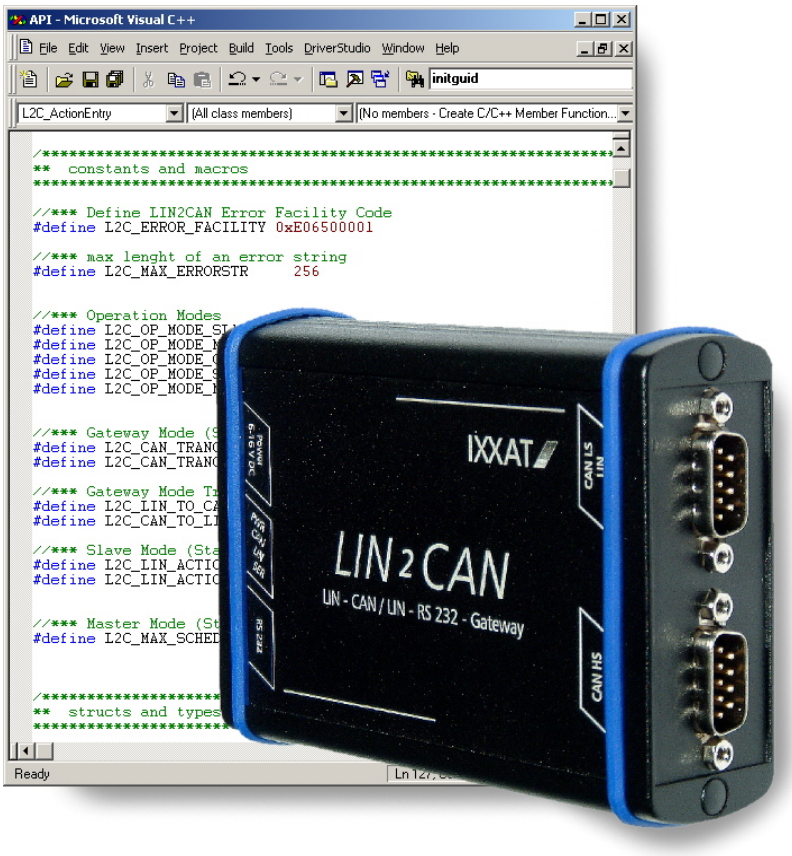
Ixxat®
BY HMS NETWORKS

LIN2CAN API (L2CAPI)

MANUAL
ENGLISH



HMS Connecting Devices™

HMS Technology Center Ravensburg GmbH
Helmut-Vetter-Straße 2
88213 Ravensburg
Germany

Tel.: +49 751 56146-0
Fax: +49 751 56146-29
Internet: www.hms-networks.de
E-Mail: info-ravensburg@hms-networks.de

Support

For problems or support with this product or other HMS products please request support at www.ixxat.com/support.

Further international support contacts can be found on our webpage www.ixxat.com

Copyright

Duplication (copying, printing, microfilm or other forms) and the electronic distribution of this document is only allowed with explicit permission of HMS Technology Center Ravensburg GmbH. HMS Technology Center Ravensburg GmbH reserves the right to change technical data without prior announcement. The general business conditions and the regulations of the license agreement do apply. All rights are reserved.

Registered trademarks

All trademarks mentioned in this document and where applicable third party registered are absolutely subject to the conditions of each valid label right and the rights of particular registered proprietor. The absence of identification of a trademark does not automatically mean that it is not protected by trademark law.

Document number: 4.01.0130.20100
Version: 1.7

1 Introduction

The LIN2CAN Gateway API (in short: L2CAPI) serves as an interface for the configuration and controlling of the LIN2CAN Gateway hardware and for the access to the LIN message traffic. All functions of the LIN2CAN device can be used in own programs with the LIN2CAN gateway API. Specific test applications or PC based network simulations can be realized. (For further information read the LIN2CAN Gateway manual.) The LIN2CAN device with the L2CAPI can be used as a developer platform. The functions are divided into function groups according to type of function or operation mode.

2 LIN-CAN-RS232 Gateway API (L2CAPI)

2.1 Basic structure

The structure of the API is similar to that of a COM object. The API is sub-divided into interfaces and has a reference counter used for releasing the API. If an interface is requested, the counter is increased by one. If an interface is no longer required, it must be released with `Release`. If the counter reaches 0, the L2CAPI object is deleted. Each interface is derived from the interface `IUnknown`. This interface contains the methods `QueryInterface`, `AddRef` and `Release`.

Every method of the API (except `AddRef` and `Release`) returns an `HRESULT` value as an error code. `S_OK (0)` stands for success.

2.2 Creating the API object

The following files are required to be able to work with the API:

- L2CAPI.lib
- L2CAPI.dll
- L2CAPI.hpp

The LIB file must be integrated in the program to be created. It loads the DLL file when the program is started. The methods of the API are defined in the HPP file.

