

# **Interface logicielle PC pour cartes CAN**

## **Guide utilisateur**

**30/06/2011**

**DUT-MUX-0191 /V1.8**

Auteur :

Cédric Rousset

Approbation :

Jean-Francois Mercier

Page laissée intentionnellement blanche

## I. Contenu

<b>I. CONTENU .....</b>	<b>1</b>
<b>II. AVERTISSEMENTS.....</b>	<b>4</b>
<b>III. BUT DU DOCUMENT .....</b>	<b>5</b>
<b>IV. PRESENTATION GENERALE .....</b>	<b>6</b>
IV.1 Architecture de l'interface logicielle .....	6
IV.2 Cartes CAN concernées .....	7
IV.3 Systèmes supportés .....	8
<b>V. MISE EN ŒUVRE DE L'INTERFACE LOGICIELLE .....</b>	<b>9</b>
V.1 Présentation .....	9
V.2 Composition de l'interface logicielle.....	9
V.2.1 NSICANEX.DLL .....	9
V.2.2 NSICANEX.LIB .....	10
V.2.3 Les fichiers .H.....	10
V.2.4 Programme de test .....	10
V.2.5 Programmes d'exemple .....	11
V.2.6 Fichiers d'installation .....	11
V.2.7 Pilotes de périphériques .....	11
V.3 Modes de fonctionnement .....	12
V.3.1 Mode BUFFER .....	12
V.3.2 Mode FIFO .....	12
V.3.3 Mode ANALYSE (périphériques USB seulement) .....	13
V.4 Liste des fonctions de l'interface logicielle .....	14
V.5 Fonctions disponibles sur les différentes cartes .....	15
V.6 Séquences d'appel des requêtes .....	16
V.6.1 Schéma global d'utilisation de l'interface .....	16
V.6.2 Tableau récapitulatif .....	17
V.7 Fonctionnalités .....	19
V.7.1 Activation d'un message.....	19
V.7.2 Réception par "polling" en mode BUFFER.....	20
V.7.3 Réception par événement en mode BUFFER .....	20
V.7.4 Réception par événement en mode FIFO .....	21
V.7.5 Gestion des émissions périodiques (périphériques USB seulement).....	22
V.7.6 Gestion du Veille-Réveil .....	23
V.7.7 Gestion des trames segmentées (Périphériques USB uniquement) .....	24
<b>VI. DESCRIPTION DES TYPES .....</b>	<b>26</b>
VI.1 t_CANObj .....	26
VI.2 t_CANevent .....	27
VI.3 t_CANbusParams.....	30
VI.4 t_CANflowParams .....	31
VI.5 t_CANcounter .....	32
VI.6 t_CANsjaCounters .....	33
VI.7 t_CANchipInfo .....	34
VI.8 t_CANdeviceInfo .....	35

<b>VII. DESCRIPTION DES FONCTIONS DE L'INTERFACE.....</b>	<b>37</b>
VII.1 Ic_ActiveId .....	37
VII.2 Ic_ChangeId.....	38
VII.3 Ic_ConfigBus.....	39
VII.4 Ic_ConfigEvent.....	41
VII.5 Ic_DeactivateId.....	43
VII.6 Ic_DeleteId .....	44
VII.7 Ic_EnumCards .....	45
VII.8 Ic_ExitDrv .....	46
VII.9 Ic_GetAPIInfo .....	47
VII.10 Ic_GetBuf .....	48
VII.11 Ic_GetChipInfo .....	49
VII.12 Ic_GetChipState.....	50
VII.13 Ic_GetCount.....	51
VII.14 Ic_GetDeviceInfo .....	52
VII.15 Ic_GetDiag.....	53
VII.16 Ic_GetEvent .....	54
VII.17 Ic_GetMode .....	55
VII.18 Ic_InitChip.....	56
VII.19 Ic_InitDrv .....	57
VII.20 Ic_InitFlowControl .....	58
VII.21 Ic_InitId .....	59
VII.22 Ic_InitInterface .....	60
VII.23 Ic_InitLineDrv .....	61
VII.24 Ic_InitPeriod .....	62
VII.25 Ic_KillPeriod .....	64
VII.26 Ic_ReadChip .....	65
VII.27 Ic_ResetBoard .....	66
VII.28 Ic_ResetChip .....	67
VII.29 Ic_SetMode.....	68
VII.30 Ic_SetRxMask .....	69
VII.31 Ic_StartPeriod .....	71
VII.32 Ic_StartChip.....	72
VII.33 Ic_StopChip.....	73
VII.34 Ic_StopPeriod .....	74
VII.35 Ic_TxMsg.....	75
VII.36 Ic_WriteChip .....	77
VII.37 Ic_WriteData .....	78
VII.38 Ic_WritePattern.....	79
<b>VIII. TRUCS ET ASTUCES .....</b>	<b>80</b>
VIII.1 Détecter et sortir de l'état BUS OFF .....	80
VIII.2 Utiliser plus de 14 identificateurs (i82527).....	82
VIII.3 Dépanner une application.....	82
VIII.4 Utiliser un compilateur non Microsoft.....	83
VIII.4.1 Linkage de l'application avec la DLL NSICANEX .....	83
VIII.4.2 Alignement des structures de données.....	83
<b>IX. INSTALLATION DE L'INTERFACE LOGICIELLE .....</b>	<b>85</b>
IX.1 Installation pour une carte CAN PCMCIA .....	86
IX.1.1 Windows 2000.....	86

IX.1.2 Windows XP.....	87
IX.1.3 Windows VISTA.....	88
IX.2 Installation d'une carte CANPCI.....	91
IX.2.1 Windows 2000.....	91
IX.2.2 Windows XP.....	92
IX.2.3 Windows VISTA.....	93
IX.2.4 Windows 7.....	95
IX.3 Installer un périphérique USB.....	100
IX.3.1 Sous Windows 2000.....	100
IX.3.2 Sous Windows XP.....	101
IX.3.3 Sous Windows VISTA.....	102
IX.3.4 Windows 7.....	103
IX.4 Dépannage de l'installation.....	108
<b>X. DESINSTALLATION DE L'INTERFACE LOGICIELLE.....</b>	<b>109</b>
X.1 Windows 2000.....	109
X.2 Windows XP.....	110
X.3 Windows VISTA.....	110
<b>XI. ANNEXE : PROTOTYPES DES FONCTIONS.....</b>	<b>111</b>
<b>XII. ANNEXE : FONCTIONS DE L'API WIN32.....</b>	<b>114</b>
<b>XIII. ANNEXE : PROGRAMME D'EXEMPLE.....</b>	<b>116</b>
XIII.1 Déclarations.....	116
XIII.2 Initialisations.....	118
XIII.2.1 Interface.....	118
XIII.2.2 Messages.....	120
XIII.2.3 Messages segmentés.....	121
XIII.2.4 Périodiques.....	122
XIII.3 Démarrage.....	123
XIII.4 Emissions.....	124
XIII.5 Arrêt du programme.....	125
XIII.6 Réceptions.....	126
<b>XIV. ANNEXE : NOTES SUR L'UTILISATION DE L'ADRESSAGE ETENDU DIAG ON CAN.....</b>	<b>127</b>
XIV.1 Initialisations.....	127
XIV.2 Déclaration des messages.....	127
XIV.3 Obtenir l'octet d'adressage utilisé par l'émetteur pour les messages en réception ...	128
XIV.4 Changement de valeur de l'octet d'adressage.....	128

## **II. Avertissements**

Les éléments contenus dans ce document sont fournis à titre d'information. Ils pourront faire l'objet de modifications sans préavis et ne sauraient en aucune manière engager la société anonyme NSI.

La société anonyme NSI ne saurait en aucun cas être tenue pour responsable d'une quelconque erreur contenue dans ce document, ainsi que des éventuelles conséquences pouvant en résulter.

Aucune partie de ce document ne peut être reproduite à d'autres fins que l'usage personnel de l'acheteur sans la permission expresse et écrite de la société anonyme NSI.

### **III. But du document**

Le but de ce document est de donner à l'utilisateur toutes les informations nécessaires à l'utilisation de l'interface logicielle pour les cartes CAN NSI.

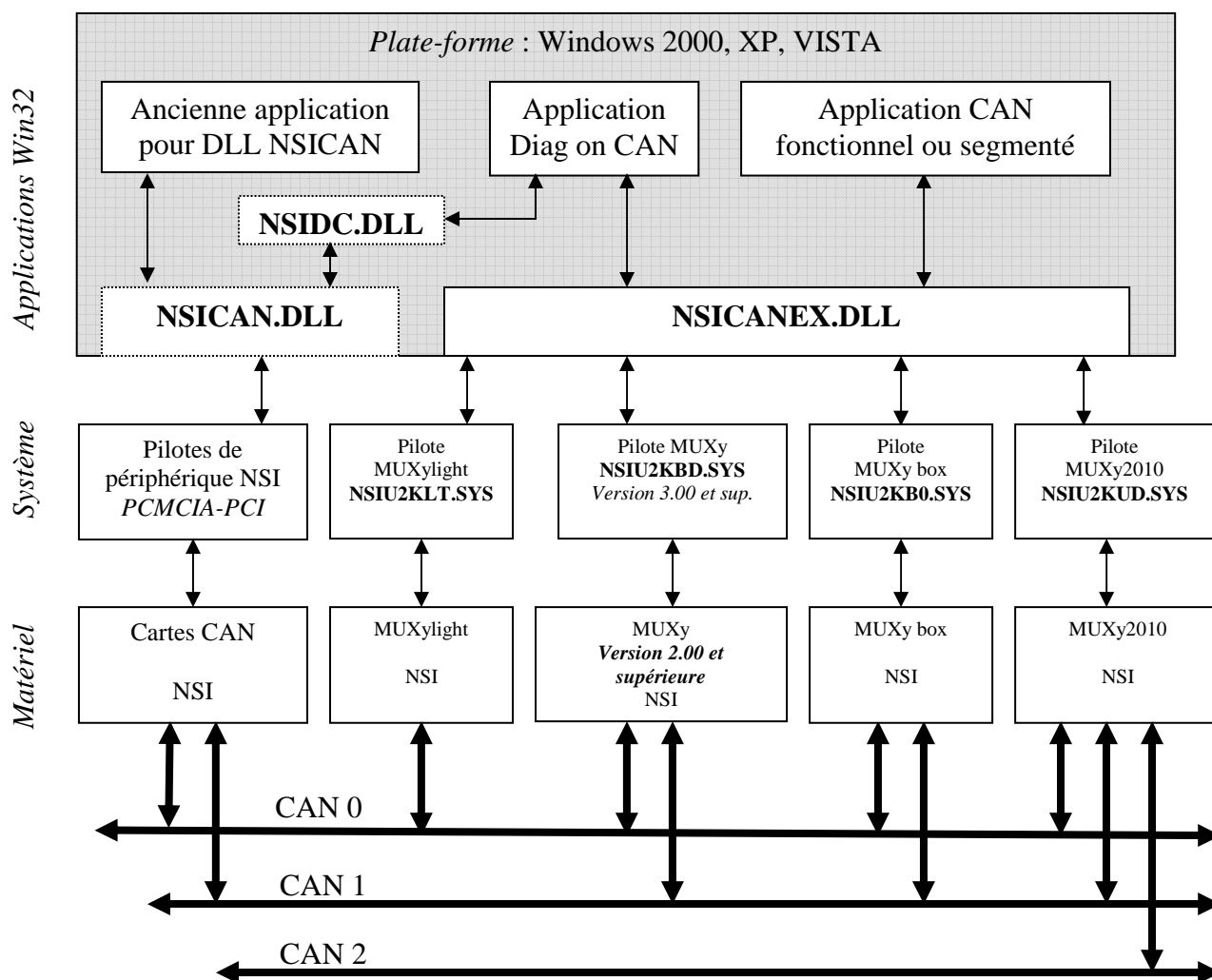
## IV. Présentation générale

L'interface logicielle CAN permet aux utilisateurs de réaliser leurs propres applications pour Windows en utilisant les cartes CAN NSI au travers d'une interface simple et rapide à mettre en œuvre.

L'interface logicielle CAN se présente sous la forme d'une série de fonctions dont l'utilisation est décrite dans ce document. Ces fonctions sont exécutées par une DLL qu'une ou plusieurs applications peuvent appeler simultanément. Cette DLL est identique pour toutes les plates formes Windows supportées.

L'interface logicielle permet à l'utilisateur de faire abstraction aussi bien du système d'exploitation que du type de carte CAN utilisé. Voir les paragraphes suivants pour connaître les caractéristiques des différentes cartes CAN et les systèmes d'exploitations supportés par cette interface logicielle. L'interface logicielle peut gérer plusieurs cartes CAN simultanément.

### IV.1 Architecture de l'interface logicielle





## IV.2 Cartes CAN concernées

Référence NSI	Désignation	Bus	Nb Canaux	Description
D31-M0039/69	CAN PCMCIA	PCMCIA	1	Carte pour bus PCMCIA.
D31-M0041/70	CAN PCMCIA /Opto	PCMCIA	1	Idem CAN PCMCIA avec isolation galvanique.
D31-M0068	CAN PCMCIA /LS	PCMCIA	1	Idem CAN PCMCIA avec interface CAN Low Speed.
KT000304	CANPCI	PCI	2	Carte pour bus PCI avec deux canaux CAN
KT000305	CANPCI /Opto	PCI	2	Idem CANPCI avec isolation galvanique
KT000306	CANPCI /LS	PCI	2	Idem CANPCI avec interface CAN Low Speed
KT000307	CANPCI /HS /LS	PCI	2	Idem CANPCI avec interface High Speed et Low Speed
KT005394	MUXy	USB	2	MUXy
KT007078	MUXy box	USB	2	MUXy box
KT007664	MUXy light	USB	1	MUXy light
KT009208	MUXy2010	USB	3	MUXy2010

### **IV.3 Systèmes supportés**

L'interface logicielle CAN fonctionne pour des ordinateurs de type PC fonctionnant avec les systèmes d'exploitation suivants :

- Windows 2000
- Windows XP (32 bits)
- Windows VISTA (32 bits)

## V. Mise en œuvre de l'interface logicielle

### V.1 Présentation

Les fonctions de l'interface logicielle permettent à des programmeurs de réaliser des applications pour Windows utilisant les cartes CAN NSI. Ces fonctions permettent de configurer les paramètres du bus CAN, de définir des messages CAN, segmentés ou non, puis de les émettre et de les recevoir. Toutes les fonctions de l'interface logicielle sont définies en Langage C dans la librairie : **NSICANEX.DLL**.

L'interface logicielle permet d'utiliser toutes les cartes CAN NSI énumérées dans le paragraphe *Cartes CAN supportées*. Chaque périphérique dispose d'un à trois canaux CAN que l'interface gère indépendamment les uns des autres. Une application peut utiliser plusieurs canaux simultanément. Deux applications peuvent utiliser des canaux indépendamment l'une de l'autre. Par contre, deux applications ne peuvent pas utiliser le même canal CAN au même moment.

Pour utiliser un canal CAN, une application doit premièrement l'ouvrir (**Ic\_InitDrv**). Lors de cette ouverture, un identificateur unique est retourné par l'interface. Cet identificateur - le "Handle" - doit ensuite être utilisé par l'application lors de chaque appel aux fonctions de l'interface pour identifier ce canal. Un canal ne peut pas être ouvert par une autre application tant qu'il n'a pas été refermé (**Ic\_ExitDrv**).

Une application peut utiliser tous les types de trames CAN avec des identificateurs standard (identificateurs codés sur 11 bits) ou étendus (identificateurs codés sur 29 bits) : trames de données, demandes de transmission distante et transmission automatique d'une trame de données en réponse à une demande de transmission distante.

Une application peut aussi utiliser des types de trames particuliers afin que les cartes CAN qui en ont la possibilité (MUXy) facilitent la gestion de la segmentation des messages CAN.

### V.2 Composition de l'interface logicielle

Les chemins des répertoires indiqués sont relatifs au répertoire **\Pilotes PC – PC drivers** du CD-ROM intitulé « CD-ROM Livraison NSI ».

#### V.2.1 NSICANEX.DLL

Le fichier **NSICANEX.DLL** (*Dynamic Link Library*) exporte toutes les fonctions de l'interface logicielle. Les applications doivent appeler ce fichier DLL pour utiliser l'interface logicielle. La DLL NSICANEX communique les ordres de l'application vers la carte CAN au travers d'un pilote de périphérique. Pour lier une application avec le fichier DLL, le compilateur utilise généralement un fichier **LIB** qui définit les points d'entrées de chaque fonction exportée. Les prototypes des fonctions sont fournis dans plusieurs fichiers **H** décrits ci-dessous.

Le fichier NSICANEX.DLL est copié par le processus d'installation dans le répertoire **System** ou **System32** d'où il est accessible par toutes les applications. Il n'est pas nécessaire de copier ce fichier à un autre emplacement. Se référer au chapitre *Installation de l'interface logicielle* pour plus de détails.

### V.2.2 NSICANEX.LIB

Le fichier **NSICANEX.LIB** permet de lier une application avec NSICANEX.DLL lors de la construction d'une application. Le fichier NSICANEX.LIB livré dans le répertoire **Logiciels\_PC-PC\_Software\Ex\_PC\_Software\_CAN\Include** est spécifique aux environnements de développement Microsoft. En cas d'incompatibilité, ce fichier peut être généré pour d'autres outils de développement grâce aux fichiers **DEF** ou **DLL**. Le fichier **LIB** doit être copié dans le répertoire du projet de l'application et inséré dans le projet comme indiqué dans la documentation de l'outil de développement utilisé. Voir le paragraphe *Utiliser un compilateur non Microsoft* dans le chapitre *Trucs et Astuces*.

### V.2.3 Les fichiers .H

Les fichiers **.H** livrés dans le répertoire **Logiciels\_PC-PC\_Software\Ex\_PC\_Software\_CAN\Include** du CD-ROM définissent en langage C les prototypes des fonctions de l'interface logicielle. Le fichier principal est **CANPCEX.H**. Celui-ci inclut deux autres fichiers qui séparent la déclaration des structures et des constantes (**CANDEFEX.H**) de la déclaration des prototypes des fonctions (**CANPROEX.H**). Ces trois fichiers doivent être copiés dans le répertoire du projet de l'application et le fichier **CANPCEX.H** doit être inclus dans le code source par la directive suivante :

```
#include "canpcex.h"
```

**Attention** : Les fichiers H de l'interface logicielle contiennent des directives de compilation dont la syntaxe est spécifique aux outils Microsoft. Celles-ci doivent impérativement être modifiées en fonction du compilateur utilisé. Voir le paragraphe *Utiliser un compilateur non Microsoft* dans le chapitre *Trucs et Astuces*.

### V.2.4 Programme de test

Un programme de test de la carte CAN (**CANEXTEST.EXE**) est livré dans le répertoire **Logiciels\_PC-PC\_Software\Ex\_PC\_Software\_CAN\Test** du CD-ROM. Ce programme permet de vérifier que le pilote de la carte et la DLL sont correctement installés et que la carte CAN fonctionne. Il est possible d'ouvrir ce programme plusieurs fois et de l'utiliser simultanément sur des canaux CAN différents. Si les canaux sont directement reliés entre eux et initialisés au même débit avec une résistance de terminaison, on pourra vérifier que la communication s'effectue bien.

Le programme CANEXTEST démarre en affichant la liste des canaux CAN détectés ainsi que leur configuration. Indiquer le numéro du canal à utiliser puis choisir le débit avec les touches du pavé numérique. Une fois que le canal est initialisé, la touche [A] permet d'émettre une trame de données. Toutes les trames de données standard sont reçues et affichées. Lorsqu'une trame est reçue ou correctement émise, une ligne affiche l'identificateur, le type et les données de cette trame. La touche ESC permet de libérer le canal et d'arrêter le programme.

### V.2.5 Programmes d'exemple

Des programmes d'exemple sont livrés dans le répertoire **Logiciels\_PC-PC\_Software\Ex\_PC\_Software\_CAN\Samples** du CD-ROM.

- Le premier qui utilise l'interface en mode FIFO est expliqué en détail dans l'annexe *Programme d'exemple*.
- Le second programme montre comment utiliser l'interface en mode BUFFER.
- Le Troisième met en œuvre les périodiques.
- Le quatrième traite de la segmentation CAN.
- Le cinquième illustre la gestion des modes Veille-Réveil.

### V.2.6 Fichiers d'installation

Les fichiers d'installation (**.INF** et **.BAT**) permettent d'installer le pilote de la carte CAN en fonction du système d'exploitation. Se référer au chapitre *Installation de l'interface logicielle* pour plus de détails sur l'utilisation de ces fichiers.

### V.2.7 Pilotes de périphériques

Les fichiers pilotes de périphérique nécessaires sont automatiquement copiés lors de l'installation de l'interface logicielle. Les applications ne doivent normalement pas accéder directement à ces fichiers :

- **NSIC2KIS.SYS** : Pilote Windows XP/2000/VISTA pour cartes PCMCIA
- **NSIC2K2P.SYS** : Pilote Windows XP/2000/VISTA pour cartes CANPCI
- **NSIU2KBD.SYS** : Pilote Windows XP/2000/VISTA pour MUXy
- **NSIU2KBO.SYS** : Pilote Windows XP/2000/VISTA pour MUXy box
- **NSIU2KLT.SYS** : Pilote Windows XP/2000/VISTA pour MUXy light
- **NSIU2KUD.SYS** : Pilote Windows XP/2000/VISTA pour MUXy2010

## V.3 Modes de fonctionnement

Le mode de fonctionnement détermine la façon dont les messages CAN, segmentés ou non, sont stockés par l'interface logicielle. Le terme message désigne ici, une trame CAN standard (8 octets), une trame CAN segmentée (jusqu'à 4095 octets) ou une erreur survenue lors de la communication CAN segmentée.

L'interface logicielle propose trois modes de fonctionnement: Le mode FIFO, le mode BUFFER et le mode ANALYSE. Le choix du mode dépend principalement du type d'application voulue. Ce mode est choisi par l'application lors de l'initialisation de chaque canal (**Ic\_InitInterface**). La majorité des fonctions sont utilisables dans tous les modes alors que d'autres sont spécifiques à un mode particulier.

### V.3.1 Mode BUFFER

En mode BUFFER, l'interface CAN alloue automatiquement un buffer permettant de mémoriser un événement (fin d'émission, réception d'un message) pour chaque message déclaré par l'application (**Ic\_InitId**). L'application peut lire un buffer lorsqu'elle veut traiter les informations mémorisées: lecture des données d'un message reçu par exemple (**Ic\_GetBuf**). Si l'application ne lit pas le contenu d'un buffer, les anciennes données sont remplacées par celles des nouveaux messages. L'application dispose ainsi des données les plus récentes dans chaque buffer déclaré.

Jusqu'à 14 messages CAN différents peuvent être actifs simultanément avec le contrôleur de protocole i82527 et jusqu'à 63 pour les périphériques USB comme MUXy ou le boîtier CAN-USB. Dans les 2 cas, les autres identificateurs, éventuellement filtrés par l'apposition d'un masque (**Ic\_SetRxMask**), peuvent être regroupés dans un buffer spécial (**\_CAN\_DUMMY\_ID**). L'arrivée d'un message dans un buffer vide peut être signalé à l'application par un événement (**Ic\_ConfigEvent**).

*Note* : Pour le cas des messages segmentés, aucun buffer n'est créé pour l'identificateur de contrôle de flux. Ces identificateurs sont stockés dans le buffer global (**\_CAN\_DUMMY\_ID**).

### V.3.2 Mode FIFO

En mode FIFO, l'interface CAN mémorise dans une file d'attente tous les événements réseau, (fin de transmission, réception d'un message). Les événements sont rangés dans l'ordre d'arrivée. Le premier événement enregistré est le premier sorti. L'application doit venir vider cette file d'attente régulièrement (**Ic\_GetEvent**). Si la file d'attente est saturée (500 messages), les nouveaux événements ne sont plus mémorisés et un événement spécial "événements perdus" est inséré dans la FIFO. Après une saturation, l'enregistrement des événements est automatiquement suspendu tant que 25% d'espace n'est pas libéré dans la FIFO.

Jusqu'à 14 messages individuels peuvent être actifs simultanément avec le contrôleur de protocole i82527 et jusqu'à 63 pour les périphériques USB comme MUXy ou le boîtier CAN-USB. Les identificateurs déclarés en réception de données, éventuellement filtrés par l'apposition d'un masque peuvent être regroupés et reçus en plus des 14 ou des 63 autres identificateurs (**Ic\_SetRxMask**). L'arrivée d'un événement dans une file d'attente vide peut être signalé à l'application par un événement (**Ic\_ConfigEvent**).

### **V.3.3 Mode ANALYSE (périphériques USB seulement)**

Le mode ANALYSE fonctionne comme le mode FIFO. La différence est que le canal CAN est automatiquement initialisé pour recevoir tous les identificateurs circulant sur le bus CAN connecté sans les acquitter. Les événements sont stockés dans la FIFO et doivent être dépilés par l'application (**Ic\_GetEvent**). Dans ce mode les transmissions sont interdites.

Note : Dans ce mode, les trames segmentées ne sont pas reconstruites par la Muxy. La reconstruction du message complet restera à la charge de l'application PC développée à l'aide des dll NSI.

## V.4 Liste des fonctions de l'interface logicielle

Nom	Fonction
<b>Configuration, initialisation</b>	
<i>Ic_EnumCards</i>	Lecture du nombre de canaux connectés et de leur configuration.
<i>Ic_InitDrv</i>	Ouverture d'un canal.
<i>Ic_GetDeviceInfo</i>	Informations sur le périphérique.
<i>Ic_GetAPIInfo</i>	Retourne les versions des couches logicielles
<i>Ic_ResetChip</i>	Réinitialisation du contrôleur CAN.
<i>Ic_InitChip</i>	Initialisation du contrôleur (débit, point d'échantillonnage ...)
<i>Ic_ConfigBus</i>	Configuration pour l'utilisation d'une interface de ligne externe.
<i>Ic_InitInterface</i>	Initialisation du mode de fonctionnement.
<i>Ic_ConfigEvent</i>	Passage du handle de l'événement signalé pour les différents modes.
<i>Ic_InitId</i>	Déclaration d'un identificateur
<i>Ic_ChangeId</i>	Modification de la valeur d'un identificateur déclaré.
<i>Ic_InitFlowControl</i>	Configuration des paramètres de flux pour un identificateur.
<i>Ic_DeactivateId</i>	Inactivation d'un identificateur déclaré.
<i>Ic_DeleteId</i>	Supprime un identificateur déclaré.
<i>Ic_SetRxMask</i>	Configuration d'un groupe de réception.
<b>Activité réseau</b>	
<i>Ic_StartChip</i>	Démarrage du contrôleur CAN.
<i>Ic_WriteData</i>	Mise à jour d'un message en transmission, sans activation de l'ident.
<i>Ic_WritePattern</i>	Mise à jour des données d'un identificateur périodique
<i>Ic_ActiveId</i>	Activation d'un message déclaré en transmission sans mise à jour.
<i>Ic_TxMsg</i>	<i>Ic_WriteData</i> + <i>Ic_Active_Id</i>
<i>Ic_GetChipInfo</i>	Lecture des informations propres au contrôleur CAN
<i>Ic_GetChipState</i>	Lecture de l'état du composant (erreur active, ...)
<i>Ic_GetBuf</i>	Lecture des données les plus anciennes d'un buffer.
<i>Ic_GetEvent</i>	Lecture d'un événement en mode FIFO
<i>Ic_GetCount</i>	Lecture des compteurs d'erreurs particuliers
<i>Ic_GetDiag</i>	Lecture de l'état de la communication en CAN Low-speed
<i>Ic_StopChip</i>	Arrêt du contrôleur CAN.
<b>Gestion des périodiques</b>	
<i>Ic_InitPeriod</i>	Déclaration d'un identificateur périodique
<i>Ic_StartPeriod</i>	Démarrage de l'émission d'un identificateur périodique
<i>Ic_StopPeriod</i>	Suspension de l'émission d'un identificateur périodique
<i>Ic_KillPeriod</i>	Suppression d'un identificateur périodique
<b>Gestion du Veille-Réveil</b>	
<i>Ic_InitLineDrv</i>	Configuration de l'interface de ligne
<i>Ic_SetMode</i>	Commande l'état du driver de ligne
<i>Ic_GetMode</i>	Détecte l'état du driver de ligne
<b>Sortie du driver</b>	
<i>Ic_ExitDrv</i>	Arrêt du programme et restitution de l'environnement.
<b>Accès direct au composant i82527</b>	
<i>Ic_ReadChip</i>	Lecture à une adresse du contrôleur CAN.
<i>Ic_WriteChip</i>	Ecriture à une adresse du contrôleur CAN.



## V.5 Fonctions disponibles sur les différentes cartes

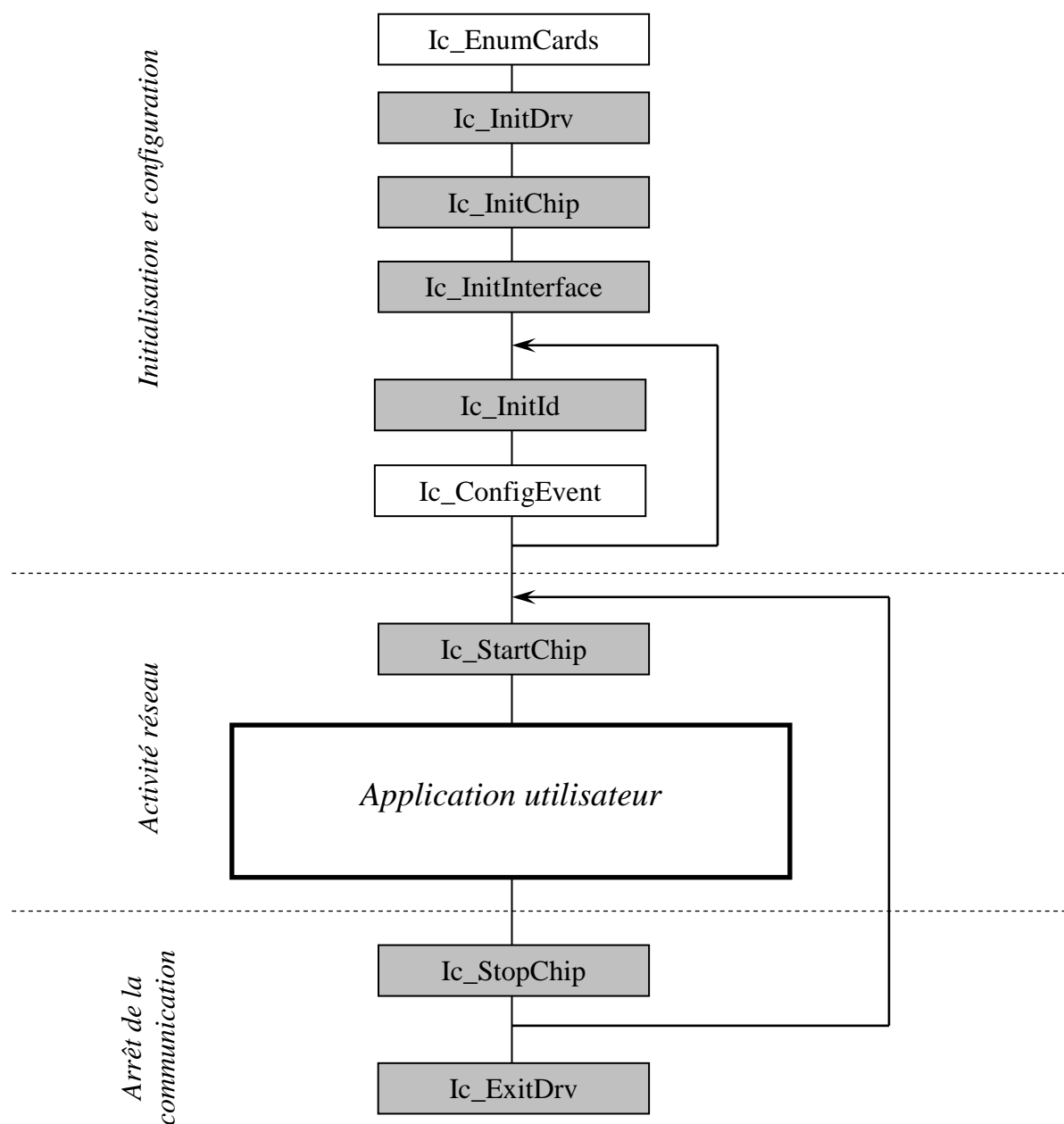
Les fonctions disponibles sur les différentes cartes NSI sont indiqués par ✓. Ces fonctions doivent retourner \_OK, si les paramètres d'appels sont correctement renseignés.

