

DLL DE LA CARTE SYSAM-PCI

Pour programmer la carte SYSAM-PCI, il faut utiliser la DLL SYSAMPCI.DLL fournie sur la disquette accompagnant ce manuel.

Il s'agit d'une DLL type 32 bits, donc utilisable en programmation Windows 32 bits (Windows 95, 98, 2000 et NT). Elle ne peut normalement pas être utilisée en programmation 16 bits. Elle est écrite avec la convention PASCAL.

En plus de la DLL, les unités ACQDATA.PAS (pour le langage PASCAL), ACQDATA.CPP et ACQDATA.H (pour le langage C) sont conçues pour être liées au projet en cours. Elles comportent des routines facilitant l'utilisation de cette DLL.

Des exemples de programmation en DELPHI et en C++, sont fournis sur la même disquette.

ATTENTION :

Pour faciliter la programmation des cartes Eurosmart (SYSAM-PCI, FASTLAB, PC-MES3 et PC-MES2) et pour garder une compatibilité entre elles, une structure commune à toutes les DLL a été créée.

La documentation qui suit retrace tous les éléments correspondant à ces DLL. Pour cette raison, il arrive de rencontrer des instructions qui peuvent ne pas être utilisables pour SYSAM-PCI de par sa structure.

La carte SYSAM-PCI possède sa propre intelligence lui permettant d'avoir une autonomie dans la gestion de ses différentes fonctionnalités telles que timer, déclenchement des acquisitions, ...etc.

MISE EN ŒUVRE DE L'UNITÉ ACQDATA

1. Chargement de la librairie

Utiliser la fonction : *ChargerDLLSysteme(NomSysteme:string):boolean;*

Paramètre : *NomSysteme* est le nom de la DLL demandée, y compris son chemin d'accès
L'extension .DLL est ajoutée automatiquement si elle n'est pas présente

Retour : Retourne la valeur TRUE si la DLL a pu être trouvée et chargée, sinon un message d'erreur s'affiche et la fonction retourne la valeur FALSE

2. Mise à disposition des ressources

L'appel de la fonction précédente provoque également, en cas de réussite, l'affectation des diverses routines de configuration et de programmation de la carte. Ces routines sont décrites dans le paragraphe suivant.

L'utilisateur doit commencer par appeler en premier la routine *SystemePresent* qui teste la présence du système et initialise certains paramètres. Si cette routine renvoie TRUE, les autres routines sont alors utilisables.

3. Déchargement de la librairie

Si la DLL ne doit plus être utilisée, il est souhaitable de libérer la mémoire qu'elle occupe à l'aide de la procédure *SupprimerDLLSysteme*.

TYPE DES ARGUMENTS UTILISES

- *Byte* : un octet non signé (8 bits)
- *Word* : un mot de deux octets non signé (16 bits)
- *ShortInt* : un octet signé (8 bits)
- *Integer* : un mot de deux octets signés (16 bits)
- *LongInt* : un mot de 4 octets signés (32 bits)
- *Double* : un réel double précision (codé sur 8 octets suivant le format Exposant/Mantisse)
- *Boolean* : un octet représentant un état logique (TRUE ou FALSE)

DESCRIPTION DES RESSOURCES DISPONIBLES

Routines générales

fonction SystemePresent: boolean

Description : L'utilisateur doit commencer par appeler en premier la routine *SystemePresent* qui teste la présence du système et initialise certains paramètres.

Retour : Si cette routine renvoie TRUE, les autres routines sont alors utilisables, sinon, il faut vérifier le bon positionnement de la carte et ses adresses physiques.

fonction RenvoyerSysteme: PtrSysteme

Description : Cette fonction renvoie un pointeur sur une structure d'information sur le système d'acquisition. Elle est automatiquement appelée si l'on utilise la routine *ChargerDLLSysteme* pour charger la DLL et place les informations dans la variable pointeur de structure *SystemeAcquis*.