In addition, the line `#define INITGUID` must be contained in a CPP file of the project. This serves to provide memory space for the GUIDs (Global Unique Identifiers). Every interface of the API has one of these 128-bit IDs for unambiguous definition.

First the API object must be created with `L2CAPI_CreateInstance()`. The hand-off parameters are the ID for the required interface and a pointer for the required interface. The call is as follows:

```
HRESULT hResult = E_FAIL;
IUnknown* pIUnknown = NULL;
hResult = L2CAPI_CreateInstance(IID_IUnknown, (void**)&pIUnknown);
```

If successful, every other interface of the API can now be called with this interface pointer. If unsuccessful, the error code can be converted into a readable string by calling `L2CAPI_FormatError()`. If the interface is no longer required, it is released again by the following call.

```
if(pIUnknown != NULL)
{
  pIUnknown->Release();
  pIUnknown = NULL;
}
```

A project example (L2CAPI_CONSOLE_APP) and a project template (L2CAPI_App- Template) is included for the easier beginning to implement an own application.

2.3 Overview of interfaces

The API implements the following interfaces:

- **ILIN2CAN_API_Ctrl**
  Serves to connect the API with the device or to disconnect the connection and query the connection status.
- **ILIN2CAN_GeneralConfig2**
  Serves to query or make the basic settings of the device such as name, LIN baud rate and startup operation.
- **ILIN2CAN_GatewayConfig2**
  Serves to make the settings that are required for the gateway mode. These include the CAN baud rate and the CAN transceiver mode and the translation table for translating LIN to CAN objects and vice versa.
- **ILIN2CAN_SlaveConfig**
  Defines the response to a LIN identifier.
- **ILIN2CAN_MasterConfig2**
  Defines the schedule table for the master operation mode.
- **ILIN2CAN_LINMsgQueue2**
  Serves to read out messages from the LIN message queue and query their status.
- **ILIN2CAN_Master_Slave2**
  With this interface it is possible to place LIN IDs and LIN objects on the LIN bus by PC call and to connect a wakeup signal to the BUS.

The methods of the interfaces can only be called if the device is either in L2C_OP_MODE_SLAVE_PC or L2C_OP_MODE_MASTER_PC mode!

2.4 Interface IUnknown

The interface provides the following methods:

- **HRESULT QueryInterface([in] REFIID riid, [out] LPVOID *ppvObj)**

  With QueryInterface, one of the interfaces listed in 2.3 can be queried. The call is the same as for L2CAPI_CreateInstance, the only difference being that here no new instance of the class is created, but a pointer of the required interfaces is returned. By calling QueryInterface, the reference counter is automatically increased by one.

- **ULONG AddRef()**

  AddRef increases the reference counter of the object by one. The return value is the new status of the reference counter. This method must be called if an interface pointer itself is cloned.

- **ULONG Release()**

  With Release, an interface that is no longer required is released again. The new reference counter status is returned.

2.5 Interface ILIN2CAN_API_Ctrl

This interface provides the following methods:

- **Connect([in] PCHAR pszName, [in] PCHAR pszMode)**

  With Connect the API is connected to the device by the stated interface. The name of the interface to be used is given in pszName e.g. “COM1” or “COM2”. In pszMode, “baud=115200 parity=N data=8 stop=1” must always be transmitted. If the connection was made successfully, S_OK is returned, otherwise an error code that can be converted into a readable string with L2CAPI_FormatError().

- **Disconnect()**

  With Disconnect the connection to the device is disconnected again.

- **IsConnected()**

  IsConnected is used to query whether the device is still connected. This function checks the status of the DSR line. In this way disconnection of the device can also be detected during operation.

2.6 Interface ILIN2CAN_GeneralConfig2

This interface implements the following methods:

- **GetDeviceName([out] PSTRING psDeviceName)**
- **SetDeviceName([in] PSTRING psDeviceName)**

  Get/SetDeviceName serves to give the device a name. 16 characters are permitted including "\0".
- **GetVersionInformation([out] PL2C_VersionInformation pVersionInfo);**

  GetVersionInformation queries the version information of the device. The structure is as follows:

```
struct
{
  char strFirmwareVersion[8];
  char strHardwareVersion[8];
}L2C_VersionInformation, *PL2C_VersionInformation;
```

  Each string is terminated with 0.
- **GetLINBaudrate([out] PWORD pwBaudrate);**
- **SetLINBaudrate([in] WORD wBaudrate);**

  Get/SetLINBaudrate defines the baud rate of the LIN bus. The following values are possible: 2400, 9600 and 19200 bauds.
- **GetOperationMode([out] PL2C_OperationMode pnOperationMode);**
- **SetOperationMode([in] L2C_OperationMode nOperationMode);**

  Define the current operation mode of the device. The following modes are possible:
  - L2C_OP_MODE_SLAVE_PC
  - L2C_OP_MODE_MASTER_PC
- **GetStartUpOpMode([out] PL2C_OperationMode pnOperationMode);**
- **SetStartUpOpMode ([in] L2C_OperationMode nOperationMode);**

