EXXOTest® MUXDLL

Software Libraries

User Guide

Document N° 00279580-v3

ANNECY ELECTRONIQUE, designer and manufacturer of products: Exxotest, Navylec and Aircraft Electronic.
Parc Altaïs - 1 rue Callisto - F 74650 CHAVANOD - Tel : 33 (0)4 50 02 01 01 Fax : 33 (0)4 50 68 58 93
S.A.S. with Capital of 207 000€ - RC ANNECY 80 B 243 - SIRET 320 140 619 00042 - APE 2651B - N° TVA FR 37 320 140 619

SUMMARY

1. Document's aim

1.1. Aim

The aim of this document is to describe the different functions that make up libraries with access to the PCI, USB, Ethernet cards and interfaces of the EXXOTest® « Communication Networks Expertise systems» products range.

Such libraries of software functions enable interfacing with a PC (or Pocket PC) and EXXOTest® cards and interfaces

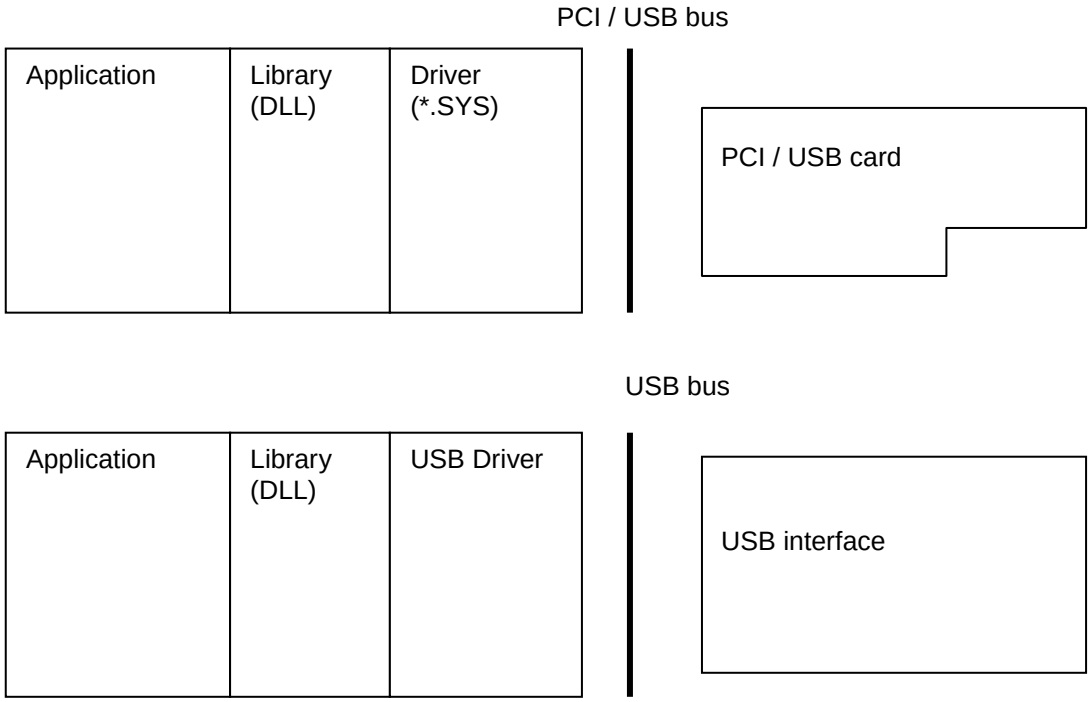
The limitations regarding the number of networks that may be connected to the card, depends on the type of card that is used. Such limitations are described in the card's guide.

Using these functions will allow the user to automatically adapt his application to his card's protocol controllers and line interfaces.

All these functions can be found within a dynamic library (DLL), which works with PCs running under 2000, XP, Vista and Seven operating systems or with Pocket PC running under Windows CE/Mobile (ARM Target).

2. Access libraries

2.1. Diagram



2.2. Presentation

The application communicates with the networks through the various functions provided by the library. These functions are identical regardless of the operating system.

The libraries allow an application to access up to 8 cards simultaneously, as well as all networks resident on them.

The libraries allow several applications to access several different cards. However, they do not allow several applications to share the same card.

2.3. Drivers

The libraries access the cards using a driver. This is the driver that established the link between the bus (USB / PCI …) and the hardware.

Three driver generations:

- **Windriver:** First driver distributed with the EXXOTest MUX cards, it is validated for Windows 2000 to XP operating systems (windrvr.sys).

- **Exxotest_v1.x:** Second driver distributed with the EXXOTest MUX cards, it is validated for Windows XP to Seven 32bits operating systems (exxusbw32.sys)

- **Exxotest_v2.x:** Driver currently distributed, it is validated for Windows XP to Seven 32 and 64bits operating systems (exxusb32.sys ou exxusb64.sys)

The driver installation is facilitated by a setup named "Exxotest Driver Kit and Utilities". This prepares the operating system to install the MUX card drivers. For more information, please consult the user manual delivered with the card to be installed.

Two exceptions exist:

- Windows CE: This operating system needs a DLL file (exxotest.dll). Installation is processed by the Setup_USBMUX.DAT file.

- Linux: under development.

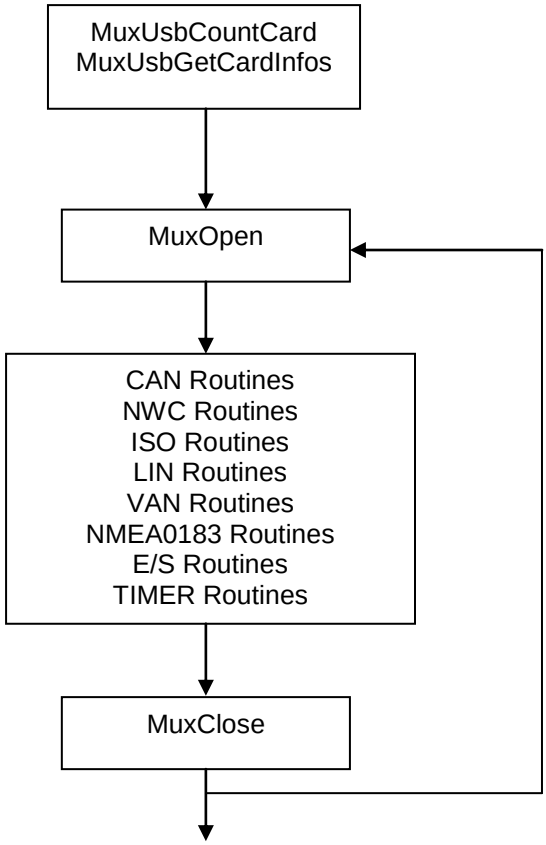
2.4. List of cards accessible by the librarie's functions

All cards / interfaces of the EXXOTest® « Communication Networks Expertise systems» products range are accessible through the software libraries, which allow an application to remain unchanged and to operate regardless of the material used.

3. Library common to all protocols

3.1. Order of call

The application must respect the sequencing according to the function calling order.



3.2. MuxCountCards: Indicates the number of cards managed by the DLLs

Prototype:
tMuxStatus MuxCountCards(unsigned long *dwCardsCount);

Description:
This function indicates the number of cards connected to the PC.

Input parameters:
None

Output parameters:
dwCardsCount : Number of cards detected

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_OPEN	Error when opening the driver

3.3. MuxPciCountCards: Indicates the number of PCI-MUX cards

Prototype :
tMuxStatus MuxPciCountCards(unsigned long *dwCardsCount);

Description :
This function indicates the number of cards on the PCI bus.

Inputs parameters:
None

Outputs parameters:
dwCardsCount : Number of PCI-MUX cards detected on the PCI bus.

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_OPEN	Error when opening the driver

3.4. MuxUsbCountCards: Indicates the number of USB-MUX cards

Prototype:
tMuxStatus MuxUsbCountCards(unsigned long *dwCardsCount);

Description:
This function indicates the number of cards on the USB bus.

Inputs parameters:
None

Outputs parameters:
dwCardsCount : Number of USB-MUX card detected on the USB bus.

Return code:

Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_OPEN	Error when opening the driver

3.5. MuxEthCountCards: Indicates the number of ETH-MUX cards

Prototype:
tMuxStatus MuxEthCountCards(unsigned long *dwCardsCount);

Description:
This function indicates the number of cards on the ETHERNET network.

None

Outputs parameters:
dwCardsCount : Number of ETH-MUX card detected on the ETHERNET network.

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_OPEN	Error when opening the driver

3.6. MuxPciGetCardInfo: Gives information on PCI-MUX cards

Prototype :
tMuxStatus MuxPciGetCardInfo(unsigned short wCard, unsigned long *dwCardBus, unsigned long *dwCardSlot, unsigned long *dwCardInfo);

Description :
This function gives information about the cards on the PCI bus.

Inputs parameters:
wCard : Index of card number to be accessed

Outputs parameters:
dwCardBus : Number of bus where the card is found
dwCardSlot : Number of slot where the card is found
dwCardInfo : Supplementary information

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_OPEN	Error when opening the driver

3.7. MuxUsbGetCardInfo: Gives information on USB-MUX cards

Prototype :
tMuxStatus MuxUsbGetCardInfo(unsigned short wCard, unsigned long *dwHubNum, unsigned long *dwPortNum, unsigned long *dwFullSpeed, unsigned long *dwDeviceAddress, unsigned long *dwCardInfo, unsigned long *dwUniqueId);

Description :
This function gives information on the cards present on the USB bus.

Inputs parameters:
wCard : Index of card number to be accessed

Outputs parameters:
dwHubNum : Number of « HUB » where the card is found
dwPortNum : Number of port where the card is found
dwFullSpeed: 1 : Indicates that the USB card operates at maximum baud rate of 12 Mbit/sec on the USB bus
dwDeviceAdress : Reserved for internal use
dwCardInfo : Supplementary information
dwUniqueId : Reserved for internal use

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_OPEN	Error when opening the driver
STATUS_ERR_TO_USB	Communication error in the USB bus

3.8. MuxEthGetCardInfo: Gives information on ETH-MUX cards

Prototype:
tMuxStatus MuxEthGetCardInfo(unsigned short wCard, unsigned char *bInfo, unsigned short *wSize);

Description :
This function gives information on the cards present on the ETHERNET network.

Example:
char *pInfo;

```
unsigned short szInfo = 0;

if (MuxEthGetCardInfo(0, NULL, &szInfo) != STATUS_OK)
{ /* Retrieval of the minimum size */
        pInfo = malloc(szInfo);
        if (MuxEthGetCardInfo (0, pInfo, &szInfo) != STATUS_OK)
        {
                printf("ERREUR”);
        }
}
```

Inputs parameters:
wCard : Index of card number to be accessed
bInfo : Pointer on the address to be filled in
wSize : Size of the buffer allocated to the filled address

Outputs parameters :
bInfo : Pointer on binary informations
wSize : Size of the valid information buffer

- The buffer returned contains information which may have fixed or variable length and are organized according to the format TLV (Type / Code, Length, Value / Data).

- Fixed size informations without data contain only one marking byte indicating the code. The “255 type” information, having a fixed length, indicates the end of information. The value of the length byte indicates the size of following data.

Code	Description
0x01	Nature of ETHERNET link (1 byte) Octet1 : 1=filaire, 2=sans fil
0x02	Software version and identifier (4 bytes) Bytes 1 and 2 : Embedded version Bytes 3 and 4 : ID of the card
0x03	Product information (32 bytes) Bytes 1 to 16 : Product part number (ASCII) Bytes 17 to 32 : Serial number (ASCII)
0x04	Network information (28 bytes) Bytes 1 to 6 : MAC target address Bytes 7 to 10 : IP target address Bytes 11 to 14 : Subnet mask target Bytes 15 to 20 : MAC source address Bytes 21 to 24 : IP source address Bytes 25 to 28 : Subnet mask source

Return code:

Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_MEMORY	Memory allocation error

3.9. MuxInit : Initialisation of the libraries' internal variables

Prototype :
tMuxStatus MuxInit(unsigned int wCard);

Description :
This function initialises the libraries' statuses and internal variables. This function is the first one to be called by the application. (after MuxLoadDLL if dynamic charge), this call is unique and situated at the start of the programme.

Inputs parameters :
wCard : Index of card number to be accessed

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_SEQUENCE	Sequence error

3.10. MuxSetEthernet: Setting Ethernet functionnality

Prototype:
tMuxStatus MuxSetEthernet (tETHState hEthState);

Description :
This function allows activating the research of MUX cards (MuxEthCountCard) connected to the Ethernet network of the PC.

Inputs parameters:
hEthState : Operating mode of the Ethernet

hEthState	ETH_Enabled: - The Ethernet cards research is enabled. ETH_Disabled: - The Ethernet cards research is

disabled.

Outputs parameters:
None

Return code:
Status: Summary of the function execution

STATUS_OK

3.11. MuxGetEthernet : Status of the Ethernet functionality

Prototype:
tMuxStatus MuxGetEthernet (tETHState *hEthState);

Description :
This function allows getting the status of the Ethernet functionality.

Inputs parameters:
hEthState: Operating mode of the Ethernet

hEthState	ETH_Enabled: - The Ethernet cards research is enabled. ETH_Disabled: - The Ethernet cards research is disabled.

Outputs parameters:
None

Return code:
Status : Summary of the function execution

STATUS_OK

3.12. MuxOpen : Opening the driver

Prototype :
tMuxStatus MuxOpen(unsigned short wCard, tMuxConfigMode *hMuxConfigMode);

Description :
This function allows initialisation of communication with the multiplexed card's driver

Inputs parameters :
wCard : Index of card number to be accessed
hMuxConfigMode : Driver's operation mode

eMuxMode	APPLI_MODE : - Reserved KERNEL_MODE : - Value to be programmed by the application DEMO_MODE : - Reserved MUXTRACE MODE: - Reserved
wBusInterface	Reserved

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_OPEN	Error when opening the software driver
STATUS_ERR_OPENKP	Error when opening the hardware driver
STATUS_ERR_MEMORY	Memory allocation error
STATUS_ERR_BANK	Repagination memory error
STATUS_ERR_SEQUENCE	Sequence error

3.13. MuxGetTimings : Returns the « opening » time of the card

Prototype:
tMuxStatus _MUXAPI MuxGetTimings (unsigned short wCard, TTimings *ptTimings, TTimingValue *tBaseCounter);

Description:
This function returns the timestamp of the last muxopen.

Inputs parameters:
wCard : Index of the card number to be accessed.
tBaseCounter:

Output parameters:
ptTimings :

3.14. MuxGetVersion : Indicates software versions

Prototype :
tMuxStatus MuxGetVersion(unsigned short wCard,unsigned long *dwVersionDll,unsigned long *dwVersionDriver,unsigned long *dwVersionKernel, tMuxHWType *eNumHWType);

Description:
This function indicates the versions of the software installed

Inputs parameters :
wCard : Index of card number to be accessed

Outputs parameters :

DwVersionDll	: Number of DLLs version
DwVersionDriver	: Number of software driver version
DwVersionKernel	: Number of hardware driver version
eNumHWType	: Type of card

PCI_MUX_DEMO	No card : demo version
PCI_MUX_C3V2L	PCI-MUX-C3V2L card
PCI_MUX_CAN	PCI-MUX-CAN card
PCI_MUX_VAN	PCI-MUX-VAN card
PCI_MUX_MultiCAN	PCI-MUX-4C2L card
USB_MUX_C3VL	USB-MUX-C3VL card
USB_MUX_4C2L	USB-MUX-4C2L card
USB_MUX_CL	USB-MUX-CL card

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error

3.15. MuxClose : Closing the driver

Prototype :
tMuxStatus MuxClose(unsigned short wCard);

Description :
This function terminates communication with the card. It is the last function called by the application.

Inputs parameters :
wCard : Index of card number to be accessed

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error

3.16. MuxGetLastErrorString: Specifying type of error

Prototype :
```
tMuxStatus  MuxGetLastErrorString(char **pString);
```

Description :
This function indicates a text with information on the content of the last error that took place

Example :
```
char *pString

if (MuxOpen(0, MODE_KERNEL) != STATUS_OK)
{ /* An error has taken place */
        MuxGetLastErrorString(&pString);
        printf("ERROR : %s",pString);
}
```

Inputs parameters :
pString : Pointer on the address

Outputs parameters :
pString : Pointer on the text chain

Return code:
Status : Summary of the function execution

STATUS_OK

3.17. MuxLoadDLL: Library's dynamic load (direct access)

Prototype :

```
tMuxStatus  MuxLoadDLL(void);
```

Description :
This function is required when the application wants to charge the functions dynamically (see chapter on inserting libraries into the project). This function is the first one to be called by the application. This call is unique and situated at the start of the programme.

Example :
```
if (MuxLoadDLL() != STATUS_OK)
{ /* An error has taken place */
        printf("ERROR library charge ");
}
```

Inputs parameters :
None

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_OPENDLL	Error when loading library functions

3.18. MuxLoadDLLClient: Library's dynamic load (Shared access)

Prototype :
```
tMuxStatus  MuxLoadDLLClient(void);
```

Description :
This function is required when the application wants to charge the functions dynamically (see chapter on inserting libraries into the project) through server. This operating mode authorizes several applications to be connected on the same card.
This function is the first one to be called by the application. This call is unique and situated at the start of the programme.

Example :
```
if (MuxLoadDLLClient () != STATUS_OK)
{ /* An error has taken place */
        printf("ERROR library load ");
}
```

Inputs parameters :
None

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_OPENDLL	Error when loading library functions
STATUS_ERR_SERVERNOTLOADED	The server is not running

3.19. MuxFreeDLL: Releasing the library's dynamic load

Prototype:
```
tMuxStatus  MuxFreeDLL(void);
```

Description:
This function is necessary when the application needs to load functions dynamically (see chapter « xxx »)
This function is the last one the application must call

Example:
```
if (MuxFreeDLL() != STATUS_OK)
{ /* An error occured */
        printf("ERREUR");
}
```

Inputs parameters:
None

Outputs:
None

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_OPENDLL	Erreur during function's call

4. CAN Library

4.1. Order of call

The application must respect the sequencing according to the functions calling order.

X : means that the function has been authorised in the present status.
X ⇒ : means that the function has been authorised in the present status and moves onto the next status.

Status	CAN_INIT	CAN_OPER	CAN_BUS	CAN_START
CanConfigOper	X ⇒			
CanConfigBus		X ⇒		
CanActivate			X ⇒	
CanDeactivate	X	X	X	X ⇒
CanSetNotification	X	X	X	
CanConfigParam		X		
canConfigClock		X		
CanConfigRangeFilter		X		
CanConfigDualFilter		X		
CanConfigStat	X	X	X	X
CanConfigTranceiverHS	X	X	X	X
CanSelectTranceiverHS	X	X	X	X
CanConfigTranceiverLS	X	X	X	X
CanSelectTranceiverLS	X	X	X	X
CanreadTranceiverLS	X	X	X	X
CanConfigPeriodic		X	X	X
CanConfigPeriodicList		X	X	X
CanCreateMsg			X	
CanSendMsg				X
CanSendMsgList				X
CanGetEvent				X
CanGetBusState				X
CanBusOn				X
CanGetStat				X
CanIsBusActive	X	X	X	X
CanReadByte	X	X	X	X
CanWriteByte	X	X	X	X
CanGetFifoRxLevels	X	X	X	X
CanClearFifoRx	X	X	X	X

4.2. CanConfigOper : Routines' operation mode

Prototype :
tMuxStatus CanConfigOper(unsigned short wCard, unsigned short wBus, tCanOper *hCanOper);

Description :
This function determines the interface mode between the application and the card.

Inputs parameters:
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
hCanOper : Type of interface with the application

eCanOperMode	Operating mode : CAN_OPER_ANA_FIFO - Analysis mode, FIFO storage CAN_OPER_ANA_BUFF - Analysis mode, Buffer storage
wFifoSize	Reserved for future use

- FIFO storage: in this mode, the events to be transferred to the application are stocked in a waiting list. These events are of the following types : end of transmission, reception, errors … When the application calls the CanGetEvent function, the first event (the oldest in time) comes off the list.
  If the waiting list is full and an event takes place, then a bit indicating loss of event is placed on the last event.

- BUFFER storage: in this mode, the events to be transferred to the application are in a unique buffer allocated during message configuration (regardless of its service). These events are either end of transmission or reception (not errors). When the application calls the CanGetEvent function, the message's communication handle allows the user to find the buffer to be read. This buffer contains the last event that has been received (the most recent in time).
  The buffer is reset with each reading. If no event takes place between two readings, the summary (EVENT_EMPTY) tells the application.

Output parameters:
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

4.3. CanConfigBus : Configuration of bus parameters

Prototype :
tMuxStatus CanConfigBus(unsigned short wCard, unsigned short wBus, tCanBus *hCanBus);

Description :
This function can configure all the parameters of the bus.

The baudrate is calculated as follows :
Baud rate (8MHz/((1+TSEG1+TSEG2)*BRP))

The sampling point is calculated as follows:
Point position = (1+TSEG1)(1+TSEG1+TSEG2)

Example : A 250 kbit/sec baudrate with an 81% sampling point
BRP=2, TSEG1=12, TSEG2=3

Inputs parameters:
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
hCanBus : Bus parameter

wBRP	Baud rate prescaler [1-64]
wTSEG1	Time before sampling point [3-16].
wTSEG2	Time after sampling point [2-8]
wSJW	Maximum value of resynchronisation jump [1-4]

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

4.4. CanConfigParam : Configuration of supplementary parameters

Prototype :

tMuxStatus CanConfigParam(unsigned short wCard, unsigned short wBus, tCanParam *hCanParam);

Description :
This function allows configuration of additional operational parameters.

For a card with a CAN PHILIPS SJA1000 protocol controller, the messages received by the controller are analysed and then placed in the corresponding buffer. This function allows the user, among other things, to parameterise the controller's acceptance filter so as to reduce the number of messages received from the network, aiming to improve the system's performance.

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
hCanParam : Auxiliary operation parameters

dwFiltIdent	Identifier of acceptance filter (0x000 by default) The acceptance filter is situated in the reception buffer. [0-0x7FF] if standard identifier [0x1FFFFFFF] if extended identifier
eTypeId	Type of identifier (CAN_ID_STD by default) CAN_ID_STD : Standard identifier (11 bits) CAN_ID_XTD : Extended identifier (29 bits)
dwFiltMask	Mask of acceptance filter (0x000 by default) [0-0x7FF] if standard identifier [0x1FFFFFFF] if extended identifier
eAckEnable	CAN_TRUE (by default) - Acknowledge generation is admitted. Messages correctly received with acknowledge request are admitted CAN_FALSE - Acknowledge generation is suppressed (spy mode)
eRxAll	CAN_TRUE (by default) - All messages received apart from those programmed are transferred to the application through the reception FIFO CAN_FALSE - Only those messages received which were programmed at reception are transferred to the application through the reception FIFO.
wSpecialModes	Reserved for future use

Note on acceptance filter: the acceptance filter allows application of a message selection filter for messages coming from the network in order to limit the charge on the PC. This filter has priority over the rest of filters.
An identifier is received if the following condition applies:
(Received identifier AND type AND Mask) = (Programmed identifier AND type AND Mask)

For example :

1. Programmed ident = xxx, mask = 000 allows reception of all identifiers
2. Programmed ident = xxx, mask = 7FF allows reception of xxx identifier only
3. Programmed ident = 001, mask = 001 allows reception of all odd identifiers

Outputs parameters :
None

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

4.5. CanConfigClock : Set up the CAN controller clock

Prototype:
tMuxStatus _MUXAPI CanConfigClock(unsigned short wCard, unsigned short wBus, tCanClockFreq eCanClock);

Description:
This function indicates to the dll the operating frequency of the associated device's CAN controller

Controller:

- SJA1000 : 16 MHZ frequency.
- TWINCAN : 40 MHZ frequency.

