EXXOTest® MUXDLL

Bibliothèques d'accès aux cartes

Documentation utilisateur

Document N° 00272799-v6

ANNECY ELECTRONIQUE, créateur et fabricant de matériel : Exxotest, Navylec et 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. au Capital de 207 000€ - RC ANNECY 80 B 243 - SIRET 320 140 619 00042 - APE 2651B - N° TVA FR 37 320 140 619

TABLE DES MATIERES

1. But du document

1.1. But du document

Le but de ce document est de décrire les différentes fonctions composant les bibliothèques d'accès aux cartes de la gamme « Systèmes d'Expertise Réseaux de Communication » EXXOTest®.

Ces bibliothèques de fonctions logicielles permettent d'interfacer une application PC (ou Pocket PC) aux différentes cartes MUX.

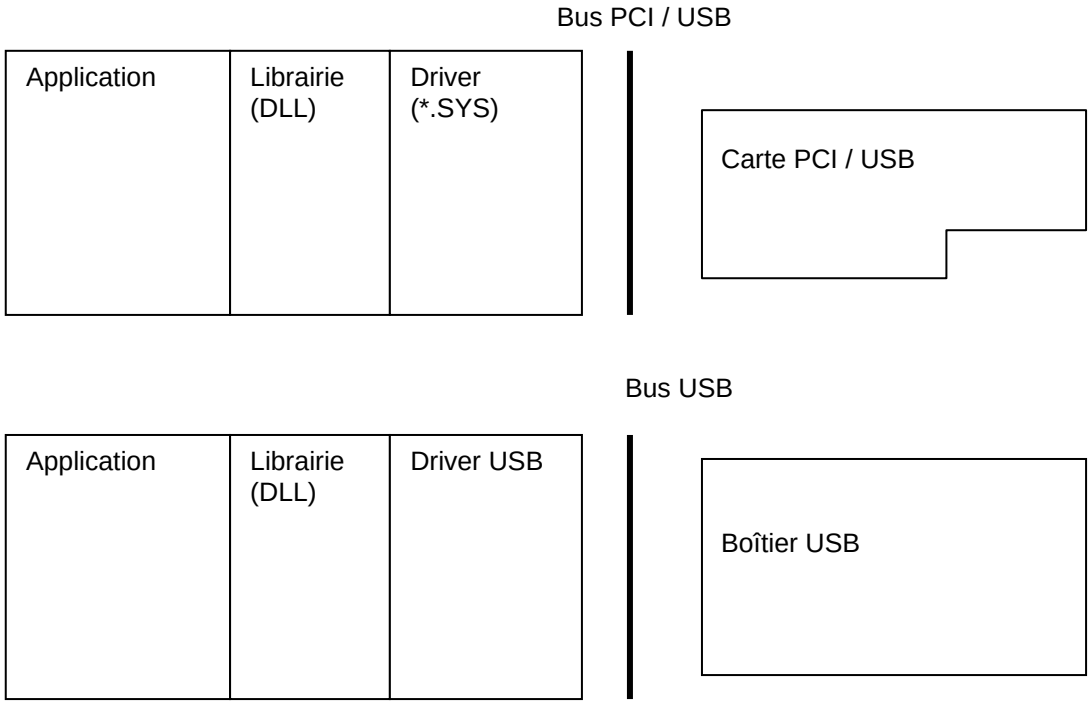
Les limitations sur les nombres de réseaux pouvant être interfacés dépendent du type de cartes utilisées, elles sont décrites dans le guide matériel de la carte.

L'utilisation des ces fonctions permet à l'utilisateur de rendre transparent l'application par rapport aux contrôleurs de protocoles et interfaces de lignes utilisées.

Toutes ces fonctions sont regroupées au sein d'une librairie dynamique (DLL), cette librairie est opérationnelle pour des PC équipés de systèmes d'exploitation Windows 2000, XP(32) et Vista(32) ou pour des Pocket PC équipés de systèmes d'exploitation Windows CE/Mobile (cible ARM).

2. Bibliothèques d'accès

2.1. Synoptique



2.2. Présentation

L'application dialogue avec les réseaux au moyen de différentes fonctions fournies par la librairie. Ces fonctions restent identiques quel que soit le système d'exploitation.

Les librairies permettent à une application d'accéder jusqu'à 8 cartes simultanément ainsi qu'à tous les réseaux disponibles sur celles-ci.

Les librairies permettent à plusieurs applications d'accéder chacune simultanément à plusieurs cartes différentes, mais ne permettent pas à plusieurs applications de partager une même carte.

2.3. Pilotes disponibles

Les librairies accèdent aux cartes à l'aide d'un pilote. C'est le pilote qui établit le lien entre le bus (USB / PCI …) et le matériel.

Trois generations de pilotes:

- **Windriver**: Premier pilote distribué avec les cartes MUX, il est validé pour les OS Windows 2000 à XP (windrvr.sys).

- **Exxotest v1.x**: Second pilote distribué, il est validé pour les OS 32 bits Windows XP à Seven. (exxusbw32.sys)

- **Exxotest v2.x**: Pilote actuellement distribué, il est validé pour les OS 32 et 64 bits Windows XP à Seven. (exxusb32.sys ou exxusb64.sys)

L'installation du pilote est facilitée par un setup « Exxotest Driver Kit and Utilities ». Celui-ci prépare le système d'exploitation à l'installation des pilotes des cartes MUX. Pour plus d'informations, consulter le guide utilisateur correspondant a la carte installée.

Il existe deux exceptions:

- Windows CE : Ce système d'exploitation nécessite un fichier DLL. (exxotest.dll) L'installation se fait par le fichier Setup_USBMUX.DAT.

- Linux : En cours de développement.

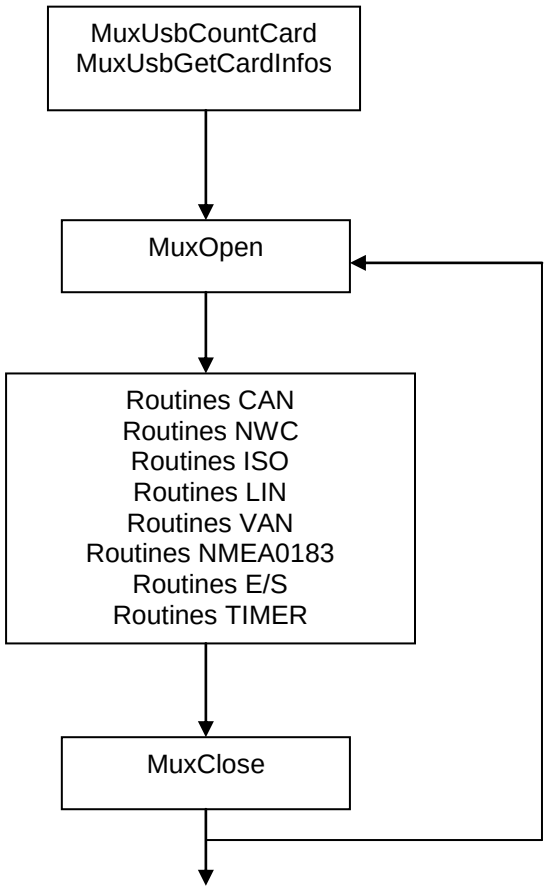
2.4. Liste des cartes accessibles par les fonctions de la bibliothèque

Toutes les cartes de la gamme « Systèmes d'Expertise Réseaux de Communication » EXXOTest® sont accessibles à l'aide des librairies logicielles, permettant ainsi à une application de fonctionner sans modifications quel que soit le type de carte utilisée.

3. Bibliothèque commune à tous les protocoles

3.1. Ordre d'appel

L'application doit respecter le séquencement suivant lors de l'appel des fonctions.



3.2. MuxCountCards: Retourne le nombre de cartes gérées par les DLLs

Prototype:
tMuxStatus MuxCountCards(unsigned long *dwCardsCount);

Description :
Cette fonction retourne le nombre de cartes connectées au PC.

Paramètres d'entrée :
Aucun

Paramètres de sortie :
dwCardsCount : Nombre de cartes détectées

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_OPEN	Erreur d'ouverture du driver

3.3. MuxPciCountCards: Retourne le nombre de cartes PCI-MUX

Prototype:
tMuxStatus MuxPciCountCards(unsigned long *dwCardsCount);

Description :
Cette fonction retourne le nombre de cartes présentes sur le bus PCI.

Paramètres d'entrée :
Aucun

Paramètres de sortie :
dwCardsCount : Nombre de cartes PCI-MUX détectées sur le bus PCI

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_OPEN	Erreur d'ouverture du driver

3.4. MuxUsbCountCards: Retourne le nombre de cartes USB-MUX

Prototype:
tMuxStatus MuxUsbCountCards(unsigned long *dwCardsCount);

Description :
Cette fonction retourne le nombre de cartes présentes sur le bus USB.

Paramètres d'entrée :
Aucun

Paramètres de sortie :
dwCardsCount : Nombre de carte USB-MUX détectée sur le bus USB

Code retour :

Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_OPEN	Erreur d'ouverture du driver

3.5. MuxEthCountCards: Retourne le nombre de cartes ETH-MUX

Prototype:
tMuxStatus MuxEthCountCards(unsigned long *dwCardsCount);

Description :
Cette fonction retourne le nombre de cartes présentes sur le réseau ETHERNET.

Paramètres d'entrée :
Aucun

Paramètres de sortie :
dwCardsCount : Nombre de carte ETH-MUX détectée sur le réseau ETHERNET

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_OPEN	Erreur d'ouverture du driver

3.6. MuxPciGetCardInfo: Retourne des informations sur les cartes PCI-MUX

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

Description :
Cette fonction retourne des informations sur les cartes présentes sur le bus PCI.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder

Paramètres de sortie :
dwCardBus : Numéro du bus sur lequel se trouve la carte
dwCardSlot : Numéro de slot sur lequel se trouve la carte
dwCardInfo : Informations supplémentaires

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_OPEN	Erreur d'ouverture du driver

3.7. MuxUsbGetCardInfo: Retourne des informations sur les cartes USB-MUX

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

Description :
Cette fonction retourne des informations sur les cartes présentes sur le bus USB.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder

Paramètres de sortie :
dwHubNum : Numéro du « HUB » lequel se trouve la carte
dwPortNum : Numéro de port sur lequel se trouve la carte
dwFullSpeed: 1 : Indique que la carte USB fonctionne au débit maximum de 12 Mbit/sec sur le bus USB.
dwDeviceAdress : Réservé pour utilisation interne
dwProductID: Informations supplémentaires
dwUniqueId : Réservé pour utilisation interne

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_OPEN	Erreur d'ouverture du driver
STATUS_ERR_TO_USB	Erreur de communication sur le bus USB

3.8. MuxEthGetCardInfo: Retourne des informations sur les cartes ETH-MUX

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

Description :
Cette fonction retourne des informations sur les cartes présentes sur le réseau ETHERNET.

Exemple:
```
char *pInfo;
unsigned short szInfo = 0;
```

```
if (MuxEthGetCardInfo(0, NULL, &szInfo) != STATUS_OK)
{ /* On récupère la taille minimale requise */
        pInfo = malloc(szInfo);
        if (MuxEthGetCardInfo (0, pInfo, &szInfo) != STATUS_OK)
        {
                printf("ERREUR");
        }
}
```

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
bInfo : Pointeur sur l'adresse à renseigner
wSize : Taille du buffer alloué à l'adresse renseignée

Paramètres de sortie :
bInfo : Pointeur sur les informations binaires
wSize : Taille du buffer d'information valide

- Le buffer retourné contient des informations qui peuvent être de longueur fixe ou variable et qui sont organisées suivant le format TLV (Type/Code, Longueur, Valeur/Données).

- Les informations de taille fixe sans données ne comportent qu'un octet de marquage indiquant le code. L'information de type 255, de taille fixe, indique la fin des informations. La valeur de l'octet de longueur indique la taille des données qui suivent.

Code	Description
0x01	Nature du lien ETHERNET (1 octet) Octet1 : 1=filaire, 2=sans fil
0x02	Version logicielle et identificateur (4 octets) Octets 1 et 2 : Version embarquée Octets 3 et 4 : ID de la carte
0x03	Informations produit (32 octets) Octets 1 à 16 : Référence produit (ASCII) Octets 17 à 32 : Numéro de série (ASCII)
0x04	Informations réseau (28 octets) Octets 1 à 6 : Adresse MAC cible Octets 7 à 10 : Adresse IP cible Octets 11 à 14 : Masque de sous réseau cible Octets 15 à 20 : Adresse MAC source Octets 21 à 24 : Adresse IP source Octets 25 à 28 : Masque de sous réseau source

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_MEMORY	Erreur d'allocation mémoire

3.9. MuxInit : Initialisation des variables internes des librairies

Prototype:
tMuxStatus MuxInit(unsigned int wCard);

Description :
Cette fonction initialise les états et variables internes des librairies. Cette fonction est la première fonction que doit appeler l'application (après MuxLoadDLL si chargement dynamique), cet appel est unique et se situe en début de programme.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_SEQUENCE	Erreur de séquence

3.10. MuxSetEthernet : Paramètre la fonctionnalité Ethernet.

Prototype:
tMuxStatus MuxSetEthernet (tETHState hEthState);

Description :
Cette fonction permet d'activer la recherche des cartes MUX (MuxEthCountCard) connectées au réseau Ethernet de l'ordinateur.

Paramètres d'entrée :
hEthState : Mode de fonctionnement de l'Ethernet

hEthState	ETH_Enabled : - La recherche des cartes Ethernet est activée. ETH_Disabled : - La recherche des cartes Ethernet est désactivée.

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK

3.11. MuxGetEthernet : Etat de la fonctionnalité Ethernet.

Prototype:
tMuxStatus MuxGetEthernet (tETHState *hEthState);

Description :
Cette fonction permet d'obtenir l'état de la fonctionnalité Ethernet.

Paramètres d'entrée :
hEthState : Mode de fonctionnement de l'Ethernet

hEthState	ETH_Enabled : - La recherche des cartes Ethernet est activée. ETH_Disabled : - La recherche des cartes Ethernet est désactivée.

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK

3.12. MuxOpen : Ouverture du driver

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

Description :
Cette fonction permet d'initialiser la communication avec le driver de la carte multiplexée.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
hMuxConfigMode : Mode de fonctionnement du driver

eMuxMode	MODE_APPLI : - Réservé MODE_KERNEL : - Valeur à programmer par l'application MODE_DEMO : - Réservé MODE_MUXTRACE : - Réservé
wBusInterface	Réservé

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_OPEN	Erreur d'ouverture du driver logiciel
STATUS_ERR_OPENKP	Erreur d'ouverture du driver matériel
STATUS_ERR_MEMORY	Erreur d'allocation mémoire
STATUS_ERR_BANK	Erreur repagination mémoire
STATUS_ERR_SEQUENCE	Erreur de séquence

3.13. MuxGetTimings : Retourne l'heure d'ouverture de la carte

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

Description :
Cette fonction retourne la datation du dernier muxopen.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder.
tBaseCounter:

Paramètres de sortie :
ptTimings :

3.14. MuxGetVersion : Retourne les versions des logiciels

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

Description :
Cette fonction retourne les versions de logiciels installés.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder

Paramètres de sortie :
DwVersionDll : Numéro de version des DLL
DwVersionDriver : Numéro de version du driver logiciel
DwVersionKernel : Numéro de version du driver matériel
eNumHWType : Type de carte

PCI_MUX_DEMO	Carte absente : version de démonstration
PCI_MUX_C3V2L	Carte PCI-MUX-C3V2L
PCI_MUX_CAN	Carte PCI-MUX-CAN
PCI_MUX_VAN	Carte PCI-MUX-VAN
PCI_MUX_MultiCAN	Carte PCI-MUX-4C2L
USB_MUX_C3VL	Carte USB-MUX-C3VL
USB_MUX_4C2L	Carte USB-MUX-4C2L
USB_MUX_CL	Carte USB-MUX-CL

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres

3.15. MuxClose : Fermeture du driver

Prototype :
tMuxStatus MuxClose(unsigned short wCard);

Description :
Cette fonction termine la communication avec la carte. Cette fonction est la dernière fonction appelée par l'application.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder

Paramètres de sortie :

Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres

3.16. MuxGetLastErrorString: Précision du type d'erreur

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

Description :
Cette fonction retourne un texte indiquant le contenu de la dernière erreur survenue.

Exemple:
```
char *pString

if (MuxOpen(0, MODE_KERNEL) != STATUS_OK)
{ /* Une erreur s'est produite */
        MuxGetLastErrorString(&pString);
        printf("ERREUR : %s",pString);
}
```

Paramètres d'entrée :
pString : Pointeur sur l'adresse à renseigner

Paramètres de sortie :
pString : Pointeur sur la chaine de texte

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK

3.17. MuxLoadDLL: Chargement dynamique de la librairie (Accès direct)

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

Description :

Cette fonction est nécessaire lorsque l'application désire charger les fonctions dynamiquement (Voir chapitre insertion des librairies au projet) en utilisant le mode d'accès direct aux cartes.
Cette fonction est la première fonction que doit appeler l'application, cet appel est unique et se situe en début de programme.

Exemple:
```
if (MuxLoadDLL() != STATUS_OK)
{ /* Une erreur s'est produite */
        printf("ERREUR chargement librairie");
}
```

Paramètres d'entrée :
Aucun

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK

3.18. MuxLoadDLLClient: Chargement dynamique de la librairie (Accès partagé)

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

Description :
Cette fonction est nécessaire lorsque l'application désire charger les fonctions dynamiquement (Voir chapitre : insertion des librairies au projet) en utilisant le serveur d'accès aux cartes. Ce mode de fonctionnement autorise plusieurs applications à se connecter sur une même carte.
Cette fonction est la première fonction que doit appeler l'application, cet appel est unique et se situe en début de programme.

Exemple:
```
if (MuxLoadDLLClient() != STATUS_OK)
{ /* Une erreur s'est produite */
        printf("ERREUR chargement librairie");
}
```

Paramètres d'entrée :
Aucun

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_OPENDLL	Erreur chargement des fonctions de la librairie
STATUS_ERR_SERVERNOTLOADED	Le serveur d'accès n'est pas en cours d'exécution

3.19. MuxFreeDLL: Libération du chargement dynamique de la librairie

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

Description:
Cette fonction est nécessaire lorsque l'application désire charger les fonctions dynamiquement (Voir chapitre. Insertion des librairies au projet).
Cette fonction est la dernière fonction que doit appeler l'application.

Exemple:
```
if (MuxFreeDLL() != STATUS_OK)
{ /* Une erreur s'est produite */
        printf(« ERREUR") ;
}
```

Paramètres d'entrée :
Aucun

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_OPENDLL	Erreur lors de l'appel de la fonction

4. Bibliothèque CAN

4.1. Ordre d'appel

L'application doit respecter le séquencement suivant lors de l'appel des fonctions.
X : signifie que la fonction est autorisée dans l'état en cours.
X ⇒ : signifie que la fonction est autorisée dans l'état en cours et passe à l'état suivant.

Etat	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 : Mode de fonctionnement des routines

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

Description :
Cette fonction détermine le mode d'interface entre l'application et la carte.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hCanOper : Type d'interface avec l'application

eCanOperMode	Mode de fonctionnement : CAN_OPER_ANA_FIFO - Mode analyse, stockage FIFO CAN_OPER_ANA_BUFF - Mode analyse, stockage Buffer
WfifoSize	Réservé pour utilisation future

- Le stockage FIFO : Dans ce mode, les événements à remonter à l'application sont stockés dans une file d'attente. Ces événements sont soit des fins de transmission, soit des réceptions, soit des erreurs…Lorsque l'application appelle la fonction CanGetEvent, le premier des événements (le plus ancien dans le temps) est retiré de la file.
  Si la file d'attente est pleine et qu'un événement intervient, alors un bit indiquant une perte d'événement est placé sur le dernier événement.

- Le stockage BUFFER : Dans ce mode, les événements à remonter à l'application sont dans un buffer unique alloué lors de la configuration du message (quelque soit le service de celui-ci). Ces événements sont soit des fins de transmission, soit des réceptions (pas les erreurs). Lorsque l'application appelle la fonction CanGetEvent, le handle de communication message permet de repérer le buffer à lire. Ce buffer contient le dernier événement reçu (le plus récent dans le temps).
  Lors de chaque lecture, le buffer est réinitialisé. Si entre deux lectures aucun événement n'est survenu, le compte rendu (EVENT_EMPTY) l'indique à l'application.

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK

STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

4.3. CanConfigBus : Configuration des paramètres du bus

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

Description :
Cette fonction permet de configurer l'ensemble des paramètres du bus.

Le débit du bus est calculé de la manière suivante :
Débit (8MHz/((1+TSEG1+TSEG2)*BRP))

Le point d'échantillonnage est calculé de la manière suivante :
Position du point = (1+TSEG1)(1+TSEG1+TSEG2)

Exemple : Un débit de 250 kbit/sec avec un point d'échantillonnage à 81%
BRP=2, TSEG1=12, TSEG2=3

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hCanBus : Paramètre du bus

wBRP	Baud rate prescaler [1-64]
wTSEG1	Délai avant le point d'échantillonnage [3-16].
wTSEG2	Délai après le point d'échantillonnage [2-8]
wSJW	Valeur maximale du saut de resynchronisation [1-4]

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

4.4. CanConfigParam : Configuration des paramètres supplémentaires

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

Description :
Cette fonction permet de configurer des paramètres supplémentaires de fonctionnement.

Pour une carte équipée d'un contrôleur de protocole CAN PHILIPPS SJA1000, les messages reçus par celui-ci sont analysés puis placés dans le buffer correspondant. Cette fonction permet entre autres, de paramétrer le filtre d'acceptance du contrôleur pour diminuer le nombre de messages reçus en provenance du réseau afin d'améliorer les performances du système.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hCanParam : Paramètres auxiliaires de fonctionnement

dwFiltIdent	Identificateur du filtre d'acceptance (0x000 par défaut) Le filtre d'acceptance est apposé sur le buffer de réception. [0-0x7FF] si identificateur standard [0x1FFFFFFF] si identificateur étendu
eTypeId	Type d'identificateur (CAN_ID_STD par défaut) CAN_ID_STD : Identificateur standard (11 bits) CAN_ID_XTD : Identificateur étendu (29 bits)
dwFiltMask	Masque du filtre d'acceptance (0x000 par défaut) [0-0x7FF] si identificateur standard [0x1FFFFFFF] si identificateur étendu
eAckEnable	CAN_TRUE (par défaut) - La génération de l'acquittement est validée, les messages reçus correctement avec demande d'acquittement sont acquittés CAN_FALSE - La génération de l'acquittement est inhibée (mode espion)
eRxAll	CAN_TRUE (par défaut) - Tous les messages reçus en plus de ceux programmés sont remontés à l'application par l'intermédiaire de la FIFO de réception CAN_FALSE - Seul les messages reçus programmés en réception sont remontés à l'application par l'intermédiaire de la FIFO de réception
wSpecialModes	Réservé pour utilisation future

Note filtre d'acceptance : Le filtre d'acceptance permet d'appliquer un filtre de sélection des messages reçus en provenance de réseaux pour limiter la charge sur le PC. Ce filtre est prioritaire sur les autres filtres.
Un identificateur est reçu si la condition suivante est vraie :

(Identificateur reçu ET type ET Masque) = (Identificateur programmé ET type ET Masque)

Par exemple :

1. Ident programmé = xxx, masque = 000 permet de recevoir tous les identificateurs
2. Ident programmé = xxx, masque = 7FF permet de recevoir uniquement l'identificateur xxx
3. Ident programmé = 001, masque = 001 permet de recevoir tous les identificateurs impaires

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

4.5. CanConfigClock : Configure l'horloge du contrôleur CAN

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

Description :
Cette fonction permet d'indiquer à la dll la fréquence de fonctionnement du contrôleur CAN du périphérique associé.
Pour les cartes avec un contrôleur:

- SJA1000 : fréquence fixé à 16 MHZ.
- TWINCAN : fréquence fixé à 40 MHZ.

Remarque : Si la fonction n'est pas appelée, seules les configurations compatibles avec la carte seront valides.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
eCanClock : configuration de la fréquence
CAN_CLOCK_16MHZ : 16 MHZ
CAN_CLOCK_40MHZ: 40 MHZ
CAN_CLOCK_5MHZ: 5 MHZ

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_WARNING	Carte non supportée

4.6. CanConfigRangeFilter : Configuration d'une plage d'acceptance

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

Description :
Cette fonction permet de définir une plage d'identifiant qui seront remonté à l'application. La plage sera continue entre les deux identifiants définie.

Cette fonction doit être appelée après la fonction CanConfigParam et remplace le filtre d'acceptance défini par celle-ci.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hCanRangeFilter : configuration de la plage

eTypeId	Type d'identificateur (CAN_ID_STD par défaut) CAN_ID_STD : Identificateur standard (11 bits) CAN_ID_XTD : Identificateur étendu (29 bits)
dwLowIdent	Identificateur bas du filtre sur plage. [0-0x7FF] si identificateur standard [0xFFFF] si identificateur étendu
dwHighIdent	Identificateur haut du filtre sur plage. [0-0x7FF] si identificateur standard [0xFFFF] si identificateur étendu

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK

STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_WARNING	Carte non supportée

4.7. CanConfigDualFilter : Configuration du filtre d'acceptance double

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

Description :
Cette fonction utilise la capacité du contrôleur de protocole CAN PHILIPPS SJA1000 pour définir 2 filtres d'acceptances distincts.

- Pour les identificateurs standards, les filtres sont apposés sur l'ensemble de l'identificateur (11 bits)
- Pour les identificateurs étendus, les filtres sont apposés sur les 2 premiers octets de l'identificateur (16 bits sur 29)

Cette fonction doit être appelée après la fonction CanConfigParam et remplace le filtre d'acceptance défini par celle-ci

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hCanDualFilter : Configuration des filtres

eTypeId	Type d'identificateur (CAN_ID_STD par défaut) CAN_ID_STD : Identificateur standard (11 bits) CAN_ID_XTD : Identificateur étendu (29 bits)
wFiltIdent1	Identificateur du filtre d'acceptance n°1. [0-0x7FF] si identificateur standard [0xFFFF] si identificateur étendu
dwFiltMask1	Masque du filtre d'acceptance n°1 [0-0x7FF] si identificateur standard [0xFFFF] si identificateur étendu
dFiltIdent2	Identificateur du filtre d'acceptance n°2. [0-0x7FF] si identificateur standard [0xFFFF] si identificateur étendu
dwFiltMask2	Masque du filtre d'acceptance n°2 [0-0x7FF] si identificateur standard [0xFFFF] si identificateur étendu

Note filtre d'acceptance : Le filtre d'acceptance permet d'appliquer un filtre de sélection des messages reçus en provenance de réseaux pour limiter la charge sur le PC. Ce filtre est prioritaire sur les autres filtres.

