acontis technologies GmbH

SOFTWARE

Link-Layer Developer Manual

Version 1.0

Edition: 2016-10-26

Content

1 Preface

The main goal of this document is to offer to a third party developer support in development of *Link-Layer* functionality. The *Link-Layer* is the part of the EC-Master providing access to the Ethernet hardware resources and is essential for proper EtherCAT communication.

1.1 Link-Layer Overview

The *Link-Layer* is defined in the following files:

- `\SDK\INC\EcLink.h`, main include file, contains basic definitions
- `\Sources\LinkLayer\<Link-LayerName>`, this folder contains all the files of the link layer, include files and sources as well
- `\Sources\LinkOsLayer\<Platform>`, this folder contains include files and sources OS-Layer implementation for given platform to use with the Link-Layer

2 Testing Setup

2.1 Overview

Link-Layer developer kit consists of two main parts:

- Link-Layer driver
- Test application.

Test application is designed to help developer to test and debug the Link-Layer driver. It shows the proper usage of the Link-Layer function as they will called by the EC-Master, i.e how to open/close the Link-Layer or how to send/receive Ethernet frames.

2.2 Dummy Link-Layer Project

Dummy Link-Layer driver consists of following files:

- `EcDeviceDummy.cpp`, implementation of Link-Layer functions, s. Link-Layer Functions
- `LinkOsLayer.cpp/LinkOsLayer.h`, implementation of OS-Layer functions for given platform in order to be used with Link-Layer driver
- `EcDeviceDummy.h`, header file for Link-Layer driver
- `EcDeviceDummy.def`, definitions of functions to be exported by Link-Layer
- `LinkOsPlatform.h`, platform specific OS-Layer definitions.

2.3 llAccept Project

Link-Layer test application consists of following files:

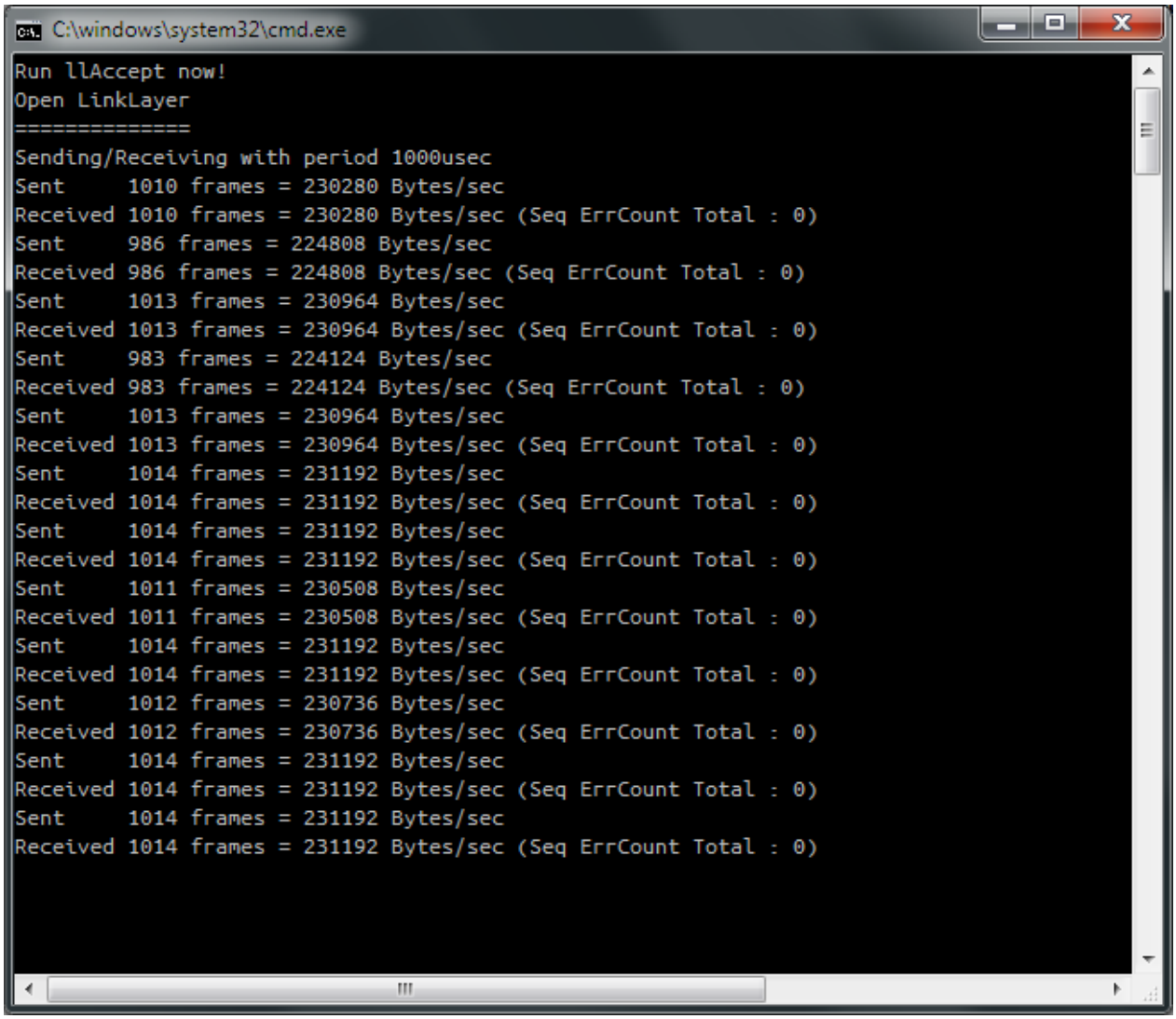
- `llAcceptMain.cpp`, contains the main() function, initializes and starts test functions
- `llAccept.cpp/llAccept.h`, contains the test function and support functions as well
- `EcOs.cpp`, implementation of OS-Layer functions
- `selectLink-Layer.cpp/selectLink-Layer.h`, parses the command line and initializes Link-Layer parameters for using in tests
- `EcTimer.cpp`, helper timer class