Note: In case this function is not called, only the card's compatible settings will be valid.

Input parameters :
wCard : Index of the card to be accessed
wBus : Index of the bus number [0-x]
eCanClock : frequency settings
CAN_CLOCK_16MHZ : 16 MHZ
CAN_CLOCK_40MHZ: 40 MHZ
CAN_CLOCK_5MHZ: 5 MHZ

Ouptut parameters:
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_WARNING	Card not supported

4.6. CanConfigRangeFilter : Setting of a range of acceptance

Prototype :
tMuxStatus _MUXAPI CanConfigRangeFilter(unsigned short wCard, unsigned short wBus, tCanRangeFilter *hCanRangeFilter);

Description:
This function allows to define an identifiers range to be sent back to the application. The range will be continued between both defined identifiers.

This function must be called after the CanConfigParam function and replaces the acceptance filter previously definied by it.

Inputs parameters:
wCard : Index of the card number to be accessed
wBus : Index of the bus number [0-x]
hCanRangeFilter : range setting

eTypeId	Identifier's type (CAN_ID_STD default) CAN_ID_STD : standard identifier(11 bits) CAN_ID_XTD : extended identifier (29 bits)
dwLowIdent	Low identifier of the range filter [0-0x7FF] if standard identifier [0xFFFF] if extended identifier
dwHighIdent	High identifier of the range filter [0-0x7FF] if standard identifier [0xFFFF] if extended identifier

Outputs parameters:
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

STATUS_WARNING	Card not supported

4.7. CanConfigDualFilter : Configuration of double acceptance filter

Prototype :
tMuxStatus CanConfigDualFilter(unsigned short wCard, unsigned short wBus, tCanDualFilter *hCanDualFilter);

Description :
This function uses CAN PHILIPPS SJA1000's protocol controller's capacity to define 2 different acceptance filters.

- For standard identifiers, the filters are placed on the whole of the identifier (11 bits)
- For extended identifiers, the filters are place on the identifier's first 2 bytes (16 bits of 29)

This function must be called after the CanConfigParam function and it replaces the acceptance filter defined by the latter.

Inputs parameters:
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
hCanDualFilter : Filter configuration

eTypeId	Type of identifier (CAN_ID_STD by default) CAN_ID_STD : Standard identifier (11 bits) CAN_ID_XTD : Extended identifier (29 bits)
wFiltIdent1	Identifier of acceptance filter number 1. [0-0x7FF] if standard identifier [0xFFFF] if extended identifier
dwFiltMask1	Mask of acceptance filter number 1 [0-0x7FF] if standard identifier [0xFFFF] if extended identifier
dFiltIdent2	Identifier of acceptance filter number 2. [0-0x7FF] if standard identifier [0xFFFF] if extended identifier
dwFiltMask2	Mask of acceptance filter number 2 [0-0x7FF] if standard identifier [0xFFFF] if extended identifier

Note on acceptance filter: The acceptance filter allows application of a message selection filter for messages coming from the networks in order to limit the charge on the PC. This filter has priority over the rest of filters.
Un identifier is received if the following condition applies:

(Received identifier AND type AND Mask) = (wFiltIdent1 AND type AND dwFiltMask1)

OR

(wFiltIdent2 AND type AND dwFiltMask2)

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

4.8. CanConfigStat : Configuration of statistics

Prototype :
tMuxStatus CanConfigStat(unsigned short wCard, unsigned short wBus, unsigned short wBusLoadTime);

Description :
This function allows configuration of the duration on which the bus charge is calculated. Duration of 0 inhibits the charge calculation.

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
wBusLoadTime : This parameter defines the duration on which the bus charge is calculated. It is expressed in ms. Value: 0 indicates that no bus charge has been transferred to the application. The bus charge is transferred in FIFO mode by means of the event EVENT_CAN_BUSLOAD and of the wBusLoad parameter (0 by default)

Outputs parameters:
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

4.9. CanConfigTransceiverHS: Configuration of signal slope

Prototype :

tMuxStatus CanConfigTransceiverHS(unsigned short wCard, unsigned short wBus, tCanSlope eCanSlope);

Description :
This function controls signal slope on CAN high and CAN low lines.

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
eCanSlope: Signal slope CANH and CAN low.
CAN_SLOPE_CONTROL : slope edge
CAN_SLOPE_HIGH_SPEED : straight edge

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

4.10. CanSelectTransceiverHS: Configuration of type of line interface

Prototype :
tMuxStatus CanSelectTransceiverHS(unsigned short wCard, unsigned short wBus, tCanBoolean eCanTxHS);

Description :
This function allows selection of type of bus line interface.
- CAN high speed
- CAN low speed – fault tolerant

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
eCanTxHS: Type of line interface.
CAN_TRUE : high speed interface (by default)
CAN_FALSE : low speed interface – fault tolerant

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_BOARD	Card not supported

4.11. CanConfigTransceiverLS: Configuration of low speed line interface

Prototype :
tMuxStatus CanConfigTransceiverLS(unsigned short wCard, unsigned short wBus, unsigned short wStandBy, unsigned short wEnable, , unsigned short wWakeUp);

Description :
This function allows the user to control the different control signals from the CAN low speed - fault tolerant line interface.

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
wStandby : Value of « stand by » pin [0-1]
wEnable : Value of « enable » pin [0-1]
wWakeUp : Value of « wake up » pin [0-1]

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_WARNING	This type of material is not supported

4.12. CanReadTransceiverLS : LS transceiver status reading

Prototype:
tMuxStatus _MUXAPI CanReadTransceiverLS(unsigned short wCard, unsigned short wBus, unsigned short *wLineState) ;

Description :
This function allows to read the logical level of INH and ERR pins of the CAN low speed – fault tolerant tranceiver.

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]

Outputs parameters:
wLineState : tranceiver's status
Bit 0 : ERR line's status.
Bit 1 : INH line's status.

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

4.13. CanConfigTerminationLS: Configuration of pull-up resistors from CAN LS networks

Prototype :
tMuxStatus CanConfigTerminationLS(unsigned short wCard, unsigned short wBus, unsigned short wValue);

Description :
This function allows the user to parameterise the different values of the pull-up resistors on the CANLS line interface. These resistor values have an influence on the network's total impedance and consequently on the transition slopes when going from recessive to dominant on CANH and CANL lines.

Note :
- A CANLS network's total impedance must always be below 1Ko
- This function is only available on the first CAN channel of USB-MUX-4C2L cases and on all CAN channels of MUX-4C4L and MUX-6C6L cases.

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
wValue : Value of the pull-up resistor
wValue=0 Approximately 15 Ko resistor (6.8 Ko for MUX-4C4L/6C6L)
wValue=1 Approximately 6 Ko resistor (2.2 Ko for MUX-4C4L/6C6L)
wValue=2 Approximately 1,3 Ko resistor (1.6 Ko for MUX-4C4L/6C6L)
wValue=3 Approximately 0,5 Ko resistor

Outputs parameters :

None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_WARNING	Not supported with this type of material

4.14. CanConfigPeriodic: Programming a periodic message

Prototype :
tMuxStatus CanConfigPeriodic(unsigned short wCard, unsigned short wBus, unsigned short wOffset, unsigned short wParam, tCanMsg *hCanMsg);

Description :
This function allows the user to start, stop and update the emission of periodic messages.

Inputs parameters:
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
wOffset : Index of periodic message [0-15]
The application is composed of 16 periodic messages. This parameter corresponds to the message index the application wants to access. Note that the 16 messages are dealt with independently.

wParam : Message parameter
Bit 0 : Indicator of periodic message in operation
- 0 : The periodic message is stopped
- 1 : The periodic message is activated (start of periodicity or update)

Bit 1 : Indicates transference of transmissions endings
It is not always necessary for an application to receive the events of transmission endings related to the emission of periodic messages.
- 0 : The transmissions endings are not transferred to the applications
- 1 : The transmissions endings are transferred to the application

hCanMsg : Parameter of message to be transmitted

wHandleMsg	Communication index For the BUFFER mode: - This index has previously been indicated by the CanCreateMsg function during the creation of the message. As for configuration, only the data (bData) and their length (wDataLen) are taken into account

For the FIFO analysis mode :
- The DLL puts the messages in series. This parameter is not used.

dwIdent	CAN message identifier [0-0x7FF] if standard identifier [0x1FFFFFFF] if extended identifier
eTypeId	Type of identifier CAN_ID_STD : Standard identifier (11 bits) CAN_ID_XTD : Extended identifier (29 bits)
dwMask	Unused
eService	CAN_SVC_TRANSMIT_DATA - Data transmission CAN_SVC_TRANSMIT_RTR - Request for distant transmission
lPeriod	Periodicity of the message in ms.
dwReserved1	Reserved for future use
dwReserved2	Reserved for future use
wDataLen	Maximum length of dada (sent or received) [0-8]
bData	Content of data (for transmitting)

Outputs parameters:
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_MSGSVC	Service not supported
STATUS_ERR_MSGEXCEED	Wrong message offset

4.15. CanConfigPeriodicList : Programming a periodic messages list

Prototype:
tMuxStatus _MUXAPI CanConfigPeriodicList(unsigned short wCard, unsigned short wBus, unsigned short wPeriodicCount, tCanPeriodicMsg *hPeriodicCanMsgList)

Description :
This function allows the start, stop and update of a periodic messages list mission.

Input parameters:
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]

wPeriodicCount: Quantity of periodic contained in the table.
hPeriodicCanMsg: Table containing the list of periodic frames.

- wOffset: Index of the periodic message [0-15]

The application disposes of 16 periodic messages. This parameter corresponds to the index of the message the application wish to access. Note: these 16 messages are treated independently.

- wParam : Massage's parameter

Bit 0 : Periodic message operating indicator

- 0 : The periodic message is inactivated
- 1 : The periodic message is activated (start or update of the periodic)

Bit 1 : End of transmissions ascent indicator

For the sake of performance, it is not systematically needed for an application to receive the end of transmission events linked to the periodic messages emission.

- 0 : Les fins de transmissions ne sont pas remontées à l'application
- 1 : Les fins de transmissions sont remontées à l'application

- hCanMsg : Parameters of the message to be emitted

wHandleMsg	Communication index BUFFER mode: - this index has previously been returned by the CanCreateMsg function when creating the message. Compared to the configuration, only the data (BDAT) and lengths (wDataLen) are considered. FIFO analyse mode: - Messages are serialized by the DLL, this parameter is not used.
dwIdent	CAN identifier of the message [0-0x7FF] if standard identifier [0x1FFFFFFF] if extended identifier
eTypeId	Identifier type CAN_ID_STD : standard identifier (11 bits) CAN_ID_XTD : extended identifier (29 bits)
dwMask	Not used
eService	CAN_SVC_TRANSMIT_DATA - Data transmission CAN_SVC_TRANSMIT_RTR - Request of distant transmission
lPeriod	Message periodicity: ms.
dwReserved1	Reserved
dwReserved2	Reserved
wDataLen	Data maximum length (sent or received) [0-8]

bData	Content of data (for emission)

Outputs parameters:
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_MSGEXCEED	Wrong message offset

4.16. CanCreateMsg : Configuration of communication messages

Prototype:
tMuxStatus CanCreateMsg(unsigned short wCard, unsigned short wBus, tCanMsg *hCanMsg);

Description :
This function allows the user to declare the messages managed by the application.

Inputs parameters:
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
hCanMsg : Message parameters

wHandleMsg	See output parameter
dwIdent	CAN message identifier [0-0x7FF] if standard identifier [0x1FFFFFFF] if extended identifier
eTypeId	Type of identifier CAN_ID_STD : Standard identifier (11 bits) CAN_ID_XTD : Extended identifier (29 bits)
dwMask	Reception mask associated to the message identifier [0-0x7FF] if standard identifier [0x1FFFFFFF] if extended identifier The mask is used for the (CAN_SVC_RECEIVE_DATA and CAN_SVC_RECEIVE_RTR) type services (C.F. note 1)
eService	CAN_SVC_TRANSMIT_DATA - Data transmission CAN_SVC_RECEIVE_DATA - Data reception CAN_SVC_TRANSMIT_RTR

	- Request for distant transmission CAN_SVC_RECEIVE_RTR - Reception of request for distant transmission (C.F. note 2)
lPeriod	Complete with 0 (Reserved for future use)
dwReserved1	Complete with 0 (Reserved for future use)
dwReserved2	Complete with 0 (Reserved for future use)
wDataLen	Maximum length of data (sent or received) [0-8]
bData	Content of data (for transmitting)

Outputs parameters :

wHandleMsg	Communication index (only valid in BUFFER mode). This index is indicated by the DLL to the application so as to inform about the allocated buffer number. This index is then used for recovering the network events through the CanGetEvent function or for sending messages through the CanSendMessage function.

Note 1 :
The masks allow reception of a family of identifiers (see example of CanConfigParam function). However, certain precautions must be taken:

- In the analysis mode, the mask defined by the CanConfigParam function has priority.
- Regardless of the mode, it is important that there is no intersection between two families. Should that be the case, the identifier would be received by the firs declared corresponding message.

Note 2 :
In the FIFO mode analysis, it is not necessary to declare the initiating messages (service: CAN_SVC_TRANSMIT_DATA and CAN_SVC_TRANSMIT_RTR).

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_MSGEXCEED	Too many declared messages

4.17. CanSetNotification : Declaration of the application event

Prototype :
tMuxStatus CanSetNotification (unsigned short wCard, unsigned short wBus, HANDLE hWinEvent);

Description :
This function allows the application to use the task synchronisation methods provided by the Windows operation systems (API WIN32). This function allows transference of an event handle created by the Windows function « CreateEvent » to the DLL. This event is used during communication when an event is transferred from the DLL to the application (for example: end of transmission ending, reception, an error…).
From then on, the application can wait for events through waiting functions such as WaitForSingleObject or WaitForMultipleObject.

Example :

```
/* Request for indication of event */
HANDLE hWinEvent ;
tCanEvent hCanEvent ;
tMuxStatus Status ;
unsigned short wCard=0,wBus=0 ;

hWinEvent=CreateEvent(NULL,FALSE,FALSE,NULL) ;
if (hWinEvent == NULL) return ;
Status=CanSetNotification (wCard,wBus,hWinEvent) ;
if (Status != STATUS_OK) return ;
Status=CanActivate(wCard,wBus) ;
if (Status != STATUS_OK) return ;
if (WaitForSingleObject(hWinEvent,INFINITE) == WAIT_OBJECT_0)
{
        CanGetEvent(wCard,wBus,&hCanEvent);
}
```

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
hWinEvent : « handle » indicated by the CreateEvent function.

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

4.18. CanActivate : Starting communication

Prototype :
tMuxStatus CanActivate(unsigned int wCard, unsigned int wBus);

Description :
This function allows communication with the network to start. After executing this function, the messages received are acknowledged and sent towards the application.
The application can also send messages.

Inputs parameters
wCard : Index of card number to be accessed
wBus : Index of bus number

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

4.19. CanDeactivate : Stopping communication

Prototype :
tMuxStatus CanDeactivate(unsigned int wCard, unsigned int wBus);

Description :
This function stops communication with the network. After executing this function, messages are no longer received, acknowledged or sent.

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error

STATUS_ERR_SEQUENCE	Sequence error

4.20. CanSendMsg: Sending a message

Prototype :
tMuxStatus CanSendMsg(unsigned short wCard, unsigned short wBus, tCanMsg *hCanMsg);

Description :
This function allows sending a data message or a request of distant transmission.

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
hCanMsg : Parameter of message to be sent

wHandleMsg	Communication index For the BUFFER mode: - This index has previously been indicated by the CanCreateMsg function during the creation of the message. As for configuration, only the data (bData) and their length (wDataLen) are taken into account. For the analysis mode FIFO : - The messages are placed in series by the DLL. This parameter is not used.
dwIdent	CAN message identifier [0-0x7FF] if standard identifier [0x1FFFFFFF] if extended identifier
eTypeId	Type of identifier CAN_ID_STD : Standard identifier (11 bits) CAN_ID_XTD : Extended identifier (29 bits)
dwMask	Unused
eService	CAN_SVC_TRANSMIT_DATA - Data transmission CAN_SVC_TRANSMIT_RTR - Request for distant transmission
lPeriod	Reserved for future use
dwReserved1	Reserved for future use
dwReserved2	Reserved for future use
wDataLen	Maximum length of data (sent or received) [0-8]
bData	Content of data (for transmitting)

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_FIFOFULL	The new message cannot be taken into account. The transmitting FIFO is full.
STATUS_ERR_BUSOFF	The new message cannot be taken into account. The CAN protocol controller is disconnected from the bus.
STATUS_ERR_MSGSVC	Non supported service
STATUS_ERR_HANDLE	Parameter wHandleMsg is not valid
STATUS_ERR_MSGBUSY	The new message cannot be taken into account. The previous message is being transmitted.

4.21. CanSendMsgList : Sending several messages

Prototype :
tMuxStatus _MUXAPI CanSendMsgList(unsigned short wCard, unsigned short wBus, unsigned short wMsgCount, tCanMsg *hCanMsgList)

Description :
This function allows the sending of one or several data messages or distant transmission request.

Inputs parameters:
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
wMsgCount : Quantity of messages to be emitted
hCanMsg : Parameter of the message to be emitted

wHandleMsg	Communication index For the BUFFER mode: - This index has previously been indicated by the CanCreateMsg function during the creation of the message. As for configuration, only the data (bData) and their length (wDataLen) are taken into account. For the analysis mode FIFO : - The messages are placed in series by the DLL. This parameter is not used.
dwIdent	CAN message identifier [0-0x7FF] if standard identifier

	[0x1FFFFFFF] if extended identifier
eTypeId	Type of identifier CAN_ID_STD : Standard identifier (11 bits) CAN_ID_XTD : Extended identifier (29 bits)
dwMask	Unused
eService	CAN_SVC_TRANSMIT_DATA - Data transmission CAN_SVC_TRANSMIT_RTR - Request for distant transmission
lPeriod	Reserved for future use
dwReserved1	Reserved for future use
dwReserved2	Reserved for future use
wDataLen	Maximum length of data (sent or received) [0-8]
bData	Content of data (for transmitting)

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_FIFOFULL	The new message cannot be taken into account. The transmitting FIFO is full.
STATUS_ERR_BUSOFF	The new message cannot be taken into account. The CAN protocol controller is disconnected from the bus.
STATUS_ERR_MSGSVC	Non supported service
STATUS_ERR_HANDLE	Parameter wHandleMsg is not valid
STATUS_ERR_MSGBUSY	The new message cannot be taken into account. The previous message is being transmitted.

4.22. CanGetEvent: Reading of an event

Prototype :
tMuxStatus CanGetEvent(unsigned short wCard, unsigned short wBus, tCanEvent *hCanEvent);

Description :
This function allows the user to recover an event from the network.

Inputs parameters :

wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
hCanEvent : Pointer on the event to complete

Outputs parameters : :
hCanEvent : Event complete

wBus	Number of bus managing the event
wHandleMsg	By way of example : Number of DLL's internal channel (C.F. CanCreateMsg)
eTypeEvent	Type of event EVENT_EMPTY : No event. EVENT_CAN_MSGTX : - End of correct transmission of a message EVENT_CAN_MSGRX : - Correct reception of a message EVENT_CAN_ERROR : - Network error EVENT_CAN_BUSCHANGE - Change in the protocol controller status EVENT_CAN_BUSLOAD - Bus charge EVENT_TIMER - Applicable timer EVENT_TIMEERROR - Loss of IT timer EVENT_FIFO_OVF - Bit indicating that one or several events after such bit are lost because the FIFO reception was full.
dwIdent	CAN message identifier [0-0x7FF] if standard identifier [0x1FFFFFFF] if extended identifier
eTypeId	Type of identifier CAN_ID_STD : Standard identifier (11 bits) CAN_ID_XTD : Extended identifier (29 bits)
dwTimeStamp	Arrival time of the event in a multiple of 100 µSec (1 ms precision for PCI-MUX cards). This parameter is correct if wTimePrecision is zero. C.F. ANNEXE : Time stamp indication with PCI-MUX cards
wTimePrecision	Precision of dwTimeStamp in multiple of 100 µSec.
eService	For events related to a message, message service (see CanCreateMsg function).
wError	For events of EVENT_CAN_ERROR type, indication of type of error :

Bit 7 to 6 type of error :
- 0 : Bit error
- 1 : Format error
- 2 : Stuffing error
- 3 : Another type of error

Bit 5 Direction when the error takes place :
- 0 : During transmission
- 1 : During reception

Bit 4 à 0 Field where the error has taken place :
- 0x02 : ID20 to ID21
- 0x03 : Start of frame
- 0x04 : SRTR bit
- 0x05 : IDE bit
- 0x06 : ID20 to ID18
- 0x07 : ID17 to ID13
- 0x08 : CRC field
- 0x09 : Bit reserved 0
- 0x0A : DATA field
- 0x0B : DLC field
- 0x0C : RTR bit
- 0x0D : Bit reserved 1
- 0x0E : ID4 to ID0
- 0x0F : ID12 to ID5
- 0x11 : Active error flag
- 0x12 : Intermission
- 0x13 : Dominant bits
- 0x16 : Passive error flag
- 0x17 : Error delimiter
- 0x18 : CRC delimiter
- 0x19 : Acknowledge slot
- 0x1A : End of frame
- 0x1B : Acknowledge delimiter
- 0x1C : Overload flag

eChipState — Status of CAN protocol controller
- CAN_BUS_ACTIVE : Active error
- CAN_BUS_PASSIVE : Passive error
- CAN_BUS_OFF : Disconnected component (bus off)

wLineState — Bit 0 : Status of ERR bit in low speed line interface
- 1 : Communication in nominal mode.
- 0 : Communication in degraded mode.