Un identificateur est reçu si la condition suivante est vraie :

(Identificateur reçu ET type ET Masque) = (wFiltIdent1 ET type ET dwFiltMask1) OU (wFiltIdent2 ET type ET dwFiltMask2)

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

4.8. CanConfigStat : Configuration des statistiques

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

Description :
Cette fonction permet de configurer la durée sur laquelle la charge bus est calculée. Une durée égale à 0 inhibe le calcul de charge.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wBusLoadTime : Ce paramètre définit la durée sur laquelle la charge bus est calculée. Il est exprimé en ms, une valeur égale à 0 indique qu'aucune charge bus n'est remontée à l'application. La charge du bus est remontée en mode FIFO par l'intermédiaire de l'événement EVENT_CAN_BUSLOAD et du paramètre wBusLoad. (0 par défaut)

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

4.9. CanConfigTransceiverHS: Configuration de la pente des signaux

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

Description :
Cette fonction permet de contrôler la pente des signaux sur les lignes CAN high et CAN low.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
eCanSlope: Pente des signaux CANH et CAN low.
CAN_SLOPE_CONTROL: Front couché
CAN_SLOPE_HIGH_SPEED: Front droit

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

4.10. CanSelectTransceiverHS: Configuration du type d'interface de ligne

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

Description :
Cette fonction permet de sélectionner le type d'interface de ligne du bus.
- CAN high speed
- CAN low speed – fault tolerant

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
eCanTxHS: Type d'interface de ligne.
CAN_TRUE: Interface high speed (par défaut)
CAN_FALSE: Interface low speed – fault tolerant

Paramètres de sortie :

Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_BOARD	Carte non supportée

4.11. CanConfigTransceiverLS: Configuration de l'interface de ligne low speed

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

Description :
Cette fonction permet de contrôler les différents signaux de contrôle de l'interface de ligne CAN low speed – fault tolerant.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wStandby : Valeur de la broche « stand by » [0-1]
wEnable : Valeur de la broche « enable » [0-1]
wWakeUp : Valeur de la broche « wake up » [0-1]

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_WARNING	Non supporté sur ce type de matériel

4.12. CanReadTransceiverLS : lecture de l'état de l'interface de ligne LS

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

Description :

Cette fonction permet de lire le niveau logique des broches INH et ERR de l'interface de ligne CAN low speed – fault tolerant.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]

Paramètres de sortie :
wLineState : état de l'interface
Bit 0 : Etat de la ligne ERR.
Bit 1 : Etat de l'état de la ligne INH.

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

4.13. CanConfigTerminationLS: Configuration des résistances de rappel du réseau CAN LS

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

Description :
Cette fonction permet de paramétrer différentes valeurs de résistance de terminaison de l'interface de ligne CANLS. Ces valeurs de résistances ont une influence sur l'impédance totale du réseau et par conséquence sur les pentes des transitions lors d'un passage récessif / dominant sur les lignes CANH et CANL.

A noter :
- L'impédance totale d'un réseau CANLS doit toujours être inférieure à 1Ko
- Cette fonction est disponible sur le canal CAN N°0 des boîtiers USB-MUX-4C2L et sur tous les canaux CAN des boîtiers MUX-4C4L et MUX-6C6L.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wValue : Valeur de la résistance de terminaison
wValue=0 Résistance de 15 Ko environ (6.8 Ko pour MUX-4C4L/6C6L)
wValue=1 Résistance de 6 Ko environ (2.2 Ko pour MUX-4C4L/6C6L)
wValue=2 Résistance de 1,3 Ko environ (1.6 Ko pour MUX-4C4L/6C6L)
wValue=3 Résistance de 0,5 Ko environ

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_WARNING	Non supporté sur ce type de matériel

4.14. CanConfigPeriodic: Programmation d'un message périodique

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

Description :
Cette fonction permet le démarrage, l'arrêt et la mise à jour de l'émission de messages périodiques.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wOffset : Indice du message périodique [0-15]
L'application dispose de 16 messages périodiques, ce paramètre correspond à l'indice du message que l'application désire accéder. Il est à noter que ces 16 messages sont traités de manière indépendante.

wParam : Paramètre du message
Bit 0 : Indicateur de fonctionnement du message périodique
- 0 : Le message périodique est arrêté
- 1 : Le message périodique est activé (démarrage de la périodique ou mise à jour)

Bit 1 : Indicateur de remontée des fins de transmissions
Dans un souci de performance, il n'est pas toujours nécessaire pour une application de recevoir les événements de fins de transmission liés à l'émission des messages périodiques.
- 0 : Les fins de transmissions ne sont pas remontées à l'application
- 1 : Les fins de transmissions sont remontées à l'application

hCanMsg : Paramètre du message à émettre

wHandleMsg	Index de communication Pour le mode BUFFER : - Cet index a été précédemment retourné par la fonction CanCreateMsg lors de la création du

message. Par rapport à la configuration, seul les données (bData) et leurs longueurs (wDataLen) sont prises en compte.

Pour le mode analyse FIFO :

- Les messages sont sérialisés par la DLL, ce paramètre n'est pas utilisé.

dwIdent	Identificateur CAN du message [0-0x7FF] si identificateur standard [0x1FFFFFFF] si identificateur étendu
eTypeId	Type d'identificateur CAN_ID_STD : Identificateur standard (11 bits) CAN_ID_XTD : Identificateur étendu (29 bits)
dwMask	Inutilisé
eService	CAN_SVC_TRANSMIT_DATA - Transmission de données CAN_SVC_TRANSMIT_RTR - Demande de transmission distante
lPeriod	Périodicité du message en ms.
dwReserved1	Réservé pour utilisation future
dwReserved2	Réservé pour utilisation future
wDataLen	Longueur maximale des données (émises ou reçues) [0-8]
bData	Contenu des données (pour émission)

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_MSGSVC	Service non supporté
STATUS_ERR_MSGEXCEED	Offset du message incorrect

4.15. CanConfigPeriodicList : Programmation d'une liste de message périodique

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

Description :

Cette fonction permet le démarrage, l'arrêt et la mise à jour de l'émission d'une liste de messages périodiques.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wPeriodicCount : Nombre de périodique contenu dans le tableau.
hPeriodicCanMsg : tableau contenant la liste des trames périodique.

- wOffset : Indice du message périodique [0-15]

L'application dispose de 16 messages périodiques, ce paramètre correspond à l'indice du message que l'application désire accéder. Il est à noter que ces 16 messages sont traités de manière indépendante.

- wParam : Paramètre du message

Bit 0 : Indicateur de fonctionnement du message périodique

- 0 : Le message périodique est arrêté
- 1 : Le message périodique est activé (démarrage de la périodique ou mise à jour)

Bit 1 : Indicateur de remontée des fins de transmissions
Dans un souci de performance, il n'est pas toujours nécessaire pour une application de recevoir les événements de fins de transmission liés à l'émission des messages périodiques.

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

- hCanMsg : Paramètre du message à émettre

wHandleMsg	Index de communication Pour le mode BUFFER : - Cet index a été précédemment retourné par la fonction CanCreateMsg lors de la création du message. Par rapport à la configuration, seul les données (bData) et leurs longueurs (wDataLen) sont prises en compte. Pour le mode analyse FIFO : - Les messages sont sérialisés par la DLL, ce paramètre n'est pas utilisé.
dwIdent	Identificateur CAN du message [0-0x7FF] si identificateur standard [0x1FFFFFFF] si identificateur étendu
eTypeId	Type d'identificateur CAN_ID_STD : Identificateur standard (11 bits) CAN_ID_XTD : Identificateur étendu (29 bits)
dwMask	Inutilisé
eService	CAN_SVC_TRANSMIT_DATA - Transmission de données CAN_SVC_TRANSMIT_RTR

- Demande de transmission distante

lPeriod	Périodicité du message en ms.
dwReserved1	Réservé pour utilisation future
dwReserved2	Réservé pour utilisation future
wDataLen	Longueur maximale des données (émises ou reçues) [0-8]
bData	Contenu des données (pour émission)

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_MSGEXCEED	Offset du message incorrect

4.16. CanCreateMsg : Configuration d'un message de communication

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

Description :
Cette fonction permet de déclarer un des messages gérés par l'application.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hCanMsg : Paramètres du message

wHandleMsg	Voir paramètre de sortie
dwIdent	Identificateur CAN du message [0-0x7FF] si identificateur standard [0x1FFFFFFF] si identificateur étendu
eTypeId	Type d'identificateur CAN_ID_STD : Identificateur standard (11 bits) CAN_ID_XTD : Identificateur étendu (29 bits)
dwMask	Masque de réception associée à l'identificateur du message. [0-0x7FF] si identificateur standard [0x1FFFFFFF] si identificateur étendu Le masque est utilisé pour les services de type (CAN_SVC_RECEIVE_DATA et CAN_SVC_RECEIVE_RTR)

	(C.F. note 1)
eService	CAN_SVC_TRANSMIT_DATA - Transmission de données CAN_SVC_RECEIVE_DATA - Réception de données CAN_SVC_TRANSMIT_RTR - Demande de transmission distante CAN_SVC_RECEIVE_RTR - Réception de demande de transmission distante (C.F. note 2)
lPeriod	Renseigner à 0 (Réservé pour utilisation future)
dwReserved1	Renseigner à 0 (Réservé pour utilisation future)
dwReserved2	Renseigner à 0 (Réservé pour utilisation future)
wDataLen	Longueur maximale des données (émises ou reçues) [0-8]
bData	Contenu des données (pour émission)

Paramètres de sortie :

wHandleMsg	Index de communication (valide uniquement en mode BUFFER). Cet index est retourné par les DLL à l'application pour indiquer le numéro de buffer alloué. Cet indice est utilisé par la suite pour récupérer les événements réseau par l'intermédiaire de la fonction CanGetEvent ou pour émettre des messages par l'intermédiaire de la fonction CanSendMessage.

Note 1 :
Le masque permet de recevoir une famille d'identificateurs (voir exemple de la fonction CanConfigParam). Néanmoins certaines précautions doivent être prises :

- Dans le mode analyse, le masque défini par la fonction CanConfigParam est prioritaire par rapport à ce masque.
- Quelque soit le mode, il est important qu'il n'y ait aucune intersection entre deux familles. Dans ce cas, l'identificateur serait reçu par le premier message correspondant déclaré.

Note 2 :
Dans le mode analyse FIFO, il n'est pas nécessaire de déclarer les messages initiateurs (service : CAN_SVC_TRANSMIT_DATA et CAN_SVC_TRANSMIT_RTR).

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_MSGEXCEED	Erreur trop de messages déclarés

4.17. CanSetNotification : Déclaration de l'événement application

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

Description :
Cette fonction permet à l'application d'utiliser les méthodes de synchronisation de tâches fournies par les systèmes d'exploitation Windows (API WIN32). Cette fonction permet de passer un handle d'événement créé par la fonction Windows « CreateEvent » aux DLL. Cet événement est utilisé en cours de communication lorsqu'un événement est remonté des DLL vers l'application (par exemple : un fin de transmission, une réception, une erreur…).
Dès lors, l'application peut se mettre en attente d'événements à l'aide des fonctions d'attente telles que WaitForSingleObject ou WaitForMultipleObject.

Exemple :

```
/* Demande d'indication d'événement */
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);
}
```

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hWinEvent : « handle » retourné par la fonction CreateEvent.

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

4.18. CanActivate : Démarrage de la communication

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

Description :
Cette fonction permet de démarrer la communication avec le réseau. Après exécution de cette fonction, les messages reçus sont acquittés et remontés en direction de l'application. L'application peut également émettre des messages.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

4.19. CanDeactivate : Arrêt de la communication

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

Description :
Cette fonction arrête la communication avec le réseau. Après exécution de cette fonction, les messages ne sont plus reçus, ni acquittés, ni émis.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

4.20. CanSendMsg: Emission d'un message

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

Description :
Cette fonction permet l'émission d'un message de donnée ou demande de transmission distante.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hCanMsg : Paramètre du message à émettre

wHandleMsg	Index de communication Pour le mode BUFFER : - Cet index a été précédemment retourné par la fonction CanCreateMsg lors de la création du message. Par rapport à la configuration, seul les données (bData) et leurs longueurs (wDataLen) sont prises en compte. Pour le mode analyse FIFO : - Les messages sont sérialisés par la DLL, ce paramètre n'est pas utilisé.
dwIdent	Identificateur CAN du message [0-0x7FF] si identificateur standard [0x1FFFFFFF] si identificateur étendu
eTypeId	Type d'identificateur CAN_ID_STD : Identificateur standard (11 bits) CAN_ID_XTD : Identificateur étendu (29 bits)
dwMask	Inutilisé
eService	CAN_SVC_TRANSMIT_DATA - Transmission de données CAN_SVC_TRANSMIT_RTR - Demande de transmission distante
lPeriod	Réservé pour utilisation future
dwReserved1	Réservé pour utilisation future
dwReserved2	Réservé pour utilisation future
wDataLen	Longueur maximale des données (émises ou reçues) [0-8]

bData	Contenu des données (pour émission)

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_FIFOFULL	Le nouveau message ne peut être pris en compte. La FIFO d'émission est saturée.
STATUS_ERR_BUSOFF	Le nouveau message ne peut être pris en compte. Le contrôleur de protocole CAN est déconnecté du bus.
STATUS_ERR_MSGSVC	Service non supporté
STATUS_ERR_HANDLE	Le paramètre wHandleMsg est invalide
STATUS_ERR_MSGBUSY	Le nouveau message ne peut être pris en compte. Le message précédent est en cours d'émission.

4.21. CanSendMsgList : Emission de plusieurs messages

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

Description :
Cette fonction permet l'émission d'un ou plusieurs messages de données ou demandes de transmission distante.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wMsgCount : nombre de message à émettre
hCanMsg : Paramètre du message à émettre

wHandleMsg	Index de communication Pour le mode BUFFER : - Cet index a été précédemment retourné par la fonction CanCreateMsg lors de la création du message. Par rapport à la configuration, seul les données (bData) et leurs longueurs (wDataLen) sont prises en compte. Pour le mode analyse FIFO :

- Les messages sont sérialisés par la DLL, ce paramètre n'est pas utilisé.

dwIdent	Identificateur CAN du message [0-0x7FF] si identificateur standard [0x1FFFFFFF] si identificateur étendu
eTypeId	Type d'identificateur CAN_ID_STD : Identificateur standard (11 bits) CAN_ID_XTD : Identificateur étendu (29 bits)
dwMask	Inutilisé
eService	CAN_SVC_TRANSMIT_DATA - Transmission de données CAN_SVC_TRANSMIT_RTR - Demande de transmission distante
lPeriod	Réservé pour utilisation future
dwReserved1	Réservé pour utilisation future
dwReserved2	Réservé pour utilisation future
wDataLen	Longueur maximale des données (émises ou reçues) [0-8]
bData	Contenu des données (pour émission)

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_FIFOFULL	Le nouveau message ne peut être pris en compte. La FIFO d'émission est saturée.
STATUS_ERR_BUSOFF	Le nouveau message ne peut être pris en compte. Le contrôleur de protocole CAN est déconnecté du bus.
STATUS_ERR_MSGSVC	Service non supporté
STATUS_ERR_HANDLE	Le paramètre wHandleMsg est invalide
STATUS_ERR_MSGBUSY	Le nouveau message ne peut être pris en compte. Le message précédent est en cours d'émission.

4.22. CanGetEvent: Lecture d'un événement

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

Description :
Cette fonction permet de récupérer un événement en provenance du réseau.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hCanEvent : Pointeur sur l'événement à renseigner

Paramètres de sortie :
hCanEvent : Evénement renseigné

wBus	Numéro de bus ayant généré l'événement
wHandleMsg	A titre indicatif : Numéro de canal interne au DLL (C.F. CanCreateMsg)
eTypeEvent	Type d'événement EVENT_EMPTY : Aucun événement n'est présent. EVENT_CAN_MSGTX : - Fin de transmission correcte d'un message EVENT_CAN_MSGRX : - Réception correcte d'un message EVENT_CAN_ERROR : - Erreur réseau EVENT_CAN_BUSCHANGE - Changement de l'état du contrôleur de protocole EVENT_CAN_BUSLOAD - Charge bus EVENT_TIMER - Timer applicatif EVENT_TIMEERROR - Perte d'IT timer EVENT_FIFO_OVF - Bit indiquant qu'un ou plusieurs événements suivants celui-ci a été perdu car la FIFO de réception était pleine.
dwIdent	Identificateur CAN du message [0-0x7FF] si identificateur standard [0x1FFFFFFF] si identificateur étendu
eTypeId	Type d'identificateur CAN_ID_STD : Identificateur standard (11 bits) CAN_ID_XTD : Identificateur étendu (29 bits)
dwTimeStamp	Heure d'arrivée de l'événement en multiple de 100 µSec (précision de 1 ms pour les cartes PCI-MUX). Ce paramètre est correct si wTimePrecision est nul. C.F. ANNEXE : horodatage des cartes PCI-MUX

wTimePrecision	Précision de la datation dwTimeStamp en multiple de 100 µSec.
eService	Pour les événements liés à un message, service du message (voir fonction CanCreateMsg).
wError	Pour les événements de type EVENT_CAN_ERROR, indication du type d'erreur : Bit 7 à 6 type d'erreur : - 0 : Erreur bit - 1 : Erreur de format - 2 : Erreur de stuffing - 3 : Autre type d'erreur Bit 5 Direction quand l'erreur s'est produite : - 0 : Pendant une transmission - 1 : Pendant une réception Bit 4 à 0 Champ où l'erreur s'est produite : - 0x02 : ID20 à ID21 - 0x03 : Start of frame - 0x04 : Bit SRTR - 0x05 : Bit IDE - 0x06 : ID20 à ID18 - 0x07 : ID17 à ID13 - 0x08 : Champ CRC - 0x09 : Bit reserved 0 - 0x0A : Champ DATA - 0x0B : Champ DLC - 0x0C : Bit RTR - 0x0D : Bit reserved 1 - 0x0E : ID4 à ID0 - 0x0F : ID12 à 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	Etat du contrôleur de protocole CAN - CAN_BUS_ACTIVE : Erreur active - CAN_BUS_PASSIVE : Erreur passive - CAN_BUS_OFF : Composant déconnecté (bus off)
wLineState	Bit 0 : Etat du bit ERR de l'interface de ligne low speed - 1 : La communication est en mode nominal.

	- 0 : La communication est en mode dégradé. Bit 1 : Etat de la ligne INH de l'interface low speed (uniquement carte PCI-MUX-CAN et PCI-MUX-MultiCAN) - 1 : La sortie INH est active - 0 : La sortie INH est inactive
dwReserved1	Réservé pour utilisation future
dwReserved2	Réservé pour utilisation future
wBusLoad	Pour l'événement de type EVENT_CAN_BUSLOAD, valeur de la charge bus en %.
wDataLen	Longueur des données [0-8]
bData	Buffer de données

Code retour :

Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

Exemple :

```
/* Indication d'événement */
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 :          /* Plus d'événement à traiter */
                break ;
        case EVENT_CAN_MSGTX :   /* Fin de transmission correcte */
                break ;
        case EVENT_CAN_MSGRX :   /* Réception correcte */
                break ;
        case EVENT_CAN_ERROR :    /* Erreur réseau*/
                break ;
        case EVENT_CAN_BUSCHANGE :/* Changement d'état du composant */
                break ;
        case EVENT_CAN_BUSLOAD :          /* Charge bus */
                break ;
        case EVENT_TIMER :          /* Timer applicatif */
                break ;
        case EVENT_TIMEERROR :    /* Perte IT timer */
                break ;
        default  :                          /* Réservé pour utilisation future */
```

```
            break ;
}
```

4.23. CanGetStat : Lecture des compteurs de statistiques

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

Description :
Cette fonction permet de lire les compteurs de fonctionnement du réseau.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hCanStat : Pointeur sur la zone statistique à renseigner

Paramètres de sortie :
hCanStat : Evénement à renseigner

wBusLoad	Charge réseau exprimée en pourcent
wReserved	Réservé pour utilisation future
dwTxRq	Nombre de demande de transmissions
dwTxOk	Nombre de fins de transmissions correctes
dwRxOk	Réception correcte
dwErrTxBit	Nombre d'erreurs de type Bit en transmission
dwErrRxBit	Nombre d'erreurs de type Bit en réception
dwErrTxForm	Nombre d'erreurs de format en transmission
dwErrRxForm	Nombre d'erreurs de format en réception
dwErrTxStuff	Nombre d'erreurs de stuffing en transmission
dwErrRxStuff	Nombre d'erreurs de stuffing en réception
dwErrTxOther	Nombre d'erreurs en transmission autres que celles précédemment signalées
dwErrRxSOther	Nombre d'erreurs de stuffing en réception autres que celles précédemment signalées.
dwErrFifoOvf	Nombre de messages perdus (FIFO de réception pleine)

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

4.24. CanIsBusActive : Fournit l'état du contrôleur CAN

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

Description :
Cette fonction permet de connaître l'état d'activation d'un bus CAN.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]

Paramètres de sortie :
wState : état du bus
0 = bus CAN inactif.
1 = bus CAN actif.

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

4.25. CanGetBusState : Lecture de l'état du contrôleur CAN

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

Description :
Cette fonction permet de lire directement l'état du contrôleur de protocole CAN. Cette fonction remonte également la valeur des compteurs de transmission et de réception servant à gérer les états erreur active, erreur passive et bus off.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
eCanChipState: Pointeur sur état à renseigner
bTxErrCount: Pointeur sur le compteur à renseigner
bRxErrCount: Pointeur sur le compteur à renseigner

Paramètres de sortie :
eCanChipState: Etat du composant
bTxErrCount: Compteur de transmission

bRxErrCount: Compteur de réception

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

4.26. CanBusOn : Reconnexion du contrôleur après bus off

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

Description :
Cette fonction permet de reconnecter le contrôleur de protocole CAN après détection d'un passage en bus off.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

4.27. CanReadByte: Lecture directe du contrôleur de protocole CAN

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

Description :
Cette fonction permet de lire directement les registres du contrôleur de protocole CAN.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder

wBus : Indice du numéro de bus [0-2]
wOffset : Offset de lecture
bData : Pointeur sur la valeur à lire

Paramètres de sortie :
bData : Valeur lue

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

4.28. CanWriteByte : Ecriture directe dans le contrôleur de protocole CAN

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

Description :
Cette fonction permet de décrire directement dans les registres du contrôleur de protocole CAN.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wOffset : Offset d'écriture
bData : Valeur à écrire

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

4.29. CanGetFifoRxLevel : Niveau de remplissage de la file d'attente événements

Prototype:

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

Description :
Cette fonction permet de lire le niveau de remplissage de la file d'attente des événements remontés à l'application.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wParam : Paramètres de la fonction
Bit 0 : Type de file d'attente (uniquement si boîtier USB)
- 0 : Le nombre d'événement remonté est celui situé dans la file d'attente coté PC
- 1 : Le nombre d'événement remonté est celui situé dans la file d'attente coté boitier USB

Paramètres de sortie :
wCount : Nombre d'événement actuellement en file d'attente
wMaxCount : Nombre d'événement maximum que peut contenir la file d'attente

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

4.30. CanClearFifoRx : Purge la fifo de réception

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

Description :
Cette fonction permet de purger la file d'attente des évènements de réception.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

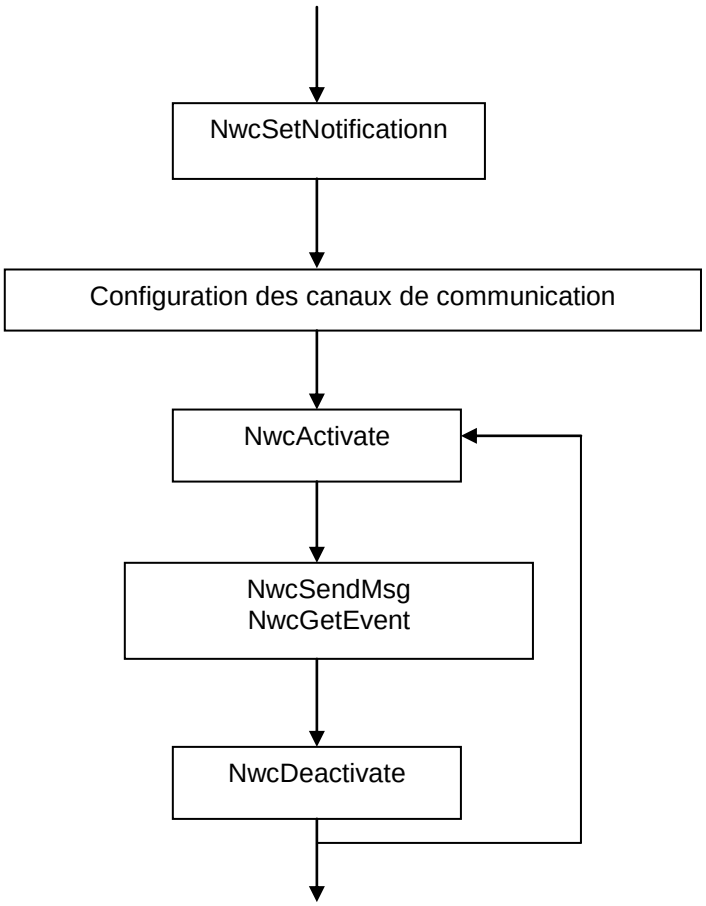
5. Bibliothèque NWC

5.1. Ordre d'appel

5.1.1 Séquence des échanges

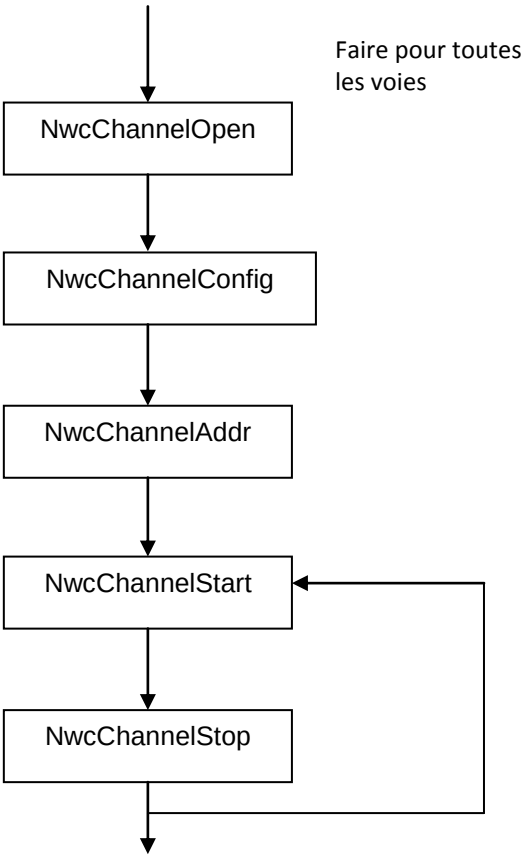
1ère étape : Configurer tous les canaux de communication : L'objectif est de renseigner les différents messages qui vont être utilisées par l'application ainsi que les différents paramètres de communication (identificateur CAN physique utilisés, délai et time-out).
2ème étape : Démarrer la communication à l'aide de la requête NwcActivate.
3ème étape : Communiquer avec les entités distantes à l'aide de deux requêtes principales.