2.4 Link-LayerDevKit Solution

There is a demo solution for Visual Studio 2013 as an example available. The demo solution contains both dummy Link-Layer and llAccept projects.
To run tests with llAccept it is necessary to set command line parameters, i.e. for the dummy Link-Layer driver:

```
llAccept.exe -v 2 -trx -dummy 1 1
```

where '–trx' define "transmit/receive" test, '-dummy 1 1' means Link-Layer 'dummy' with parameters '1' (Link-Layer instance) and '1' (mode "polling").
llAccept takes also some more parameters, to see full list, please run llAccept.exe without command line parameters. In order to support a new Link-Layer driver by the llAccept it necessary to extend the command line parser in files selectLinkLayer.cpp and selectLinkLayer.h.
Test results can be checked in the console window, i.e. for the dummy Link-Layer driver the proper "transmit/receive" test results look like on the screenshot below:


C:\windows\system32\cmd.exe
Run llAccept now!
Open LinkLayer
==============
Sending/Receiving with period 1000usec
Sent 1010 frames = 230280 Bytes/sec
Received 1010 frames = 230280 Bytes/sec (Seq ErrCount Total : 0)
Sent 986 frames = 224808 Bytes/sec
Received 986 frames = 224808 Bytes/sec (Seq ErrCount Total : 0)
Sent 1013 frames = 230964 Bytes/sec
Received 1013 frames = 230964 Bytes/sec (Seq ErrCount Total : 0)
Sent 983 frames = 224124 Bytes/sec
Received 983 frames = 224124 Bytes/sec (Seq ErrCount Total : 0)
Sent 1013 frames = 230964 Bytes/sec
Received 1013 frames = 230964 Bytes/sec (Seq ErrCount Total : 0)
Sent 1014 frames = 231192 Bytes/sec
Received 1014 frames = 231192 Bytes/sec (Seq ErrCount Total : 0)
Sent 1014 frames = 231192 Bytes/sec
Received 1014 frames = 231192 Bytes/sec (Seq ErrCount Total : 0)
Sent 1011 frames = 230508 Bytes/sec
Received 1011 frames = 230508 Bytes/sec (Seq ErrCount Total : 0)
Sent 1014 frames = 231192 Bytes/sec
Received 1014 frames = 231192 Bytes/sec (Seq ErrCount Total : 0)
Sent 1012 frames = 230736 Bytes/sec
Received 1012 frames = 230736 Bytes/sec (Seq ErrCount Total : 0)
Sent 1014 frames = 231192 Bytes/sec
Received 1014 frames = 231192 Bytes/sec (Seq ErrCount Total : 0)
Sent 1014 frames = 231192 Bytes/sec
Received 1014 frames = 231192 Bytes/sec (Seq ErrCount Total : 0)

3 Developing a new Link-Layer

3.1 Example Link-Layer

The porting kit contains example (dummy) Link-Layers for the following platforms:

- Windows

The dummy Link-Layer stores last transmitted frame and echoes it when the receive frame function will be called.