Les fonctions non disponibles retournent \_DRV\_PARAM\_ERR ou \_BOARD\_ERR.

Requêtes	CANPCMCIA	CANPCI	CANPCMCIA /LS CANPCI /LS	MUXy MUXy light	MUXy box MUXy2010
Ic_ActiveId	✓	✓	✓	✓	✓
Ic_ChangeId	✓	✓	✓	✓	✓
Ic_ConfigBus	✓	✓	✓		
Ic_ConfigEvent	✓	✓	✓	✓	✓
Ic_DeactivateId	✓	✓	✓	✓	✓
Ic_DeleteId				✓	✓
Ic_EnumCards	✓	✓	✓	✓	✓
Ic_ExitDrv	✓	✓	✓	✓	✓
Ic_GetAPIInfo	✓	✓	✓	✓	✓
Ic_GetBuf	✓	✓	✓	✓	✓
Ic_GetChipInfo	✓	✓	✓	✓	✓
Ic_GetChipState	✓	✓	✓	✓	✓
Ic_GetCount	✓	✓	✓	✓	✓
Ic_GetDeviceInfo	✓	✓	✓	✓	✓
Ic_GetDiag			✓		✓
Ic_GetEvent	✓	✓	✓	✓	✓
Ic_GetMode			✓		✓
Ic_InitChip	✓	✓	✓	✓	✓
Ic_InitDrv	✓	✓	✓	✓	✓
Ic_InitFlowControl				✓	✓
Ic_InitId	✓	✓	✓	✓	✓
Ic_InitInterface	✓	✓	✓	✓	✓
Ic_InitLineDrv			✓		✓
Ic_InitPeriod				✓	✓
Ic_KillPeriod				✓	✓
Ic_ReadChip	✓	✓	✓		
Ic_ResetBoard	✓	✓	✓	✓	✓
Ic_ResetChip	✓	✓	✓	✓	✓
Ic_SetMode			✓		✓
Ic_SetRxMask	✓	✓	✓	✓	✓
Ic_StartChip	✓	✓	✓	✓	✓
Ic_StartPeriod				✓	✓
Ic_StopChip	✓	✓	✓	✓	✓
Ic_StopPeriod				✓	✓
Ic_TxMsg	✓	✓	✓	✓	✓
Ic_WriteChip	✓	✓	✓		
Ic_WriteData	✓	✓	✓	✓	✓
Ic_WritePattern				✓	✓

## V.6 Séquences d'appel des requêtes

### V.6.1 Schéma global d'utilisation de l'interface



L'appel de ces fonctions est impératif et l'ordre doit être respecté.



L'appel de ces fonctions est facultatif ; il dépend de l'application.

## V.6.2 Tableau récapitulatif

L'appel des fonctions de l'interface doit respecter un certain ordre. Si la séquence d'appel n'est pas respectée, l'interface logicielle retourne le code d'erreur **\_SEQ\_ERR** (erreur de séquence).

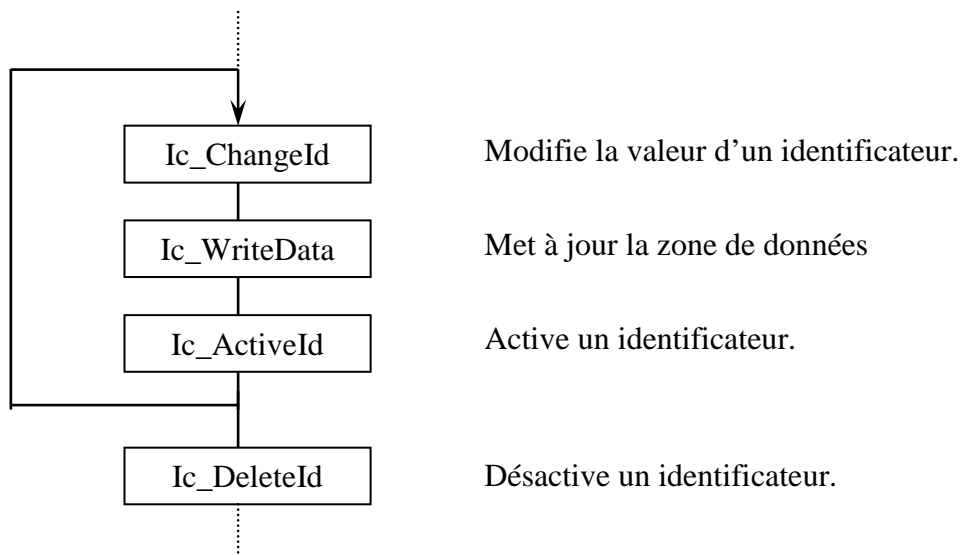
Requêtes	Etat	REPOS	INIT DRV OK	INIT CHIP OK	INIT INTERFACE OK	START CHIP OK	START PERIOD OK
Ic_EnumCards		X	X	X	X	X	X
Ic_InitDrv		X					
Ic_GetAPIInfo		X	X	X	X	X	X
Ic_GetDeviceInfo			X	X	X	X	X
Ic_InitChip			X				
Ic_ResetBoard*			X	X	X	X	X
Ic_ResetChip**			X	X	X	X	X
Ic_InitLineDrv			X	X	X		
Ic_InitInterface				X			
Ic_ConfigBus				X	X		
Ic_InitId					X		
Ic_SetRxMask					X		
Ic_InitFlowControl					X	X	X
Ic_ConfigEvent					X	X	X
Ic_InitPeriod					X	X	X
Ic_ChangeId					X	X	X
Ic_StartChip					X		
Ic_DeactivateId						X	X
Ic_DeleteId						X	X
Ic_WriteData						X	X
Ic_WritePattern					X	X	
Ic_ActiveId						X	X
Ic_TxMsg						X	X
Ic_StartPeriod						X	
Ic_StopPeriod						X	X
Ic_KillPeriod						X	X
Ic_GetMode						X	X
Ic_SetMode						X	X
Ic_GetDiag						X	X
Ic_GetChipState						X	X
Ic_GetBuf						X	X
Ic_GetEvent						X	X
Ic_GetCount						X	X
Ic_GetChipInfo						X	X
Ic_StopChip						X	X
Ic_ExitDrv			X	X	X	X	X
Ic_ReadChip			X	X	X	X	X
Ic_WriteChip			X	X	X	X	X

Le tableau suivant précise quelles requêtes provoquent les changements d'état de l'interface. Le code retour doit indiquer **\_OK** pour que le changement ait eu lieu.

Etats	Requêtes de transition
REPOS	Ic_ExitDrv
INIT_DRV_OK	Ic_InitDrv
INIT_CHIP_OK	Ic_InitChip
INIT_INTERFACE_OK	Ic_InitInterface Ic_StopChip
START_CHIP_OK	Ic_StartChip
START_PERIOD_OK	Ic_StartPeriod ou Ic_StartChip avec autoStart

## V.7 Fonctionnalités

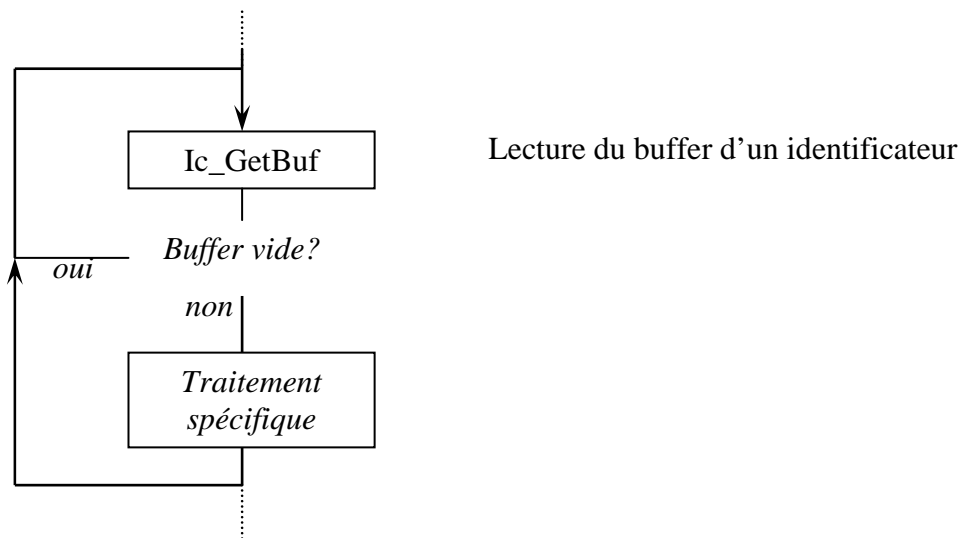
### V.7.1 Activation d'un message



Pour activer un message, l'appel de la fonction **Ic\_ActiveId** suffit. Il est possible de modifier la valeur d'un identificateur déclaré après avoir démarré le contrôleur avec **Ic\_ChangeId** puis de mettre à jour les données avec la fonction **Ic\_WriteData**.

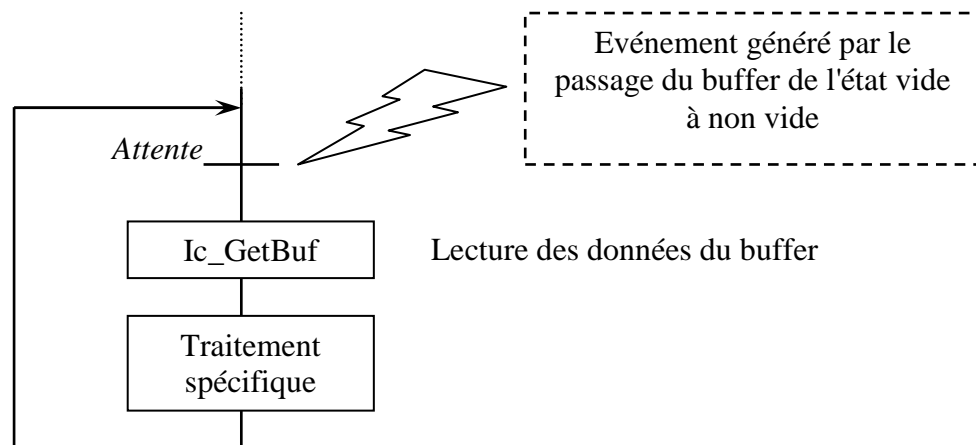
Remarque : La séquence **Ic\_WriteData** suivie de **Ic\_ActiveId** peut être optimisée par un seul appel à la fonction **Ic\_TxMsg**.

### V.7.2 Réception par "polling" en mode BUFFER



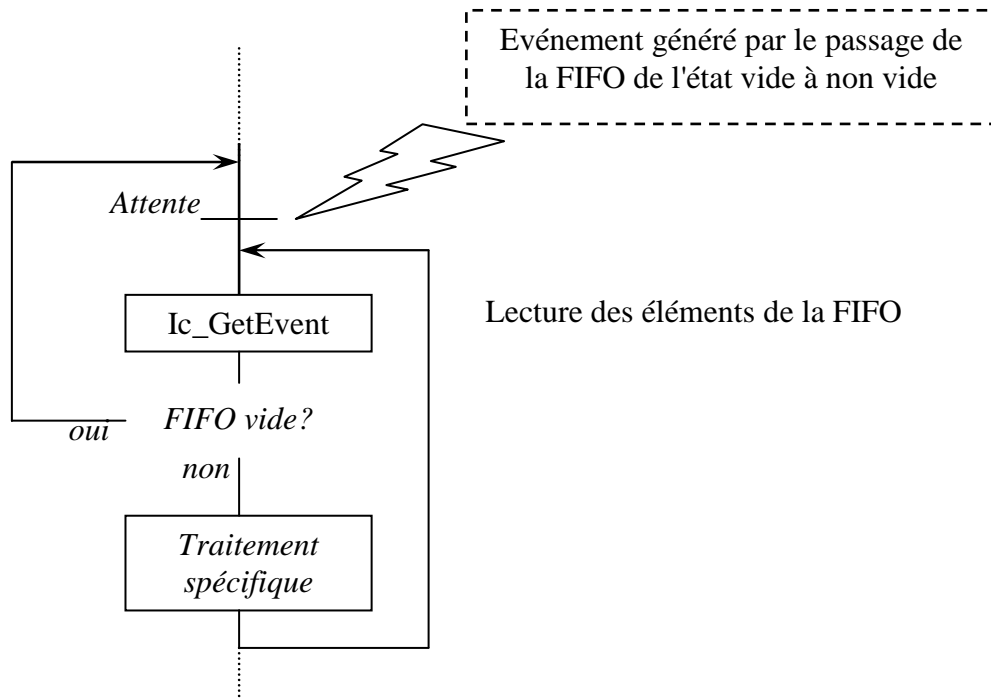
L'application vient scruter l'état d'un BUFFER lorsqu'elle le désire (**Ic\_GetBuf**). Celui-ci peut être vide ou peut contenir les dernières données de l'identificateur concerné.

### V.7.3 Réception par événement en mode BUFFER



L'application attend que le buffer passe de l'état vide à non vide pour lire les données enregistrées. Chaque buffer peut disposer d'un événement individuel ce qui permet de déclencher un traitement différent (Thread) pour chaque message déclaré.

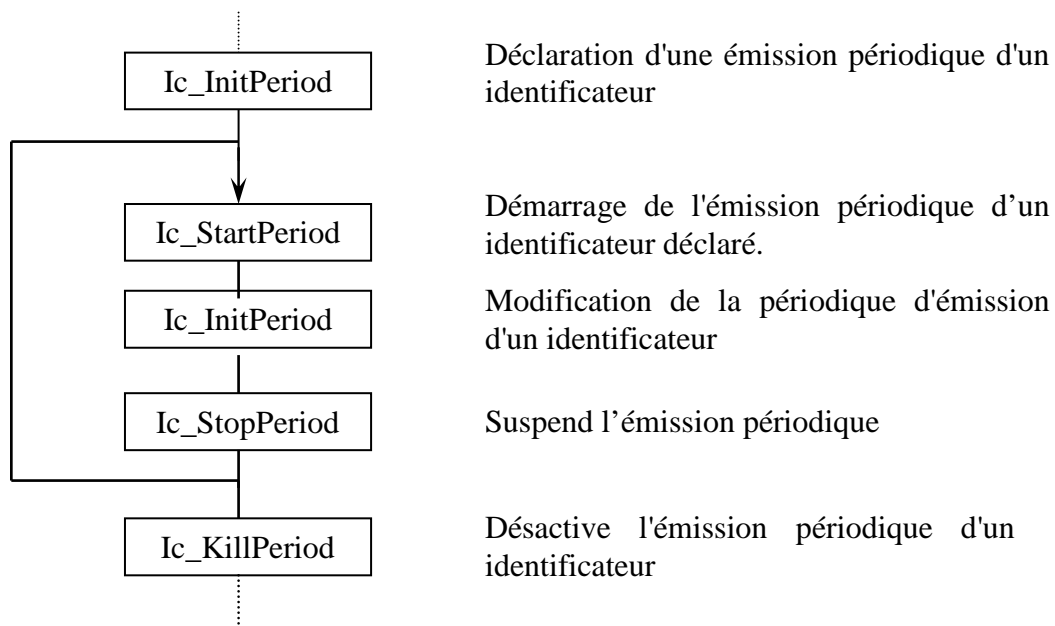
## V.7.4 Réception par événement en mode FIFO



Lorsque la FIFO n'est plus vide, un évènement est généré par l'interface. L'application peut le détecter et venir dépiler tous les éléments présents dans la FIFO. En effet, plus d'un élément peut avoir été mis dans la FIFO entre l'occurrence de l'évènement et le traitement par l'application. Si la FIFO n'est pas entièrement vidée pour chaque évènement reçu, aucun évènement n'est plus généré et la FIFO risque de "déborder".

### V.7.5 Gestion des émissions périodiques (périphériques USB seulement)

Il est possible de décharger le PC de la gestion de l'émission d'identificateurs périodiques. Ces identificateurs périodiques peuvent être entièrement gérés par la partie embarquée des périphériques USB grâce à 4 fonctions de l'interface logicielle : **Ic\_InitPeriod**, **Ic\_StartPeriod**, **Ic\_StopPeriod**, **Ic\_KillPeriod**.



Après avoir déclaré un identificateur et initialisé sa période (**Ic\_InitPeriod**), l'émission peut être démarrée (**Ic\_StartPeriod**) ou arrêtée (**Ic\_StopPeriod**) à tout moment. Il est possible de modifier la valeur de la période en cours de fonctionnement (**Ic\_InitPeriod**). Si un nombre de répétitions a été spécifié avec la fonction **Ic\_InitPeriod**, il n'est pas nécessaire d'appeler la fonction **Ic\_StopPeriod**, l'interface se chargera d'arrêter automatiquement l'émission de l'identificateur quand le nombre de répétitions sera atteint.

Il est aussi possible d'émettre des motifs périodiques supérieurs à 8 octets grâce à la fonction **Ic\_WritePattern**. Si un motif a été programmé avec la fonction **Ic\_WritePattern**, le périphérique se chargera du découpage en paquets dont la taille est spécifiable par la même fonction (8 octets maximum). Cette fonction ne peut être appelée si l'émission périodique est démarrée. A l'exception des boîtiers CAN-USB, il est possible d'utiliser la fonction **Ic\_WriteData** (limitée à 8 octets) pour modifier les données d'une trame sans arrêter l'émission périodique.

*Note:* Il est impossible de définir un message CAN segmenté (**\_CAN\_TX\_SEG\_DATA** et **\_CAN\_RX\_SEG\_DATA**) comme étant périodique.



## V.7.6 Gestion du Veille-Réveil

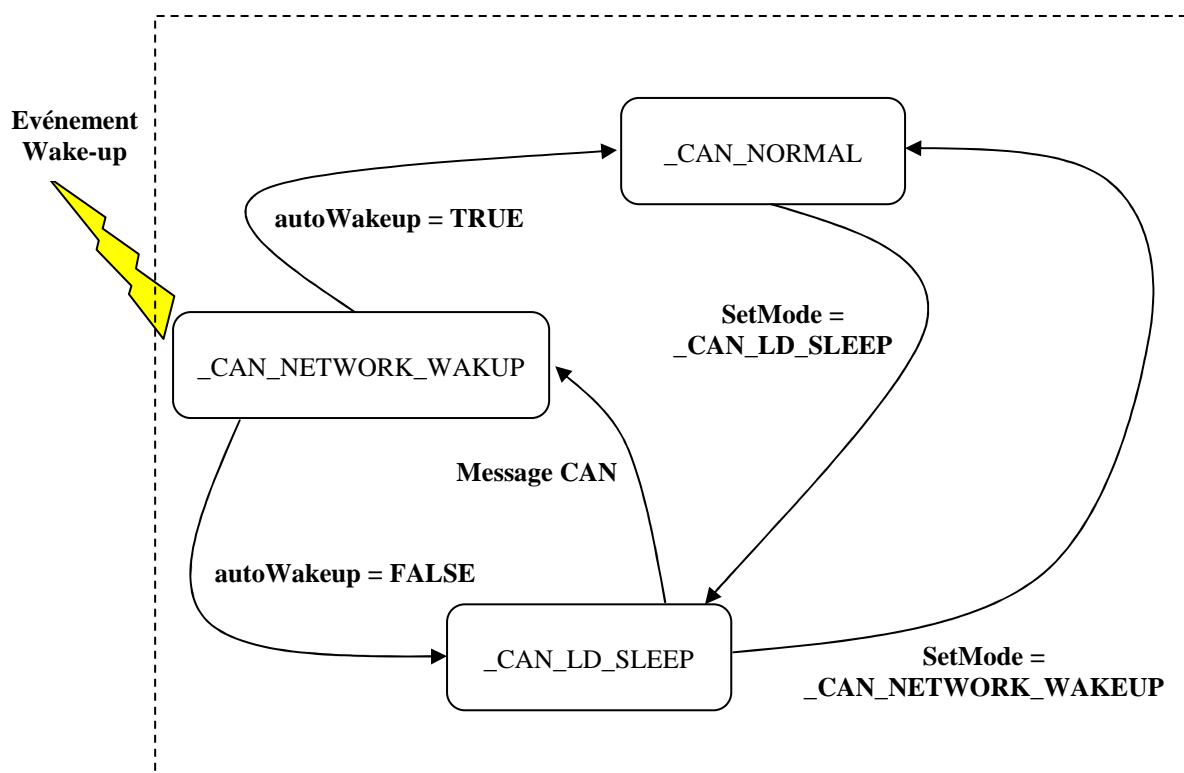
La gestion du Veille / Réveil CAN concerne les cartes suivantes :

- CANPCI équipées des d'interfaces de lignes Low-Speed sur carte fille,
- CAN-USB,
- MUXy box,
- MUXy2010

L'interface logicielle CAN est capable de choisir le mode de l'interface de ligne (High speed ou Low speed) avec la fonction **Ic\_InitLineDrv**.

Une fois que l'interface est en **mode Low-Speed**, la fonction **Ic\_SetMode** permet de changer l'état de l'interface de ligne. Les fonctions **Ic\_GetMode** et **Ic\_GetDiag** permettent respectivement, de lire l'état de l'interface de ligne et de lire le diagnostic de ligne du bus CAN. Les boîtiers **MUXy box** et **MUXy2010** permettent de gérer les fonctions Veille / Réveil en **mode CAN High speed**.

Le diagramme d'état suivant synthétise les différents modes de l'interface de ligne. Ce mode se récupère via la fonction **Ic\_GetMode**.



### V.7.7 Gestion des trames segmentées (Périphériques USB uniquement)

A la base rien ne les différencie des émissions de trames CAN normales. Cependant, la mise en forme des données ainsi que la gestion des timings (STMin, Timeouts) sont gérées par le périphérique CAN. Il est possible de gérer jusqu'à 8 identificateurs segmentés par canal CAN.

Lorsqu'une émission est demandée (identificateur déclaré comme `_CAN_TX_SEG_DATA`), la mise en forme des données à émettre sur le bus (SINGLE FRAME ou FIRST FRAME suivie de CONSECUTIVE FRAME, les champs N\_PCI) est effectuée automatiquement. Les timings sont aussi gérés automatiquement et un compte-rendu de l'émission est effectué seulement à la fin de l'émission (effectuée correctement ou partiellement accompagnée de la cause de l'erreur).

En réception (identificateur déclaré comme `_CAN_RX_SEG_DATA`), les trames reçues sur le bus (SINGLE FRAME ou FIRST FRAME suivie de CONSECUTIVE FRAME) sont traitées par le périphérique et mises en formes afin que l'utilisateur ne récupère que les données utiles de la trames.

*Exemple :*

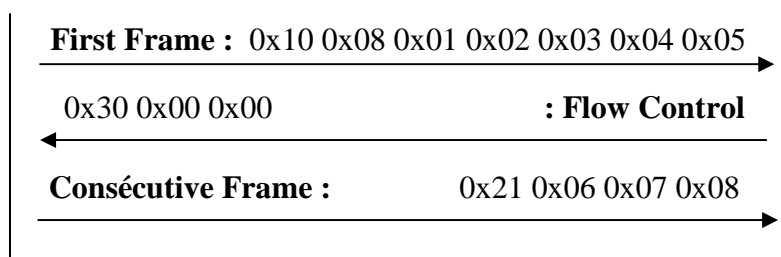
*Déclaration d'une trame segmentée en émission portant l'identificateur 0x100 avec 8 octets de données {0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,}*

*On utilise les 2 canaux CAN de la prise MUXy rebouclés.*

*Sur le bus CAN on aura les trames suivantes :*

**MUXy canal CAN 1**

**MUXy canal CAN 2**

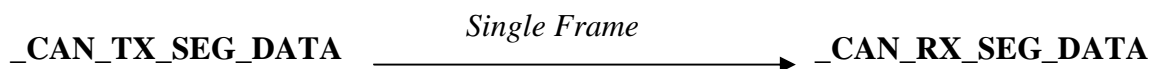


Les différents types d'évènements signalés par l'interface logicielle dans le cas d'échanges segmentés sont consignés ci-dessous.

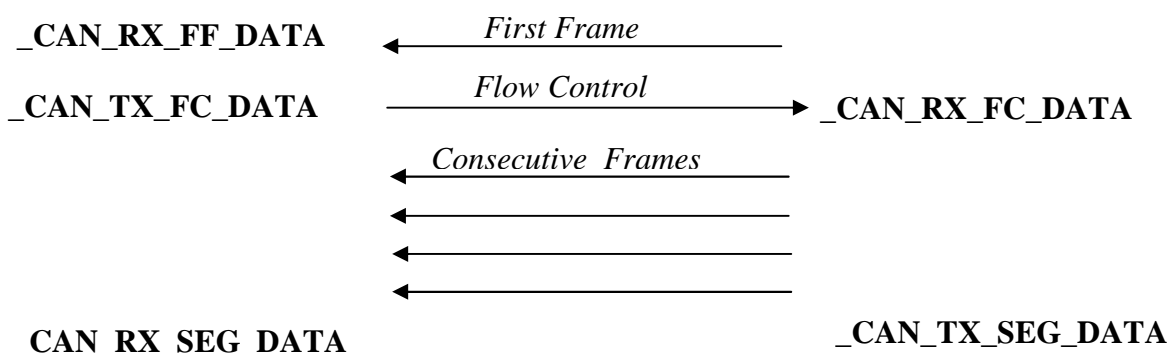
### Transmission

### Réception

#### 1. Single Frame



#### 2. First Frame + Consecutives Frames



*Remarque* : Seuls les identificateurs déclarés par la fonction **Ic\_InitId** avec le champ *status* = *\_STATUS* sont signalés dans la FIFO. Lors de la déclaration de l'identificateur segmenté, il est possible de signaler ou non les événements FLOW\_CONTROL avec le champ *flowControlStatus*.

## VI. Description des types

**Notation** : pour les définitions de types (typedef), le préfixe **t\_** est utilisé.

### VI.1 t\_CANobj

La structure de type objet CAN (t\_CANobj) est utilisée par la requête **Ic\_InitId**. Elle permet d'initialiser un message.

```
typedef struct {
    unsigned long      ident;
    t_CANidentType     identType;
    t_CANframeType     frameType;
    t_StatusRq         statusRq;
    unsigned long      identFlowControl;
    t_StatusRq         flowControlStatusRq;
    unsigned char      sourceAddress;
    unsigned char      targetAddress;
    unsigned short      dlc;
    unsigned long      reserved;
    unsigned char      Data[_CAN_MAX_DATA_EX];
} t_CANobj;
```

#### Paramètres :

**ident :** Valeur de l'identificateur CAN du message. De 000h à 7FFh pour les identificateurs standards et de 0h à 1FFFFFFFh pour les identificateurs étendus.

**identType :** Type d'identificateur.

- **\_CAN\_STD** : Type d'identificateur Standard (11 bits)
- **\_CAN\_EXT** : Type d'identificateur Etendu (29 bits)

**frameType :** Type de trame. Les valeurs possibles sont les suivantes :

- **\_CAN\_TX\_DATA** : Transmission d'une trame de données.
- **\_CAN\_RX\_DATA** : Réception d'une trame de données.
- **\_CAN\_TX\_RX\_REMOTE** : Emission d'une demande de transmission distante et réception de la réponse éventuelle.
- **\_CAN\_TX\_AUTO\_REMOTE** : Transmission automatique de la réponse sur réception d'une demande de transmission distante.
- **\_CAN\_RX\_REMOTE** : Réception d'une demande de transmission distante sans réponse automatique.
- **\_CAN\_RX\_SEG\_DATA** : Réception d'un message segmenté.
- **\_CAN\_TX\_SEG\_DATA** : Emission d'un message segmenté.

**statusRq :** Ce champ est significatif en mode FIFO uniquement. Si le compte-rendu est demandé, chaque fin d'échange (fin de transmission...) provoque un événement qui est ajouté dans la file d'attente des événements. La désactivation du compte

rendu permet d'éviter de saturer la FIFO avec les événements de fin de transmission dans certaines applications :

- **\_STATUS** : Compte-rendu d'échange demandé.
- **\_NO\_STATUS** : Pas de compte-rendu d'échange demandé.

Pour les messages en réception ce champ est automatiquement mis à **\_STATUS**.

#### *identFlowControl:*

Ce champ est significatif seulement pour les types de trames segmentées **\_CAN\_RX\_SEG\_DATA** et **\_CAN\_TX\_SEG\_DATA**. Il s'agit de l'identificateur de contrôle de flux.

#### *flowControlStatusRq :*

Ce champ est significatif en mode FIFO uniquement. Si le compte-rendu est demandé, chaque émission ou réception de l'identificateur de contrôle de flux provoque un événement qui est ajouté dans la file d'attente des événements.

- **\_STATUS** : Compte-rendu d'échange demandé.
- **\_NO\_STATUS** : Pas de compte-rendu d'échange demandé.

#### *sourceAddress :*

Champ actuellement non-utilisé.

#### *targetAddress :*

Octet utilisé pour l'adressage étendu (voir en Annexe).

#### *dlc :*

Nombre d'octets de la zone données à initialiser. Ce champ est significatif pour les types de trame **\_CAN\_TX\_DATA**, **\_CAN\_TX\_RX\_REMOTE**, **\_CAN\_TX\_SEG\_DATA** et **\_CAN\_TX\_AUTO\_REMOTE**

- Valeurs autorisées : de **0** à **8** pour les messages standards (non segmentés)
- Valeurs autorisées : de **0** à **4095** pour les messages **\_CAN\_TX\_SEG\_DATA**

#### *reserved :*

Réservé

#### *data[\_CAN\_MAX\_DATA\_EX] :*

Tableau d'octets contenant les données à initialiser. Ce champ est significatif pour les types de trame **\_CAN\_TX\_DATA**, **\_CAN\_TX\_SEG\_DATA** et **\_CAN\_TX\_AUTO\_REMOTE**.

*Note* : Le tableau est dimensionné pour les types **\_CAN\_RX\_SEG\_DATA** et **\_CAN\_TX\_SEG\_DATA** où la longueur maximale est de 4095 octets.

## VI.2 t\_CANevent

La structure **t\_CANevent** est utilisée par toutes les requêtes relatives aux événements provoqués par les messages CAN (fin de transmission, réception, etc.)

```
typedef struct
{
    t_CANframeType    eventType;
    t_CANerr          CANerr;
    t_DCerr           DCerr;
    unsigned long     ident;
```

```

        t_CANidentType    identType;
        unsigned short    dlc;
        unsigned char      data[_CAN_MAX_DATA_EX];
        unsigned long      timeStamp;
        unsigned long      reserved;
    } t_CANevent;

```

Paramètres :**eventType :** Type de l'évènement.