  These functions define the startup behavior of the device if the serial cable is not connected to the PC. The following startup operation modes are permitted:
  - L2C_OP_MODE_SLAVE
  - L2C_OP_MODE_MASTER
  - L2C_OP_MODE_GATEWAY_SLAVE
  - L2C_OP_MODE_GATEWAY_MASTER
- **GetStandbyTime([out] PWORD pwStandbyTime);**
- **SetStandbyTime([in] WORD wStandbyTime);**

  Here it is possible to define after how much time (in seconds) without bus traffic the device switches to standby mode (LIN and CAN). If the device is connected to the PC, it always remains "awake". If the value 0xFFFF is transmitted, the device never goes into standby mode.

- **GetSystemTime([out] PDWORD pdwSystemTime);**
- **SetSystemTime([in] DWORD dwSystemTime);**

  These functions serve to set and query the system time of the device. The system time is a counter value in millisecond steps that is used for the timestamp of the LIN messages of the message queue.
- **GetLinMessageConfig ([out] PL2C_MessageConfig pMsgConfig);**
- **SetLinMessageConfig ([in] PL2C_MessageConfig pMsgConfig);**

  These functions set or query the length and CRC calculation mode of the LIN message packages. The structure is as follows:

```
struct
{
  BYTE                bMsgLen[64];
  L2C_MessageCrcType nMsgCrcType[64];
}L2C_MessageConfig, *PL2C_MessageConfig;
```

- **LoadSettingsFromFlash();**

  This reads out the setting from the flash. Settings that are not saved are lost.
- **SaveSettingsToFlash;**

  With this the current settings are stored non-volatile in the flash. This applies to the settings of GeneralConfig without system time, GatewayConfig, SlaveConfig and MasterConfig.

2.7 Interface ILIN2CAN_GatewayConfig2

Here the following settings can be made:

- **GetCANConfig([out] PL2C_CAN_Config pCANConfig);**
- **SetCANConfig([in] PL2C_CAN_Config pCANConfig);**

  This defines the setting of the CAN controller. The following structure is used:

```
struct
{
 DWORD dwBaudrate;
 L2C_CANTransceiverMode nTransceiverMode;
}L2C_CANConfig, *PL2C_CANConfig;
```

  The following baud rates in kilobauds are permitted:

  - 10
  - 20
  - 50
  - 100
  - 125
  - 250
  - 500
  - 1000

  The CAN transceiver can be set to one of the following modes:

  - L2C_CAN_TRANSCEIVER_LOWSPEED
  - L2C_CAN_TRANSCEIVER_HIGHSPEED

- **GetGatewayTranslation([out] PL2C_TranslationConfig2 pTranslationConfig2)**
- **SetGatewayTranslation([in] PL2C_TranslationConfig2 pTranslationConfig2)**

  These two functions are used for the configuration of the translation table from LIN to CAN and vice versa. A CAN ID can be stored here for each LIN ID. The LIN ID corresponds to the index of IdentifierTranslation. In addition it is possible to send a LIN error on the CAN bus, to store a CAN ID that activates the gateway (data bytes of the CAN message not equal to 0) or deactivated (data bytes equal to 0) and to store a CAN ID that switches the active schedule table in gateway master mode (data byte 0 specifies the number of the schedule table). The following structure is used:

```
struct
{
  L2C_IdentifierTranslation stLIN_IdentifierTranslation[64];
  L2C_IdentifierTranslation stLIN_ErrorTranslation;
  L2C_IdentifierTranslation stGatewayActivation;
  L2C_IdentifierTranslation stScheduleSwitching;
}L2C_TranslationConfig2, *PL2C_TranslationConfig2;
```

Each of the 66 entries has the following structure:

```
struct
{
  BOOL bValid;
  L2C_TranslationDirection nTranslationDir;
  BOOL b29BitIdentifier;
  DWORD dwCANID;
}L2C_IdentifierTranslation, *PL2C_IdentifierTranslation;
```

With bValid it is possible to define whether the entry is valid (TRUE for valid).

nTranslationDir defines the direction of the translation:

- L2C_LIN_TO_CAN_TRANSLATION
- L2C_CAN_TO_LIN_TRANSLATION

b29BitIdentifier defines whether it is a 29Bit CAN ID (TRUE for 29bit).

dwCANID defines the CAN identifier. Make sure, that a CAN ID is not assigned twice.