Retour : Le pointeur renvoyé pointe sur une structure de type TSysteme, ainsi définie :

```
TSysteme = object
  NomSysteme: string[20];      { nom de la carte }
  NomConstructeur: string[15]; { nom du constructeur de la carte }
  OctetEtat : word;           { octet d'état de la carte }
  AdrBase1: integer;          { adresse de base de la carte }
  AdrBase2: integer;          { seconde adresse de base de la carte }
  VoieDebut: byte;            { n° de première entrée analogique }
  VoieFin : byte;             { n° de dernière entrée analogique }
  SortieDebut: byte;          { n° de première sortie analogique }
  SortieFin: byte;            { n° de dernière sortie analogique }
  NbDeBits: byte;             { Nb de bits de la carte }
  MaxCalibreEntree : byte;     { Nb de calibres d'entrée }
  MaxCalibreSortie : byte;     { Nb de calibres de sortie }
  MaxGain : byte;             { Nb de gains programmables }
  MaxPorts : byte;            { Nb maxi de ports logiques d'entrée }
end;
```

Exemples : Pour le PASCAL : SystemeAcquis^.NomSysteme
Pour le C : SystemeAcquis->NomSysteme

fonction LireInfos(CodeGenre,Numero,Indice:integer):PChar

Description : Cette fonction permet de récupérer des informations de configuration du système

Paramètres : *CodeGenre* est une constante entière qui indique la nature des informations demandées
Numero indique le numéro d'une entrée, d'une sortie ou d'un port logique
Indice est un indice en cas de configuration possédant plusieurs valeurs possibles

Retour : Le résultat est un pointeur sur une chaîne à zéro terminal dont le contenu dépend de l'information demandée. Dans tous les cas :

- Si la demande n'est pas pertinente (par exemple paramètre *Numero* ou *Indice* hors des limites acceptées par le système), la fonction renvoie une chaîne vide.
- Si un paramètre n'est pas utile pour le type de demande, on peut mettre sa valeur à 0

CodeGenre	Paramètres	Valeur renvoyée
ca_Version	Numero : Indice de renvoi 1 : Nom du système + n° de version 2 : comme 1 plus nom du constructeur autres : n° de version seul	Chaîne contenant le numéro de la version de la DLL et d'autres informations suivant la valeur du paramètre <i>Numero</i> .
ca_Entree	Numero : n° d'entrée analogique	Nom de l'entrée analogique (de forme "Eai")
ca_Sortie	Numero : n° de sortie analogique	Nom de sortie analogique (de forme "SAi")

ca_Port	<i>Numero</i> : n° de port logique <i>Indice</i> : Type d'information demandée	Indice=1 : Nom du port logique (de forme "Port B" ou "Port C") Indice=2 : Nombre de bits utilisables Indice=3 : '0' si le port est en entrée '1' si le port en est sortie '2' si le port est inhibé Indice=4 : '1' si le port est bidirectionnel sinon '0'
ca_CalibreEntree	<i>Numero</i> : n° d'entrée analogique <i>Indice</i> : Indice du calibre d'entrée	Renvoie pour une entrée de n° donnée, le calibre d'entrée d'indice donné sous la forme : 'Min/Max'. Ex : '-10/+10'
ca_CalibreSortie	<i>Numero</i> : n° de sortie analogique <i>Indice</i> : Indice du calibre de sortie	Renvoie pour une sortie de n° donnée, le calibre de sortie d'indice donné sous la forme : 'Min/Max'. Ex : '-10/+10'
ca_Gain	<i>Numero</i> : n° d'entrée analogique <i>Indice</i> : Indice du gain	Renvoie pour une entrée de n° donnée, la valeur du gain de numéro donné. Ex : '10'

Exemple : LireInfos(ca_Port,1,1) renvoie 'Port B' LireInfos(ca_CalibreEntree,0,0) renvoie '-10/+10'

procedure ProgrammerEtatSysteme:(Code:word; Valide:boolean)

Description : Permet de positionner certains indicateurs régissant le fonctionnement du système