- **\_CAN\_TX\_DATA :** Transmission d'une trame de données.
- **\_CAN\_RX\_DATA :** Réception d'une trame de données.
- **\_CAN\_TX\_AUTO\_REMOTE :** Emission automatique d'une réponse suite à une réception d'une demande de transmission distante.
- **\_CAN\_TX\_REMOTE :** Fin de transmission d'une demande de transmission distante.
- **\_CAN\_RX\_DATA\_REMOTE :** Réception de la réponse à une demande de transmission distante.
- **\_CAN\_RX\_REMOTE :** Réception d'une demande de transmission distante.
- **\_CAN\_RX\_SEG\_DATA :** Réception d'un message segmenté
- **\_CAN\_TX\_SEG\_DATA :** Emission d'un message segmenté
- **\_CAN\_RX\_FF\_DATA :** Réception d'un message *FIRST FRAME*. Il informe l'utilisateur qu'une réception de message segmenté est entamée.
- **\_CAN\_RX\_FC\_DATA :** Réception d'un message *FLOW CONTROL*. Il informe l'utilisateur que le message *FIRST FRAME* est bien arrivé à son destinataire.
- **\_CAN\_TX\_FC\_DATA :** Emission d'un message *FLOW CONTROL*. Il informe l'utilisateur que la réception d'un message segmenté va débiter.
- **\_CAN\_LOST\_MSG :** Indication que des évènements ont été perdus. FIFO saturée (modes FIFO et ANALYSE seulement).
- **\_CAN\_ERR :** Indication d'erreur CAN ne pouvant être reliée à un message spécifique (ie. BusOff)

**CANerr :** Etat du contrôleur de protocole :

- **\_CAN\_ACTIVE\_ERR :** Erreur active (mode normal).
- **\_CAN\_PASSIVE\_ERR :** Erreur passive.
- **\_CAN\_BUS\_OFF :** Contrôleur déconnecté du bus (Bus off).
- **\_CAN\_UNKNOWN :** Inconnu.

**DCerr :** Statut de l'échange segmenté (Optionnel) :

- **\_DC\_OK** La communication s'est déroulée correctement (entre autres, les timings définis ont été respectés).

- **\_DC\_TIMEOUT\_Ar** La trame FLOW CONTROL n'a pas été acquitté avant le temps **Ar**.
- **\_DC\_TIMEOUT\_As** Une trame CONSECUTIVE FRAME n'a pas été acquittée avant le temps **As**.
- **\_DC\_TIMEOUT\_Bs** La trame FLOW CONTROL n'a pas été reçue sur un message de type **\_CAN\_TX\_SEG\_DATA**.
- **\_DC\_TIMEOUT\_N\_Cr** Une trame CONSECUTIVE FRAME n'a pas été reçue avant le temps **Cr** sur un message **\_CAN\_RX\_SEG\_DATA**.
- **\_DC\_WRONG\_SN** Une trame CONSECUTIVE FRAME a été reçue avec un numéro de séquence incorrect sur un message de type **\_CAN\_RX\_SEG\_DATA**.
- **\_DC\_UNEXPECTED\_PDU** La trame reçue comporte des octets inattendus sur un message de type **\_CAN\_RX\_SEG\_DATA**.
- **\_DC\_DATA\_LAYER\_ERR** La trame reçue comporte une erreur de protocole sur un message de type **\_CAN\_RX\_SEG\_DATA**.
- **\_DC\_UNEXPECTED\_ERR** Une erreur interne s'est produite durant la communication.

**Ident :** Valeur de l'identificateur concerné par l'évènement.

- De **0** à **0x1FFFFFFF**

**identType :** Type de l'identificateur :

- **\_CAN\_STD** : Type d'identificateur Standard (11 bits)
- **\_CAN\_EXT** : Type d'identificateur Etendu (29 bits)

**dlc :** Nombre d'octets de la zone de données.

- de **0** à **8** pour les messages standards
- de **0** à **4095** pour les messages segmentés. En cas d'erreur dans la communication (**DCerr** ≠ **\_DC\_OK**), ce nombre peut être inférieur à celui déclaré car il indique le nombre d'octets correctement reçus.

**data :** Tableau d'octets de la zone de données du message reçu. En cas d'erreur (**DCerr** ≠ **\_DC\_OK**), seules les données reçues correctement sont disponibles (champs **dlc**). Dans le cas d'une transmission, les octets de données ne sont pas disponibles.

**datation :** Date d'arrivée de la trame CAN (précision 100µs).

**reserved :** Réserve

### VI.3 t\_CANbusParams

La structure t\_CANbusParams permet de configurer les paramètres du bus CAN.

```
typedef struct
{
    UCHAR    baudpresc;
    UCHAR    tseg1;
    UCHAR    tseg2;
    UCHAR    sjw;
    UCHAR    sample;
} t_CANbusParams;
```

#### Paramètres :

**baudpresc :** (Baud Rate Prescaler : BRP) Diviseur d'horloge. La valeur paramétrée dans les registres "Bit Timing Register" du contrôleur.

Valeurs autorisées : 1 à 64.

**tseg1 :** Temps du segment 1 (segment précédant le point d'échantillonnage). La valeur paramétrée dans les registres "Bit Timing Register" du contrôleur.

Valeurs autorisées : 3 à 16 (4 à 16 pour MUXy2010)

**tseg2 :** Temps du segment 2 (segment suivant le point d'échantillonnage). La valeur paramétrée dans les registres "Bit Timing Register" du contrôleur.

Valeurs autorisées : 2 à 8.

**sjw :** (Re)Synchronization Jump Width. La valeur paramétrée dans les registres "Bit Timing Register" du contrôleur.

Valeurs autorisées : 1 à 4.

**Attention, sur MUXy et MUXy box ce paramètre doit être inférieur ou égal à la plus petite valeur entre tseg1 et tseg2.**

**sample :** (SPL) Mode d'échantillonnage. La valeur paramétrée dans les registres "Bit Timing Register" du contrôleur.

- **0** : 1 échantillon par bit
- **1** : 3 échantillons par bit (*impossible pour les prises MUXy et MUXy2010*)

Note : Les valeurs passées à la fonction **Ic\_InitChip** sont les valeurs "réelles" (physiques) du bit. Les valeurs programmées dans les registres du contrôleur CAN sont les valeurs réelles moins 1.

Le débit obtenu peut être calculé avec la formule suivante en utilisant les valeurs réelles :

$$\text{Débit} = \frac{16'000}{2 \times brp \times (1 + tseg1 + tseg2)} \text{ kbits / s}$$



La position du point d'échantillonnage par rapport au début du bit peut être calculée par la formule suivante :

$$Position = \frac{1 + tseg1}{1 + tseg1 + tseg2} \times 100\%$$

Quelques exemples de débits usuels:

Débit	Point d'échant.	BaudPresc	Tseg1	Tseg2
25 kbit/s	50 %	20	7	8
62,5 kbit/s	75 %	8	11	4
125 kbit/s	75 %	4	11	4
250 kbit/s	81 %	2	12	3
500 kbit/s	81 %	1	12	3
1000 kbit/s	75 %	1	5	2

#### VI.4 t\_CANflowParams

La structure t\_CANflowParams permet de configurer les paramètres de flux pour un ou tous les identificateurs CAN segmentés.

```
typedef struct
{
    USHORT    blockSize;    // Block size
    USHORT    STmin;        // ST min
    ULONG     As;           // As
    ULONG     Ar;           // Ar
    ULONG     Bs;           // Bs
    ULONG     Cr;           // Cr
    ULONG     Br;           // Br
} t_CANflowParams;
```

##### Paramètres :

**blockSize :** Nombre de trames CAN entre chaque trame de contrôle de flux (FC). Cette valeur est utile si l'identificateur est déclaré en réception. En émission, cette valeur est renseigné par le calculateur distant.

- de 0 à 255

**STmin :** En réception (\_CAN\_RX\_SEG\_DATA): temps minimum imposé entre deux CONSECUTIVE FRAME. Ce paramètre est renvoyé par la trame de FLOW CONTROL automatique.

- De 0 à 127 : Temps STmin en ms
- De 241 à 249 : Temps STmin de 100µs (241d = F1h) à 900µs (249d=F9h)

En transmission (\_CAN\_TX\_SEG\_DATA) : ce temps est imposé par la trame de FLOW CONTROL reçue après l'émission de la FIRST FRAME. Ce

paramètre indique à l'émetteur la durée minimale à insérer entre la fin d'émission d'une Consecutive Frame et le début de transmission de la Consecutive Frame suivante.

- De **0** à **127** : Temps STmin en ms
- De **241** à **249** : Temps STmin de 100µs (241d = F1h) à 900µs (249d=F9h)

**As :** Durée maximale d'émission d'une SINGLE FRAME, FIRST FRAME, CONSECUTIVE FRAME.

- De **10** à **65535 ms**

**Ar :** Durée maximale d'émission d'une trame de FLOW CONTROL.

- De **10** à **65535 ms**

**Bs :** Timeout de reception d'une trame de FLOW CONTROL.

- De **10** à **65535 ms**

**Br :** Délai entre la réception d'une FIRST FRAME, CONSECUTIVE FRAME et l'émission d'une trame de FLOW CONTROL.

- De **0** à **65535 ms**

**Cr :** Timeout de réception d'une CONSECUTIVE FRAME.

- De **10** à **65535 ms**

## VI.5 t\_CANcounter

La structure t\_CANcounter est retournée par la fonction **Ic\_GetCount**. Elle comptabilise les émissions et les réceptions ainsi que les erreurs détectées par le contrôleur de protocole sur le bus CAN. Les compteurs sont remis à zéro lors de l'initialisation de l'interface.

```
typedef struct{
    unsigned long tx_ok;           // transmission OK
    unsigned long rx_ok;           // reception OK
    unsigned long bus_off;         // erreur de bus off
    unsigned long passive;         // erreur passive
    unsigned long stuff_err;       // erreur de codage
    unsigned long form_err;        // erreur de format
    unsigned long ack_err;         // erreur d'acquittement
    unsigned long bit1_err;        // hors champ d'arbitrage
    unsigned long bit0_err;        // hors champ d'arbitrage
    unsigned long crc_err;         // erreur de checksum
} t_CANcounter;
```

## VI.6 t\_CANsjaCounters

La structure `t_CANsjaCounters` est incluse dans la structure `t_CANchipInfo` utilisée par la fonction `Ic_GetChipInfo`. La structure représente les différents compteurs d'erreurs entretenus par le contrôleur de protocole SJA1000 (Boîtier CAN-USB seulement). Ces compteurs sont remis à zéro lors du démarrage du composant `Ic_StartChip` ou lors d'un reset du boîtier `Ic_ResetBoard`.

```
typedef struct {  
    WORD bit_error;    // erreur sur un bit  
    WORD form_err;     // erreur de format  
    WORD stuff_err;    // erreur de stuffing  
    WORD other_err;    // autre erreur  
    WORD id4_0;         // erreur sur l'ident. (bits 0 à 4)  
    WORD id12_5;        // erreur sur l'ident. (bits 5 à 12)  
    WORD id17_13;       // erreur sur l'ident. (bits 13 à 17)  
    WORD id20_18;       // erreur sur l'ident. (bits 18 à 20)  
    WORD id28_21;       // erreur sur l'ident. (bits 21 à 28)  
    WORD start_fr;  
    WORD bit_SRTR;  
    WORD bit_IDE;  
    WORD CRC_Seq;  
    WORD reserved0;  
    WORD data_field;  
    WORD data_length;  
    WORD bit_RTR;  
    WORD reserved1;  
    WORD empty_0;  
    WORD act_err;  
    WORD interm;  
    WORD tolerate_dom;  
    WORD empty_1;  
    WORD pas_err;  
    WORD err_del;  
    WORD CRC_del;  
    WORD ack_slot;  
    WORD eof;  
    WORD ack_del;  
    WORD overload_flag;  
    WORD lost_Rx;      // messages perdus  
    WORD frame_ok;  
    WORD bus_off;      // erreur de bus off  
    WORD passive;      // erreur passive  
}t_CANsjaCounters;
```

*Note :* Les significations exactes des champs de cette structure sont définies dans la documentation technique du contrôleur SJA1000.

## VI.7 t\_CANchipInfo

La structure t\_CANchipInfo est renseignée par la fonction **Ic\_GetChipInfo**.

```
typedef struct
{
    ULONG chipType;    // Type de contrôleur CAN
    union
    {
        struct
        {
            t_CANsjaCounters txCounters;
            t_CANsjaCounters rxCounters;
        } CAN_SJA;

        char reserved[512];
    }
} t_CANchipInfo;
```

### Paramètres :

<b>chipType :</b>	Type de contrôleur de protocole CAN du canal utilisé. <ul style="list-style-type: none"><li>• <b>_CAN_82527</b> : Intel i82527 (Cartes CANPC, CANPCI, ...)</li><li>• <b>_CAN_SJA100</b> : Philips SJA1000 (Boîtier CAN-USB)</li><li>• <b>_CAN_90F543</b> : Microcontrôleur Fujitsu 90F543 (MUXy)</li><li>• <b>_CAN_V850</b> : Microcontrôleur NEC V850 (MUXy2010)</li></ul>
<b>.txCounters:</b>	Compteurs en transmission du contrôleur SJA1000. Ce champ n'est renseigné que pour les boîtiers CAN-USB
<b>.rxCounters:</b>	Compteurs en réception du contrôleur SJA1000. Ce champ n'est renseigné que pour les boîtiers CAN-USB
<b>Reserved :</b>	Champ réservé.

### *Notes :*

Cette structure permettait de renseigner différentes valeurs de compteurs accessibles sur les gestionnaires de protocoles SJA1000 utilisés sur les boîtiers CAN-USB. Pour les autres types de boîtiers, seul le champ **chipType** est renseigné.

## VI.8 t\_CANdeviceInfo

La structure t\_CANdeviceInfo est retournée par la fonction **Ic\_GetDeviceInfo**. Elle permet d'obtenir des informations sur un périphérique CAN.

```
typedef struct{
    ULONG deviceType;           // Type de périphérique
    union {
        struct CAN_USB{
            short vendorID;      //VendorID
            short deviceID;      //DeviceID
            char productName[0x40]; //Nom produit
            char manufacturerName[0x40]; //Fabricant
            char serialNumber[0x80]; //N° de série
            short firmwareVersion; //Version code
            ULONG boardType;      //Type de carte
            ULONG reserved2;      //Réservé
            ULONG hardwareVersion; //Version carte
        };
        struct CAN_PCI{
            ULONG IOBaseAddress; //Adresse IO
            ULONG memoryBaseAddress[3]; //Mémoire base
            ULONG IRQLineNumber; //N° IRQ
            ULONG boardType;      //Type de carte
            char cardNameString[80]; //Nom de carte
            ULONG reserved1;      //Réservé
            ULONG reserved2;      //Réservé
        };
        struct CAN_ISA{
            ULONG IOBaseAddress; //Adresse IO
            ULONG IRQLineNumber; //N° IRQ
            ULONG boardType;      //Type de carte
            char cardNameString[80]; //Nom du canal
            ULONG reserved1;      //Réservé
            ULONG reserved2;      //Réservé
        };
        char reserved[512];
    }
} t_CANdeviceInfo;
```

**deviceType** : Type de périphérique/pilote CAN du canal utilisé.

- **\_CANPCISA** : Cartes CANPCMCIA  
- Pilote **NSICXXIS.XXX**
- **\_CANPCI2P** : Cartes CANPCI  
- Pilote **NSICXX2P.XXX**
- **\_OBDUSB** : Prises MUXy et MUXylight  
- Pilotes **NSIU2KBD.SYS** et **NSIU2KLT.SYS**
- **\_USBBOX** : MUXy box  
- Pilote **NSIU2KBO.SYS**
- **\_OBDUSB2010** : Prises MUXy2010  
- Pilote **NSIU2KUD.SYS**

Notes sur différents champs apparaissant dans les structures :

**firmwareVersion:** Version du code embarqué.

**hardwareVersion:** Version de la carte.

*Note:* Les versions sont retournées sous la forme: (version majeure x 100) + version mineure.

*Exemple :* 102 s'interprète comme la version 1.02

**boardType :** Type de carte CAN du canal utilisé.

- **\_CANPCMCIA** : Carte CANPCMCIA
- **\_CANPCMCIA\_LS** : Carte CANPCMCIA
- **\_CANPCMCIA\_OPTO** : Carte CANPCMCIA avec isolation galvanique
- **\_CANPCI** : Carte CANPCI
- **\_CANPCI\_LS** : Carte CANPCI Low-Speed
- **\_CANPCI\_OPTO** : Carte CANPCI avec isolation galvanique
- **\_CAN\_OBDUSB** : Prise MUXy standard
- **\_CANEX\_OBDUSB** : Prise MUXy (ISO 15765-2)
- **\_MUXYBOX** : MUXy box
- **\_MUXY2010** : MUXy2010

## VII. Description des fonctions de l'interface

### VII.1 Ic\_ActiveId

Déclenche l'émission d'un identificateur déclaré en transmission de données (`_CAN_TX_DATA`, `_CAN_TX_SEG_DATA`) ou de demande de transmission distante (`_CAN_TX_RX_REMOTE`). Cette fonction ne met pas à jour la zone données. Cette requête permet de réactiver les identificateurs en réception invalidés par **Ic\_KillId**. Si la transmission précédente n'est pas terminée ou n'a pas pu avoir lieu - problème d'acquittement sur le bus par exemple - le code d'erreur `_BUF_OCC` est retourné. Si le contrôleur CAN est en état BUS OFF (déconnexion logique du bus), le code `_CHIP_ERR` est retourné par cette fonction. Utiliser la fonction **Ic\_GetChipState** pour connaître l'état du contrôleur. L'appel de cette fonction sur une trame périodique ne déclenchera pas l'envoi d'une seule trame, mais bien de la séquence périodique complète.

```
short Ic_ActiveId(      HANDLE      hdrv,
                      unsigned long ident,
                      unsigned short dlc      );
```

#### Paramètres :

**hdrv** : Identificateur du canal retourné par la fonction **Ic\_InitDrv**.

**ident** : Valeur de l'identificateur, ou `_CAN_DUMMY_ID` pour réactiver le buffer global s'il a été désactivé par **Ic\_KillId**.

**dlc** : Taille de la zone de données à transmettre.

- jusqu'à **8** pour `_CAN_TX_DATA`, `_CAN_TX_AUTO_REMOTE`.
- jusqu'à **4095** pour `_CAN_TX_SEG_DATA`.

#### Code retour :

<code>_OK</code> :	Trame activée.
<code>_SEQ_ERR</code> :	Séquence invalide. Cette requête ne peut avoir lieu que lorsque le contrôleur est démarré.
<code>_PARAM_ERR</code> :	Paramètre invalide.
<code>_UNKNOWN_ID</code> :	Identificateur inconnu
<code>_BUF_OCCUPIED</code> :	Transmission en cours, requête non prise en compte. Vérifier que la station émettrice n'est pas seule sur le bus.
<code>_INVALID_OP</code> :	La valeur de <b>hdrv</b> est invalide.
<code>_INTERFACE_ERR</code> :	Fonction incompatible avec le mode d'interface (l'interface est probablement en mode ANALYSE)
<code>_SLEEP_MODE</code> :	Interface de ligne en mode veille.
<code>_DRV_PARAM_ERR</code> :	Problème d'accès au pilote. Vérifier l'installation de la carte.
<code>_CHIP_ERR</code> :	Problème d'accès au contrôleur ou contrôleur en état BUS OFF. Utiliser la fonction <b>Ic_GetChipState</b> pour déterminer l'état du contrôleur du protocole.
<code>_USB_ERR</code> :	Erreur de transmission USB.
<code>_BOARD_TIMEOUT</code> :	Pas d'acquittement du périphérique USB.

## VII.2 Ic\_ChangeId

Modifie la valeur d'un identificateur déclaré ainsi qu'éventuellement son identificateur de contrôle de flux. Seule la valeur de l'identificateur est modifiée. Les autres paramètres de la déclaration initiale (type d'identificateur, type de trame, périodicité...) sont conservés.

```
short Ic_ChangeId(      HANDLE      drv,
                      unsigned long  oldIdent,
                      unsigned long  newIdent,
                      unsigned long  newFlowControl  );
```

### Paramètres :

- drv :** Identificateur du canal retourné par la fonction **Ic\_InitDrv**.
- oldIdent :** Valeur de l'identificateur à modifier.
- newIdent :** Nouvelle valeur de l'identificateur.
- newFlowControl :** Nouvelle valeur de l'identificateur pour les trames de contrôle de flux. Ce paramètre n'est significatif que pour les trames déclarées en **\_CAN\_TX\_SEG\_DATA** et **\_CAN\_RX\_SEG\_DATA**.

### Code retour :

- \_OK :** Valeur de l'identificateur modifiée.
- \_SEQ\_ERR :** Séquence invalide.
- \_PARAM\_ERR :** Valeur de paramètre invalide.
- \_UNKNOWN\_ID :** Identificateur à modifier non déclaré.
- \_INTERFACE\_ERR :** Fonction incompatible avec le mode d'interface (l'interface est probablement en mode ANALYSE)
- \_INVALID\_OP :** La valeur de **drv** est invalide.
- \_DRV\_PARAM\_ERR :** Problème d'accès au pilote. Vérifier l'installation de la carte.
- \_CHIP\_ERR :** Problème d'accès au contrôleur. Pour une carte ISA vérifier la configuration.
- \_USB\_ERR :** Erreur de transmission USB.
- \_BOARD\_TIMEOUT :** Pas d'acquittement du périphérique USB.



### VII.3 Ic\_ConfigBus

Définit la configuration de l'interface physique du canal CAN. Cette fonction n'est pas disponible pour les périphériques USB. En cas d'appel à cette fonction dans ce cas, le code `_BOARD_ERR` est retourné. Si la requête **Ic\_ConfigBus** n'est pas appelée par l'application, l'interface utilise les valeurs par défaut.

```
Short  Ic_ConfigBus(      HANDLE hdrv,
                          short compbypass,
                          short polarite,
                          short disconnectTx1,
                          short disconnectRx1,
                          short disconnectRx0 );
```

Paramètres :

**hdrv :** Identificateur du canal retourné par la fonction **Ic\_InitDrv**.

**compbypass :** Permet de désactiver le comparateur d'entrée.

- **0** : Comparateur d'entrée actif.
- **1** : Comparateur d'entrée inactif. Réception sur Rx0. **DisconnectRx0** doit être à 0. Valeur par défaut pour l'utilisation de l'interface physique interne.

**polarity :** Uniquement si le comparateur d'entrée est inactif **compbypass** = 1

- **0** : Sur Rx0 : 0 logique = bit dominant, 1 logique = bit récessif.
- **1** : Sur Rx0 : 1 logique = bit dominant, 0 logique = bit récessif.

**disconnectTx1 :** Permet de déconnecter la sortie Tx1.

- **0** : Tx1 connectée.
- **1** : Tx1 déconnectée. Valeur par défaut pour utilisation de l'interface de ligne interne.

**disconnectRx1 :** Permet de connecter l'entrée Rx1 à l'entrée inversée du comparateur d'entrée.

- **0** : Rx1 connectée à l'entrée inversée du comparateur d'entrées. Valeur par défaut en utilisation de l'interface interne.
- **1** : Entrée Rx1 déconnectée.

**disconnectRx0 :** Permet de connecter l'entrée Rx0 à l'entrée non inversée du comparateur d'entrée.

- **0** : Entrée Rx0 connectée à l'entrée non inversée du comparateur d'entrées. Valeur par défaut en utilisation de l'interface interne.
- **1** : Entrée Rx0 déconnectée.

Attention : N'appeler cette fonction qu'en cas d'utilisation d'une interface physique externe. La configuration par défaut fonctionne avec l'interface physique interne de la carte.

Code retour :

_OK :	Configuration réussie
_SEQ_ERR :	Séquence invalide
_PARAM_ERR :	Paramètre invalide
_BOARD_ERR :	Type de carte ne supportant pas cette fonction
_INVALID_OP :	La valeur de <b>hdrv</b> est invalide.
_DRV_PARAM_ERR :	Problème d'accès au pilote. Vérifier l'installation de la carte.
_CHIP_ERR :	Problème d'accès au contrôleur. Pour une carte ISA vérifier la configuration.

## VII.4 Ic\_ConfigEvent

Indique à l'interface logicielle le "*HANDLE*" d'un événement système pour signaler le changement d'état d'un BUFFER, de la FIFO ou le passage du contrôleur dans l'état BUS OFF. Cette requête est optionnelle. Par défaut, aucun événement n'est généré par l'interface.

- En mode **FIFO**, l'événement est signalé lorsque la file d'attente des événements réseau passe de l'état « vide » à « non vide ».
- En mode **BUFFER**, l'événement est signalé lorsque le buffer de l'identificateur passé en paramètre passe de l'état « vide » à « non vide ».
- En mode **ANALYSE**, l'événement est signalé lorsque la file d'attente des événements réseau passe de l'état « vide » à « non vide ».

```
short  Ic_ConfigEvent(      HANDLE          hdrv,
                           HANDLE          hEvent
                           unsigned long    ident      );
```

## Paramètres :

**hdrv :** Identificateur du canal retourné par la fonction **Ic InitDrv**.

**hEvent :** Identificateur de l'événement obtenu par la fonction **Win32 CreateEvent** ou **NULL** pour annuler le signalement des événements. Cet événement peut être utilisé pour bloquer un *thread* en attente grâce à la fonction **WaitForSingleObject**.

**ident :** En mode BUFFER, identificateur CAN du buffer qui doit provoquer cet événement en passant de "vide" à "non vide".  
En mode FIFO ou ANALYSE, il faut passer la valeur 0 pour signaler le passage de la file de "vide" à "non vide".

Dans tous les cas, ce paramètre peut aussi permettre de spécifier le passage du contrôleur dans un état particulier.

- |                                   |  |
|-----------------------------------|--|
| • <b>_CAN_EVENT_BOFF</b>          | Contrôleur en bus off.   |
| • <b>_CAN_EVENT_WAKE_UP</b>       | Réveil du contrôleur<br>(CAN-USB /LS, MUXy box<br>uniquement, non<br>implémenté dans<br>MUXy2010). |
| • <b>_CAN_EVENT_PASSIVE_ERROR</b> | Passage en erreur passive<br>(CAN-USB, MUXy box<br>uniquement).                                    |

Code retour :

<u>_OK :</u>	Configuration de l'événement réussie.
<u>_SEQ_ERR :</u>	Séquence invalide.
<u>_UNKNOWN_ID :</u>	En mode BUFFER uniquement : Identificateur non déclaré.
<u>_PARAM_ERR :</u>	La valeur <b>hEvent</b> est invalide.
<u>_INVALID_OP :</u>	La valeur de <b>hdrv</b> est invalide.

`_DRV_PARAM_ERR` : Problème d'accès au pilote. Vérifier l'installation de la carte.

## VII.5 Ic\_DeactivateId

Désactive un identificateur déclaré et suspend l'émission des trames périodiques associées. Cet identificateur peut être revalidé par un appel à la fonction **Ic\_ActiveId**. Les identificateurs en réception sont automatiquement validés lors de l'appel à **Ic\_StartChip**.

```
Short Ic_DeactivateId( HANDLE      hdrv,  
                      unsigned long ident );
```

### Paramètres :

**hdrv :** Identificateur du canal retourné par la fonction **Ic\_InitDrv**.

**ident :** Valeur de l'identificateur déclaré ou **\_CAN\_DUMMY\_ID** pour désactiver le buffer de réception global (15 pour le i82527 et 63 pour les périphérique USB).

### Code retour :

<b>_OK :</b>	Identificateur désactivé.
<b>_SEQ_ERR :</b>	Séquence invalide.
<b>_UNKNOWN_ID :</b>	Identificateur non déclaré.
<b>_INVALID_OP :</b>	La valeur de <b>hdrv</b> est invalide.
<b>_DRV_PARAM_ERR :</b>	Problème d'accès au pilote. Vérifier l'installation de la carte.
<b>_CHIP_ERR :</b>	Problème d'accès au contrôleur. Pour une carte ISA vérifier la configuration.
<b>_USB_ERR :</b>	Erreur de transmission USB.
<b>_BOARD_TIMEOUT:</b>	Pas d'acquittement du périphérique USB.

## VII.6 Ic\_DeleteId

Cette fonction supprime totalement un identificateur déclaré. Cela libère ainsi une place pour faire un nouveau **Ic\_InitId**. Cette fonction est disponible uniquement pour les périphériques USB.

```
Short Ic_DeleteId(      HANDLE      hdrv,  
                     unsigned long  ident );
```

### Paramètres :

**hdrv :** Identificateur du canal retourné par la fonction **Ic\_InitDrv**.

**ident :** Identificateur du message à supprimer

### Code retour :

<b>_OK :</b>	Suppression de l'identificateur réussie.
<b>_SEQ_ERR :</b>	Séquence invalide.
<b>_UNKNOWN_ID :</b>	Identificateur non déclaré.
<b>_PARAM_ERR :</b>	Identificateur invalide.
<b>_INVALID_OP :</b>	La valeur de <b>hdrv</b> est invalide.
<b>_BOARD_ERR</b>	Type de carte ne supportant pas cette fonction.
<b>_DRV_PARAM_ERR :</b>	Problème d'accès au pilote. Vérifier l'installation de la carte.
<b>_CHIP_ERR :</b>	Problème d'accès au contrôleur.
<b>_USB_ERR :</b>	Erreur de transmission USB.
<b>_BOARD_TIMEOUT:</b>	Pas de réponse du périphérique USB.

## VII.7 Ic\_EnumCards

Cette fonction retourne les informations de configuration et l'état des canaux CAN utilisables. L'application doit fournir à cette fonction un tableau de structures **t\_CardData**. L'application doit veiller à réserver un espace mémoire suffisant pour contenir toutes ces informations.

Les données du premier canal (index = 0) sont placées dans le premier élément du tableau et ainsi de suite. L'index des canaux correspond à la valeur **cno** utilisée par la fonction **Ic\_InitDrv** pour identifier les différents canaux. Le nombre total de canaux est retourné dans la variable **cardcnt**.

```
Short Ic_EnumCards(      unsigned long*   cardcnt,
                        t_CardData*   carddata,
                        unsigned long carddatasz );
```

### Paramètres :

**cardcnt** : Pointeur sur une variable *unsigned long*. Au retour, celle-ci est initialisée avec le nombre de canaux utilisables.

**carddata** : Pointeur sur un tableau de structures **t\_Carddata**. Si le code retour vaut **\_OK**, les champs de N (N = **cardcnt**) structures sont initialisés. La position de chaque canal dans le tableau correspond à son numéro. Voir la fonction **Ic\_InitDrv**.

```
Typedef struct{
    unsigned long IOBaseAddress;      //Adresse IO de base
    unsigned long memoryBaseAddress;  //Adresse mémoire de base
    unsigned long IRQLineNumber;      //Ligne d'interruption
    unsigned long DMACHannelNumber;    //Numéro de canal DMA
    unsigned long cardAlreadyOpen;     //1 = Carte utilisée
    unsigned long boardType;           //Type de la carte (Cf.VI.8)
    char          cardNameString[80];  //Texte descriptif du canal
    unsigned long reserved;            //Réservé
} t_Carddata;
```

**carddatasz** : Taille en octets du tableau pointé par **carddata** et passé à la fonction.

### Code retour :

**\_OK** : Le tableau contient les informations de "**cardcnt**" canaux.  
**\_MEM\_ERR** : Le tableau n'est pas assez grand pour contenir toutes les informations.

Remarque: Dans le cas des périphériques USB, la plupart des informations ne sont pas renseignées. Pour obtenir les informations sur le périphérique il faut préférer la fonction **Ic\_GetDeviceInfo**.

## VII.8 Ic\_ExitDrv

Désactive le contrôleur de protocole et restitue l'environnement initial du canal. Ne pas oublier d'appeler cette requête en fin d'utilisation d'un canal. En effet, tant qu'un canal n'a pas été libéré par cette fonction, d'autres applications ne peuvent pas l'utiliser.

```
short Ic_ExitDrv( HANDLE hdrv );
```

### Paramètres :

**hdrv :** Identificateur du canal retourné par la fonction **Ic\_InitDrv**.

### Code retour :

<b>_OK :</b>	Canal libéré.
<b>_SEQ_ERR :</b>	Séquence invalide.
<b>_INVALID_OP :</b>	La valeur de <b>hdrv</b> est invalide.
<b>_DRV_PARAM_ERR :</b>	Problème d'accès au pilote. Vérifier l'installation de la carte.
<b>_USB_ERR :</b>	Erreur de transmission USB.
<b>_BOARD_TIMEOUT :</b>	Pas d'acquittement du périphérique USB.