- NwcSendMsg : Emission d'une trame de longueur compris entre 0 et 4095 octets.
- NwcGetEvent : Récupérer l'ensemble des événements stocké dans une file d'attente. Ces événements signalent des informations telles que fins de transmission, réceptions, fins de transmissions en erreur, réceptions en erreur.



5.1.2 Configuration des canaux de communication

L'application doit respecter la séquence suivante pour configurer les canaux. Il est à noter que les fonctions autorisant ou non la communication sur un canal (NwcChannelStart et NwcChannelStop) peuvent être appelées en cours de communication



5.2. NwcRegister: Autorisation d'utilisation des fonctions

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

Description :
Cette fonction autorise l'application d'utiliser les différentes fonctions de gestion des couches de communication CAN (obsolète depuis la version 5.40)

Paramètres d'entrée :
szString : Chaîne de texte d'enregistrement
szKey : Chaîne associée à la chaîne d'enregistrement

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

5.3. NwcGetChannelCount: Nombre de canaux disponible

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

Description :
La communication entre deux entités se fait au moyen d'un canal de communication. Ce canal est défini par son mode de fonctionnement (adressage physique ou fonctionnel, type d'adressage…), ses adresses de communication (TA, SA, AE…) et ses paramètres de communications (STMIN, BS, timeout…).
Plusieurs canaux de communication peuvent être ouvert simultanément, cette fonction retourne le nombre de canaux qu'il est possible d'ouvrir simultanément maximum.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]

Paramètres de sortie :

wChannelCount : Nombre de canaux disponibles.

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_REGISTER	Requête non enregistrée

5.4. NwcChannelOpen: Ouverture d'un canal de communication

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

Description :
Cette requête permet d'ouvrir un canal de communication. Ce canal de communication permet de relier logiquement deux entités communicantes.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]

Paramètres de sortie :
wChannel : Numéro de canal ouvert.

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_REGISTER	Requête non enregistrée
STATUS_ERR_MEMORY	Plus de canal disponible

5.5. NwcChannelConfig: Configuration d'un canal de communication

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

Description :

Cette requête permet de configurer un canal de communication.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wChannel : Indice du numéro de canal (obtenu par la fonction NwcChannelOpen)
hNwcChannelConfig: Paramètre du canal

eNwcAddrMode	Mode d'adressage : NWC_MODE_PHYSICAL : Adressage physique - Dans ce mode les messages sont transmis entre 2 entités physiques. Les échanges se font en point à point (un calculateur vers un calculteur). Ce mode autorise la segmentation des messages (jusqu'à 4095 octets de données). NWC_MODE_FUNCTIONNAL : Adressage fonctionnel - Dans ce mode les messages sont diffusés vers plusieurs entités. Les échanges se font en diffusion (un calculateur vers plusieurs calculateurs). Plusieurs calculateurs sont susceptibles de répondre à une requête. Ce mode n'autorise pas la segmentation (limité à 6 ou 7 octets de données en fonction du type d'adressage.
eNwcAddrFormat	Type d'adressage : NWC_FORMAT_NORMAL : Adressage normal - Dans ce mode, chaque combinaison d'adresse SA, TA et TAType est représentée par un identificateur de communication CAN choisi arbitrairement par le gestionnaire de réseau. L'identificateur peut être codé sur 11 ou 29 bits. NWC_FORMAT_NORMAL_FIXED: Adressage normal fixe - Ce mode est un sous format du mode normal. Dans ce mode, l'emplacement des adresses SA et TA est figé dans l'identificateur CAN. L'identificateur est codé sur 29 bits. NWC_FORMAT_EXTENDED: Adressage étendu - Dans ce mode, chaque combinaison d'adresse SA, et TAType est représentée par un identificateur de communication CAN choisi arbitrairement par le gestionnaire de réseau. Le premier octet de donnée contient l'adresse TA. L'identificateur peut être codé sur 11 ou 29 bits. NWC_FORMAT_MIXED: Adressage mixe - Dans ce mode, l'emplacement des adresses SA et TA est figé dans l'identificateur CAN. Le premier octet de donnée contient le champ AE (extended address).

	L'identificateur est codé sur 29 bits.
eNwcCommMode	NWC_COMM_HALF_DUPLEX (non supporté) : - Ce mode autorise la communication dans un seul sens en même temps. Il est impossible de recevoir en cours de transmission d'un message et inversement de transmettre en cours de réception. NWC_COMM_FULL_DUPLEX : - Ce mode autorise la communication dans les deux sens simultanément. Il est d'émettre et de recevoir simultanément.
eNwcService	Service de la requête : NWC_SVC_TRANSMIT_DATA : Transmission de données. NWC_SVC_RECEIVE_DATA : Réception de données.
wParam	Champ de bits définissant les paramètres de fonctionnement Longueur des trames : NWC_VARIABLE_LEN : La longueur des trames émises est variable en fonction des données utiles à émettre. NWC_FIXED_LEN : La longueur des trames émises est figée à 8 octets. Evénement niveau protocole : NWC_CAN_NO_EVENT : L'ensemble des échanges liés à la couche de communication n'est pas remonté à l'application. NWC_CAN_EVENT : L'ensemble des échanges liés à la couche de communication est remonté à l'application. Filtrage des FF (first frame) : NWC_FF_NO_EVENT : Les événements indiquant la réception ou la fin de transmission d'une FF ne sont pas remontés à l'application. NWC_FF_EVENT : Les événements indiquant la réception ou la fin de transmission d'une FF sont remontés à l'application.

Paramètres de sortie :
Aucun.

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_REGISTER	Requête non enregistrée

5.6. NwcChannelSetBytePadding : Personnalisation des octets de remplissage

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

Description :
Cette requête permet de configurer les octets de remplissage lors de la segmentation d'une trame. Ce remplissage a lieu lorsque le paramètre wParam de la fonction précédente possède le flag NWC_FIXED_LEN.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wChannel : Indice du numéro de canal (obtenu par la fonction NwcChannelOpen)
hNwcBytePadding : Paramètre de configuration

bData	Contient les octets qui seront utilisé lors du remplissage.
dwReserved1	Réservé pour une utilisation future
dwReserved2	Réservé pour une utilisation future

Paramètres de sortie :
Aucun.

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK
STATUS_ERR_PARAM Erreur de paramètres

5.7. NwcChannelAddr: Définition des identificateurs de communication

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

Description :
Cette requête permet de configurer les identificateurs du canal de communication.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]

wChannel : Indice du numéro de canal (obtenu par la fonction NwcChannelOpen)
hNwcChannelAddr : Paramètres de communication

bSA	"Source address" : Adresse source du message
bTA	« Target address» : Adresse du destinataire du message
bAE	« Address extended » : Adresse étendue
bReserved	Réservé pour alignement
dwIdentTx	Identificateur d'émission
eTypeIdTx	Type d'identificateur d'émission CAN_ID_STD : Identificateur standard (11 bits) CAN_ID_XTD : Identificateur étendu (29 bits)
dwIdentFC	Identificateur pour contrôle de flux
eTypeIdFC	Type d'identificateur pour contrôle de flux CAN_ID_STD : Identificateur standard (11 bits) CAN_ID_XTD : Identificateur étendu (29 bits)

Paramètres de sortie :
Aucun.

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_REGISTER	Requête non enregistrée

5.8. NwcChannelParam : Paramètres de communication du canal

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

Description :
Cette requête permet de configurer les paramètres de communication d'un canal. Ces paramètres permettent de gérer les délais d'émission des messages segmentés ainsi que le contrôle de flux. Il est possible de modifier dynamiquement ces paramètres sous réserve qu'il n'y ait aucune transmission ou réception en cours.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wChannel : Indice du numéro de canal (obtenu par la fonction NwcChannelOpen)
hNwcChannelParam: Paramètre de communication du canal

wBlockSize	Nombre de blocs consécutif émis [0-255] 0 indique qu'aucune trame de contrôle de flux n'est attendue	0
wSTmin	Délai d'émission entre deux transmissions consécutives [0-255] ms, correspond aussi au paramètre Cs Pour les canaux programmés en réception, ce paramètre correspond à la valeur du STMIN envoyé dans la trame FC. Pour les canaux programmés en transmission, ce paramètre vient s'ajouter à la valeur du STMIN reçu dans la trame FC pour générer le délai Cs (Temps entre deux CF)	10
wN_As	Délai max d'attente d'émission d'une trame côté émetteur [0-65535] ms	10
wN_Ar	Délai max d'attente d'émission d'une trame côté récepteur [0-65535] ms	10
wN_Bs	Délai jusqu'à réception du FC [0-65535] ms	20
wN_Br	Délai jusqu'à transmission du FC [0-65535] ms	0
wN_Cr	Délai jusqu'à réception du CF [0-65535] ms	20
wN_WFTmax	Nombre de FC max. attendue [0-65535]. 0 indique une attente infini.	0

Paramètres de sortie :
Aucun.

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_REGISTER	Requête non enregistrée
STATUS_ERR_BUSY	Communication en cours

5.9. NwcChannelStart : Autorise la communication sur le canal

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

Description :
Cette requête permet d'autoriser la communication sur le canal.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wChannel : Indice du numéro de canal (obtenu par la fonction NwcChannelOpen)

Paramètres de sortie :
Aucun.

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_REGISTER	Requête non enregistrée

5.10. NwcChannelStop : Arrête la communication sur le canal

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

Description :
Cette requête permet d'arrêter la communication sur le canal.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wChannel : Indice du numéro de canal (obtenu par la fonction NwcChannelOpen)

Paramètres de sortie :
Aucun.

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_REGISTER	Requête non enregistrée

5.11. NwcActivate : Démarrage de la communication

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

Description :

Cette fonction permet de démarrer la communication avec le réseau. Après exécution de cette fonction, les messages peuvent être émis et reçus par l'application.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_REGISTER	Requête non enregistrée

5.12. NwcDeactivate : Arrêt de la communication

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

Description :
Cette fonction arrête la communication avec le réseau. Après exécution de cette fonction, les messages ne sont plus reçus, ni émis.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_REGISTER	Requête non enregistrée

5.13. NwcChannelSendMsg: Emission d'un message

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

Description :
Cette fonction permet l'émission d'un message de donnée.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wChannel : Indice du numéro de canal (obtenu par la fonction NwcChannelOpen)
hNwcMsg : Paramètre du message à émettre

wDataLen	Longueur maximale des données (émises ou reçues) [0-4095]
bData	Contenu des données (pour émission)

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_MSGSVC	Le canal n'est pas un canal de transmission
STATUS_ERR_MSGBUSY	Le nouveau message ne peut être pris en compte. Le message précédent est en cours d'émission.

5.14. NwcGetEvent : Lecture d'un événement

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

Description :
Cette fonction permet de récupérer un événement en provenance du réseau.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]

hNwcEvent : Pointeur sur l'événement à renseigner

Paramètres de sortie :
hNwcEvent : Evénement renseigné

wBus	Numéro de bus ayant généré l'événement
wChannel	Numéro de canal ayant généré l'événement [0-n] ou 0xFFFF si événement du mode espion.
eTypeEvent	Type d'événement EVENT_EMPTY : Aucun événement n'est présent. EVENT_NWC_MSGTX : - Fin de transmission correcte d'un message EVENT_NWC_MSGRX : - Réception correcte d'un message EVENT_NWC_MSGTXERR : - Fin de transmission incorrecte EVENT_NWC_MSGRXERR - Réception incorrecte EVENT_NWC_MSGRXFF - Indication de réception d'une first frame EVENT_NWC_MSGTXFF - Indication de fin de transmission d'une first frame EVENT_FIFO_OVF - Bit indiquant qu'un ou plusieurs événements suivants celui-ci a été perdu car la FIFO de réception était pleine.
dwIdent	Identificateur CAN du message de transmission (FF ou SF) [0-0x7FF] si identificateur standard [0x1FFFFFFF] si identificateur étendu
eTypeId	Type d'identificateur CAN_ID_STD : Identificateur standard (11 bits) CAN_ID_XTD : Identificateur étendu (29 bits)
dwTimeStamp	Heure d'arrivée de l'événement en multiple de 100 µSec (précision de 1 ms pour les cartes PCI-MUX). Ce paramètre est correct si wTimePrecision est nul. C.F. ANNEXE : horodatage des cartes PCI-MUX
wTimePrecision	Précision de la datation dwTimeStamp en multiple de 100 µSec.
eService	Service de la requête NWC_SVC_TRANSMIT_DATA : Transmission de données NWC_SVC_RECEIVE_DATA : Réception de données.
eError	Pour les événements de type EVENT_NWC_MSGTXERR ou EVENT_NWC_MSGRXERR, indication du type d'erreur : - NWC_NO_ERR : Aucune erreur - NWC_ERR_TOUT_AS : Délai dépassé pour l'émission d'un message coté émetteur (paramètre wN_As) - NWC_ERR_TOUT_AR : Délai dépassé pour l'émission d'un message coté récepteur (paramètre wN_Ar) - NWC_ERR_TOUT_BS : Délai dépassé pour l'attente d'une

	trame FC « Flow Control » (paramètre wN_Bs) - NWC_ERR_TOUT_CR : Délai dépassé pour l'attente d'une trame CF « Consecutive Frame » (paramètre wN_Cr) - NWC_ERR_SN : Réception d'un numéro de séquence incorrecte. La réception est interrompue - NWC_ERR_FIFO_OVF : La file d'attente d'émission des messages CAN est pleine, la transmission ou réception en cours est interrompue. - NWC_ERR_MEMORY : Plus d'espace mémoire disponible pour traiter le message. - NWC_ERR_INV_PDU : Format de PDU reçu incorrecte
dwReserved1	Réservé pour utilisation future
dwReserved2	Réservé pour utilisation future
wDataLen	Longueur des données [0-4095]
bData	Buffer de données

Code retour :

Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

Exemple :

```
/* Indication d'événement */
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 :          /* Plus d'événement à traiter */
                break ;
        case EVENT_NWC_MSGTX :  /* Fin de transmission correcte */
                break ;
        case EVENT_NWC_MSGRX :  /* Réception correcte */
                break ;
        case EVENT_NWC_MSGTXERR :       /* Fin de transmission en erreur */
                break ;
        case EVENT_NWC_MSGRXERR :       /* Réception en erreur*/
                break ;
        case EVENT_NWC_MSGRXFF :        /* Réception d'une FF "First Frame" */
                break ;
        case EVENT_NWC_MSGTXFF :        /* Fin de transmission d'une FF */
```

```
            break ;
        default  :                      /* Réservé pour utilisation future */
            break ;
}
```

5.15. NwcGetIdent : Retourne les identificateurs CAN de communication

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

Description :
Cette fonction retourne les identificateurs CAN utilisés pour la communication en fonction des types et mode d'adressage sélectionné.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
pstNwcConfig : Pointeur sur une structure tNwcConfig (voir fonction NwcChannelConfig) renseignant sur le mode d'adressage.
pstNwcAddr : Pointeur sur une structure tNwcAddr (voir fonction NwcChannelAddr) renseignant sur les adresses de communication.

Paramètres de sortie :
PstNwcAddr : Contient les identificateurs CAN utilisés pour la communication

dwIdentTx	Identificateur d'émission
eTypeIdTx	Type d'identificateur d'émission CAN_ID_STD : Identificateur standard (11 bits) CAN_ID_XTD : Identificateur étendu (29 bits)
dwIdentFC	Identificateur pour contrôle de flux
eTypeIdFC	Type d'identificateur pour contrôle de flux CAN_ID_STD : Identificateur standard (11 bits) CAN_ID_XTD : Identificateur étendu (29 bits)

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

5.16. NwcGetFifoRxLevel : Niveau de remplissage de la file d'attente événements

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

Description :
Cette fonction permet de lire le niveau de remplissage de la file d'attente des événements remontés à l'application.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wParam : Paramètres de la fonction
Bit 0 : Type de file d'attente (uniquement si boîtier USB)
- 0 : Le nombre d'événement remonté est celui situé dans la file d'attente coté PC
- 1 : Réservé

Paramètres de sortie :
wCount : Nombre d'événement actuellement en file d'attente
wMaxCount : Nombre d'événement maximum que peut contenir la file d'attente

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

5.17. NwcConfigSpyMode : Configure le mode espion

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

Description :
Cette fonction permet de configurer le mode espion. L'espionnage du bus vérifie la présence d'une FF (first frame) ou SF(single frame), le bon enchaînement des trames segmentées (SN number) et retourne le message complet. L'événement retourné dispose du numéro de canal arbitraire 0x000.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hNwcSpyAddr: Paramètres de la fonction

eTypeId	Type d'identificateur CAN_ID_STD : Identificateur standard (11 bits) CAN_ID_XTD : Identificateur étendu (29 bits)
dwIdentReq	Identificateur de la requête ou FC réponse
dwIdentRsp	Identificateur de la réponse ou FC requête
dwMask	Masque apposé sur les identificateurs. Ce paramètre à l'aide des identificateurs dwIdentReq et dwIdentRsp permet de définir deux groupes d'identificateur à recevoir. Chaque identificateur reçu est interprété conformément à la norme ISO 15765-2..
wParam	Champ de bits définissant les paramètres de fonctionnement Longueur des trames NWC_VARIABLE_LEN : La longueur des trames est variable en fonction des données utiles à émettre. NWC_FIXED_LEN : La longueur des trames est figée à 8 octets Evénement niveau protocole NWC_CAN_NO_EVENT : L'ensemble des échanges liés à la couche de communication n'est pas remonté à l'application NWC_CAN_NO_EVENT : L'ensemble des échanges liés à la couche de communication est remonté à l'application Filtrage des FF (first frame) : NWC_FF_NO_EVENT : Les événements indiquant la réception ou la fin de transmission d'une FF ne sont pas remontés à l'application NWC_FF_NO_EVENT : Les événements indiquant la réception ou la fin de transmission d'une FF sont remontés à l'application

Paramètres de sortie :
Aucun.

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

5.18. NwcGetChannelState : Etat courant du canal

Prototype:

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

Description :
Cette fonction permet d'obtenir l'état courant du canal de communication.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wChannel : Indice du numéro de canal (obtenu par la fonction NwcChannelOpen)

Paramètres de sortie :
wChannelState : Etat courant du canal de communication
Bit NWC_CHANNEL_STATE_STARTED
0 : Le canal n'a pas été activé
1 : Le canal est activé (prêt à émettre ou recevoir)
Bit NWC_CHANNEL_STATE_BUSY
0 : Le canal est activé et au repos
1 : Le canal est activé et communique, une transmission ou une réception est en cours

eLastTxStatus : Status du dernier fin de transmission

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

5.19. NwcIsBusActive : Etat du bus

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

Description :
Cette fonction permet connaître si le bus à été activé à l'aide de la fonction NwcActivate.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]

Paramètres de sortie :
wState: Indique si le bus est activé ou non (valeur différent de 0)

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

5.20. NwcChannelClose : Fermeture d'un canal de communication

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

Description :
Cette fonction permet de fermer un canal de communication.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wChannel : Numéro du canal à fermer

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

5.21. NwcSetNotification : Déclaration de l'événement application

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

Description :
Cette fonction permet à l'application d'utiliser les méthodes de synchronisation de tâches fournies par les systèmes d'exploitation Windows (API WIN32). Cette fonction permet de passer un handle d'événement créé par la fonction Windows « CreateEvent » aux DLL. Cet événement est utilisé en cours de communication lorsqu'un événement est remonté des DLL vers l'application (par exemple : un fin de transmission, une réception, une erreur…).
Dès lors, l'application peut se mettre en attente d'événements à l'aide des fonctions d'attente telles que WaitForSingleObject ou WaitForMultipleObject.

Exemple :

```
/* Demande d'indication d'événement */
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);
}
```

Paramètres d'entrée :

wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hWinEvent : « handle » retourné par la fonction CreateEvent.

Paramètres de sortie :

Aucun

Code retour :

Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

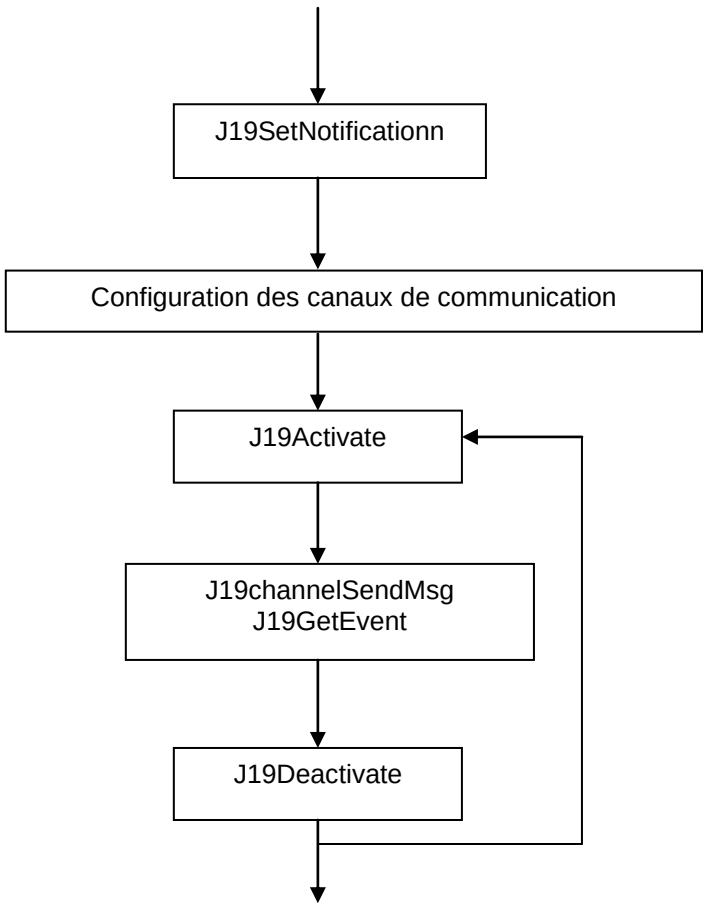
6. Bibliothèque J1939

6.1. Ordre d'appel

6.1.1 Séquence des échanges

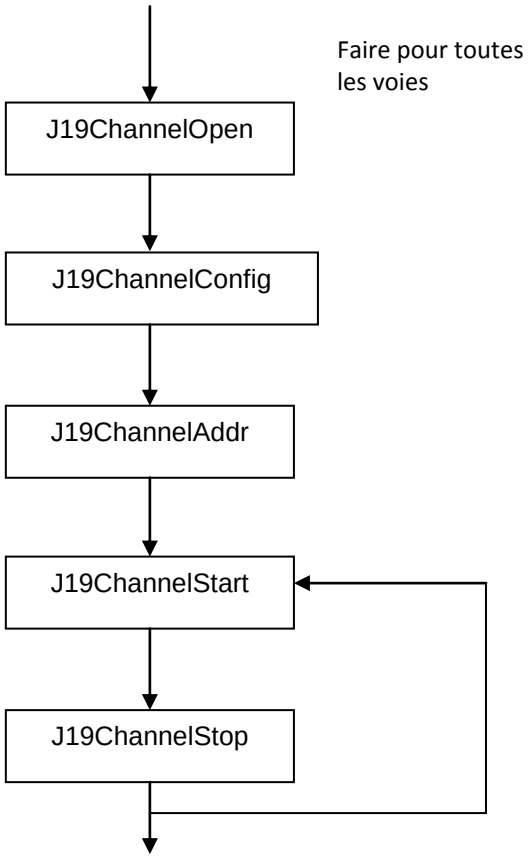
1ère étape Configurer tous les canaux de communication : L'objectif est de renseigner les différents messages qui vont être utilisées par l'application ainsi que les différents paramètres de communication (identificateur CAN physique utilisés, délai et time-out).
2ème étape Démarrer la communication à l'aide de la requête J19Activate
3ème étape Communiquer avec les entités distantes à l'aide de deux requêtes principales

- J19ChannelSendMsg: Emission d'une trame de longueur compris entre 0 et 1785 octets
- J19GetEvent : Récupérer l'ensemble des événements stocké dans une file d'attente. Ces événements signalent des informations telles que fins de transmission, réception, fins de transmissions en erreur, réception en erreur.



6.1.2 Configuration des canaux de communication

L'application doit respecter la séquence suivante pour configurer les canaux. Il est à noter que les fonctions autorisant ou non la communication sur un canal (J19ChannelStart et J19ChannelStop) peuvent être appelées en cours de communication.



6.2. J19ChannelOpen: Ouverture d'un canal de communication

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

Description :
Cette requête permet d'ouvrir un canal de communication. Ce canal de communication permet de relier logiquement deux entités communicantes.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]

Paramètres de sortie :
wChannel : Numéro de canal ouvert.

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_MEMORY	Plus de canal disponible

6.3. J19ChannelAddr: Définition des identificateurs de communication

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

Description :
Cette requête permet de configurer les identificateurs du canal de communication.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wChannel : Indice du numéro de canal (obtenu par la fonction J19ChannelOpen)
hJ19Addr: Paramètres de communication

bPriority	Priorité du message (de 0 à 7)
dwPGN	PGN qui sera géré par le canal
bSA	Adresse source du canal

Paramètres de sortie :
Aucun.

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

6.4. J19Param : Paramètres de communication du canal

Prototype:

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

Description :
Cette requête permet de configurer les paramètres de communication d'un canal. Ces paramètres permettent de gérer les délais d'émission des messages. Il est possible de modifier dynamiquement ces paramètres sous réserve qu'il n'y ait aucune transmission ou réception en cours.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hJ19ChannelParam: Paramètre de communication du canal

wTr	Temps maximum avant l'émission d'une réponse	200
wTh	Temps maximum avant l'émission d'une trame de maintien de communication (Non utilisé – prévu pour le futur)	500
wT1	Délai jusqu'à réception d'une autre trame TP.DT	750
wT2	Délai jusqu'à réception de la première trame TP.DT	1250
wT3	Délai jusqu'à réception de la première trame CTS.	1250
wT4	Délai jusqu'à réception d'une autre trame CTS.	1050

Paramètres de sortie :
Aucun.

