeof(MbxGatewayClientConnection));
    MbxGatewayClientConnection.oAddr.sAddr = { 127, 0, 0, 1 }; /* localhost */
    MbxGatewayClientConnection.wPort = EC_MBX_GATEWAY_DEFAULT_PORT;
    dwRes = emMbxGatewayClntAddConnection(&MbxGatewayClientConnection, &
↪dwMbxGatewayConId);

    if (dwRes != EC_E_NOERROR)
    {
        EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
↪ "emMbxGatewayClntAddConnection failed! %s (Result = 0x%x)\\n",␣
↪ecatGetText(dwRes), dwRes));
```

(continues on next page)

(continued from previous page)

```
        goto Exit;
    }
}

/* read vendorId from CoE object dictionary (index 0x1018, sub index 1) of slave␣
↪1001 */
{
    EC_T_BYTE abyData[] = { 0,0,0,0 };
    EC_T_DWORD dwOutDataLen = 0;
    dwRes = emMbxGatewayClntCoeSdoUpload(dwMbxGatewayConId, 1001, 0x1018, 1,␣
↪abyData, sizeof(abyData), &dwOutDataLen, 5000, 0);

    if (dwRes != EC_E_NOERROR)
    {
        EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
↪"emMbxGatewayClntInit failed! %s (Result = 0x%x)\\n", ecatGetText(dwRes),␣
↪dwRes));
        goto Exit;
    }
    EcLogMsg(EC_LOG_LEVEL_INFO, (pEcLogContext, EC_LOG_LEVEL_INFO, "Vendor ID: 0x
↪%x\\n", EC_GETDWORD(abyData)));
}

Exit:
/* remove connection from mailbox gateway */

dwRes = emMbxGatewayClntRemoveConnection(dwMbxGatewayConId);

if (dwRes != EC_E_NOERROR)
{
    EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
↪"emMbxGatewayClntRemoveConnection failed! %s (Result = 0x%x)\\n",␣
↪ecatGetText(dwRes), dwRes));
}

/* terminate mailbox gateway*/

dwRes = emMbxGatewayClntDeinit();

if (dwRes != EC_E_NOERROR)
{
    EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
↪"emMbxGatewayClntDeinit failed! %s (Result = 0x%x)\\n", ecatGetText(dwRes),␣
↪dwRes));
}
```