2.8 Interface ILIN2CAN_SlaveConfig

This interface serves to configure the action table. The following methods are provided:

- **GetLINIDAction([in/out] PL2C_ActionEntry pActionEntry);**
- **SetLINIDAction([in] PL2C_ActionEntry pActionEntry);**

  The methods serve to query or set an entry of the action table. The following structure is used:

```
struct
{
 BYTE bLINID;
 L2C_LINIDAction nAction;
 BYTE pbAddInfo[64];
 BYTE bAddInfoLen;
}L2C_ActionEntry, *PL2C_ActionEntry;
```

  bLINID is always an [in] parameter, which defines for which LIN ID the action is set or queried.

  nAction defines the action to be executed. The following settings are valid:

  - L2C_CAN_ACTION_IGNORE
  - L2C_CAN_ACTION_SEND_LINDATA

  If the action is set to Ignore, no AddInfo data may be sent.

  If Send_LINData is selected, the data to be sent must be present in AddInfo with suitable length according to the LIN ID. The length is transmitted in bAddInfoLen.

2.9 Interface ILIN2CAN_MasterConfig2

The Master Schedule table can be edited here. The following methods are available for this:

- **GetScheduleTable([in] BYTE bNumberScheduleTable, [out] PL2C_ScheduleEntry pScheduleTable, [out] PBYTE pbNumberEntries);**
- **SetScheduleTable([in] BYTE bNumberScheduleTable, [in] PL2C_ScheduleEntry pScheduleTable ,[in] BYTE bNumberEntries);**

  The schedule table to read or to write is specified with bNumberScheduleTable. The number of supported schedule tables can be retrieved by a call of GetScheduleTableCount.

  Up to 64 entries can be written in the schedule table. The following structure is used, which is transferred in a field. When fetching, the field must have 64 entries (L2C_MAX_SCHEDULETABLE_SIZE). pbNumberEntries gives the number of entries that were present. When setting, the number of entries must correspond to the size of the field and there must not be more than 64 entries.

```
struct
{
 BYTE bLINID;
 BYTE bWaitingTime;
} L2C_ScheduleEntry;
```

- **GetScheduleTableCount([out] PBYTE pbScheduleTableCount);**

  With this method the number of schedule tables supported by the device can be queried.

2.10 Interface ILIN2CAN_LINMsgQueue2

This interface serves to access the LIN message queue. The following functions are provided:

- **GetLINMsgQueueStatus([out] L2C_QueueStatus pQueueStatus);**

  The status of the LIN queue can be queried here.

```
struct
{
  BOOL bOverrun;
  BYTE bQueueSize;
  BYTE bUsedEntries;
}L2C_QueueStatus, *PL2C_QueueStatus;
```

  If an overrun has occurred in the LIN message queue, bOverrun is set to TRUE. The queue size is returned in bQueueSize and the currently used entries in bUsesEntries.

- **ResetLINMsgQueueOverrun();**

  The function must be called to reset the overrun status of the queue.

- **GetLINMsgQueueEntry([out] PL2C_LINMessage pLINMessage);**

  The function must be called to read out a message. The following structure is filled:

```
struct
{
 DWORD dwTimeStamp;
 BYTE bLINID;
 BYTE pbData[8];
 BYTE bDataLen;
 L2C_MessageCrcType nMsgCrcType;
}L2C_LINMessage2, *PL2C_LINMessage2;
```

  dwTimeStamp contains the receiving time of the LIN object in [ms]. The timestamp is based on the system time of the device, which overruns every 48 days.

  bLINID contains the received LIN ID or 0x40 if it is an error code.

  pbData together with bDataLen contain the LIN data and the length if it is a valid LIN message.

  nMsgCrcType contains the CRC type with which the LIN message was transmitted on the LIN network.

  If a LIN error has occurred, this is signaled by the ID 0x40. Any LIN ID received is then found in data byte 0 (otherwise 0xFF) and the error code in data byte 1.

2.10.1 Error overview

data byte 1	data byte 2	Description
FF	01	Received data is not a valid sync field.
FF	02	Received data is not a valid ID field.
FF	03	Only sync break detected before timeout.
FF	04	Only sync break and sync field detected before timeout.
LIN-ID	05	Only sync break, sync field and ID field detected before timeout. **(Slave does not respond!)**
LIN-ID	06	Only sync break, sync field, ID field and at least one data byte detected before timeout.
LIN-ID	07	Calculated checksum does not match the received checksum. If the LIN2CAN GW is operating in LIN Spec. 2.0 mode, this error may occur due **data collision or invalid CRC mode setting**.
LIN-ID	08	Received data does not match the sent data. The error may occur due data collision in LIN Spec. 2.0 mode.
FF	09	UART error, received byte does not match UART standard (e.g. StopBit missing).
FF	0A	Dominant level transmitted on the LIN BUS **(Wakeup signal)**
FF	0B	Dominant level received on the LIN BUS, but it is too short to be a sync-break. (Note: This error code is only available in slave operation mode after at least one valid sync break was received. If another error code was received before, this error code has no relevance.)

2.11 Interface ILIN2CAN_Master_Slave2

This interface enables a master emulation by PC. The following functions are provided:

- **SendLINID([in] BYTE bLINID);**

  The function sends the specified LIN ID on the bus. The LIN ID must be between 0x00 and 0x3F.