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_BUSY	Communication en cours

6.5. J19ChannelConfig: Configuration d'un canal de communication

Prototype:
tMuxStatus J19ChannelConfig(unsigned short wCard, unsigned short wBus, unsigned short wChannel, tJ19Config *hJ19ChannelConfig);

Description :
Cette requête permet de configurer un canal de communication.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder

wBus : Indice du numéro de bus [0-x]
wChannel : Indice du numéro de canal (obtenu par la fonction J19ChannelOpen)
hJ19ChannelConfig: Paramètre du canal

eJ19Service	Service de la requête J19_SVC_TRANSMIT_DATA : Transmission de données J19_SVC_RECEIVE_DATA : Réception de données.
wParam	Non Utilisé. Réservé pour utilisation future.

Paramètres de sortie :
Aucun.

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

6.6. J19ConfigBus: Configuration d'un bus

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

Description :
Cette requête permet de configurer un canal de communication.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wParam: Champ de bits définissant les paramètres de fonctionnement
Evénement niveau protocole :

- J19_CAN_NO_EVENT : L'ensemble des échanges liés à la couche de communication n'est pas remonté à l'application
- J19_CAN_EVENT : L'ensemble des échanges liés à la couche de communication est remonté à l'application

Filtrage des RTS :

- J19_RTS_NO_EVENT : Les événements indiquant la réception ou la fin de transmission d'une trame RTS ne sont pas remontés à l'application
- J19_RTS_EVENT : Les événements indiquant la réception ou la fin de transmission d'une trame RTS sont remontés à l'application

Comportement :

- J19_SPY_MODE : Les trames seront traitées au niveau du BUS. Les transmissions BAM seront remontées dé segmentés.

6.7. J19ChannelStart : Autorise la communication sur le canal

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

Description :
Cette requête permet d'autoriser la communication sur le canal.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wChannel : Indice du numéro de canal (obtenu par la fonction J19ChannelOpen)

Paramètres de sortie :
Aucun.

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

6.8. J19ChannelStop : Arrête la communication sur le canal

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

Description :
Cette requête permet d'arrêter la communication sur le canal.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wChannel : Indice du numéro de canal (obtenu par la fonction J19ChannelOpen)

Paramètres de sortie :
Aucun.

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

6.9. J19Activate : Démarrage de la communication

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

Description :
Cette fonction permet de démarrer la communication avec le réseau. Après exécution de cette fonction, les messages peuvent être émis et reçus par l'application.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

6.10. J19Deactivate : Arrêt de la communication

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

Description :
Cette fonction arrête la communication avec le réseau. Après exécution de cette fonction, les messages ne sont plus reçus, ni émis.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

6.11. J19ChannelSendMsg: Emission d'un message

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

Description :
Cette fonction permet l'émission d'un message de donnée.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wChannel : Indice du numéro de canal (obtenu par la fonction J19ChannelOpen)
hJ19Msg : Paramètre du message à émettre

wDataLen	Longueur maximale des données (émises ou reçues) [0-1785]
bData	Contenu des données (pour émission)

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_MSGSVC	Le canal n'est pas un canal de transmission
STATUS_ERR_MSGBUSY	Le nouveau message ne peut être pris en compte. Le message précédent est en cours d'émission.

6.12. J19ChannelSendNMEAFastPacket: Emission d'un message

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

Description :
Cette fonction permet l'émission d'un message de donnée FAST PACKET (NMEA2000).

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wChannel : Indice du numéro de canal (obtenu par la fonction J19ChannelOpen)
hJ19Msg : Paramètre du message à émettre

wDataLen	Longueur maximale des données (émises ou reçues) [0-223]
bData	Contenu des données (pour émission)

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_MSGSVC	Le canal n'est pas un canal de transmission
STATUS_ERR_MSGBUSY	Le nouveau message ne peut être pris en compte. Le message précédent est en cours d'émission.

6.13. J19GetEvent : Lecture d'un événement

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

Description :
Cette fonction permet de récupérer un événement en provenance du réseau.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hJ19Event : Pointeur sur l'événement à renseigner

Paramètres de sortie :
hJ19Event : Evénement renseigné

wBus	Numéro de bus ayant généré l'événement
wChannel	Numéro de canal ayant généré l'événement [0-n] ou 0xFFFF si événement du mode espion.
eTypeEvent	Type d'événement EVENT_EMPTY : Aucun événement n'est présent. EVENT_J19_MSGTX : - Fin de transmission correcte d'un message EVENT_J19_MSGRX : - Réception correcte d'un message EVENT_J19_MSGTXERR : - Fin de transmission incorrecte EVENT_J19_MSGRXERR - Réception incorrecte EVENT_J19_MSGRXFF - Indication de réception d'une trame RTS EVENT_J19_MSGTXFF - Indication de fin de transmission d'une trame RTS EVENT_FIFO_OVF - Bit indiquant qu'un ou plusieurs événements suivants celui-ci a été perdu car la FIFO de réception était pleine.
dwIdent	Identificateur CAN du message de transmission [0x1FFFFFFF]
eTypeId	Type d'identificateur CAN_ID_XTD : Identificateur étendu (29 bits)
dwTimeStamp	Heure d'arrivée de l'événement en multiple de 100 µSec (précision de 1 ms pour les cartes PCI-MUX). Ce paramètre est correct si wTimePrecision est nul. C.F. ANNEXE : horodatage des cartes PCI-MUX
wTimePrecision	Précision de la datation dwTimeStamp en multiple de 100 µSec.
eService	Service de la requête J19_SVC_TRANSMIT_DATA : Transmission de données J19_SVC_RECEIVE_DATA : Réception de données.
eError	Pour les événements de type EVENT_J19_MWGTXERR ou EVENT_J19_MSGRXERR, indication du type d'erreur : - J19_NO_ERR : Aucune erreur - J19_ERR_TR : Délai dépassé pour l'émission d'un message coté émetteur - J19_ERR_TH : Délai dépassé pour l'émission d'un message de maintien de communication. - J19_ERR_T1 : Délai dépassé pour l'attente d'une autre trame TP.DT - J19_ERR_T2 : Délai dépassé pour l'attente d'une première trame TP.DT

	- J19_ERR_T3 : Délai dépassé pour l'attente d'une première trame CTS - J19_ERR_T4 : Délai dépassé pour l'attente d'une autre trame CTS
dwPGN	Numéro de PGN
bSA	Adresse source
dwReserved1	Réservé pour utilisation future
dwReserved2	Réservé pour utilisation future
wDataLen	Longueur des données [0-8]
bData	Buffer de données

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

Exemple :

```
/* Indication d'événement */
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 : Niveau de remplissage de la file d'attente événements

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

Description :
Cette fonction permet de lire le niveau de remplissage de la file d'attente des événements remontés à l'application.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wParam : Paramètres de la fonction
Bit 0 : Type de file d'attente (uniquement si boîtier USB)
- 0 : Le nombre d'événement remonté est celui situé dans la file d'attente coté PC
- 1 : Réservé

Paramètres de sortie :
wCount : Nombre d'événement actuellement en file d'attente
wMaxCount : Nombre d'événement maximum que peut contenir la file d'attente

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

6.15. J19StoreNMEAFastPacket : Réception d'un message FastPacket

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

Description :
Cette fonction permet de reconstituer une trame FAST PACKET (NMEA2000) en fonction de l'identificateur et de sa taille.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
dwIdent : Identificateur de la trame
wSize : Taille de la trame à reconstituée

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

6.16. J19ChannelClose : Fermeture d'un canal de communication

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

Description :
Cette fonction permet de fermer un canal de communication.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wChannel : Numéro du canal à fermer

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

6.17. J19SetNotification : Déclaration de l'événement application

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

Description :
Cette fonction permet à l'application d'utiliser les méthodes de synchronisation de tâches fournies par les systèmes d'exploitation Windows (API WIN32). Cette fonction permet de passer un handle d'événement créé par la fonction Windows « CreateEvent » aux DLL. Cet événement est utilisé en cours de communication lorsqu'un événement est remonté des DLL vers l'application (par exemple : un fin de transmission, une réception, une erreur…).

Dès lors, l'application peut se mettre en attente d'événements à l'aide des fonctions d'attente telles que WaitForSingleObject ou WaitForMultipleObject.

Exemple :

```
/* Demande d'indication d'événement */
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);
}
```

Paramètres d'entrée :

wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hWinEvent : « handle » retourné par la fonction CreateEvent.

Paramètres de sortie :

Aucun

Code retour :

Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

7. Bibliothèque ISO

7.1. Ordre d'appel

L'application doit respecter le séquencement suivant lors de l'appel des fonctions.
X : signifie que la fonction est autorisée dans l'état en cours recense.
X ⇒ : signifie que la fonction est autorisée dans l'état en cours et passe à l'état suivant.

Etat	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 : Mode de fonctionnement des routines

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

Description :
Cette fonction détermine le mode d'interface entre l'application et la carte.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hIsoOper : Type d'interface avec l'application

eIsoOperMode Mode de fonctionnement :

ISO_OPER_TESTER
- Mode testeur : l'application désire simuler un outil testeur. Les fonctions autoriseront la transmission de requêtes et la réception de réponse

ISO_OPER_ANALYZER
- Mode analyseur : l'application désire se comporter en espion pour analyser la communication entre un testeur et un calculateur

ISO_OPER_ SIMU
- Mode simulateur : l'application désire se comporter en calculateur. Sur réception d'une requête, il est possible d'émettre une réponse. (Attention : ce mode ne gère pas la séquence d'initialisation à 5 bauds)

wFifoSize Réservé pour utilisation future

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

7.3. IsoConfigBus : Configuration des paramètres du bus

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

Description :
Cette fonction permet de configurer l'ensemble des paramètres du bus.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hIsoBus : Paramètre du bus

Paramètres liées à une trame d'initialisation à 5 bauds

wW0	Délai du non activité du bus avant émission d'une trame d'initialisation. (Mode TESTER uniquement) Valeur conseillée 300 ms
wW1	Délai d'attente du code synchro. (Mode TESTER uniquement) Valeur conseillée 300 ms

wW2	Délai d'attente du mot clef 1. (Mode TESTER uniquement) Valeur conseillée 20 ms
wW3	Délai d'attente du mot clef 2. (Mode TESTER uniquement) Valeur conseillée 20 ms
wW4a	Délai avant émission du mot clef 2 inversé. (Mode TESTER uniquement) Valeur conseillée 25 ms
wW4b	Délai d'attente du code adresse inversé. (Mode TESTER uniquement) Valeur conseillée 50 ms
wW5	Délai avant répétition d'une initialisation. (Mode TESTER uniquement) Valeur conseillée 300 ms
WP0	Délai avant émission de la 1ere requête et la séquence d'initialisation. (Mode TESTER uniquement) Valeur conseillée 5 ms
eParity	Parité du code adresse à 5 bauds - ISO_PARITY_NOCHANGE : Pas de parité - ISO_PARITY_ODD : Parité impaire - ISO_PARITY_EVEN : Parité paire

Paramètres liées à une trame d'initialisation rapide

wTIdle	Délai du non activité du bus avant émission d'une trame d'initialisation. (Mode TESTER uniquement) Valeur conseillée 300 ms
wTInitL	Durée du pattern sur la ligne L Valeur conseillée 25 ms
wTWup	Durée totale de la séquence d'initialisation Valeur conseillée 50 ms

Paramètres de communication

eBaudRate	ISO_BAUD_9600 : 9600 bauds ISO_BAUD_10400 : 10400 bauds ISO_BAUD_62500 : 62500 bauds
wP1	Délai inter-caractères de la réponse
wP2	Délai entre une requête et réponse
wP3	Délai entre la réponse et une nouvelle requête
wP4	Délai inter-caractères de la requête

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

7.4. IsoConfigParam : Configuration des paramètres avancés

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

Description :
Cette fonction permet de configurer des paramètres avancés de fonctionnement.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hIsoParam : Paramètres avancés de fonctionnement

wFiltTP	Ce paramètre permet de filtrer les requêtes et réponses dont la fonction est TesterPresent (maintien de la communication) 0 : Pas de filtre 1 : Filtre du TesterPresent (les événements ne sont pas remontés à l'application)
wSpecialModes	Réservé pour utilisation future

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

7.5. IsoConfigStat : Configuration des statistiques

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

Description :
Cette fonction permet de configurer la durée sur laquelle la charge bus est calculée. Une durée égale à 0 inhibe le calcul de charge.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wBusLoadTime : Ce paramètre définit la durée sur laquelle la charge bus est calculée. Il est exprimé en ms, une valeur égale à 0 indique qu'aucune charge bus n'est remontée à l'application. La charge bus est remontée en mode FIFO par l'intermédiaire de l'événement EVENT_ISO_BUSLOAD et du paramètre wBusLoad. (0 par défaut)

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

7.6. IsoSetNotification : Déclaration de l'événement application

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

Description :
Cette fonction permet à l'application d'utiliser les méthodes de synchronisation de tâches fournies par les systèmes d'exploitation Windows (API WIN32). Cette fonction permet de passer un handle d'événement créé par la fonction Windows « CreateEvent » aux DLL. Cet événement est utilisé en cours de communication lorsqu'un événement est remonté des DLL vers l'application (par exemple : un fin de transmission, une réception, une erreur…).
Dès lors, l'application peut se mettre en attente d'événements à l'aide des fonctions d'attente telles que WaitForSingleObject ou WaitForMultipleObject.

Exemple :
```
/* Demande d'indication d'événement */
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);
}
```

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hWinEvent : « handle » retourné par la fonction CreateEvent.

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

7.7. IsoActivate : Démarrage de la communication

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

Description :
Cette fonction permet de démarrer la communication avec le réseau. Après exécution de cette fonction, les messages sont remontés en direction de l'application ou peuvent être émis par celle-ci.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

7.8. IsoDeactivate : Arrêt de la communication

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

Description :
Cette fonction arrête la communication avec le réseau. Après exécution de cette fonction, les messages ne sont plus reçus, ni émis.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

7.9. IsoConfigPeriodic: Programmation d'un message périodique

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

Description :
Cette fonction permet le démarrage, l'arrêt et la mise à jour de l'émission de messages périodiques. Cette fonction est souvent utilisée dans le cadre de la gestion de maintien de la communication (requête Tester Present)

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wOffset : Indice du message périodique [0-1]

L'application dispose de 2 messages périodiques, ce paramètre correspond à l'indice du message que l'application désire accéder. Il est à noter que ces 2 messages sont traités de manière indépendante.

wParam : Paramètre du message
Bit 0 : Indicateur de fonctionnement du message périodique
- 0 : Le message périodique est arrêté
- 1 : Le message périodique est activé (démarrage de la périodique ou mise à jour)

hIsoMsg : Paramètre du message à émettre

eTypeMsg	ISO_MSG_NORMAL : Emission d'une requête ISO_MSG_INIT5BDS : Emission d'une requête précédée d'une séquence d'initialisation à 5 bauds ISO_MSG_INITFAST : Emission d'une requête précédée d'une séquence d'initialisation rapide
wCodeAddr	Valeur du code adresse de l'initialisation à 5 bauds
LPeriod	Périodicité du message en ms.
wDataLen	Longueur maximale des données (émises ou reçues) [0-254]
bData	Contenu des données (pour émission)

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

7.10. IsoSendMsg: Emission d'un message

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

Description :
Cette fonction permet en mode testeur :
- l'émission d'une requête et l'attente d'une réponse

Cette fonction permet en mode simulation :
- L'émission d'une réponse.

Cette fonction est inactive en mode analyseur

Cette fonction permet l'émission de données brutes. Les données d'entête et de CRC sont à la charge de l'application.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hIsoMsg : Paramètre du message à émettre

eTypeMsg	ISO_MSG_NORMAL : Emission d'une requête ISO_MSG_INIT5BDS : Emission d'une requête précédée d'une séquence d'initialisation à 5 bauds ISO_MSG_INITFAST : Emission d'une requête précédée d'une séquence d'initialisation rapide ISO_MSG_WAITRESP : Se mets en attente d'une réponse sans émettre de requête au préalable.
wCodeAddr	Valeur du code adresse de l'initialisation à 5 bauds
lPeriod	Réservé pour utilisation future
wDataLen	Longueur maximale des données (émises ou reçues) [0-254]
bData	Contenu des données (pour émission)

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_FIFOFULL	Le nouveau message ne peut être pris en compte. La FIFO d'émission est saturée.
STATUS_ERR_MSGSVC	Service non supporté

7.11. Iso14230SendMsg: Emission d'un message

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

Description :
Cette fonction permet en mode testeur :
- l'émission d'une requête et l'attente d'une réponse

Cette fonction permet en mode simulation :
- L'émission d'une réponse.

Cette fonction est inactive en mode analyseur

Cette fonction est identique à la fonction IsoSendMsg. Cette fonction génère automatiquement les octets d'entête et de checksum du protocole KWP2000.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hIso14230Msg : Paramètre du message à émettre

eTypeMsg	ISO_MSG_NORMAL : Emission d'une requête ISO_MSG_INIT5BDS : Emission d'une requête précédée d'une séquence d'initialisation à 5 bauds ISO_MSG_INITFAST : Emission d'une requête précédée d'une séquence d'initialisation rapide
eIsoTypeHeader	Format des octets d'entête ISO_HEADER_NOADDR : Aucune information d'adresse ISO_HEADER_CARB : Mode d'exception CARB ISO_HEADER_PHYS : Entête avec information d'adressage physique ISO_HEADER_FUNCT : Entête avec information d'adressage fonctionnel
wCodeAddr	Valeur du code adresse de l'initialisation à 5 bauds
lPeriod	Réservé pour utilisation future
wSrcAddr	Adresse source
wDstAddr	Adresse cible
wDataLen	Longueur maximale des données (émises ou reçues) [0-255]
bData	Contenu des données (pour émission)

Paramètres de sortie :
Aucun

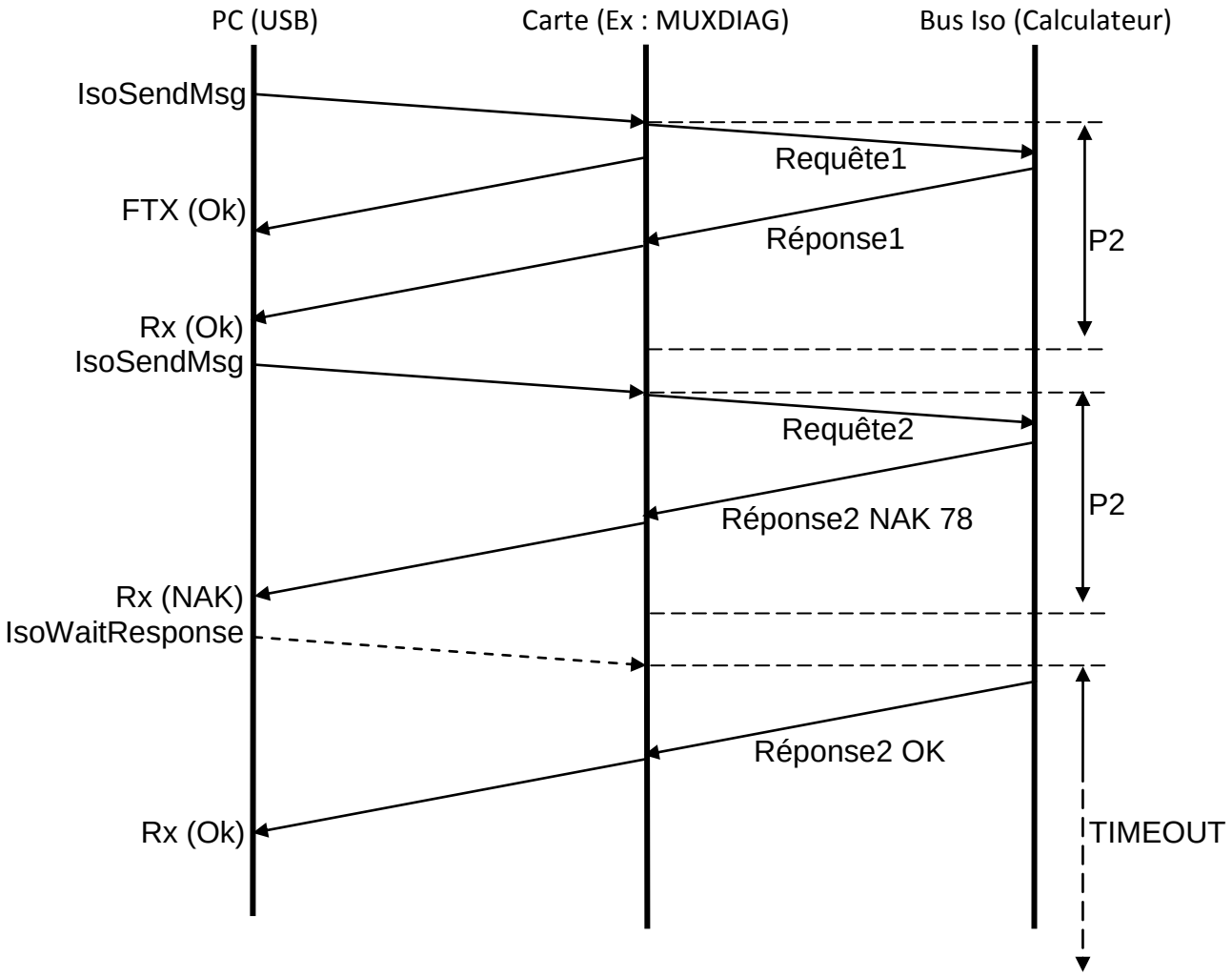
Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_FIFOFULL	Le nouveau message ne peut être pris en compte. La FIFO d'émission est saturée.
STATUS_ERR_MSGSVC	Service non supporté

7.12. IsoWaitResponse : Attente d'une nouvelle réponse

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

Description :
Cette fonction utilisée en mode testeur permet de relancer l'attente d'une réponse pour la gestion des requêtes longues.
En effet, la durée d'exécution de certaines requêtes peut dépasser le temps P2. Pour maintenir la communication, le calculateur envoie une réponse négative avec le code 0x78 (requête correctement reçue, réponse en attente). Après réception de cette réponse, l'application appelle la fonction « IsoWaitResponse » pour relancer l'attente d'une nouvelle réponse sans émettre de nouvelle requête.



Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wTimeOut : Délai d'attente de la réponse

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_FIFOFULL	Le nouveau message ne peut être pris en compte. La FIFO d'émission est saturée.

7.13. IsoChangeBaudRate : Changement de débit de la ligne

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

Description :
Cette fonction permet de changer le débit de communication en cours de fonctionnement

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
eBaudRateTx : Nouveau débit de communication
eBaudRateRx : Réservé

Note : Se reporter à la fonction IsoConfigBus pour obtenir la liste des débits autorisés

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

7.14. IsoGetEvent: Lecture d'un événement

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

Description :
Cette fonction permet de récupérer un événement en provenance du réseau.

Paramètres d'entrée :

wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hIsoEvent : Pointeur sur l'événement à renseigner

Paramètres de sortie :
hIsoEvent : Evénement renseigné

wBus	Numéro de bus ayant généré l'événement
wHandleMsg	Réservé pour utilisation future
eTypeEvent	Type d'événement EVENT_EMPTY : Aucun événement n'est présent. EVENT_ISO_MSGTX : - Fin de transmission correcte d'un message EVENT_ISO_MSGRX : - Réception correcte d'un message EVENT_ISO_MSGTXERR : - Fin de transmission en erreur d'un message EVENT_ISO_MSGRXERR : - Réception en erreur d'un message EVENT_ISO_ERROR - Réception d'un caractère en erreur (framing error, break…) EVENT_ISO_BUSLOAD - Charge bus EVENT_TIMER - Timer applicatif EVENT_TIMEERROR - Perte d'IT timer EVENT_FIFO_OVF - Bit indiquant qu'un ou plusieurs événements suivants celui-ci a été perdu car la FIFO de réception était pleine.
dwTimeStamp	Heure d'arrivée de l'événement en multiple de 100 µSec (précision de 1 ms pour les cartes PCI-MUX). Ce paramètre est correct si wTimePrecision est nul. C.F. ANNEXE : horodatage des cartes PCI-MUX
wTimePrecision	Précision de la datation dwTimeStamp en multiple de 100 µSec.
eError	Indication du type de l'erreur - Pour les événements de type EVENT_ISO_ERROR : ISO_ERR_FE : Erreur de bit de stop ISO_ERR_PE : Erreur de bit de parité ISO_ERR_OE : Erreur de dépassement du buffer de caractère ISO_ERR_BREAK : Détection d'un break - Pour les événements de type EVENT_ISO_MSGTXERR ou EVENT_ISO_MSGRXERR :

ISO_ERR_NOECHO : Caractère émis non reçu en écho
ISO_ERR_BADECHO : Echo incorrect
ISO_ERR_TO_SYNCHRO : Time out attente du caractère de synchronisation
ISO_ERR_BAD_SYNCHRO : Valeur du caractère de synchronisation incorrecte
ISO_ERR_TO_KEY1 : Time out attente du mot clef 1
ISO_ERR_BAD_KEY1 : Valeur du mot clef 1 incorrecte
ISO_ERR_TO_KEY2 : Time out attente du mot clef 2
ISO_ERR_BAD_KEY2 : Valeur du mot clef 2 incorrecte
ISO_ERR_TO_ADDR : Time out attente de l'adresse inversée
ISO_ERR_BAD_ADDR : Valeur de l'adresse inversée incorrecte
ISO_ERR_TO_RESP : Time out attente de la réponse
ISO_ERR_RX_OVER : Longueur de la réponse trop longue
ISO_ERR_BAD_LEN : Longueur de la réponse incorrecte
ISO_ERR_BAD_CRC : Checksum de la réponse incorrect

wBusLoad	Pour l'événement de type EVENT_ISO_BUSLOAD, valeur de la charge bus en %.
dwReserved1	Réservé pour utilisation future
dwReserved2	Réservé pour utilisation future
wDataLen	Longueur des données [0-254]
bData	Buffer de données

Code retour :

Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

Exemple :

```
/* Indication d'événement */
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 :              /* Plus d'événement à traiter */
                break ;
        case EVENT_ISO_MSGTX :          /* Fin de transmission correcte */
                break ;
        case EVENT_ISO_MSGRX :          /* Réception correcte */
```

