Manuel d'utilisation
Bibliothèques de communication

Table Of Contents

LIBRAIRIE NSJSON

Cette librairie permet de parser les fichiers JSON, par exemple ceux qui sont récupérés après un appel de service Web en mode REST.

Exemple JSON (Librairie NSJSON)

Voici un exemple d'utilisation de la librairie NS_JSON.

```
local Pointer pMain, pointer pName, pointer pTab, pointer pObject
local pointer pObject2, pointer pObject3, pointer pElem
local DynStr text$
local dynstr ds, myBool%(1), myInt%

text$ = '{\
   "name": "Gest1",\
   "principals": [   {\
      "id": "01fd937a-a1b0-4539-960c-6f441cd64301",\
      "anonymous": false,\
      "tenantIdentifier": {"id": -100},\
      "languageIdentifier": {\
         "id": 501,\
         "locale": "fr" },\
      "authenticated": true,\
      "name": "Gest2",\
      "userIdentifier": {\
         "id": 10718,\
         "tenantIdentifier": {"id": -100},\
         "name": "Gest2"}\
   }]\
}'

; Parsing de la chaine JSON
pMain = NSJSON_PARSE(text$)
if pMain = 0
   message "Parsing", "echec"
   exit
endIf

pName = NSJSON_GET(pMain, 'name')
if pName <> 0
   ds = NSJSON_TOSTRING(pName)
   Trace "name ", ds
   NSJSON_DISPOSE pName
endIf

; Recuperation du tableau
pTab = NSJSON_GET(pMain, 'principals')
trace "NSJSON_SIZE (pTab)=" , NSJSON_SIZE (pTab)
if pTab <> 0
   ; Recuperation du premier objet du tableau
   pObject = NSJSON_GETAT(pTab, 0)
   if pObject <> 0
      pElem = NSJSON_GET(pObject, "id")
      if pElem <> 0
```

```
      ds = NSJSON_TOSTRING(pElem)
      trace "Id", ds
      NSJSON_DISPOSE pElem
   endIf
   pElem = NSJSON_GET(pObject, "anonymous")
   if pElem <> 0
      myBool% = NSJSON_TOBOOL(pElem)
      Trace "anonymous", mybool%
      NSJSON_DISPOSE pElem
   endIf
   pObject2 = NSJSON_GET(pObject, "tenantIdentifier")
   if pObject2 <> 0
      pElem = NSJSON_GET(pObject2, "id")
      if pElem <> 0
         myInt% = NSJSON_TOINT(pElem)
         Trace "tenantIdentifier/id", myInt%
         NSJSON_DISPOSE pElem
      endIf
      NSJSON_DISPOSE pObject2
   endIf
   pObject2 = NSJSON_GET(pObject, "languageIdentifier")
   if pObject2 <> 0
      pElem = NSJSON_GET(pObject2, "id")
      if pElem <> 0
         myInt% = NSJSON_TOINT(pElem)
         Trace "languageIdentifier/id", myInt%
         NSJSON_DISPOSE pElem
      endIf
      pElem = NSJSON_GET(pObject2, "locale")
      if pElem <> 0
         ds = NSJSON_TOSTRING(pElem)
         Trace "locale", ds
         NSJSON_DISPOSE pElem
      endIf
      NSJSON_DISPOSE pObject2
   endIf
   pElem = NSJSON_GET(pObject, "authenticated")
   if pElem <> 0
      myBool% = NSJSON_TOBOOL(pElem)
      Trace "authenticated", mybool%
      NSJSON_DISPOSE pElem
   endIf
   pElem = NSJSON_GET(pObject, 'name')
   if pElem <> 0
      ds = NSJSON_TOSTRING(pElem)
      Trace "name", ds
      NSJSON_DISPOSE pElem
   endIf
   pObject2 = NSJSON_GET(pObject, "userIdentifier")
   pElem = NSJSON_GET(pObject2, "id")
   if pObject2 <> 0
      if pElem <> 0
         myInt% = NSJSON_TOINT(pElem)
         Trace "userIdentifier/id", myInt%
         NSJSON_DISPOSE pElem
      endIf
      pElem = NSJSON_GET(pObject2, "name")
      if pElem <> 0
```

```
            ds = NSJSON_TOSTRING(pElem)
            Trace "name", ds
            NSJSON_DISPOSE pElem
         endIf
         pObject3 = NSJSON_GET(pObject2, "tenantIdentifier")
         if pObject3 <> 0
            pElem = NSJSON_GET(pObject3, "id")
            if pElem <> 0
               myInt% = NSJSON_TOINT(pElem)
               Trace "userIdentifier/tenantIdentifier/id", myInt%
               NSJSON_DISPOSE pElem
            endIf
            NSJSON_DISPOSE pObject3
         endIf
         NSJSON_DISPOSE pObject2
      endIf
      NSJSON_DISPOSE pObject
   endIf
   NSJSON_DISPOSE pTab
endIf
NSJSON_DISPOSE pmain
```

***Voir aussi** NSJSON_DISPOSE*

Référence de la librairie NSJSON

Fonction NSJSON_GET (Librairie NSJSON)

Recherche un objet dans une chaîne de caractère jSon.