3.2 Roadmap

The following steps have to be performed in order to develop a new Link-Layer driver:

1.) Porting of the Link-OS-Layer (files LinkOsLayer.h ans LinkOsLayer.cpp) for the given platform. For this the functions described in Link-OS-Layer Functions have to be implemented.
Important note: no other files except LinkOsLayer.h and LinkOsLayer.cpp must be changed!
2.) (If not present) Porting of the OS-Layer (files EcOsPlatform.h ans EcOs.cpp) for the given platform, please s. OS-Layer Developer Manual.
3.) Developing the Link-Layer driver
   a. Create EcDeviceXXXX.cpp and EcDeviceXXXX.h for new Ethernet device. The EcDeviceDummy.cpp and EcDeviceDummy.h can be taken as reference.
   b. Create emllXXXX.def (if needed). The emllDummy.def can be taken as reference.
   c. Append description of new device in LinkOsLayer.h. For this either extend existing LOSAL_KNOWN_XXXX_CARDS structures or define a new one if needed.

Important note: no other files except EcDeviceXXXX.cpp and EcDeviceXXXX.h (emllXXXX.def) must be changed!

4.) Use the llAccept application for running tests

4 Link-OS-Layer Functions

4.1 LinkOsOpen

Initializes required link layer.

EC_T_DWORD LinkOsOpen();

Parameters
-

Return
Error code (s. “EC-Master Stack Class B”, Appendix 4.1 and 4.2)

Comment

Initializes link layer itself, i.e. time functions or memory allocation.

4.2 LinkOsSleep

Suspends the execution of the current thread for a specified interval. Aligned to ticks it sleeps until the next dwMsec/(tick period) tick.

EC_T_VOID LinkOsSleep(EC_T_DWORD dwMsec);

Parameters
dwMsec
[in] delay interval in ms

Return
-

Comment
Has to be implemented in order to allow proper link layer, the implementation depends on platform abilities. Usually LinkOsSleep(0) is used to give back remaining CPU time, in this case delay performed by LinkOsSleep(0) may not exceed delay performed by LinkOsSleep(1).

4.3 LinkOsCreateLock

Creates a lock object of given type.

EC_T_VOID* LinkOsCreateLock (EC_T_OS_LOCK_TYPE eLockType);

Parameters
eLockType
[in] lock type

eLockType_DEFAULT,	default for current operating system lock (usually SPIN lock)
eLockType_SPIN,	lock data access between JobTask and application thread
eLockType_INTERFACE,	lock data access between application threads itself

Return
*EC_T_VOID**
Pointer to the new lock object.

Comment
An implementation of this function has to implement *eLockType_DEFAULT* lock. A lock has to be destroyed by LinkOsDeleteLock.

4.4 LinkOsDeleteLock

Destroys an existing lock object created by LinkOsCreateLock.

EC_T_VOID LinkOsDeleteLock(EC_T_VOID* pvLock);

Parameters

pvLock

[in] pointer to a lock object

Return

-

Comment

Has to be called in order to free the system resources allocated by LinkOsCreateLock.

4.5 LinkOsLock

Activates an exiting lock object.

EC_T_VOID LinkOsLock(EC_T_VOID* pvLock);

Parameters

pvLock

[in] pointer to a lock object

Return

-

Comment

To deactivate a lock object the LinkOsUnlock function has to be used. In order to

4.6 LinkOsUnlock

Deactivates an exiting lock object activated by LinkOsLock.

EC_T_VOID LinkOsUnlock(EC_T_VOID* pvLock);

Parameters

pvLock

[in] pointer to a lock object

Return

-

Comment

4.7 String Functions

String functions are used to manipulate null-terminated character strings. In most cases they are direct substituted (via #define) by standard library string functions (s. Table 1). For more details please consult the standard library documention.
If needed Link-Layer string functions can be redefined in the Link-Layer implementation.