```
            break ;
        case EVENT_ISO_MSGTXERR :       /* Erreur fin de transmission */
            break ;
        case EVENT_ISO_MSGRXERR :       /* Erreur reception */
            break ;
        case EVENT_ISO_ERROR:           /* Erreur de bit */
            break ;
        case EVENT_ISO_BUSLOAD :        /* Charge bus */
            break ;
        case EVENT_TIMER :              /* Timer applicatif */
            break ;
        case EVENT_TIMEERROR :          /* Perte IT timer */
            break ;
        default  :                      /* Réservé pour utilisation future */
            break ;
}
```

7.15. IsoGetStat : Lecture des compteurs de statistiques

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

Description :
Cette fonction permet de lire les compteurs de fonctionnement du réseau.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hIsoStat : Pointeur sur la zone statistique à renseigner

Paramètres de sortie :
hIsoStat : Compteur de statistiques

WBusLoad	Charge réseau exprimée en pourcent
WReserved	Réservé pour utilisation future
DwTxRq	Nombre de demande de transmissions
DwTxOk	Nombre de fins de transmissions correctes
DwTxErr	Nombre de fins de transmissions en erreur
DwRxOk	Nombre de réceptions correctes
DwRxErr	Nombre de réceptions en erreur
DwErrBreak	Nombre d'erreurs de type Break
DwErrFE	Nombre d'erreurs de type Format
DwErrPE	Nombre d'erreurs de type Parité
DwErrOE	Nombre d'erreurs de type dépassement
DwErrFifoOvf	Nombre de messages perdus (FIFO de réception pleine)

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

7.16. IsoGetFifoRxLevel : Niveau de remplissage de la file d'attente événements

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

Description :
Cette fonction permet de lire le niveau de remplissage de la file d'attente des événements remontés à l'application.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wParam : Paramètres de la fonction
Bit 0 : Type de file d'attente (uniquement si boîtier USB)
- 0 : Le nombre d'événement remonté est celui situé dans la file d'attente coté PC
- 1 : Le nombre d'événement remonté est celui situé dans la file d'attente coté boitier USB

Paramètres de sortie :
wCount : Nombre d'événement actuellement en file d'attente
wMaxCount : Nombre d'événement maximum que peut contenir la file d'attente

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

7.17. IsoIsBusActive : Etat du bus

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

Description :
Cette fonction permet connaître si le bus à été activé à l'aide de la fonction IsoActivate.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]

Paramètres de sortie :
wState: Indique si le bus est activé ou non (valeur différente de 0)

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

8. Bibliothèque LIN

8.1. Ordre d'appel

L'application doit respecter le séquencement suivant lors de l'appel des fonctions.
X : signifie que la fonction est autorisée dans l'état en cours.
X ⇒ : signifie que la fonction est autorisée dans l'état en cours et passe à l'état suivant.

Etat	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 : Mode de fonctionnement des routines

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

Description :
Cette fonction détermine le mode d'interface entre l'application et la carte.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hLinOper : Type d'interface avec l'application

eLinOperMode	Mode de fonctionnement : LIN_OPER_ANA_FIFO - Mode analyse, stockage FIFO
wFifoSize	Réservé pour utilisation future

- Le stockage FIFO : Dans ce mode, les événements à remonter à l'application sont stockés dans une file d'attente. Ces événements sont soit des fins de transmission, soit des réceptions, soit des erreurs…Lorsque l'application appelle la fonction LinGetEvent, le premier des événements (le plus ancien dans le temps) est retiré de la file.
  Si la file d'attente est pleine et qu'un événement intervient, alors un bit indiquant une perte d'événement est placé sur le dernier événement.

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

8.3. LinConfigBus : Configuration des paramètres du bus

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

Description :
Cette fonction permet de configurer l'ensemble des paramètres du bus.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hLinBus : Paramètre du bus

eBaudrate	LIN_BAUD_2400 : - débit 2400 bauds LIN_BAUD_9600 : - débit 9600 bauds

LIN_BAUD_19200 :
- débit 19200 bauds

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

8.4. LinConfigUart : Configuration des paramètres avancés du bus

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

Description :
Cette fonction permet de configurer les paramètres avancés du bus.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hUartConfig : Paramètres avancés du bus

eParity
NO_PARITY :
- Pas de bit de parité

ODD_PARITY :
- Parité impaire

EVEN_PARITY :
- Parité paire

eStopBit
ONE_STOP_BIT :
- 1 bit de stop

TWO_STOP_BIT :
- 2 bits de stop

eDataBit
FIVE_DATA_BIT :
- 5 bits de données

SIX_DATA_BIT :
- 6 bits de données

SEVEN_DATA_BIT :
- 7 bits de données

EIGHT_DATA_BIT :
- 8 bits de données

	NINE_DATA_BIT : - 9 bits de données
dwBaudrate	Débit réel du bus en bit/s
wError	Précision du débit en ‰ (une erreur est retournée s'il est impossible d'obtenir un débit dans la tolérance indiquée)

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_NOT_SUPPORTED	Non supporté sur ce type de matériel

8.5. LinConfigParam : Configuration des paramètres avancés

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

Description :
Cette fonction permet de configurer des paramètres avancés de fonctionnement.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hLinParam : Paramètres avancés de fonctionnement

eDefDelayBusIdle	Valeur par défaut du délai de détection de timeout du bus LIN_TRUE : (par défaut) - Le délai est calculé par les librairies conformément à la spécification LIN. LIN_FALSE : - Le délai est passé en paramètre par l'utilisateur (dwDelayBusIdle)
dwDelayBusIdle	Valeur en ms de délai de détection de l'absence de communication sur le bus (événement EVENT_LIN_BUSCHANGE)
wSpecialModes	Bit 0 : Indicateur de traitement des messages en réception - 0 : La taille des messages est déterminée par

l'identificateur (LIN 1.1 et 1.2)
- 1 : L'identificateur n'est pas utilisé pour déterminer la taille du message (compatibilité pour LIN 1.3 et 2.x)

Bit 1 à 15: Réservé pour utilisation future

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

8.6. LinConfigStat : Configuration des statistiques

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

Description :
Cette fonction permet de configurer la durée sur laquelle la charge bus est calculée. Une durée égale à 0 inhibe le calcul de charge.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wBusLoadTime : Ce paramètre définit la durée sur laquelle la charge bus est calculée. Il est exprimé en ms, une valeur égale à 0 indique qu'aucune charge bus n'est remontée à l'application. La charge bus est remontée en mode FIFO par l'intermédiaire de l'événement EVENT_LIN_BUSLOAD et du paramètre wBusLoad. (0 par défaut)

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

8.7. LinSetVersion : Indication de la version du protocole

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

Description :
Cette fonction indique la version de la norme LIN (la différence entre la V1 et la V2 se situe principalement au niveau du calcul de CRC).

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
dwVersion : Version demandée au format BCD hexadécimal (ex. 0x0210 pour V2.1)

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

8.8. LinConfigTransceiver: Configuration de l'interface de ligne

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

Description :
Cette fonction permet de contrôler les différents signaux de contrôle de l'interface de ligne.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wEnable : Valeur de la broche « enable » [0-1]
wRMasterWakeUp : Valeur de la résistance de pull up
- RMaster = 0 : 30 Ko (configuration mode esclave)
- RMaster = 1 : 1 Ko (configuration mode maître)

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_WARNING	Non supporté sur ce type de matériel

8.9. LinSetNotification : Déclaration de l'événement application

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

Description :
Cette fonction permet à l'application d'utiliser les méthodes de synchronisation de tâches fournies par les systèmes d'exploitation Windows (API WIN32). Cette fonction permet de passer un handle d'événement créé par la fonction Windows « CreateEvent » aux DLL. Cet événement est utilisé en cours de communication lorsqu'un événement est remonté des DLL vers l'application (par exemple : un fin de transmission, une réception, une erreur…).
Dès lors, l'application peut se mettre en attente d'événements à l'aide des fonctions d'attente telles que WaitForSingleObject ou WaitForMultipleObject.

Exemple :
```
/* Demande d'indication d'événement */
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);
}
```

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hWinEvent : « handle » retourné par la fonction CreateEvent.

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

8.10. LinActivate : Démarrage de la communication

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

Description :
Cette fonction permet de démarrer la communication avec le réseau. Après exécution de cette fonction, les messages sont remontés en direction de l'application ou peuvent être émis par celle-ci.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

8.11. LinDeactivate : Arrêt de la communication

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

Description :
Cette fonction arrête la communication avec le réseau. Après exécution de cette fonction, les messages ne sont plus reçus, ni émis.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

8.12. LinConfigPeriodic: Programmation d'un message périodique

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

Description :
Cette fonction permet le démarrage, l'arrêt et la mise à jour de l'émission de messages périodiques.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wOffset : Indice du message périodique [0-15]
L'application dispose de 16 messages périodiques, ce paramètre correspond à l'indice du message que l'application désire accéder. Il est à noter que ces 16 messages sont traités de manière indépendante.

wParam: Paramètre du message
Bit 0 : Indicateur de fonctionnement du message périodique
- 0 : Le message périodique est arrêté
- 1 : Le message périodique est activé (démarrage de la périodique ou mise à jour)

Bit 1 : Indicateur de remonté des fins de transmissions
Dans un souci de performance, il n'est pas toujours nécessaire pour une application de recevoir les événements de fins de transmission liés à l'émission des messages périodiques.
- 0 : Les fins de transmissions ne sont pas remontées à l'application
- 1 : Les fins de transmissions sont remontées à l'application

hLinMsg : Paramètre du message à émettre

wHandleMsg	Réservé
wIdent	Identificateur LIN du message [0-0x1F] si longueur de données de 2 octets [0-0x0F] si longueur de données de 4 ou 8 octets
eService	LIN_SVC_TRANSMIT_DATA - Transmission de données LIN_SVC_REQUEST_IFR - Demande de réponse dans la trame LIN_SVC_UPDATE_IFR - Mise à jour pour réponse dans la trame
lPeriod	Périodicité du message en ms.
eLevel3Enable	Réservé pour utilisation future
dwReserved1	Réservé pour utilisation future
eLinGenErr	Possibilité de générer un message avec une erreur de protocole (les valeurs ci-dessous sont exclusives). LIN_GEN_NO_ERR - Aucune erreur effectuée LIN_GEN_ERR_P0 - Erreur du bit de parité sur le bit P0 LIN_GEN_ERR_P1 - Erreur du bit de parité sur le bit P1 LIN_GEN_ERR_CRC - Erreur de CRC (CRC inversé) LIN_GEN_ERR_SYNCH - Erreur de caractère de synchro (0xAA au lieu de 0x55) LIN_GEN_ERR_DATP1 - 1 octet à 0xFF rajouté dans le champ de données LIN_GEN_ERR_DATP2 - 2 octets à 0xFF rajoutés dans le champ de données LIN_GEN_ERR_DATL1 - 1 octet ôté du champ de données LIN_GEN_ERR_DATP1 - 2 octets ôtés du champ de données Possibilité de modifier les propriétés d'un message (les valeurs ci-dessous ne sont pas exclusives). LIN_GEN_FREE_IDENT - Identificateur indépendant de la longueur. Le choix de l'identificateur est compris entre [0 et 63], la longueur des données comprise entre [0 et 8] octets LIN_GEN_CLASSIC_CRC - Le checksum du message est calculé uniquement à partir du champ de données
dwReserved2	Réservé pour utilisation future
wDataLen	Longueur maximale des données (émises ou reçues) [0-8]
bData	Contenu des données (pour émission)

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_FIFOFULL	Le nouveau message ne peut être pris en compte. La FIFO d'émission est saturée.
STATUS_ERR_MSGEXCEED	Offset du message incorrect
STATUS_ERR_MSGSVC	Service non supporté

8.13. LinConfigPeriodicList: Programmation d'une liste de messages périodiques

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

Description :
Cette fonction permet le démarrage, l'arrêt et la mise à jour de l'émission de messages périodiques.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wPeriodicCount : Nombre de messages (limité à 16)
hLinPeriodicMsgList : Tableau de messages

wOffset	Indice du message périodique [0-15]
wParam	Paramètre du message Bit 0 : Indicateur de fonctionnement du message périodique - 0 : Le message périodique est arrêté - 1 : Le message périodique est activé (démarrage de la périodique ou mise à jour) Bit 1 : Indicateur de remonté des fins de transmissions - 0 : Les fins de transmissions ne sont pas remontées à l'application - 1 : Les fins de transmissions sont remontées à l'application
hLinMsg	Paramètre du message à émettre (cf LinConfigPeriodic)

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_FIFOFULL	Le nouveau message ne peut être pris en compte. La FIFO d'émission est saturée.
STATUS_ERR_MSGSVC	Service non supporté

8.14. LinSendMsg: Emission d'un message

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

Description :
Cette fonction permet pour un maître :
- l'émission d'un message d'écriture de donnée
- L'émission d'un message d'interrogation d'un esclave

Cette fonction permet pour un esclave :
- La mise à jour d'une réponse.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hLinMsg : Paramètre du message à émettre

wHandleMsg	Réservé
wIdent	Identificateur LIN du message [0-0x1F] si longueur de données de 2 octets [0-0x0F] si longueur de données de 4 ou 8 octets
eService	LIN_SVC_TRANSMIT_DATA - Transmission de données LIN_SVC_REQUEST_IFR - Demande de réponse dans la trame LIN_SVC_UPDATE_IFR - Mise à jour pour réponse dans la trame
lPeriod	Réservé pour utilisation future
eLevel3Enable	Réservé pour utilisation future
dwReserved1	Réservé pour utilisation future
eLinGenErr	Possibilité de générer un message avec une erreur de protocole (les valeurs ci-dessous sont exclusives). LIN_GEN_NO_ERR

- Aucune erreur effectuée

LIN_GEN_ERR_P0
- Erreur du bit de parité sur le bit P0

LIN_GEN_ERR_P1
- Erreur du bit de parité sur le bit P1

LIN_GEN_ERR_CRC
- Erreur de CRC (CRC inversé)

LIN_GEN_ERR_SYNCH
- Erreur de caractère de synchro (0xAA au lieu de 0x55)

LIN_GEN_ERR_DATP1
- 1 octet à 0xFF rajouté dans le champ de données

LIN_GEN_ERR_DATP2
- 2 octets à 0xFF rajoutés dans le champ de données

LIN_GEN_ERR_DATL1
- 1 octet ôté du champ de données

LIN_GEN_ERR_DATP1
- 2 octets ôtés du champ de données

Possibilité de modifier les propriétés d'un message (les valeurs ci-dessous ne sont pas exclusives).

LIN_GEN_FREE_IDENT
- Identificateur indépendant de la longueur. Le choix de l'identificateur est compris entre [0 et 63], la longueur des données comprise entre [0 et 8] octets

LIN_GEN_CLASSIC_CRC
- Le checksum du message est calculé uniquement à partir du champ de données

dwReserved2	Réservé pour utilisation future
wDataLen	Longueur maximale des données (émises ou reçues) [0-8]
bData	Contenu des données (pour émission)

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_FIFOFULL	Le nouveau message ne peut être pris en compte. La FIFO d'émission est saturée.
STATUS_ERR_MSGSVC	Service non supporté

8.15. LinSendMsgList: Emission d'une liste de messages

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

Description :
Cette fonction permet pour un maître :
- l'émission d'un ou plusieurs messages d'écriture de donnée
- L'émission d'un ou plusieurs messages d'interrogation d'un esclave

Cette fonction permet pour un esclave :
- La mise à jour d'une ou plusieurs réponses.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wMsgCount: Nombre de messages (limité à 32)
hLinMsgList : Tableau de messages (cf LinSendMsg pour le détail)

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_FIFOFULL	Le nouveau message ne peut être pris en compte. La FIFO d'émission est saturée.
STATUS_ERR_MSGSVC	Service non supporté

8.16. LinGetEvent: Lecture d'un événement

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

Description :
Cette fonction permet de récupérer un événement en provenance du réseau.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hLinEvent : Pointeur sur l'événement à renseigner

Paramètres de sortie :

hLinEvent : Evénement renseigné

wBus	Numéro de bus ayant généré l'événement
wHandleMsg	Réservé pour utilisation future
eTypeEvent	Type d'événement EVENT_EMPTY : Aucun événement n'est présent. EVENT_LIN_MSGTX : - Fin de transmission correcte d'un message EVENT_LIN_MSGRX : - Réception correcte d'un message EVENT_LIN_MSGTXERR : - Fin de transmission en erreur d'un message EVENT_LIN_MSGRXERR : - Réception en erreur d'un message EVENT_LIN_BUSCHANGE - Changement d'état des compteurs de communication EVENT_LIN_BUSLOAD - Charge bus EVENT_TIMER - Timer applicatif EVENT_TIMEERROR - Perte d'IT timer EVENT_FIFO_OVF - Bit indiquant qu'un ou plusieurs événements suivants celui-ci a été perdu car la FIFO de réception était pleine.
wIdent	Identificateur LIN du message [0-0x1F] si longueur de données de 2 octets [0-0x0F] si longueur de données de 4 ou 8 octets
dwTimeStamp	Heure d'arrivée de l'événement en multiple de 100 µSec (précision de 1 ms pour les cartes PCI-MUX). Ce paramètre est correct si wTimePrecision est nul. C.F. ANNEXE : horodatage des cartes PCI-MUX
wTimePrecision	Précision de la datation dwTimeStamp en multiple de 100 µSec.
eService	LIN_SVC_TRANSMIT_DATA - Transmission de données LIN_SVC_RECEIVE_DATA - Réception de données LIN_SVC_REQUEST_IFR - Demande de réponse dans la trame LIN_SVC_UPDATE_IFR - Mise à jour pour réponse dans la trame LIN_SVC_WAKEUP

- Indication signal de réveil

eError	Pour les événements de type EVENT_LIN_MSGTXERR ou EVENT_LIN_MSGRXERR, indication du type d'erreur : - LIN_ERR_BIT : Caractère reçu différent du caractère émis - LIN_ERR_CRC : Erreur de CRC - LIN_ERR_TIMEOUT : Pas de réponse de l'esclave - LIN_ERR_PARITY : Erreur de parité dans le champ identificateur - LIN_ERR_SYNCHRO : Erreur de champ synchro - LIN_ERR_LENGTH : Erreur de longueur - LIN_ERR_TO_TX : Time out en transmission - LIN_ERR_TO_SYNCHRO : Time out sur attente de réception du caractère de synchro - LIN_ERR_TO_IDENT : Time out sur attente de réception du caractère ident - LIN_ERR_TO_DATA : Time out sur attente de réception des données - LIN_ERR_TO_CRC : Time out sur attente du champ CRC
eChipState	Etat interne des compteurs de statistiques - LIN_BUS_NOMINAL : Nominal State (les compteurs d'erreurs en transmission et réception sont inférieurs à 64) - LIN_BUS_ERROR : Etat dégradé (les compteurs d'erreurs en transmission ou réception sont supérieurs à 64) - LIN_BUS_IDLE : Etat bus idle (Absence de communication détectée)
wBusLoad	Pour l'événement de type EVENT_LIN_BUSLOAD, valeur de la charge bus en %.
dwReserved1	Réservé pour utilisation future
dwReserved2	Réservé pour utilisation future
wDataLen	Longueur des données [0-8]
bData	Buffer de données

Code retour :

Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

Exemple :