- **SendLINIDandData([in] PL2C_LINMessage2 pLINMessage2);**

  The function sends a complete LIN object to the bus. The structure is described in 2.10 GetLINMsgQueueEntry. The timestamp has no significance for the transmission.

- **SendWakeup([in] BYTE bSignalDuration);**

  The function connects a dominant level to the LIN bus according to the specified time. The duration bSignalDuration is given in [ms] between 1 and 255.

2.12 Overview: Permitted API calls / operation mode

Function \ Operation mode	Slave with PC	Master with PC	Slave Stand Alone	Master Stand Alone	Gateway Slave	Gateway Slave
GetDeviceName	x	x	-	-	-	-
SetDeviceName	x	x	-	-	-	-
GetVersionInformation	x	x	-	-	-	-
GetLINBaudrate	x	x	-	-	-	-
SetLINBaudrate	x	x	-	-	-	-
GetOperationMode	x	x	x	x	x	x
SetOperationMode	x	x	-	-	-	-
GetStartUpOpMode	x	x	-	-	-	-
SetStartUpOpMode	x	x	-	-	-	-
GetStandbyTime	x	x	-	-	-	-
SetStandbyTime	x	x	-	-	-	-
GetSystemTime	x	x	-	-	-	-
SetSystemTime	x	x	-	-	-	-
GetLinMessageConfig	x	x	-	-	-	-
SetLinMessageConfig	x	x	-	-	-	-
LoadSettingsFromFlash	x	x	-	-	-	-
SaveSettingsToFlash	x	x	-	-	-	-
GetCANConfig	x	x	-	-	-	-
SetCANConfig	x	x	-	-	-	-
GetGatewayTranslation	x	x	-	-	-	-
SetGatewayTranslation	x	x	-	-	-	-
GetLINIDAction	x	x	-	-	-	-
SetLINIDAction	x	x	-	-	-	-
GetScheduleTable	x	x	-	-	-	-
SetScheduleTable	x	x	-	-	-	-
GetScheduleTableCount	x	x	-	-	-	-
GetLINMsgQueueStatus	x	x	-	-	-	-
ResetLINMsgQueueOverrun	x	x	-	-	-	-
GetLINMsgQueueEntry	x	x	-	-	-	-
SendLINID	-	x	-	-	-	-
SendLINIDandData	-	x	-	-	-	-
SendWakeup	-	x	-	-	-	-

X → Call permitted in this mode.
- → Call ***not*** permitted in this mode.

2.13 Brief overview of API

All methods return an HRESULT error code → S_OK in the event of success, otherwise a Windows error code

2.13.1 Interface: ILIN2CAN_GeneralConfig2

- ***GetDeviceName([out] PSTRING psDeviceName)***
  Name of the device: pointer to a string with max. 16 characters. The last valid character is "\0".

- ***SetDeviceName([in] PSTRING psDeviceName)***
  Name of the device: pointer to a string with max. 16 characters. The last valid character must be "\0".

- ***GetVersionInformation([out] PL2C_VersionInformation pVersionInfo);***
  struct
  {
  char strFirmwareVersion[8];
  char strHardwareVersion[8];
  }L2C_VersionInformation, *PL2C_VersionInformation;

- ***GetLINBaudrate([out] PWORD pwBaudrate);***
  Baud rates: 2400, 9600, 19200
- ***SetLINBaudrate([in] WORD wBaudrate);***

- ***GetOperationMode([out] PL2C_OperationMode pnOperationMode);***
  typedef BYTE L2C_OperationMode;
  #define L2C_OP_MODE_SLAVE_PC 1
  #define L2C_OP_MODE_MASTER_PC 2

- ***SetOperationMode([in] L2C_OperationMode nOperationMode);***

- ***GetStartUpOpMode([out] PL2C_OperationMode pnOperationMode);***
  #define L2C_OP_MODE_SLAVE 4
  #define L2C_OP_MODE_MASTER 5
  #define L2C_OP_MODE_GATEWAY_SLAVE 3
  #define L2C_OP_MODE_GATEWAY_MASTER 3

- ***SetStartUpOpMode ([in] L2C_OperationMode nOperationMode);***

- ***GetStandbyTime([out] PWORD pwStandbyTime);***
  Resolution in [s]
  0xFFFF for never
- ***SetStandbyTime([in] WORD wStandbyTime);***

- ***GetSystemTime([out] PDWORD pdwSystemTime);***
  Time in [ms]
- ***SetSystemTime([in] DWORD dwSystemTime);***

- ***GetLinMessageConfig([out] PL2C_MessageConfig pMsgConfig)***
  struct
  {
  BYTE bMsgLen[64];
  L2C_MessageCrcType nMsgCrcType[64];
  }L2C_MessageConfig, *PL2C_MessageConfig;