Paramètres : *Code* est une constante entière désignant l'indicateur à positionner. Les valeurs possibles sont
ca_InhiberTimer permet d'ignorer la durée imposée par la programmation du timer du système
ca_ModeDiff permet de sélectionner ou non le fonctionnement en mode différentiel
ca_Arret provoque un arrêt immédiat de la séquence d'acquisition
Valide à TRUE pour activer l'indicateur, à FALSE pour le désactiver

Exemple : l'instruction *ProgrammerEtatSysteme(ca_Arret,true)* provoque un arrêt immédiat de la séquence d'acquisition. Elle doit être placée dans la routine qui gère l'acquisition.

fonction OkEtatSysteme:(Code:word):boolean

Description : Permet de connaître l'état d'un indicateur

Paramètres : *Code* est une constante désignant l'indicateur dont on veut connaître l'état .

Ces valeurs caractérisent le système d'acquisition utilisé et ne sont pas modifiables :

ca_CalibresGlobal Commutation globale des calibres d'entrée seule possible

ca_ModeDiffProg Le système peut travailler en mode différentiel

ca_EntreesConsecutives Indicateur positionné si le système utilisé nécessite que les entrées soient employées par ordre de numéro croissant

Ces valeurs peuvent être modifiées par l'utilisateur à l'aide de la fonction *ProgrammerEtatSysteme*:

ca_InhiberTimer Inhibition de la programmation du timer du système

ca_ModeDiff Fonctionnement en mode différentiel

ca_Arret Arrêt immédiat de la séquence d'acquisition

Retour : renvoie TRUE si l'indicateur est actif sinon FALSE

procedure EcrireTempo(Tempo_us:integer)

Description : Programme une temporisation pour la commutation de calibre et de gain

Paramètres : *Tempo_us* valeur de la temporisation en µs

fonction LireTempo:integer

Retour : renvoie en µs la temporisation pour la commutation de calibre et de gain

Routines pour le timer

ATTENTION : Ces routines ne sont pas valables pour la carte SYSAM-PCI

Elles sont conservées pour garder une compatibilité avec les anciennes cartes d'Eurosmart.

procedure ProgrammerTimer(Duree:Double; Mode:byte)

Description : Programme le timer du système

Paramètres : *Duree* durée de comptage du timer en µs

Mode Mode de travail du timer (voir notice de la carte). Les modes les plus utiles sont

- Mode=0 : l'état du timer passe à 0 dès la validation du timer puis repasse à 1 à la fin du comptage
- Mode=1 : Le timer fonctionne en oscillateur permanent et génère un signal dont la période est égale à la durée programmée. Son état passe donc alternativement de 0 à 1 pour chaque durée égale à la moitié de la durée programmée

procedure ValiderTimer

Description : Démarre le timer du système

procedure DevaliderTimer

Description : Arrête le timer du système

fonction LireTimer:byte

Retour : Renvoie l'état de la sortie du timer (0 ou 1)

Exemple : Les instructions suivantes programment le timer en mode 0 sur une durée de 10s et attendent la fin du comptage

Cet exemple n'est pas valable pour la carte SYSAM-PCI.

```
var B :byte;  
ProgrammerTimer(10000000,0);  
ValiderTimer;  
repeat  
  B:=LireTimer;  
  until B=1;  
DevaliderTimer;
```

Routines pour les entrées analogiques

procedure LimitesCalibreEA(NumEntree,Indice:byte; var Min,Max:Double)

Description : Renvoie les limites minimales et maximales d'un calibre d'entrée analogique

Paramètres : *NumEntree* Numéro de l'entrée analogique

Indice Indice du calibre (de 1 à 4)

Retour : Les valeurs minimales et maximales sont renvoyées dans les variables MIN et MAX
Si le numéro d'entrée ou l'indice ne sont pas valides, MIN et MAX contiennent la valeur 0

fonction LireCalibreEA(NumEntree:byte):byte

Description : Renvoie l'indice du calibre actuel de l'entrée analogique donnée

Paramètres : *NumEntree* Numéro de l'entrée analogique

Retour : Valeur de l'indice. Utiliser la fonction *LimitesCalibresEA* pour connaître l'étendue du calibre à partir de l'indice
Retourne 1 si l'entrée n'existe pas (correspond au calibre le plus élevé)

procedure EcrireCalibreEA(NumEntree,Indice:byte)

Description : Programme sur l'entrée donnée le calibre d'indice transmis

Paramètres : *NumEntree* Numéro de l'entrée analogique

Indice Indice du calibre (de 1 à 4)

fonction LireGain(NumEntree:byte):byte

Attention : Cette fonction n'est pas utilisée pour SYSAM-PCI. Elle est conservée pour garder une compatibilité avec les anciennes cartes d'Eurosmart.