```
/* Indication d'événement */
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 :          /* Plus d'événement à traiter */
                break ;
        case EVENT_LIN_MSGTX :      /* Fin de transmission correcte */
                break ;
        case EVENT_LIN_MSGRX :      /* Réception correcte */
                break ;
        case EVENT_LIN_MSGTXERR :         /* Erreur fin de transmission */
                break ;
        case EVENT_LIN_MSGRXERR :         /* Erreur reception */
                break ;
        case EVENT_LIN_BUSCHANGE :/* Changement d'état des compteurs */
                break ;
        case EVENT_LIN_BUSLOAD : /* Charge bus */
                break ;
        case EVENT_TIMER :          /* Timer applicatif */
                break ;
        case EVENT_TIMEERROR :      /* Perte IT timer */
                break ;
        default  :                        /* Réservé pour utilisation future */
                break ;
}
```

8.17. LinGetStat : Lecture des compteurs de statistiques

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

Description :
Cette fonction permet de lire les compteurs de fonctionnement du réseau.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hLinStat : Pointeur sur la zone statistique à renseigner

Paramètres de sortie :
hLinStat : Evénement à renseigner

wBusLoad	Charge réseau exprimée en pourcent

wReserved	Réservé pour utilisation future
dwTxRq	Nombre de demande de transmissions
dwTxOk	Nombre de fins de transmissions correctes
dwRxOk	Nombre de réceptions correctes
dwErrBit	Nombre d'erreurs de type Bit
dwErrCrc	Nombre d'erreurs de CRC
dwErrTimeOut	Nombre d'erreurs de timeout d'attente de réponse esclave
dwErrSynchro	Nombre d'erreurs de caractère de synchro
dwErrIdent	Nombre d'erreurs de caractère ident
dwErrOther	Nombre d'erreurs autres que celles précédemment signalées
dwErrFifoOvf	Nombre de messages perdus (FIFO de réception pleine)

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

8.18. LinGetBusState : Lecture de l'état de la communication

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

Description :
Cette fonction permet de l'état des compteurs de communication. Ces compteurs aux nombre de deux, un pour la transmission et un pour la réception, sont incrémentés de 8 en cas de messages en erreur et décrémentés de 1 en cas de message correct.
Lorsqu'un des deux compteurs atteint la valeur 64 alors l'état de la communication passe du mode nominal au mode dégradé. Inversement si les deux compteurs sont inférieurs à 64 alors la communication redevient nominale.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
eLinChipState: Pointeur sur l'état à renseigner
bTxErrCount: Pointeur sur le compteur à renseigner
bRxErrCount: Pointeur sur le compteur à renseigner

Paramètres de sortie :
eLinChipState: Etat du composant (LIN_BUS_NOMINAL, LIN_BUS_ERROR ou LIN_BUS_IDLE)
bTxErrCount: Compteur de transmission
bRxErrCount: Compteur de réception

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

8.19. LinSetSleepMode: Emission d'une trame de mise en veille

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

Description :
Cette fonction permet à un module maître de mettre en veille le réseau LIN. Cette fonction émet une trame de commande (identificateur = 0x0C) de longueur de 8 octets dont le premier octet est à 0.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

8.20. LinSetWakeUpMode: Demande de réveil du bus

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

Description :
Cette fonction permet à un module esclave de réveiller le réseau LIN.
Elle permet d'émettre le caractère 0x80 sur le bus sans aucun autre contrôle. Si après émission de ce caractère, aucun message ne circule sur le réseau avant un délai de 64*Tbit alors le caractère est réémis de nouveau. Cette opération est renouvelée jusqu'à 3 fois si aucun message n'apparaît.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

8.21. LinClearBufferIFR : Purge du buffer d'émission des IFR

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

Description :
Cette fonction permet de vider le buffer des messages IFR programmés dans la carte.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

8.22. LinGetFifoRxLevel : Niveau de remplissage de la file d'attente événements

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

Description :
Cette fonction permet de lire le niveau de remplissage de la file d'attente des événements remontés à l'application.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wParam : Paramètres de la fonction
Bit 0 : Type de file d'attente (uniquement si boîtier USB)

- 0 : Le nombre d'événement remonté est celui situé dans la file d'attente coté PC
- 1 : Le nombre d'événement remonté est celui situé dans la file d'attente coté boitier USB

Paramètres de sortie :
wCount : Nombre d'événement actuellement en file d'attente
wMaxCount : Nombre d'événement maximum que peut contenir la file d'attente

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

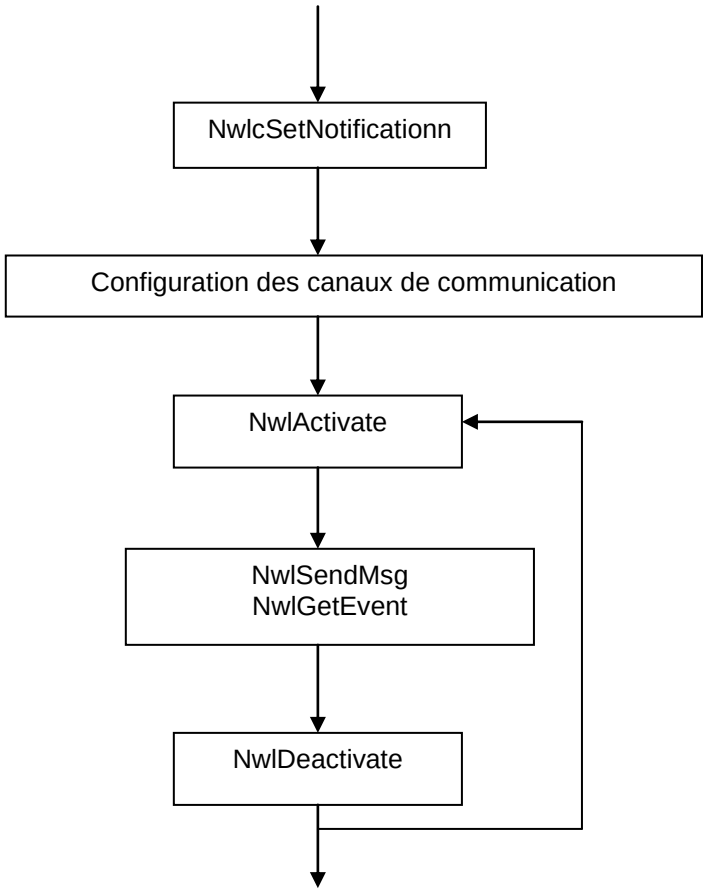
9. Bibliothèque NWL

9.1. Ordre d'appel

9.1.1 Séquence des échanges

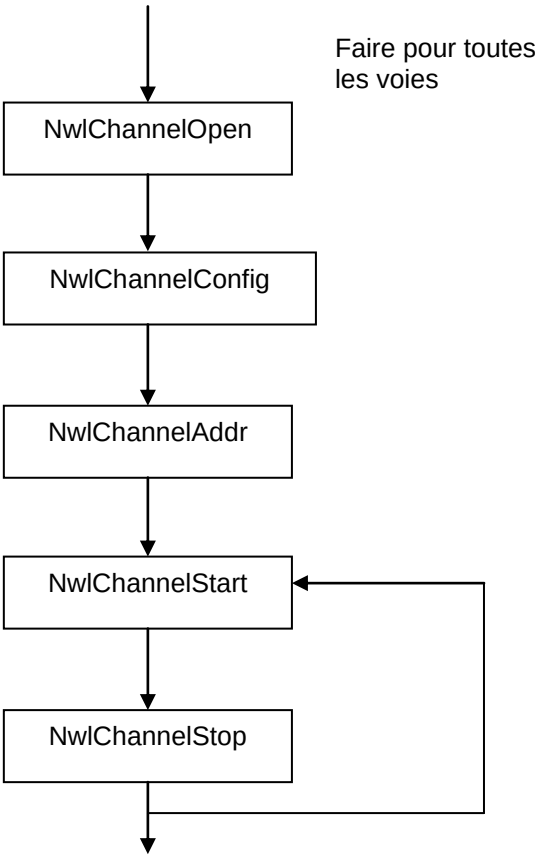
1ère étape Configurer tous les canaux de communication : L'objectif est de renseigner les différents messages qui vont être utilisées par l'application ainsi que les différents paramètres de communication (identificateur LIN physique utilisés, délai et time-out).
2ème étape Démarrer la communication à l'aide de la requête NwlActivate
3ème étape Communiquer avec les entités distantes à l'aide de deux requêtes principales

- NwlSendMsg : Emission d'une trame de longueur comprise entre 0 et 4095 octets
- NwlGetEvent : Récupérer l'ensemble des événements stocké dans une file d'attente. Ces événements signalent des informations telles que fins de transmission, réception, fins de transmissions en erreur, réception en erreur.



9.1.2 Configuration des canaux de communication

L'application doit respecter la séquence suivante pour configurer les canaux. Il est à noter que les fonctions autorisant ou non la communication sur un canal (NwlChannelStart et NwlChannelStop) peuvent être appelées en cours de communication



9.2. NwlGetChannelCount: Nombre de canaux disponibles

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

Description :
La communication entre deux entités se fait au moyen d'un canal de communication. Ce canal est défini, entre autres, par son adresse de communication (NAD) et ses paramètres temporels (CS, AS, timeout…).
Plusieurs canaux de communication peuvent être ouverts simultanément, cette fonction retourne le nombre maximum de canaux qu'il est possible d'ouvrir simultanément.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]

Paramètres de sortie :
wChannelCount : Nombre de canaux disponibles.

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

9.3. NwlChannelOpen: Ouverture d'un canal de communication

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

Description :
Cette requête permet d'ouvrir un canal de communication. Ce canal de communication permet de relier logiquement deux entités communicantes.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]

Paramètres de sortie :
wChannel : Numéro de canal ouvert.

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_MEMORY	Plus de canal disponible

9.4. NwlChannelConfig: Configuration d'un canal de communication

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

Description :
Cette requête permet de configurer un canal de communication.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wChannel : Indice du numéro de canal (obtenu par la fonction NwlChannelOpen)
hNwlChannelConfig: Paramètre du canal

eNwlService	Service de la requête NWL_SVC_TRANSMIT_DATA : Transmission de données NWL_SVC_RECEIVE_DATA : Réception de données.
dwReserved1	Réservé pour utilisation future
dwReserved2	Réservé pour utilisation future
wParam	Champ de bits définissant les paramètres de fonctionnement Evénement niveau protocole NWL_LIN_NO_EVENT : L'ensemble des échanges liés à la couche de communication n'est pas remonté à l'application NWL_LIN_EVENT : L'ensemble des échanges liés à la couche de communication est remonté à l'application Filtrage des FF (First Frame) : NWL_FF_NO_EVENT : Les événements indiquant la réception ou la fin de transmission d'une FF ne sont pas remontés à l'application NWL_FF_EVENT : Les événements indiquant la réception ou la fin de transmission d'une FF sont remontés à

	l'application

Paramètres de sortie :
Aucun.

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

9.5. NwlChannelAddr: Définition des identificateurs de communication

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

Description :
Cette requête permet de configurer les identificateurs du canal de communication.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wChannel : Indice du numéro de canal (obtenu par la fonction NwlChannelOpen)
hNwlChannelAddr : Paramètres de communication

wNad	Adresse utilisée pour identifier un nœud logique (esclave)
dwReserved1	Réservé pour utilisation future
dwReserved2	Réservé pour utilisation future

Paramètres de sortie :
Aucun.

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

9.6. NwlChannelParam : Paramètres de communication du canal

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

Description :
Cette requête permet de configurer les paramètres de communication d'un canal. Ces paramètres permettent de gérer les délais d'émission et de réception des messages segmentés. Il est possible de modifier dynamiquement ces paramètres sous réserve qu'il n'y ait aucune transmission ou réception en cours.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wChannel : Indice du numéro de canal (obtenu par la fonction NwlChannelOpen)
hNwlChannelParam: Paramètre de communication du canal

wN_Cs	Délai d'émission entre deux transmissions consécutives [0-255] ms, correspond aussi au paramètre STmin	10
wN_As	Délai max d'attente d'émission d'une trame côté émetteur [0-65535] ms	10
wN_Cr	Délai jusqu'à réception du CF [0-65535] ms	20
wP2min	Délai d'attente, coté maître, avant d'envoyer la demande de réponse [0-500] ms	50
wP2max	Délai d'attente maximum de la réponse coté maître [0-500] ms	500

Paramètres de sortie :
Aucun.

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_BUSY	Communication en cours

9.7. NwlChannelStart : Autorise la communication sur le canal

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

Description :
Cette requête permet d'autoriser la communication sur le canal.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wChannel : Indice du numéro de canal (obtenu par la fonction NwlChannelOpen)

Paramètres de sortie :
Aucun.

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

9.8. NwlChannelStop : Arrête la communication sur le canal

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

Description :
Cette requête permet d'arrêter la communication sur le canal.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wChannel : Indice du numéro de canal (obtenu par la fonction NwlChannelOpen)

Paramètres de sortie :
Aucun.

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

9.9. NwlActivate : Démarrage de la communication

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

Description :
Cette fonction permet de démarrer la communication avec le réseau. Après exécution de cette fonction, les messages peuvent être émis et reçus par l'application.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

9.10. NwlDeactivate : Arrêt de la communication

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

Description :
Cette fonction arrête la communication avec le réseau. Après exécution de cette fonction, les messages ne sont plus reçus, ni émis.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

9.11. NwlChannelSendMsg: Emission d'un message

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

Description :
Cette fonction permet l'émission d'un message de donnée.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wChannel : Indice du numéro de canal (obtenu par la fonction NwlChannelOpen)
hNwlMsg : Paramètre du message à émettre

wDataLen	Longueur maximale des données (émises ou reçues) [0-4095]
bData	Contenu des données (pour émission)

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_MSGSVC	Le canal n'est pas un canal de transmission
STATUS_ERR_MSGBUSY	Le nouveau message ne peut être pris en compte. Le message précédent est en cours d'émission.

9.12. NwlChannelReceiveMsg: Réception d'un message

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

Description :
Cette fonction permet, pour le maître, de demander (ou re-demander) au nœud esclave la transmission d'un message de donnée.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wChannel : Indice du numéro de canal (obtenu par la fonction NwlChannelOpen)

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_MSGSVC	Le canal n'est pas un canal de réception
STATUS_ERR_NWLBUSY	La demande ne peut être prise en compte. Le canal est en cours de communication.

9.13. NwlGetEvent : Lecture d'un événement

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

Description :
Cette fonction permet de récupérer un événement en provenance du réseau.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hNwlEvent : Pointeur sur l'événement à renseigner

Paramètres de sortie :
hNwlEvent : Evénement renseigné

wBus	Numéro de bus ayant généré l'événement
wChannel	Numéro de canal ayant généré l'événement [0-n].
eTypeEvent	Type d'événement EVENT_EMPTY : Aucun événement n'est présent. EVENT_NWL_MSGTX : - Fin de transmission correcte d'un message EVENT_NWL_MSGRX : - Réception correcte d'un message EVENT_NWL_MSGTXERR :

	- Fin de transmission incorrecte EVENT_NWL_MSGRXERR - Réception incorrecte EVENT_NWL_MSGRXFF - Indication de réception d'une first frame EVENT_NWL_MSGTXFF - Indication de fin de transmission d'une first frame EVENT_FIFO_OVF - Bit indiquant qu'un ou plusieurs événements suivants celui-ci a été perdu car la FIFO de réception était pleine.
wNad	Identificateur LIN du message de transmission ou en réception
dwTimeStamp	Heure d'arrivée de l'événement en multiple de 100 µSec. Ce paramètre est correct si wTimePrecision est nul.
wTimePrecision	Précision de la datation dwTimeStamp en multiple de 100 µSec.
eService	Service de la requête NWL_SVC_TRANSMIT_DATA : Transmission de données NWL_SVC_RECEIVE_DATA : Réception de données.
eError	Pour les événements de type EVENT_NWL_MSGTXERR ou EVENT_NWL_MSGRXERR, indication du type d'erreur : - NWL_NO_ERR : Aucune erreur - NWL_ERR_TOUT_AS : Délai dépassé pour l'émission d'un message coté émetteur (paramètre wN_As) - NWL_ERR_TOUT_CR : Délai dépassé pour l'attente d'une trame CF « Consecutive Frame » (paramètre wN_Cr) - NWL_ERR_TOUT_P2 : Délai dépassé pour la demande ou l'attente d'une réponse (paramètres wP2min et wP2max) - NWL_ERR_SN : Réception d'un numéro de séquence incorrecte. La réception est interrompue - NWL_ERR_FIFO_OVF : La file d'attente d'émission des messages LIN est pleine, la transmission ou réception en cours est interrompue. - NWL_ERR_MEMORY : Plus d'espace mémoire disponible pour traiter le message. - NWL_ERR_INV_PDU : Format de PDU reçu incorrecte
dwReserved1	Réservé pour utilisation future
dwReserved2	Réservé pour utilisation future
wDataLen	Longueur des données [0-4095]
bData	Buffer de données

Code retour :

Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

Exemple :

```
/* Indication d'événement */
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 :              /* Plus d'événement à traiter */
                break ;
        case EVENT_NWL_MSGTX :          /* Fin de transmission correcte */
                break ;
        case EVENT_NWL_MSGRX :          /* Réception correcte */
                break ;
        case EVENT_NWL_MSGTXERR :       /* Fin de transmission en erreur */
                break ;
        case EVENT_NWL_MSGRXERR :       /* Réception en erreur*/
                break ;
        case EVENT_NWL_MSGRXFF :        /* Réception d'une FF "First Frame" */
                break ;
        case EVENT_NWL_MSGTXFF :        /* Fin de transmission d'une FF */
                break ;
        default  :                      /* Réservé pour utilisation future */
                break ;
}
```

9.14. NwlGetFifoRxLevel : Niveau de remplissage de la file d'attente événements

Prototype:

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

Description :

Cette fonction permet de lire le niveau de remplissage de la file d'attente des événements remontés à l'application.

Paramètres d'entrée :

wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wParam : Paramètres de la fonction
Bit 0 : Type de file d'attente (uniquement si boîtier USB)

- 0 : Le nombre d'événement remonté est celui situé dans la file d'attente coté PC

- 1 : Le nombre d'événement remonté est celui situé dans la file d'attente coté embarqué

Paramètres de sortie :
wCount : Nombre d'événement actuellement en file d'attente
wMaxCount : Nombre d'événement maximum que peut contenir la file d'attente

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

9.15. NwlGetChannelState : Etat courant du canal

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

Description :
Cette fonction permet d'obtenir l'état courant du canal de communication.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wChannel : Indice du numéro de canal (obtenu par la fonction NwlChannelOpen)

Paramètres de sortie :
wChannelState : Etat courant du canal de communication
Bit NWL_CHANNEL_STATE_STARTED
0 : Le canal n'a pas été activé
1 : Le canal est activé (prêt à émettre ou recevoir)
Bit NWL_CHANNEL_STATE_BUSY
0 : Le canal est activé et au repos
1 : Le canal est activé et communique, une transmission ou une réception est en cours

eLastTxStatus : Status du dernier fin de transmission

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres

STATUS_ERR_SEQUENCE	Erreur de séquence

9.16. NwlIsBusActive : Etat du bus

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

Description :
Cette fonction permet connaître si le bus à été activé à l'aide de la fonction NwlActivate.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]

Paramètres de sortie :
wState: Indique si le bus est activé ou non (valeur différent de 0)

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

9.17. NwlChannelClose : Fermeture d'un canal de communication

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

Description :
Cette fonction permet de fermer un canal de communication.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wChannel : Numéro du canal à fermer

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK

STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

9.18. NwlSetNotification : Déclaration de l'événement application

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

Description :
Cette fonction permet à l'application d'utiliser les méthodes de synchronisation de tâches fournies par les systèmes d'exploitation Windows (API WIN32). Cette fonction permet de passer un handle d'événement créé par la fonction Windows « CreateEvent » aux DLL. Cet événement est utilisé en cours de communication lorsqu'un événement est remonté des DLL vers l'application (par exemple : un fin de transmission, une réception, une erreur…).
Dès lors, l'application peut se mettre en attente d'événements à l'aide des fonctions d'attente telles que WaitForSingleObject ou WaitForMultipleObject.

Exemple :
```
/* Demande d'indication d'événement */
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);
}
```

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hWinEvent : « handle » retourné par la fonction CreateEvent.

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

10. Bibliothèque VAN

10.1. Ordre d'appel

L'application doit respecter le séquencement suivant lors de l'appel des fonctions.
X : signifie que la fonction est autorisée dans l'état en cours.
X ⇒ : signifie que la fonction est autorisée dans l'état en cours et passe à l'état suivant.

Etat	VAN_INIT	VAN_OPER	VAN_BUS	VAN_START
VanConfigOper	X ⇒			
VanConfigBus		X ⇒		
VanActivate			X ⇒	
VanDeactivate	X	X	X	X ⇒
VanSetNotification	X	X	X	
VanConfigParam		X		
VanConfigStat	X	X	X	X
VanConfigPeriodic		X	X	X
VanCreateMsg			X	
VanSetTIP				X
VanSendMsg				X
VanGetEvent				X
VanGetBusState				X
VanChangeMsgId				X
VanGetStat				X
VanGetCount	X	X	X	X
VanSetSleepMode	X	X	X	X
VanSetWakeUpMode	X	X	X	X
VanGetBusState	X	X	X	X
VanGetSDCValue	X	X	X	X
VanReadByte	X	X	X	X
VanWriteByte	X	X	X	X
VanGetFifoRxLevel	X	X	X	X

10.2. VanConfigOper : Mode de fonctionnement des routines

Prototype:
tMuxStatus VanConfigOper(unsigned int wCard, unsigned int wBus, tVanOper *hVanOper);

Description :
Cette fonction détermine le mode d'interface entre l'application et la carte.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hVanOper : Type d'interface avec l'application

eVanOperMode	Mode de fonctionnement : VAN_OPER_SIMU_FIFO : - Mode simulation, stockage FIFO VAN_OPER_SIMU_BUFF - Mode simulation, stockage BUFFER VAN_OPER_ANA_FIFO - Mode analyse, stockage FIFO VAN_OPER_ANA_BUFF - Mode analyse, stockage Buffer
wFifoSize	Réservé pour utilisation future

- Le mode simulation : Ce mode est utilisé dans le cas d'applications désirant simuler un ou plusieurs organes présents sur le bus. Dans ce mode, les routines d'interfaces utilisent pour chaque message déclaré un canal du contrôleur VAN ce qui permet de gérer les réponses dans la trame ou l'acquittement de manière sélective par rapport à un identificateur.

- Le mode analyse : Ce mode est utilisé dans le cas d'applications désirant visualiser les messages en transit su le réseau VAN. Dans ce mode, les routines d'interfaces utilisent uniquement 2 canaux pour les réceptions. Ceci permet lors de la réception d'un message, de pouvoir recevoir sur le second canal pendant que le premier message est traité.

- Le stockage FIFO : Dans ce mode, les événements à remonter à l'application sont stockés dans une file d'attente. Ces événements sont soit des fins de transmission, soit des réceptions, soit des erreurs…Lorsque l'application appelle la fonction VanGetEvent, le premier des événements (le plus ancien dans le temps) est retiré de la file.
  Si la file d'attente est pleine et qu'un événement intervient, alors un bit indiquant une perte d'événement est placé sur le dernier événement.

- Le stockage BUFFER : Dans ce mode, les événements à remonter à l'application sont dans un buffer unique alloué lors de la configuration du message (quelque soit le service de celui-ci). Ces événements sont soit des fins de transmission, soit des réceptions (pas les erreurs). Lorsque l'application appelle la fonction VanGetEvent, le handle de communication message permet de repérer le buffer à lire. Ce buffer contient le dernier événement reçu (le plus récent dans le temps).
  Lors de chaque lecture, le buffer est réinitialisé. Si entre deux lectures aucun événement n'est survenu, le compte rendu (EVENT_EMPTY) l'indique à l'application.

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

10.3. VanConfigBus : Configuration des paramètres du bus

Prototype:
tMuxStatus VanConfigBus(unsigned short wCard, unsigned short wBus, tVanBus *hVanBus);

Description :
Cette fonction permet de configurer l'ensemble des paramètres du bus.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hVanBus : Paramètre du bus

eBaudRate	Débit du bus en kilo time slot / seconde
eLineInvertTx	VAN_LINE_NOT_INVERTED (par défaut) - Le signal de transmission du contrôleur de protocole VAN n'est pas inversé VAN_LINE_INVERTED - Le signal de transmission est inversé
eLineInvertRx	VAN_LINE_NOT_INVERTED (par défaut) - Le signal de transmission du contrôleur de protocole VAN n'est pas inversé VAN_LINE_INVERTED - Le signal de transmission est inversé
eLineCoding	VAN_CODING_MANCHESTER (par défaut) - Transmission et réception des données codées en mode Manchester comprimé VAN_CODING_PULSED (par défaut) - Transmission et réception des données codées en mode pulsé
wMaxRetries	Nombre maximum de répétition [0-15]
eModuleType	VAN_MODULE_SYNCHRONOUS - Module synchrone : toute émission n'est transmise sur le réseau qu'après apparition d'un SOF. Cette configuration est notamment pour les équipements

esclaves du réseau carrosserie.

VAN_MODULE_AUTONOMOUS
- Module autonome : les émissions sont transmises immédiatement. Cette configuration est notamment utilisée pour tous les équipements maîtres.

eTIPEnable — VAN_FALSE (par défaut)
- Le diagnostic en émission n'est pas validé.

VAN_TRUE
- Le diagnostic en émission est validé

Ce paramètre est utilisé dans les procédures de répétitions des couches de communication (Voir la fonction VanSetTIP)

eSDCEnable — VAN_FALSE
- Le diagnostic avec horloge SDC n'est pas validé

VAN_TRUE (par défaut)
- Le diagnostic avec horloge SDC est validé (paramètre wSDCValue valide)

Ce paramètre représente le délai entre l'instant ou un défaut sur une des lignes data ou datab disparaît et l'instant ou le contrôleur reprend une communication en mode nominal.

wSDCValue — Ce paramètre est valide si le paramètre eSDCEnable est valide. Il dépend du débit réseau (voir la fonction VanGetSDCValue)
Pour un réseau à 62.5 kTS/s horloge de 100ms : 7
Pour un réseau à 125 kTS/s horloge de 1000ms : 11

eRxMode — VAN_RX_FORCED_RXD0
- Communication forcée sur RXD0

VAN_RX_FORCED_RXD1
- Communication forcée sur RXD1

VAN_RX_FORCED_RXD2
- Communication forcée sur RXD2

VAN_RX_AUTOMATIC
- Sélection automatique

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

10.4. VanConfigParam : Configuration des paramètres supplémentaires

Prototype:
tMuxStatus VanConfigParam(unsigned int wCard, unsigned int wBus, tVanParam *hVanParam);

Description :
Cette fonction permet de configurer des paramètres supplémentaires de fonctionnement. En effet, la différence fondamentale entre le mode analyse et le mode simulation intervient sur la répartition des canaux par rapport aux messages configurés :

- En mode simulation : l'allocation des canaux et de la mémoire du contrôleur a lieu après chaque configuration de message.
- En mode analyse, l'allocation des canaux a été optimisée pour permettre la capture de tous les messages en transit sur le réseau.
  - 2 canaux ont été réservés pour la réception de données sur le composant 0 (1 pour recevoir pendant que l'on traite l'autre).
  - 1 canal a été réservé pour la transmission de données sur le composant 0, les émissions provenance de l'application sont sérialisées.
  - 2 canaux ont été réservés pour la réception de données différées sur le composant 1 (1 pour recevoir pendant que l'on traite l'autre).
  - 1 canal a été réservé pour la demande d'IFR sur le composant 1, les émissions en provenance de l'application sont sérialisées.

  Le reste de la mémoire et des canaux du composant 1 est réservé aux IFR.

La fonction VanConfigBus permet de jouer sur les valeurs de ces paramètres pour optimiser le nombre d'IFR disponibles.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hVanParam : Paramètres auxiliaires du mode analyse

wSizeTx	Taille maximale des transmissions et demande d'IFR (28 par défaut)
wSizeRx	Taille maximale des réceptions de données et réceptions différées (28 par défaut)
wSizeIfr	Taille maximale des IFR (28 par défaut)
wFiltIdent	Identificateur du filtre d'acceptance (0x000 par défaut) Le filtre d'acceptance est apposé sur les buffers de réception de données et réceptions différées (voir note)
wFiltMask	Masque du filtre d'acceptance (0x000 par défaut)
eAckEnable	VAN_FALSE (par défaut) - La génération de l'acquittement est inhibé (mode espion) VAN_TRUE - La génération de l'acquittement est validée, les messages reçus correctement avec demande

d'acquittement sont acquittés

eRxAll	VAN_TRUE (par défaut) - Tous les messages reçus en plus de ceux programmés sont remontés à l'application par l'intermédiaire de la FIFO de réception VAN_FALSE - Seuls les messages reçus programmés en réception sont remontés à l'application par l'intermédiaire de la FIFO de réception
wSpecialModes	Réservé pour utilisation future

Note filtre d'acceptance : Le filtre d'acceptance permet d'appliquer un filtre de sélection des messages reçus en provenance de réseau pour limiter la charge sur le PC. Ce filtre est valide pour des applications de mode BUFFER et est prioritaire sur les autres filtres.
Un identificateur est reçu si la condition suivante est vraie :

(Identificateur reçu ET Masque) = (Identificateur programmé ET Masque)

Par exemple :

4. Ident programmé = xxx, masque = 000 permet de recevoir tous les identificateurs
5. Ident programmé = xxx, masque = FFF permet de recevoir uniquement l'identificateur xxx
6. Ident programmé = 001, masque = 001 permet de recevoir tous les identificateurs impairs

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

10.5. VanConfigStat : Configuration des statistiques

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

Description :

Cette fonction permet de configurer la durée sur laquelle la charge bus est calculée. Une durée égale à 0 inhibe le calcul de charge.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wBusLoadTime : Ce paramètre définit la durée sur laquelle la charge bus est calculée. Il est exprimé en ms, une valeur égale à 0 indique qu'aucune charge bus n'est remontée à l'application. La charge bus est remontée en mode FIFO par l'intermédiaire de l'événement EVENT_VAN_BUSLOAD et du paramètre wBusLoad. (0 par défaut)

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

10.6. VanConfigPeriodic: Programmation d'un message périodique

Prototype:
tMuxStatus VanConfigPeriodic(unsigned short wCard, unsigned short wBus, unsigned short wOffset, unsigned short wParam, tVanMsg *hVanMsg);

Description :
Cette fonction permet le démarrage, l'arrêt et la mise à jour de l'émission de messages périodiques.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wOffset : Indice du message périodique [0-15]
L'application dispose de 16 messages périodiques, ce paramètre correspond à l'indice du message que l'application désire accéder. Il est à noter que ces 16 messages sont traités de manière indépendante.

wOffset : Paramètre du message
Bit 0 : Indicateur de fonctionnement du message périodique
- 0 : Le message périodique est arrêté
- 1 : Le message périodique est activé (démarrage de la périodique ou mise à jour)

Bit 1 : Indicateur de remonté des fins de transmissions

Dans un souci de performance, il n'est pas toujours nécessaire pour une application de recevoir les événements de fins de transmission liés à l'émission des messages périodiques.

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

hVanMsg : Paramètres du message

wHandleMsg	Voir paramètre de sortie
wIdent	Identificateur VAN du message [0-0xFFF]
wMask	Masque de réception associé à l'identificateur du message [0-0xFFF]. Le masque est utilisé pour les services de type (VAN_SVC_RECEIVE_DATA, VAN_SVC_UPDATE_IFR et VAN_SVC_RX_DIFFERED) (C.F. note 1)
eService	VAN_SVC_TRANSMIT_DATA - Transmission de données VAN_SVC_RECEIVE_DATA - Réception de données VAN_SVC_REQUEST_IFR - Demande de réponse dans la trame VAN_SVC_UPDATE_IFR - Mise à jour de réponse dans la trame VAN_SVC_TX_DIFFERED - Transmission de réponse différée VAN_SVC_RX_DIFFERED - Réception de réponse différée (la réponse différée peut être interprétée comme une demande de réponse dans la trame avec réponse dans la trame). (C.F. note 2)
eAckEnable	VAN_TRUE - Demande d'acquittement VAN_FALSE - Pas de demande d'acquittement
lPeriod	Périodicité du message en ms.
eLevel3Enable	VAN_TRUE - Gestion des couches de communication VAN_FALSE - Pas de gestion des couches de communication
	Renseigner à VAN_FALSE (Réservé pour utilisation future)
wNumChip	Voir paramètre de sortie
wNumChannel	Voir paramètre de sortie
wMaxSize	Réservé pour usage interne
dwReserved1	Renseigner à 0 (Réservé pour utilisation future)
dwReserved2	Renseigner à 0 (Réservé pour utilisation future)
wDataLen	Longueur maximale des données (émises ou reçues) [0-28]

bData	Contenu des données (pour émission)

10.7. VanCreateMsg : Configuration des messages de communication

Prototype:
tMuxStatus VanCreateMsg(unsigned int wCard, unsigned int wBus, tVanMsg *hVanMsg);

Description :
Cette fonction permet de déclarer les messages gérés par l'application.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hVanMsg : Paramètres du message

wHandleMsg	Voir paramètre de sortie
wIdent	Identificateur VAN du message [0-0xFFF]
wMask	Masque de réception associé à l'identificateur du message [0-0xFFF]. Le masque est utilisé pour les services de type (VAN_SVC_RECEIVE_DATA, VAN_SVC_UPDATE_IFR et VAN_SVC_RX_DIFFERED) (C.F. note 1)
eService	VAN_SVC_TRANSMIT_DATA - Transmission de données VAN_SVC_RECEIVE_DATA - Réception de données VAN_SVC_REQUEST_IFR - Demande de réponse dans la trame VAN_SVC_UPDATE_IFR - Mise à jour de réponse dans la trame VAN_SVC_TX_DIFFERED - Transmission de réponse différée VAN_SVC_RX_DIFFERED - Réception de réponse différée (la réponse différée peut être interprétée comme une demande de réponse dans la trame avec réponse dans la trame). (C.F. note 2)
eAckEnable	VAN_TRUE - Demande d'acquittement VAN_FALSE - Pas de demande d'acquittement
lPeriod	Renseigner à 0 (Réservé pour utilisation future)
eLevel3Enable	VAN_TRUE - Gestion des couches de communication

	VAN_FALSE - Pas de gestion des couches de communication
	Renseigner à VAN_FALSE (Réservé pour utilisation future)
wNumChip	Voir paramètre de sortie
wNumChannel	Voir paramètre de sortie
wMaxSize	Réservé pour usage interne
dwReserved1	Renseigner à 0 (Réservé pour utilisation future)
dwReserved2	Renseigner à 0 (Réservé pour utilisation future)
wDataLen	Longueur maximale des données (émises ou reçues) [0-28]
bData	Contenu des données (pour émission)

Paramètres de sortie :

wHandleMsg	Index de communication (valide uniquement en mode BUFFER). Cet index est retourné par les DLL à l'application pour indiquer le numéro de buffer alloué. Cet indice est utilisé par la suite pour récupérer les événements réseau par l'intermédiaire de la fonction VanGetEvent ou pour émettre des messages par l'intermédiaire de la fonction VanSendMessage.
wNumChip	Indice du composant (0 ou 1) ayant en charge le message
wNumChannel	Numéro de canal dans le composant

Note 1 :
Le masque permet de recevoir une famille d'identificateurs (voir exemple de la fonction VanConfigParam). Néanmoins certaines précautions doivent être prises :

- Dans le mode analyse, le masque défini par la fonction VanConfigParam est prioritaire par rapport à ce masque.
- Quelque soit le mode, il est important qu'il n'y ait aucune intersection entre deux familles. Dans ce cas, l'identifcateur serait reçu par le message ayant le numéro de canal le plus faible (wNumChannel).

Note 2 :
Dans le mode analyse FIFO, il n'est pas nécessaire de déclarer les messages initiateurs (service : VAN_SVC_TRANSMIT_DATA, VAN_SVC_REQUEST_IFR, VAN_SVC_UPDATE_IFR et VAN_SVC_TX_DIFFERED).

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_MSGEXCEED	Erreur trop de messages déclarés
STATUS_ERR_TSSOVER	Erreur dépassement mémoire ou canaux

du TSS

10.8. VanSetNotification : Déclaration de l'événement application

Prototype:
tMuxStatus VanSetNotification (unsigned int wCard, unsigned int wBus, HANDLE hWinEvent);

Description :
Cette fonction permet à l'application d'utiliser les méthodes de synchronisation de tâches fournies par les systèmes d'exploitation Windows (API WIN32). Cette fonction permet de passer un handle d'événement créé par la fonction Windows « CreateEvent » aux DLL. Cet événement est utilisé en cours de communication lorsqu'un événement est remonté des DLL vers l'application (par exemple : un fin de transmission, une réception, une erreur…).
Dès lors, l'application peut se mettre en attente d'événements à l'aide des fonctions d'attente telles que WaitForSingleObject ou WaitForMultipleObject.

Exemple :
```
/* Demande d'indication d'événement */
HANDLE hWinEvent ;
tVanEvent hVanEvent ;
tMuxStatus Status ;
unsigned short wCard=0,wBus=0 ;