- ***SetLinMessageConfig([in] PL2C_MessageConfig pMsgConfig)***

- ***LoadSettingsFromFlash();***
- ***SaveSettingsToFlash;***

2.13.2 Interface: ILIN2CAN_GatewayConfig2

- ***GetCANConfig([out] PL2C_CAN_Config pCANConfig);***

  struct
  {
  DWORD dwBaudrate;
  L2C_CANTransceiverMode nTransceiverMode;
  }L2C_CANConfig;

  CAN transceiver baud rate
  10, 20, 50, 100, 125, 250, 500, 1000

  CAN transceiver modes
  #define L2C_CAN_TRANSCEIVER_LOWSPEED 0
  #define L2C_CAN_TRANSCEIVER_HIGHSPEED 1
- ***SetCANConfig([in] PL2C_CAN_Config pCANConfig);***

- ***GetGatewayTranslation([out] PL2C_TranslationConfig2 pTranslationConfig2)***

  struct
  {
  L2C_IdentifierTranslation stLIN_IdentifierTranslation[64];
  L2C_IdentifierTranslation stLIN_ErrorTranslation;
  L2C_IdentifierTranslation stGatewayActivation;
  L2C_MessageCrcType nMsgCrcType;
  }L2C_TranslationConfig2, *PL2C_TranslationConfig2;

  struct
  {
  BOOL bValid;
  L2C_TranslationDirection nTranslationDir;
  BOOL b29BitIdentifier;
  DWORD dwCANID;
  }L2C_IdentifierTranslation, *PL2C_IdentifierTranslation;

- ***SetGatewayTranslation([in] PL2C_TranslationConfig2 pTranslationConfig2)***

2.13.3 Interface: ILIN2CAN_SlaveConfig

- ***GetLINIDAction([in/out] PL2C_ActionEntry pActionEntry);***
  [in] pActionEntry->bLINID
  [out] "remaining"
  struct
  {
  BYTE bLINID;
  L2C_LINIDAction nAction;
  BYTE pbAddInfo[64];
  BYTE bAddInfoLen;
  }L2C_ActionEntry;

  //*** Slave Mode (Standalone) LIN ID Actions
  #define L2C_CAN_ACTION_IGNORE 4
  #define L2C_CAN_ACTION_SEND_LINDATA 5
- ***SetLINIDAction([in] PL2C_ActionEntry pActionEntry);***

2.13.4 Interface: ILIN2CAN_MasterConfig2

- ***GetScheduleTable([out] PL2C_ScheduleEntry pScheduleTable, [out] PBYTE pbNumberEntries);***
  struct
  {
  BYTE bLINID;
  BYTE bWaitingTime;
  } L2C_ScheduleEntry;
  pbNumberEntries out → number of entries in the schedule table

  //*** Master Mode (Standalone) Schedule Table Size
  #define L2C_MAX_SCHEDULETABLE_SIZE 64

- ***SetScheduleTable([in] PL2C_ScheduleEntry pScheduleTable, [in] BYTE bNumberEntries);***

- ***GetScheduleTableCount ([out] PBYTE pbScheduleTableCount);***

2.13.5 Interface: ILIN2CAN_LINMsgQueue2

- ***GetLINMsgQueueStatus([out] L2C_QueueStatus pQueue-Status);***

```
struct
{
  BOOL bOverrun;
  BYTE bQueueSize;
  BYTE bUsedEntries;
}L2C_QueueStatus;
```

- ***ResetLINMsgQueueOverrun();***

- ***GetLINMsgQueueEntry([out] PL2C_LINMessage2 pLIN-Message2);***

```
struct
{
  DWORD dwTimeStamp;
  BYTE bLINID;
  BYTE pbData[8];
  BYTE bDataLen;
  L2C_MessageCrcType nMsgCrcType;
}L2C_LINMessage2, *PL2C_LINMessage2;
```

If a LIN error has occurred, this is signaled by the ID 0x40. Any LIN ID received is then found in data byte 0 (otherwise 0xFF) and the error code in data byte 1.

2.13.6 Interface: ILIN2CAN_Master_Slave2

- ***SendLINID([in] BYTE bLINID);***
  bLINID→LIN ID between 0x00 and 0x3F

- ***SendLINIDandData([in] PL2C_LINMessage2 pLINMessage2);***
  pLINMessage→dwTimeStamp not used

- ***SendWakeup([in] BYTE bSignalDuration);***
  bSignalDuration→ Duration of the wakeup signals in [ms], value between 1 and 255