Description : Renvoie l'indice du gain actuel de l'entrée analogique donnée

Paramètres : *NumEntree* Numéro de l'entrée analogique

Retour : Valeur de l'indice. Utiliser la fonction *LireInfo* pour obtenir la valeur du gain
1 si l'entrée n'existe pas ou si ne comporte pas de gain programmable

procedure EcrireGain(NumEntree,Indice:byte)

Attention : Cette procédure n'est pas utilisée pour SYSAM-PCI. Elle est conservée pour garder une compatibilité avec les anciennes cartes d'Eurosmart.

Description : Programme sur l'entrée donnée le gain d'indice transmis

Paramètres : *NumEntree* Numéro de l'entrée analogique
Indice Indice du gain (suivant possibilités : Indice de 1 à 3 pour Fastlab et 1 pour PCMES)

procedure ProgrammerAccesRapide(Activer:boolean; NumEntree,NMoy:integer)

Description : Ce mode permet d'accélérer les lectures successives de valeurs par les routines *LireEA_Entier* ou *LireEA_Reel*

Paramètres : *Activer* Rend actif (true) ou non (false) le mode de lecture répétée rapide
Si la valeur est false les deux paramètres suivant sont sans signification
NumEntree Numéro de l'entrée analogique
NMoy Nombre de lectures à effectuer (de 1 à 100). le résultat étant alors la moyenne de ces mesures

Exemple : `ProgrammerLectureRapide(true,2,1);`
`for I:=1 to 100 do`
`Table[I]:=LireEA_Reel(2,1);`
`ProgrammerLectureRapide(false,0,0);`

Remarque : L'accélération obtenue est sensible sur la carte Fastlab et surtout sur la carte SYSAM-PCI.

fonction LireEA_Entier(NumEntree,NMoy:integer):integer

Description : Lecture d'un entier sur une entrée analogique

Paramètres : *NumEntree* Numéro de l'entrée analogique
NMoy Nombre de lectures à effectuer (de 1 à 100). le résultat étant alors la moyenne de ces mesures

Retour : Valeur entière acquise sur l'entrée (codage suivant possibilité de la carte)
0 si l'entrée n'existe pas

fonction LireEA_Reel(NumEntree,NMoy:integer):Double

Description : Lecture d'un réel sur une entrée analogique

Paramètres : *NumEntree* Numéro de l'entrée analogique
NMoy Nombre de lectures à effectuer (de 1 à 100). le résultat étant alors la moyenne de ces mesures

Retour : Valeur réelle (en volt) acquise sur l'entrée (suivant le calibre programmé)
0 si l'entrée n'existe pas

fonction BinVoltEA(NiveauB, ICalibre:integer):Double

Description : Convertit une valeur entière en une valeur réelle

Paramètres : *NiveauB* Valeur entière à convertir
ICalibre Indice du calibre d'entrée à utiliser pour la conversion

Retour : Valeur réelle convertie d'après le calibre transmis
La valeur retournée est tronquée en fonction des limites réelles du calibre
Retourne 0 si ICalibre n'est pas un indice valide

fonction VoltBinEA(NiveauV:Double; ICalibre:integer):integer

Description : Convertit une valeur réelle en une valeur entière

Paramètres : *NiveauV* Valeur réelle (en volt) à convertir
ICalibre Indice du calibre d'entrée à utiliser pour la conversion