## VII.9 Ic\_GetAPIinfo

Retourne les informations concernant l'interface logicielle : version du pilote de la carte utilisée et version de la DLL NSICANEX.

```
Short  Ic_GetAPIinfo(  HANDLE      hdrv,  
                      short*    DLLversion,  
                      short*    DRVversion,  
                      unsigned long* reserved )
```

### Paramètres :

**hdrv :** Identificateur du canal retourné par la fonction **Ic\_InitDrv**.

**DLLversion:** Version de la DLL **NSICANEX.DLL**.

**DRVversion:** Version du pilote de périphérique de la carte utilisée.

**Reserved :** Réservé

*Note:* Les versions sont retournées sous la forme:  $(version\ majeure * 100) + version\ mineure$ .

*Exemple :* 102 s'interprète comme la version 1.02

### Code retour :

<b>_OK :</b>	Versions retournées correctement.
<b>_INVALID_OP :</b>	La valeur de <b>hdrv</b> est invalide.
<b>_PARAM_ERR :</b>	Paramètre invalide.

## VII.10 Ic\_GetBuf

Retourne les données d'un buffer associé à un identificateur déclaré. Cette requête est disponible en mode BUFFER uniquement. Si les données d'un buffer ne sont pas lues, elles sont remplacées lors de chaque nouvel événement par les données les plus récentes.

```
Short Ic_GetBuf( HANDLE          hdrv,
                 unsigned long   ident,
                 t_CANevent*     msg );
```

### Paramètres :

- hdrv :** Identificateur du canal retourné par la fonction **Ic\_InitDrv**.
- ident :** Valeur de l'identificateur déclaré ou **\_CAN\_DUMMY\_ID** pour lire le données du buffer "global" (15 ou 64). Voir la fonction **Ic\_SetRxMask**.
- msg :** Pointeur sur une structure de type **t\_CANevent**. Lorsque le code retour vaut **\_OK**, tous les champs de la structure sont initialisés avec les données du buffer.

### Remarques :

Pour le type **\_CAN\_TX\_RX\_REMOTE**, deux événements sont générés. Un premier événement signalant l'émission de la demande (**eventType** = **\_CAN\_TX\_REMOTE**) puis un second pour indiquer que la trame de réponse a été reçue. (**eventType** = **\_CAN\_RX\_DATA\_REMOTE**). Si le buffer n'est pas scruté assez rapidement, l'événement **\_CAN\_TX\_REMOTE** peut être écrasé par la réception de la réponse.

Pour le type **\_CAN\_RX\_REMOTE**, le champ **dlc** n'est pas renseigné. Le contrôleur CAN i82527 ne fournit pas cette information.

Pour les identificateurs **\_CAN\_RX\_SEG\_DATA** avec un **dlc** supérieur à 6, deux événements sont générés, Un premier signalant l'arrivée de la trame **FIRST\_FRAME** (**eventType** = **\_CAN\_RX\_FF\_DATA**) puis un second pour indiquer que le message segmenté a été reçu correctement ou non (**eventType** = **\_CAN\_RX\_SEG\_DATA**). Si le buffer n'est pas scruté assez rapidement, l'événement **\_CAN\_RX\_FF\_DATA** peut être écrasé par la réponse.

Pour les identificateurs segmentés (**\_CAN\_TX\_SEG\_DATA** et **\_CAN\_RX\_SEG\_DATA**), les événements relatifs aux identificateurs de contrôle de flux sont stockés dans le buffer de réception global (**\_CAN\_DUMMY\_ID**).

### Code retour :

- |                         |   |
|-------------------------|---|
| <b>_OK :</b>            | Données retournées.   |
| <b>_SEQ_ERR :</b>       | Séquence invalide.  |
| <b>_PARAM_ERR :</b>     | Paramètre invalide.   |
| <b>_UNKNOWN_ID :</b>    | Identificateur inconnu.   |
| <b>_EMPTY_BUF :</b>     | Buffer vide.  |
| <b>_INTERFACE_ERR :</b> | Requête non supportée dans le mode d'interface configuré. Vérifier que l'interface n'est pas en mode <b>_FIFO</b> . |
| <b>_INVALID_OP :</b>    | La valeur de <b>hdrv</b> est invalide.  |
| <b>_DRV_PARAM_ERR :</b> | Problème d'accès au pilote. Vérifier l'installation de la carte.  |

## VII.11 Ic\_GetChipInfo

La fonction n'existe que pour les périphériques USB. Elle permet de retourner certaines informations concernant le gestionnaire de protocole CAN utilisé par le canal.

```
Short Ic_GetChipInfo( HANDLE hdrv, t_CANchipInfo* chipInfo );
```

### Paramètres :

**hdrv :** Identificateur du canal retourné par la fonction **Ic\_InitDrv**.

**ChipInfo:** Pointeur sur une structure de type **t\_CANchipInfo**.

### Code retour :

<b>_OK :</b>	Informations sur le contrôleur retournées correctement.
<b>_SEQ_ERR :</b>	Séquence invalide.
<b>_PARAM_ERR :</b>	Paramètre invalide.
<b>_INVALID_OP :</b>	La valeur de <b>hdrv</b> est invalide.
<b>_DRV_PARAM_ERR :</b>	Problème d'accès au pilote. Vérifier l'installation de la carte.
<b>_CHIP_ERR :</b>	Problème d'accès au contrôleur. Pour une carte ISA vérifier la configuration.
<b>_USB_ERR :</b>	Erreur de transmission USB.
<b>_BOARD_TIMEOUT:</b>	Erreur d'acquittement USB

## VII.12 Ic\_GetChipState

Retourne le niveau d'erreur du contrôleur de protocole CAN (Erreur active, erreur passive, BUS OFF). Cet état dépend des erreurs détectées par le contrôleur sur le bus.

```
Short Ic_GetChipState( HANDLE      hdrv,
                      t_CANerr*   CANerr,
                      t_CANstat*  etateveil );
```

### Paramètres :

**hdrv** : Identificateur du canal retourné par la fonction **Ic\_InitDrv**.

**CANerr** : Niveau d'erreur du composant.

- **\_CAN\_ACTIVE\_ERR** : Erreur active, mode normal.
- **\_CAN\_PASSIVE\_ERR** : Erreur passive.
- **\_CAN\_BUS\_OFF** : Composant déconnecté du bus.
- **\_CAN\_UNKNOW** : Etat inconnu.

**etateveil** : Etat du contrôleur de protocole.

- **\_CAN\_WAKE\_UP** : Etat Réveillé.

### Code retour :

**\_OK** : Etat retourné.

**\_SEQ\_ERR** : Séquence invalide.

**\_INVALID\_OP** : La valeur de **hdrv** est invalide.

**\_PARAM\_ERR** : Paramètre invalide.

**\_DRV\_PARAM\_ERR** : Problème d'accès au pilote. Vérifier l'installation de la carte.

**\_CHIP\_ERR** : Problème d'accès au contrôleur. Pour une carte ISA vérifier la configuration.

**\_USB\_ERR** : Erreur de transmission USB.

**\_BOARD\_TIMEOUT** : Erreur d'acquittement USB.

### VII.13 Ic\_GetCount

Retourne les compteurs entretenus par l'interface logicielle. Les compteurs sont mis à zéro lors de l'appel à la fonction **Ic\_StartChip**. Dans le cas du boîtier CAN-USB il est possible d'obtenir des informations plus détaillées en utilisant la fonction **Ic\_GetChipInfo**.

```
Short Ic_GetCount(      HANDLE      hdrv,  
                      t_CANcounter* cpt      );
```

Paramètres :

**hdrv :** Identificateur du canal retourné par la fonction **Ic\_InitDrv**.

**Cpt :** Pointeur sur une structure de type **t\_CANcounter**.

```
Typedef struct{  
    unsigned long tx_ok;           // transmission OK  
    unsigned long rx_ok;           // reception OK  
    unsigned long bus_off;         // erreur de bus off  
    unsigned long passive;         // erreur passive  
    unsigned long stuff_err;       // erreur de codage  
    unsigned long form_err;        // erreur de format  
    unsigned long ack_err;         // erreur d'acquittement  
    unsigned long bit1_err;        // hors champ d'arbitrage  
    unsigned long bit0_err;        // hors champ d'arbitrage  
    unsigned long crc_err;         // erreur de checksum  
} t_CANcounter;
```

Code retour :

<b>_OK :</b>	Valeurs retournées.
<b>_SEQ_ERR :</b>	Séquence invalide.
<b>_INVALID_OP :</b>	La valeur de <b>hdrv</b> est invalide.
<b>_PARAM_ERR :</b>	Paramètre invalide.
<b>_DRV_PARAM_ERR :</b>	Problème d'accès au pilote. Vérifier l'installation de la carte.
<b>_CHIP_ERR :</b>	Problème d'accès au contrôleur. Pour une carte ISA vérifier la configuration.
<b>_USB_ERR :</b>	Erreur de transmission USB.
<b>_BOARD_TIMEOUT:</b>	Erreur d'acquittement USB.

*Note:* Les champs **bit1\_err**, **bit0\_err**, **ack\_err**, **crc\_err**, **stuff\_err**, **form\_err** ne sont pas pris en charge par MUXy ni MUXy box.

## VII.14 Ic\_GetDeviceInfo

Retourne diverses informations sur le périphérique.

```
Short Ic_GetDeviceInfo( HANDLE hdrv,  
                        t_CANdeviceInfo* deviceInfo );
```

Paramètres :

**hdrv :** Identificateur du canal retourné par la fonction **Ic\_InitDrv**.

**DeviceInfo :** Pointeur sur une structure **t\_CANdeviceInfo**.

Code retour :

<b>_OK :</b>	Informations sur le périphérique retournées correctement.
<b>_INVALID_OP :</b>	La valeur de <b>hdrv</b> est invalide.
<b>_PARAM_ERR :</b>	Paramètre invalide.
<b>_DRV_PARAM_ERR :</b>	Problème d'accès au pilote. Vérifier l'installation de la carte.
<b>_USB_ERR :</b>	Erreur de transmission USB.
<b>_BOARD_TIMEOUT:</b>	Erreur d'acquittement USB.

## VII.15 Ic\_GetDiag

Retourne l'état du diagnostic de ligne. Cette fonction est disponible uniquement avec les boîtiers CAN-USB, MUXy box et MUXy2010.

```
Short Ic_GetDiag(      HANDLE      hdrv,  
                    t_CANdiag*    CANdiag    );
```

### Paramètres :

**hdrv :**            Identificateur du canal retourné par la fonction **Ic\_InitDrv**.

**CANdiag:**        Etat du diagnostic de ligne.

- **\_CAN\_NOMINAL :**            Mode nominal.
- **\_CAN\_DEGRADED :**        Communication sur 1 ligne CAN.
- **\_CAN\_MAJOR\_ERR :**        Aucune communication.
- **\_CAN\_UNKNOWN :**        Inconnu.

### Code retour :

<b>_OK :</b>	Etat du diagnostic de ligne retourné avec succès.
<b>_INVALID_OP :</b>	Handle (hdrv) du canal invalide.
<b>_SEQ_ERR :</b>	Séquence invalide.
<b>_PARAM_ERR :</b>	Paramètre invalide.
<b>_BOARD_ERR :</b>	Type de carte ne supportant pas cette fonction.

## VII.16 Ic\_GetEvent

Dépile le premier événement dans la file d'attente des événements réseau. Les événements sont retournés chronologiquement: Premier entré, premier sorti. Cette requête est disponible en mode FIFO uniquement.

```
Short Ic_GetEvent(      HANDLE      hdrv,
                       t_CANevent*  event
                       ) ;
```

### Paramètres :

**hdrv :** Identificateur du canal retourné par la fonction **Ic\_InitDrv**.

**event :** Pointeur sur une structure de type **t\_CANevent**. Lorsque le code retour vaut **\_OK**, tous les champs de la structure **t\_CANevent** sont renseignés avec les données du premier événement de la FIFO.

### Remarques :

Pour le type **\_CAN\_TX\_RX\_REMOTE**, deux événements sont retournés dans la FIFO pour le même identificateur. Un premier événement signalant l'émission de la demande (eventType = **\_CAN\_TX\_REMOTE**) puis un second pour indiquer que la réponse a été reçue (eventType = **\_CAN\_RX\_DATA\_REMOTE**).

Pour le type **\_CAN\_RX\_REMOTE**, le champ **dlc** n'est pas renseigné. Le contrôleur CAN i82527 ne fournit pas l'information.

Seuls les événements des identificateurs déclarés par la fonction **Ic\_InitId** avec le champ **status** = **\_STATUS** sont signalés dans la FIFO.

La taille de la file d'attente est fixe (500 événements). Lorsque la file est pleine et que des événements sont perdus, la valeur spéciale **\_CAN\_LOST\_EVENT** est retournée dans le champ **eventType** pour le dernier événement empilé.

Note : Si l'appel à **Ic\_GetEvent** est effectué après **Ic\_StartChip** il faut penser à vider à FIFO avant de pouvoir recevoir un événement (puisque ceux-ci ne se déclenche que sur un passage de la FIFO de « vide » à « non vide »). Par exemple en utilisant une boucle :

```
while(Ic_GetEvent() != _EMPTY_FIFO) ;
```

### Code retour :

<b>_OK :</b>	Données retournées.
<b>_EMPTY_FIFO :</b>	File d'attente vide.
<b>_SEQ_ERR :</b>	Séquence invalide.
<b>_PARAM_ERR :</b>	Paramètre invalide.
<b>_INTERFACE_ERR :</b>	Requête non supportée par le mode d'interface configuré. Vérifier que l'interface n'est pas en mode <b>_BUFFER</b> .
<b>_INVALID_OP :</b>	La valeur de <b>hdrv</b> est invalide.
<b>_DRV_PARAM_ERR :</b>	Problème d'accès au pilote. Vérifier l'installation de la carte.



## VII.17 Ic\_GetMode

Cette fonction permet d'obtenir l'état de l'interface de ligne CAN.

```
Short Ic_GetMode( HANDLE      hdrv,  
                  unsigned long* mode ) ;
```

### Paramètres :

**hdrv :** Identificateur du canal retourné par la fonction **Ic\_InitDrv**.

**mode :** Pointeur vers une variable qui est initialisée par cette fonction avec l'état de l'interface de ligne si le code d'erreur retourné par cette fonction est **\_OK**. Voici une liste des codes d'état possibles pour le mode du driver de ligne :

- **\_CAN\_NORMAL :** Etat réveillé, état normal.
- **\_CAN\_LD\_SLEEP :** Interface en mode veille.
- **\_CAN\_NETWORK\_WAKEUP :** Interface de ligne en veille, demande de réveil par le réseau détectée.

### Code retour :

<b>_INVALID_OP :</b>	Handle (hdrv) du canal invalide.
<b>_SEQ_ERR :</b>	Séquence invalide
<b>_PARAM_ERR :</b>	Paramètre invalide.
<b>_BOARD_ERR :</b>	Type de carte ne supportant pas cette fonction.

## VII.18 Ic\_InitChip

Initialisation du contrôleur de protocole CAN. Cette fonction définit le débit CAN, la position du point d'échantillonnage dans le bit et le nombre d'échantillons par bit.

```
Short  Ic_InitChip(      HANDLE      hdrv,
                        t_CANBusparams CANparams,
                        t_CANaddressing addressing,
                        t_CANpadding  padding );
```

Paramètres :

**hdrv :** Identificateur du canal retourné par la fonction **Ic\_InitDrv**.

**CANparams:** Structure *t\_CANBusparams* à initialiser en fonction du débit voulu (voir VI.3).

**addressing :** Mode d'adressage (CAN segmenté uniquement).

- **\_DC\_NORMAL :** Adressage normal (selon le type d'identificateur)
- **\_DC\_NORMAL\_FIXED :** Adressage normal fixe (identificateurs sur 29bits)
- **\_DC\_EXTENDED :** Adressage 8bits dans le premier octet des données CAN (identificateurs sur 11bits)

**padding :** Taille fixe des trames (CAN segmenté uniquement):

- **\_DC\_NO\_PADDING :** La taille des trames CAN est adaptée en fonction des données réelles.
- **\_DC\_PADDING :** La taille des trames CAN est fixe (8 octets). Les octets permettant de compléter les trames sont émis avec la valeur FFh.

**Nota :** Le mode **\_DC\_EXTENDED** n'est disponible que pour les produits MUXy et MUxy2010.

Code retour :

<b>_OK :</b>	Configuration réussie
<b>_SEQ_ERR :</b>	Séquence invalide.
<b>_PARAM_ERR :</b>	Un des paramètres est invalide.
<b>_INVALID_OP :</b>	La valeur de <b>hdrv</b> est invalide.
<b>_DRV_PARAM_ERR :</b>	Problème d'accès au pilote. Vérifier l'installation de la carte.
<b>_CHIP_ERR :</b>	Problème d'accès au contrôleur.
<b>_USB_ERR :</b>	Erreur de transmission USB.
<b>_BOARD_TIMEOUT:</b>	Erreur d'acquittement USB

## VII.19 Ic\_InitDrv

Ouverture d'un canal CAN. Cette fonction doit être appelée avant toute autre pour accéder à un canal particulier. Il est conseillé d'utiliser la fonction **Ic\_EnumCards** pour déterminer les canaux disponibles. La valeur retournée dans la variable **hDrv** en échange du numéro de canal **cno** permet d'identifier ce canal pour toutes les opérations suivantes.

Une même application peut ouvrir plusieurs canaux en appelant cette fonction avec des numéros de canal différents. A chaque fois, la valeur retournée dans **hDrv** est différente. Un même canal ne peut pas être ouvert simultanément par deux applications. Un canal ouvert doit toujours être libéré après utilisation par la fonction **Ic\_ExitDrv**.

```
Short Ic_InitDrv( short      cno ,  
                  HANDLE*   hDrv );
```

### Paramètres :

**cno :** Index du canal à ouvrir. Cette valeur correspond à la position du canal dans le tableau retourné par la fonction **Ic\_EnumCards**. Le premier canal correspond à la valeur **cno** = 0.

**hDrv :** Pointeur sur une variable de type HANDLE (void\*). Au retour de la fonction, cette variable est initialisée avec l'identificateur du canal. Cette valeur doit être ensuite passée à toutes les autres fonctions de l'interface pour agir sur ce canal particulier.

### Code retour :

**\_OK :** Canal ouvert.  
**\_INVALID\_OP :** La valeur **cno** est invalide ou le canal est déjà utilisé.  
**\_OPENING\_DRV\_ERR :** Problème d'accès au pilote de la carte. Vérifier l'installation du pilote de périphérique.

## VII.20 Ic\_InitFlowControl

Configure les paramètres de flux et les différentes temporisations pour un identificateur déclaré en `_CAN_TX_SEG_DATA` ou en `_CAN_RX_SEG_DATA`. Il est possible de définir des valeurs uniques pour tous les identificateurs précédemment déclarés (`_CAN_ALL_ID`).

```
Short Ic_InitFlowControl(    HANDLE        hdrv,
                           unsigned long    ident
                           t_CANflowParams  flowParams );
```

Cette fonction est facultative. Si la fonction n'est pas appelée, les valeurs suivantes seront définies dans la norme par défaut:

<b>blockSize :</b>	0 ms.
<b>StMin :</b>	0 ms.
<b>As :</b>	1000 ms.
<b>Ar :</b>	1000 ms.
<b>Bs :</b>	1000 ms.
<b>Br :</b>	0 ms.
<b>Cr :</b>	1000 ms.

### Paramètres :

**hdrv :** Identificateur du canal retourné par la fonction **Ic\_InitDrv**.

**ident :** Identificateur CAN déclaré ou `_CAN_ALL_ID` pour tous les identificateurs précédemment déclarés.

**flowParams :** Structure contenant les paramètres de flux.

### Code retour :

<code>_OK :</code>	Suppression de l'identificateur réussi.
<code>_SEQ_ERR :</code>	Séquence invalide.
<code>_UNKNOWN_ID :</code>	Identificateur non déclaré.
<code>_PARAM_ERR :</code>	Paramètre invalide.
<code>_BOARD_ERR</code>	Type de carte ne supportant pas cette fonction.
<code>_FRAME_TYPE_ERR :</code>	Type de trame invalide (identificateur non segmenté)
<code>_INVALID_OP :</code>	La valeur de <b>hdrv</b> est invalide.
<code>_DRV_PARAM_ERR :</code>	Problème d'accès au pilote. Vérifier l'installation de la carte.
<code>_CHIP_ERR :</code>	Problème d'accès au contrôleur.
<code>_BOARD_TIMEOUT:</code>	Pas de réponse du périphérique USB.
<code>_USB_ERR :</code>	Erreur de transmission invalide

## VII.21 Ic\_InitId

Déclare et initialise un identificateur CAN. En cas de re-déclaration (même identificateur déclaré plusieurs fois), les nouveaux paramètres remplacent les précédents. En mode ANALYSE, les identificateurs non segmentés n'ont pas besoin d'être déclarés, si tel est le cas la fonction retourne `_INTERFACE_ERR`. Dans ce mode, seuls des identificateurs segmentés peuvent être déclarés afin qu'ils soient interprétés comme tels par le périphérique

```
short Ic_InitId( HANDLE      hdrv,  
                 t_CANobj*   msg );
```

### Paramètres :

**hdrv** : Identificateur du canal retourné par la fonction **Ic\_InitDrv**.

**msg** : Pointeur sur une structure de type **t\_CANobj**. Les valeurs des champs doivent être initialisés comme indiqué dans le chapitre "Description des types".

### Code retour :

<code>_OK</code> :	Configuration de l'identificateur réussie.
<code>_PARAM_ERR</code> :	Paramètres invalides dans les champs de la structure <b>msg</b> .
<code>_SEQ_ERR</code> :	Séquence invalide.
<code>_ID_OVERFLOW</code> :	Nombre maximum d'identificateurs déclarés est atteint (63 identificateurs dont 8 identificateurs segmentés maximum)
<code>_INVALID_OP</code> :	La valeur de <b>hdrv</b> est invalide.
<code>_INTERFACE_ERR</code> :	Fonction incompatible avec le mode d'interface.
<code>_DRV_PARAM_ERR</code> :	Problème d'accès au pilote. Vérifier l'installation de la carte.
<code>_BOARD_ERR</code>	Type de carte ne supportant pas cette fonction.
<code>_CHIP_ERR</code> :	Problème d'accès au contrôleur. Pour une carte ISA vérifier la configuration.
<code>_USB_ERR</code> :	Erreur de transmission USB.
<code>_BOARD_TIMEOUT</code> :	Erreur d'acquittement USB

## VII.22 Ic\_InitInterface

Initialise le mode de fonctionnement de l'interface. Voir le chapitre "Mise en œuvre de l'interface" pour plus de détails.

```
Short Ic_InitInterface( HANDLE          hdrv,
                       t_Interface      interface );
```

### Paramètres :

**hdrv :** Identificateur du canal retourné par la fonction **Ic\_InitDrv**.

**interface :** Mode d'interface désiré.

- **\_BUFFER** : Interface en mode BUFFER. Les événements sont enregistrés message par message. Les événements les plus récents écrasent les données non lues.
- **\_FIFO** : Interface en mode FIFO. Les événements sont enregistrés dans l'ordre chronologique de leur arrivée puis dépilés par l'application. La taille de la FIFO est de 500 événements.
- **\_ANALYSE** : Interface en mode ANALYSE. Tous les messages détectés sur le bus sont enregistrés sans être acquittés puis dépilés par l'application. Les transmissions sont impossibles. Les messages déclarés comme segmentés sont stockés comme tels et aucune trame FLOW\_CONTROL n'est envoyée si l'identificateur a été déclaré en \_CAN\_RX\_SEG\_DATA.

### Code retour :

<b>_OK :</b>	Interface correctement initialisée
<b>_SEQ_ERR :</b>	Séquence invalide.
<b>_PARAM_ERR :</b>	Mode d'interface invalide.
<b>_MEM_ERR :</b>	Mémoire insuffisante (mode FIFO ou ANALYSE)
<b>_INVALID_OP :</b>	La valeur de <b>hdrv</b> est invalide.
<b>_BOARD_ERR</b>	Type de carte ne supportant pas cette fonction
<b>_DRV_PARAM_ERR :</b>	Problème d'accès au pilote. Vérifier l'installation de la carte.
<b>_CHIP_ERR :</b>	Problème d'accès au contrôleur. Pour une carte ISA vérifier la configuration.
<b>_USB_ERR :</b>	Erreur de transmission USB.
<b>_BOARD_TIMEOUT:</b>	Erreur d'acquiescement USB

## VII.23 Ic\_InitLineDrv

Cette fonction permet de configurer le comportement de l'interface logicielle CAN lors de l'appel des fonctions de Veille / Réveil. Cette fonction n'est valide que pour la CANPCI, MUXy box et MUXy2010. L'appel à cette fonction n'est pas indispensable pour utiliser l'interface de ligne High Speed. Attention, si aucun appel à **Ic\_InitLineDrv** n'est fait en présence d'une interface Low Speed l'utilisateur ne recevra aucun message.

```

Ic_InitLineDrv(    HANDLE      hdrv ,
                  short      mode ,
                  HANDLE     hEvent ,
                  BOOL       autoWakeup );

```

### Paramètres :

**hdrv** : HANDLE du canal CAN retourné par la fonction **Ic\_InitDrv**. Ce paramètre identifie le canal (carte) concerné par cette fonction.

**mode** : Configure le type d'interface de ligne utilisée (Low-speed ou Hi-speed)

- **\_CAN\_HIGH\_SPEED**
- **\_CAN\_LOW\_SPEED**

**hEvent** : HANDLE d'un événement signalé lorsque l'interface de ligne détecte une demande de réveil par le réseau. Si ce HANDLE est NULL, cela signifie qu'aucun événement ne doit être signalé (comportement par défaut).

*Pour les cartes CANPCI, ce paramètre n'est significatif que si le mode choisi est **\_CAN\_LOW\_SPEED**.*

*Pour MUXy box et MUXy2010, ce paramètre est significatif dans les 2 modes.*

**autoWakeup** : Booléen qui autorise ou non le réveil automatique du driver de ligne lorsqu'une demande de réveil est détectée.

- **TRUE** : Le driver de ligne est automatiquement réveillé par la réception d'un message CAN.
- **FALSE** : La réception de trames CAN ne réveille pas le driver de ligne. Le réveil de l'interface s'effectue uniquement avec la fonction **Ic\_SetMode**.

*Pour les cartes CANPCI, ce paramètre n'est significatif que si le mode choisi est **\_CAN\_LOW\_SPEED**.*

*Pour MUXy box et MUXy2010, ce paramètre est significatif dans les 2 modes*

### Code retour :

<b>_OK</b> :	Interface correctement initialisée
<b>_SEQ_ERR</b> :	Séquence invalide.
<b>_PARAM_ERR</b> :	Paramètre invalide.
<b>_BOARD_ERR</b>	Type de carte ne supportant pas cette fonction
<b>_INVALID_OP</b> :	La valeur de <b>hdrv</b> est invalide.
<b>_DRV_PARAM_ERR</b> :	Problème d'accès au pilote. Vérifier l'installation de la carte.
<b>_CHIP_ERR</b> :	Problème d'accès au contrôleur.
<b>_USB_ERR</b> :	Erreur de transmission USB.
<b>_BOARD_TIMEOUT</b> :	Erreur d'acquittement USB

## VII.24 Ic\_InitPeriod

Déclare un identificateur en émission périodique. Un identificateur peut être déclaré en émission périodique s'il a été auparavant déclaré par la requête **Ic\_InitId** en tant que message non segmenté. Cette requête peut être appelée avant ou après la requête **Ic\_StartChip**. Un appel à cette fonction pour un identificateur déjà déclaré en émission périodique initialise la valeur de la période avec les nouvelles valeurs.

Cette fonction n'est implémentée que pour les périphériques USB.

```
Short Ic_InitPeriod(    HANDLE          hdrv,
                      unsigned long    ident,
                      unsigned short   period,
                      BOOL              autostart,
                      long              repeat )
```

### Paramètres :

**hdrv :** Identificateur du canal retourné par la fonction **Ic\_InitDrv**.

**ident:** Valeur de l'identificateur concerné (16 identificateurs périodiques maximum).  
Pour MUXy, MUXy light et MUXy2010, il est possible de déclarer 32 identificateurs périodiques.  
Pour MUXy box, il est possible de déclarer 16 identificateurs périodiques.

**period:** Période d'émission en milliseconde ( 0ms à 65535ms ). Si la période est nulle l'identificateur est émis dès que la transmission du précédent est terminée.

**autostart:** Mode de démarrage automatique. Si ce paramètre est TRUE, l'émission périodique démarre lors de l'appel à la fonction **Ic\_StartChip** ou immédiatement si **Ic\_StartChip** a déjà été appelée.

**repeat:** Nombre d'émission des données du buffer d'émission (1 à plusieurs millions). Une fois le nombre de répétitions effectué, l'identificateur est désactivé, il faut faire un **Ic\_ActiveId** afin de le réactiver et réémettre les *repeat* trames à nouveau. Il est possible de spécifier **\_CAN\_INFINITE** afin d'émettre en boucle le buffer d'émission (jusqu'à l'appel de **Ic\_StopPeriod** ou **Ic\_KillPeriod**).

### Code retour :

<b>_OK :</b>	Interface correctement initialisée
<b>_SEQ_ERR :</b>	Séquence invalide.
<b>_PARAM_ERR :</b>	Paramètre invalide.
<b>_UNKNOWN_ID:</b>	Identificateur inconnu. Vérifier que le message est bien déclaré.
<b>_PERIOD_OVERFLOW</b>	Trop de périodiques déclarés (32 identificateurs maximum).
<b>_BOARD_ERR</b>	Type de carte ne supportant pas cette fonction .
<b>_INTERFACE_ERR:</b>	Fonction incompatible avec le mode d'interface (l'interface est probablement en mode ANALYSE)
<b>_FRAME_TYPE_ERR</b>	Type de trame invalide (identificateur probablement déclaré en réception ou comme segmenté : <b>_CAN_TX_SEG_DATA</b> ou <b>_CAN_RX_SEG_DATA</b> ).
<b>_INVALID_OP :</b>	La valeur de <b>hdrv</b> est invalide.
<b>_DRV_PARAM_ERR :</b>	Problème d'accès au pilote. Vérifier l'installation de la carte.



_CHIP_ERR :	Problème d'accès au contrôleur.
_USB_ERR :	Erreur de transmission USB.
_BOARD_TIMEOUT:	Erreur d'acquittement USB

## VII.25 Ic\_KillPeriod

Arrête et supprime l'émission périodique d'un message. Cette fonction n'est disponible que pour les périphériques USB.

```
Short Ic_KillPeriod(    HANDLE    hdrv,  
                        unsigned short ident )
```

### Paramètres :

**hdrv :** Identificateur du canal retourné par la fonction **Ic\_InitDrv**.

**ident:** Valeur de l'identificateur concerné.

### Code retour :

<b>_OK :</b>	Interface correctement initialisée
<b>_SEQ_ERR :</b>	Séquence invalide.
<b>_UNKNOWN_ID:</b>	Identificateur inconnu.
<b>_UNKNOWN_PERIOD:</b>	Identificateur non déclaré en périodique.
<b>_INTERFACE_ERR:</b>	Fonction incompatible avec le mode d'interface (l'interface est probablement en mode ANALYSE)
<b>_BOARD_ERR</b>	Type de carte ne supportant pas cette fonction.
<b>_INVALID_OP :</b>	La valeur de hdrv est invalide.
<b>_DRV_PARAM_ERR :</b>	Problème d'accès au pilote. Vérifier l'installation de la carte.
<b>_USB_ERR :</b>	Erreur de transmission USB.
<b>_BOARD_TIMEOUT:</b>	Erreur d'acquittement USB.

## VII.26 Ic\_ReadChip

Retourne le contenu d'une adresse du contrôleur de protocole CAN. Cette fonction n'est pas disponible pour les périphériques USB (MUXy, MUXyLight, MUXybox, MUXy2010).

```
Unsigned char Ic_ReadChip(    HANDLE          hdrv,  
                             unsigned char      ad_reg    ) ;
```

### Paramètres :

**hdrv :**            Identificateur du canal retourné par la fonction **Ic\_InitDrv**.

**ad\_reg :**        Adresse du registre à lire dans le contrôleur.

Attention : L'appel de cette requête en cours d'utilisation "normale" de l'interface peut en perturber le fonctionnement. La lecture de certains registres du composant peut acquitter accidentellement une interruption ou modifier les valeurs de certains registres.