2.13.7 Data type definitions

```
/*****************************************************************************
**  constants and macros
*****************************************************************************/

//*** Define LIN2CAN Error Facility Code
#define L2C_ERROR_FACILITY 0xE0670000

//*** max lenght of an error string
#define L2C_MAX_ERRORSTR     256

//*** LIN Message CRC Types
#define L2C_MSG_CRC_TYPE_DEFAULT    0
#define L2C_MSG_CRC_TYPE_SPEC_1_3   1
#define L2C_MSG_CRC_TYPE_SPEC_2_0   2

//*** Operation Modes
#define L2C_OP_MODE_SLAVE_PC        1
#define L2C_OP_MODE_MASTER_PC       2
#define L2C_OP_MODE_SLAVE           4
#define L2C_OP_MODE_MASTER          5
#define L2C_OP_MODE_GATEWAY_SLAVE   3
#define L2C_OP_MODE_GATEWAY_MASTER  6

//*** Gateway Mode (Standalone) CAN Transceiver Modes
#define L2C_CAN_TRANSCEIVER_LOWSPEED  0
#define L2C_CAN_TRANSCEIVER_HIGHSPEED 1

//*** Gateway Mode Translation Direction
#define L2C_LIN_TO_CAN_TRANSLATION 0
#define L2C_CAN_TO_LIN_TRANSLATION 1

//*** Slave Mode (Standalone) LIN ID Actions
#define L2C_LIN_ACTION_IGNORE       4
#define L2C_LIN_ACTION_SEND_LINDATA 5

//*** Master Mode (Standalone) Schedule Table Size
#define L2C_MAX_SCHEDULETABLE_SIZE 64


/*****************************************************************************
**  structs and types
*****************************************************************************/

//*** LIN Message CRC type
typedef BYTE L2C_MessageCrcType, *PL2C_MessageCrcType;

//*** Operation Mode type
typedef BYTE L2C_OperationMode, *PL2C_OperationMode;

//*** Gateway Mode (Standalone) Transceiver Mode type
typedef BYTE L2C_CANTransceiverMode, *PL2C_CANTransceiverMode;

//*** Gateway Translation Mode
typedef BYTE L2C_TranslationDirection, *PL2C_TranslationDirection;

//*** Slave Mode (Standalone) Action type
typedef BYTE L2C_LINID_Action, *PL2C_LINID_Action;
```

```
//*** Version Information Structure
typedef struct
{
  char strFirmwareVersion[8]; //Firmware Version String
  char strHardwareVersion[8]; //Hardware Version String
}L2C_VersionInformation, *PL2C_VersionInformation;


//*** LIN Message Config
typedef struct
{
  BYTE bMsgLen[64];                     //LIN Data Length: 0 - 8 Bytes
  L2C_MessageCrcType nMsgCrcType[64];   //LIN CRC calculation type
}L2C_MessageConfig, *PL2C_MessageConfig;


//*** Gateway Mode (Standalone) CAN Controller Configuration Structure
typedef struct
{
  DWORD                 dwBaudrate;         //CAN Baudrate in [kBaud]
  L2C_CANTransceiverMode nTransceiverMode;  //Mode of CAN Transceiver
                                            //(High or Lowspeed)
}L2C_CANConfig, *PL2C_CAN_Config;


//*** Gateway Mode (Standalone) Identifier Translation
typedef struct
{
  BOOL bValid;                              //Translation valid? (Valid = TRUE)
  L2C_TranslationDirection nTranslationDir; //Translation Direction (LIN 2 CAN |
                                              CAN 2 LIN)
                                            //only valid for LIN Identifier
                                              Translation
  BOOL b29BitIdentifier;                    //Is CAN ID a 29 Bit Identifier
                                            //       (29bit = TRUE)
  DWORD dwCANID;                            //CAN ID (0 - 2047      [11bit])
                                            //       (0 - 536870911 [29bit])
}L2C_IdentifierTranslation, *PL2C_IdentifierTranslation;
```

```
//*** Gateway Mode (Standalone) Translation Configuration
typedef struct
{
  L2C_IdentifierTranslation stLIN_IdentifierTranslation[64]; //Field for Identifier
                                                               Translation
                                                             //(LIN ID = Index)
  L2C_IdentifierTranslation stLIN_ErrorTranslation;          //LIN Error to CAN ID
                                                               Translation
  L2C_IdentifierTranslation stGatewayActivation;             //Gateway Activation
                                                               CAN ID
  L2C_IdentifierTranslation stScheduleSwitching;             //Master Schedule
                                                               table switching
                                                               CAN ID
}L2C_TranslationConfig2, *PL2C_TranslationConfig2;


//*** Slave Mode (Standalone) LINID Action Table entry Structure
typedef struct
{
  BYTE             bLINID;         //LIN ID (0 - 63)
  L2C_LINID_Action nAction;        //Action to perform on LINID
  BYTE             pbAddInfo[64];  //Additional Info for Action
                                   //LIN Data for instance
  BYTE             bAddInfoLen;    //Length of pbAddInfo