Bit 1 : Status of INH line in the low speed interface (PCI-MUX-CAN and PCI-MUX-MultiCAN cards only)
- 1 : INH output is active
- 0 : INH output is inactive

dwReserved1	Reserved for future use
dwReserved2	Reserved for future use
wBusLoad	For an event of the EVENT_CAN_BUSLOAD type, value of bus charge in %.
wDataLen	Length of data [0-8]
bData	Data buffer

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

Example :

```
/* Indication of event */
tCanEvent hCanEvent ;
tMuxStatus Status ;
unsigned short wCard=0,wBus=0 ;

Status=CanGetEvent(wCard,wBus,&hCanEvent);
if (Status != STATUS_OK) return ;
switch(hCanEvent.eTypeEvent &~ EVENT_FIFO_OVF)
{
        case EVENT_EMPTY :              /* No more events to be dealt with */
                break ;
        case EVENT_CAN_MSGTX :          /* Correct end of transmission */
                break ;
        case EVENT_CAN_MSGRX :          /* Correct reception */
                break ;
        case EVENT_CAN_ERROR :          /* Network error*/
                break ;
        case EVENT_CAN_BUSCHANGE :      /* Change in the component's status */
                break ;
        case EVENT_CAN_BUSLOAD :        /* Bus charge */
                break ;
        case EVENT_TIMER :              /* Applicable timer */
                break ;
        case EVENT_TIMEERROR :          /* Loss IT timer */
                break ;
        default  :                      /* Reserved for future use */
                break ;
}
```

4.23. CanGetStat : Reading statistics counters

Prototype :
tMuxStatus CanGetStat(unsigned short wCard, unsigned short wBus, tCanStat *hCanStat);

Description :
This function allows the user to read the network's operational counters.

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
hCanStat : Pointer on the statistic area to be completed

Outputs parameters :
hCanStat : Event to be completed

wBusLoad	Network charge expressed in percentage
wReserved	Reserved for future use
dwTxRq	Number of transmission requests
dwTxOk	Number of correct transmission endings
dwRxOk	Correct reception
dwErrTxBit	Number of Bit type of errors in transmission
dwErrRxBit	Number of Bit type of errors in reception
dwErrTxForm	Number of format errors in transmission
dwErrRxForm	Number of format errors in reception
dwErrTxStuff	Number of stuffing errors in transmission
dwErrRxStuff	Number of stuffing errors in reception
dwErrTxOther	Number of stuffing errors in transmission different from the ones mentioned above
dwErrRxSOther	Number of stuffing errors in reception different from the ones mentioned above
dwErrFifoOvf	Number of lost messages (full reception FIFO)

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

4.24. CanIsBusActive : GIves CAN controller status

Prototype:
tMuxStatus _MUXAPI CanIsBusActive(unsigned short wCard, unsigned short wBus, unsigned short *wState)

Description:
This function allows getting the activation status of a CAN bus.

Inputs parameters:
wCard: Index of the card's number to be accessed
wBus: Index of the bus number [0-x]

Outputs parameters:
wState : bus status
0 = CAN bus inactive
1 = CAN bus active

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

4.25. CanGetBusState : Reading CAN controller status

Prototype :
tMuxStatus CanGetBusState(unsigned short wCard, unsigned short wBus, tCanChipState *eCanChipState, unsigned char *bTxErrCount, unsigned char *bRxErrCount);

Description :
This function allows the user to directly read the status of the CAN protocol controller. This function also transfers the value of the transmission and reception counters used to manage the following statuses : active error, passive error and bus off.

Inputs parameters:
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
eCanChipState: Pointer on the status to be completed
bTxErrCount: Pointer on the counter to be completed
bRxErrCount: Pointer on the counter to be completed

Outputs parameters:
eCanChipState: Status of component
bTxErrCount: Transmission counter
bRxErrCount: Reception counter

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

4.26. CanBusOn : Controller reconnection after bus off

Prototype :
tMuxStatus _MUXAPI CanBusOn(unsigned short wCard, unsigned short wBus);

Description :
This function allows the user to reconnect the CAN protocol controller after switch to bus mode has been detected.

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

4.27. CanReadByte: Direct reading of CAN protocol controller

Prototype :
tMuxStatus CanReadByte(unsigned short wCard, unsigned short wBus, unsigned short wOffset,unsigned char *bData);

Description :
This function allows the user to read directly the registers of the CAN protocol controller.

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-2]
wOffset : Reading offset

bData : Pointer on the value to be read

Outputs parameters :
bData : Read value

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

4.28. CanWriteByte : Direct writing in the CAN protocol controller

Prototype :
tMuxStatus CanWriteByte(unsigned short wCard, unsigned short wBus, unsigned short wOffset,unsigned char bData);

Description :
This function allows the user to write directly in the registers of the CAN protocol controller.

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
wOffset : Writing offset
bData : Value to be written

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

4.29. CanGetFifoRxLevel : Level of filling of events waiting list

Prototype :
tMuxStatus CanGetFifoRxLevel (unsigned short wCard, unsigned short wBus, unsigned short wParam, unsigned short *wCount, unsigned short *wMaxCount);

Description :

This function allows the user to see how full the waiting list of events transferred to the application is.

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
wParam : Function's parameters
Bit 0 : Type of waiting list (only for USB cases)
- 0 : Event queue is situated on the PC.
- 1 : Reserved for future use

Outputs parameters :
wCount : Number of event currently on waiting list
wMaxCount : Maximum number of events accepted on the waiting list

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

4.30. CanClearFifoRx : Clears the reception FIFO

Prototype:
tMuxStatus _MUXAPI CanClearFifoRx(unsigned short wCard, unsigned short wBus) ;

Description :
This function allows to clear the reception events queue.

Inputs parameters:
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]

Outputs parameters:
None

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

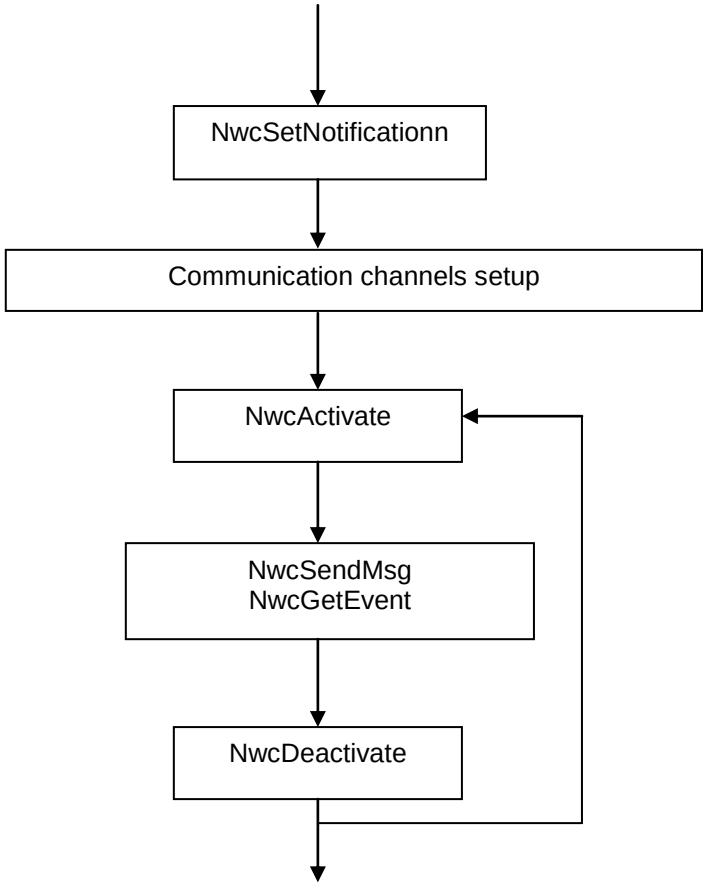
5. NWC Library

5.1. Order of call

5.1.1 Exchanges sequence

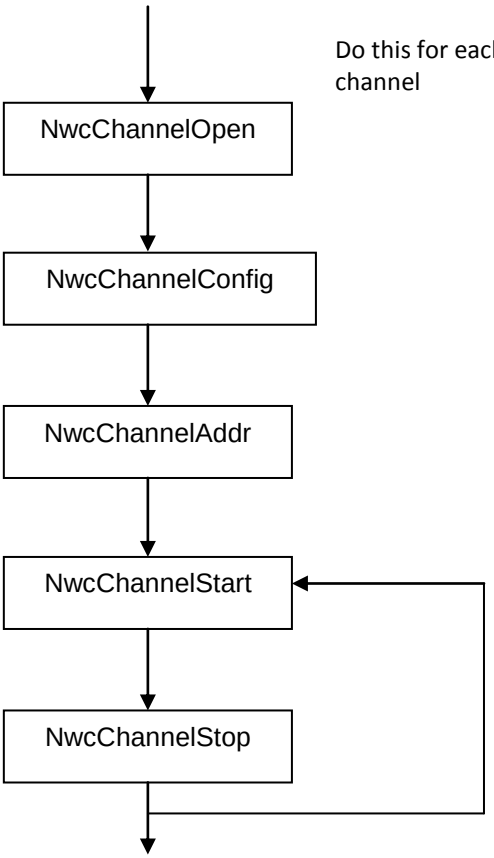
First Step: Set up all communication channels: the goal is to fill the different messages to be used by the application and also the communication parameters (CAN identifiers, delay and time-out).
Second Step: Start communication using « NwcActivate » function.
Third Step: Send a message.

- NwcSendMsg: Send a frame with a length between 0 and 4095 bytes.
- NwcGetEvent: Retrieve transmission events like ends of transmission, transmission errors, reception and reception errors.



5.1.2 Communication channels setup

Application must respect this sequence for each channel. Note that the functions allowing or not the communication on a channel (NwcChannelStart and NwcChannelStop) can be called during a communication.



5.2. NwcRegister: Enable NWC funtions use

Prototype:
tMuxStatusNwcRegister(char *szString, char *szKey);

Description:
This function enables application to use NWC functions (deprecated since version 5.40).

Inputs parameters:
szString: register string
szKey: register key

Outputs parameters:
None

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameter error
STATUS_ERR_SEQUENCE	calling sequence error

5.3. NwcGetChannelCount: Count available communication channels

Prototype:
tMuxStatus NwcGetChannelCount(unsigned short wCard, unsigned short wBus, unsigned short * wChannelCount);

Description:
One communication channel is defined as a communication between a sender « a » and a receiver « b ». An application can open several channels at the same time. This function returns the maximum of channels that the application can open at the same time.

Inputs parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]

Outputs parameters:
wChannelCount: Number of available channels.

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameter error
STATUS_ERR_SEQUENCE	calling sequence error
STATUS_ERR_REGISTER	Request not register

5.4. NwcChannelOpen: Open a communication channel

Prototype :
tMuxStatus NwcChannelOpen(unsigned short wCard, unsigned short wBus, unsigned short *wChannel);

Description:
This function open a logical communication channel for communicate between a sender « a » and a receiver « b ».

Inputs parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]

Outputs parameters:
wChannel: Index of the open channel.

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameter error
STATUS_ERR_SEQUENCE	calling sequence error
STATUS_ERR_REGISTER	Request not register
STATUS_ERR_MEMORY	No more available channel

5.5. NwcChannelConfig: Configure a communication channel

Prototype:
tMuxStatus _MUXAPI NwcChannelConfig(unsigned short wCard, unsigned short wBus, unsigned short wChannel, tNwcConfig *hNwcChannelConfig);

Description:
This function can configure parameters of the communication channel.

Inputs parameters:
wCard: Index of card number to be accessed

wBus: Index of bus number [0-x]
wChannel: Index of the open channel (get with the function: NwcChannelOpen)
hNwcChannelConfig: Channel parameters

eNwcAddrMode	Addressing mode : NWC_MODE_PHYSICAL : Physical addressing - In this mode, message will be transmitted between 2 physical entities. It's a point to point communication (1 to 1 communication), and this mode allows message length up to 4095 bytes. NWC_MODE_FUNCTIONNAL : Functional addressing - In this mode, message will be transmitted from 1 entity to n entities (1 to n communication). Several ECU are allowed to answer, but shall only be supported for Single Frame communication.
eNwcAddrFormat	Addressing format : NWC_FORMAT_NORMAL : Normal addressing - In this mode, for each combination of N_SA (Start Address), N_TA (Target Address), N_TAtype (Physical addressing or Functional addressing) and Mtype (diagnostics or remote diagnostics) a unique CAN identifier is assigned. N_PCI (Protocol Control Information) and N_Data (Data Field) is placed in the CAN frame data field. Identifier length = 11 or 29 bits. NWC_FORMAT_NORMAL_FIXED: Normal fixed addressing - Normal fixed addressing is a sub-format of normal addressing where the mapping of the address information into the CAN identifier is further defined. In the general case of normal addressing, described above, the correspondence between N_AI (Address Information) and the CAN identifier is left open. For normal fixed addressing only 29 bit CAN identifiers are allowed. The following tables define the mapping of the address information (N_AI) into the CAN identifier, depending on the target address type (N_TAtype). N_PCI and N_Data is placed in the CAN frame data field. NWC_FORMAT_EXTENDED: Extended addressing - For each combination of N_SA, N_TAtype and Mtype a unique CAN identifier is assigned. N_TA is placed in the first data byte of the CAN frame data field. N_PCI and N_Data is placed in the remaining bytes of the CAN frame data field. Identifier length = 11 or 29 bits.

	NWC_FORMAT_MIXED: Mixed addressing - Mixed addressing is the addressing format to be used if Mtype is set to remote diagnostics. In this mode, N_SA (Start Address), N_TA (Target Address) is placed in the identifier field. For mixed addressing only 29 bit CAN identifiers are allowed. N_PCI and N_Data is placed in the remaining bytes of the CAN frame data field.
eNwcCommMode	NWC_COMM_HALF_DUPLEX: (not supported) - Point-to-point communication between two nodes is only possible in one direction at a time. NWC_COMM_FULL_DUPLEX: - Point-to-point communication between two nodes is possible in both directions at a time.
eNwcService	Request Service: NWC_SVC_TRANSMIT_DATA : Data transmission NWC_SVC_RECEIVE_DATA : Data receipt
wParam	Field of bits defining working parameters Frame length: NWC_VARIABLE_LEN: Frame length depends of necessary data length. NWC_FIXED_LEN: Frame length is fixed to 8 bytes. Protocol Events: NWC_CAN_NO_EVENT: Application will not receive events corresponding to the low level communication. NWC_CAN_EVENT: Application will receive all Events. Filtering FF (first frame): NWC_FF_NO_EVENT: Application will not receive Events corresponding to a First Frame receipt or a First Frame end of transmission. NWC_FF_EVENT: Application will receive Events corresponding to a First Frame receipt or a First Frame end of transmission.

Outputs parameters:
None

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_REGISTER	Request not register

5.6. NwcChannelSetBytePadding: Customization of padding bytes

Prototype :
tMuxStatus _MUXAPI NwcChannelSetBytePadding(unsigned short wCard,unsigned short wBus, unsigned short wChannel, tNwcBytePadding *hNwcBytePadding) ;

Description :
This request allows to set padding bytes during frame's segmenting. This padding occurs when the parameter wParam of the previous function has the flag NWC_FIXED_LEN.

Inputs parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
wChannel: Index of the open channel (get with the function: NwcChannelOpen)
hNwcBytePadding : configuration parameter

bData	Contains the bytes to be used during padding
dwReserved1	Reserved for a future use
dwReserved2	Reserved for a future use

Output parameters:
None

Return code :
Status: Summary of the function execution

STATUS_OK
STATUS_ERR_PARAM Parameters error

5.7. NwcChannelAddr: Communication identifiers definition

Prototype:
tMuxStatus NwcChannelAddr(unsigned short wCard, unsigned short wBus, unsigned short wChannel, tNwcAddr *hNwcChannelAddr);

Description:
This function allows configuration of the identifiers for a communication channel.

Inputs parameters:
wCard: Index of card number to be accessed

wBus: Index of bus number [0-x]
wChannel: Index of the open channel (get with the function: NwcChannelOpen)
hNwcChannelAddr: Communication channel address

bSA	"Source address" : Address of source
bTA	« Target address» : Address of target
bReserved	Reserved
dwIdentTx	Frame identifier
eTypeIdTx	Identifier type CAN_ID_STD : Standard address(11 bits) CAN_ID_XTD : Extended address (29 bits)
dwIdentFC	Identifier of the flow control frame.
eTypeIdFC	Identifier type for flow control frame CAN_ID_STD : Standard address(11 bits) CAN_ID_XTD : Extended address (29 bits)

Outputs parameters:
None

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_REGISTER	Request not register

5.8. NwcChannelParam : Channel communication parameters

Prototype:
tMuxStatus NwcChannelParam(unsigned short wCard, unsigned short wBus, unsigned short wChannel, tNwcChannelParam *hNwcChannelParam);

Description:
This function allows the application to configure channel communication parameters. Theses parameters define time limit for segmented messages and flow control frames. Application is allowed to change theses parameters dynamically, if there is no communication running.

Inputs parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
wChannel: Index of the open channel (get with the function: NwcChannelOpen)
hNwcChannelParam: Communication parameters

wBlockSize	Number of consecutive blocks sent [0-255] 0 means that none flow control is waiting	0
wSTmin	The Separation Time (STmin) parameter value specifies the minimum time gap allowed between the transmissions of consecutive frame network protocol data units. The units of STmin are absolute milliseconds ([0-255] ms). The measurement of the Separation Time (STmin) starts after completion of transmission of a Consecutive Frame (CF) and ends at the request for the transmission of the next Consecutive Frame (CF). For receiver channels this parameter corresponding to the STmin value sent inside the flow control frame. For sender channels this parameter is added to the STmin value send inside the flow control frame to generate Cs time.	10
wN_As	Maximum transmission time of the transmitter [0-65535] ms	10
wN_Ar	Maximum transmission time of the receiver [0-65535] ms	10
wN_Bs	Time until reception of flow control [0-65535] ms	20
wN_Br	Time until transmission of flow control [0-65535] ms	0
wN_Cr	Time until transmission of Consecutive Frame [0-65535] ms	20
wN_WFTmax	Number of maximum flow control expected [0-65535]. 0 define unlimited waiting time.	0

Outputs parameters:
None

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_REGISTER	Request not register
STATUS_ERR_BUSY	Communication running

5.9. NwcChannelStart : Start communication for a channel

Prototype:
tMuxStatus NwcChannelStart(unsigned short wCard, unsigned short wBus, unsigned short wChannel);

Description:
This function starts communication on the specified channel.

Inputs parameters:

wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
wChannel: Index of the open channel (get with the function: NwcChannelOpen)

Outputs parameters:
None

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_REGISTER	Request not register

5.10. NwcChannelStop : Stop communication for a channel

Prototype:
tMuxStatus NwcChannelStop(unsigned short wCard, unsigned short wBus, unsigned short wChannel);

Description:
This function stops communication on the specified channel.

Inputs parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
wChannel: Index of the open channel (get with the function: NwcChannelOpen)

Outputs parameters:
None

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_REGISTER	Request not register

5.11. NwcActivate : Start communication

Prototype:
tMuxStatus NwcActivate(unsigned int wCard, unsigned int wBus);

Description :
This function starts communication on the network. After execution of this function, application can send or receive messages.

Inputs parameters :
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]

Outputs parameters:
None

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_REGISTER	Request not register

5.12. NwcDeactivate : Stop communication

Prototype:
tMuxStatus NwcDeactivate(unsigned int wCard, unsigned int wBus);

Description :
This function stops communication on the network. After execution of this function, application can't send or receive messages.

Inputs parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]

Outputs parameters:
None

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_REGISTER	Request not register

5.13. NwcChannelSendMsg: Send a message

Prototype:
tMuxStatus NwcSendMsg(unsigned short wCard, unsigned short wBus, unsigned short wChannel, tNwcMsg *hNwcMsg);

Description:
This function sends a message.

Inputs parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
wChannel: Index of the open channel (get with the function: NwcChannelOpen)
hNwcMsg: Message parameters:

wDataLen	Max data length [0-4095]
bData	Data to send

Outputs parameters:
None

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_MSGSVC	Channel is not a valid transmission channel
STATUS_ERR_MSGBUSY	New message could not be sent because a message is currently transmitted.

5.14. NwcGetEvent : Reading of an event

Prototype:
tMuxStatus NwcGetEvent(unsigned short wCard, unsigned short wBus, tNwcEvent *hNwcEvent);

Description:
This function allows the user to recover an event from the network.

Inputs parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
hNwcEvent: Pointer on the event to complete

Outputs parameters:

hNwcEvent: Event complete

wBus	Index of bus managing the event
wChannel	Index of channel managing the event [0-n] (or 0xFFFF if spy mode is activate)
eTypeEvent	Type of event EVENT_EMPTY: No event. EVENT_NWC_MSGTX: - End of correct transmission of a message EVENT_ NWC_MSGRX : - Correct reception of a message EVENT_NWC_MSGTXERR: - End of transmission error EVENT_NWC_MSGRXERR : - Receipt error EVENT_NWC_MSGRXFF: - Correct reception of a First Frame EVENT_NWC_MSGTXFF: - Correct end of transmission of a First Frame EVENT_FIFO_OVF - Bit indicating that one or several events after such bit are lost because the FIFO reception was full.
dwIdent	CAN message identifier [0-0x7FF] if standard identifier [0x1FFFFFFF] if extended identifier
eTypeId	Type of identifier CAN_ID_STD : Standard identifier (11 bits) CAN_ID_XTD : Extended identifier (29 bits)
dwTimeStamp	Arrival time of the event in a multiple of 100 µSec (1 ms precision for PCI-MUX cards). This parameter is correct if wTimePrecision is zero. C.F. ANNEXE : Time stamp indication with PCI-MUX cards
wTimePrecision	Precision of dwTimeStamp in multiple of 100 µSec.
eService	Request service : NWC_SVC_TRANSMIT_DATA : Data Transmission NWC_SVC_RECEIVE_DATA: Data receipt.
eError	For events of type EVENT_NWC_MSGTXERR or EVENT_NWC_MSGRXERR, indication of type of error : - NWC_NO_ERR : No error - NWC_ERR_TOUT_AS : Maximum transmission time of the transmitter elapsed (wN_As) - NWC_ERR_TOUT_AR : Maximum transmission time of the receiver elapsed (wN_Ar) - NWC_ERR_TOUT_BS : Time until reception of flow control elapsed (wN_Bs)

	- NWC_ERR_TOUT_CR : Time until transmission of Consecutive Frame elapsed (wN_Cr) - NWC_ERR_SN: Sequence Number error. Receipt is stopped - NWC_ERR_FIFO_OVF: FIFO for transmit CAN messages is full. Receipt or transmission is stopped - NWC_ERR_MEMORY : No enough memory - NWC_ERR_INV_PDU : PDU Format receipt error
dwReserved1	Reserved for future use
dwReserved2	Reserved for future use
wDataLen	Length of data [0-4095]
bData	Data buffer

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

Example:

```
/* Indication of event */
tNwcEvent hNwcEvent ;
tMuxStatus Status ;
unsigned short wCard=0,wBus=0 ;

Status=NwcGetEvent(wCard,wBus,&hNwcEvent);
if (Status != STATUS_OK) return ;
switch(hNwcEvent.eTypeEvent &~ EVENT_FIFO_OVF)
{
        case EVENT_EMPTY :              /* No more events to be dealt with */
                break ;
        case EVENT_NWC_MSGTX:           /* Correct end of transmission */
                break ;
        case EVENT_NWC_MSGRX:           /* Correct reception */
                break ;
        case EVENT_NWC_MSGTXERR:        /* End of transmission error*/
                break ;
        case EVENT_NWC_MSGRXERR:        /* Receipt error*/
                break ;
        case EVENT_NWC_MSGRXFF:         /* A First Frame is receipted*/
                break ;
        case EVENT_NWC_MSGTXFF:         /* End of transmission of a First Frame*/
                break ;
        default  :                      /* Reserved for future use */
                break ;
```

}

5.15. NwcGetIdent : Get CAN communication identifiers

Prototype:
tMuxStatus NwcGetIdent (tNwcConfig *pstNwcConfig, tNwcAddr *pstNwcAddr);

Description:
This function return all CAN identifiers used.