### Code retour :

**valeur** : Valeur lue dans le registre. Pour plus d'informations sur le contenu des registres, se reporter à la documentation du composant CAN.

## VII.27 Ic\_ResetBoard

Initialisation matérielle de la carte CAN. Pour les cartes CANPCI, cette initialisation affecte les deux canaux quel que soit leur état. **Attention, cette fonction ne doit être utilisée qu'en cas de défaillance complète du contrôleur ou de la carte.**

Cette fonction n'est pas disponible pour les périphériques MUXy2010.

```
Short Ic_ResetBoard( HANDLE hdrv );
```

### Paramètres :

**hdrv :**           Identificateur du canal retourné par la fonction **Ic\_InitDrv**.

### Code retour :

<b>_OK :</b>	Initialisation réussie
<b>_SEQ_ERR :</b>	Séquence invalide.
<b>_INVALID_OP :</b>	La valeur de <b>hdrv</b> est invalide.
<b>_DRV_PARAM_ERR :</b>	Problème d'accès au pilote. Vérifier l'installation de la carte.
<b>_USB_ERR :</b>	Erreur de transmission USB.
<b>_BOARD_TIMEOUT :</b>	Erreur d'acquittement USB.
<b>_BOARD_ERR :</b>	Fonction non supportée (MUXy2010).

**Cette fonction est destinée aux cartes PC sans microcontrôleur et équipées d'un gestionnaire de protocole CAN i82527. Il est déconseillé d'utiliser cette fonction sur des périphériques USB.**

## VII.28 Ic\_ResetChip

Ré-initialisation matérielle du contrôleur de protocole CAN.

Attention : Cette fonction est sans effet pour les cartes **CANPCI**. Elle retourne **\_OK** mais le contrôleur n'est pas initialisé. Pour les périphériques USB cette fonction est équivalente à la fonction **Ic\_ResetBoard**.

Cette fonction n'est pas disponible pour les périphériques MUXy2010.

```
Short Ic_ResetChip( HANDLE hdrv );
```

Paramètres :

**hdrv** :           Identificateur du canal retourné par la fonction **Ic\_InitDrv**.

Code retour :

<b>_OK</b> :	Réinitialisation réussie ( <u>sauf</u> CANPCI)
<b>_SEQ_ERR</b> :	Séquence invalide.
<b>_INVALID_OP</b> :	La valeur de <b>hdrv</b> est invalide.
<b>_DRV_PARAM_ERR</b> :	Problème d'accès au pilote. Vérifier l'installation de la carte.
<b>_BOARD_ERR</b> :	Fonction non supportée (MUXy2010).

**Cette fonction est destinée aux cartes PC sans microcontrôleur et équipées d'un gestionnaire de protocole CAN i82527. Il est déconseillé d'utiliser cette fonction sur des périphériques USB.**

## VII.29 Ic\_SetMode

Permet de commander l'état du driver de ligne CAN. Cette fonction n'est valide que pour les boîtiers CAN-USB, les cartes CANPCI, les boîtiers MUXy box et MUXy2010.

```
Short Ic_SetMode( HANDLE          hdrv,  
                  unsigned long    mode );
```

### Paramètres :

**hdrv :** HANDLE du canal CAN retourné par la fonction **Ic\_InitDrv**. Ce paramètre identifie le canal (carte) concerné par cette fonction.

**Mode :** Valeur qui indique l'état désiré de l'interface de ligne.

- **\_CAN\_NETWORK\_WAKEUP :** Demande de réveil.
- **\_CAN\_LD\_SLEEP :** Demande de mise en veille.

### Code retour :

<b>_OK :</b>	Configuration réussie.
<b>_SEQ_ERR :</b>	Séquence invalide.
<b>_PARAM_ERR :</b>	Paramètre invalide.
<b>_BOARD_ERR :</b>	Type de carte ne supportant pas cette fonction.
<b>_INVALID_OP :</b>	La valeur de <b>hdrv</b> est invalide.
<b>_DRV_PARAM_ERR :</b>	Problème d'accès au pilote. Vérifier l'installation de la carte.
<b>_CHIP_ERR :</b>	Problème d'accès au contrôleur. Pour une carte ISA vérifier la configuration.
<b>_USB_ERR :</b>	Erreur de transmission USB.
<b>_BOARD_TIMEOUT :</b>	Erreur d'acquittement USB

### VII.30 Ic\_SetRxMask

Configure et active le buffer de réception global. Le buffer de réception global permet de recevoir des identificateurs non déclarés. Le filtrage des identificateurs reçus dans le buffer est réalisé par l'apposition d'un masque sur les identificateurs reçus. Ce buffer ne peut recevoir que les identificateurs d'un même type (standard ou étendus) et il ne peut pas recevoir les demandes de transmission distantes. Un identificateur déclaré dans un buffer qui appartient également à la famille spécifiée pour le buffer global est reçu dans le buffer déclaré. Cette requête est disponible en mode BUFFER, FIFO et ANALYSE. En mode FIFO et ANALYSE, les messages sont insérés dans la file d'attente. En mode BUFFER, l'application utilise la fonction **Ic\_GetBuf** avec une valeur particulière du paramètre **ident** pour accéder aux données reçues dans ce buffer.

Pour le contrôleur de protocole i82527 la fonction configure le "message object" 15 du contrôleur. Le "message object" 15 est un buffer de réception particulier du composant CAN. Pour les périphériques USB, le buffer de réception global est le buffer 64.

```
Short Ic_SetRxMask(      HANDLE      hdrv,
                        unsigned long code,
                        unsigned long mask,
                        t_CANidentType identType );
```

#### Paramètres :

**hdrv** : Identificateur du canal retourné par la fonction **Ic\_InitDrv**.

**code** : Valeur à comparer avec les identificateurs.

**mask** : Valeur du masque à appliquer sur les identificateurs reçus.

**identType** : Type des identificateurs à recevoir.

- **\_CAN\_STD** : Identificateurs standard
- **\_CAN\_EXT** : Identificateurs étendus
- **\_CAN\_ALL** : Identificateurs standards et étendus

#### Fonctionnement :

Chaque identificateur reçu est comparé bit à bit en fonction du masque avec la valeur du code.

- Bit à 0 dans le masque : Pas de comparaison du bit reçu avec le bit correspondant du code.
- Bit à 1 dans le masque : Comparaison du bit reçu avec le bit correspondant du code.

Tous les bits comparés de l'identificateur reçu doivent être identiques à ceux du code pour que la trame soit acceptée.

#### Exemples :

Pour recevoir tous les identificateurs d'un type défini :

**code** = X (quelconque)

**mask** = 0

Pour ne recevoir aucun identificateur dans le canal de réception :

Ne pas appeler la requête **Ic\_SetRxMask**.

Pour recevoir les identificateurs pairs :

**code** = 0

**mask** = 1

Pour recevoir les identificateurs impairs :

**code** = 1

**mask** = 1

Code retour :

_OK :	Configuration réussie.
_SEQ_ERR :	Séquence invalide.
_PARAM_ERR :	Paramètre invalide.
_INVALID_OP :	La valeur de <b>hdrv</b> est invalide.
_DRV_PARAM_ERR :	Problème d'accès au pilote. Vérifier l'installation de la carte.
_CHIP_ERR :	Problème d'accès au contrôleur. Pour une carte ISA vérifier la configuration.
_USB_ERR :	Erreur de transmission USB.
_BOARD_TIMEOUT:	Erreur d'acquittement USB



### VII.31 Ic\_StartPeriod

Active l'émission périodique d'un message. La période d'émission de ce message doit avoir été définie par la requête **Ic\_InitPeriod**. Cette fonction n'est exécutée que pour les périphériques USB.

```
Short Ic_StartPeriod( HANDLE      hdrv,  
                      unsigned long ident )
```

Paramètres :

**hdrv :**           Identificateur du canal retourné par la fonction **Ic\_InitDrv**.

**ident:**           Valeur de l'identificateur concerné.

Code retour :

<b>_OK :</b>	Interface correctement initialisée
<b>_SEQ_ERR :</b>	Séquence invalide.
<b>_PARAM_ERR :</b>	Identificateur invalide.
<b>_SLEEP_MODE:</b>	Mode veille.
<b>_UNKNOWN_ID:</b>	Identificateur inconnu.
<b>_UNKNOWN_PERIOD:</b>	Identificateur non déclaré en périodique.
<b>_INTERFACE_ERR:</b>	Fonction incompatible avec le mode d'interface (l'interface est probablement en mode ANALYSE)
<b>_BOARD_ERR</b>	Type de carte ne supportant pas cette fonction.
<b>_INVALID_OP :</b>	La valeur de <b>hdrv</b> est invalide.
<b>_DRV_PARAM_ERR :</b>	Problème d'accès au pilote. Vérifier l'installation de la carte.
<b>_USB_ERR :</b>	Erreur de transmission USB.
<b>_BOARD_TIMEOUT:</b>	Erreur d'acquittement USB.

### VII.32 Ic\_StartChip

Démarre le contrôleur CAN du canal en le sortant du mode initialisation. Le contrôleur de protocole est alors capable de recevoir et d'envoyer des trames sur le bus. Le contrôleur doit avoir été entièrement paramétré avant de pouvoir appeler cette fonction (mode de l'interface, débit, déclaration des identificateurs...). Cette fonction permet également de déclencher la procédure de sortie de l'état BUS OFF.

Tous les identificateurs déclarés sont automatiquement validés (mode BUFFER et FIFO). Les identificateurs déclarés en réception (\_CAN\_RX\_DATA, \_CAN\_RX\_SEG\_DATA et \_CAN\_RX\_REMOTE) sont activés. Les identificateurs déclarés en mise à jour de données pour réponse automatique (\_CAN\_TX\_AUTO\_REMOTE) sont automatiquement réactivés à chaque émission. Les trames émises (\_CAN\_TX\_DATA, \_CAN\_TX\_SEG\_DATA et \_CAN\_TX\_RX\_REMOTE) doivent être activées par l'application avec la fonction **Ic\_ActiveId**.

```
Short Ic_StartChip( HANDLE hdrv );
```

#### Paramètres :

**hdrv :** Identificateur du canal retourné par la fonction **Ic\_InitDrv**.

#### Code retour :

<b>_OK :</b>	Contrôleur CAN démarré.
<b>_SEQ_ERR :</b>	Séquence invalide. Vérifier que la séquence d'initialisation a été respectée.
<b>_INVALID_OP :</b>	La valeur de <b>hdrv</b> est invalide.
<b>_DRV_PARAM_ERR :</b>	Problème d'accès au pilote. Vérifier l'installation de la carte.
<b>_CHIP_ERR :</b>	Problème d'accès au contrôleur. Pour une carte ISA vérifier la configuration.
<b>_USB_ERR :</b>	Erreur de transmission USB.
<b>_BOARD_TIMEOUT :</b>	Erreur d'acquittement USB.

### VII.33 Ic\_StopChip

Désactive le contrôleur CAN en le plaçant en mode initialisation. Le contrôleur n'est plus connecté au bus. Toutes les valeurs des registres du composant sont préservées. La configuration initialement paramétrée peut redémarrer à tout moment par un appel à **Ic\_StartChip**. Pour modifier la configuration initiale, appeler **Ic\_ExitDrv**, puis reprendre la séquence à partir de la fonction **Ic\_InitDrv** incluse.

```
Short Ic_StopChip( HANDLE hdrv );
```

Paramètres :

**hdrv** :           Identificateur du canal retourné par la fonction **Ic\_InitDrv**.

Code retour :

<b>_OK</b> :	Composant arrêté.
<b>_SEQ_ERR</b> :	Séquence invalide.
<b>_INVALID_OP</b> :	La valeur de <b>hdrv</b> est invalide.
<b>_DRV_PARAM_ERR</b> :	Problème d'accès au pilote. Vérifier l'installation de la carte.
<b>_USB_ERR</b> :	Erreur de transmission USB.
<b>_BOARD_TIMEOUT</b> :	Erreur d'acquittement USB.

### VII.34 Ic\_StopPeriod

Suspend l'émission périodique d'un identificateur périodique. Cette fonction n'est exécutée que pour les périphériques USB.

```
Short Ic_StartPeriod( HANDLE      hdrv,  
                     unsigned long ident )
```

Paramètres :

**hdrv :** Identificateur du canal retourné par la fonction **Ic\_InitDrv**.

**ident:** Valeur de l'identificateur concerné.

Code retour :

<b>_OK :</b>	Interface correctement initialisée
<b>_SEQ_ERR :</b>	Séquence invalide.
<b>_PARAM_ERR :</b>	Paramètre invalide.
<b>_SLEEP_MODE:</b>	Mode veille.
<b>_UNKNOWN_ID:</b>	Identificateur inconnu.
<b>_UNKNOWN_PERIOD:</b>	Identificateur non déclaré en périodique.
<b>_INTERFACE_ERR:</b>	Fonction incompatible avec le mode d'interface (l'interface est probablement en mode ANALYSE)
<b>_BOARD_ERR</b>	Type de carte ne supportant pas cette fonction.
<b>_INVALID_OP :</b>	La valeur de <b>hdrv</b> est invalide.
<b>_DRV_PARAM_ERR :</b>	Problème d'accès au pilote. Vérifier l'installation de la carte.
<b>_USB_ERR :</b>	Erreur de transmission USB.
<b>_BOARD_TIMEOUT:</b>	Erreur d'acquittement USB.

### VII.35 Ic\_TxMsg

Met à jour la zone **data** d'un identificateur déclaré en transmission de (`_CAN_TX_DATA`, `_CAN_TX_SEG_DATA`, `_CAN_TX_AUTO_REMOTE`) puis active l'identificateur (transmission du message ou autorisation de la réponse automatique). Cette fonction est équivalente à **Ic\_WriteData** suivie de **Ic\_ActiveId**. Si la ou les transmissions précédentes ne sont pas terminées ou n'ont pu avoir lieu – problème d'acquittement sur le bus par exemple – le code d'erreur `_BUF_OCC` est retourné. Dans ce cas, les données ne sont pas mises à jour. Pour MUXy, plusieurs transmissions peuvent être bufferisées avant de retourner le code `_BUF_OCC`. Pour toutes les cartes, si le composant est en état BUS OFF, le code `_CHIP_ERR` est retourné.

L'appel de cette fonction sur une trame périodique ne déclenchera pas l'envoi d'une seule trame, mais bien de la séquence périodique complète.

```
Short Ic_TxMsg( HANDLE          hdrv,
                unsigned long   ident,
                unsigned short   dlc,
                unsigned char*   data );
```

#### Paramètres :

**hdrv** : Identificateur du canal retourné par la fonction **Ic\_InitDrv**.

**ident** : Valeur de l'identificateur déclaré.

**dlc** : Nombre d'octets de données à mettre à jour et à activer.

- jusqu'à **8** pour `_CAN_TX_DATA`, `_CAN_TX_AUTO_REMOTE`.
- jusqu'à **4095** pour `_CAN_TX_SEG_DATA`.

**\*data** : Adresse d'un tableau de caractères contenant les données à mettre à jour.

#### Code retour :

<code>_OK</code> :	Données mises à jour et trame activée
<code>_PARAM_ERR</code> :	Paramètre invalide.
<code>_SEQ_ERR</code> :	Séquence invalide. Le contrôleur doit être démarré ( <b>Ic_StartChip</b> ).
<code>_UNKNOWN_ID</code> :	Identificateur inconnu.
<code>_FRAME_TYPE_ERR</code> :	Type de trame invalide (identificateur probablement déclaré en réception).
<code>_BUF_OCCUPIED</code> :	Transmission en cours, requête non prise en compte. Vérifier que la station émettrice n'est pas seule sur le bus.
<code>_INTERFACE_ERR</code> :	Fonction incompatible avec le mode d'interface (l'interface est probablement en mode ANALYSE)
<code>_INVALID_OP</code> :	La valeur de <b>hdrv</b> est invalide.
<code>_SLEEP_MODE</code> :	Interface de ligne en mode veille.
<code>_DRV_PARAM_ERR</code> :	Problème d'accès au pilote. Vérifier l'installation de la carte.
<code>_CHIP_ERR</code> :	Problème d'accès au contrôleur ou contrôleur en état BUS OFF. Utiliser la fonction <b>Ic_GetChipState</b> pour déterminer l'état du contrôleur du protocole.
<code>_USB_ERR</code> :	Erreur de transmission USB.
<code>_BOARD_TIMEOUT</code> :	Erreur d'acquittement USB.



### VII.36 Ic\_WriteChip

Ecrit une valeur à une adresse du contrôleur de protocole CAN. Cette fonction n'est pas disponible pour les périphériques USB.

```
Void Ic_WriteChip(      HANDLE      hdrv,  
                     unsigned char ad_reg,  
                     unsigned char reg );
```

Paramètres :

**hdrv :**           Identificateur du canal retourné par la fonction **Ic\_InitDrv**.

**ad\_reg :**       Adresse du registre à écrire .

**reg :**           Valeur à écrire. Pour plus d'information sur le contenu des registres, se reporter à la documentation du composant CAN.

Attention :

L'appel de cette requête en cours d'utilisation "normale" de l'interface peut en perturber le fonctionnement.

### VII.37 Ic\_WriteData

Met à jour la zone de données d'un identificateur déclaré en transmission (`_CAN_TX_DATA` ou `_CAN_TX_AUTO_REMOTE`). Cette fonction n'active pas la transmission des identificateurs déclarés avec le type `_CAN_TX_DATA`. Les identificateurs déclarés avec le type `_CAN_TX_AUTO_REMOTE` sont émis automatiquement à chaque interrogation distante avec les données les plus récentes. Si la trame est en cours de transmission, le code `_BUF_OCCUPIED` est retourné.

```
Short  Ic_WriteData(    HANDLE      hdrv,
                        unsigned long ident,
                        unsigned short dlc,
                        unsigned char* data );
```

#### Paramètres :

- hdrv** : Identificateur du canal retourné par la fonction **Ic\_InitDrv**.
- ident** : Valeur de l'identificateur déclaré.
- dlc** : Taille des données à mettre à jour.
- jusqu'à **8** pour `_CAN_TX_DATA`, `_CAN_TX_AUTO_REMOTE`.
  - jusqu'à **4095** pour `_CAN_TX_SEG_DATA`.
- \*data** : Pointeur sur un tableau d'octets contenant les données à mettre à jour.

#### Code retour :

- \_OK** : Données de l'identificateur mises à jour.
- \_SEQ\_ERR** : Séquence invalide.
- \_PARAM\_ERR** : Paramètre invalide.
- \_UNKNOWN\_ID** : Identificateur non déclaré.
- \_BUF\_OCCUPIED** : Une transmission est en cours.
- \_FRAME\_TYPE\_ERR** : Type de trame invalide. L'identificateur est probablement déclaré en réception.
- \_INTERFACE\_ERR** : Fonction incompatible avec le mode d'interface (l'interface est probablement en mode ANALYSE)
- \_INVALID\_OP** : La valeur de **hdrv** est invalide.
- \_DRV\_PARAM\_ERR** : Problème d'accès au pilote. Vérifier l'installation de la carte.
- \_CHIP\_ERR** : Problème d'accès au contrôleur ou contrôleur en état BUS OFF. Utiliser la fonction **Ic\_GetChipState** pour déterminer l'état du contrôleur du protocole.
- \_USB\_ERR** : Erreur de transmission USB.
- \_BOARD\_TIMEOUT** : Erreur d'acquittement USB.



## VII.38 Ic\_WritePattern

Met à jour les données d'un identificateur périodique afin de pouvoir émettre des "motifs périodiques". Cette fonction est disponible uniquement pour les périphériques USB. La taille des données écrites (*size*) peut être supérieure à 8 (256 octets maximum), dans ce cas le périphérique se charge de découper le motif en trame CAN de *dlc* octets.

La fonction ne peut être appelée que si l'émission périodique n'est pas en cours. Dans le cas de MUXy et MUXy2010, il est possible de changer les données d'un identificateur périodique sans arrêter l'émission périodique, en utilisant la fonction **Ic\_WriteData** (limitée à 8 octets).

```
Short  Ic_WritePattern( HANDLE          hdrv,
                        unsigned long    ident,
                        unsigned short    size,
                        unsigned char*    data,
                        unsigned char     dlc );
```

### Paramètres :

***hdrv*** : Identificateur du canal retourné par la fonction **Ic\_InitDrv**.

***ident*** : Valeur de l'identificateur déclaré.

***size*** : Taille des données à mettre à jour.  
 • de **1** à **256** octets

***\*data*** : Pointeur sur un tableau d'octets contenant les données à mettre à jour.

***dlc*** : Taille des trames CAN émises sur le bus pour l'émission du motif.  
 • de **1** à **8** octets

### Code retour :

**\_OK** : Données de l'identificateur mises à jour.

**\_SEQ\_ERR** : Séquence invalide.

**\_PARAM\_ERR** : Paramètre invalide.

**\_UNKNOWN\_ID** : Identificateur non déclaré.

**\_UNKNWON\_PERIOD** : Identificateur non périodique

**\_INTERFACE\_ERR** : Fonction incompatible avec le mode d'interface (l'interface est probablement en mode ANALYSE)

**\_FRAME\_TYPE\_ERR** : Type de trame invalide. L'identificateur est probablement déclaré en réception.

**\_INVALID\_OP** : La valeur de **hdrv** est invalide.

**\_DRV\_PARAM\_ERR** : Problème d'accès au pilote. Vérifier l'installation de la carte.

**\_CHIP\_ERR** : Problème d'accès au contrôleur ou contrôleur en état BUS OFF. Utiliser la fonction **Ic\_GetChipState** pour déterminer l'état du contrôleur du protocole.

**\_USB\_ERR** : Erreur de transmission USB.

**\_BOARD\_TIMEOUT** : Erreur d'acquiescement USB.

## VIII. Trucs et astuces

### VIII.1 Détecter et sortir de l'état BUS OFF

Le passage du contrôleur dans l'état BUS OFF est signalé à l'application de 3 manières différentes :

1. Un événement peut être signalé lorsque le composant passe en mode BUS OFF. Cet événement est configuré par la fonction **Ic\_ConfigEvent**.
2. En scrutant régulièrement l'état du composant avec la fonction **Ic\_GetChipState**.
3. Par le code retour des fonctions **Ic\_ActiveId**, **Ic\_WriteData** et **Ic\_TxMsg**. Si l'une de ces fonctions retourne le code **\_CHIP\_ERR**, il est probable que le composant est passé dans l'état BUS OFF. Vérifier l'état avec la fonction **Ic\_GetChipState**.

L'interface logicielle signale le passage du contrôleur dans l'état BUS OFF (déconnexion logique du bus suite à des erreurs de communication) mais elle ne prend aucune initiative quant au traitement de cette erreur. Pour sortir le composant de cet état, l'application dispose de deux possibilités :

#### Solution 1 (recommandée) :

1. Arrêter le composant avec la fonction: **Ic\_StopChip**.
2. Si l'application veut connaître la séquence qui a provoqué le passage en BUS OFF, elle doit dépiler les événements à ce moment. En effet, la fonction **Ic\_StartChip** vide la FIFO ou les BUFFERS en fonction du mode.
3. Redémarrer le composant avec la fonction **Ic\_StartChip** lance la procédure de sortie de l'état BUS OFF. Cette procédure remet à zéro les compteurs d'erreurs internes (transmission et réception) du contrôleur. Lorsque le contrôleur a détecté 128 paquets de 11 bits récessifs consécutifs, il sort de l'état BUS OFF.

Remarque : Les compteurs retournés par la fonction **Ic\_GetCount** ne sont pas affectés par cette opération.

4. Eventuellement appeler **Ic\_GetChipState** pour vérifier l'état du composant, puis reprendre le fonctionnement "normal".

#### Solution 2 :

1. Fermer le canal avec la fonction **Ic\_ExitDrv**.

2. Reprendre tous les traitements à partir de **Ic\_InitDrv** inclus. L'appel à **Ic\_StartChip** lance la procédure de sortie du BUS OFF décrite ci-dessus.

## VIII.2 Utiliser plus de 14 identificateurs (i82527)

Le contrôleur de protocole Intel 82527 dispose de 14 "message objects" permettant chacun d'émettre et de recevoir un identificateur particulier et d'un 15<sup>ème</sup> buffer spécial qui peut recevoir des trames de données pour un groupe d'identificateurs. Pour certaines applications, ce nombre peut être trop limité. Voici quelques astuces pour contourner ces limitations:

### Pour les réceptions de données :

Utiliser la fonction **Ic\_SetRxMask** qui permet de créer un groupe de réception dans le quinzième buffer du contrôleur i82527. Tous les identificateurs reçus appartenant au groupe déclaré sont transmis à l'application soit dans la FIFO, soit dans le buffer 15 accessible avec la valeur **\_CAN\_DUMMY\_ID** en mode BUFFER.

### Pour les émissions (données et demandes de transmission) :

Déclarer un identificateur de chaque type de message désiré (type de trame et type d'identificateur). Avant chaque activation, utiliser la fonction **Ic\_ChangeId** pour déclarer la valeur de l'identificateur à utiliser puis l'activer.

Pour les réponses automatiques (**\_CAN\_TX\_AUTO\_REMOTE**) et les réceptions de demandes de transmission (**\_CAN\_RX\_REMOTE**), l'utilisation d'un buffer pour chaque identificateur est nécessaire. On peut éventuellement utiliser un deuxième canal dans la même application que l'on connecte sur le même réseau CAN.

## VIII.3 Dépanner une application

### La fonction **Ic\_InitDrv** ne fonctionne pas :

Il est impossible d'obtenir le HANDLE d'un canal :

- Appeler la fonction **Ic\_EnumCards** pour connaître le nombre de canaux disponibles et vérifier que le canal désiré (**cno**) n'est pas déjà utilisé.
- Vérifier que la carte est correctement installée dans le *Gestionnaire des Périphériques*.

### Pas de réception ou de compte-rendu de fin d'échange :

Les codes retournés par les requêtes **Ic\_GetEvent** ou **Ic\_GetBuf** valent toujours **\_EMPTY\_FIFO** ou **\_EMPTY\_BUF**, vérifier :

- La présence d'au moins une autre station CAN pour l'acquittement des messages.
- La connexion au bus CAN : Branchements, résistances de terminaison.
- La correspondance du débit programmé avec celui des autres stations.
- En mode FIFO, vérifier également que lors de la déclaration d'un identificateur (**Ic\_InitId**), le champ **statusRq** dans la structure **t\_CANObj** vaut bien **\_STATUS**

### Le code **CHIP\_ERR** est retourné

- Par la fonction **Ic\_InitChip** : Appeler la fonction **Ic\_ResetBoard** puis reprendre la fonction **Ic\_InitChip**.
- Par les fonctions **Ic\_ActiveId**, **Ic\_TxMsg**, **Ic\_WriteData** : Appeler la fonction **Ic\_GetChipState** et vérifier que le contrôleur est en état BUS OFF. Si oui, appliquer la procédure décrite précédemment. Sinon, appeler la fonction **Ic\_ResetBoard**.

## VIII.4 Utiliser un compilateur non Microsoft

### VIII.4.1 Linkage de l'application avec la DLL NSICANEX

Le fichier **NSICANEX.LIB** permet à l'outil de développement de lier l'application avec le fichier **NSICANEX.DLL**. Ce fichier est spécifique au compilateur Microsoft qui a servi à réaliser le fichier DLL. Il n'est pas compatible avec d'autres environnements de développement. Voici différentes solutions à ce problème :

1. Utiliser le fichier DEF ou DLL à la place du fichier LIB dans le projet. Ceci est possible avec les outils C++ Borland ou Labview.
2. Régénérer le fichier LIB à partir du fichier DLL ou du fichier DEF grâce à un utilitaire livré avec l'outil de développement utilisé.

**Exemple :** Les outils Borland C++ sont livrés avec un programme IMPLIB.EXE qui permet de générer un fichier **LIB** au format Borland à partir du fichier **NSICANEX.DEF**.

3. Obtenir les points d'entrée des fonctions de la DLL par la fonction **GetProcAddress**. Celle-ci retourne un pointeur sur chaque fonction de la DLL à partir de son nom. Dans ce cas, le fichier CANPROEX.H doit être modifié.

### VIII.4.2 Alignement des structures de données

Les directives de compilation d'alignement des structures de données et de convention d'appel des fonctions sont spécifiques aux compilateurs Microsoft. Il faut les modifier en fonction du compilateur utilisé.

- Alignement des structures sur 2 octets dans le fichier **CANDEFEX.H**. Si cet alignement n'est pas respecté, l'interface est inutilisable. Le choix de cet alignement peut être spécifié par une option générale du compilateur ou par une directive de compilation dans le code (recommandé) :

Microsoft :

```
#pragma pack(push,2)
// Définition des structures
#pragma pack(pop)
```

Borland :

```
#pragma option -a2
// Définition des structures
#pragma option -a-
```

*Note :* Afin d'aligner les types énumérés (enum) sur des *int* il est aussi nécessaire sous Borland d'ajouter la directive de compilation suivante :

```
#pragma option -b
```

- Déclaration des fonctions de l'interface comme étant importées d'une DLL avec la convention d'appel Microsoft standard (**stdcall**). Cette déclaration est faite par le symbole **\_CANAPI** défini dans le fichier **CANPCEX.H** :

Microsoft

```
#define _CANAPI __declspec(dllimport) __stdcall
```

Borland :

```
#define _CANAPI _import _stdcall
```

## **IX. Installation de l'interface logicielle**

L'installation de l'interface logicielle permet au système de configurer la ou les cartes CAN à utiliser et de recopier les fichiers du pilote et de la DLL sur le disque dur du PC. L'installation doit être effectuée dans les deux cas suivants :

1. **Première installation de l'interface sur un PC.** Si la carte CAN est déjà utilisée par une autre application ou une version précédente de l'interface logicielle, celle-ci doit être désinstallée avant de poursuivre.
2. **Installation d'une nouvelle carte CAN.** L'interface logicielle est capable de gérer plusieurs cartes de types différents dans un même PC.

Cette annexe permet d'effectuer les installations des cartes suivantes :

- **CAN PCMCIA,**
- **CANPCI.**
- **MUXy, MUXy box, MUXylight, MUXy2010**

pour les systèmes suivants :

- **Windows 2000,**
- **Windows XP 32 bits,**
- **Windows VISTA 32 bits**

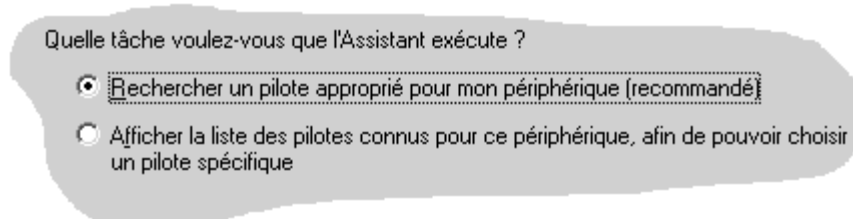
Les chemins des répertoires indiqués dans la procédure d'installation de chaque carte pour chaque système sont relatifs au répertoire **\Pilotes PC – PC drivers** du CD-ROM.

## IX.1 Installation pour une carte CAN PCMCIA

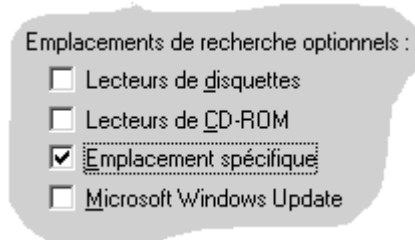
La procédure d'installation des cartes PCMCIA décrite ci-dessous n'a besoin d'être réalisée que lors de la première insertion d'une carte CANPCMCIA dans le PC. Par la suite, le système reconnaît et configure automatiquement la carte PCMCIA lors de chaque nouvelle insertion. La procédure dépend du système utilisé. Se reporter au paragraphe correspondant :

### IX.1.1 Windows 2000

- Fermer toutes les applications en cours...
  - Insérer la carte CANPCMCIA dans un emplacement PCMCIA libre.
  - Windows 2000 indique qu'il a détecté un nouveau périphérique et lance l'assistant d'installation...
- Insérer le CD-ROM « CD Livraison NSI » dans le lecteur du PC et cliquer sur **Suivant**.
  - Sélectionner l'option "*Rechercher un pilote approprié...*" puis cliquer sur **Suivant**.