}L2C_ActionEntry, *PL2C_ActionEntry;


//*** Master Mode (Standalone) Schedule Table entry Structure
typedef struct
{
  BYTE bLINID;        //LIN ID (0 - 63)
  BYTE bWaitingTime;  //Time to wait before sending the next LIN Identifier
}L2C_ScheduleEntry, *PL2C_ScheduleEntry;


//*** Slave Mode (with PC) Queue Status Structure
typedef struct
{
  BOOL bOverrun;     //Indicates a LIN Queue Overrun (TRUE = Overrun)
  BYTE bQueueSize;   //Size of the LIN Message Queue
  BYTE bUsedEntries; //Used Entries of the LIN Message Queue
}L2C_QueueStatus, *PL2C_QueueStatus;


//*** Master / Slave Mode (with PC) LIN Message Structure
typedef struct
{
  DWORD               dwTimeStamp;  //Timestamp of LIN Message in [ms]
  BYTE                bLINID;       //LIN ID (0 - 63) LIN Error (64)
  BYTE                pbData[8];    //LIN Data
  BYTE                bDataLen;     //Length of pbData
  L2C_MessageCrcType  nMsgCrcType;  //LIN CRC calculation type
}L2C_LINMessage2, *PL2C_LINMessage2;
```

```
//*** Error List Lookup Table Structure
typedef struct
{
  BOOL fIdValid;              //Flag if ID is valid
  BYTE bErrorCode;            //Error Code
  char strErrorMessage[512];  //Human readable error string
}L2C_LINErrorLookupTable, *PL2C_LINErrorLookupTable;


//*** Error List Lookup Table Structure
#define L2C_LIN_ERROR_LOOKUP_TABLE_ENTRY_COUNT 12
const L2C_LINErrorLookupTable
L2C_k_stLINErrorLookupTable[L2C_LIN_ERROR_LOOKUP_TABLE_ENTRY_COUNT] =
{
  {
    FALSE,
    0x00,
    ""
  },
  {
    FALSE,
    0x01,
    "Received data is not a valid sync-field."
  },
  {
    FALSE,
    0x02,
    "Received data is not a valid ID-field."
  },
  {
    FALSE,
    0x03,
    "Only sync-break detected before timeout."
  },
  {
    FALSE,
    0x04,
    "Only sync-break and sync-field detected before timeout."
  },
  {
    TRUE,
    0x05,
    "Only sync-break, sync-field and ID-field detected before timeout.
    (Slave does not respond!)"
  },
  {
    TRUE,
    0x06,
    "Only sync-break, sync-field, ID-field and at least one data byte detected
    before timeout."
  },
```

```
  {
    TRUE,
    0x07,
    "The calculated checksum does not match the received checksum. If the LIN2CAN
    GW is operating in LIN Spec. 2.0 mode, this error may occur due data collision
    or invalid CRC mode setting."
  },
  {
    TRUE,
    0x08,
    "The received data does not match the sent data. this error may occur due data
    collision in LIN Spec. 2.0 mode."
  },
  {
    FALSE,
    0x09,
    "UART error, received byte was not like UART standard (\"StopBit missing\" for
    instance)."
  },
  {
    FALSE,
    0x0A,
    "A dominant level was transmitted on the LIN BUS. (Wakeup signal)"
  },
  {
    FALSE,
    0x0B,
    "A dominant level was received on the LIN BUS, but it was to short to be a
    sync-break. (Note: This error code is only available in slave operation mode
    after at least one valid sync-break was received. If an other error code was
    received before, this error code has no relevance.)"
  }
};
```

```
/*****************************************************************************
** LIN Error (LIN ID 0x40)
**
** An Error is signaled if bLINID is set to 0x40
** The LIN ID of the error message (if available) is stored in pbData[0]
** else it is 0xFF, while the errorcode is stored in pbData[1]
**
** Error Overview!
** - 0x01 Received data is not a valid sync-field.
** - 0x02 Received data is not a valid ID-field.
** - 0x03 Only sync-break detected before timeout.
** - 0x04 Only sync-break and sync-field detected before timeout.
** - 0x05 Only sync-break, sync-field and ID-field detected before timeout.
**        (Slave does not respond!)
** - 0x06 Only sync-break, sync-field, ID-field and at least one data byte
**        detected before timeout.
** - 0x07 The calculated checksum does not match the received checksum.
**        If the LIN2CAN GW is operating in LIN Spec. 2.0 mode, this error
**        may occur due data collision or invalid CRC mode setting.
** - 0x08 The received data does not match the sent data. this error may
**        occurre due data colission in LIN Spec. 2.0 mode.
** - 0x09 UART error, received byte was not like UART standard
**        ("StopBit missing" for instance).
** - 0x0A A dominant level was transmitted on the LIN BUS. (Wakeup signal)
** - 0x0B A dominant level was received on the LIN BUS, but it was to short
**        to be a sync-break.
**        (Note: This error code is only available in slave operation
**        mode after at least one valid syncbreak was received. If an other
**        error code was received before, this error code has no relevance.)
*****************************************************************************/
```