Inputs parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
pstNwcConfig: Pointer on a tNwcConfig structure (see NwcChannelConfig)
pstNwcAddr: Pointer on tNwcAddr structure (see NwcChannelAddr)

Outputs parameters:
PstNwcAddr: Array of CAN identifiers used.

dwIdentTx	Frame identifier
eTypeIdTx	Identifier type CAN_ID_STD : Standard address(11 bits) CAN_ID_XTD : Extended address (29 bits)
dwIdentFC	Identifier of the flow control frame.
eTypeIdFC	Identifier type for flow control frame CAN_ID_STD : Standard address(11 bits) CAN_ID_XTD : Extended address (29 bits)

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

5.16. NwcGetFifoRxLevel : Level of filling of events waiting list

Prototype:
tMuxStatus NwcGetFifoRxLevel (unsigned short wCard, unsigned short wBus, unsigned short wParam, unsigned short *wCount, unsigned short *wMaxCount);

Description:
This function allows the user to see how full the waiting list of events transferred to the application is.

Inputs parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
wParam: Function's parameters
Bit 0 : Type of waiting list (only for USB cases)
- 0 : Event queue is situated on the PC.
- 1 : Reserved for future use

Outputs parameters:
wCount: Number of event currently on waiting list
wMaxCount: Maximum number of events accepted on the waiting list

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

5.17. NwcConfigSpyMode : Configure spy mode

Prototype:
tMuxStatus NwcConfigSpyMode(unsigned short wCard, unsigned short wBus, tNwcSpyAddr *hNwcSpyAddr);

Description:
This function allows configuration of spy mode. This mode checks segmentation conformity and allows application to return get a frame. The return Event will have a channel index fixed at 0x000.

Inputs parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
hNwcSpyAddr: Function parameters

eTypeId	Type of identifier CAN_ID_STD : Standard identifier (11 bits) CAN_ID_XTD : Extended identifier (29 bits)
dwIdentReq	Request identifier or response Flow Control
dwIdentRsp	Response identifier or request Flow Control
dwMask	Mask of acceptance filter
wParam	Field of bits for define working parameters Frame length : NWC_VARIABLE_LEN: Frame length depends of

	necessary data length. NWC_FIXED_LEN: Frame length is fixed to 8 bytes. Protocol Events : NWC_CAN_NO_EVENT: Application will not receive Events corresponding to the low level communication. NWC_CAN_EVENT: Application will receive all Events. Filtering FF (first frame) : NWC_FF_NO_EVENT: Application will not receive Events corresponding to a First Frame receipt or a First Frame end of transmission. NWC_FF_NO_EVENT: Application will receive Events corresponding to a First Frame receipt or a First Frame end of transmission.

Outputs parameters:
None

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

5.18. NwcGetChannelState : Current state of a channel

Prototype:
tMuxStatus NwcGetChannelState(unsigned short wCard, unsigned short wBus, unsigned short wChannel, unsigned short *wChannelState, tNwcError *eLastTxStatus);

Description:
This function allows the user to directly read the status of communication channel.

Inputs parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
wChannel: Index of the open channel (get with the function: NwcChannelOpen)

Outputs parameters:
wChannelState: Status of communication channel
Bit NWC_CHANNEL_STATE_STARTED:
0 : Channel is not activated

1 : Channel is activated
Bit NWC_CHANNEL_STATE_BUSY
0 : Channel is activated
1 : Channel is currently transmitting or receiving a message

eLastTxStatus : Status of the last end of transmission

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

5.19. NwcIsBusActive : Bus Status

Prototype:
tMuxStatus NwcIsBusActive(unsigned short wCard, unsigned short wBus, unsigned short *wState);

Description :
This function allows to indicate if the bus has been activated using the NwcActivate function.

Input parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]

Ouptut parameters:
wState: Indicates if the bus is activated or not (value different than 0)

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

5.20. NwcChannelClose : Closing a communication channel

Prototype:
tMuxStatus NwcChannelClose(unsigned short wCard, unsigned short wBus, unsigned short wChannel);

Description :
This function allows closing a communication channel.

Input parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
wChannel : Number of the channel to be closed

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

5.21. NwcSetNotification: Declaration of the application event

Prototype:
tMuxStatus NwcSetNotification (unsigned short wCard, unsigned short wBus, HANDLE hWinEvent);

Description :
This function allows the application to use the methods of tasks synchronization provided by the Windows operating system (API WIN32). This function allows to pass to the DLL an event handle created by the Windows function « CreateEvent ». This event is used during the communication when an event is sent back by the DLL to the application (e.g. end of transmission, reception, error, …).
Then the application can wait for events using waiting functions as WaitForSingleObject or WaitForMultipleObject.

Example:

```
/* Request of event indications */
HANDLE hWinEvent ;
tNwcEvent hNwcEvent ;
tMuxStatus Status ;
unsigned short wCard=0,wBus=0 ;

hWinEvent=CreateEvent(NULL,FALSE,FALSE,NULL) ;
if (hWinEvent == NULL) return ;
Status=NwcSetNotification (wCard,wBus,hWinEvent) ;
if (Status != STATUS_OK) return ;
Status=NwcActivate(wCard,wBus) ;
if (Status != STATUS_OK) return ;
if (WaitForSingleObject(hWinEvent,INFINITE) == WAIT_OBJECT_0)
{
```

```
        NwcGetEvent(wCard,wBus,&hNwcEvent);
}
```

Input parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
hWinEvent : Handle returned by the function CreateEvent.

Output parameters:
None

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

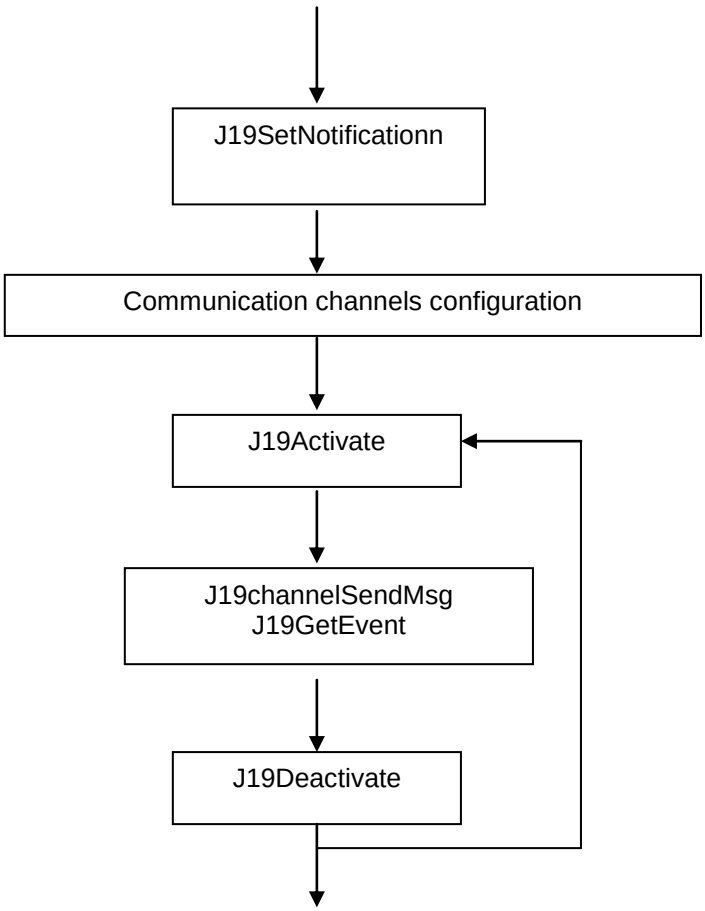
6. J1939 Library

6.1. Order of call

6.1.1 Exchanges sequence

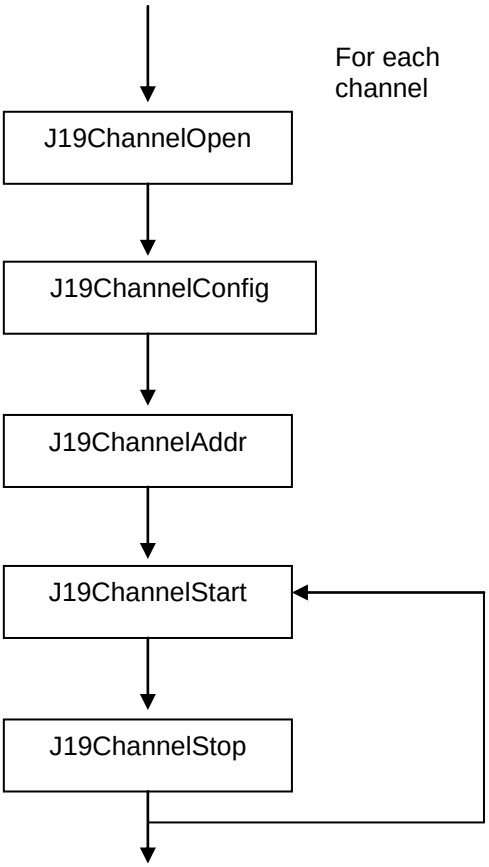
1st step, configure all communication channel :
The aim is to define the different messages to be used by the application and communication parameters (physical CAN identifiers used, time and time-out).
2nd step, start the communication using J19Activate request
3rd step, communicate with remote entities using two main requests:

- J19ChannelSendMsg: Emission of a frame of lenghts between 0 and 1785 bytes
- J19GetEvent: Retrieve all the events stored in a queue. These events indicate information such as end of transmission, reception, end of wrong transmission, wrong reception.



6.1.2 Communication channels configuration

The application must respect de following sequence for channel configuration.
Note: The functions allowing or not the communication on a channel (J19ChannelStart and J19ChannelStop) can be called during the communication.



6.2. J19ChannelOpen: Opening a communication channel

Prototype:
tMuxStatus J19ChannelOpen(unsigned short wCard, unsigned short wBus, unsigned short *wChannel);

Description :
This request allows the opening of a communication channel. This communication channel allows to connect logically 2 communicating entities.

Input parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]

Output parameters:
wChannel : Number of the opened channel.

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_MEMORY	No more channel available

6.3. J19ChannelAddr: Définition of communication identifiers

Prototype:
tMuxStatus J19ChannelAddr(unsigned short wCard, unsigned short wBus, unsigned short wChannel, tJ19Addr * hJ19Addr);

Description :
This request allows to configure the identifiers of the communication channel.

Input parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
wChannel : Index of the channel number (obtained by the J19ChannelOpen function)
hJ19Addr: Communication parameters

bPriority	Message priority (from 0 to 7)
dwPGN	PGN managed by the channel
bSA	Source address of the channel

Output parameters:
None

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

6.4. J19Param : Parameters of the communication channel

Prototype:
tMuxStatus J19ChannelParam(unsigned short wCard, unsigned short wBus, tJ19Param *hJ19ChannelParam);

Description :
This request allows configuring the communication parameters of a channel. These parameters allow managing the messages emission time. It is possible to modify these parameters dynamically, only in case there ir no transmission or reception in progress.

Input parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
hJ19ChannelParam: Channel communication parameters

wTr	Maximum time before the emission of an answer (ms)	200
wTh	Maximum time before the emission of a communication maintain frame (ms) (Not used)	500
wT1	Time until the reception of another TP.DT frame (ms)	750
wT2	Time until the reception of the 1st TP.DT (frame ms)	1250
wT3	Time until the reception of the 1st CTS (ms)	1250
wT4	Time until the reception of another CTS frame (ms)	1050

Output parameters:
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_BUSY	Communication in progress

6.5. J19ChannelConfig: Configuration of a communication channel

Prototype:
tMuxStatus J19ChannelOpen(unsigned short wCard, unsigned short wBus, unsigned short wChannel, tJ19Config *hJ19ChannelConfig);

Description :
This request allows configuring a communication channel.

Input parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
wChannel : Index of the channel number (obtained by the J19ChannelOpen function)
hJ19ChannelConfig: Canal parameters

eJ19Service	Service of the request J19_SVC_TRANSMIT_DATA : Data transmission J19_SVC_RECEIVE_DATA : Data reception
wParam	Not used. Reserved for future use

Output parameters:
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

6.6. J19ConfigBus: Bus configuration

Prototype:
tMuxStatus J19ConfigBus(unsigned short wCard, unsigned short wBus, unsigned short wParam);

Description:
This request allows configuring a communication channel.

Input parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
wParam: bits field defining the operating parameters.

Protocol level parameters:
- J19_CAN_NO_EVENT : All exchanges related to the communication layer is not sent back to the application
- J19_CAN_EVENT : All exchanges related to the communication layer is sent back to the application

Filtrage des RTS :
- J19_RTS_NO_EVENT : Events indicating the reception or end of transmission of a RTS frame are not sent back to the application

- J19_RTS_EVENT : Events indicating the reception or end of transmission of a RTS frame are sent back to the application

Comportment :

- J19_SPY_MODE : the frames will be processed at BUS level. The BAM transmissions will be sent back unsegmented

6.7. J19ChannelStart : Autorize the communication on the channel

Prototype:
tMuxStatus J19ChannelStart(unsigned short wCard, unsigned short wBus, unsigned short wChannel);

Description :
This request autorizes the communication on the channel.

Input parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
wChannel : Index of the channel number (obtained by the J19ChannelOpen function)

Output parameters:
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

6.8. J19ChannelStop : Stops the communication on the channel

Prototype:
tMuxStatus J19ChannelStop(unsigned short wCard, unsigned short wBus, unsigned short wChannel);

Description :
This request stops the communication on the channel.

Input parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
wChannel : Index of the channel number (obtained by the J19ChannelOpen function)

Output parameters:
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

6.9. J19Activate : Starting the communication

Prototype:
tMuxStatus J19Activate(unsigned int wCard, unsigned int wBus);

Description :
This function allows starting the communication with the network. After the execution of this function, the messages can be emitted or received by the application.

Input parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
wChannel : Index of the channel number (obtained by the J19ChannelOpen function)

Output parameters:
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

6.10. J19Deactivate : Stopping the communication

Prototype:
tMuxStatus J19Deactivate(unsigned int wCard, unsigned int wBus);

Description :
This function stops the communication with the network. After the execution of this function, the messages are not received or transmitted anymore.

Input parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
wChannel : Index of the channel number (obtained by the J19ChannelOpen function)

Output parameters:
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

6.11. J19ChannelSendMsg: Emission of a message

Prototype:
tMuxStatus J19ChannelSendMsg(unsigned short wCard, unsigned short wBus, unsigned short wChannel, tJ19Msg *hJ19Msg);

Description :
This function allows the emission of a data message.

Input parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
wChannel : Index of the channel number (obtained by the J19ChannelOpen function)
hJ19Msg : Parameters of the message to be emitted

wDataLen	Maximal data length (emitted or recived) [0-1785]
bData	Data contain (emission only)

Outputs parameters:
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_MSGSVC	The channel is not a transmission channel
STATUS_ERR_MSGBUSY	The new message cannot be considered. The

	previous message is being sent.

6.12. J19ChannelSendNMEAFastPacket: Emission of a message

Prototype:
tMuxStatus J19ChannelSendNMEAFastPacket (unsigned short wCard, unsigned short wBus, unsigned short wChannel, tJ19Msg *hJ19Msg);

Description :
This function allows the emission of a FAST PACKET data message (NMEA2000).

Input parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
wChannel : Index of the channel number (obtained by the J19ChannelOpen function)
hj19Msg : Parameters of the message to be emitted

wDataLen	Maximal data length (emitted or recived) [0-223]
bData	Data contain (emission only)

Outputs parameters:
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_MSGSVC	The channel is not a transmission channel
STATUS_ERR_MSGBUSY	The new message cannot be considered. The previous message is being sent.

6.13. J19GetEvent : Reading of an event

Prototype:
tMuxStatus J19GetEvent(unsigned short wCard, unsigned short wBus, tJ19Event *hJ19Event);

Description:
This function retrieves an event from the network.

Input parameters:

wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
hJ19Event : Pointer on the event to be filled in

Outputs parameters:
hJ19Event : Filled in event

wBus	Bus number that generated the event
wChannel	Channel number that generated the event [0-n] or 0xFFFF if event from the spy mode.
eTypeEvent	Event type EVENT_EMPTY : No events are present EVENT_J19_MSGTX : - End of message's correct transmission EVENT_J19_MSGRX : - Correct message's reception EVENT_J19_MSGTXERR : - Incorrect end of transmission EVENT_J19_MSGRXERR - Incorrect reception EVENT_J19_MSGRXFF - RTS frame reception indication EVENT_J19_MSGTXFF - RTS en of transmission indication EVENT_FIFO_OVF - Bit showing that one or several events after this one were lost (due to a full reception FIFO).
dwIdent	CAN identifier of the transmission message [0x1FFFFFFF]
eTypeId	Identifier type CAN_ID_XTD : Extended identifier (29 bits)
dwTimeStamp	Arrival time of the event as a multiple of 100 µSec (1 ms accuracy for PCI-MUX cards). This parameter is correct if wTimePrecision is null See ANNEX : Timestamp of PCI-MUX cards
wTimePrecision	Time accuracy of dwTimeStamp as a multiple of 100 µSec.
eService	Request's service J19_SVC_TRANSMIT_DATA : Data transmission J19_SVC_RECEIVE_DATA : Data reception.
eError	Error type indication for EVENT_J19_MWGTXERR or EVENT_J19_MSGRXERR events: - J19_NO_ERR : No error - J19_ERR_TR : Time out for the transmission of a message (sender side) - J19_ERR_TH : Time out for sending a maintain communication's message - J19_ERR_T1 : Timeout waiting for another TP.DT. frame

	- J19_ERR_T2 : Timeout waiting for a first TP.DT. frame - J19_ERR_T3 : Timeout waiting for a first CTS frame - J19_ERR_T4 : Timeout waiting for another CTS frame
dwPGN	PGN Number
bSA	Source address
dwReserved1	Reserved for a future use
dwReserved2	Reserved for a future use
wDataLen	Data length [0-8]
bData	Data buffer

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

Example :

```
/* Event indication */
tJ19Event hJ19Event ;
tMuxStatus Status ;
unsigned short wCard=0,wBus=0 ;

Status=J19GetEvent(wCard,wBus,&hJ19Event);
if (Status != STATUS_OK) return ;
switch(hJ19Event.eTypeEvent &~ EVENT_FIFO_OVF)
{
        case EVENT_EMPTY :              /* Plus d'événement à traiter */
                break ;
        case EVENT_J19_MSGTX :          /* Fin de transmission correcte */
                break ;
        case EVENT_ J19_MSGRX :         /* Réception correcte */
                break ;
        case EVENT_ J19_MSGTXERR :      /* Fin de transmission en erreur */
                break ;
        case EVENT_ J19_MSGRXERR :      /* Réception en erreur*/
                break ;
        case EVENT_ J19_MSGRXFF :       /* Réception d'un RTS*/
                break ;
        case EVENT_ J19_MSGTXFF :       /* Fin de transmission d"un RTS */
                break ;
        default  :                      /* Réservé pour utilisation future */
                break ;
}
```

6.14. J19GetFifoRxLevel : Fill level of the events queue

Prototype:
tMuxStatus J19GetFifoRxLevel (unsigned short wCard, unsigned short wBus, unsigned short wParam, unsigned short *wCount, unsigned short *wMaxCount);

Description :
This function allows reading the fill level of the queue of events sent back to the application.

Input parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
wParam : Function parameters
Bit 0 : Type of queue (only if USB interface)
- 0 : The quantity of events sent back corresponds to this placed in the PC side queue
- 1 : Reserved

Output parameters:
wCount : Quantity of events currently queued
wMaxCount : Quantity of events that can hold up the queue

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

6.15. J19StoreNMEAFastPacket : FastPacket reception

Prototype:
tMuxStatus J19StoreNMEAFastPacket(unsigned short wCard, unsigned short wBus, unsigned long dwIdent, unsigned short wSize);

Description :
This function allows you to rebuild a FAST PACKET (NMEA2000) frame according to its identifier and size.

Input parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
dwIdent : Frame identifier
wSize : Size of the frame to rebuild

Output parameters:

None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

6.16. J19ChannelClose : Closing a communication channel

Prototype:
tMuxStatus J19ChannelClose(unsigned short wCard, unsigned short wBus, unsigned short wChannel);

Description :
This function closes a communication channel.

Input parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
wChannel : Number of the channel to be closed

Output parameters:
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

6.17. J19SetNotification : Declaration of the application event

Prototype:
tMuxStatus J19SetNotification (unsigned short wCard, unsigned short wBus, HANDLE hWinEvent);

Description :
This function allows the application to use the task synchronisation methods provided by the Windows operation systems (API WIN32). This function allows transference of an event handle created by the Windows function « CreateEvent » to the DLL. This event is used

during communication when an event is transferred from the DLL to the application (for example: end of transmission ending, reception, an error…).
From then on, the application can wait for events through waiting functions such as WaitForSingleObject or WaitForMultipleObject.

Example:

```
/* Request of event's indication */
HANDLE hWinEvent ;
tJ19Event hJ19Event ;
tMuxStatus Status ;
unsigned short wCard=0,wBus=0 ;

hWinEvent=CreateEvent(NULL,FALSE,FALSE,NULL) ;
if (hWinEvent == NULL) return ;
Status=J19SetNotification (wCard,wBus,hWinEvent) ;
if (Status != STATUS_OK) return ;
Status=J19Activate(wCard,wBus) ;
if (Status != STATUS_OK) return ;
if (WaitForSingleObject(hWinEvent,INFINITE) == WAIT_OBJECT_0)
{
        J19GetEvent(wCard,wBus,&hJ19Event);
}
```

Input parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
hWinEvent : « handle » sent back by the CreateEvent function.

Output parameters:
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

7. ISO library – 14230 (KWP2000)

7.1. Order of cal

The application must respect the sequencing according to the functions calling order.

X : means that the function has been authorised in the present status.
X ⇒ : means that the function has been authorised in the present status and moves onto the next status.

Status	ISO_INIT	ISO_OPER	ISO_BUS	ISO_START
IsoConfigOper	X ⇒			
IsoConfigBus		X ⇒		
IsoActivate			X ⇒	
IsoDeactivate	X	X	X	X ⇒
IsoSetNotification	X	X	X	
IsoConfigParam		X		
IsoConfigStat	X	X	X	X
IsoConfigPeriodic		X	X	X
IsoSendMsg				X
Iso14230SendMsg				X
IsoGetEvent				X
IsoWaitResponse				X
IsoChangeBaudRate				X
IsoGetStat				X
IsoGetFifoRxLevel	X	X	X	X

7.2. IsoConfigOper : Routines mode of operation

Prototype :
tMuxStatus IsoConfigOper(unsigned short wCard, unsigned short wBus, tIsoOper *hIsoOper);

Description :
This function determines the interface mode between the application and the card.