- Cocher **uniquement** la ligne "*Emplacement spécifique*" puis cliquer sur **Suivant**.



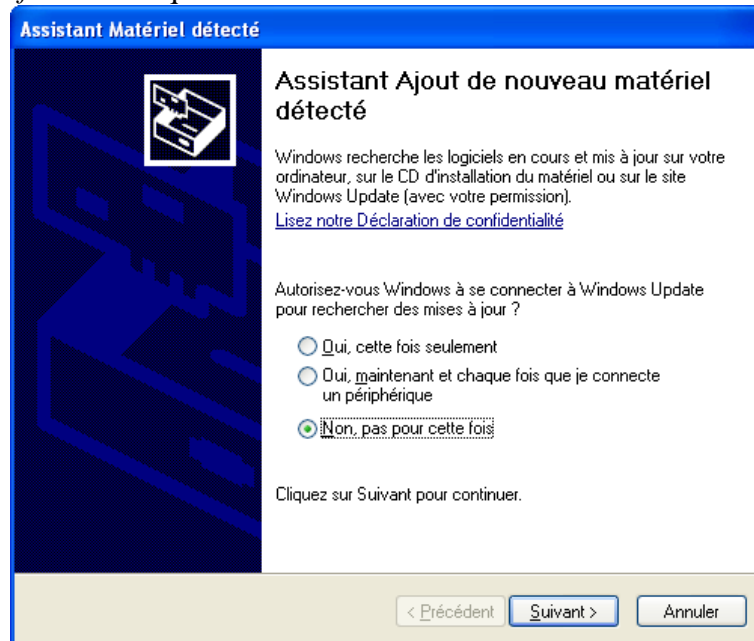
- Cliquer sur le bouton **Parcourir** puis sélectionner le répertoire **Pilotes\_PC-PC\_Drivers\Interfaces\_CAN\_LIN\_K**.
  - Cliquer sur **Ok**.
  - Windows 2000 recherche le pilote et indique qu'il a trouvé un pilote pour le périphérique *PCMCIA Device*. Cliquer sur **Suivant**.
- Windows 2000 recopie les fichiers du pilote sur le disque dur.
- Lorsque la copie des fichiers est finie, cliquer sur **Terminer**.
  - **L'interface logicielle est prête à être utilisée.**

- Vérifier que la carte fonctionne en démarrant le programme **CANEXTEST.EXE** qui est dans le répertoire **Logiciels\_PC-PC\_Software\Ex\_PC\_Software\_CAN\Test** du CD-ROM. En cas de problème, consulter le paragraphe «Dépannage de l'installation» suivant.

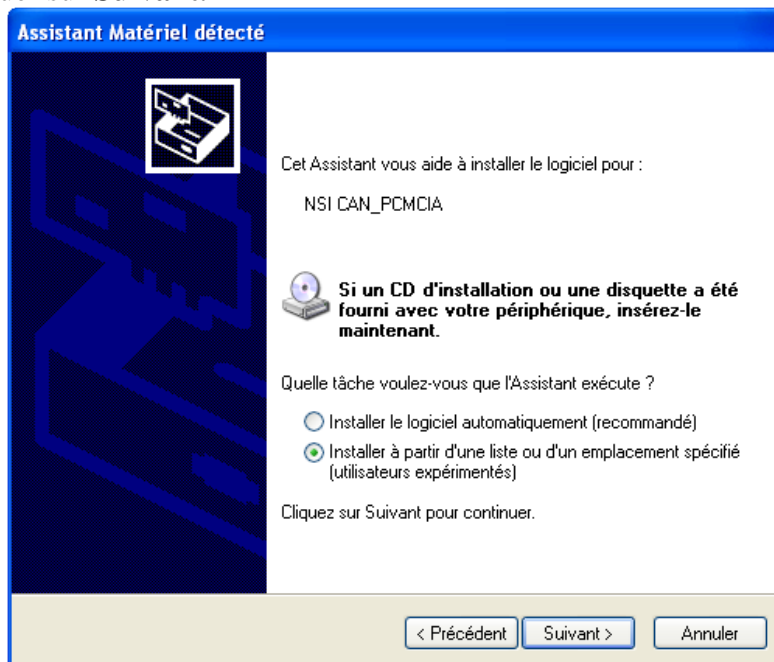


### IX.1.2 Windows XP

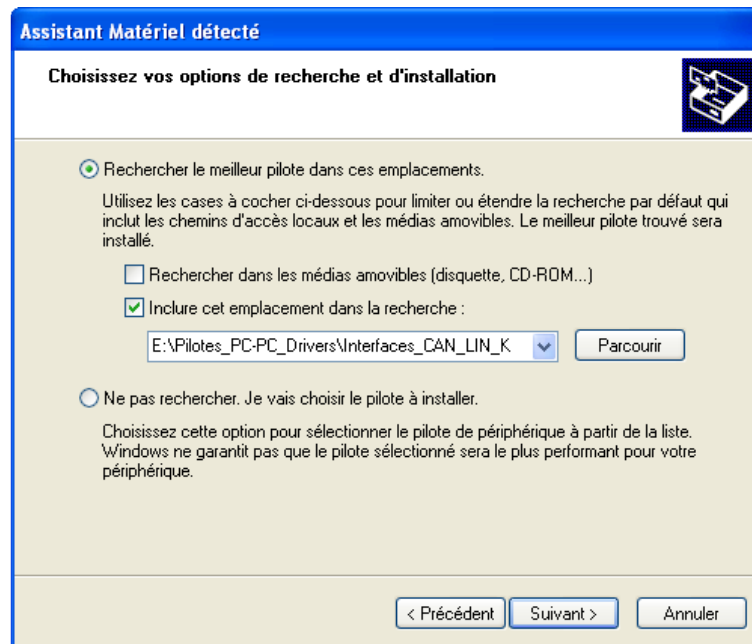
- Fermer toutes les applications en cours...
  - Insérer la carte CANPCMCIA dans un emplacement PCMCIA libre.
  - Windows XP indique qu'il a détecté un nouveau périphérique et lance l'assistant d'installation...
- Insérer le CD-ROM « CD Livraison NSI » dans le lecteur du PC, cochez l'option « *Non, pas pour cette fois* » et cliquer sur **Suivant**.



- Sélectionner l'option "Installer à partir d'une liste ou d'un emplacement spécifié..." puis cliquer sur **Suivant**.



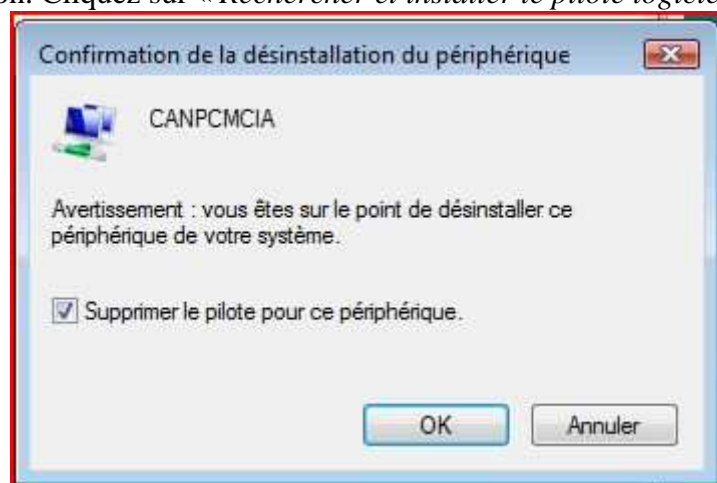
- Cocher **uniquement** la ligne "Inclure cet emplacement dans la recherche" puis cliquer sur **Parcourir** et sélectionner le répertoire **Pilotes\_PC-PC\_Drivers\Interfaces\_CAN\_LIN\_K**.



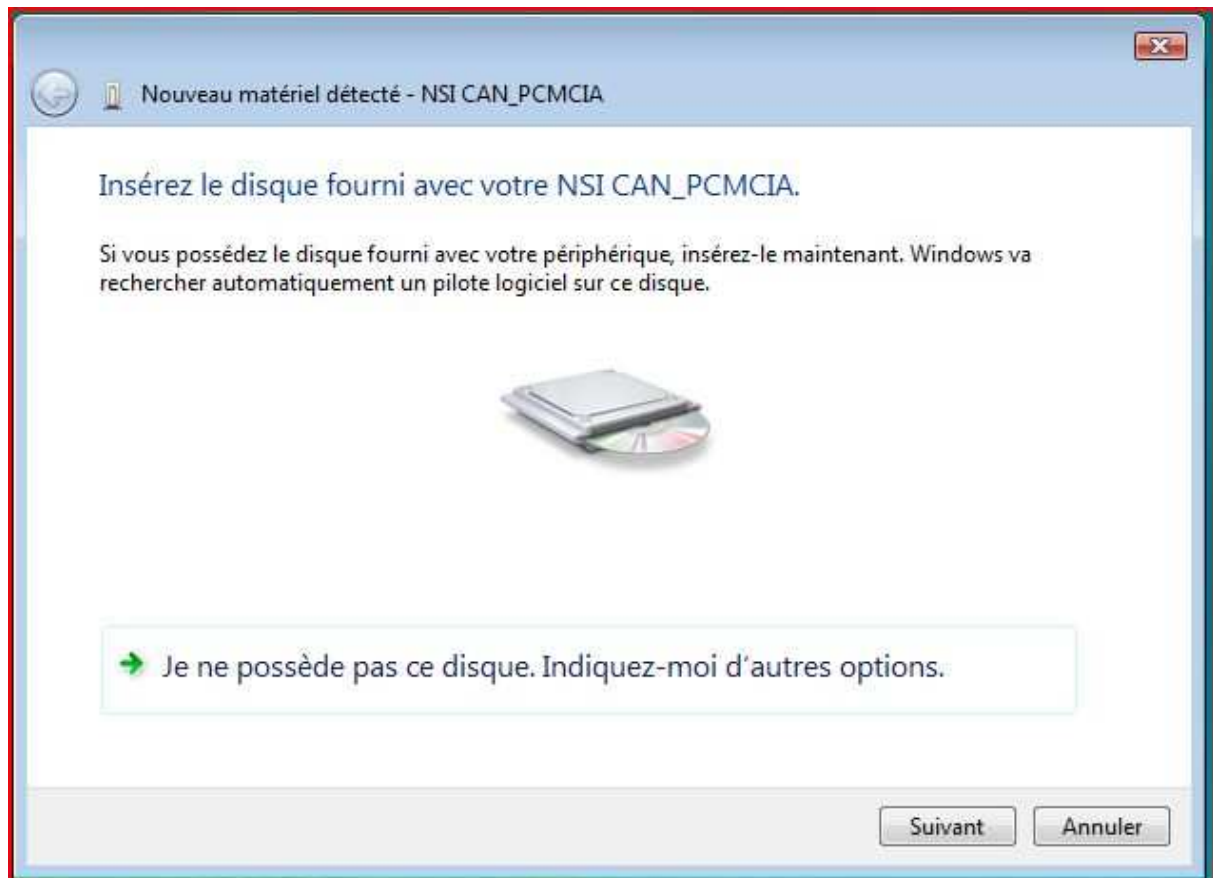
- Cliquer sur **Suivant**.
- Windows XP recherche et installe le pilote pour le périphérique *PCMCIA Device*.
- Lorsque la copie des fichiers est finie, cliquer sur **Terminer**.
- **L'interface logicielle est prête à être utilisée.**
- Vérifier que la carte fonctionne en démarrant le programme **CANEXTEST.EXE** qui est dans le répertoire **Logiciels\_PC-PC\_Software\Ex\_PC\_Software\_CAN\Test** du CD-ROM. En cas de problème, consulter le paragraphe «Dépannage de l'installation» suivant.

### IX.1.3 Windows VISTA

- Fermer toutes les applications en cours...
  - Insérer la carte CANPCMCIA dans un emplacement PCMCIA libre.
  - Windows VISTA indique qu'il a détecté un nouveau périphérique et lance l'assistant d'installation. Cliquez sur « *Rechercher et installer le pilote logiciel* ».



- Insérer le CD-ROM « CD Livraison NSI » dans le lecteur du PC et cliquez sur **Suivant**.



- Windows VISTA recherche le pilote adéquat sur le CD. Cliquez sur « *installer ce pilote quand même* » pour compléter l'installation.



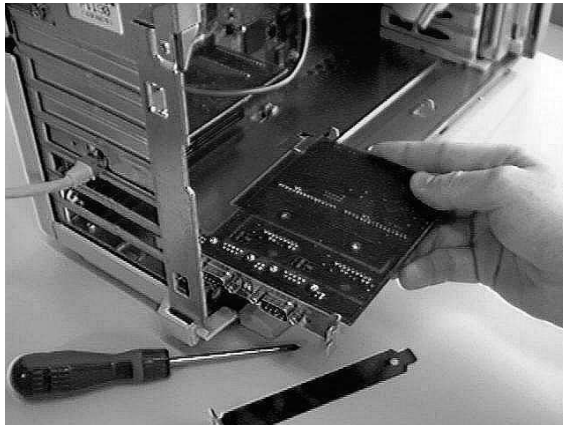
- **L'interface logicielle est prête à être utilisée.**

- Vérifier que la carte fonctionne en démarrant le programme **CANEXTEST.EXE** qui est dans le répertoire **Logiciels\_PC-PC\_Software\Ex\_PC\_Software\_CAN\Test** du CD-ROM. En cas de problème, consulter le paragraphe «Dépannage de l'installation» suivant.

**AJOUTER POUR WINDOWS 7 (il faut un pc portable en windows 7)**

## IX.2 Installation d'une carte CANPCI

Pour installer l'interface logicielle pour la première fois avec une carte CANPCI, il faut dans un premier temps insérer la carte dans un emplacement PCI du PC. Pour cela, suivre la procédure décrite dans ce paragraphe. Par la suite, l'installation dépend du système d'exploitation utilisé. Se reporter aux paragraphes correspondants.



- Arrêter le système et **débrancher le cordon du secteur**. En effet, sur certains PC, le bus PCI dispose d'une alimentation auxiliaire même lorsque le PC est éteint.
- Ouvrir le capot du PC et insérer la carte CANPCI dans un emplacement libre comme illustré sur l'image ci-contre.
- Rebrancher et redémarrer le PC.

- Suivre la procédure d'installation du pilote en fonction du système d'exploitation décrite dans les paragraphes suivants :

### IX.2.1 Windows 2000

- Arrêter le PC et installer la carte comme il est expliqué au début de ce chapitre.
- Après le démarrage du système, Windows 2000 indique qu'il a détecté un nouveau périphérique et lance l'assistant d'installation...
- Insérer le CD-ROM « CD Livraison NSI » dans le lecteur du PC et cliquer sur **Suivant**.
  - Sélectionner l'option "*Rechercher un pilote approprié...*" puis cliquer sur **Suivant**.

Quelle tâche voulez-vous que l'Assistant exécute ?

- ☒ Rechercher un pilote approprié pour mon périphérique (recommandé)
- ☐ Afficher la liste des pilotes connus pour ce périphérique, afin de pouvoir choisir un pilote spécifique

- Cocher uniquement la ligne "*Emplacement spécifique*" puis cliquer sur **Suivant**.

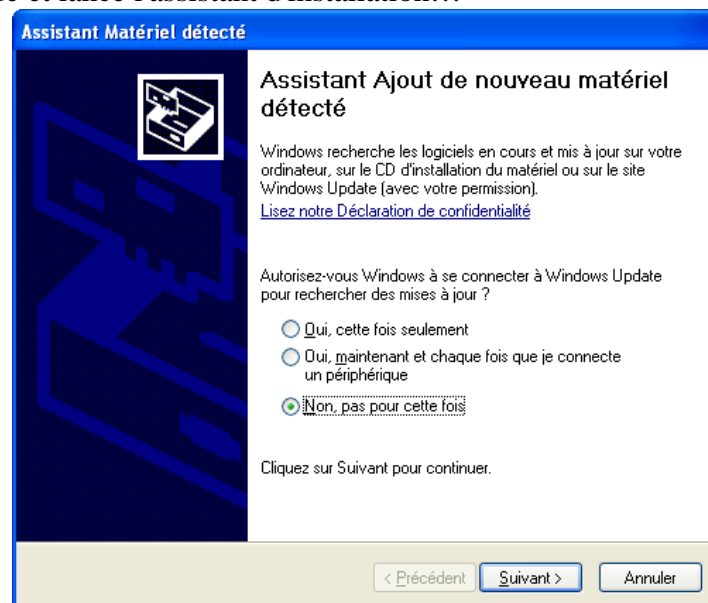
Emplacements de recherche optionnels :

- ☐ Lecteurs de disquettes
- ☐ Lecteurs de CD-ROM
- ☒ Emplacement spécifique
- ☐ Microsoft Windows Update

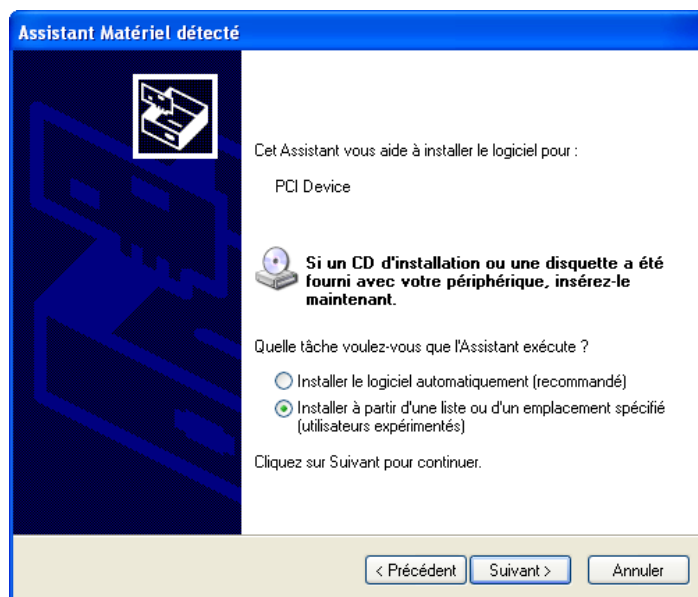
- Cliquer sur le bouton **Parcourir** puis sélectionner le répertoire **Pilotes\_PC-PC\_Drivers\Interfaces\_CAN\_LIN\_K** du CD-ROM
  - Cliquer sur **Suivant**.
  - Windows 2000 recherche le pilote et indique qu'il a trouvé un pilote pour la carte CANPCI. Cliquer sur **Suivant**.
  - Windows 2000 recopie les fichiers du pilote sur le disque dur.
  - Lorsque la copie est finie, cliquer sur **Terminer**.
  - **L'interface logicielle est prête à être utilisée.**
- 
- Vérifier que la carte fonctionne en démarrant le programme **CANEXTEST.EXE** qui est dans le répertoire **Logiciels\_PC-PC\_Software\Ex\_PC\_Software\_CAN\Test** du CD-ROM. En cas de problème, consulter le paragraphe «Dépannage de l'installation» suivant.

### IX.2.2 Windows XP

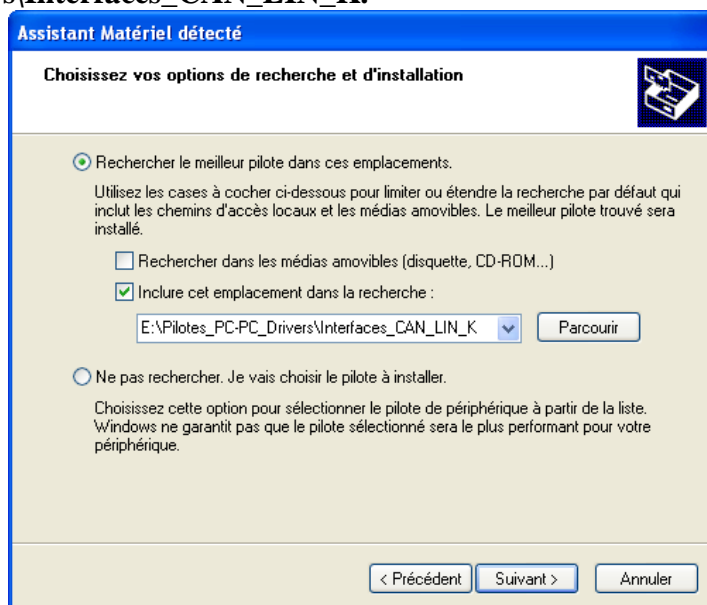
- **Arrêter le PC et installer la carte comme il est expliqué au début de ce chapitre.**
- Après le démarrage du système, Windows XP indique qu'il a détecté un nouveau périphérique et lance l'assistant d'installation...



- Insérer le CD-ROM « CD Livraison NSI » dans le lecteur du PC, choisissez l'option « Non, pas pour cette fois » et cliquer sur **Suivant**.
  - Sélectionner l'option "Installer à partir d'une liste ou d'un emplacement spécifié..." puis cliquer sur **Suivant**.



- Cocher uniquement la ligne *"Inclure cet emplacement dans la recherche"* puis cliquer sur **Parcourir** et sélectionner le répertoire **Pilotes\_PC-PC\_Drivers\Interfaces\_CAN\_LIN\_K**.

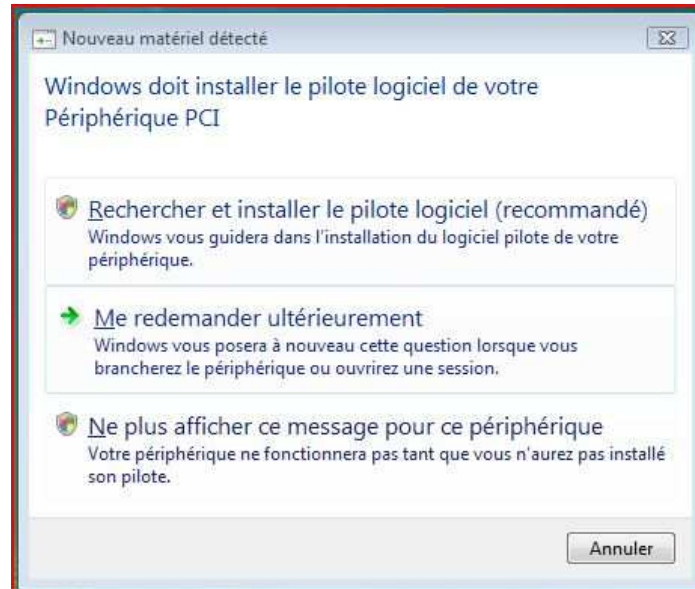


- Cliquer sur **Suivant**.
- Windows XP recherche le pilote et installe le pilote pour la carte CANPCI.
- Lorsque la copie est finie, cliquer sur **Terminer**.
- **L'interface logicielle est prête à être utilisée.**
- Vérifier que la carte fonctionne en démarrant le programme **CANEXTEST.EXE** qui est dans le répertoire **Logiciels\_PC-PC\_Software\Ex\_PC\_Software\_CAN\Test** du CD-ROM. En cas de problème, consulter le paragraphe «Dépannage de l'installation» suivant.

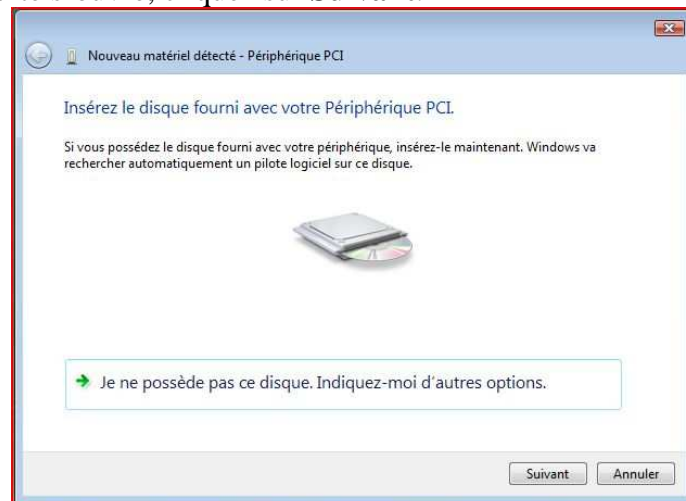
### IX.2.3 Windows VISTA



- Arrêter le PC et installer la carte comme il est expliqué au début de ce chapitre.
- Après le démarrage du système, Windows VISTA indique qu'il a détecté un nouveau périphérique et lance l'assistant d'installation...

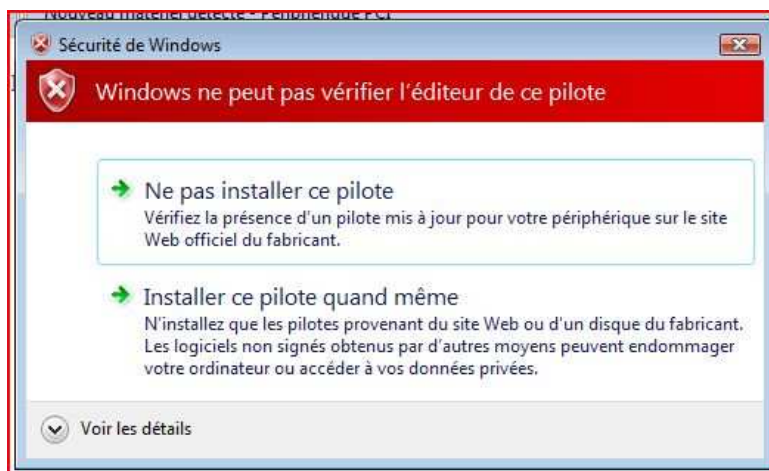


- Insérer le CD-ROM « CD Livraison NSI » dans le lecteur du PC, choisissez l'option « Rechercher et installer le pilote logiciel ».
- La fenêtre suivante s'ouvre, cliquez sur **Suivant**.

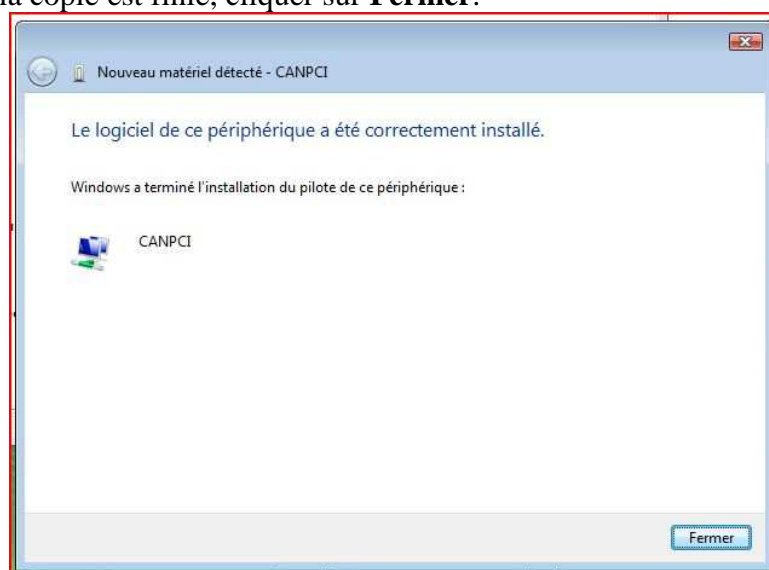


- Windows VISTA recherche le pilote et vous demande de confirmer avant de terminer l'installation. Cliquez sur « installer ce pilote quand même »





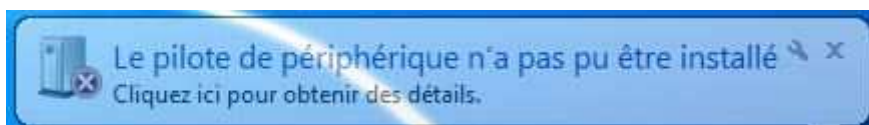
- Lorsque la copie est finie, cliquer sur **Fermer**.



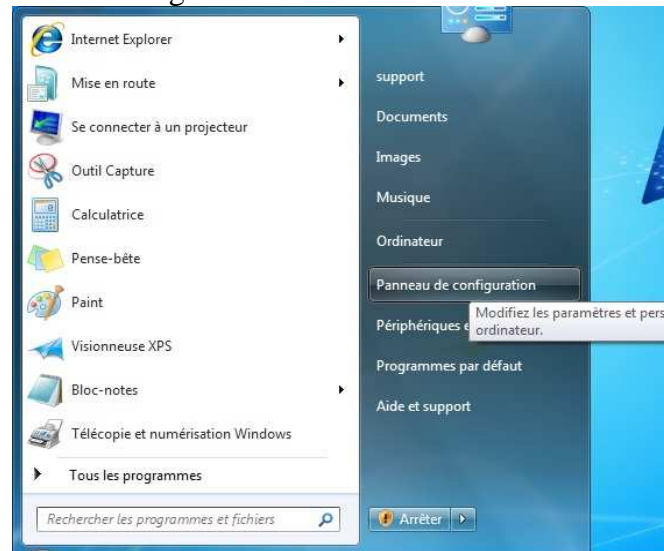
- **L'interface logicielle est prête à être utilisée.**
- Vérifier que la carte fonctionne en démarrant le programme **CANEXTEST.EXE** qui est dans le répertoire **Logiciels\_PC-PC\_Software\Ex\_PC\_Software\_CAN\Test** du CD-ROM. En cas de problème, consulter le paragraphe «Dépannage de l'installation» suivant.

## IX.2.4 Windows 7

- **Arrêter le PC et installer la carte comme il est expliqué au début de ce chapitre.**
- Après le démarrage du système, Windows VISTA indique qu'il a détecté un nouveau périphérique mais qu'il n'a pas réussi à installer le pilote.