Syntaxe	**NSJSON_GET** *(Value, key$)*			
Paramètres	value	pointer	i	objet jSon à parser
	key$	cString	I	objet jSon recherché
Valeur retournée	POINTER Pointeur sur l'objet trouvé Si la valeur retournée vaut 0, l'objet n'a pas été trouvé.			

***Voir aussi** NSJSON_DISPOSE, Exemple jSon*

Fonction NSJSON_GETAT (Librairie NSJSON)

Recherche un objet élément de tableau dans une chaîne de caractère jSon.

Syntaxe	**NSJSON_GETAT** *(Value, index%)*			
Paramètres	value	pointer	i	objet jSon à parser
	index%	int(4)	I	index de l'objet cherché
Valeur retournée	POINTER Pointeur sur l'objet trouvé			

	Si la valeur retournée vaut 0, l'objet n'a pas été trouvé.

***Voir aussi** NSJSON_DISPOSE, Exemple jSon*

Fonction NSJSON_PARSE (Librairie NSJSON)

Permet d'effectuer une recherche dans le contenu d'un texte.

Syntaxe	**NSJSON_PARSE** *(Input)*			
Paramètre	input	dynStr	I	Chaine dynamique à parser
Valeur retournée	POINTER Pointeur sur l'élément noeud racine			

Exemple :

```
pMain = NSJSON_PARSE(text$)
if pMain = 0
   message "Parsing", "echec"
   exit
endIf
pName = NSJSON_GET(pMain, 'name')
```

***Voir aussi** NSJSON_DISPOSE, Exemple jSon*

Fonction NSJSON_SIZE (Librairie NSJSON)

Retourne le nombre d'éléments d'un tableau d'objets JSON.

Syntaxe	**NSJSON_SIZE** *(Value)*			
Paramètre	value	pointer	I	objet jSon de type tableau
Valeur retournée	INT(4) nombre d'éléments du tableau			

Les éléments individuels peuvent être accédés via la fonction NSJSON_GETAT

***Voir aussi** Exemple jSon, NSJSON_GETAT*

Fonction NSJSON_TOBOOL (Librairie NSJSON)

Retourne le Boolean ou INT(1) correspondant à l'objet JSON.

Syntaxe	**NSJSON_TOBOOL** *(Value)*			
Paramètres	value	pointer	I	objet jSon à parser
Valeur retournée	INT(1) valeur de type booléen de l'objet Si la valeur retournée vaut 0, l'objet n'a pas été trouvé.			

***Voir aussi** Exemple jSon*

Fonction NSJSON_TOINT (Librairie NSJSON)

Retourne l'entier correspondant à l'objet JSON.

Syntaxe	**NSJSON_TOINT** *(Value)*			
Paramètres	value	pointer	I	objet jSon à parser
Valeur retournée	INT(4) valeur de type entière de l'objet Si la valeur retournée vaut 0, l'objet n'a pas été trouvé.			

***Voir aussi** Exemple jSon*

Fonction NSJSON_TONUM (Librairie NSJSON)

Retourne le réel correspondant à l'objet JSON.

Syntaxe	**NSJSON_TONUM** *(Value)*			
Paramètres	value	pointer	I	objet jSon à parser
Valeur retournée	NUM valeur de type réel de l'objet Si la valeur retournée vaut 0, l'objet n'a pas été trouvé.			

***Voir aussi** Exemple jSon*

Fonction NSJSON_TOSTRING (Librairie NSJSON)

Retourne la String correspondant à l'objet JSON.

Syntaxe	**NSJSON_TOSTRING** *(Value)*			
Paramètres	value	pointer	I	objet jSon à parser
Valeur retournée	DYNSTR valeur de l'objet String Si la valeur retournée vaut 0, l'objet n'a pas été trouvé.			

Exemple :

```
pMain = NSJSON_PARSE(text$)
pName = NSJSON_GET(pMain, 'name')
if pName <> 0
   ds = NSJSON_TOSTRING(pName)
   ....
   NSJSON_DISPOSE pName
endIf
```

***Voir aussi** Exemple jSon*

Instruction NSJSON_DISPOSE (Librairie NSJSON)

Permet de libérer un objet jSon.

Syntaxe	**NSJSON_DISPOSE** *(value)*			
Paramètres	value	pointer	I	objet à libérer

Exemple :

```
pMain = NSJSON_PARSE(text$)
if pMain = 0
   message "Parsing", "echec"
   exit
endIf
NSJSON_DISPOSE(pMain)
```

Voir aussi NSJSON_PARSE

LIBRAIRIE EHLLAPI NS3270

La librairie EHLLAPI permet d'accéder à l'émulateur d'un terminal 3270 à travers NCL. Il est possible de lire l'ensemble des informations stockées dans le buffer 3270, de simuler programmatiquement l'appui de touches par un utilisateur, etc...
Un ensemble complet d'instructions de synchronisation permettent d'attendre l'affichage complet du buffer 3270 avant d'analyser son contenu.
Un outil complémentaire, NSFIELDS.EXE, permet de récupérer l'ensemble des informations concernant l'écran 3270 couramment affiché. Ces informations peuvent ensuite être utilisées pour extraire du buffer 3270 des données dont une application graphique de réhabillage a besoin.