Retour : Valeur entière convertie d'après le calibre transmis
La valeur retournée est tronquée en fonction des limites binaire du calibre
Retourne 0 si ICalibre n'est pas un indice valide

Routines pour la sortie analogique

procedure LimitesCalibreSA(NumSortie,Indice:byte; var Min,Max:Double)

Description : Renvoie les limites minimales et maximales d'un calibre de sortie analogique

Paramètres : *NumSortie* Numéro de la sortie analogique
Indice Indice du calibre (Indice=1)

Retour : Les valeurs minimales et maximales sont renvoyées dans les variables MIN et MAX
Si le numéro d'entrée ou l'indice ne sont pas valides, MIN et MAX contiennent la valeur 0

fonction LireCalibreSA(NumSortie:byte):byte

Attention : Cette fonction n'est pas utilisée pour SYSAM-PCI. Elle est conservée pour garder une compatibilité avec les anciennes cartes d'Eurosmart.

Description : Renvoie l'indice du calibre actuel de la sortie analogique donnée

Paramètres : *NumSortie* Numéro de la sortie analogique

Retour : Valeur de l'indice. Utiliser la fonction *LimitesCalibresSA* pour connaître l'étendue du calibre à partir de l'indice
Retourne 1 si la sortie n'existe pas (correspond au calibre le plus élevé)

procedure EcrireCalibreSA(NumSortie,Indice:integer)

Attention : Cette procédure n'est pas utilisée pour SYSAM-PCI. Elle est conservée pour garder une compatibilité avec les anciennes cartes d'Eurosmart.

Description : programme sur la sortie donnée le calibre d'indice transmis

Paramètres : *NumSortie* Numéro de la sortie analogique
Indice Indice du calibre (suivant possibilités : Indice 1 pour Fastlab et 1 ou 2 pour PCMES)

*procedure EcrireSA_Entier(NumSortie, Valeur:integer) ******

Description : Écriture d'un entier sur une sortie analogique

Paramètres : *NumSortie* Numéro de la sortie analogique
Valeur Valeur entière à écrire (codage suivant les possibilités de la carte)

procedure EcrireSA_Reel(NumSortie:byte; Valeur:Double)

Description : Écriture d'un réel sur une sortie analogique

Paramètres : *NumSortie* Numéro de la sortie analogique
Valeur Valeur réelle (en volt) à écrire (suivant le calibre programmé)

fonction BinVoltSA(NiveauB, ICalibre:integer):Double

Description : Convertit une valeur entière en une valeur réelle

Paramètres : *NiveauB* Valeur entière à convertir
ICalibre Indice du calibre de sortie à utiliser pour la conversion

Retour : Valeur réelle convertie d'après le calibre transmis
La valeur retournée est tronquée en fonction des limites réelles du calibre
Retourne 0 si ICalibre n'est pas un indice valide

fonction VoltBinSA(NiveauV:Double; ICalibre:integer):integer

Description : Convertit une valeur réelle en une valeur entière

Paramètres : *NiveauV* Valeur réelle (en volt) à convertir

ICalibre Indice du calibre de sortie à utiliser pour la conversion

Retour : Valeur entière convertie d'après le calibre transmis

La valeur retournée est tronquée en fonction des limites binaire du calibre

Retourne 0 si ICalibre n'est pas un indice valide

Routines pour les entrées/sorties logiques

Les routines qui suivent font référence à un numéro de port logique.

Le port B reçoit le numéro 1, le port C reçoit le numéro 2.

fonction LirePort(NumPort:byte):byte

Description : Lecture d'un port logique en entrée

Paramètres : *NumPort* numéro du port logique

Retour : Valeur binaire (sur 8 bits) lue sur l'entrée demandée

Si le port logique demandé n'existe pas, la valeur retournée n'a pas de signification

procedure EcrirePort(NumPort,Valeur:byte)

Description : Écriture d'un port logique en sortie

Paramètres : *NumPort* numéro du port logique

Valeur Valeur binaire (sur 8 bits) à écrire

fonction LireBit(NumPort,NumBit:byte):byte

Description : Lecture d'une entrée logique

Paramètres : *NumPort* numéro du port logique

NumBit numéro du bit à lire sur le port logique (de 1 à 8)