- Allez dans « Panneau de configuration »



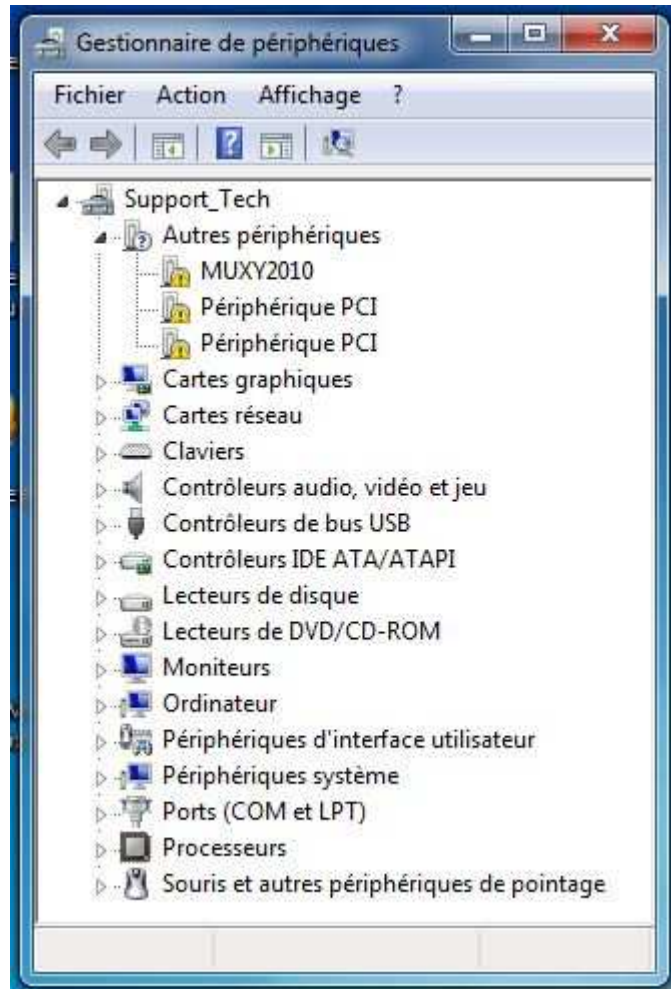
- Cliquez sur « Système »



- Cliquez sur « Gestionnaire de périphérique »



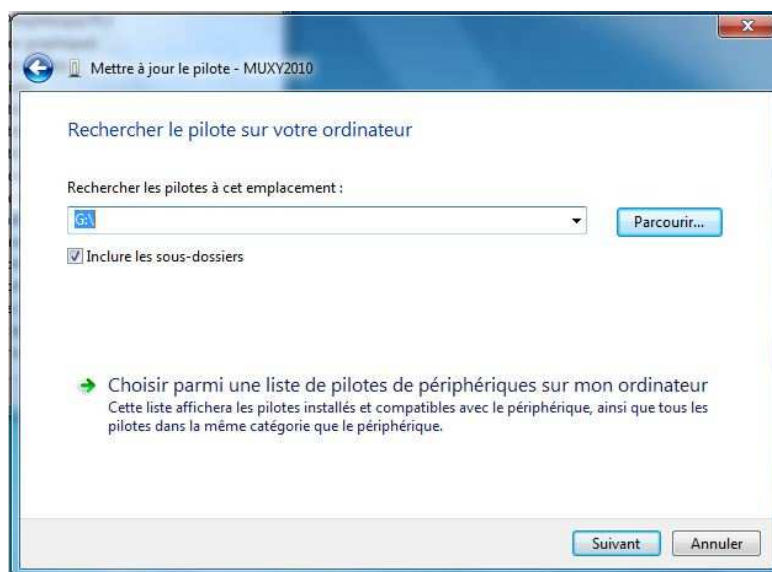
- Insérer le CD-ROM « CD Livraison NSI » dans le lecteur du PC.
- Bouton de droite sur le produit à installer (ici le périphérique PCI), et cliquez sur « Mettre à jour le pilote »



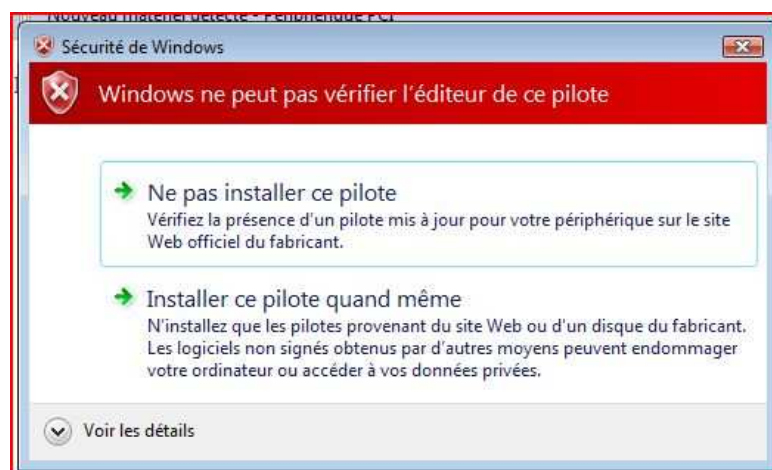
- La fenêtre suivante s'ouvre, cliquez sur « Rechercher un pilote sur mon ordinateur ».



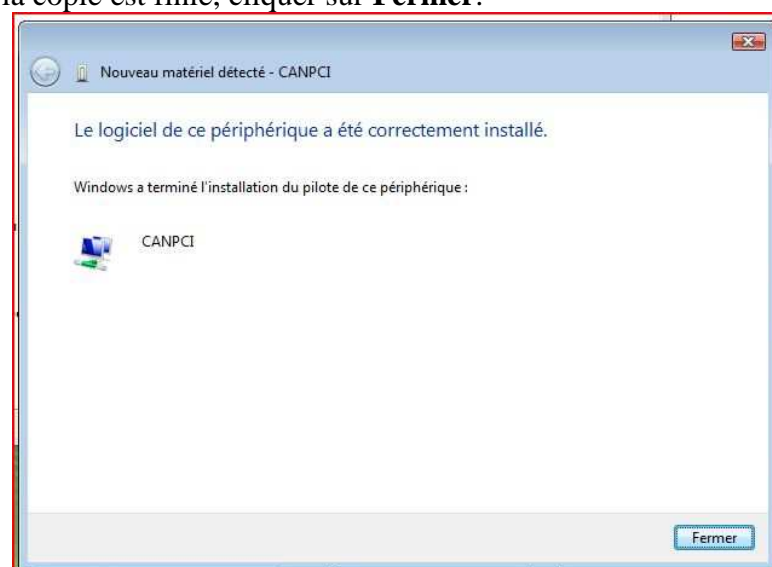
- Sélectionner le lecteur de CD/DVD dans lequel vous avez insérer le CD NSI, et cochez la case « inclure les sous-dossiers », puis cliquez sur **Suivant**.



- La fenêtre suivante apparaît. Cliquez sur « installer ce pilote quand même »



- Lorsque la copie est finie, cliquer sur **Fermer**.



- L'interface logicielle est prête à être utilisée.

- Vérifier que la carte fonctionne en démarrant le programme **CANEXTEST.EXE** qui est dans le répertoire **Logiciels\_PC-PC\_Software\Ex\_PC\_Software\_CAN\Test** du CD-ROM. En cas de problème, consulter le paragraphe «Dépannage de l'installation» suivant.

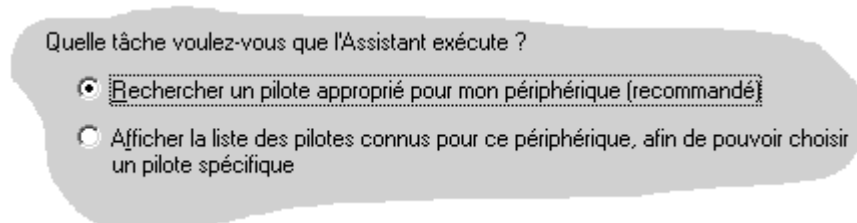
### IX.3 Installer un périphérique USB

Pour installer l'interface logicielle pour la première fois avec un périphérique USB, il faut suivre la procédure décrite dans ce paragraphe. L'installation dépend du système d'exploitation utilisé. Se reporter au paragraphe correspondant.

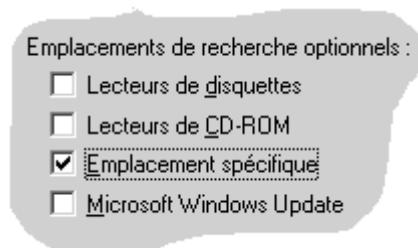
- Fermer toutes les applications en cours.
- Brancher le câble USB entre le PC et le périphérique.

#### IX.3.1 Sous Windows 2000

- Windows indique qu'il vient de détecter un nouveau périphérique USB.
- L'assistant d'installation de nouveau périphérique démarre.
- Cliquer sur le bouton "Suivant".
- Sélectionner l'option "*Rechercher un pilote approprié pour mon périphérique*".



- Cocher uniquement la ligne "*Emplacement spécifique*" puis cliquer sur **Suivant**.



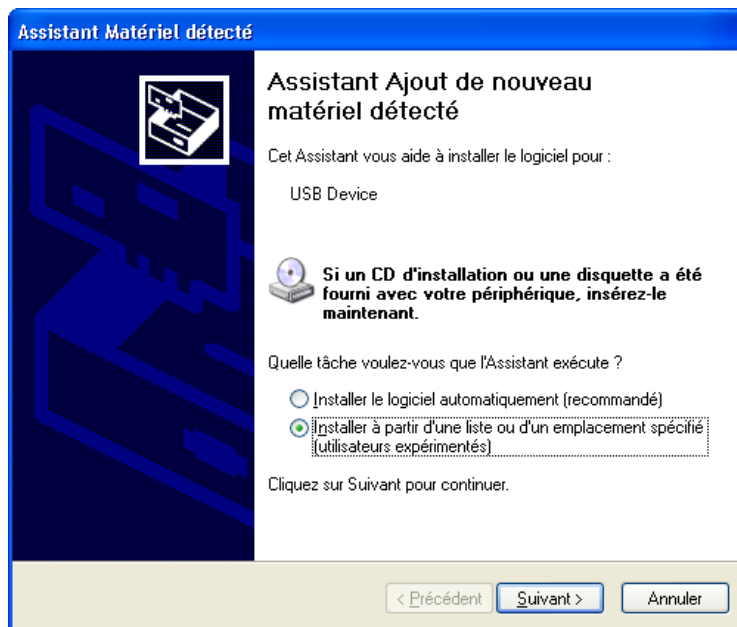
- Sélectionner le bouton **Parcourir** et sélectionner le répertoire **Pilotes\_PC-PC\_Drivers\Interfaces\_CAN\_LIN\_K**.
  - Cliquer sur **Ok**.
  - Windows 2000 recherche le pilote et indique qu'il a trouvé un pilote pour le périphérique. Cliquer sur **Suivant**.
  - Windows 2000 recopie les fichiers du pilote sur le disque dur.
  - Lorsque la copie est finie, cliquer sur **Terminer**.

**L'interface logicielle est prête à être utilisée.**

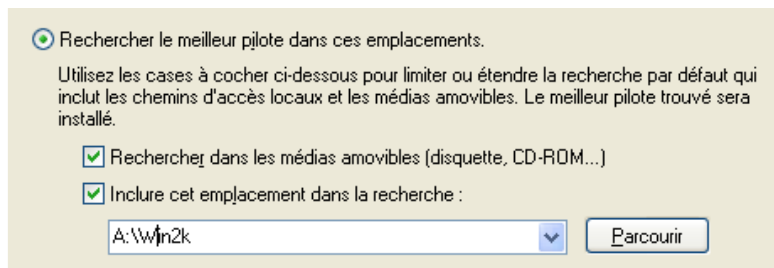
Vérifier que la carte fonctionne en démarrant le programme **CANEXTEST.EXE** qui est dans le répertoire **Logiciels\_PC-PC\_Software\Ex\_PC\_Software\_CAN\Test** du CD-ROM. En cas de problème, consulter le paragraphe «Dépannage de l'installation».

### IX.3.2 Sous Windows XP

- Windows indique qu'il vient de détecter un nouveau périphérique USB.
- L'assistant d'installation de nouveau périphérique démarre.
  - La fenêtre suivante apparaît :



- Sélectionner l'option "*Installer à partir d'une liste ou d'un emplacement spécifié*".
  - Cliquer sur **Suivant**.
- Sélectionner l'option "*Rechercher le meilleur pilote dans ces emplacements*".



- Insérer le CD-ROM « CD Livraison NSI » dans le lecteur du PC et cliquer sur **Parcourir**
- Sélectionner le répertoire **Pilotes\_PC-PC\_Drivers\Interfaces\_CAN\_LIN\_K** du CD-ROM d'installation.
- Cliquer sur **Suivant**.
  - Windows cherche et indique qu'il a trouvé le pilote USB pour le périphérique.
  - Cliquer sur **Suivant**.
  - Windows copie les fichiers sur le disque dur puis indique qu'il a terminé l'installation.
  - Cliquer sur **Terminer**.

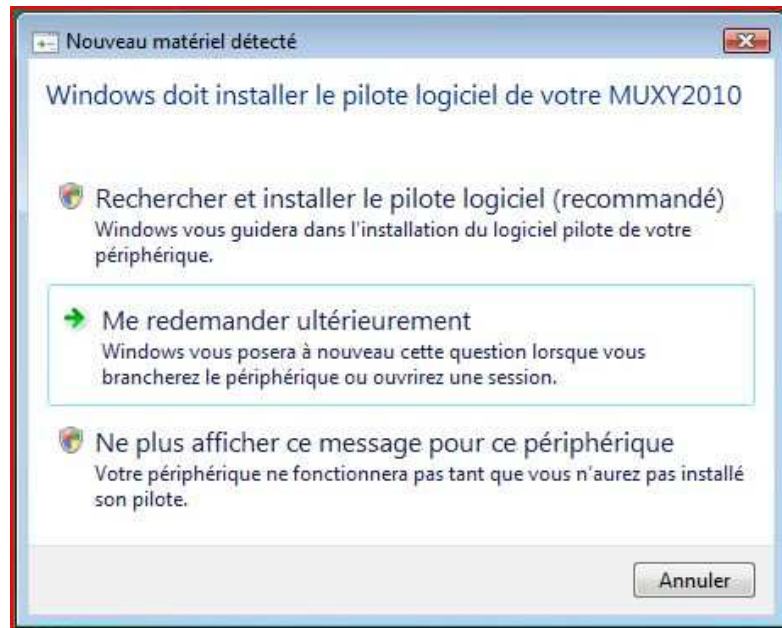
**L'interface logicielle est prête à être utilisée.**



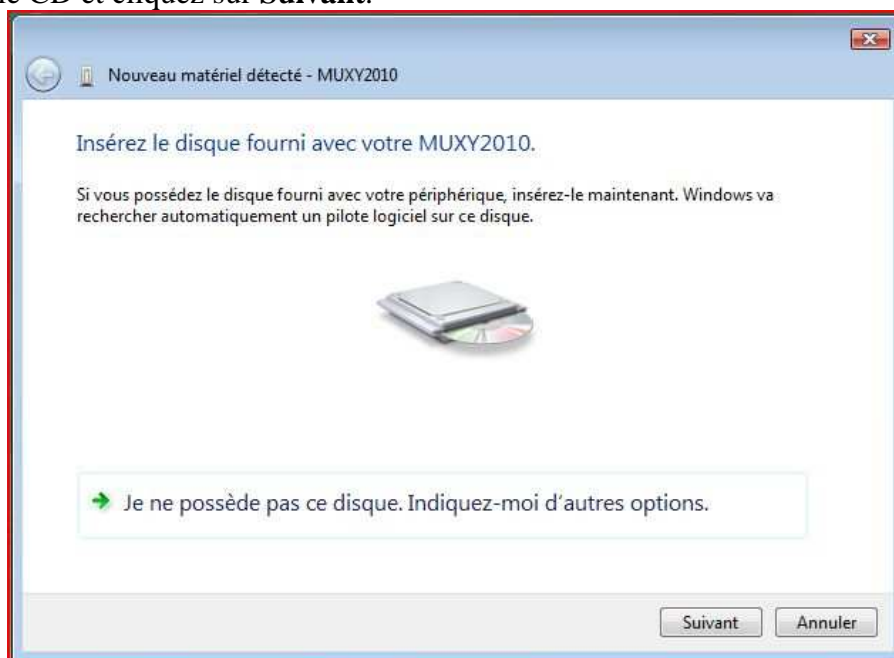
Vérifier que la carte fonctionne en démarrant le programme **CANEXTEST.EXE** qui est dans le répertoire **Logiciels\_PC-PC\_Software\Ex\_PC\_Software\_CAN\Test** du CD-ROM. En cas de problème, consulter le paragraphe «Dépannage de l'installation».

### IX.3.3 Sous Windows VISTA

- Windows indique qu'il vient de détecter un nouveau périphérique USB.
- L'assistant d'installation de nouveau périphérique démarre.
  - La fenêtre suivante apparaît :

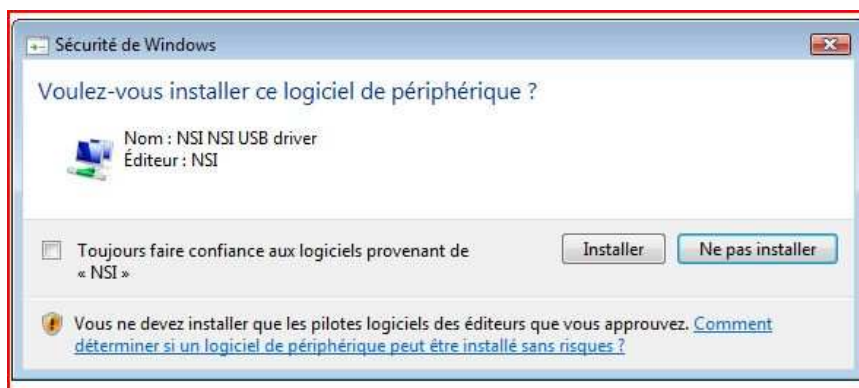


- Cliquez sur « *Rechercher et installer le pilote logiciel* ».
- Insérer le CD et cliquez sur **Suivant**.

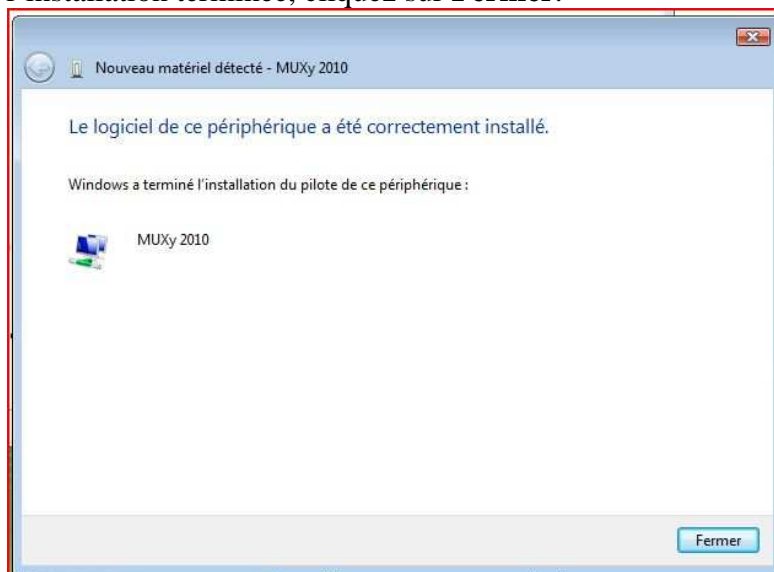


- La fenêtre suivant apparaît, cliquez sur **Installer**





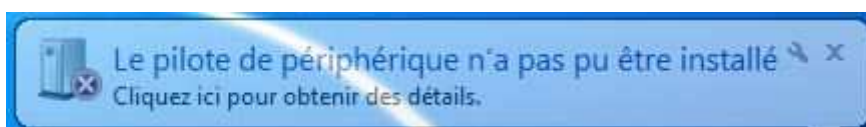
- Une fois l'installation terminée, cliquez sur **Fermer**.



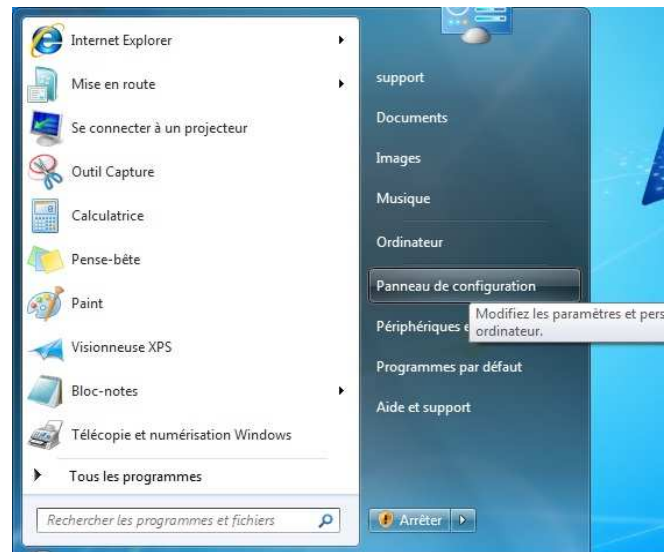
- **L'interface logicielle est prête à être utilisée.**
- Vérifier que la carte fonctionne en démarrant le programme **CANEXTEST.EXE** qui est dans le répertoire **Logiciels\_PC-PC\_Software\Ex\_PC\_Software\_CAN\Test** du CD-ROM. En cas de problème, consulter le paragraphe «Dépannage de l'installation».

### IX.3.4 Windows 7

- Windows indique qu'il vient de détecter un nouveau périphérique USB, mais qu'il n'a pas réussi à installer le pilote.



- Allez dans « Panneau de configuration »



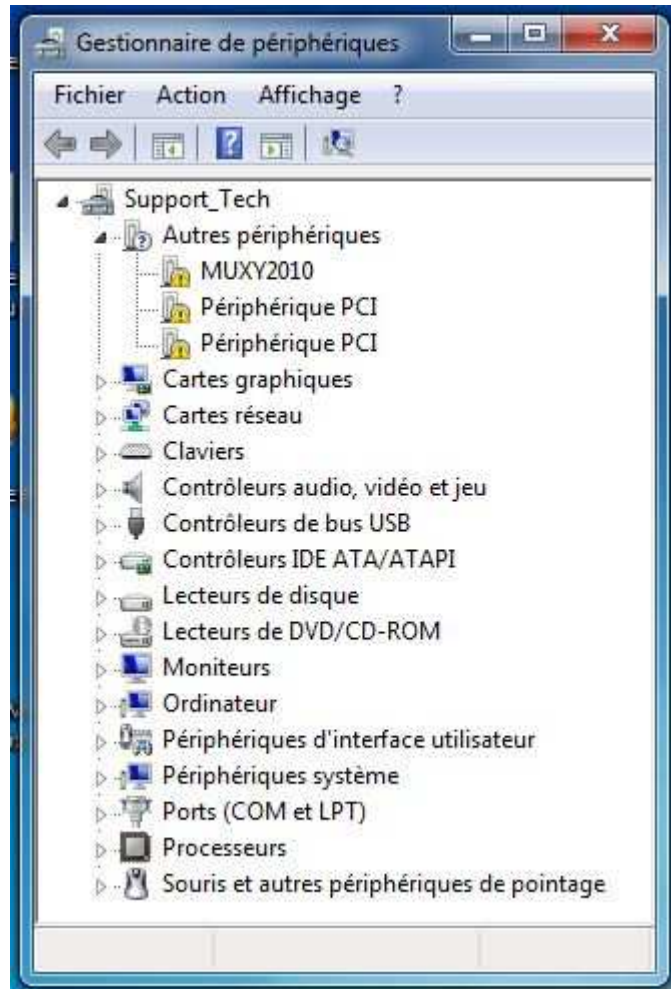
- Cliquez sur « Système »



- Cliquez sur « Gestionnaire de périphérique »



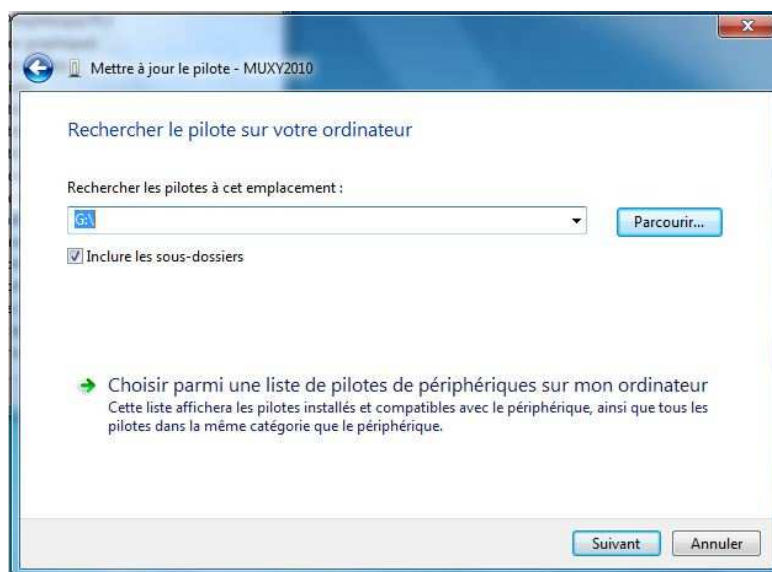
- Insérer le CD-ROM « CD Livraison NSI » dans le lecteur du PC.
- Bouton de droite sur le produit à installer (ici le MUXY2010), et cliquez sur « Mettre à jour le pilote »



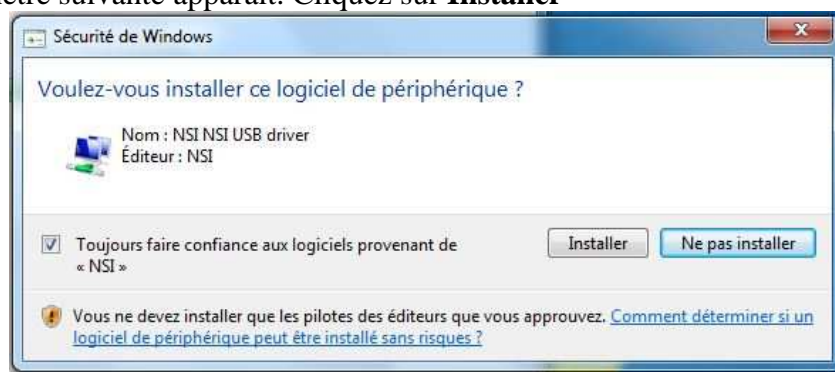
- La fenêtre suivante s'ouvre, cliquez sur « Rechercher un pilote sur mon ordinateur ».



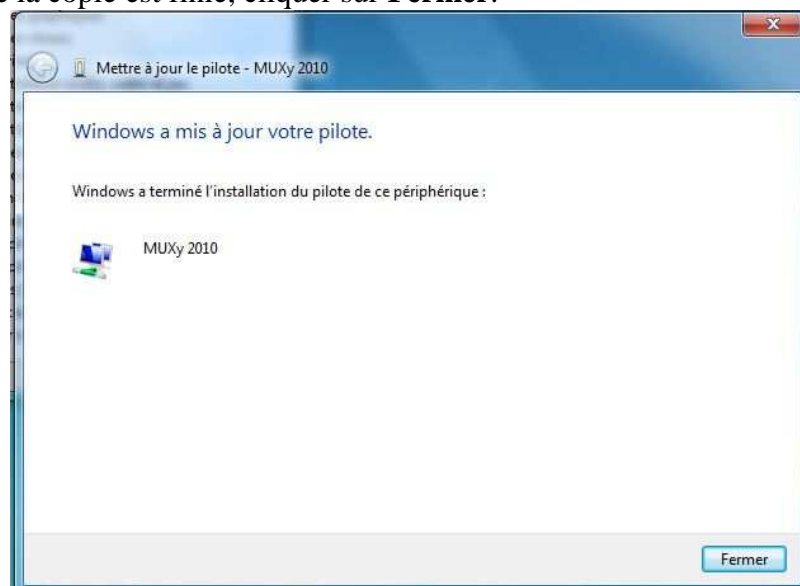
- Sélectionner le lecteur de CD/DVD dans lequel vous avez insérer le CD NSI, et cochez la case « inclure les sous-dossiers », puis cliquez sur **Suivant**.



- La fenêtre suivante apparaît. Cliquez sur **Installer**



- Lorsque la copie est finie, cliquer sur **Fermer**.



- **L'interface logicielle est prête à être utilisée.**
- Vérifier que la carte fonctionne en démarrant le programme **CANEXTEST.EXE** qui est dans le répertoire **Logiciels\_PC-PC\_Software\Ex\_PC\_Software\_CAN\Test** du

CD-ROM. En cas de problème, consulter le paragraphe «Dépannage de l'installation» suivant

## IX.4 Dépannage de l'installation

### Win2k, XP, VISTA: Windows ne détecte pas la carte au démarrage

- Vérifier dans le gestionnaire de périphériques que Windows ne détecte pas de conflit de ressources. Dans ce cas, résoudre les conflits et éventuellement désinstaller des périphériques non utilisés. Redémarrer le PC.

Ou

- Lancer l'installation de nouveau périphérique à partir du *Panneau de Configuration*.

Ou

- Vérifier que les pilotes sont dans les répertoires appropriés. Dans le cas contraire, recommencer l'installation de l'interface:

#### Windows 2000 :

Cartes PCI	<b>NSIC2K2P.SYS</b> et/ou
Cartes ISA et PCMCIA	<b>NSIC2KIS.SYS</b> et/ou
Interface CAN-USB	<b>NSIU2KBX.SYS</b> et/ou
MUXy	<b>NSIU2KBD.SYS</b> et/ou
MUXy box	<b>NSIU2KBO.SYS</b> et/ou
MUXy light	<b>NSIU2KLT.SYS</b> et/ou
MUXy2010	<b>NSIU2KUD.SYS</b> dans <b>WinNT\System32\Drivers</b>

#### Windows XP 32 bits :

Cartes PCI	<b>NSIC2K2P.SYS</b> et/ou
Cartes ISA et PCMCIA	<b>NSIC2KIS.SYS</b> et/ou
Interface CAN-USB	<b>NSIU2KBX.SYS</b> et/ou
MUXy	<b>NSIU2KBD.SYS</b> et/ou
MUXy box	<b>NSIU2KBO.SYS</b> et/ou
MUXy light	<b>NSIU2KLT.SYS</b> et/ou
MUXy2010	<b>NSIU2KUD.SYS</b> dans <b>Windows\System32\Drivers</b>

#### Windows VISTA 32 bits :


Cartes PCI	<b>NSIC2K2P.SYS</b> et/ou
Cartes ISA et PCMCIA	<b>NSIC2KIS.SYS</b> et/ou
Interface CAN-USB	<b>NSIU2KBX.SYS</b> et/ou
MUXy	<b>NSIU2KBD.SYS</b> et/ou
MUXy box	<b>NSIU2KBO.SYS</b> et/ou
MUXy light	<b>NSIU2KLT.SYS</b> et/ou
MUXy2010	<b>NSIU2KUD.SYS</b> dans <b>Windows\System32\Drivers</b>

### Win2k, XP, VISTA : La carte est signalée en erreur dans le gestionnaire de périphériques

- Cliquer sur l'icône de la carte CAN dans le Gestionnaire de Périphériques puis cliquer sur **Supprimer**. Eteindre complètement le PC puis redémarrer. La détection d'un nouveau périphérique est signalée. Fournir de nouveau les pilotes si Windows le demande.