hWinEvent=CreateEvent(NULL,FALSE,FALSE,NULL) ;
if (hWinEvent == NULL) return ;
Status=VanSetNotification (wCard,wBus,hWinEvent) ;
if (Status != STATUS_OK) return ;
Status=VanActivate(wCard,wBus) ;
if (Status != STATUS_OK) return ;
if (WaitForSingleObject(hWinEvent,INFINITE) == WAIT_OBJECT_0)
{
        VanGetEvent(wCard,wBus,&hVanEvent);
}
```

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hWinEvent : « handle » retourné par la fonction CreateEvent.

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

10.9. VanActivate : Démarrage de la communication

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

Description :
Cette fonction permet de démarrer la communication avec le réseau. Après exécution de cette fonction, les messages reçus sont acquittés et remontés en direction de l'application. L'application peut également émettre des messages.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

10.10. VanDeactivate : Arrêt de la communication

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

Description :
Cette fonction arrête la communication avec le réseau. Après exécution de cette fonction, les messages ne sont plus reçus, ni acquittés, ni émis.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]

Paramètres de sortie :

Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

10.11. VanSendMsg: Emission d'un message

Prototype :
tMuxStatus VanSendMsg(unsigned int wCard, unsigned int wBus, tVanMsg *hVanMsg);

Description :
Cette fonction permet l'émission d'un message sur le bus ou la mise à jour de données pour des réponses dans la trame.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hVanMsg : Paramètre du message à émettre

wHandleMsg	Index de communication Pour les modes BUFFER et simulation FIFO : - Cet index a été précédemment retourné par la fonction VanCreateMsg lors de la création du message. Par rapport à la configuration, seules les données (bData) et leurs longueurs (wDataLen) sont prises en compte. Pour le mode analyse FIFO : - Les messages sont sérialisés par la DLL, ce paramètre n'est pas utilisé.
wIdent	Identificateur VAN du message [0-0xFFF]
wMask	Masque de réception associé à l'identificateur du message [0-0xFFF]. Le masque est utilisé par le service VAN_SVC_UPDATE_IFR
eService	VAN_SVC_TRANSMIT_DATA - Transmission de données VAN_SVC_REQUEST_IFR - Demande de réponse dans la trame VAN_SVC_UPDATE_IFR - Mise à jour de réponse dans la trame

VAN_SVC_TX_DIFFERED
- Transmission de réponse différée

eAckEnable	VAN_TRUE - Demande d'acquittement VAN_FALSE - Pas de demande d'acquittement
wNumChip	Inutilisé
wNumChannel	Inutilisé
dwReserved1	Réservé pour utilisation future
dwReserved2	Réservé pour utilisation future
wDataLen	Longueur des données [0-longueur maximale configurée]
bData	Contenu des données

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_FIFOFULL	Le nouveau message ne peut être pris en compte. La FIFO d'émission est saturée.
STATUS_ERR_TSSOVER	Le nouveau message ne peut être pris en compte. Plus de canaux ou de mémoire disponible dans le contrôleur VAN.
STATUS_ERR_MSGSVC	Service non supporté
STATUS_ERR_HANDLE	Le paramètre wHandleMsg est invalide
STATUS_ERR_MSGBUSY	Le nouveau message ne peut être pris en compte. Le message précédent est en cours d'émission.

10.12. VanGetEvent: Lecture d'un événement

Prototype:
tMuxStatus VanGetEvent(unsigned int wCard, unsigned int wBus, tVanEvent *hVanEvent);

Description :
Cette fonction permet de récupérer un événement en provenance du réseau.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]

hVanEvent : Pointeur sur l'événement à renseigner

Paramètres de sortie :
hVanEvent : Evénement renseigné

wBus	Numéro de bus ayant généré l'événement
wHandleMsg	A titre indicatif : Numéro de canal interne au DLL (C.F. VanCreateMsg)
eTypeEvent	Type d'événement EVENT_EMPTY : Aucun événement n'est présent. EVENT_VAN_MSGTX: - Fin de transmission correcte d'un message EVENT_VAN_MSGRX : - Réception correcte d'un message EVENT_VAN_MSGTXERR : - Fin de transmission d'un message en erreur EVENT_VAN_MSGRXERR : - Réception d'un message en erreur EVENT_VAN_BUSCHANGE - Changement de la ligne de réception détectée EVENT_VAN_BUSLOAD - Charge bus EVENT_TIMER - Timer applicatif EVENT_TIMEERROR - Perte d'IT timer EVENT_FIFO_OVF - Bit indiquant qu'un ou plusieurs événements suivants celui-ci a été perdu car la FIFO de réception était pleine.
wIdent	Identificateur du message reçu [0-0xFFF]
dwTimeStamp	Heure d'arrivée de l'événement en multiple de 100 µSec (précision de 1 ms pour les cartes PCI-MUX). Ce paramètre est correct si wTimePrecision est nul. C.F. ANNEXE : horodatage des cartes PCI-MUX
wTimePrecision	Précision de la datation dwTimeStamp en multiple de 100 µSec.
eService	Pour les événements liés à un message, service du message (voir fonction VanCreateMsg).
eAckEnable	Pour les événements liés à un message, indication si un acquittement est présent. VAN_TRUE - Avec acquittement VAN_FALSE

- Pas de présence d'acquittement

eError	Pour les événements de type EVENT_VAN_MSGTXERR ou EVENT_VAN_MSGRXERR, indication du type d'erreur : - VAN_ERR_BOC : Buffer occupé - VAN_ERR_BOV : Dépassement du buffer - VAN_ERR_TYPE : Erreur du service du message - VAN_ERR_FCSE : Erreur de CRC - VAN_ERR_ACKE : Erreur d'acquittement - VAN_ERR_CV : Viol de code - VAN_ERR_FV : Viol de trame - VAN_ERR_NOIFR : Plus de place pour IFR
eChipState	Etat de la communication des contrôleurs de protocole VAN - VAN_RX_NOMINAL : Communication différentiel - VAN_RX_FAULT_ON_DATA : Communication sur datab - VAN_RX_FAULT_ON_DATAB : Communication sur data - VAN_RX_MAJOR_ERROR : Time out de communication
dwReserved1	Réservé pour utilisation future
dwReserved2	Réservé pour utilisation future
wBusLoad	Pour l'événement de type EVENT_VAN_BUSLOAD, valeur de la charge bus en %.
wDataLen	Longueur des données [0-28]
bData	Buffer de données

Code retour :

Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

Exemple :

```
/* Indication d'événements */
tVanEvent hVanEvent ;
tMuxStatus Status ;
unsigned short wCard=0,wBus=0 ;

Status=VanGetEvent(wCard,wBus,&hVanEvent);
if (Status != STATUS_OK) return ;
switch(hVanEvent.eTypeEvent &~ EVENT_FIFO_OVF)
{
        case EVENT_EMPTY :          /* Plus d'événements à traiter */
```

```
            break ;
        case EVENT_VAN_MSGTX :   /* Fin de transmission correcte */
            break ;
        case EVENT_VAN_MSGRX :   /* Réception correcte */
            break ;
        case EVENT_VAN_MSGTXERR :       /* Fin de transmission avec erreur*/
            break ;
        case EVENT_VAN_BUSCHANGE :/* Changement de ligne de Rx */
            break ;
        case EVENT_VAN_BUSLOAD :        /* Charge bus */
            break ;
        case EVENT_TIMER :  /* Timer applicatif */
            break ;
        case EVENT_TIMEERROR :    /* Perte d'IT timer */
            break ;
        default :                       /* Réservé pour utilisation future */
            break ;
}
```

10.13. VanGetStat : Lecture des compteurs de statistiques

Prototype:
tMuxStatus VanGetCount(unsigned int wCard, unsigned int wBus, tVanStat *hVanStat);

Description :
Cette fonction permet de lire les compteurs de fonctionnement du réseau.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hVanStat : Pointeur sur la zone statistique à renseigner

Paramètres de sortie :
hVanStat : Evénement à renseigner

wBusLoad	Charge réseau exprimée en pourcent
wReserved	Réservé pour utilisation future
dwTxRq	Nombre de demande de transmissions
dwTxOk	Nombre de fins de transmissions correctes
dwTxErr	Nombre de fins de transmissions en erreur
dwRxOk	Réception correcte
dwRxErr	Réception en erreur
dwErrBoc	Nombre d'erreurs buffer occupé
dwErrBov	Nombre d'erreurs de dépassement de buffer
dwErrType	Nombre d'erreurs de service VAN

dwErrFcse	Nombre d'erreurs de CRC
dwErrAcke	Nombre d'erreurs d'acquittement
dwErrCv	Nombre d'erreurs de viol de code
dwErrFv	Nombre d'erreurs de viol de trame
dwErrFifoOvf	Nombre de messages perdus (FIFO de réception pleine)

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

10.14. VanGetBusState : Lecture du mode de communication des composants

Prototype:
tMuxStatus VanGetBusState(unsigned int wCard, unsigned int wBus, tVanChipState *eVanChip0State, tVanChipState *eVanChip1State);

Description :
Cette fonction permet de lire directement le mode de réception des contrôleurs VAN.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
eVanChip0State: Pointeur sur état à renseigner
eVanChip1State: Pointeur sur état à renseigner

Paramètres de sortie :
eVanChip0State: Etat du composant n°0
eVanChip1State: Etat du composant n°1
Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

10.15. VanSetSleepMode: Veille / réveil du bus (station maître)

Prototype:

tMuxStatus VanSetSleepMode(unsigned int wCard, unsigned int wBus, tVanSleepMode eVanSleepMode);

Description :
Cette fonction permet à un module maître de mettre en veille ou réveiller le réseau VAN. Cette fonction agit directement sur la broche « Sleepb » de l'interface de ligne VAN.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
eVanBusSleep: Commande du bus

VAN_SLEEP_MODE	Mise en veille du réseau
VAN_NO_SLEEP_MODE	Réveil du réseau (par défaut)

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

10.16. VanSetWakeUpMode: Réveil du bus (station esclave)

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

Description :
Cette fonction permet à un module esclave de réveiller le réseau VAN. Cette fonction agit directement sur la broche « Wake up » de l'interface de ligne VAN.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
eVanWakeUpMode: Commande du bus

VAN_WAKEUP_MODE	Réveil du réseau
VAN_NO_WAKEUP_MODE	Pas d'action (par défaut)

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

10.17. VanGetSleepMode: Etat du bus veille / réveil

Prototype:
tMuxStatus VanGetSleepMode(unsigned short wCard, unsigned short wBus, tVanSleepMode *eVanSleepMode);

Description :
Cette fonction permet lire l'état de la broche « SREG » de l'interface de ligne VAN. La broche SREG est une sortie ayant pour fonction de commander l'alimentation des modules esclaves sur détection d'un réveil du réseau

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
eVanSleepMode: Pointeur sur l'état à renseigner

Paramètres de sortie :
eVanSleepMode: Etat du veille / réveil

VAN_SLEEP_MODE	Réseau en veille
VAN_NO_SLEEP_MODe	Réseau en mode nominal

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

10.18. VanSetTIP : Gestion du diagnostic en émission

Prototype:
tMuxStatus VanSetTIP(unsigned short wCard, unsigned short wBus, tVanBoolean eTIPEnable);

Description :

Cette fonction permet de gérer le diagnostic en émission. Ce diagnostic permet de détecter lors d'une émission la présence d'un circuit ouvert sur une des lignes data ou datab.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
eTIPEnable: Commande de diagnostic

VAN_FALSE	(par défaut). Pas de commande du diagnostic en emission
VAN_TRUE	Diagnostic en émission validé

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

10.19. VanGetSDCValue : Retourne la valeur de l'horloge SDC

Prototype:
tMuxStatus VanGetSDCValue(tVanBaudRate eBaudRate,unsigned long dwSDCTimems,unsigned long *dwSDCValue);

Description :
Cette fonction permet de retourner la valeur de l'horloge SDC en fonction de la durée de l'horloge désirée et du débit du bus. L'horloge SDC est un délai entre la fin de détection d'un défaut détecté sur une des lignes data ou datab et le retour en mode nominal.

Paramètres d'entrée :
eBaudRate : Débit du bus
dwSDCTimems : Durée de l'horloge (exprimée en milliseconde)
dwSDCValue : Pointeur sur la valeur à renseigner

Paramètres de sortie :
dwSDCValue : Valeur de l'horloge SDC. Cette valeur est à renseigner dans les paramètres du bus (fonction VanConfigBus).

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres

10.20. VanReadByte: Lecture directe du contrôleur de protocole VAN

Prototype:
tMuxStatus VanReadByte(unsigned int wCard, unsigned int wBus, unsigned int wChip, unsigned int wOffset,unsigned char *bData);

Description :
Cette fonction permet de lire directement les registres du contrôleur de protocole VAN.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wChip : Indice du contrôleur de protocole [0-1]
wOffset : Offset de lecture
bData : Pointeur sur la valeur à lire

Paramètres de sortie :
bData : Valeur lue

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

10.21. VanWriteByte : Ecriture directe dans le contrôleur de protocole VAN

Prototype:
tMuxStatus VanWriteByte (unsigned int wCard, unsigned int wBus, unsigned int wChip, unsigned int wOffset,unsigned char bData);

Description :
Cette fonction permet de d'écrire directement dans les registres du contrôleur de protocole VAN.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wChip : Indice du contrôleur de protocole [0-1]

wOffset : Offset d'écriture
bData : Valeur à écrire

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

10.22. VanGetFifoRxLevel : Niveau de remplissage de la file d'attente événements

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

Description :
Cette fonction permet de lire le niveau de remplissage de la file d'attente des événements remontés à l'application.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wParam : Paramètres de la fonction
Bit 0 : Type de file d'attente (uniquement si boîtier USB)

- 0 : Le nombre d'événement remonté est celui situé dans la file d'attente coté PC
- 1 : Le nombre d'événement remonté est celui situé dans la file d'attente coté boitier USB

Paramètres de sortie :
wCount : Nombre d'événement actuellement en file d'attente
wMaxCount : Nombre d'événement maximum que peut contenir la file d'attente

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

11. Bibliothèque NMEA0183

11.1. Ordre d'appel

L'application doit respecter le séquencement suivant lors de l'appel des fonctions.
X : signifie que la fonction est autorisée dans l'état en cours.
X ⇒ : signifie que la fonction est autorisée dans l'état en cours et passe à l'état suivant.

Etat	NME_INIT	NME_OPER	NME_BUS	NME_START
NMEA0183ConfigOper	X ⇒			
NMEA0183ConfigBus		X ⇒		
NMEA0183Activate			X ⇒	
NMEA0183Deactivate	X	X	X	X ⇒
NMEA0183SetNotification	X	X	X	
NMEA0183ConfigParam		X		
NMEA0183ConfigStat	X	X	X	X
NMEA0183ConfigPeriodic		X	X	X
NMEA0183SendMsg				X
NMEA0183GetEvent				X
NMEA0183GetBusState				X
NMEA0183GetStat				X
NMEA0183GetFifoRxLevel	X	X	X	X

11.2. NMEA0183ConfigOper : Mode de fonctionnement des routines

Prototype:
tMuxStatus NMEA0183ConfigOper(unsigned short wCard, unsigned short wBus, tNMEA0183Oper *hNMEA0183Oper);

Description :
Cette fonction détermine le mode d'interface entre l'application et la carte.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hNMEA0183Oper : Type d'interface avec l'application

eNMEA0183OperMode Mode de fonctionnement :
NMEA0183_OPER_ANA_FIFO
- Mode analyse, stockage FIFO

wFifoSize	Réservé pour utilisation future

- Le stockage FIFO : Dans ce mode, les événements à remonter à l'application sont stockés dans une file d'attente. Ces événements sont soit des fins de transmission, soit des réceptions, soit des erreurs…Lorsque l'application appelle la fonction NMEA0183GetEvent, le premier des événements (le plus ancien dans le temps) est retiré de la file.
  Si la file d'attente est pleine et qu'un événement intervient, alors un bit indiquant une perte d'événement est placé sur le dernier événement.

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

11.3. NMEA0183ConfigBus : Configuration des paramètres du bus

Prototype:
tMuxStatus NMEA0183ConfigBus(unsigned short wCard, unsigned short wBus, tNMEA0183Bus *hNMEA0183Bus);

Description :
Cette fonction permet de configurer l'ensemble des paramètres du bus.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hNMEA0183Bus : Paramètre du bus

eBaudrate	NMEA0183_BAUD_4800	/* Débit 4800 bauds */

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres

STATUS_ERR_SEQUENCE	Erreur de séquence

11.4. NMEA0183ConfigParam : Configuration des paramètres avancés

Prototype:
tMuxStatus NMEA0183ConfigParam(unsigned short wCard, unsigned short wBus, tNMEA0183Param *hNMEA0183Param);

Description :
Cette fonction permet de configurer des paramètres avancés de fonctionnement.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hNMEA0183Param : Paramètres avancés de fonctionnement

dwDelayBusIdle	Valeur en ms du délai de détection de l'absence de communication sur le bus (événement EVENT_ NMEA0183_BUSCHANGE)
dwDelayMsgTimeout	Valeur en ms du délai inter-caractère d'un message en réception. Si le délai inter-caractère d'un message en cours de réception est supérieur au délai programmé, alors le message sera remonté avec le code erreur NMEA0183_ERR_TO_MSG. Note : un délai à 0 signifie qu'il n'y a pas de time-out programmé.
wControlCrc	Bit 0 : - 0 : Les messages avec CRC correct ou sans CRC (pas de caractère *) reçus sont remontés correctement à l'application - 1 : Seuls les messages avec CRC corrects sont remontés avec succès à l'application
wSpecialModes	Réservé pour utilisation future

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

11.5. NMEA0183ConfigStat : Configuration des statistiques

Prototype:
tMuxStatus NMEA0183ConfigStat(unsigned short wCard, unsigned short wBus, unsigned short wBusLoadTime);

Description :
Cette fonction permet de configurer la durée sur laquelle la charge bus est calculée. Une durée égale à 0 inhibe le calcul de charge.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wBusLoadTime : Ce paramètre définit la durée sur laquelle la charge bus est calculée. Il est exprimé en ms, une valeur égale à 0 indique qu'aucune charge bus n'est remontée à l'application. La charge bus est remontée en mode FIFO par l'intermédiaire de l'événement EVENT_NMEA0183_BUSLOAD et du paramètre wBusLoad. (0 par défaut)

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

11.6. NMEA0183SetNotification : Déclaration de l'événement application

Prototype:
tMuxStatus NMEA0183SetNotification (unsigned short wCard, unsigned short wBus, HANDLE hWinEvent);

Description :
Cette fonction permet à l'application d'utiliser les méthodes de synchronisation de tâches fournies par les systèmes d'exploitation Windows (API WIN32). Cette fonction permet de passer un handle d'événement créé par la fonction Windows « CreateEvent » aux DLL. Cet

événement est utilisé en cours de communication lorsqu'un événement est remonté des DLL vers l'application (par exemple : un fin de transmission, une réception, une erreur…).
Dès lors, l'application peut se mettre en attente d'événements à l'aide des fontions d'attente telles que WaitForSingleObject ou WaitForMultipleObject.

Exemple :

```
/* Demande d'indication d'événement */
HANDLE hWinEvent ;
tNMEA0183Event hNMEA0183Event ;
tMuxStatus Status ;
unsigned short wCard=0,wBus=0 ;

hWinEvent=CreateEvent(NULL,FALSE,FALSE,NULL) ;
if (hWinEvent == NULL) return ;
Status=NMEA0183SetNotification (wCard,wBus,hWinEvent) ;
if (Status != STATUS_OK) return ;
Status=NMEA0183Activate(wCard,wBus) ;
if (Status != STATUS_OK) return ;
if (WaitForSingleObject(hWinEvent,INFINITE) == WAIT_OBJECT_0)
{
        NMEA0183GetEvent(wCard,wBus,&hNMEA0183Event);
}
```

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hWinEvent : « handle » retourné par la fonction CreateEvent.

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

11.7. NMEA0183Activate : Démarrage de la communication

Prototype:

```
tMuxStatus NMEA0183Activate(unsigned int wCard, unsigned int wBus);
```

Description :

Cette fonction permet de démarrer la communication avec le réseau. Après exécution de cette fonction, les messages sont remontés en direction de l'application ou peuvent être émis par celle-ci.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

11.8. NMEA0183Deactivate : Arrêt de la communication

Prototype:
tMuxStatus NMEA0183Deactivate(unsigned int wCard, unsigned int wBus);

Description :
Cette fonction arrête la communication avec le réseau. Après exécution de cette fonction, les messages ne sont plus reçus, ni émis.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

11.9. NMEA0183ConfigPeriodic: Programmation d'un message périodique

Prototype:

tMuxStatus NMEA0183ConfigPeriodic(unsigned short wCard, unsigned short wBus, unsigned short wOffset, unsigned short wParam, tNMEA0183Msg *hNMEA0183Msg);

Description :
Cette fonction permet le démarrage, l'arrêt et la mise à jour de l'émission de messages périodiques.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wOffset : Indice du message périodique [0-15]
L'application dispose de 16 messages périodiques, ce paramètre correspond à l'indice du message que l'application désire accéder. Il est à noter que ces 16 messages sont traités de manière indépendante.

wOffset : Paramètre du message
Bit 0 : Indicateur de fonctionnement du message périodique
- 0 : Le message périodique est arrêté
- 1 : Le message périodique est activé (démarrage de la périodique ou mise à jour)

Bit 1 : Indicateur de remonté des fins de transmissions
Dans un souci de performance, il n'est pas toujours nécessaire pour une application de recevoir les événements de fins de transmission liés à l'émission des messages périodiques.
- 0 : Les fins de transmissions ne sont pas remontées à l'application
- 1 : Les fins de transmissions sont remontées à l'application

hNMEA0183Msg : Paramètre du message à émettre

wHandleMsg	Réservé
eNMEA0183GenErr	NMEA0183_GEN_RAW - Transmission intégrale de ce que fourni l'application NMEA0183_GEN_NO_CRC - Transmission du buffer fournie par l'application avec ajout du caractère d'entête $ et des caractères de fin CR rt LF NMEA0183_GEN_CRC - Transmission du buffer fournie par l'application avec ajout du caractère d'entête $, du CRC et du CRC délimiteur (*) ainsi que des caractères de fin CR rt LF.
lPeriod	Périodicité du message en ms.
dwReserved1	Réservé pour utilisation future
dwReserved2	Réservé pour utilisation future
wDataLen	Longueur maximale des données (émises ou reçues) [0-77] 77 pour un message de type NMEA0183_GEN_RAW

	74 pour un message de type NMEA0183_GEN_NO_CRC 71 pour un message de type NMEA0183_GEN_CRC
bData	Contenu des données (pour émission)

11.10. NMEA0183SendMsg: Emission d'un message

Prototype:
tMuxStatus NMEA0183SendMsg(unsigned short wCard, unsigned short wBus, t NMEA0183Msg *h NMEA0183Msg);

Description :
Cette fonction permet d'émettre un message sur la liaison NMEA0183

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hNMEA0183Msg : Paramètre du message à émettre

wHandleMsg	Réservé
eNMEA0183GenErr	NMEA0183_GEN_RAW - Transmission intégrale de ce que fourni l'application NMEA0183_GEN_NO_CRC - Transmission du buffer fournie par l'application avec ajout du caractère d'entête $ et des caractères de fin CR rt LF NMEA0183_GEN_CRC - Transmission du buffer fournie par l'application avec ajout du caractère d'entête $, du CRC et du CRC délimiteur (*) ainsi que des caractères de fin CR rt LF.
lPeriod	Périodicité du message en ms.
dwReserved1	Réservé pour utilisation future
dwReserved2	Réservé pour utilisation future
wDataLen	Longueur maximale des données (émises ou reçues) [0-77] 77 pour un message de type NMEA0183_GEN_RAW 74 pour un message de type NMEA0183_GEN_NO_CRC 71 pour un message de type NMEA0183_GEN_CRC
bData	Contenu des données (pour émission)

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence
STATUS_ERR_FIFOFULL	Le nouveau message ne peut être pris en compte. La FIFO d'émission est saturée.

11.11. NMEA0183GetEvent: Lecture d'un événement

Prototype:
tMuxStatus NMEA0183GetEvent(unsigned short wCard, unsigned short wBus, tNMEA0183Event *hNMEA0183Event);

Description :
Cette fonction permet de récupérer un événement en provenance du réseau.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
hNMEA0183Event : Pointeur sur l'événement à renseigner

Paramètres de sortie :
hNMEA0183Event : Evénement renseigné

wBus	Numéro de bus ayant généré l'événement
wHandleMsg	Réservé pour utilisation future
eNMEA0183Event	Type d'événement EVENT_EMPTY : Aucun événement n'est présent. EVENT_ NMEA0183_MSGTX : - Fin de transmission correcte d'un message EVENT_ NMEA0183_MSGRX : - Réception correcte d'un message EVENT_ NMEA0183_MSGTXERR : - Fin de transmission en erreur d'un message EVENT_ NMEA0183_MSGRXERR : - Réception en erreur d'un message EVENT_ NMEA0183_BUSCHANGE - Changement d'état des compteurs de communication EVENT_ NMEA0183_BUSLOAD - Charge bus

EVENT_TIMER
- Timer applicatif

EVENT_TIMEERROR
- Perte d'IT timer

EVENT_FIFO_OVF
- Bit indiquant qu'un ou plusieurs événements suivants celui-ci a été perdu car la FIFO de réception était pleine.

dwTimeStamp	Heure d'arrivée de l'événement en multiple de 100 µSec (précision de 1 ms pour les cartes PCI-MUX). Ce paramètre est correct si wTimePrecision est nul. C.F. ANNEXE : horodatage des cartes PCI-MUX
eError	Pour les événements de type EVENT_NMEA0183_MSGTXERR ou EVENT_NMEA0183_MSGRXERR, indication du type d'erreur : - NMEA0183_ERR_RX_OVER: Longueur du message trop grand - NMEA0183_ERR_INV_PDU: Format du message incorrect - NMEA0183_ERR_BAD_CRC : Erreur de CRC - NMEA0183_ERR_BAD_EOS : Erreur de séquence sur les caractères de fins (CR et LF) - NMEA0183_ERR_TO_MSG : Timeout inter-caractère du message reçu
eChipState	Etat interne des compteurs de statistiques - NMEA0183_BUS_NOMINAL : Etat nominal (les compteurs d'erreurs en transmission et reception sont inférieurs à 64) - NMEA0183_BUS_ERROR : Etat dégradé (les compteurs d'erreurs en transmission ou réception sont supérieurs à 64) - NMEA0183_BUS_IDLE : Etat bus idle (Absence de communication détectée)
wBusLoad	Pour l'événement de type EVENT_ NMEA0183_BUSLOAD, valeur de la charge bus en %.
dwReserved1	Réservé pour utilisation future
dwReserved2	Réservé pour utilisation future
wDataLen	Longueur des données [0-8]
bData	Buffer de données

Code retour :

Status : Compte rendu d'exécution de la fonction

STATUS_OK

STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

Exemple :

```
/* Indication d'événement */
tNMEA0183Event hNMEA0183Event ;
tMuxStatus Status ;
unsigned short wCard=0,wBus=0 ;

Status= NMEA0183GetEvent(wCard,wBus,&h NMEA0183Event);
if (Status != STATUS_OK) return ;
switch(h NMEA0183Event.eTypeEvent &~ EVENT_FIFO_OVF)
{
        case EVENT_EMPTY :                  /* Plus d'événement à traiter */
                break ;
        case EVENT_NMEA0183_MSGTX :         /* Fin de transmission correcte */
                break ;
        case EVENT_NMEA0183_MSGRX :         /* Réception correcte */
                break ;
        case EVENT_NMEA0183_MSGTXERR :      /* Erreur fin de transmission */
                break ;
        case EVENT_NMEA0183_MSGRXERR :      /* Erreur reception */
                break ;
        case EVENT_NMEA0183_BUSCHANGE :     /* Changement d'état des compteurs */
                break ;
        case EVENT_NMEA0183_BUSLOAD :       /* Charge bus */
                break ;
        case EVENT_TIMER :                  /* Timer applicatif */
                break ;
        case EVENT_TIMEERROR :              /* Perte IT timer */
                break ;
        default  :                          /* Réservé pour utilisation future */
                break ;
}
```

11.12. NMEA0183GetStat : Lecture des compteurs de statistiques

Prototype:

```
tMuxStatus NMEA0183GetStat(unsigned short wCard, unsigned short wBus, t
NMEA0183Stat *hNMEA0183Stat);
```

Description :

Cette fonction permet de lire les compteurs de fonctionnement du réseau.

Paramètres d'entrée :

wCard : Indice du numéro de carte à accéder

wBus : Indice du numéro de bus [0-x]
hNMEA0183Stat : Pointeur sur la zone statistique à renseigner

Paramètres de sortie :
hNMEA0183Stat : Evénement à renseigner

wBusLoad	Charge réseau exprimée en pourcent
wReserved	Réservé pour utilisation future
dwTxRq	Nombre de demande de transmissions
dwTxOk	Nombre de fins de transmissions correctes
dwRxOk	Nombre de réceptions correctes
dwErrBreak	Nombre d'erreurs de type Break
dwErrFE	Nombre d'erreurs de type Framing error
DwErrPE	Nombre d'erreurs de type Parity error
DwErrOE	Nombre d'erreurs de type Overrun error
dwErrFifoOvf	Nombre de messages perdus (FIFO de réception pleine)

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

11.13. NMEA0183GetBusState : Lecture de l'état de la communication

Prototype:
tMuxStatus NMEA0183GetBusState(unsigned short wCard, unsigned short wBus, tNMEA0183ChipState *eNMEA0183ChipState, unsigned char *bTxErrCount, unsigned char *bRxErrCount);

Description :
Cette fonction permet de l'état des compteurs de communication. Ces compteurs aux nombre de deux, un pour la transmission et un pour la réception, sont incrémentés de 8 en cas de messages en erreur et décrémentés de 1 en cas de message correct.
Lorsqu'un des deux compteurs atteint la valeur 64 alors l'état de la communication passe du mode nominal au mode dégradé. Inversement si les deux compteurs sont inférieurs à 64 alors la communication redevient nominale.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
eNMEA0183ChipState: Pointeur sur l'état à renseigner
bTxErrCount: Pointeur sur le compteur à renseigner
bRxErrCount: Pointeur sur le compteur à renseigner

Paramètres de sortie :
eNMEA0183ChipState: Etat du composant (NMEA0183_BUS_NOMINAL, NMEA0183_BUS_ERROR ou NMEA0183_BUS_IDLE)
bTxErrCount: Compteur de transmission
bRxErrCount: Compteur de réception

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

11.14. NMEA0183GetFifoRxLevel : Niveau de remplissage de la file d'attente événements

Prototype:
tMuxStatus NMEA0183GetFifoRxLevel (unsigned short wCard, unsigned short wBus, unsigned short wParam, unsigned short *wCount, unsigned short *wMaxCount);

Description :
Cette fonction permet de lire le niveau de remplissage de la file d'attente des événements remontés à l'application.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wBus : Indice du numéro de bus [0-x]
wParam : Paramètres de la fonction
Bit 0 : Type de file d'attente (uniquement si boîtier USB)
- 0 : Le nombre d'événement remonté est celui situé dans la file d'attente coté PC
- 1 : Le nombre d'événement remonté est celui situé dans la file d'attente coté boitier USB

Paramètres de sortie :
wCount : Nombre d'événement actuellement en file d'attente
wMaxCount : Nombre d'événement maximum que peut contenir la file d'attente

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

12. Bibliothèque Entrées/Sorties

12.1. IOSetOutput: Activation des sorties logiques

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

Description :
Cette fonction permet d'activer les sorties logiques présentes sur la carte.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wOutputValue : Valeur des sorties
wOutputMask : Masque indiquant les sorties à positionner. Un bit à 1 indique que la valeur de la sortie est à prendre en compte, un bit à 0 indique que la valeur de la sortie ne doit pas être modifiée.

Exemple:
Mise à 1 de la sortie 0 : IsoSetOutput(wCard, 0x01, 0x01);
Mise à 0 de la sortie 0 : IsoSetOutput(wCard, 0x00, 0x01);
Mise à 1 des sorties 2 et 3 : IsoSetOutput(wCard, 0x0C, 0x0C);
Mise à 0 de la sortie 3 : IsoSetOutput(wCard, 0x00, 0x08);

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

Signification des sorties

Bit	Signification
0	Sortie n°0 ou ligne RTS1
1	Sortie n°1 ou ligne RTS2
2	Sortie n°2
3	Sortie n°3
4	Sortie n°4 ou commande directe de la ligne K du réseau n°1

5	Sortie n°5 ou commande directe de la ligne L du réseau n°1
6	Sortie n°6 ou commande directe de la ligne K du réseau n°2
7	Sortie n°7 ou commande directe de la ligne L du réseau n°2
8	Sortie n°8
9	Sortie n°9
10	Sortie n°10
11	Sortie n°11
12	Sortie n°12
13	Sortie n°13
14	Sortie n°14
15	Sortie n°15

12.2. IOGetInput: Lecture des entrées logiques

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

Description :
Cette fonction permet lire l'état des entrées logiques présentes sur la carte.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wInputValue : Pointeur sur la valeur à renseigner

Paramètres de sortie :
wInputValeur : Etat des entrées

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

Signification des entrées

Bit	Signification
0	Entrée n°0 ou ligne CTS1 ou présence VBAT
1	Entrée n°1 ou ligne CTS2 ou état présence +APC
2	Entrée n°2
3	Entrée n°3 ou état présence +APC sur PCI-XXXX
4	Entrée n°4 ou état présence +VBAT sur PCI-XXXX
5	Entrée n°5 ou état de la ligne K du réseau n°1
6	Entrée n°6 ou état de la ligne K du réseau n°2

7	Entrée n°7
8	Entrée n°8
9	Entrée n°9
10	Entrée n°10
11	Entrée n°11
12	Entrée n°12
13	Entrée n°13
14	Entrée n°14
15	Entrée n°15

13. Bibliothèque Timer

Pour faciliter l'écriture des programmes utilisateurs, il est possible d'utiliser les librairies pour fournir une base de temps à l'application basée sur un timer de 1 milliseconde.

13.1. TimerSet : Activation / désactivation d'une base de temps en milliseconde

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

Description :
Cette fonction permet de démarrer ou d'arrêter une base de temps exprimée en millisecondes.

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
wTimerValue : Valeur de la base de temps exprimée en millisecondes. Une valeur à 0 arrête la base de temps

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

13.2. TimerSetNotification : Déclaration de l'événement application

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

Description :
Cette fonction permet à l'application d'utiliser les méthodes de synchronisation de tâches fournies par les systèmes d'exploitation Windows (API WIN32). Cette fonction permet de

passer un handle d'événement créé par la fonction Windows « CreateEvent » aux DLL. Cet événement est utilisé pour remonter l'événement timer à l'application.
Dès lors, l'application peut se mettre en attente d'événements à l'aide des fonctions d'attente telles que WaitForSingleObject ou WaitForMultipleObject.

Exemple :

```
/* Demande de timer 1 milliseconde */
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(« « ) ;
        */
}
```

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder
hWinEvent : « handle » retourné par la fonction CreateEvent.

Paramètres de sortie :
Aucun

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres
STATUS_ERR_SEQUENCE	Erreur de séquence

13.3. TimerRead: Lecture de l'horloge courante

Prototype :

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

Description :

Cette fonction permet la valeur courante de l'horloge interne gérée par les DLLs. Cette horloge exprimée en multiple de 1 ms est mise en route lors de l'ouverture du driver par la fonction MuxOpen.
Cette fonction a bien entendu pour rôle de donner une base de temps à l'application synchronisée par rapport aux événements réseaux mais peut être utilisée aussi pour récupérer des événements timer qui n'aurait pas été traités par l'application. En effet, une application Windows (ou « thread ») est dans le cas de système d'exploitation multitâche susceptible d'être suspendu pendant un temps inconnu. Pour compenser ce gîte, il est alors nécessaire lors de la reprise de la tâche d'effectuer une boucle pour rattraper le temps perdu lors de la suspension de la tâche.

Exemple :

```
/* Demande de timer 1 milliseconde */
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(« « ) ;
        */
}
```

Paramètres d'entrée :
wCard : Indice du numéro de carte à accéder

Paramètres de sortie :
dwTimerValue : Compteur 1 ms incrémenté depuis la fonction MuxOpen
wTimeError : Compteur de perte d'interruption horloge (C.F. ANNEXE : horodatage des cartes PCI-MUX)
wMaxTimePrecision : Erreur maximale de précision d'horloge

Code retour :
Status : Compte rendu d'exécution de la fonction

STATUS_OK	
STATUS_ERR_PARAM	Erreur de paramètres

STATUS_ERR_SEQUENCE	Erreur de séquence

14. Insertion des librairies au projet

Il existe deux méthodes pour inclure les DLLs au projet :

- La méthode dites par chargement statique
- La méthode dites par chargement dynamique

14.1. Méthode par chargement statique

Cette méthode consiste à appeler directement les fonctions de la librairie en référence externe, comme n'importe quelles fonctions standard, puis au moment de l'édition de lien d'insérer le fichier *.LIB

Procédure à suivre

1 – Insérer dans chaque fichier source faisant appel à au moins une fonction des librairies le fichier de définition des prototypes REFMUX.H

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

2 – Ajouter au projet le fichier MUXDLL.LIB (pour un accès direct aux cartes) ou AEBRIDGE.DLL (pour un accès partagé utilisant le serveur d'accès).

Remarques : En VISUAL C++ et BORLAND C++, cette opération se réalise par le menu « Projet » puis « ajouter au projet »

Avantages et inconvénients

Avantage :

1. Méthode très simple à mettre en œuvre

Inconvénients :

1. Il se peut qu'en fonction de certains compilateurs utilisés, le fichier .LIB ne soit pas compatible. Dans ce cas une erreur de format est produite à l'utilisateur et seule la méthode de chargement dynamique est possible.
2. Une fois l'application générée, l'application est dépendante de la version du fichier MUXDLL.DLL. L'exécutable de l'application et le fichier *.DLL doivent toujours aller de paire.

14.2. Méthode par chargement dynamique

Cette méthode consiste à appeler indirectement les fonctions de la librairie. Cela se réalise par un premier appel à la fonction **MuxLoadDLL** qui a pour rôle d'initialiser des pointeurs de fonctions vers les librairies. Par la suite, l'appel aux fonctions a lieu comme si celles-ci étaient directement liées au projet

Procédure à suivre

1 - Insérer dans un fichier source le fichier de définition des prototypes REFMUX.H avec au préalable le mot clef MUX_DYNAMIC_DLL indiquant le chargement dynamique des fonctions

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

2 – Appeler la fonction MuxLoadDLL avant tout autre appel de fonction

3 – Insérer dans les autres fichiers faisant éventuellement appel aux librairies le fichier de définition REFMUX.H avec le mot clef MUX_DYNAMIC_DLL (chargement dynamique) et MUX_DYNAMIC_DLL_EXTERN pour ne pas re-déclarer les pointeurs de fonctions.

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

4 – Appeler la fonction MuxFreeDLL en fin de programme pour libérer les pointeurs de fonctions

Avantages et inconvénients

Avantages :

1. L'application accède aux fonctions par l'intermédiaire de requêtes windows (API WIN32), l'application peut se compiler sans problème de format de fichier avec la librairie.
2. L'application n'est plus dépendante du fichier MUXDLL.DLL, une mise à jour uniquement des DLLs peut être envisagée sans re-générer l'application.

Inconvénients :

1. Méthode légèrement plus compliquée à mettre en œuvre que la méthode par chargement statique

Annexe 1: Prototypes communs

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);

Annexe 2 : Prototypes CAN

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);

Annexe 3 : Prototypes NWC

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);

Annexe 4 : Prototypes J1939

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 **J19StoreNMEAFastPacket**(unsigned short wCard, unsigned short wBus, unsigned long dwIdent, unsigned short wSize);

Annexe 5 : Prototypes ISO

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);

Annexe 6 : Prototypes LIN

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);

Annexe 7 : Prototypes NWL

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);

Annexe 8 : Prototypes VAN

tMuxStatus _MUXAPI **VanConfigOper**(unsigned short wCard, unsigned short wBus, tVanOper *hVanOper);
tMuxStatus _MUXAPI **VanConfigBus**(unsigned short wCard, unsigned short wBus, tVanBus *hVanBus);
tMuxStatus _MUXAPI **VanConfigParam**(unsigned short wCard, unsigned short wBus, tVanParam *hVanParam);
tMuxStatus _MUXAPI **VanConfigStat**(unsigned short wCard, unsigned short wBus, unsigned short wBusLoadTime);
tMuxStatus _MUXAPI **VanCreateMsg**(unsigned short wCard, unsigned short wBus, tVanMsg *hVanMsg);
tMuxStatus _MUXAPI **VanConfigPeriodic**(unsigned short wCard, unsigned short wBus, unsigned short wOffset, unsigned short wParam, tVanMsg *hVanMsgNew);
tMuxStatus _MUXAPI **VanSetNotification** (unsigned short wCard, unsigned short wBus, HANDLE hWinEvent);
tMuxStatus _MUXAPI **VanActivate**(unsigned short wCard, unsigned short wBus);
tMuxStatus _MUXAPI **VanDeactivate**(unsigned short wCard, unsigned short wBus);
tMuxStatus _MUXAPI **VanSendMsg**(unsigned short wCard, unsigned short wBus, tVanMsg *hVanMsg);
tMuxStatus _MUXAPI **VanGetEvent**(unsigned short wCard, unsigned short wBus, tVanEvent *hVanEvent);
tMuxStatus _MUXAPI **VanGetFifoRxLevel** (unsigned short wCard, unsigned short wBus, unsigned short wParam, unsigned short *wCount, unsigned short *wMaxCount);
tMuxStatus _MUXAPI **VanGetStat**(unsigned short wCard, unsigned short wBus, tVanStat *hVanStat);
tMuxStatus _MUXAPI **VanGetBusState**(unsigned short wCard, unsigned short wBus, tVanChipState *eVanChip0State, tVanChipState *eVanChip1State);
tMuxStatus _MUXAPI **VanSetSleepMode**(unsigned short wCard, unsigned short wBus, tVanSleepMode eVanSleepMode);
tMuxStatus _MUXAPI **VanSetWakeUpMode**(unsigned short wCard, unsigned short wBus, tVanWakeUpMode eVanWakeUpMode);
tMuxStatus _MUXAPI **VanGetSleepMode**(unsigned short wCard, unsigned short wBus, tVanSleepMode *eVanSleepMode);
tMuxStatus _MUXAPI **VanSetTIP**(unsigned short wCard, unsigned short wBus, tVanBoolean eTIPEnable);
tMuxStatus _MUXAPI **VanGetSDCValue**(tVanBaudRate eBaudRate,unsigned long dwSDCTimeMs,unsigned long *dwSDCValue);
tMuxStatus _MUXAPI **VanReadByte**(unsigned short wCard, unsigned short wBus, unsigned short wChip, unsigned short wOffset,unsigned char *bData);
tMuxStatus _MUXAPI **VanWriteByte**(unsigned short wCard, unsigned short wBus, unsigned short wChip, unsigned short wOffset,unsigned char bData);

Annexe 9 : Prototypes NMEA0183

tMuxStatus _MUXAPI **NMEA0183ConfigOper** (unsigned short wCard, unsigned short wBus, tNMEA0183Oper *hNMEA0183Oper);
tMuxStatus _MUXAPI **NMEA0183ConfigBus** (unsigned short wCard, unsigned short wBus, tNMEA0183Bus *hNMEA0183Bus);
tMuxStatus _MUXAPI **NMEA0183ConfigParam** (unsigned short wCard, unsigned short wBus, tNMEA0183Param *hNMEA0183Param);
tMuxStatus _MUXAPI **NMEA0183ConfigStat** (unsigned short wCard, unsigned short wBus, unsigned short wBusLoadTime);
tMuxStatus _MUXAPI **NMEA0183ConfigPeriodic**(unsigned short wCard, unsigned short wBus, unsigned short wOffset, unsigned short wParam, tNMEA0183Msg *hNMEA0183MsgNew);
tMuxStatus _MUXAPI **NMEA0183SetNotification** (unsigned short wCard, unsigned short wBus, HANDLE hWinEvent);
tMuxStatus _MUXAPI **NMEA0183Activate**(unsigned short wCard, unsigned short wBus);
tMuxStatus _MUXAPI **NMEA0183Deactivate**(unsigned short wCard, unsigned short wBus);
tMuxStatus _MUXAPI **NMEA0183SendMsg** (unsigned short wCard, unsigned short wBus, tNMEA0183Msg *hNMEA0183Msg);
tMuxStatus _MUXAPI **NMEA0183GetEvent** (unsigned short wCard, unsigned short wBus, tNMEA0183Event *hNMEA0183Event);
tMuxStatus _MUXAPI **NMEA0183GetStat** (unsigned short wCard, unsigned short wBus, tNMEA0183Stat *hNMEA0183Stat);
tMuxStatus _MUXAPI **NMEA0183GetBusState** (unsigned short wCard, unsigned short wBus, tNMEA0183ChipState *eNMEA0183ChipState,
unsigned char *bTxErrCount, unsigned char *bRxErrCount);
tMuxStatus _MUXAPI **NMEA0183GetFifoRxLevel** (unsigned short wCard, unsigned short wBus, unsigned short wParam, unsigned short *wCount,
unsigned short *wMaxCount);

Annexe 10 : Prototypes gestion des E/S et timer

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);

Annexe 11 : Horodatage des cartes PCI-MUX

Les cartes PCI-MUX ne sont pas équipées de microprocesseurs dont le rôle pourrait être de dater les événements en transit sur le réseau. Il est donc à la charge des fonctions d'accès d'effectuer cette opération.
Dans la plupart des cas, la datation est correcte. Néanmoins, dans le cas de certaines configurations du PC (économiseur d'écran validé, anti-virus…), il se peut que cette datation soit différée par le PC.
Ne pouvant remédier à ce problème, il a été prévu un dispositif pour signaler à l'utilisateur que celui-ci était survenu et que la datation des événements était imprécise. Pour cela, les cartes PCI-MUX sont équipées d'un timer cadencé à 1 ms et d'une logique de contrôle de traitement par le PC.

Lorsque la datation est correcte, le PC gère immédiatement les interruptions en provenance des contrôleurs de protocole de la carte et du timer. Le paramètre « wTimerError » de la structure événement est égal à 0.
Lorsque le PC ne gère pas immédiatement les interruptions, la datation de l'événement vis à vis de son instant réel est décalée. Si une interruption timer intervient alors qu'une autre est déjà présente mais non traitée, alors cette information est mémorisée sur la carte et est traitée lorsque le PC reprend la gestion des interruptions pour incrémenter « wTimeError ».
En résumé, « wTimeError » s'incrémente de 1 lorsqu'une erreur de datation de 1 ou plusieurs millisecondes a été constatée.

Préconisations
Pour diminuer le risque d'erreur de datation, il est conseillé :

- D'inhiber les économiseurs d'écrans et de disques durs
- De dévalider les logiciels d'anti-virus
- De manière générale, de supprimer toute application fonctionnant en parallèle avec l'application réseau
- …

Annexe 12 : Suivi des versions de la librairie MUXDLL

Version	Date	Evolutions / Modifications
6.1.6	28/10/2011	• Arrêt du support pour les pilotes USB Exxotest v1.x.x et Jungo v5.2.2 • Support pour les pilotes USB Exxotest v2.x.x et le pilote PCI Jungo v6.0.3 (Kernel Plugin v1.2.4, VXD 4.34) • Suppression des comptages de cartes superflus dans les fonctions MuxInit, MuxPciGetCardInfo et MuxUsbGetCardInfo • Les librairies Winsock 2 and IP Helper sont maintenant chargées dynamiquement au point d'entrée • Modification de MuxEthCountCards et ajout d'une nouvelle fonction MuxEthGetCardInfo • Modification des fonctions MuxOpen & MuxClose pour la gestion des cartes Ethernet (filaire et sans fil), mise à jour automatique de la table de routage • Les fonctions MuxWifiCountCards & MuxWifiGetCardInfo sont désormais obsolètes : utiliser MuxEthCountCards & MuxEthGetCardInfo • Changement mineur dans la routine d'initialisation de l'EEPROM pour les cartes PCI • La fonction MuxGetHardwareID est valide uniquement pour les cartes USB • Les erreurs CAN redondantes ne sont maintenant signalées qu'une seule fois par ms (cartes PCI) • Modification mineure dans la gestion des IFR LIN (cartes PCI) • Correction dans le traitement de la longueur des trames pour les bus ISO 9141 & ISO 14230 (cartes PCI) • Ajout de la fonction NMEA0183ConfigUart

Liste des éditions successives

Version	Date	Créé / Modifié par
1	01/09/2009	Cyril BOJIDAROVITCH
Modification		
Création du document		
Version	**Date**	**Créé / Modifié par**
2	21/10/2010	Gaël PERAGOUX
Modification		
Mise à jour du document		
Version	**Date**	**Créé / Modifié par**
3	14/03/2011	Gaël PERAGOUX
Modification		
Mise à jour du document		
Version	**Date**	**Créé / Modifié par**
4	29/06/2011	Cyril BOJIDAROVITCH
Modification		
Mise à jour du document		
Version	**Date**	**Créé / Modifié par**
5	21/10/2011	Nouredine BOUNEMOURA
Modification		
Mise à jour du document		
Version	**Date**	**Créé / Modifié par**
6	28/10/2011	Gaël PERAGOUX
Modification		
Mise à jour du document		