Inputs parameters:
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
hIsoOper : Type of interface with the application

eIsoOperMode	Operating mode : ISO_OPER_TESTER - Tester mode: for simulate a diagnostic tool. ISO_OPER_ANALYZER - Analysis mode: Spy application for analyse communication between a tester and an ECU. ISO_OPER_ SIMU - ECU mode: Application will work like an ECU. (5 bauds init is not supported in this mode)
wFifoSize	Reserved for future use

Output parameters:
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

7.3. IsoConfigBus : Configuration des paramètres du bus

Prototype:
tMuxStatus IsoConfigBus(unsigned short wCard, unsigned short wBus, tIsoBus *hIsoBus);

Description:
This function can configure all the parameters of the bus.

Inputs parameters:
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
hIsoBus: Bus parameter

5 bauds initialization parameters:

wW0	Bus idle time before sending an initialization frame (Only for tester mode). Recommended value: 300ms
wW1	Time between the end of the address byte and the beginning of the synchronization pattern (Only for tester mode) Recommended value: 300ms
wW2	Time between the end of the synchronization pattern and the beginning of key byte 1 (Only for tester mode) Recommended value: 20ms
wW3	Time between key byte 1 and key byte 2 (Only for tester mode).

	Recommended value: 20ms
wW4a	Time between key byte 2 (coming from the UCE) and its inversion, carried out by the MuxTrace (Only for tester mode) Recommended value: 25ms
wW4b	Time between inverted key byte 2 and the inverted address coming from the UCE (Only for tester mode) Recommended value: 50ms
wW5	Time between two initialization (Only for tester mode) Recommended value: 5ms
WP0	Time between reception of the inverted address and the beginning of the emission of the StartCommunication request (Only for tester mode). Recommended value: 5ms
eParity	Parity code address for 5 bauds init - ISO_PARITY_NOCHANGE : No parity - ISO_PARITY_ODD : odd-parity - ISO_PARITY_EVEN : even-parity

Fast init parameters:

wTIdle	The tool waits at least 300 ms (w5) of line inactivity before sending the first WakeUp Pattern or to repeat a new WakeUp Pattern when the ECU is out of diagnostic Recommended value: 300ms
wTInitL	Low time after an idle time Recommended value: 25ms
wTWup	If there is no reception or an error during the "Start Communication" reception the ECU has to come back in waiting for the WakeUp Pattern before the end of the Tidle timing. Recommended value: 50ms

Communication parameters:

eBaudRate	ISO_BAUD_9600 : 9600 bauds ISO_BAUD_10400 : 10400 bauds ISO_BAUD_62500 : 62500 bauds
wP1	Inter-byte delay of the ECU's response.
wP2	Time-out between a tester request and the ECU's response in ms.
wP3	Delay between a response from the ECU and a new request from the tester in ms.
wP4	Inter-byte delay of the tester's request.

Outputs parameters :
None

Return code:

Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

7.4. IsoConfigParam : Configuration of supplementary parameters

Prototype :
tMuxStatus IsoConfigParam(unsigned short wCard, unsigned short wBus, tIsoParam *hIsoParam);

Description :
This function allows configuration of additional operational parameters.

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
hIsoParam : Auxiliary operation parameters

wFiltTP	Allow to filter “Tester Present” frames : 0 : no Tester Present filter 1 : Tester Present activated (the events don't go up to the application)
wSpecialModes	Reserved for future use

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

7.5. IsoConfigStat : Configuration of statistics

Prototype :
tMuxStatus IsoConfigStat(unsigned short wCard, unsigned short wBus, unsigned short wBusLoadTime);

Description :
This function allows configuration of the duration on which the bus charge is calculated. Duration of 0 inhibits the charge calculation.

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
wBusLoadTime : This parameter defines the duration on which the bus charge is calculated. It is expressed in ms. Value: 0 indicates that no bus charge has been transferred to the application. The bus charge is transferred in FIFO mode by means of the event EVENT_ISO_BUSLOAD and of the wBusLoad parameter (0 by default)

Outputs parameters:
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

7.6. IsoSetNotification : Declaration of the application event

Prototype :
tMuxStatus IsoSetNotification (unsigned short wCard, unsigned short wBus, HANDLE hWinEvent);

Description :
This function allows the application to use the task synchronisation methods provided by the Windows operation systems (API WIN32). This function allows transference of an event handle created by the Windows function « CreateEvent » to the DLL. This event is used during communication when an event is transferred from the DLL to the application (for example: end of transmission ending, reception, an error…).
From then on, the application can wait for events through waiting functions such as WaitForSingleObject or WaitForMultipleObject.

Example:
```
/* Request for indication of event */
HANDLE hWinEvent ;
tIsoEvent hIsoEvent ;
tMuxStatus Status ;
unsigned short wCard=0,wBus=0 ;

hWinEvent=CreateEvent(NULL,FALSE,FALSE,NULL) ;
```

```
if (hWinEvent == NULL) return ;
Status=IsoSetNotification (wCard,wBus,hWinEvent) ;
if (Status != STATUS_OK) return ;
Status=IsoActivate(wCard,wBus) ;
if (Status != STATUS_OK) return ;
if (WaitForSingleObject(hWinEvent,INFINITE) == WAIT_OBJECT_0)
{
        IsoGetEvent(wCard,wBus,&hIsoEvent);
}
```

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
hWinEvent : « handle » indicated by the CreateEvent function.

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

7.7. IsoActivate : Starting communication

Prototype :
```
tMuxStatus IsoActivate(unsigned int wCard, unsigned int wBus);
```

Description :
This function allows communication with the network to start. After executing this function, the messages received are acknowledged and sent towards the application.
The application can also send messages.

Inputs parameters
wCard : Index of card number to be accessed
wBus : Index of bus number

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

7.8. IsoDeactivate : Stopping communication

Prototype :
tMuxStatus IsoDeactivate(unsigned int wCard, unsigned int wBus);

Description :
This function stops communication with the network. After executing this function, messages are no longer received, acknowledged or sent.

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

7.9. IsoConfigPeriodic: Programming a periodic message

Prototype :
tMuxStatus IsoConfigPeriodic(unsigned short wCard, unsigned short wBus, unsigned short wOffset, unsigned short wParam, tIsoMsg *hIsoMsg);

Description :
This function allows the user to start, stop and update the emission of periodic messages. This function is often used within the framework of the management of maintenance of the communication (Tester Present frame).

Inputs parameters:
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
wOffset : Index of periodic message [0-15]

The application is composed of 16 periodic messages. This parameter corresponds to the message index the application wants to access. Note that the 16 messages are dealt with independently.

wParam : Message parameter
Bit 0 : Indicator of periodic message in operation
- 0 : The periodic message is stopped
- 1 : The periodic message is activated (start of periodicity or update)

hIsoMsg : Parameter of message to be transmitted

eTypeMsg	ISO_MSG_NORMAL: Emission of a request ISO_MSG_INIT5BDS: Emission of a request preceded by a sequence of intialisation with 5 bauds ISO_MSG_INITFAST: Emission of a request preceded by a fast sequence of initialization
wCodeAddr	Value of the code addresses initialization to 5 bauds
LPeriod	Periodicity of the message in ms
wDataLen	Maximum length of dada (sent or received) [0-254]
bData	Content of data (for transmitting)

Outputs parameters:
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

7.10. IsoSendMsg: Sending a message

Prototype :
tMuxStatus IsoSendMsg(unsigned short wCard, unsigned short wBus, tIsoMsg *hIsoMsg);

Description :
This function allows as a tester mode:
- the emission of a request and waiting of an answer

This function allows in what if mode:
- the emission of an answer.

This function is inactive in analyzer mode

This function allows the emission of raw data. Data header and CRC must be managed by the application.

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
hIsoMsg : Parameter of message to be sent

eTypeMsg	ISO_MSG_NORMAL: Emission of a request ISO_MSG_INIT5BDS: Emission of a request preceded by a sequence of intialisation with 5 bauds ISO_MSG_INITFAST: Emission of a request preceded by a fast sequence of initialization ISO_MSG_WAITRESP: wait a response without emitting a request (for slave application or/and multi-request in tester mode).
wCodeAddr	Value of the code addresses initialization to 5 bauds
LPeriod	Periodicity of the message in ms
wDataLen	Maximum length of dada (sent or received) [0-254]
bData	Content of data (for transmitting)

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_FIFOFULL	The new message cannot be taken into account. The transmitting FIFO is full.
STATUS_ERR_MSGSVC	Non supported service

7.11. Iso14230SendMsg: Sending a message

Prototype :
tMuxStatus Iso14230SendMsg(unsigned short wCard, unsigned short wBus, tIso14230Msg *hIso14230Msg);

Description :
This function allows as a tester mode:
- the emission of a request and waiting of an answer

This function allows in what if mode:
- the emission of an answer.

This function is inactive in analyzer mode

This function is identical to the IsoSendMsg function. This function automatically generates the header bytes and checksum of protocol KWP2000.

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
hIso14230Msg: Parameter of message to be sent

eTypeMsg	ISO_MSG_NORMAL: Emission of a request ISO_MSG_INIT5BDS: Emission of a request preceded by a sequence of intialisation with 5 bauds ISO_MSG_INITFAST: Emission of a request preceded by a fast sequence of initialization
eIsoTypeHeader	Format of the bytes of heading : ISO_HEADER_NOADDR: No information of address ISO_HEADER_CARB: Mode of exception CARB ISO_HEADER_PHYS: header bytes with physical information of addressing ISO_HEADER_FUNCT: header bytes with information of functional addressing
wCodeAddr	Value of the code addresses initialization to 5 bauds
LPeriod	Periodicity of the message in ms
wSrcAddr	Source address
wDstAddr	Target address
wDataLen	Maximum length of dada (sent or received) [0-254]
bData	Content of data (for transmitting)

Outputs parameters :
None

Return code:
Status : Summary of the function execution

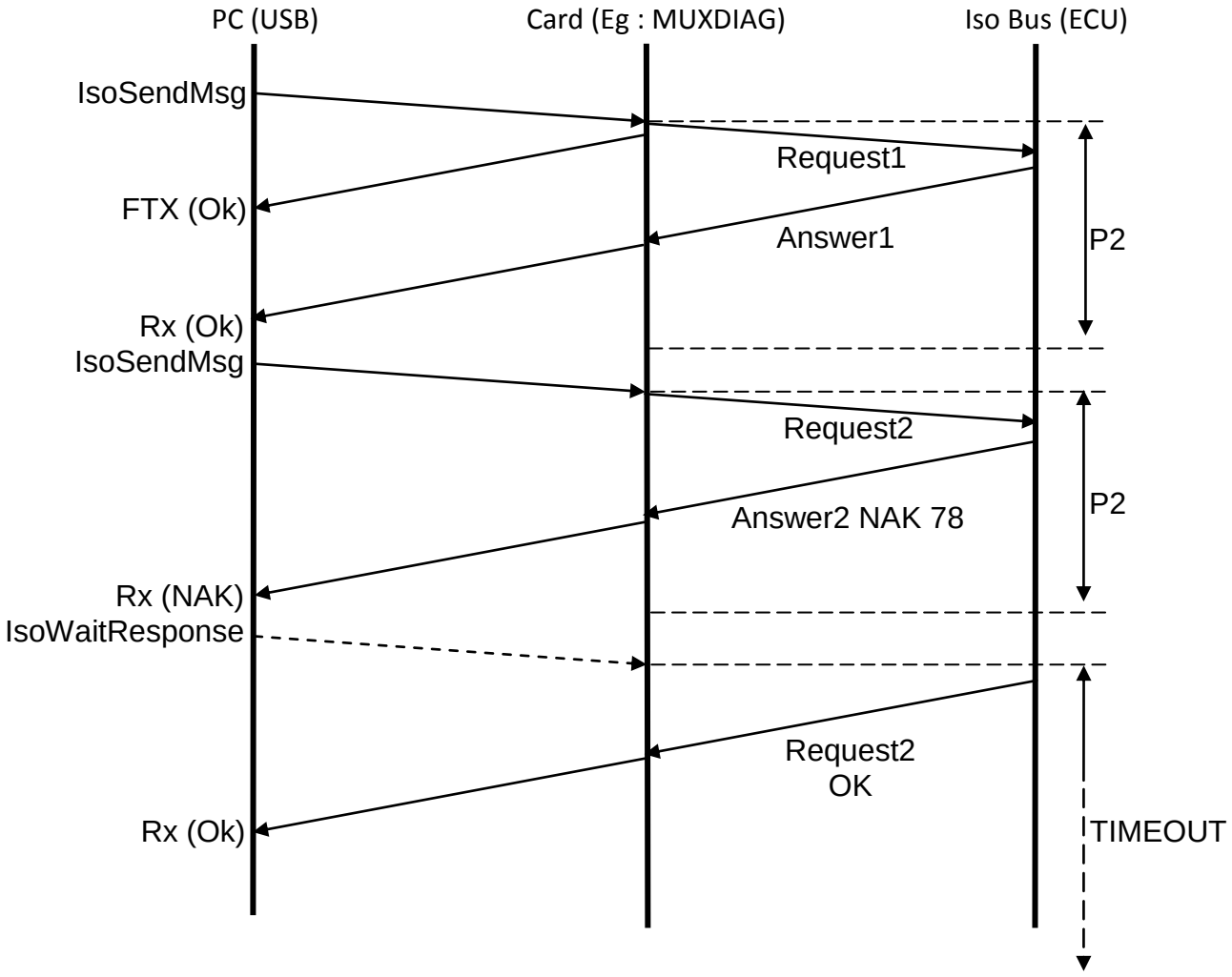
STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_FIFOFULL	The new message cannot be taken into account. The transmitting FIFO is full.
STATUS_ERR_MSGSVC	Non supported service

7.12. IsoWaitResponse : Waiting a new answer

Prototype :
tMuxStatus IsoWaitResponse(unsigned short wCard, unsigned short wBus, unsigned short wTimeOut);

Description :
This function used with tester mode makes it possible to start again waiting of a response for the management of the long requests.

Indeed, the execution time of certain requests can exceed P2 time. To maintain the communication, the calculator sends a negative response with the code 0x78 (correctly received request, response on standby). After reception of this answer, the application can called the function "IsoWaitResponse" to start again waiting of a new response without emitting of new request.



Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
wTimeOut : Answer timeout

Outputs parameters :
None

Return code:

Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_FIFOFULL	The new message cannot be taken into account. The transmitting FIFO is full.

7.13. IsoChangeBaudRate : change baud rate

Prototype :
tMuxStatus IsoChangeBaudRate(unsigned short wCard, unsigned short wBus, tIsoBaudRate eBaudRateTx, tIsoBaudRate eBaudRateRx);

Description :
This function makes it possible to change the bit rate of communication during the application running.

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
eBaudRateTx : New bit rate.
eBaudRateRx : Reserved

Note: To refer to the IsoConfigBus function to obtain the list of the authorized bit rate.

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

7.14. IsoGetEvent: Reading of an event

Prototype :
tMuxStatus IsoGetEvent(unsigned short wCard, unsigned short wBus, tIsoEvent *hIsoEvent);

Description :
This function allows the user to recover an event from the network.

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
hIsoEvent : Pointer on the event to complete

Outputs parameters :
hIsoEvent : Event complete

wBus	Number of bus managing the event
wHandleMsg	Reserved for future use
eTypeEvent	Type of event EVENT_EMPTY : No event. EVENT_ISO_MSGTX : - End of correct transmission of a message EVENT_ISO_MSGRX : - Correct reception of a message EVENT_ISO_MSGTXERR: - End of transmission in error EVENT_ISO_MSGRXERR: - Reception in error EVENT_ISO_ERROR: - Reception of a character in error (framing error, station-wagon...) EVENT_ISO_BUSLOAD - Bus charge EVENT_TIMER - Applicable timer EVENT_TIMEERROR - Loss of IT timer EVENT_FIFO_OVF - Bit indicating that one or several events after such bit are lost because the FIFO reception was full.
dwTimeStamp	Arrival time of the event in a multiple of 100 µSec (1 ms precision for PCI-MUX cards). This parameter is correct if wTimePrecision is zero. C.F. ANNEXE : Time stamp indication with PCI-MUX cards
wTimePrecision	Precision of dwTimeStamp in multiple of 100 µSec.
eError	Indication of type of error : - For the events of the type EVENT_ISO_ERROR: ISO_ERR_FE: Error of bit of stop ISO_ERR_PE: Error of bit of parity ISO_ERR_OE: Char buffer overflow ISO_ERR_BREAK: Break detection - For the events of the type EVENT_ISO_MSGTXERR or EVENT_ISO_MSGRXERR: ISO_ERR_NOECHO: Emitted character not received in echo

	ISO_ERR_BADECHO: Incorrect echo ISO_ERR_TO_SYNCHRO: “Synchro field” Timeout ISO_ERR_BAD_SYNCHRO: Incorrect value of Synchro Field ISO_ERR_TO_KEY1: key word 1 timeout ISO_ERR_BAD_KEY1: Value of the key word 1 Incorrect ISO_ERR_TO_KEY2: key word 2 Timeout ISO_ERR_BAD_KEY2: Value of the key word 2 incorrect ISO_ERR_TO_ADDR: reversed address timeout ISO_ERR_BAD_ADDR: Incorrect value of the reversed address ISO_ERR_TO_RESP: Answer Timeout ISO_ERR_RX_OVER: Length of answer too long ISO_ERR_BAD_LEN: Incorrect length of the answer ISO_ERR_BAD_CRC: Incorrect checksum of the answer
wBusLoad	For an event of the EVENT_ISO_BUSLOAD type, value of bus charge in %.
dwReserved1	Reserved for future use
dwReserved2	Reserved for future use
wDataLen	Length of data [0-254]
bData	Data buffer

Return code:

Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

Example :

```
/* Indication of event */
tIsoEvent hIsoEvent ;
tMuxStatus Status ;
unsigned short wCard=0,wBus=0 ;

Status=IsoGetEvent(wCard,wBus,&hIsoEvent);
if (Status != STATUS_OK) return ;
switch(hIsoEvent.eTypeEvent &~ EVENT_FIFO_OVF)
{
        case EVENT_EMPTY :              /* No more events to be dealt with */
                break ;
        case EVENT_ISO_MSGTX :          /* Correct end of transmission */
                break ;
        case EVENT_ISO_MSGRX:           /* Correct reception */
                break ;
        case EVENT_ISO_MSGTXERR :       /* End of frame error*/
                break ;
        case EVENT_ISO_MSGRXERR :       /* Receipt Error */
```

```
            break ;
        case EVENT_ISO_ERROR:           /* Bit Error*/
            break ;
        case EVENT_ISO_BUSLOAD :        /* Bus load*/
            break ;
        case EVENT_TIMER :              /* Timer*/
            break ;
        case EVENT_TIMEERROR :          /* Loss IT timer */
            break ;
        default  :                      /* Reserved for future use */
            break ;
}
```

7.15. IsoGetStat : Reading statistics counters

Prototype :
tMuxStatus IsoGetStat(unsigned short wCard, unsigned short wBus, tIsoStat *hIsoStat);

Description :
This function allows the user to read the network's operational counters.

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
hIsoStat : Pointer on the statistic area to be completed

Outputs parameters :
hIsoStat : Event to be completed

wBusLoad	Network charge expressed in percentage
wReserved	Reserved for future use
dwTxRq	Number of transmission requests
dwTxOk	Number of correct transmission endings
DwTxErr	Number of ends of transmissions in error
dwRxOk	Correct reception
DwRxErr	Number of receptions in error
DwErrBreak	Number of break field in error
DwErrFE	Number of format error
DwErrPE	Number of parity error
DwErrOE	Number of overflow error
DwErrFifoOvf	Number of message lost (Receipt FIFO is full)

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

7.16. IsoGetFifoRxLevel : Level of filling of events waiting list

Prototype :
tMuxStatus IsoGetFifoRxLevel (unsigned short wCard, unsigned short wBus, unsigned short wParam, unsigned short *wCount, unsigned short *wMaxCount);

Description :
This function allows the user to see how full the waiting list of events transferred to the application is.

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
wParam : Function's parameters
Bit 0 : Type of waiting list (only for USB cases)
- 0 : Event queue is situated on the PC.
- 1 : Reserved for future use

Outputs parameters :
wCount : Number of event currently on waiting list
wMaxCount : Maximum number of events accepted on the waiting list

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

7.17. IsoIsBusActive : Bus status

Prototype:
tMuxStatus IsoIsBusActive(unsigned short wCard, unsigned short wBus, unsigned short *wState);

Description :
This function allows to know if the bus has been activated using the IsoActivate function.

Inputs parameters :

wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]

Outputs parameters :
wState: Indicates if the bus is activated or not (value different from 0)

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

8. LIN Library

8.1. Order of call

The application must respect the sequencing according to the functions calling order.

X : means that the function has been authorised in the present status.
X ⇒ : means that the function has been authorized in the present status and moves onto the next status.

Status	LIN_INIT	LIN_OPER	LIN_BUS	LIN_START
LinConfigOper	X ⇒			
LinConfigBus		X ⇒		
LinConfigUart			X	
LinActivate			X ⇒	
LinDeactivate	X	X	X	X ⇒
LinSetNotification	X	X	X	
LinConfigParam		X		
LinConfigStat	X	X	X	X
LinSetVersion	X	X	X	X
LinConfigTransceiver	X	X	X	X
LinConfigPeriodic		X	X	X
LinConfigPeriodicList		X	X	X
LinSendMsg				X
LinSendMsgList				X
LinGetEvent				X
LinGetBusState				X
LinSetSleepMode				X
LinSetWakeUpMode				X
LinGetStat				X
LinClearBufferIFR	X	X	X	X
LinGetFifoRxLevel	X	X	X	X

8.2. LinConfigOper : Routines' operation mode

Prototype :
tMuxStatus LinConfigOper(unsigned short wCard, unsigned short wBus, tLinOper *hLinOper);

Description :

This function determines the interface mode between the application and the card.

Inputs parameters:
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
hLinOper : Type of interface with the application

eLinOperMode	Operating mode : LIN_OPER_ANA_FIFO - Analysis mode, FIFO storage
wFifoSize	Reserved for future use

- FIFO storage: in this mode, the events to be transferred to the application are stocked in a waiting list. These events are of the following types : end of transmission, reception, errors … When the application calls the LinGetEvent function, the first event (the oldest in time) comes off the list.
  If the waiting list is full and an event takes place, then a bit indicating loss of event is placed on the last event.

Output parameters:
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

8.3. LinConfigBus : Configuration of bus parameters

Prototype :
tMuxStatus LinConfigBus(unsigned short wCard, unsigned short wBus, tLinBus *hLinBus);

Description :
This function can configure all the parameters of the bus.