## X. Désinstallation de l'interface logicielle

### X.1 Windows 2000

- Fermer toutes les applications en cours...
  - Double cliquer sur les icônes suivantes :
    1. Poste de Travail
    2. Panneau de Configuration
    3. Système
- 
- Cliquer sur l'onglet **Matériel** puis sur le bouton **Gestionnaire de Périphériques**.
  - Double cliquer sur **NSI** ou **Autres périphériques**.
  - Sélectionner la ligne de la carte CAN NSI
  - Dans le menu **Action**, cliquer sur **Désinstaller**.
  - Confirmer la suppression de ce périphérique.
  - Répéter l'opération pour toutes les cartes CAN installées.
  - Fermer le **Gestionnaire de Périphériques**
  - Ouvrir l'Explorateur Windows
  - Supprimer les fichiers suivants :
    - dans le répertoire **\WinNT\System32**
      - NSICANEX.DLL.
    - dans le répertoire **\WinNT\System32\Drivers**
      - NSIC2KIS.SYS (Carte ISA et PCMCIA)
      - NSIC2K2P.SYS (Carte CANPCI)
      - NSIU2KBX.SYS (Boîtier CAN-USB)
      - NSIU2KBD.SYS (MUXy)
      - NSIU2KBO.SYS (MUXy box)
      - NSIU2KLT.SYS (MUXy light)
      - NSIU2KUD.SYS (MUXy2010)



## X.2 Windows XP

- Fermer toutes les applications en cours...
- Ouvrir le menu **Démarrer** et cliquer sur les icônes suivantes :
  1. Panneau de Configuration
  2. Système
- Cliquer sur l'onglet **Matériel** puis sur le bouton **Gestionnaire de Périphériques**.
- Double cliquer sur **NSI** ou **Autres périphériques**.
- Sélectionner la ligne de la carte CAN NSI



- Dans le menu **Pilote**, cliquer sur **Désinstaller**.
- Confirmer la suppression de ce périphérique.
- *Répéter l'opération pour toutes les cartes CAN installées.*
- Fermer le **Gestionnaire de Périphériques**

## X.3 Windows VISTA

- Fermer toutes les applications en cours...
- Ouvrir le menu **Windows** et cliquer sur les icônes suivantes :
  3. Panneau de Configuration
  4. Système (s'il n'apparaît pas, cliquez sur « *affichage classique* »)
- Cliquer sur l'onglet **Gestionnaire de Périphériques**.
- Double cliquer sur **NSI** ou **Autres périphériques**.
- Sélectionner la ligne de la carte CAN NSI
- Cliquez avec le bouton de droite de la souris, et cliquez sur « *désinstaller* »



## XI. ANNEXE : Prototypes des Fonctions

```

short Ic_ActiveId(          HANDLE          hdrv,
                           unsigned long    ident,
                           unsigned short    dlc          );

short Ic_ChangeId(         HANDLE          hdrv,
                           unsigned long    oldIdent,
                           unsigned long    newIdent,
                           unsigned long    newFlowControlIdent
                           );

short Ic_ConfigBus(        HANDLE          hdrv,
                           short            compbypass,
                           short            polarite,
                           short            disconnectTx1,
                           short            disconnectRx1,
                           short            disconnectRx0   );

short Ic_ConfigEvent(      HANDLE          hDrv,
                           HANDLE          hEvent,
                           unsigned long    ident          );

short Ic_DeactivateId(     HANDLE          hDrv,
                           unsigned long    ident          );

short Ic_DeleteId(         HANDLE          hDrv,
                           unsigned long    ident          );

short Ic_EnumCards(        unsigned long*   cardcnt,
                           t_CardData*     carddata,
                           unsigned long    carddatasz    );

short Ic_ExitDrv(          HANDLE          hdrv          );

short Ic_GetAPIInfo(       HANDLE          hdrv,
                           short*          DLLversion,
                           short*          DRVversion,
                           unsigned long*   reserved      );

short Ic_GetBuf(           HANDLE          hdrv,
                           unsigned long    ident,
                           t_CANevent*     msg            );

short Ic_GetChipInfo(      HANDLE          hdrv,
                           t_CANchipInfo*   chipInfo       );

short Ic_GetChipState(     HANDLE          hDrv,
                           t_CANerr*        CANerr,
                           t_CANstat*       CANstat        );

short Ic_GetCount(         HANDLE          hdrv,
                           t_CANcounter*    cptu           );

short Ic_GetDeviceInfo(    HANDLE          hdrv,
                           t_CANdeviceInfo* deviceInfo     );

```

short <b>Ic_GetDiag</b> (	HANDLE t_CANDiag*	hdrv, CANDiag	) ;
short <b>Ic_GetEvent</b> (	HANDLE t_CANevent*	hdrv, pEvent	) ;
short <b>Ic_GetMode</b> (	HANDLE unsigned long*	hCanal, mode	) ;
short <b>Ic_InitChip</b> (	HANDLE t_CANbusParams t_CANaddressing t_CANpadding	hdrv, busParams, addressing, padding	) ;
short <b>Ic_InitDrv</b> (	short HANDLE*	cno, hdrv	) ;
short <b>Ic_InitFlowControl</b> (	HANDLE unsigned long t_CANflowParams	hdrv, ident, flowParams)	;
short <b>Ic_InitId</b> (	HANDLE t_CANobj*	hdrv, msg	) ;
short <b>Ic_InitInterface</b> (	HANDLE t_Interface	hdrv, interface	) ;
short <b>Ic_InitLineDrv</b> (	HANDLE short HANDLE BOOL	hdrv, mode, hEvent, autoWakeup	) ;
short <b>Ic_InitPeriod</b> (	HANDLE unsigned long unsigned short BOOL long	hdrv, ident, period, autostart, repeat	) ;
short <b>Ic_KillPeriod</b> (	HANDLE unsigned long	hdrv, ident	) ;
unsigned char <b>Ic_ReadChip</b> (	HANDLE unsigned char	hdrv, ad_reg	) ;
short <b>Ic_ResetBoard</b> (	HANDLE	hdrv	) ;
short <b>Ic_ResetChip</b> (	HANDLE	hdrv	) ;
short <b>Ic_SetMode</b> (	HANDLE unsigned long	hdrv, mode	) ;
short <b>Ic_SetRxMask</b> (	HANDLE unsigned long unsigned long t_CANidentType	hdrv, code, mask, identType	) ;
short <b>Ic_StartChip</b> (	HANDLE	hdrv	) ;
short <b>Ic_StopChip</b> (	HANDLE	hdrv	) ;

short <b>Ic_StartPeriod</b> (	HANDLE unsigned long	<b>hdrv,</b> <b>ident</b>	) ;
short <b>Ic_StopPeriod</b> (	HANDLE unsigned long	<b>hdrv,</b> <b>ident</b>	) ;
short <b>Ic_TxMsg</b> (	HANDLE unsigned long unsigned short unsigned char*	<b>hdrv,</b> <b>ident,</b> <b>dlc,</b> <b>data</b>	) ;
void <b>Ic_WriteChip</b> (	HANDLE unsigned char unsigned char	<b>hdrv,</b> <b>ad_reg,</b> <b>reg</b>	) ;
short <b>Ic_WriteData</b> (	HANDLE unsigned long unsigned short unsigned char*	<b>hdrv,</b> <b>ident,</b> <b>dlc,</b> <b>data</b>	) ;
short <b>Ic_WritePattern</b> (	HANDLE unsigned long unsigned short unsigned char* unsigned char	<b>hdrv,</b> <b>ident,</b> <b>size,</b> <b>data,</b> <b>dlc</b>	) ;

## XII. Annexe : Fonctions de l'API Win32

L'interface Win32 est une API (*Application Programming Interface*) développée par Microsoft pour ses systèmes d'exploitation 32 bits: Windows NT et Windows 95 / 98. L'API Win32 permet de réaliser des applications capables de fonctionner sous tous les systèmes implémentant l'interface Win32. L'API Win32 est dite "Multi Thread".

### Qu'est ce qu'un "Thread" ?

En terme de code, un "*Thread*" est simplement une séquence de code qui est exécutée par le système d'exploitation dans le contexte d'un processus (généralement une application). Un processus débute son exécution par son *Thread* principal qui, dans un programme C traditionnel, est la fonction `main()`. Une fois démarré, le programme peut créer de nouveaux "*Threads*" en effectuant un appel système spécifiant l'adresse de la fonction initiale du nouveau "*Thread*". Le système d'exploitation fait basculer le contrôle du processeur entre les *Threads* de manière préemptive pour donner l'impression que les *Threads* s'exécutent en parallèle et de manière asynchrone.

### Les fonctions Win32 à connaître

**`hEvent = CreateEvent( &sa, fManual, fInitial, pszName )`**

Création d'un événement. Un événement peut avoir deux états : "signalé" ou "non signalé". Les paramètres *sa* et *pszName* ne sont significatifs que lorsque les événements sont partagés entre des processus. Dans un processus unique, ces paramètres sont définis à NULL. Le paramètre *fInitial* indique si l'événement sera "signalé" (*fInitial* = TRUE) ou "non signalé" (*fInitial* = FALSE) à sa création. Le paramètre *fManual* indique si l'événement est repassé automatiquement (*fManual* = FALSE) à l'état "non signalé" par la fonction *WaitForSingleObject* ou s'il est géré par une application (*fManual* = TRUE). *hEvent* est l'identificateur de l'événement créé.

**`WaitForSingleObject( hEvent, dwTimeout )`**

Mise en attente du *Thread* qui appelle cette fonction de l'occurrence d'un événement. Le paramètre *hEvent* indique le *Handle* de l'événement attendu. Le paramètre *dwTimeout* précise le temps d'attente maximal. Si l'événement est "signalé", la fonction retourne aussitôt sinon elle suspend le *Thread* jusqu'à ce que l'événement devienne "signalé". *DwTimeout* peut être égal à une valeur en millisecondes de façon à ce que la fonction retourne à la fin de ce délai dans le cas où l'événement reste "non signalé". Quand *dwTimeout* est égal à INFINITE, le *Thread* reprend son cours uniquement lorsque l'événement est signalé.

**`HThread = CreateThread( lpSa, cbStack, lpStartAddr, lpvThreadParm, fdwCreate, lpIDThread )`**

Cette fonction crée un nouveau *Thread*. Le paramètre *lpSa* est un pointeur sur une structure SECURITY\_ATTRIBUTES. La valeur NULL permet d'utiliser les attribut par défaut. Le paramètre *cbStack* définit la quantité d'espace d'adressage que le thread est autorisé à employer pour sa propre pile. La valeur 0 crée une pile de 1Mo par défaut. Le paramètre

*lpStartAddr* indique l'adresse de la fonction que le nouveau *Thread* doit exécuter. Le paramètre *lpvThreadParm* est passé à la fonction lorsque le thread commence son exécution. Le paramètre *fdwCreate* spécifie des options additionnelles contrôlant la création du thread. Si cette valeur est 0, le thread commence son exécution immédiatement. *Hthread* est l'identificateur du thread créé. Le paramètre *lpIDThread* est une adresse valide où sera enregistré l'ID que le système affecte au nouveau thread.

**TerminateThread( hThread, dwExitcode )**

Cette fonction termine le *Thread* identifié par le paramètre *hThread* et place le code de sortie à la valeur *dwExitCode*. Cette fonction permet aussi de terminer un *Thread* s'il ne répond plus aux commandes. **Attention, cette fonction est dangereuse pour la stabilité du système d'exploitation et ne doit être utilisée que dans les cas les plus extrêmes.** Il est nécessaire de bien contrôler que le *Thread* qui doit être terminé ne fait plus aucun appel au pilote du périphérique.

**CloseHandle( handle )**

Cette fonction permet la fermeture d'un objet (Ex: un événement). Le paramètre *handle* est l'identificateur de l'objet à fermer.

Remarque : Le HANDLE **hdrv** retourné par la fonction **Ic\_InitDrv** ne doit pas être fermé par l'application. Il faut appeler la fonction **Ic\_ExitDrv**.

### XIII. Annexe : Programme d'exemple

Ce programme est donné à titre d'exemple. Il a été écrit en langage C et compilé avec l'outil *Developer Studio* de Microsoft. Les conséquences de toute erreur ou mauvais fonctionnement de ce programme ne sauraient mettre en cause la responsabilité de la société NSI. La totalité de ce programme d'exemple est fourni sur la disquette d'installation de l'interface logicielle dans le répertoire **Samples\FIFO**.

- CANDEMO.C
- CANDEMO.H

Ce programme montre comment émettre et recevoir des trames CAN en mode FIFO. L'émission des trames est déclenchée par l'appui d'une touche du clavier. La réception est réalisée par un Thread qui attend le remplissage de la FIFO pour afficher les événements CAN.

Pour faciliter la compréhension du code, tous les cas d'erreurs possibles ne sont pas traités. L'appel de chaque fonction est suivi de l'affichage des paramètres et du code retourné.

#### XIII.1 Déclarations

Dans cette partie du programme, on déclare les prototypes des fonctions de l'interface logicielle puis on crée les ressources systèmes nécessaires au fonctionnement de l'interface en mode FIFO.

```
#include <windows.h>
#include <conio.h>
#include <stdio.h>

// Inclusion des fichiers de déclaration des structures
// et des fonctions de l'interface CAN.
//
#include "canpcex.h"

// Prototypes des fonctions du programme d'exemple
//
#include "candemo.h"

// Variables globales
// -----
//
// Synchronisation de l'affichage
HANDLE hMutex;

// Handle du canal ouvert
HANDLE hCanal = INVALID_HANDLE_VALUE;

// Handle de l'événement FIFO
HANDLE hEvent = INVALID_HANDLE_VALUE;

// MAIN( )
//-----
```

```
// Fonction principale du programme d'exemple.
//
void main( void )
{
    // Variables internes diverses
    int i;
    short cr;
    BOOL done = FALSE;
    short canal;
    char key[2];
    short DLL,DRV;
    unsigned long cardCount;
    unsigned char data[8];

    // Variables de contrôle du Thread
    DWORD threadId;
    HANDLE hThread;

    // Structure d'initialisation d'un message CAN
    t_CANObj canObj;

    // Structure d'initialisation du bus CAN
    t_CANparams canParams;

    // Tableau pour recevoir la configuration des canaux
    t_CardData cardData[10];

    // Crée les ressources systèmes nécessaires au programme
    // pour fonctionner en mode FIFO.
    // -----
    //
    // En mode FIFO, un événement est signalé à chaque
    // fois que la FIFO n'est plus vide. Ceci permet de réveiller
    // une partie du programme placée en attente de cet événement.
    // Ce programme en attente est un "Thread" dont la fonction
    // est de dépiler les événements de la FIFO puis de se
    // rendormir lorsque la FIFO est de nouveau vide. Cet event doit
    // être créé par l'application puis fourni à l'interface.
    //
    hEvent = CreateEvent( NULL, FALSE, FALSE, NULL );

    // Crée l'objet "mutex" qui permet d'éviter les conflits
    // entre le thread principal (main) et le thread d'extraction
    // de la FIFO (Thread)
    //
    hMutex = CreateMutex( NULL, FALSE, NULL );
```

```
// En mode FIFO, les réceptions et les émission de messages CAN
// sont placées dans la FIFO. Une partie du programme (le Thread)
// attend que la FIFO ne soit plus vide pour en extraire les
// informations. La fonction CreateThread crée et démarre ce
// THREAD. Voir la fonction Thread( ).
//
hThread = CreateThread(
    NULL,
    0,
    (LPTHREAD_START_ROUTINE)Thread,
    hEvent,
    0,
    &threadId );
```

## XIII.2 Initialisations

### XIII.2.1 Interface

Cette partie du programme choisit un canal CAN disponible et l'initialise. Le mode de fonctionnement de l'interface choisi est le mode FIFO. Le débit est 250 kbps.

```
// Ic_EnumCards
// -----
// Lorsque plusieurs canaux CAN sont disponibles dans un même
// PC, il est utile de connaître les particularités de chaque
// canal pour choisir lequel utiliser. Cette fonction retourne
// la liste des canaux utilisables dans un tableau. L'indice
// du tableau correspond au numéro du canal.
//
cr = Ic_EnumCards( &cardCount, cardData, sizeof( cardData ) );

Echo( "Ic_EnumCards : %s.", GetCodeString( cr ) );
if( cr != _OK )
{
    Echo( " Failed to get CAN channels list." );
    getch( );
    return;
}

// Affiche la liste des canaux (cartes) et trouve le premier
// canal disponible. L'index de ce canal est placé dans la
// variable "canal". Sinon -1 ce qui indique aucun canal libre.
//
Echo( " Detected Boards : %d", cardCount );
canal = -1;
for( i=0; i<(int)cardCount; i++ )
{
    Echo( " %d - IO:% 3xh IRQ:%d %s (%d)",
        i,
        cardData[i].IOBaseAddress,
        cardData[i].IRQLineNumber,
        cardData[i].cardNameString,
        cardData[i].cardAlreadyOpen );
```



```
// Trouve le premier canal disponible cad non utilisé au même
// moment par une autre application.
//
if( (canal == -1) && (cardData[i].cardAlreadyOpen == FALSE ))
{
    canal = i;
}
}

// Vérifie qu'au moins un canal est disponible
if( canal < 0 )
{
    Echo(" CANTEST needs at least one available CAN channel..." );
    getch( );
    return;
}

// Ic_InitDrv
// -----
// Sélectionne et initialise un canal CAN. La valeur du premier paramètre
// indique l'indice du canal que l'on veut utiliser. Celui-ci correspond
// à l'indice dans le tableau retourné par Ic_EnumCards. Un "Handle" est
// retourné dans la variable "hCanal". Cette valeur identifie ce canal
// et permet éventuellement d'utiliser plusieurs canaux. Cette valeur est
// passée à toutes les autres fonctions de l'interface logicielle CAN.
//
cr = Ic_InitDrv( (short)canal, &hCanal );

Echo("Ic_InitDrv( %d,%08Xh ) : %s.", canal, hCanal, GetCodeString( cr ));
if( cr != _OK )
{
    Echo(" Unable to select CAN channel #%d.", canal );
    getch( );
    return;
}

// Ic_GetAPIInfo
// -----
// Retourne le type de la carte du canal initialisé et les version des
// fichiers de l'interface logicielle.
//
cr = Ic_GetAPIInfo( hCanal, &DLL, &DRV );

Echo("Ic_GetAPIInfo : %s (%s,Dll:%.2f,Drv:%.2f)",
    GetCodeString(cr),
    DLL / 100.0,
    DRV / 100.0 );
if( cr != _OK ) done = TRUE;
```

```
// Ic_InitChip
// -----
// Initialise le controleur CAN: Le débit (250kbit, echant. 75%)

CANparams.baudpresc = 4;
CANparams.tseg1     = 5;
CANparams.tseg2     = 2;
CANparams.sjw       = 1;
CANparams.sample    = 0;

cr = Ic_InitChip( hCanal, CANparams , _DC_NORMAL, _DC_NO_PADDING );

Echo("Ic_InitChip( 250kbit,75%,1SJW,1SPL ) : %s.", GetCodeString(cr));
if( cr != _OK ) done = TRUE;

// Ic_InitInterface
// -----
// Initialise l'interface CAN: Le mode de gestion des évènements
// (FIFO ou Buffer)
//
cr = Ic_InitInterface( hCanal, _FIFO);

Echo("Ic_InitInterface( FIFO ) : %s.", GetCodeString(cr));
if( cr != _OK ) done = TRUE;

// Ic_ConfigEvent
// -----
// Indique à l'interface logicielle le HANDLE de l'événement devant être
// utilisé pour signaler les évènements concernant la FIFO. Voir les
// fonction Thread et CreateEvent.
//
cr = Ic_ConfigEvent( hCanal, hEvent, 0 );

Echo( "Ic_ConfigEvent( ) : %s.", GetCodeString(cr));
if( cr != _OK ) done = TRUE;
```

### XIII.2.2 Messages

Dans cette partie du programme, on déclare l'émission d'une trame de donnée et la réception de toutes les trames de données avec des identificateurs standards de valeur paires.

```
// Déclaration d'une trame de données en émission portant l'ident
// CAN_IDENT. Dans un premier temps, on initialise la structure
// 'canObj' contenant les informations du message CAN.
// -----
//
// Valeur de l'identificateur du message 0x000-0x7FF pour un ident STD.
canObj.ident = CAN_IDENT;

// Type de l'identificateur
canObj.identType = _CAN_STD;

// Type du message CAN
canObj.frameType = _CAN_TX_DATA;

// Taille de la zone de données du message en octets 0-8
canObj.dlc = 5;
```

```
// Demande de compte rendu pour ce message. En mode FIFO, ce
// paramètre est indispensable pour que le compte rendu de
// l'échange soit placé dans la FIFO.
CanObj.statusRq = _STATUS;

// Valeurs des octets de données
canObj.data[0] = 0x01;
canObj.data[1] = 0x02;
canObj.data[2] = 0x03;
canObj.data[3] = 0x04;
canObj.data[4] = 0x05;

// Ic_InitId
// -----
// Déclare le message CAN dont les données sont passées dans
// la structure pointée par le second paramètre (canObj).
//
cr = Ic_InitId( hCanal, &canObj );

Echo( "Ic_InitId( %03Xh,%d,TX_DATA ) : %s.",
      canObj.ident,
      canObj.dlc,
      GetCodeString(cr));
if( cr != _OK ) done = TRUE;

// Ic_SetRxMask
// -----
// Déclare la réception des trames de données d'un groupe
// d'identificateur définis par un masque de bits. Dans cet
// exemple, on ne reçoit que les identificateurs STD pairs.
//
cr = Ic_SetRxMask( hCanal, 0x001, 0x000, _STD );

Echo( "Ic_SetRxMask( Mask=0x001, Val=0x000 ) : %s.", GetCodeString(cr));
if( cr != _OK ) done = TRUE;
```

### XIII.2.3 Messages segmentés

Dans cette partie du programme, on déclare l'émission segmentée d'une trame de données. Attention, cette partie de code est donnée à titre d'exemple et ne fait pas partie de l'exemple en mode FIFO car elle déclare le même identificateur que le paragraphe précédent.

```
// Déclaration d'une trame de données segmentée en émission portant
// l'ident CAN_IDENT avec demande d'acquittement. Dans un premier
// temps, on initialise la structure 'canObj' contenant les informations
// du message CAN.
// -----
//
// Valeur de l'identificateur du message 0x000-0x7FF pour un ident STD.
CanObj.ident = CAN_IDENT;

// Demande de compte rendu pour ce message. En mode FIFO, ce
// paramètre est indispensable pour que le compte rendu de
// l'échange soit placé dans la FIFO.
CanObj.statusRq = _STATUS;

// Type de l'identificateur
canObj.identType = _CAN_STD;
```

```
// Valeur de l'identificateur de contrôle de flux pour l'ident CAN_IDENT.
CanObj.identFlowControl    = CAN_FC_IDENT;

// Demande de compte rendu pour le message de contrôle de flux.
CanObj.flowControlStatusRq = _STATUS;

// Type du message CAN
canObj.frameType          = _CAN_TX_SEG_DATA;

// Taille de la zone de données du message en octets 0-4095
canObj.dlc                = 32;

// Valeurs des octets de données
for( i = 0; i < canObj.dlc; i++ ) canObj.data[0] = i;

// Ic_InitId
// -----
// Déclare le message CAN dont les données sont passées dans
// la structure pointée par le second paramètre (canObj).
//
cr = Ic_InitId( hCanal, &canObj );

Echo( "Ic_InitId( %03Xh,%d,TX_SEG_DATA ) : %s.",
      canObj.ident,
      canObj.dlc,
      GetCodeString(cr) );

if( cr != _OK ) done = TRUE;
```

### XIII.2.4 Périodiques

Cette partie du programme ne figure pas dans l'exemple général FIFO. La gestion des émission périodiques fonctionne pour les périphériques USB uniquement. Ceci permet de déclarer la période d'émission du message émis. La valeur de la période d'émission peut être changée en cours de fonctionnement (après **Ic\_StartChip** ou **Ic\_StartPeriod**).

```
// Ic_InitPeriod
// -----
// Les périphériques USB peuvent gérer automatiquement les émissions
// périodiques. Cette fonction déclare et initialise la période
// d'émission de la trame dont l'identificateur est passé en
// paramètre. La valeur de la période est en millisecondes.
// Le paramètre autoStart indique quand l'émission doit démarrer.
// TRUE : Démarrage automatique lors de l'appel de Ic_StartChip
// FALSE : Démarrage lors de l'appel à Ic_StartPeriod
// Il est possible de spécifier un nombre d'émissions de l'identificateur
// ou _CAN_INFINITE pour l'émettre jusqu'à l'appel de Ic_StopChip ou de
// Ic_StopPeriod.
//
cr = Ic_InitPeriod( hCanal, CAN_IDENT, 1000, TRUE, _CAN_INFINITE, NULL );

Echo( "Ic_InitPeriod( %03Xh,1000ms ) : %s.",
      canObj.ident,
      GetCodeString(cr));

if( cr != _OK ) done = TRUE;
```

```
// Ic_WritePattern
// Pour tout identificateur défini comme périodique par la fonction
// Ic_InitPeriod, les périphériques USB peuvent émettre des motifs
// supérieurs aux 8 octets déclarés avec Ic_InitId ou Ic_WriteData
// Il est possible de définir des motifs allant jusqu'à 256 octets
// de données. Le boîtier effectue alors le découpage en trames
// de x octets CAN où x est le paramètre dlc de la fonction.

// Prépare un motif de 50 octets
//
for( i=0; i<50; i++ ) data[i]++;

// Définit l'identificateur CAN_IDENT comme émettant un motif
// de 50 octets. Après démarrage de la période on observera sur le bus
// 6 trames de 8 octets et une trame de 2 octets pour un motif. Ce
// motif sera répété périodiquement de cette manière autant de fois
// qu'il a été spécifié auparavant dans la fonction Ic_InitPeriod
cr = Ic_WritePattern( hCanal, CAN_IDENT, 50, data, 8);

Echo( "Ic_WritePattern ( %03Xh, 50, 8 ) : %s.",
      CAN_IDENT,
      GetCodeString(cr));

if( cr != _OK ) done = TRUE;
```

### XIII.3 Démarrage

Cette partie du programme démarre le contrôleur CAN du canal. A cet instant, les trames déclarées peuvent être émises et reçues. Les trames correctement formatées sont acquittées.

```
// Ic_StartChip
// -----
// Démarre le contrôleur CAN. Les réceptions, démarrent à cet instant.
// Le contrôleur acquitte les trames correctement formatées qui
// circulent sur le réseau.
//
cr = Ic_StartChip( hCanal );
Echo( "Ic_StartChip : %s.", GetCodeString(cr));
if( cr != _OK ) done = TRUE;

// Teste que l'initialisation et le démarrage du canal CAN on été
// réussies. En cas d'échec, le programme est arrêté.
//
if( done )
{
    Echo( " Failed to initialize CAN interface. Program will stop now." );
}
else
{
    Echo( " " );
    Echo( " Press [A] key to transmit Ids " );
    Echo( " Press [ESC] to exit program " );
}
key[1] = 0;
```

## XIII.4 Emissions

Cette partie du programme est la boucle principale d'attente des commandes du clavier. Il est important de comprendre que la partie "Réceptions" de ce programme fonctionne en parallèle avec cette boucle. Chacune de ces deux parties est exécutée par un Thread indépendant.

```
// Boucle tant que la touche ESC n'est pas pressée.
//
while( !done )
{
    key[0] = getch( );
   strupr( key );
    switch( key[0] )
    {
        case 'A' :

            // Prépare les données à transmettre dans le message
            //
            for( i=0; i<5; i++ ) data[i]++;

            // Ic_TxMsg
            // -----
            // Met a jour les données et active la transmission du message
            // portant l'identificateur passé en paramètre.
            //
            cr = Ic_TxMsg( hCanal, CAN_IDENT, 5, data );

            Echo( "Ic_TxMsg( %03Xh,5,DATA ) : %s",
                CAN_IDENT,
                GetCodeString( cr ));
            break;

        case (char)27 :
            // Indique que la boucle while doit se terminer
            done = TRUE;
            break;

        default :
            Echo( "[A]ctiveID - [G]o Period - [S]top period - [ESC] Exit" );
    }
}
```

### XIII.5 Arrêt du programme

Cette partie du programme est exécutée lorsque l'utilisateur presse sur la touche ESC. On doit alors refermer le canal CAN utilisé puis libérer les ressources système utilisées. Ceci comprend le Thread de réception qui est arrêté (voir le code de ce Thread ci dessous).

```
// Ic_ExitDrv
// -----
// Ferme le handle du canal et libère le canal pour que d'autres
// applications puissent l'utiliser
//
cr = Ic_ExitDrv( hCanal );

Echo( "Ic_ExitDrv : %s", GetCodeString(cr));

// Arrête l'exécution du Thread. Dans ce cas on utilise cette fonction
// car on est sûr que l'application ne communique plus avec le pilote
// (Ic_ExitDrv déjà appelé).
TerminateThread( hThread, 0 );

// Libère les ressources système
if( hEvent ) CloseHandle( hEvent );
if( hMutex ) CloseHandle( hMutex );

puts( " Done : Press Key..." );
getch( );

// Fin du programme principal
}
```

## XIII.6 Réceptions

Cette partie du programme est le *Thread* qui est exécuté en parallèle avec le programme principal (qui est lui même un autre *Thread*). Ce *Thread* est une boucle qui attend que des éléments soient mis dans la FIFO pour aller les dépiler et les afficher. Le remplissage de la FIFO est signalé par un *event*. L'exécution du *Thread* est suspendue tant que cet événement n'a pas eu lieu. La fonction qui permet d'attendre un événement est **WaitForSingleObject**. Voir les fonctions **Ic\_ConfigEvent** et **CreateEvent** qui permettent de configurer cet événement.

```
// Thread
// -----
// Le programme Thread est créé par le programme principal.
// Il s'exécute en parallèle et de manière asynchrone. Cette
// fonction est exécutée par le Thread. Elle attend que la
// FIFO passe de l'état VIDE à NON VIDE. Ceci est signalé par
// un "event". Le thread attend que l'event soit signalé pour
// dépiler le ou les événements présents dans la FIFO. Le
// HANDLE de l'event est passé en paramètre à la fonction du
// Thread. Ce thread est arrêté par le Thread principal
// lorsqu'il se termine.
//
DWORD Thread( HANDLE hEvent )
{
    t_Event event;
    long count = 0;
    char string[0x100];

    // Boucle "infinie" (tant que le thread principal fonctionne)
    while( 1 )
    {
        // Cette fonction place le thread en attente de l'événement.
        // Noter que ceci n'occupe pas de temps CPU. Le thread est
        // endormi jusqu'à ce que le système le réveille lorsque
        // l'événement "hEvent" est signalé.
        //
        if( WaitForSingleObject( hEvent, INFINITE ) == WAIT_OBJECT_0 )
        {
            // Boucle tant que des événements sont dépilés de la FIFO
            while( Ic_GetEvent( hCanal, &event ) == _OK )
            {
                count++;
                // Affiche une ligne de texte décrivant l'événement
                GetEventString( &event, string );
                Echo( " Evt% 4d %s", count, string );
            }
        }
    }
    return 0;
}
```



## XIV. Annexe : Notes sur l'utilisation de l'adressage étendu Diag On Can

**Avertissement :** Ce mode d'adressage n'est disponible que pour :

- les cartes Muxy équipées d'une version logicielle supérieure ou égale à v5.30, du driver PC version 4.10 ou supérieure et de la DLL NSICANEX 1.13 ou supérieure.
- MUXy2010

### XIV.1 Initialisations

Pour activer l'utilisation en transmission et en réception du mode d'adressage étendu il faut le spécifier lors de l'appel de la fonction `Ic_InitChip`.

```
cr = Ic_InitChip( hCanal, CANparams , _DC_EXTENDED, _DC_NO_PADDING );
```

### XIV.2 Déclaration des messages

Lors de la déclaration des messages CAN il faut utiliser le champ **targetAddress** pour spécifier la valeur de l'adresse utilisée lors des échanges sur le bus.

Exemple :

```
t_CANObj msg[2] ;
msg[0].dlc = 4095;
for( i=0; i<msg[0].dlc; i++ ) msg[0].data[i] = i;
msg[0].data[0] = 0x17;
msg[0].data[1] = 0xFF;
msg[0].data[2] = 0x00;
msg[0].frameType = _CAN_TX_SEG_DATA;
msg[0].statusRq = _STATUS;
msg[0].flowControlStatusRq = _STATUS;
msg[0].ident = 0x688;
msg[0].identType = _CAN_STD;
msg[0].identFlowControl = 0x6A8;
msg[0].targetAddress = 0x10;
cr = Ic_InitId( hCanal, &msg[0] );
Echo("Ic_InitIdent( %d, 0x%03X, %s ) : %s.",
    canal,
    msg[0].ident,
    GetFrameType(msg[0].frameType),
    GetCodeString(cr));
if( cr != _OK ) return FALSE;
```

### XIV.3 Obtenir l'octet d'adressage utilisé par l'émetteur pour les messages en réception

Pour savoir quelle est la valeur de l'octet d'adressage qui a été reçu pour un message en réception, il faut utiliser la valeur des 8bits de LSB du champ **reserved** de l'objet `t_CANevent` de l'événement `_CAN_RX_FF_DATA`.

Exemple :

```
if(pEvent->eventType == _CAN_RX_FF_DATA)
{
    sprintf( tmp, " Tgt:%02x ", pEvent->reserved&0x00FF );
    strcat( string, tmp );
}
```

Il est important de noter que seul l'événement `_CAN_RX_FF_DATA` permet d'obtenir l'octet d'adressage, les champs **reserved** des autres événements possèdent d'autres significations.

**Nota :** Lors de la réception d'une **SINGLE\_FRAME** en mode d'adressage étendu (`_DC_EXTENDED`) un événement fictif `_CAN_RX_FF_DATA` précède l'événement `_CAN_RX_SEG_DATA` afin d'obtenir la valeur de l'octet d'adresse présent dans la trame CAN reçue.

### XIV.4 Changement de valeur de l'octet d'adressage

Pour changer la valeur de l'octet d'adressage pour un identificateur en réception ou en transmission, il faut supprimer le message avec la fonction `Ic_DeleteId`, puis le déclarer de nouveau à l'aide de la fonction `Ic_InitId`.

Exemple :

```
cr = Ic_DeleteId( hCanal, msg[0].ident );
Echo( "Ic_DeleteId( %d, 0x%08X ) : %s", canal, msg[0].ident,
GetCodeString( cr ));
msg[0].targetAddress = 0x99;
cr = Ic_InitId( hCanal, &msg[0] );
Echo("Ic_InitId( %d, 0x%03X, %s ) : %s.", canal,
msg[0].ident,
GetFrameType(msg[0].frameType),
GetCodeString(cr));
```