Table 1. String functions

Link-Layer	Standard library function	Description

function		
OsStrlen	strlen	Returns length of a NULL-terminated string
OsStrcmp	strcmp	Compares two NULL-terminated strings (case sensitive)

4.8 Memory Functions

Memory functions are used to manage heap memory. In most cases they are direct substituted (via #define) by standard library functions (s. Table 2). For more details please consult the standard library documention. If needed Link-Layer memory functions can be redefined in the Link-Layer implementation.

Table 2. Memory functions

Link-Layer function	Standard library function	Description
LinkOsMalloc	malloc	Allocates requested amount of bytes in heap memory
LinkOsFree	free	Frees allocated memory block
LinkOsRealloc	realloc	Changes the size of an allocated memory block
LinkOsMemset	memset	Sets each byte in memory block to given value
LinkOsMemcmp	memcmp	Compares two memory blocks
LinkOsMemmove	memmove	Duplicates a memory block into other, blocks may overlay

4.9 LinkOsCreateEvent

Creates a binary semaphore (event) object.

EC_T_VOID* LinkOsCreateEvent(EC_T_VOID);

Parameters

-

Return

*EC_T_VOID**

Pointer to the new event object.

Comment

An event object will be used to synchronize execution threads within application. The implementation of this function depends on operating system. An event has to be destroyed by LinkOsDeleteEvent.

4.10 LinkOsDeleteEvent

Destroys a binary semaphore (event) object.

EC_T_VOID LinkOsCreateEvent(EC_T_VOID* pvEvent);

Parameters
EC_T_VOID pvEvent*
[in] Pointer to an event object.

Return
-

Comment
An event object will be used to synchronize execution threads within application. An event is created by LinkOsCreateEvent.

4.11 LinkOsSetEvent

Signals a binary semaphore (event) object.

EC_T_VOID LinkOsSetEvent(EC_T_VOID* pvEvent);

Parameters
EC_T_VOID pvEvent*
[in] Pointer to an event object.

Return
-

Comment
The actual state of an event can be checked with LinkOsWaitForEvent.

4.12 LinkOsWaitForEvent

Checks whether an event is signaled or not.

EC_T_DWORD LinkOsWaitForEvent(EC_T_VOID* pvEvent, EC_T_DWORD dwTimeout);

Parameters
EC_T_VOID pvEvent*
[in] Pointer to an event object
EC_T_DWORD dwTimeout
[in] Time-out [ms], or EC_NOWAIT or EC_WAITINFINITE

Return
Error code
EC_E_TIMEOUT – the state of an event was not changed in within dwTimeout time period
EC_E_NOERROR – function call succeeded
EC_E_ERROR – function call is not succeeded

Comment
The function awaits state change of an event to “signaled” within dwTimeout period of time. If the state of the event changes the function returns immideately, otherweise after dwTimeout. There are two special cases for dwTimeout:

- EC_NOWAIT, the function just check the state of the event and returns immideately,
- EC_WAITINIFINITE, the function will be blocked until the event changes the actual state.

4.13 LinkOsMemoryBarrier

Creates a hardware memory barrier (fence) that prevents the CPU from re-ordering read and write operations. It may also prevent the compiler from re-ordering read and write operations.

EC_T_VOID LinkOsMemoryBarrier(EC_T_VOID);

Parameters
-

Return
-

Comment

4.14 Output Functions

Output functions are used to output formatted messages. In most cases they are direct substituted (via #define) by standard library string functions (s. Table 3). For more details please consult the standard library documention.
If needed Link-Layer string functions can be redefined in the Link-Layer implementation.

Table 3. Output functions

Link-Layer function	Standard library function	Description
OsVsnprintf	vsnprintf	Sends a formatted string to standard output using the elements in the variable argument list identified by *arg* instead of additional function arguments

4.15 LinkOsAddDbgMsgHook

Registers a callback function for debug messages.