Inputs parameters:
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
hLinBus : Bus parameter

eBaudrate	LIN_BAUD_2400 /* Baud rate 2400 bauds */ LIN_BAUD_9600 /* Baud rate 9600 bauds */

LIN_BAUD_19200 /* Baud rate 19200 bauds */

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

8.4. LinConfigUart : Configuration of bus advanced settings

Prototype :
tMuxStatus LinConfigUart(unsigned short wCard, unsigned short wBus, tUartConfig *hUartConfig);

Description :
This function enables configuration of bus advanced settings.

Input parameters:
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
hUartConfig : Bus advanced settings

eParity	NO_PARITY: - No parity bit ODD_PARITY: - Odd parity EVEN_PARITY: - Even parity
eStopBit	ONE_STOP_BIT : - 1 stop bit TWO_STOP_BIT : - 2 stop bits
eDataBit	FIVE_DATA_BIT : - 5 data bits SIX_DATA_BIT : - 6 data bits SEVEN_DATA_BIT : - 7 data bits EIGHT_DATA_BIT : - 8 data bits NINE_DATA_BIT :

- 9 data bits

dwBaudrate	Real bus baud rate in bit/s
wError	Baud rate precision in ‰ (an error is indicated if it is not possible to get baud rate within indicated tolerance)

Output parameters:
None

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_NOT_SUPPORTED	This type of material is not supported

8.5. LinConfigParam : Configuration of supplementary parameters

Prototype :
tMuxStatus LinConfigParam(unsigned short wCard, unsigned short wBus, tLinParam *hLinParam);

Description :
This function allows configuration of additional operational parameters.

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
hLinParam : Auxiliary operation parameters

eDefDelayBusIdle	Default value for detect a timeout on the bus LIN_TRUE : (default value) - This time is calculated by the libraries in conformity with LIN specification. LIN_FALSE : - User defines the bus idle time (dwDelayBusIdle)
dwDelayBusIdle	Value in ms of time of detection of the absence of communication on the bus (event EVENT_LIN_BUSCHANGE)
wSpecialModes	Bit 0: Incoming message handling indicator - 0 : Size of messages is determined by identifier (LIN 1.1 and 1.2) - 1 : Identifier is not used to determine size of message (compatibility for LIN 1.3 and 2.x)

Bit 1 to 15: Reserved for future use

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

8.6. LinConfigStat : Configuration of statistics

Prototype :
tMuxStatus LinConfigStat(unsigned short wCard, unsigned short wBus, unsigned short wBusLoadTime);

Description :
This function allows configuration of the duration on which the bus charge is calculated. Duration of 0 inhibits the charge calculation.

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
wBusLoadTime : This parameter defines the duration on which the bus charge is calculated. It is expressed in ms. Value: 0 indicates that no bus charge has been transferred to the application. The bus charge is transferred in FIFO mode by means of the event EVENT_LIN_BUSLOAD and of the wBusLoad parameter (0 by default)

Outputs parameters:
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

8.7. LinSetVersion : Indication of protocol version

Prototype :

tMuxStatus LinSetVersion(unsigned short wCard, unsigned short wBus, unsigned long dwVersion);

Description :
This function indicates version of LIN standard (the difference between V1 and V2 appears when it comes to calculating CRC).

Input parameters:
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
dwVersion : Requested version in BCD hexadecimal format (ex. 0x0210 for V2.1)

Output parameters:
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

8.8. LinConfigTransceiver: Transceiver configuration

Prototype :
tMuxStatus LinConfigTransceiver(unsigned short wCard, unsigned short wBus, unsigned short wEnable, unsigned short wRMaster);

Description :
This function makes it possible to control the various signals of the LIN transceiver.

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
wEnable : PIN value : « enable » [0-1]
wRMasterWakeUp : Pull-up value
- RMaster = 0 : 30 Ko (slave mode)
- RMaster = 1 : 1 Ko (master mode)

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_WARNING	Not supported on this type of material

8.9. LinSetNotification : Declaration of the application event

Prototype :
tMuxStatus LinSetNotification (unsigned short wCard, unsigned short wBus, HANDLE hWinEvent);

Description :
This function allows the application to use the task synchronization methods provided by the Windows operation systems (API WIN32). This function allows transference of an event handle created by the Windows function « CreateEvent » to the DLL. This event is used during communication when an event is transferred from the DLL to the application (for example: end of transmission ending, reception, an error…).
From then on, the application can wait for events through waiting functions such as WaitForSingleObject or WaitForMultipleObject.

Example :

```
/* Request for indication of event */
HANDLE hWinEvent ;
tLinEvent hLinEvent ;
tMuxStatus Status ;
unsigned short wCard=0,wBus=0 ;

hWinEvent=CreateEvent(NULL,FALSE,FALSE,NULL) ;
if (hWinEvent == NULL) return ;
Status=LinSetNotification (wCard,wBus,hWinEvent) ;
if (Status != STATUS_OK) return ;
Status=LinActivate(wCard,wBus) ;
if (Status != STATUS_OK) return ;
if (WaitForSingleObject(hWinEvent,INFINITE) == WAIT_OBJECT_0)
{
        LinGetEvent(wCard,wBus,&hLinEvent);
}
```

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
hWinEvent : « handle » indicated by the CreateEvent function.

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

8.10. LinActivate : Starting communication

Prototype :
tMuxStatus LinActivate(unsigned int wCard, unsigned int wBus);

Description :
This function allows communication with the network to start. After executing this function, the messages received are acknowledged and sent towards the application.
The application can also send messages.

Inputs parameters
wCard : Index of card number to be accessed
wBus : Index of bus number

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

8.11. LinDeactivate : Stopping communication

Prototype :
tMuxStatus LinDeactivate(unsigned int wCard, unsigned int wBus);

Description :
This function stops communication with the network. After executing this function, messages are no longer received, acknowledged or sent.

Inputs parameters :
wCard : Index of card number to be accessed

wBus : Index of bus number [0-x]

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

8.12. LinConfigPeriodic: Programming a periodic message

Prototype :
tMuxStatus LinConfigPeriodic(unsigned short wCard, unsigned short wBus, unsigned short wOffset, unsigned short wParam, tLinMsg *hLinMsg);

Description :
This function allows the user to start, stop and update the emission of periodic messages.

Inputs parameters:
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
wOffset : Index of periodic message [0-15]
The application is composed of 16 periodic messages. This parameter corresponds to the message index the application wants to access. Note that the 16 messages are dealt with independently.

wParam : Message parameter
Bit 0 : Indicator of periodic message in operation
- 0 : The periodic message is stopped
- 1 : The periodic message is activated (start of periodicity or update)

Bit 1 : Indicates transference of transmissions endings
It is not always necessary for an application to receive the events of transmission endings related to the emission of periodic messages.
- 0 : The transmissions endings are not transferred to the applications
- 1 : The transmissions endings are transferred to the application

hLinMsg : Parameter of message to be transmitted

wHandleMsg	Reserved for future use
wIdent	LIN message identifier [0-0x1F] if data length = 2 bytes [0-0x0F] if data length equal 4 bytes or 8 bytes

eTypeId	Type of identifier CAN_ID_STD : Standard identifier (11 bits) CAN_ID_XTD : Extended identifier (29 bits)
eService	LIN_SVC_TRANSMIT_DATA - Data transmission LIN_SVC_REQUEST_IFR - Request for an In Frame Response LIN_SVC_UPDATE_IFR - Update In Frame Response
lPeriod	Periodicity of the message in ms.
eLevel3Enable	Reserved for future use
dwReserved1	Reserved for future use
eLinGenErr	Possibility of generating a message with a protocol error (values below are exclusive). LIN_GEN_NO_ERR - No error generated LIN_GEN_ERR_P0 - Parity bit error on P0 bit LIN_GEN_ERR_P1 - Parity bit error on P1 bit LIN_GEN_ERR_CRC - CRC error (inverted CRC) LIN_GEN_ERR_SYNCH - Synchro bite error (0xAA instead of 0x55) LIN_GEN_ERR_DATP1 - 1 data byte set to 0xFF is added to the data field LIN_GEN_ERR_DATP2 - 2 data bytes set to 0xFF are added to the data field LIN_GEN_ERR_DATL1 - 1 data byte is cut out of the data field LIN_GEN_ERR_DATP1 - 2 data bytes are cut out of the data field Possibility to modify message properties (the values below are not exclusive). LIN_GEN_FREE_IDENT - Message can be sent without data length being defined by the identifier. The identifier can take a value of between [0 and 63], the data length can take a value of between [0 and 8] bytes LIN_GEN_CLASSIC_CRC - The message's checksum is calculated only from the data field.
dwReserved2	Reserved for future use
wDataLen	Maximum length of dada (sent or received) [0-8]
bData	Content of data (for transmitting)

Outputs parameters:

None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_FIFOFULL	The new message cannot be taken into account. The transmitting FIFO is full.
STATUS_ERR_MSGEXCEED	Wrong message offset
STATUS_ERR_MSGSVC	Not supported service

8.13. LinConfigPeriodicList: Programming a list of periodic messages

Prototype :
tMuxStatus LinConfigPeriodicList(unsigned short wCard, unsigned short wBus, unsigned short wPeriodicCount, tLinPeriodicMsg *hLinPeriodicMsgList);

Description :
This function allows the user to start, stop and update the sending of periodic messages.

Input parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
wPeriodicCount: Number of messages (limited to 16)
hLinPeriodicMsgList : Message board

wOffset	Index of periodic message [0-15]
wParam	Message parameter Bit 0: Indicator of periodic message operation - 0 : Periodic message is stopped - 1 : Periodic message is activated (start of periodicity or update) Bit 1 : Indicates transferrence of transmission endings - 0 : Transmission endings are not transferred to the applications - 1 : Transmission endings are transferred to the applications
hLinMsg	Paremeter of message to be transmitted (cf LinConfigPeriodic)

Output parameters:
None

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_FIFOFULL	The new message cannot be taken into account. Transmission FIFO is saturated.
STATUS_ERR_MSGSVC	Not supported service

8.14. LinSendMsg: Sending a message

Prototype :
tMuxStatus LinSendMsg(unsigned short wCard, unsigned short wBus, tLinMsg *hLinMsg);

Description :
This function allows :
Master case :
- to send a message of data
- to send a message with an In Frame Response

Slave case :
- Update data for an In Frame Response

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
hLinMsg : Parameter of message to be sent

wHandleMsg	Reserved for future use
wIdent	LIN message identifier [0-0x1F] if data length = 2 bytes [0-0x0F] if data length equal 4 bytes or 8 bytes
eTypeId	Type of identifier CAN_ID_STD : Standard identifier (11 bits) CAN_ID_XTD : Extended identifier (29 bits)
eService	LIN_SVC_TRANSMIT_DATA - Data transmission LIN_SVC_REQUEST_IFR - Request for an In Frame Response LIN_SVC_UPDATE_IFR - Update In Frame Response
lPeriod	Reserved for future use
eLevel3Enable	Reserved for future use
dwReserved1	Reserved for future use
eLinGenErr	Possibility of generating a message with a protocol error

(values below are exclusive).
LIN_GEN_NO_ERR
- No generated error

LIN_GEN_ERR_P0
- Parity error on P0 bit

LIN_GEN_ERR_P1
- Parity error on P1 bit

LIN_GEN_ERR_CRC
- CRC error (inverted CRC)

LIN_GEN_ERR_SYNCH
- Synchro byte error (0xAA instead of 0x55)

LIN_GEN_ERR_DATP1
- 1 data byte set to 0xFF is added in the data field

LIN_GEN_ERR_DATP2
- 2 data bytes set to 0xFF are added in the data field

LIN_GEN_ERR_DATL1
- 1 data byte is cut out of the data field

LIN_GEN_ERR_DATP1
- 2 data bytes are cut out of the data field

Possibility to modify message properties (values below are not exclusive).
LIN_GEN_FREE_IDENT
- Message can be sent without data length being defined by the identifier. The identifier can take a value of between [0 and 63], the data length can take a value of between [0 and 8] bytes

LIN_GEN_CLASSIC_CRC
- The message's checksum is calculated only from the data field.

dwReserved2	Reserved for future use
wDataLen	Maximum length of dada (sent or received) [0-8]
bData	Content of data (for transmitting)

Outputs parameters :
None

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_FIFOFULL	The new message cannot be taken into account. The transmitting FIFO is full.
STATUS_ERR_MSGSVC	Non supported service

8.15. LinSendMsgList: Sending a list of messages

Prototype :
tMuxStatus LinSendMsgList(unsigned short wCard, unsigned short wBus, unsigned short wMsgCount, tLinMsg *hLinMsgList);

Description :
This function allows:
Master case to:
- send one or several data messages
- to send one or several messages with an In Frame Response

This function allows:
Slave case to:
- update data for one or several In Frame Responses.

Input parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
wMsgCount: Number of messages (limited to 32)
hLinMsgList : Message board (cf LinSendMsg for details)

Output parameters:
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_FIFOFULL	The new message cannot be taken into account. Transmission FIFO is saturated.
STATUS_ERR_MSGSVC	Not supported service

8.16. LinGetEvent: Reading of an event

Prototype :
tMuxStatus LinGetEvent(unsigned short wCard, unsigned short wBus, tLinEvent *hLinEvent);

Description :
This function allows the user to recover an event from the network.

Inputs parameters :
wCard : Index of card number to be accessed

wBus : Index of bus number [0-x]
hLinEvent : Pointer on the event to complete

Outputs parameters : :
hLinEvent : Event complete

wBus	Number of bus managing the event
wHandleMsg	Reserved for future use
eTypeEvent	Type of event EVENT_EMPTY : No event. EVENT_LIN_MSGTX : - Correct end of transmission of a message EVENT_LIN_MSGRX : - Correct reception of a message EVENT_LIN_MSGTXERR : - End of transmission error EVENT_LIN_MSGRXERR : - Message received with an error EVENT_LIN_BUSCHANGE - Change in the protocol controller status EVENT_LIN_BUSLOAD - Bus charge EVENT_TIMER - Applicable timer EVENT_TIMEERROR - Loss of IT timer EVENT_FIFO_OVF - Bit indicating that one or several events after such bit are lost because the FIFO reception was full.
wIdent	LIN message identifier [0-0x1F] if data length = 2 bytes [0-0x0F] if data length equal 4 bytes or 8 bytes
dwTimeStamp	Arrival time of the event in a multiple of 100 µSec (1 ms precision for PCI-MUX cards). This parameter is correct if wTimePrecision is zero. C.F. ANNEXE : Time stamp indication with PCI-MUX cards
wTimePrecision	Precision of dwTimeStamp in multiple of 100 µSec.
eService	LIN_SVC_TRANSMIT_DATA - Data transmission LIN_SVC_REQUEST_IFR - Request for an In Frame Response LIN_SVC_UPDATE_IFR Update In Frame Response
wError	For events of EVENT_LIN_MSGTXERR type or EVENT_LIN_MSGRXERR type, indication of type of error : - LIN_ERR_BIT: Received byte different from the emitted byte

- LIN_ERR_CRC : CRC error
- LIN_ERR_TIMEOUT : No answer of the slave
- LIN_ERR_PARITY : Error of parity in the identifying field
- LIN_ERR_SYNCHRO : Synchro field error
- LIN_ERR_LENGTH : Data length error
- LIN_ERR_TO_TX : Time out in transmission
- LIN_ERR_TO_SYNCHRO : Time out in receipt of the synchro byte
- LIN_ERR_TO_IDENT : Time out in receipt of the identifier byte
- LIN_ERR_TO_DATA : Time out in receipt of data
- LIN_ERR_TO_CRC : out in receipt of CRC field

eChipState	Status of statistic counters : - LIN_BUS_NOMINAL : Nominal state (the error counters in transmission and reception are lower than 64) - LIN_BUS_ERROR : Degraded state (the error counters in transmission or reception are higher than 64) - LIN_BUS_IDLE : Bus idle state (Absence of communication)
wBusLoad	For an event of the EVENT_LIN_BUSLOAD type, value of bus charge in %.
dwReserved1	Reserved for future use
dwReserved2	Reserved for future use
wDataLen	Length of data [0-8]
bData	Data buffer

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

Example :

```
/* Indication of event */
tLinEvent hLinEvent ;
tMuxStatus Status ;
unsigned short wCard=0,wBus=0 ;

Status=LinGetEvent(wCard,wBus,&hLinEvent);
if (Status != STATUS_OK) return ;
switch(hLinEvent.eTypeEvent &~ EVENT_FIFO_OVF)
{
        case EVENT_EMPTY :              /* No more events to be dealt with */
```

```
            break ;
        case EVENT_LIN_MSGTX :          /* Correct end of transmission */
            break ;
        case EVENT_LIN_MSGRX :          /* Correct reception */
            break ;
        case EVENT_LIN_MSGTXERR :       /* End of transmission error */
            break ;
        case EVENT_LIN_MSGRXERR :       /* Receipt error */
            break ;
        case EVENT_LIN_BUSCHANGE :      /* Change in the component's status */
            break ;
        case EVENT_LIN_BUSLOAD :        /* Bus charge */
            break ;
        case EVENT_TIMER :              /* Applicable timer */
            break ;
        case EVENT_TIMEERROR :          /* Loss IT timer */
            break ;
        default  :                      /* Reserved for future use */
            break ;
}
```

8.17. LinGetStat : Reading statistics counters

Prototype :

```
tMuxStatus LinGetStat(unsigned short wCard, unsigned short wBus, tLinStat *hLinStat);
```

Description :

This function allows the user to read the network's operational counters.

Inputs parameters :

wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
hLinStat : Pointer on the statistic area to be completed

Outputs parameters :

hLinStat : Event to be completed

wBusLoad	Network charge expressed in percentage
wReserved	Reserved for future use
dwTxRq	Number of transmission requests
dwTxOk	Number of correct transmission endings
dwRxOk	Correct reception
dwErrBit	Number of Bit type of errors
dwErrCrc	Number of CRC errors
dwErrTimeOut	Number of timeout waiting an In Frame Response

dwErrSynchro	Number of Synchro byte errors
dwErrIdent	Number of identifier byte errors
dwErrOther	Number of other errors
dwErrFifoOvf	Number of lost messages (full reception FIFO)

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

8.18. LinGetBusState : Reading of the communication state

Prototype :
tMuxStatus LinGetBusState(unsigned short wCard, unsigned short wBus, tCanChipState *eCanChipState, unsigned char *bTxErrCount, unsigned char *bRxErrCount);

Description :
This function allows the user to directly read the status of LIN error counters. There is one transmission error counter and one receipt error counter. When an error occurs, one of the counter (switch Tx or Rx Message) is incremented of 8. When an message is correctly received, one of the counter (switch Tx or Rx Message) is decrease of 1. When one of the two counters reaches value 64 then the state of the communication passes from the nominal mode to the degraded mode. Conversely if the two counters are lower than 64 then the communication becomes again nominal.

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]
eLinChipState: Pointer on the status to be completed
bTxErrCount: Pointer on the counter to be completed
bRxErrCount: Pointer on the counter to be completed

Outputs parameters : :
eLinChipState: Status of component
bTxErrCount: Transmission counter
bRxErrCount: Reception counter

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

8.19. LinSetSleepMode: Send a master request frame for force all slaves into sleep mode

Prototype :
tMuxStatus LinSetSleepMode(unsigned int wCard, unsigned int wBus);

Description :
This function allows a master to send a request which force all slaves into sleep mode. This function sends a request frame (frame identifier = 0x3c) with data length equal 8 and with the first data byte equal to zero.

Inputs parameters:
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]

Outputs parameters :
None.

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

8.20. LinSetWakeUpMode: Request a wake up

Prototype :
tMuxStatus LinSetWakeUpMode(unsigned int wCard, unsigned int wBus) eVanWakeUpMode);

Description :
This function allows a slave to request a wake-up. This function sends a byte 0x80 without any check. If the master does not issue frame headers within 64*TBit from the wake up request, this function will try issuing a new wake up request. After three (failing) requests this function stops to send wake up request.

Inputs parameters :
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]

Outputs parameters :

None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

8.21. LinClearBufferIFR : Purging the IFR transmission buffer

Prototype:
tMuxStatus LinClearBufferIFR(unsigned short wCard, unsigned short wBus);

Description:
This function allows the user to empty the buffer of IFR messages programmed in the card.

Input parameters:
wCard : Index of card number to be accessed
wBus : Index of bus number [0-x]

Output parameters:
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

8.22. LinGetFifoRxLevel : Level of filling of events waiting list

Prototype :
tMuxStatus LinGetFifoRxLevel (unsigned short wCard, unsigned short wBus, unsigned short wParam, unsigned short *wCount, unsigned short *wMaxCount);

Description :
This function allows the user to see how full the waiting list of events transferred to the application is.

Inputs parameters :
wCard : Index of card number to be accessed

wBus : Index of bus number [0-x]
wParam : Function's parameters
Bit 0 : Type of waiting list (only for USB cases)

- 0 : The number of transferred event is that situated on the PC's waiting list.
- 1 : The number of transferred event is that situated on the USB case's waiting list.

Outputs parameters :
wCount : Number of event currently on waiting list
wMaxCount : Maximum number of events accepted on the waiting list

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

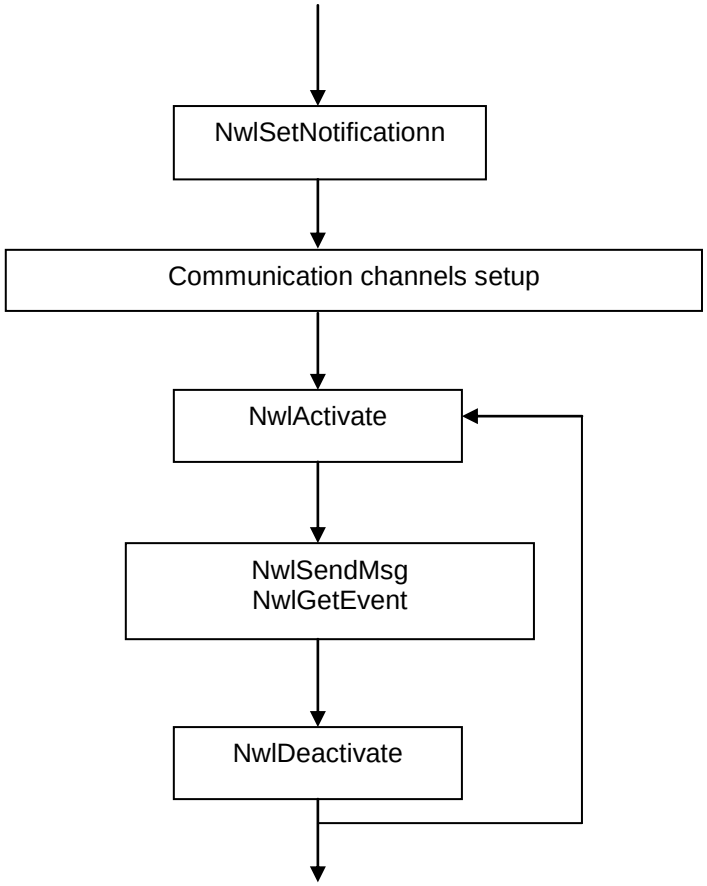
9. NWL Library

9.1. Order of call

9.1.1 Exchanges sequence

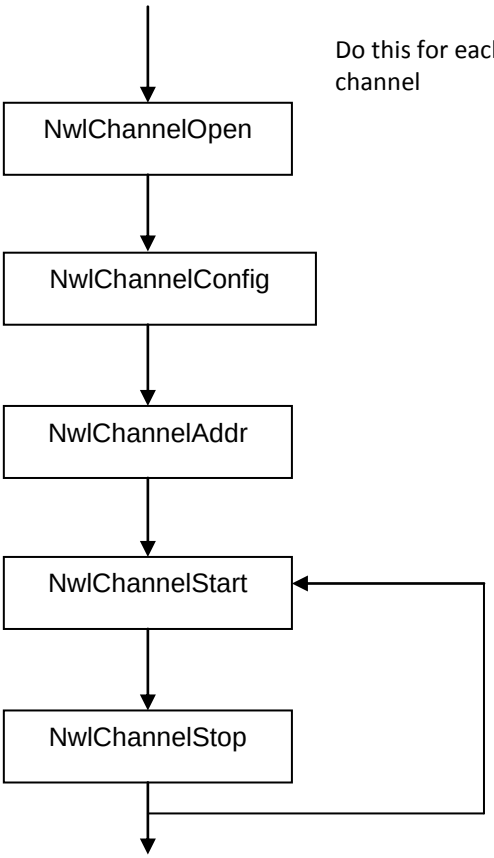
First Step: Set up all communication channels: the goal is to fill the different messages to be used by the application and also the communication parameters (LIN identifiers, delay and time-out).
Second Step: Start communication using « NwlActivate » function.
Third Step: Send a message.