Retour : État logique de l'entrée (0 ou 1)

Si le port logique demandé n'existe pas, la valeur retournée n'a pas de signification

procedure EcrireBit(NumPort,NumBit:byte; Etat:boolean)

Description : Écriture d'une sortie logique

Paramètres : *NumPort* numéro du port logique

NumBit numéro du bit à écrire sur le port logique (de 1 à 8)

Etat État logique du bit à écrire (0 ou 1)

fonction LireSensPort(NumPort:byte):byte

Description : Lecture du sens de transfert d'un port logique

Paramètres : *NumPort* numéro du port logique

Retour : Retourne une valeur entière en fonction du sens du port. Les valeurs possibles sont

ca_PortEnEntree (valeur 0) si le port logique est en entrée

ca_PortEnSortie (valeur 1) si le port logique est en sortie

ca_PortInhibe (valeur 2) si le port logique est inhibé (non utilisable)

ca_PortInexistant (valeur 3) si le port logique n'existe pas

fonction ProgrammerSensPort(NumPort,Dir:byte):boolean

Description : Programme un port logique dans un sens donné

Paramètres : *NumPort* numéro du port logique

Dir constante décrivant le sens du port. Les valeurs possibles sont :

ca_PortEnEntree (valeur 0) si le port logique doit être placé en entrée

ca_PortEnSortie (valeur 1) si le port logique doit être placé en sortie

Retour : Retourne TRUE si le port logique est programmable sinon FALSE

Routines évoluées

procedure PreparerAcquisition(Points,Moy:integer; Duree:Double; PTableVoie:PtrTableVoie; TacheFond:TypeProcTache)

Description : Transmet plusieurs paramètres en vue d'une acquisition globale à l'aide de la routine *AcquerirTable*

Paramètres : *Points* Nombre de points à acquérir (limité par la constante globale *NPMMaximum* qui peut se modifier)
Moy Nombre de lectures à effectuer. La valeur prise sera la moyenne de ces mesures (de 1 à 100)
Duree Durée d'échantillonnage entre points en µs

PTableVoie Pointeur de type *PtrTableVoie* sur un tableau décrivant les entrées à utiliser.

PtrTableVoie = *^TTableVoie*;

TTableVoie = array[0..TotalVoiesSorties] of ShortInt;

Ce tableau doit être rempli à partir de son indice 0 par le numéro des entrées à utiliser, dans l'ordre où elles doivent être acquises, en terminant par une cellule contenant la valeur -1.

Exemple : *TVoies* : *TTableVoie*;

PTvoies: *PtrTableVoie*

TVoies[0]:=1; { dans l'ordre les entrées 1 puis 2 puis 0 }

TVoies[1]:=2;

TVoies[2]:=0;

TVoies[3]:=-1; { marque de fin d'exploration de tableau }

PTVoies:=@*TVoies*;

TacheFond Pointeur facultatif sur une routine appelée à la fin de chaque point acquis.

Si une routine est transmise, elle doit gérer la détection de l'appui sur la touche ESCAPE qui permet d'arrêter prématurément l'acquisition.

Si aucune routine n'est prévue, transmettre la valeur NIL. Dans ce cas, la détection de l'appui sur la touche ESCAPE est automatiquement réalisée par une routine par défaut.

Exemple : *PreparerAcquisition(100,200,1,PTVoies,Nil)*;

fonction AcquerirTable(PTable:PtrTableVal; var DernierPoint:integer):integer

Description : Lance une acquisition globale, avec les paramètres définis dans la routine *PreparerAcquisition*

Paramètres : *PTable* Pointe sur un tableau d'entiers de type *TTableVal* qui recevra les valeurs acquises.