EC_T_VOID LinkOsAddDbgMsgHook(EC_PF_LINKOSDBGMSGHK pfOsDbgMsgHook);

Parameters
EC_PF_LINKOSDBGMSGHK pfOsDbgMsgHook
Callback function

Return
-

Comment

typedef EC_T_BOOL (*EC_PF_LINKOSDBGMSGHK)(const EC_T_CHAR* szFormat, EC_T_VALIST vaArgs);

Description
EC_T_CHAR* szFormat
Format string, s. printf().
…
Parameters according to szFormat, s. printf().

The registred callback function will be called each time a debug message issued.

5 Link-Layer Functions

5.1 EcLinkOpen

Initializes the link layer.

```
EC_T_DWORD EcLinkOpen(
   EC_T_VOID*                  pvLinkParms,              /* [in]  link parameters */
   EC_T_RECEIVEFRAMECALLBACK   pfReceiveFrameCallback,   /* [in]  pointer to rx callback
function */
   EC_T_LINK_NOTIFY            pfLinkNotifyCallback,     /* [in]  pointer to notification
callback function */
   EC_T_VOID*                  pvContext,                /* [in]  caller context, to be used in
callback functions */
   EC_T_VOID**                 ppvInstance               /* [out] instance handle */
   );
```

Parameters

EC_T_VOID pvLinkParms*
[in] pointer to link parameters structure
EC_T_RECEIVEFRAMECALLBACK pfReceiveFrameCallback
[in] pointer to a callback function, it will be called for each received frame, can be EC_NULL
EC_T_LINK_NOTIFY pfLinkNotifyCallback
[in] pointer to a callback function, it will be called each time the link layer sends a notification, can be EC_NULL
EC_T_VOID pvContext*
[in] caller context, to be used in callback functions, can be EC_NULL
*EC_T_VOID** ppvInstance*
[out] pointer to a instance handle

Return

Error code (s. “EC-Master Stack Class B”, Appendix 4.1 and 4.2)

Comment

Instance handle will be used in further calls. If callback functions parameters provided, they will be called

5.2 EcLinkClose

Closes the Link-Layer.

```
EC_T_DWORD EcLinkClose(
   EC_T_VOID* pvInstance        /* [in] instance handle */
);
```

Parameters

EC_T_VOID ppvInstance*
[in] instance handle

Return

Error code (s. “EC-Master Stack Class B”, Appendix 4.1 and 4.2)

Comment

Closes Link-Layer.

5.3 EcLinkSendFrame

Invokes transmission of given frame.

EC_T_DWORD EcLinkSendFrame(
EC_T_VOID* pvInstance,
EC_T_LINK_FRAMEDESC* pLinkFrameDesc /* [in] link frame descriptor */
);

Parameters

EC_T_VOID ppvInstance*
[in] instance handle
*EC_T_LINK_FRAMEDESC * pLinkFrameDesc*
[in] pointer to a frame descriptor structure

Return

Error code (s. “EC-Master Stack Class B”, Appendix 4.1 and 4.2)

Comment

```
typedef struct _EC_T_LINK_FRAMEDESC
{
    EC_T_VOID*          pvContext;          /*< link layer context */
    EC_T_BYTE*         pbyFrame;           /*< frame data buffer */
    EC_T_DWORD      dwSize;             /*< size of the frame buffer */
    EC_T_BOOL          bBuffersFollow;     /*< if EC_TRUE try to queue next frame in link layer,
                                               if EC_FALSE fill up DMA descriptors to force
immediate send */

    EC_T_DWORD*         pdwTimeStampLo;     /*< data store to store timestamp result to */
    EC_T_DWORD*         pdwTimeStampPostLo; /*< data store to store timestamp result to */
    EC_T_PVOID          pvTimeStampCtxt;    /*< context for pfnTimeStamp */
    EC_T_LINK_GETTIMESTAMP pfnTimeStamp;    /*< function if not EC_NULL called to do timestamping */
    EC_T_DWORD*         pdwLastTSResult;    /*< result code store of last time stamp call */

    EC_T_WORD           wTimestampOffset;   /*< Place in the frame where the timestamp has to be
placed */
    EC_T_WORD           wTimestampSize;     /*< Size in byte of the timestamp */
    EC_T_UINT64         qwTimestamp;        /*< Send or receive time point */

    EC_T_DWORD          dwRepeatCnt;        /*< Repeat count 0 or 1 send once, otherwise repeat the
frame */
} EC_T_LINK_FRAMEDESC;
```

5.4 EcLinkSendAndFreeFrame

Invokes transmission of given frame and frees afterwards memory allocated for the transmission frame by EcLinkAllocSendFrame.