- NwlSendMsg: Send a frame with a length between 0 and 4095 bytes.
- NwlGetEvent: Retrieve transmission events like ends of transmission, transmission errors, reception and reception errors.



9.1.2 Communication channels setup

Application must respect this sequence for each channel. Note that the functions allowing or not the communication on a channel (NwlChannelStart and NwlChannelStop) can be called during a communication.



9.2. NwlGetChannelCount: Count available communication channels

Prototype:
tMuxStatus NwlGetChannelCount(unsigned short wCard, unsigned short wBus, unsigned short * wChannelCount);

Description:
One communication channel is defined as a communication between a sender « a » and a receiver « b ». An application can open several channels at the same time. This function returns the maximum of channels that the application can open at the same time.

Inputs parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]

Outputs parameters:
wChannelCount: Number of available channels.

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameter error
STATUS_ERR_SEQUENCE	calling sequence error

9.3. NwlChannelOpen: Open a communication channel

Prototype :
tMuxStatus NwlChannelOpen(unsigned short wCard, unsigned short wBus, unsigned short *wChannel);

Description:
This function open a logical communication channel for communicate between a sender « a » and a receiver « b ».

Inputs parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]

Outputs parameters:
wChannel: Index of the open channel.

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameter error
STATUS_ERR_SEQUENCE	calling sequence error
STATUS_ERR_MEMORY	No more available channel

9.4. NwlChannelConfig: Configure a communication channel

Prototype:
tMuxStatus _MUXAPI NwlChannelConfig(unsigned short wCard, unsigned short wBus, unsigned short wChannel, tNwlConfig *hNwlChannelConfig);

Description:
This function can configure parameters of the communication channel.

Inputs parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
wChannel: Index of the open channel (get with the function: NwlChannelOpen)
hNwlChannelConfig: Channel parameters

eNwlService	Request Service: NWL_SVC_TRANSMIT_DATA : Data transmission NWL_SVC_RECEIVE_DATA : Data receipt
dwReserved1	Reserved for future use
dwReserved2	Reserved for future use
wParam	Field of bits defining working parameters Protocol Events: NWL_LIN_NO_EVENT: Application will not receive events corresponding to the low level communication. NWL_LIN_EVENT: Application will receive all Events. Filtering FF (first frame): NWL_FF_NO_EVENT: Application will not receive Events corresponding to a First Frame receipt or a First Frame end of transmission. NWL_FF_EVENT: Application will receive Events corresponding to a First Frame receipt or a First Frame end of transmission.

Outputs parameters:
None

Return code:

Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

9.5. NwlChannelAddr: Communication identifiers definition

Prototype:
tMuxStatus NwlChannelAddr(unsigned short wCard, unsigned short wBus, unsigned short wChannel, tNwlAddr *hNwlChannelAddr);

Description:
This function allows configuration of the identifiers for a communication channel.

Inputs parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
wChannel: Index of the open channel (get with the function: NwlChannelOpen)
hNwlChannelAddr: Communication channel address

wNad	Address used to identify a logical node (slave)
dwReserved1	Reserved for future use
dwReserved2	Reserved for future use

Outputs parameters:
None

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

9.6. NwlChannelParam : Channel communication parameters

Prototype:
tMuxStatus NwlChannelParam(unsigned short wCard, unsigned short wBus, unsigned short wChannel, tNwlChannelParam *hNwlChannelParam);

Description:

This function allows the application to configure channel communication parameters. Theses parameters define time limit for transmission and reception of segmented messages. Application is allowed to change theses parameters dynamically, if there is no communication running.

Inputs parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
wChannel: Index of the open channel (get with the function: NwlChannelOpen)
hNwlChannelParam: Communication parameters

wN_Cs	Minimum time gap allowed between the transmissions of consecutive frame network protocol data units [0-255] ms, correspond also to the parameter STmin. The measurement of the Separation Time (STmin) starts after completion of transmission of a Consecutive Frame (CF) and ends at the request for the transmission of the next Consecutive Frame (CF).	10
wN_As	Maximum transmission time of the transmitter [0-65535] ms	10
wN_Cr	Time until reception of Consecutive Frame [0-65535] ms	20
wP2min	Time to wait, for the master, before sending response request [0-500] ms	50
wP2max	Time until reception of the response for the master [0-500] ms	500

Outputs parameters:
None

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_BUSY	Communication running

9.7. NwlChannelStart : Start communication for a channel

Prototype:
tMuxStatus NwlChannelStart(unsigned short wCard, unsigned short wBus, unsigned short wChannel);

Description:
This function starts communication on the specified channel.

Inputs parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
wChannel: Index of the open channel (get with the function: NwlChannelOpen)

Outputs parameters:
None

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

9.8. NwlChannelStop : Stop communication for a channel

Prototype:
tMuxStatus NwlChannelStop(unsigned short wCard, unsigned short wBus, unsigned short wChannel);

Description:
This function stops communication on the specified channel.

Inputs parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
wChannel: Index of the open channel (get with the function: NwlChannelOpen)

Outputs parameters:
None

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

9.9. NwlActivate : Start communication

Prototype:

tMuxStatus NwlActivate(unsigned int wCard, unsigned int wBus);

Description :
This function starts communication on the network. After execution of this function, application can send or receive messages.

Inputs parameters :
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]

Outputs parameters:
None

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

9.10. NwlDeactivate : Stop communication

Prototype:
tMuxStatus NwlDeactivate(unsigned int wCard, unsigned int wBus);

Description :
This function stops communication on the network. After execution of this function, application can't send or receive messages.

Inputs parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]

Outputs parameters:
None

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

9.11. NwlChannelSendMsg: Send a message

Prototype:
tMuxStatus NwlSendMsg(unsigned short wCard, unsigned short wBus, unsigned short wChannel, tNwlMsg *hNwlMsg);

Description:
This function sends a message.

Inputs parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
wChannel: Index of the open channel (get with the function: NwlChannelOpen)
hNwlMsg: Message parameters:

wDataLen	Max data length [0-4095]
bData	Data to send

Outputs parameters:
None

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_MSGSVC	Channel is not a valid transmission channel
STATUS_ERR_MSGBUSY	New message could not be sent because a message is currently transmitted.

9.12. NwlChannelReceiveMsg: Receive a message

Prototype:
tMuxStatus NwlChannelReceiveMsg(unsigned short wCard, unsigned short wBus, unsigned short wChannel);

Description :
This function allows, for the master, to request (or request again) the slave node to send a message of data.

Inputs parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]

wChannel: Index of the open channel (get with the function: NwlChannelOpen)

Outputs parameters:
None

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error
STATUS_ERR_MSGSVC	Channel is not a valid reception channel
STATUS_ERR_NWLBUSY	The request cannot be taken into account. A communication is running for that channel.

9.13. NwlGetEvent : Reading of an event

Prototype:
tMuxStatus NwlGetEvent(unsigned short wCard, unsigned short wBus, tNwlEvent *hNwlEvent);

Description:
This function allows the user to retrieve an event from the network.

Inputs parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
hNwlEvent: Pointer on the event to complete

Outputs parameters:
hNwlEvent: Completed event

wBus	Index of bus managing the event
wChannel	Index of channel managing the event [0-n].
eTypeEvent	Type of event EVENT_EMPTY: No event. EVENT_NWL_MSGTX: - End of correct transmission of a message EVENT_ NWL_MSGRX : - Correct reception of a message EVENT_NWL_MSGTXERR: - End of transmission error EVENT_NWL_MSGRXERR : - Receipt error EVENT_NWL_MSGRXFF:

	- Correct reception of a First Frame EVENT_NWL_MSGTXFF: - Correct end of transmission of a First Frame EVENT_FIFO_OVF - Bit indicating that one or several events after this are lost because the FIFO reception was full.
wNad	LIN message identifier
dwTimeStamp	Arrival time of the event in a multiple of 100 µSec. This parameter is correct if wTimePrecision is zero.
wTimePrecision	Precision of dwTimeStamp in multiple of 100 µSec.
eService	Request service : NWL_SVC_TRANSMIT_DATA : Data Transmission NWL_SVC_RECEIVE_DATA: Data receipt.
eError	For events of type EVENT_NWL_MSGTXERR or EVENT_NWL_MSGRXERR, indication of type of error : - NWL_NO_ERR : No error - NWL_ERR_TOUT_AS : Maximum transmission time of the transmitter elapsed (wN_As) - NWL_ERR_TOUT_CR : Time until reception of Consecutive Frame elapsed (wN_Cr) - NWL_ERR_TOUT_P2 : Time for the request or the waiting of a response elapsed (parameters wP2min and wP2max) - NWL_ERR_SN: Sequence Number error. Receipt is stopped - NWL_ERR_FIFO_OVF: FIFO for transmit LIN messages is full. Receipt or transmission is stopped - NWL_ERR_MEMORY : No enough memory - NWL_ERR_INV_PDU : PDU Format receipt error
dwReserved1	Reserved for future use
dwReserved2	Reserved for future use
wDataLen	Length of data [0-4095]
bData	Data buffer

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

Example:

```
/* Indication of event */
tNwlEvent hNwlEvent ;
tMuxStatus Status ;
unsigned short wCard=0,wBus=0 ;
```

```
Status=NwlGetEvent(wCard,wBus,&hNwlEvent);
if (Status != STATUS_OK) return ;
switch(hNwlEvent.eTypeEvent &~ EVENT_FIFO_OVF)
{
        case EVENT_EMPTY :              /* No more events to be dealt with */
                break ;
        case EVENT_NWL_MSGTX:           /* Correct end of transmission */
                break ;
        case EVENT_NWL_MSGRX:           /* Correct reception */
                break ;
        case EVENT_NWL_MSGTXERR:        /* End of transmission error*/
                break ;
        case EVENT_NWL_MSGRXERR:        /* Receipt error*/
                break ;
        case EVENT_NWL_MSGRXFF:         /* A First Frame is receipted*/
                break ;
        case EVENT_NWL_MSGTXFF:         /* End of transmission of a First Frame*/
                break ;
        default  :                      /* Reserved for future use */
                break ;
}
```

9.14. NwlGetFifoRxLevel : Level of filling of events waiting list

Prototype:
tMuxStatus NwlGetFifoRxLevel (unsigned short wCard, unsigned short wBus, unsigned short wParam, unsigned short *wCount, unsigned short *wMaxCount);

Description:
This function allows the user to see how full the waiting list of events transferred to the application is.

Inputs parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
wParam: Function's parameters
Bit 0 : Type of waiting list (only for USB cases)

- 0 : Event queue is situated on the PC.
- 1 : Event queue is on board

Outputs parameters:
wCount: Number of event currently on waiting list
wMaxCount: Maximum number of events accepted on the waiting list

Return code:

Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

9.15. NwlGetChannelState : Current state of a channel

Prototype:
tMuxStatus NwlGetChannelState(unsigned short wCard, unsigned short wBus, unsigned short wChannel, unsigned short *wChannelState, tNwlError *eLastTxStatus);

Description:
This function allows the user to directly read the status of communication channel.

Inputs parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
wChannel: Index of the open channel (get with the function: NwlChannelOpen)

Outputs parameters:
wChannelState: Status of communication channel
- Bit NWL_CHANNEL_STATE_STARTED:
  - 0 : Channel is not activated
  - 1 : Channel is activated
- Bit NWL_CHANNEL_STATE_BUSY
  - 0 : Channel is activated and idle
  - 1 : Channel is currently transmitting or receiving a message

eLastTxStatus : Status of the last end of transmission

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

9.16. NwlIsBusActive : Bus Status

Prototype:

tMuxStatus NwlIsBusActive(unsigned short wCard, unsigned short wBus, unsigned short *wState);

Description :
This function allows to indicate if the bus has been activated using the NwlActivate function.

Input parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]

Ouptut parameters:
wState: Indicates if the bus is activated or not (value different than 0)

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

9.17. NwlChannelClose : Closing a communication channel

Prototype:
tMuxStatus NwlChannelClose(unsigned short wCard, unsigned short wBus, unsigned short wChannel);

Description :
This function allows closing a communication channel.

Input parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
wChannel : Number of the channel to be closed

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

9.18. NwlSetNotification: Declaration of the application event

Prototype:
tMuxStatus NwlSetNotification (unsigned short wCard, unsigned short wBus, HANDLE hWinEvent);

Description :
This function allows the application to use the methods of tasks synchronization provided by the Windows operating system (API WIN32). This function allows to pass to the DLL an event handle created by the Windows function « CreateEvent ». This event is used during the communication when an event is sent back by the DLL to the application (e.g. end of transmission, reception, error, …).
Then the application can wait for events using waiting functions as WaitForSingleObject or WaitForMultipleObject.

Example:

```
/* Request of event indications */
HANDLE hWinEvent ;
tNwlEvent hNwlEvent ;
tMuxStatus Status ;
unsigned short wCard=0,wBus=0 ;

hWinEvent=CreateEvent(NULL,FALSE,FALSE,NULL) ;
if (hWinEvent == NULL) return ;
Status=NwlSetNotification (wCard,wBus,hWinEvent) ;
if (Status != STATUS_OK) return ;
Status=NwlActivate(wCard,wBus) ;
if (Status != STATUS_OK) return ;
if (WaitForSingleObject(hWinEvent,INFINITE) == WAIT_OBJECT_0)
{
        NwlGetEvent(wCard,wBus, ,&hNwlEvent);
}
```

Input parameters:
wCard: Index of card number to be accessed
wBus: Index of bus number [0-x]
hWinEvent : Handle returned by the function CreateEvent.

Output parameters:
None

Return code:
Status: Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

10. Inputs / Outputs Library

10.1. IOSetOutput: Activation of logic outputs

Prototype :
tMuxStatus IOSetOutput(unsigned short wCard, unsigned short wOutputValue, unsigned short wOutputMask);

Description :
This function allows the user to activate the logic outputs on the card.

Inputs parameters :
wCard : Index of card number to be accessed
wOutputValue : Outputs values
wOutputMask : Mask indicating the outputs to be positioned. A bit at 1 indicates that the output value can be updated; a bit at 0 indicates that the output value cannot be modified.

Example :
Value 1 in output 0 : IsoSetOutput(wCard, 0x01, 0x01);
Value 0 in output 0 : IsoSetOutput(wCard, 0x00, 0x01);
Value 1 in outputs 2 and 3 : IsoSetOutput(wCard, 0x0C, 0x0C);
Value 0 in output 3: IsoSetOutput(wCard, 0x00, 0x08);

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

Outputs table

Bit	Meaning
0	Output n. 0 or RTS1 line
1	Output n. 1 or RTS2 line
2	Output n. 2
3	Output n. 3
4	Output n. 4 or direct control from line K of network n. 1
5	Output n. 5 or direct control from line L of network n. 1

6	Output n. 6 or direct control from line K of network n. 2
7	Output n. 7 or direct control from line L of network n. 2
8	Output n. 8
9	Output n. 9
10	Output n. 10
11	Output n. 11
12	Output n. 12
13	Output n. 13
14	Output n. 14
15	Output n. 15

10.2. IOGetInput: Reading of logic inputs

Prototype :
IOGetInput(unsigned short wCard, unsigned short *wInputValue);

Description :
This function allows the user to read the status of those logic inputs present on the card.

Inputs parameters :
wCard : Index of card number to be accessed
wInputValue : Pointer on the value to be completed

Outputs parameters :
wInputValeur : Status of inputs

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

Inputs table

Bit	Meaning
0	Input n.0 or CTS1 line or +VBAT
1	Input n.1 or CTS2 line or +APC
2	Input n.2
3	Input n.3 or presence status +APC for PCI-XXXX
4	Input n.4 or presence status +VBAT for PCI-XXXX
5	Input n.5 or status of line K of network n. 1
6	Input n.6 or status of line K of network n. 2
7	Input n.7

8	Input n.8
9	Input n.9
10	Input n.10
11	Input n.11
12	Input n.12
13	Input n.13
14	Input n.14
15	Input n.15

11. Timer Library

In order to facilitate the creation of users' programmes, the libraries can be used to provide the application with a time base, based on a 1 millisecond timer.

11.1. TimerSet : Activation / de-activation of a time base in milliseconds

Prototype :
tMuxStatus _MUXAPI TimerSet(unsigned short wCard, unsigned short wTimerValue);

Description :
This function allows the user to start or stop a time base expressed in milliseconds.

Inputs parameters :
wCard : Index of card number to be accessed
wTimerValue : Value of the time base expressed in milliseconds. A 0 value stops the time base.

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

11.2. TimerSetNotification : Declaration of the application event

Prototype :
tMuxStatus TimerSetNotification (unsigned short wCard, unsigned short wBus, HANDLE hWinEvent);

Description :
This function allows the application to use the task synchronisation methods provided for Windows operating systems (API WIN32). This function allows the user to transfer a handle of event created by the Windows « CreateEvent » function to the DLL. This event is used to transfer the timer event to the application.

From then on, the application can wait for events by means of waiting functions such as WaitForSingleObject or WaitForMultipleObject.

Example :

```
/* Request for a 1 millisecond timer*/
HANDLE hWinEvent ;
tMuxStatus Status ;
unsigned short wCard=0;

hWinEvent=CreateEvent(NULL,FALSE,FALSE,NULL) ;
if (hWinEvent == NULL) return ;
Status=TimerSetNotification (wCard,0) ;
if (Status != STATUS_OK) return ;
if (WaitForSingleObject(hWinEvent,INFINITE) == WAIT_OBJECT_0)
{
        /*
        printf(« « ) ;
        */
}
```

Inputs parameters :
wCard : Index of card number to be accessed
hWinEvent : « handle » returned by the CreateEvent function.

Outputs parameters :
None

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

11.3. TimerRead: Reading of current clock

Prototype :

```
tMuxStatus _MUXAPI TimerRead(unsigned short wCard, unsigned long *dwTimerValue, unsigned short *wTimeError, , unsigned short *wMaxTimePrecision);
```

Description :
This function allows reading the present value of the internal clock managed by the DLLs. This clock, expressed in multiple of 1 ms, is activated when the driver for the MuxOpen function is open.

This function provides the synchronised application a time base regarding the network events, but it can be also used to recover the timer events that might not have been dealt with by the application. In the case of a multi-task operation system, a Windows (or « thread ») application can be suspended for an indefinite period of time. So as to compensate this signal variation, when the task is retaken it is necessary to do a loop in order to recover the time lost during suspension of the task.

Example :

```
/* Request for a 1 millisecond timer */
HANDLE hWinEvent ;
tMuxStatus Status ;
unsigned short wCard=0;
unsigned long dwTimer1, dwTimer2 ;
unsigned short wTimeError ;

hWinEvent=CreateEvent(NULL,FALSE,FALSE,NULL) ;
if (hWinEvent == NULL) return ;
Status=TimerSetNotification (wCard,0) ;
if (Status != STATUS_OK) return ;
Status=TimerRead(&dwTimer1,&wTimeError);
if (Status != STATUS_OK) return ;
if (WaitForSingleObject(hWinEvent,INFINITE) == WAIT_OBJECT_0)
{
        Status=TimerRead(&dwTimer1,&wTimeError);
        /*
        printf(« « ) ;
        */
}
```

Inputs parameters :
wCard : Index of card number to be accessed

Outputs parameters :
dwTimerValue : 1 ms increment counter from the MuxOpen function
wTimeError : Clock interruption loss counter (C.F. ANNEXE : time stamp indication in PCI-MUX cards)
wMaxTimePrecision : Maximum clock precision error

Return code:
Status : Summary of the function execution

STATUS_OK	
STATUS_ERR_PARAM	Parameters error
STATUS_ERR_SEQUENCE	Sequence error

12. Inserting libraries into the project

There are two methods to include the DLLs into the project:

- The static load method
- The dynamic load method

12.1. The static load method

This method consists of calling directly those library functions of external reference, as if they were standard functions. Then, while the link is being edited, insert the *.LIB file.

Procedure

1 – Insert the REFMUX.H prototypes' definition file into each reference file related to at least one of the library functions.

```
#include "refmux.h"
```

2 – Add the MUXDLL.LIB file to the project for a direct access to the cards or AEBRIDGE.DLL for a shared access to the cards.

Note: In VISUAL C++ and BORLAND C++, this operation is carried out by the menu « Project » then « add to project »

Advantages and disadvantages

Advantages:

1. Easy to apply

Disadvantages:

1. If certain compilers are used, the file MUXDLL.LIB might not be compatible. In this case, a format error takes place, and only the dynamic load method can be used.
2. Once the application has been generated, it is dependant of the MUXDLL.DLL file version. The application's executable file and the *.DLL file must always be associated.

12.2. The dynamic load method

This method consists of calling the library functions indirectly. This is done by first calling the **MuxLoadDLL** function, which initialises the function pointers. Then, the functions are called as if they were directly linked to the project.

Procedure

1 - Insert the REFMUX.H prototypes' definition file into each reference file, writing first the key word MUX_DYNAMIC_DLL, which indicates the dynamic load of the functions.

```
#define MUX_DYNAMIC_DLL
#include "refmux.h"
```

2 – Call the MuxLoadDLL function before calling any other functions.

3 – Insert into other files that might eventually be related with the libraries the prototypes definition file REFMUX.H by first writing the key word MUX_DYNAMIC_DLL (dynamic load) and MUX_DYNAMIC_DLL_EXTERN, so as not to declare again the function pointers.

```
#define MUX_DYNAMIC_DLL
#define MUX_DYNAMIC_DLL_EXTERN
#include "refmux.h"
```

4 – Call the MuxFreeDLL function at the end of the programme in order to free the function pointers

Advantages and disadvantages

Advantages:

1. The application has access to the functions through Windows events (API WIN32). The application can be compiled without any file format problems with the library.
2. The application is no longer dependant of the MUXDLL.DLL file. Updating the DLLs alone could be done without regenerating the application.