PTableVal=*^TTableVal*;

TypeTableVal = array[0..TotalVoiesSorties*NPMMaximum] of SmallInt;

ATTENTION : Il s'agit d'un tableau d'entiers codés sur 2 octets

L'utilisateur peut déclarer un tableau de ce type mais il est préférable, pour limiter l'utilisation de la mémoire au strict minimum de faire une affectation directe

TailleMem:=2*NbEntrees*NbPoint+2;

GetMem(*PTable*,*TailleMem*);

Après utilisation, la mémoire sera libérée par *FreeMem*(*PTable*,*TailleMem*);

ATTENTION : Aucune vérification de validité n'est effectuée. L'utilisateur est seul responsable de la bonne et suffisante affectation de la mémoire allouée au tableau.

Les valeurs entières acquises sont introduites dans le tableau à partir de l'indice 1 dans l'ordre où elles sont obtenues donc pour reprendre l'exemple précédent on obtiendrait : *V1_Voie1*,

V1_Voie2, *V1_Voie0*, *V2_Voie1*, *V2_Voie2*, *V2_Voie0*, *V3_Voie1*

Retour : Retourne 0 si tout s'est bien passé

Retourne 1 si la routine a été arrêtée prématurément par appui sur la touche ESCAPE

Retourne un code d'erreur en cas de problème

La variable *DernierPoint* reçoit le rang du dernier point acquis, qui peut être différent du nombre total de points demandé dans *PreparerAcquisition* si l'acquisition a été arrêtée à l'aide de la touche ESCAPE.

procedure ProgrammerDeclenchement(Voie:ShortInt; Niveau:Double; Condition:byte)

Description : Définit les conditions de déclenchement de l'acquisition

Paramètres : *Voie* Numéro de l'entrée sur laquelle les conditions de déclenchement sont recherchées

Niveau Valeur en volt de la tension appliquée à l'entrée qui provoquera le déclenchement

Condition Définit le sens de l'évolution du signal.

Les valeurs possibles sont : 0 pour le sens montant 1 pour le sens descendant

fonction Declencher:boolean

Attention : Cette fonction n'est pas utilisée pour SYSAM-PCI. Elle est conservée pour garder une compatibilité avec les anciennes cartes d'Eurosmart.

Description : Place le système en attente de l'apparition des conditions de déclenchement définies par la routine précédente

La touche ESCAPE permet de forcer la fin de l'attente si les conditions n'apparaissent jamais.

Retour : Retourne TRUE si les conditions de déclenchement sont trouvées sinon retourne FALSE

procedure PreparerEmission(Points,NbRep:integer; Duree:double; PTableVoie:PtrTableVoie; TacheFond:TypeProcTache)

Description : Transmet plusieurs paramètres en vue d'une émission globale à l'aide de la routine *EmettreSorties*

Paramètres : *Points* Nombre de points à acquérir (limité par la constante globale *NPMMaximum* qui peut se modifier

Duree Durée d'échantillonnage entre points en µs

NbRep Nombre de répétitions de la séquence de sortie

Une valeur négative provoque une répétition permanente jusqu'à l'appui sur la touche ESCAPE

PTableVoie Pointeur de type *PtrTableVoie* sur un tableau décrivant les sorties à utiliser.

Ce tableau est de même structure que celui décrit pour la routine *PreparerAcquisition* en utilisant les numéros de sortie à la place de ceux d'entrée.

TacheFond Pointeur facultatif sur une routine appelée à la fin de l'émission de chaque point

procedure EmettreSorties(PTableVal:PtrTableVal)

Description : Lance une émission globale sur les sorties analogiques, avec les paramètres définis dans la routine *PreparerEmission*

Paramètres : *PTable* Pointe sur un tableau d'entiers de type *TTableVal* qui contient les valeurs entières à émettre.

Ce tableau possède la même structure que celle vue pour la routine *AcquérirTable*. L'utilisateur doit le remplir préalablement, à partir de l'indice 1, avec les valeurs à envoyer, dans l'ordre où elles doivent être émises.

Exemple pour la seule sortie n° 2 : V1_Sortie2, V2_Sortie2, V3_Sortie2,

Exemple deux sorties : V1_Sortie1, V1_Sortie2, V2_Sortie1, V2_Sortie2, V3_Sortie1, V3_Sortie2....