EC_T_DWORD EcLinkSendAndFreeFrame (
EC_T_VOID* pvInstance,
EC_T_LINK_FRAMEDESC* pLinkFrameDesc /* [in] link frame descriptor */
);

Parameters

EC_T_VOID ppvInstance*
[in] instance handle
*EC_T_LINK_FRAMEDESC * pLinkFrameDesc*
[in] pointer to a frame descriptor structure

Return

Error code (s. “EC-Master Stack Class B”, Appendix 4.1 and 4.2)

Comment

typedef struct _EC_T_LINK_FRAMEDESC
{
EC_T_VOID* pvContext; /*< link layer context */

```
   EC_T_BYTE*       pbyFrame;         /*< frame data buffer */
   EC_T_DWORD       dwSize;           /*< size of the frame buffer */
   EC_T_BOOL        bBuffersFollow;   /*< if EC_TRUE try to queue next frame in link layer,
                                          if EC_FALSE fill up DMA descriptors to force immediate send */

   EC_T_DWORD*      pdwTimeStampLo;   /*< data store to store timestamp result to */
   EC_T_DWORD*      pdwTimeStampPostLo; /*< data store to store timestamp result to */
   EC_T_PVOID       pvTimeStampCtxt;  /*< context for pfnTimeStamp */
   EC_T_LINK_GETTIMESTAMP pfnTimeStamp;  /*< function if not EC_NULL called to do timestamping */
   EC_T_DWORD*      pdwLastTSResult;  /*< result code store of last time stamp call */

   EC_T_WORD        wTimestampOffset; /*< Place in the frame where the timestamp has to be placed */
   EC_T_WORD        wTimestampSize;   /*< Size in byte of the timestamp */
   EC_T_UINT64      qwTimestamp;      /*< Send or receive time point */

   EC_T_DWORD       dwRepeatCnt;      /*< Repeat count 0 or 1 send once, otherwise repeat the frame */
} EC_T_LINK_FRAMEDESC;
```

5.5 EcLinkAllocSendFrame

Allocate a frame buffer used for send.

```
EC_T_DWORD EcLinkAllocSendFrame (
   EC_T_VOID*                  pvInstance,
   EC_T_LINK_FRAMEDESC*        pLinkFrameDesc      /* [in]  link frame descriptor */
   EC_T_DWORD                  dwSize              /* [in]  size of the frame to allocate */
);
```

Parameters

EC_T_VOID ppvInstance*
[in] instance handle
EC_T_LINK_FRAMEDESC pLinkFrameDesc*
[in] pointer to a frame descriptor structure
EC_T_DWORD pLinkFrameDesc
[in] size of the frame to allocate

Return

Error code (s. “EC-Master Stack Class B”, Appendix 4.1 and 4.2)

Comment

```
typedef struct _EC_T_LINK_FRAMEDESC
{
   EC_T_VOID*       pvContext;        /*< link layer context */
   EC_T_BYTE*       pbyFrame;         /*< frame data buffer */
   EC_T_DWORD       dwSize;           /*< size of the frame buffer */
   EC_T_BOOL        bBuffersFollow;   /*< if EC_TRUE try to queue next frame in link layer,
                                          if EC_FALSE fill up DMA descriptors to force immediate send */

   EC_T_DWORD*      pdwTimeStampLo;   /*< data store to store timestamp result to */
   EC_T_DWORD*      pdwTimeStampPostLo; /*< data store to store timestamp result to */
   EC_T_PVOID       pvTimeStampCtxt;  /*< context for pfnTimeStamp */
   EC_T_LINK_GETTIMESTAMP pfnTimeStamp;  /*< function if not EC_NULL called to do timestamping */
   EC_T_DWORD*      pdwLastTSResult;  /*< result code store of last time stamp call */

   EC_T_WORD        wTimestampOffset; /*< Place in the frame where the timestamp has to be placed */
   EC_T_WORD        wTimestampSize;   /*< Size in byte of the timestamp */
   EC_T_UINT64      qwTimestamp;      /*< Send or receive time point */

   EC_T_DWORD       dwRepeatCnt;      /*< Repeat count 0 or 1 send once, otherwise repeat the frame */
} EC_T_LINK_FRAMEDESC;
```

If the link layer doesn't support frame allocation, this function must return EC_E_NOTSUPPORTED. To free the allocated memory EcLinkFreeSendFrame is used.