Disadvantages:

1. Slightly more complicated to apply than the static load method.

Annex 1: Common prototypes

tMuxStatus _MUXAPI **MuxCountCards**(unsigned long *dwCardsCount);
tMuxStatus _MUXAPI **MuxPciCountCards**(unsigned long *dwCardsCount);
tMuxStatus _MUXAPI **MuxUsbCountCards**(unsigned long *dwCardsCount);
tMuxStatus _MUXAPI **MuxPciGetCardInfo**(unsigned short wCard, unsigned long *dwCardBus, unsigned long *dwCardSlot, unsigned long *dwCardInfo);
tMuxStatus _MUXAPI **MuxUsbGetCardInfo**(unsigned short wCard, unsigned long *dwHubNum, unsigned long *dwPortNum, unsigned long *dwFullSpeed, unsigned long *dwDeviceAddress, unsigned long *dwCardInfo, unsigned long *dwUniqueId);
tMuxStatus _MUXAPI **MuxInit**(unsigned int wCard);
tMuxStatus _MUXAPI **MuxOpen**(unsigned short wCard, tMuxMode eMode);
tMuxStatus _MUXAPI **MuxGetVersion**(unsigned short wCard,unsigned long *dwVersionDll,unsigned long *dwVersionDriver,unsigned long *dwVersionKernel, tMuxCard *eNumCard);
tMuxStatus _MUXAPI **MuxClose**(unsigned short wCard);
tMuxStatus _MUXAPI **MuxGetLastErrorString**(char **pString);
tMuxStatus _MUXAPI **MuxLoadDLL**(void);
tMuxStatus _MUXAPI **MuxFreeDLL**(void);

Annex 2: CAN Prototypes

tMuxStatus _MUXAPI **CanConfigOper**(unsigned short wCard, unsigned short wBus, tCanOper *hCanOper);

tMuxStatus _MUXAPI **CanConfigBus**(unsigned short wCard, unsigned short wBus, tCanBus *hCanBus);

tMuxStatus _MUXAPI **CanConfigParam**(unsigned short wCard, unsigned short wBus, tCanParam *hCanParam);

tMuxStatus _MUXAPI **CanConfigStat**(unsigned short wCard, unsigned short wBus, unsigned short wBusLoadTime);

tMuxStatus _MUXAPI **CanConfigTransceiverHS**(unsigned short wCard, unsigned short wBus, tCanSlope eCanSlope);

tMuxStatus _MUXAPI **CanConfigTransceiverHS**(unsigned short wCard, unsigned short wBus, tCanSlope eCanSlope);

tMuxStatus _MUXAPI **CanSelectTransceiverHS**(unsigned short wCard, unsigned short wBus, tCanBoolean eCanTxHS);

tMuxStatus _MUXAPI **CanConfigTransceiverLS**(unsigned short wCard, unsigned short wBus, unsigned short wStandBy, unsigned short wEnable, unsigned short wWakeUp);

tMuxStatus _MUXAPI **CanConfigTerminationLS**(unsigned short wCard, unsigned short wBus, unsigned short wValue);

tMuxStatus _MUXAPI **CanReadTransceiverLS**(unsigned short wCard, unsigned short wBus, unsigned short *wLineState);

tMuxStatus _MUXAPI **CanCreateMsg**(unsigned short wCard, unsigned short wBus, tCanMsg *hCanMsg);

tMuxStatus _MUXAPI **CanConfigPeriodic**(unsigned short wCard, unsigned short wBus, unsigned short wOffset, unsigned short wParam, tCanMsg *hCanMsgNew);

tMuxStatus _MUXAPI **CanSetNotification** (unsigned short wCard, unsigned short wBus, HANDLE hWinEvent);

tMuxStatus _MUXAPI **CanActivate**(unsigned short wCard, unsigned short wBus);

tMuxStatus _MUXAPI **CanDeactivate**(unsigned short wCard, unsigned short wBus);

tMuxStatus _MUXAPI **CanSendMsg**(unsigned short wCard, unsigned short wBus, tCanMsg *hCanMsg);

tMuxStatus _MUXAPI **CanGetEvent**(unsigned short wCard, unsigned short wBus, tCanEvent *hCanEvent);

tMuxStatus _MUXAPI **CanGetFifoRxLevel** (unsigned short wCard, unsigned short wBus, unsigned short wParam, unsigned short *wCount, unsigned short *wMaxCount);

tMuxStatus _MUXAPI **CanGetStat**(unsigned short wCard, unsigned short wBus, tCanStat *hCanStat);

tMuxStatus _MUXAPI **CanGetBusState**(unsigned short wCard, unsigned short wBus, tCanChipState *eCanChipState, unsigned char *bTxErrCount, unsigned char *bRxErrCount);

tMuxStatus _MUXAPI **CanBusOn**(unsigned short wCard, unsigned short wBus);

tMuxStatus _MUXAPI **CanReadByte**(unsigned short wCard, unsigned short wBus, unsigned short wOffset,unsigned char *bData);
tMuxStatus _MUXAPI **CanWriteByte**(unsigned short wCard, unsigned short wBus, unsigned short wOffset,unsigned char bData);tMuxStatus _MUXAPI **CanConfigClock**(unsigned short wCard, unsigned short wBus, tCanClockFreq eCanClock);
tMuxStatus _MUXAPI **CanConfigRangeFilter**(unsigned short wCard, unsigned short wBus, tCanRangeFilter *hCanRangeFilter)
tMuxStatus _MUXAPI **CanReadTransceiverLS**(unsigned short wCard, unsigned short wBus, unsigned short *wLineState);
tMuxStatus _MUXAPI **CanConfigPeriodicList**(unsigned short wCard, unsigned short wBus, unsigned short wPeriodicCount, tCanPeriodicMsg *hPeriodicCanMsgList);
tMuxStatus _MUXAPI **CanSendMsgList**(unsigned short wCard, unsigned short wBus, unsigned short wMsgCount, tCanMsg *hCanMsgList);
tMuxStatus _MUXAPI **CanIsBusActive**(unsigned short wCard, unsigned short wBus, unsigned short *wState);
tMuxStatus _MUXAPI **CanClearFifoRx**(unsigned short wCard, unsigned short wBus);

Annex 3: NWC Prototypes

tMuxStatus _MUXAPI **NwcRegister**(char *szString, char *szKey);
tMuxStatus _MUXAPI **NwcGetChannelCount**(unsigned short wCard, unsigned short wBus, unsigned short *wChannelCount);
tMuxStatus _MUXAPI **NwcChannelOpen**(unsigned short wCard, unsigned short wBus, unsigned short *wChannel);
tMuxStatus _MUXAPI **NwcChannelConfig**(unsigned short wCard, unsigned short wBus, unsigned short wChannel, tNwcConfig *hNwcChannelConfig);
tMuxStatus _MUXAPI **NwcChannelAddr**(unsigned short wCard, unsigned short wBus, unsigned short wChannel, tNwcAddr *hNwcChannelAddr);
tMuxStatus _MUXAPI **NwcChannelParam**(unsigned short wCard, unsigned short wBus, unsigned short wChannel, tNwcParam *hNwcChannelParam);
tMuxStatus _MUXAPI **NwcChannelStart**(unsigned short wCard, unsigned short wBus, unsigned short wChannel);
tMuxStatus _MUXAPI **NwcChannelStop**(unsigned short wCard, unsigned short wBus, unsigned short wChannel);
tMuxStatus _MUXAPI **NwcChannelActivate**(unsigned short wCard, unsigned short wBus);
tMuxStatus _MUXAPI **NwcChannelDeactivate**(unsigned short wCard, unsigned short wBus);
tMuxStatus _MUXAPI **NwcChannelSendMsg**(unsigned short wCard, unsigned short wBus, unsigned short wChannel, tNwcMsg *hNwcChannelMsg);
tMuxStatus _MUXAPI **NwcGetEvent**(unsigned short wCard, unsigned short wBus, tNwcEvent *hNwcEvent);
tMuxStatus _MUXAPI **NwcGetIdent**(tNwcConfig *pstNwcConfig,tNwcAddr *pstNwcAddr);
tMuxStatus _MUXAPI **NwcGetFifoRxLevel**(unsigned short wCard, unsigned short wBus, unsigned short wParam, unsigned short *wCount, unsigned short *wMaxCount);
tMuxStatus _MUXAPI **NwcConfigSpyMode**(unsigned short wCard, unsigned short wBus, tNwcSpyAddr *hNwcSpyAddr);
tMuxStatus _MUXAPI **NwcGetChannelState**(unsigned short wCard, unsigned short wBus,unsigned short wChannel, unsigned short *wChannelState, tNwcError *eLastTxStatus);
tMuxStatus _MUXAPI **NwcIsBusActive**(unsigned short wCard, unsigned short wBus, unsigned short *wState);
tMuxStatus _MUXAPI **NwcChannelClose**(unsigned short wCard, unsigned short wBus, unsigned short wChannel);
tMuxStatus _MUXAPI **NwcSetNotification**(unsigned short wCard, unsigned short wBus, HANDLE hWinEvent);

Annex 4: J1939 Prototypes

tMuxStatus _MUXAPI **J19ChannelOpen**(unsigned short wCard, unsigned short wBus, unsigned short *wChannel);
tMuxStatus _MUXAPI **J19ChannelAddr**(unsigned short wCard, unsigned short wBus, unsigned short wChannel, tJ19Addr * hJ19Addr);
tMuxStatus _MUXAPI **J19ChannelParam**(unsigned short wCard, unsigned short wBus, unsigned short wChannel, tJ19Param *hJ19ChannelParam);
tMuxStatus _MUXAPI **J19ChannelConfig**(unsigned short wCard, unsigned short wBus, unsigned short wChannel, tJ19Config *hJ19ChannelConfig);
tMuxStatus _MUXAPI **J19ChannelStart**(unsigned short wCard, unsigned short wBus, unsigned short wChannel);
tMuxStatus _MUXAPI **J19ChannelStop**(unsigned short wCard, unsigned short wBus, unsigned short wChannel);
tMuxStatus _MUXAPI **J19ChannelClose**(unsigned short wCard, unsigned short wBus, unsigned short wChannel);
tMuxStatus _MUXAPI **J19Activate**(unsigned short wCard, unsigned short wBus);
tMuxStatus _MUXAPI **J19ConfigBus**(unsigned short wCard, unsigned short wBus, unsigned short wParam);
tMuxStatus _MUXAPI **J19Deactivate**(unsigned short wCard, unsigned short wBus);
tMuxStatus _MUXAPI **J19ChannelSendMsg**(unsigned short wCard, unsigned short wBus,unsigned short wChannel, tJ19Msg *hJ19Msg);
tMuxStatus _MUXAPI **J19ChannelSendNMEAFastPacket**(unsigned short wCard, unsigned short wBus,unsigned short wChannel, tJ19Msg *hJ19Msg);
tMuxStatus _MUXAPI **J19SetNotification**(unsigned short wCard, unsigned short wBus, HANDLE hWinEvent);
tMuxStatus _MUXAPI **J19GetEvent**(unsigned short wCard, unsigned short wBus, tJ19Event *hJ19Event);
tMuxStatus _MUXAPI **J19GetFifoRxLevel**(unsigned short wCard, unsigned short wBus, unsigned short wParam, unsigned short *wCount, unsigned short *wMaxCount);
tMuxStatus _MUXAPI J**19StoreNMEAFastPacket**(unsigned short wCard, unsigned short wBus, unsigned long dwIdent, unsigned short wSize);

Annex 5: ISO Prototypes

tMuxStatus _MUXAPI **IsoConfigOper** (unsigned short wCard, unsigned short wBus, tIsoOper *hIsoOper);
tMuxStatus _MUXAPI **IsoConfigBus** (unsigned short wCard, unsigned short wBus, tIsoBus *hIsoBus);
tMuxStatus _MUXAPI **IsoConfigParam** (unsigned short wCard, unsigned short wBus, tIsoParam *hIsoParam);
tMuxStatus _MUXAPI **IsoConfigStat** (unsigned short wCard, unsigned short wBus, unsigned short wBusLoadTime);
tMuxStatus _MUXAPI **IsoSetNotification** (unsigned short wCard, unsigned short wBus, HANDLE hWinEvent);
tMuxStatus _MUXAPI **IsoActivate**(unsigned short wCard, unsigned short wBus);
tMuxStatus _MUXAPI **IsoDeactivate**(unsigned short wCard, unsigned short wBus);
tMuxStatus _MUXAPI **IsoConfigPeriodic** (unsigned short wCard, unsigned short wBus, unsigned short wOffset, unsigned short wParam, tIsoMsg *hIsoMsg);
tMuxStatus _MUXAPI **IsoSendMsg** (unsigned short wCard, unsigned short wBus, tIsoMsg *hIsoMsg);
tMuxStatus _MUXAPI **Iso14230SendMsg** (unsigned short wCard, unsigned short wBus, tIso14230Msg *hIso14230Msg);
tMuxStatus _MUXAPI **IsoGetEvent** (unsigned short wCard, unsigned short wBus, tIsoEvent *hIsoEvent);
tMuxStatus _MUXAPI **IsoWaitResponse** (unsigned short wCard, unsigned short wBus, unsigned short wTimeOut);
tMuxStatus _MUXAPI **IsoChangeBaudRate** (unsigned short wCard, unsigned short wBus, tIsoBaudRate eBaudRateTx, tIsoBaudRate eBaudRateRx);
tMuxStatus _MUXAPI **IsoGetStat** (unsigned short wCard, unsigned short wBus, tCanStat *hCanStat);
tMuxStatus _MUXAPI **IsoGetFifoRxLevel** (unsigned short wCard, unsigned short wBus, unsigned short wParam, unsigned short *wCount, unsigned short *wMaxCount);
tMuxStatus _MUXAPI **IsoIsBusActive**(unsigned short wCard, unsigned short wBus, unsigned short *wState);

Annex 6: LIN Prototypes

tMuxStatus _MUXAPI **LinConfigOper** (unsigned short wCard, unsigned short wBus, tLinOper *hLinOper);
tMuxStatus _MUXAPI **LinConfigBus** (unsigned short wCard, unsigned short wBus, tLinBus *hLinBus);
tMuxStatus _MUXAPI **LinConfigUart**(unsigned short wCard, unsigned short wBus, tUartConfig *hUartConfig);
tMuxStatus _MUXAPI **LinConfigParam** (unsigned short wCard, unsigned short wBus, tLinParam *hLinParam);
tMuxStatus _MUXAPI **LinConfigStat** (unsigned short wCard, unsigned short wBus, unsigned short wBusLoadTime);
tMuxStatus _MUXAPI **LinSetVersion**(unsigned short wCard, unsigned short wBus, unsigned long dwVersion);
tMuxStatus _MUXAPI **LinConfigTransceiver**(unsigned short wCard, unsigned short wBus, unsigned short wEnable, unsigned short wRMaster);
tMuxStatus _MUXAPI **LinConfigPeriodic**(unsigned short wCard, unsigned short wBus, unsigned short wOffset, unsigned short wParam, tLinMsg *hLinMsgNew);
tMuxStatus _MUXAPI **LinConfigPeriodicList**(unsigned short wCard, unsigned short wBus, unsigned short wPeriodicCount, tLinPeriodicMsg *hLinPeriodicMsgList);
tMuxStatus _MUXAPI **LinSetNotification** (unsigned short wCard, unsigned short wBus, HANDLE hWinEvent);
tMuxStatus _MUXAPI **LinActivate**(unsigned short wCard, unsigned short wBus);
tMuxStatus _MUXAPI **LinDeactivate**(unsigned short wCard, unsigned short wBus);
tMuxStatus _MUXAPI **LinSendMsg** (unsigned short wCard, unsigned short wBus, tLinMsg *hLinMsg);
tMuxStatus _MUXAPI **LinSendMsgList**(unsigned short wCard, unsigned short wBus, unsigned short wMsgCount, tLinMsg *hLinMsgList);
tMuxStatus _MUXAPI **LinGetEvent** (unsigned short wCard, unsigned short wBus, tLinEvent *hLinEvent);
tMuxStatus _MUXAPI **LinSetSleepMode**(unsigned short wCard, unsigned short wBus);
tMuxStatus _MUXAPI **LinSetWakeUpMode**(unsigned short wCard, unsigned short wBus);
tMuxStatus _MUXAPI **LinGetStat** (unsigned short wCard, unsigned short wBus, tLinStat *hLinStat);
tMuxStatus _MUXAPI **LinGetBusState** (unsigned short wCard, unsigned short wBus, tLinChipState *eLinChipState, unsigned char *bTxErrCount, unsigned char *bRxErrCount);
tMuxStatus _MUXAPI **LinClearBufferIFR**(unsigned short wCard, unsigned short wBus);
tMuxStatus _MUXAPI **LinGetFifoRxLevel** (unsigned short wCard, unsigned short wBus, unsigned short wParam, unsigned short *wCount, unsigned short *wMaxCount);

Annex 7: NWL Prototypes

tMuxStatus _MUXAPI **NwlGetChannelCount**(unsigned short wCard, unsigned short wBus, unsigned short *wChannelCount);
tMuxStatus _MUXAPI **NwlChannelOpen**(unsigned short wCard, unsigned short wBus, unsigned short *wChannel);
tMuxStatus _MUXAPI **NwlChannelConfig**(unsigned short wCard, unsigned short wBus, unsigned short wChannel, tNwlConfig *hNwlChannelConfig);
tMuxStatus _MUXAPI **NwlChannelAddr**(unsigned short wCard, unsigned short wBus, unsigned short wChannel, tNwlAddr *hNwlChannelAddr);
tMuxStatus _MUXAPI **NwlChannelParam**(unsigned short wCard, unsigned short wBus, unsigned short wChannel, tNwlParam *hNwlChannelParam);
tMuxStatus _MUXAPI **NwlChannelStart**(unsigned short wCard, unsigned short wBus, unsigned short wChannel);
tMuxStatus _MUXAPI **NwlChannelStop**(unsigned short wCard, unsigned short wBus, unsigned short wChannel);
tMuxStatus _MUXAPI **NwlChannelActivate**(unsigned short wCard, unsigned short wBus);
tMuxStatus _MUXAPI **NwlChannelDeactivate**(unsigned short wCard, unsigned short wBus);
tMuxStatus _MUXAPI **NwlChannelSendMsg**(unsigned short wCard, unsigned short wBus, unsigned short wChannel, tNwlMsg *hNwlChannelMsg);
tMuxStatus _MUXAPI **NwlChannelReceiveMsg** (unsigned short wCard, unsigned short wBus, unsigned short wChannel);
tMuxStatus _MUXAPI **NwlGetEvent**(unsigned short wCard, unsigned short wBus, tNwlEvent *hNwlEvent);
tMuxStatus _MUXAPI **NwlGetFifoRxLevel**(unsigned short wCard, unsigned short wBus, unsigned short wParam, unsigned short *wCount, unsigned short *wMaxCount);
tMuxStatus _MUXAPI **NwlGetChannelState**(unsigned short wCard, unsigned short wBus,unsigned short wChannel, unsigned short *wChannelState, tNwlError *eLastTxStatus);
tMuxStatus _MUXAPI **NwlIsBusActive**(unsigned short wCard, unsigned short wBus, unsigned short *wState);
tMuxStatus _MUXAPI **NwlChannelClose**(unsigned short wCard, unsigned short wBus, unsigned short wChannel);
tMuxStatus _MUXAPI **NwlSetNotification**(unsigned short wCard, unsigned short wBus, HANDLE hWinEvent);

Annex 8: I/O and timer management Prototypes

tMuxStatus _MUXAPI **IOSetOutput**(unsigned short wCard, unsigned short wOutputValue,
unsigned short wOutputMask);
tMuxStatus _MUXAPI **IOGetInput**(unsigned short wCard, unsigned short *wInputValue);

tMuxStatus _MUXAPI **TimerSet**(unsigned short wCard, unsigned short wTimerValue);
tMuxStatus _MUXAPI **TimerSetNotification**(unsigned short wCard, HANDLE hWinEvent);

Annex 9: Date stamping of PCI-MUX cards

PCI-MUX cards are not equipped with microprocessors whose role could be to date stamp the events transiting the network. Therefore, the access functions are in charge of carrying out this operation.
In most cases, date stamping is correct. However, with certain PC configurations (activated screensaver, anti-virus…), the PC might modify it.
Since this problem cannot be solved, a device has been designed to tell the user that the problem has taken place and that the date stamping of the events is not precise. For this reason, PCI-MUX cards come with a 1 ms cadence timer and with a control logic that can be treated from the PC.

When date stamping is correct, the PC immediately manages any interruptions coming from the card and timer's protocol controllers. The « wTimerError » parameter of the event structure equals 0.
When the PC does not immediately manage the interruptions, there is a shift between the date stamp on the event and the real time at which the event took place. If there is an interruption of the timer when another one is already present but not dealt with, then this information is memorised on the card and dealt with when the PC retakes management of the interruptions so as to increase « wTimeError ».
In short, « wTimeError » increases by 1 when it is proved that a date stamping error of 1 or more milliseconds has taken place.

Advice
In order to reduce the risk of date stamping error, it is advisable to :

- Suppress screensavers and hard disk savers
- Deactivate antivirus programmes
- Generally, close any application that operates parallel with the network application.
- …

Annex 10: Versions of the MUXDLL library

Version	Date	Evolutions / Modifications
6.1.6	28/10/2011	• End of support for Exxotest v1.x.x and Jungo v5.2.2 USB drivers • Support now Exxotest USB v2.x.x & above, Jungo PCI v6.0.3 (Kernel Plugin v1.2.4, VXD 4.34) • Remove cards counting from MuxInit, MuxPciGetCardInfo and MuxUsbGetCardInfo • Dynamically load Winsock 2 and IP Helper DLLs at main entry • Change MuxEthCountCards and add new function MuxEthGetCardInfo • Change MuxOpen & MuxClose for Ethernet cards (wired and wireless), update IP routing table if necessary • MuxWifiCountCards & MuxWifiGetCardInfo deprecated : use MuxEthCountCards & MuxEthGetCardInfo instead • Minor change in EEPROM initialization routine for PCI boards • MuxGetHardwareID function valid only for USB cards • CAN error events of same type now signaled only once per ms (PCI boards) • Minor change for LIN IFR handling (PCI boards) • Correction in length management for ISO 9141 & ISO 14230 protocols (PCI boards) • Add new function NMEA0183ConfigUart

Successive editions' list

Version	Date	Created / Modified by
1	19/10/2011	Gaël PERAGOUX
Modification		
Document's creation		
Version	**Date**	**Created / Modified by**
2	28/10/2011	Gaël PERAGOUX
Modification		
Document's update		
Version	**Date**	**Created / Modified by**
3	08/11/2011	Gaël PERAGOUX
Modification		
Document's update		