5.6 EcLinkFreeSendFrame

Frees memory allocated for the transmit frame buffer.

```
EC_T_DWORD EcLinkFreeSendFrame (
   EC_T_VOID*                  pvInstance,
   EC_T_LINK_FRAMEDESC*        pLinkFrameDesc        /* [in]  link frame descriptor */
);
```

Parameters

EC_T_VOID ppvInstance*
[in] instance handle
EC_T_LINK_FRAMEDESC pLinkFrameDesc*
[in] pointer to a frame descriptor structure

Return

Error code (s. “EC-Master Stack Class B”, Appendix 4.1 and 4.2)

Comment

```
typedef struct _EC_T_LINK_FRAMEDESC
{
   EC_T_VOID*        pvContext;        /*< link layer context */
   EC_T_BYTE*        pbyFrame;         /*< frame data buffer */
   EC_T_DWORD        dwSize;           /*< size of the frame buffer */
   EC_T_BOOL         bBuffersFollow;   /*< if EC_TRUE try to queue next frame in link layer,
                                           if EC_FALSE fill up DMA descriptors to force immediate send */

   EC_T_DWORD*       pdwTimeStampLo;   /*< data store to store timestamp result to */
   EC_T_DWORD*       pdwTimeStampPostLo; /*< data store to store timestamp result to */
   EC_T_PVOID        pvTimeStampCtxt;  /*< context for pfnTimeStamp */
   EC_T_LINK_GETTIMESTAMP pfnTimeStamp;  /*< function if not EC_NULL called to do timestamping */
   EC_T_DWORD*       pdwLastTSResult;  /*< result code store of last time stamp call */

   EC_T_WORD         wTimestampOffset; /*< Place in the frame where the timestamp has to be placed */
   EC_T_WORD         wTimestampSize;   /*< Size in byte of the timestamp */
   EC_T_UINT64       qwTimestamp;      /*< Send or receive time point */

   EC_T_DWORD        dwRepeatCnt;      /*< Repeat count 0 or 1 send once, otherwise repeat the frame */
} EC_T_LINK_FRAMEDESC;
```

5.7 EcLinkRecvFrame

Get a received frame from Ethernet adapter.

```
EC_T_DWORD EcLinkRecvFrame (
   EC_T_VOID*                  pvInstance,
   EC_T_LINK_FRAMEDESC*        pLinkFrameDesc        /* [in]  link frame descriptor */
);
```

Parameters

EC_T_VOID ppvInstance*
[in] instance handle
EC_T_LINK_FRAMEDESC pLinkFrameDesc*
[in] pointer to a frame descriptor structure

Return

Error code (s. “EC-Master Stack Class B”, Appendix 4.1 and 4.2)

Comment

```
typedef struct _EC_T_LINK_FRAMEDESC
{
   EC_T_VOID*        pvContext;         /*< link layer context */
   EC_T_BYTE*        pbyFrame;          /*< frame data buffer */
   EC_T_DWORD        dwSize;            /*< size of the frame buffer */
   EC_T_BOOL         bBuffersFollow;    /*< if EC_TRUE try to queue next frame in link layer,
                                            if EC_FALSE fill up DMA descriptors to force immediate send */

   EC_T_DWORD*       pdwTimeStampLo;    /*< data store to store timestamp result to */
   EC_T_DWORD*       pdwTimeStampPostLo; /*< data store to store timestamp result to */
   EC_T_PVOID        pvTimeStampCtxt;   /*< context for pfnTimeStamp */
   EC_T_LINK_GETTIMESTAMP pfnTimeStamp;   /*< function if not EC_NULL called to do timestamping */
   EC_T_DWORD*       pdwLastTSResult;   /*< result code store of last time stamp call */

   EC_T_WORD         wTimestampOffset;  /*< Place in the frame where the timestamp has to be placed */
   EC_T_WORD         wTimestampSize;    /*< Size in byte of the timestamp */
   EC_T_UINT64       qwTimestamp;       /*< Send or receive time point */

   EC_T_DWORD        dwRepeatCnt;       /*< Repeat count 0 or 1 send once, otherwise repeat the frame */
} EC_T_LINK_FRAMEDESC;
```

If **pLinkFrameDesc-> dwSize** is equal 0 there is no data received (the adapter receive buffer is empty).

5.8 EcLinkFreeRecvFrame

Signals to the Ethernet adapter to reuse or to free the receive buffer allocated for given frame.

```
EC_T_DWORD EcLinkRecvFrame (
   EC_T_VOID*                pvInstance,
   EC_T_LINK_FRAMEDESC*      pLinkFrameDesc        /* [in]  link frame descriptor */
);
```

Parameters

EC_T_VOID ppvInstance*
[in] instance handle
EC_T_LINK_FRAMEDESC pLinkFrameDesc*
[in] pointer to a frame descriptor structure

Return

Error code (s. “EC-Master Stack Class B”, Appendix 4.1 and 4.2)

Comment

```
typedef struct _EC_T_LINK_FRAMEDESC
{
   EC_T_VOID*        pvContext;         /*< link layer context */
   EC_T_BYTE*        pbyFrame;          /*< frame data buffer */
   EC_T_DWORD        dwSize;            /*< size of the frame buffer */
   EC_T_BOOL         bBuffersFollow;    /*< if EC_TRUE try to queue next frame in link layer,
                                            if EC_FALSE fill up DMA descriptors to force immediate send */

   EC_T_DWORD*       pdwTimeStampLo;    /*< data store to store timestamp result to */
   EC_T_DWORD*       pdwTimeStampPostLo; /*< data store to store timestamp result to */
   EC_T_PVOID        pvTimeStampCtxt;   /*< context for pfnTimeStamp */
   EC_T_LINK_GETTIMESTAMP pfnTimeStamp;   /*< function if not EC_NULL called to do timestamping */
   EC_T_DWORD*       pdwLastTSResult;   /*< result code store of last time stamp call */

   EC_T_WORD         wTimestampOffset;  /*< Place in the frame where the timestamp has to be placed */
   EC_T_WORD         wTimestampSize;    /*< Size in byte of the timestamp */
   EC_T_UINT64       qwTimestamp;       /*< Send or receive time point */
```

```
   EC_T_DWORD        dwRepeatCnt;       /*< Repeat count 0 or 1 send once, otherwise repeat the frame */
} EC_T_LINK_FRAMEDESC;
```

5.9 EcLinkGetEthernetAddress

Returns the MAC address of the Ethernet adapter.

```
EC_T_DWORD EcLinkRecvFrame (
   EC_T_VOID*          pvInstance,
   EC_T_BYTE*          pbyEthernetAddress   /* [out]  Ethernet MAC address */
);
```

Parameters

EC_T_VOID ppvInstance*
[in] instance handle
EC_T_BYTE pbyEthernetAddress*
[out] pointer to a MAC address buffer

Return

Error code (s. “EC-Master Stack Class B”, Appendix 4.1 and 4.2)

Comment

The output buffer pointed by pbyEthernetAddress has to be at least 6 bytes long.

5.10 EcLinkGetStatus

Returns the actual link status.

```
EC_T_LINKSTATUS EcLinkGetStatus(EC_T_VOID* pvInstance);
```

Parameters

EC_T_VOID ppvInstance*
[in] instance handle

Return

eLinkStatus_UNDEFINED = 0,
eLinkStatus_OK,
eLinkStatus_DISCONNECTED,
eLinkStatus_HALFDUPLEX,

Comment

5.11 EcLinkGetSpeed

Returns the actual link speed.

```
EC_T_LINKSTATUS EcLinkGetSpeed(EC_T_VOID* pvInstance);
```

Parameters

EC_T_VOID ppvInstance*
[in] instance handle

Return

The actual connection speed in bits/s (10, 100 or 1000) or error code if function call fails.

Comment

5.12 EcLinkGetMode

Returns the actual link mode.

EC_T_LINKMODE EcLinkGetMode(EC_T_VOID* pvInstance);

Parameters

EC_T_VOID ppvInstance*
[in] instance handle

Return

The actual link mode, possible values are:

EcLinkMode_UNDEFINED = 0,
EcLinkMode_INTERRUPT = 1,
EcLinkMode_POLLING = 2,

Comment