Installation

Déclarez NS3270.NCL dans les librairies nécessaires au développement de votre application.
Vérifiez que le fichier NSxx3270.DLL est bien dans un des répertoires du PATH sous Windows.

NSFIELDS

NSFIELDS est un utilitaire permettant de lire le contenu de l'écran 3270 affiché dans la fenêtre de l'émulateur. La position et les attributs de chaque champ 3270 sont ensuite écrits dans un fichier au format texte. Le contenu du fichier résultat peut ensuite être exploité afin de construire une application graphique de réhabillage des écrans 3270 en fonction des positions et des attributs des champs.
Pour lancer NSFIELDS.EXE, il faut avoir préalablement démarré l'émulateur 3270 de façon à ce que l'interface EHLLAPI soit chargée.
Il suffit ensuite de taper dans la ligne de commande de File/Run (Fichier/Exécuter) du Program Manager de Windows :

```
NSFIELDS [-Session] [-DLL:name.functionname] Nom_de_fichier
```

où :

- [-Session] désigne le nom court (A, B, C ou D) de la session dont vous désirez capturer les informations. Ce paramètre est optionnel. S'il est omis, la session A est automatiquement utilisée. Le résultat est stocké dans le fichier de nom Nom_de_fichier.
- [-DLL] spécifie la DLL à utiliser pour dialoguer avec l'émulateur 3270 si celle ci est différente de la DLL par défaut (ACS3EHAP). name est le nom de la DLL à utiliser. functionname est le nom de la fonction d'entrée de la DLL à appeler (voir syntaxe de l'instruction ENTRY3270).

Une fois l'écriture sur le fichier réalisée, un message d'avertissement est affiché.
Exemples :

```
NSFIELDS -B SCREEN.TXT
```

Stocke, dans le fichier ECRAN.TXT, le descriptif de l'écran 3270 et les informations associées à la session B.

```
NSFIELDS SCREEN2.TXT
```

Stocke, dans le fichier ECRAN2.TXT, le descriptif de l'écran 3270 et les informations associées à la session A.

```
NSFIELDS -A -DLL:EHLLAPI.API SCREEN3.TXT
```

Stocke, dans le fichier ECRAN3.TXT, le descriptif de l'écran 3270 et les informations associées à la session A, en utilisant la DLL EHLLAPI et la fonction d'entrée API.
Les informations fournies par NSFIELDS sont les suivantes :

- informations générales sur EHLLAPI : numéro de version, date de release, langue, type de moniteur supporté,
- informations sur la session,
- copie de l'écran courant 3270 : les numéros de colonnes et de lignes sont affichées respectivement en haut et à gauche du contenu de l'écran,
- position courante du curseur lors de la copie écran,
- tableau indiquant champ par champ :
    - son numéro d'ordre,
    - ses positions de début et de fin,
    - sa longueur,
    - ses attributs (Protégé / Non protégé, Numérique / Alphanumérique, Visible / Invisible, Modifié ou non),
    - son code attribut,
    - ses 30 premiers caractères.

Fichier NS3270.NCL

Les verbes de la librairie NS3270, décrits ci après, sont déclarés dans le fichier texte écrit en NCL, de nom NS3270.NCL. Ce fichier peut aussi contenir des verbes complémentaires (API non publique). Vous pouvez donc désirer consulter ce fichier pour avoir la référence exhaustive de la librairie.
Pour consulter le fichier NS3270.NCL :

1. Placez vous dans le répertoire <NATSTAR>\NCL ou <NSDK>\NCL.

<NATSTAR> représente le répertoire que vous avez choisi au moment de l'installation de NatStar et <NSDK> celui de NS-DK.

2. Editez le fichier NS3270.NCL avec n'importe quel éditeur de texte.

Catégories fonctionnelles de la librairie NS3270

Démarrer, piloter et terminer une connexion 3270

Instruction CONNECT3270 (Librairie EHLLAPI NS3270)

Crée une connexion avec une session donnée.

Syntaxe	**CONNECT3270** *session*			
Paramètre	session	CSTRING	I	nom de la session connectée

1. La session indiquée devient la session connectée active. La session connectée active précédente reste connectée mais n'est plus active.
2. Ce n'est pas forcément une erreur de passer le nom d'une session qui est déjà connectée. Au contraire, c'est la manière qui permet de la rendre à nouveau active.
3. Le nom utilisé pour la session est le nom court (une lettre, en général A, B, C ou D).

Exemple :

```
INIT3270 ; Initialise la librairie EHLLAPI
CONNECT3270 "A" ; Connecte à la session A
; La session A est connectée et active
CONNECT3270 "B" ; Connecte à la session B
; La session B est connectée et active
; La session A est connectée et inactive
CONNECT3270 "A" ; Rend active la session connectée A
STOP3270 ; Termine les appels à EHLLAPI
; Déconnecte les sessions A et B
```

Voir aussi DISCONNECT3270

Instruction DISCONNECT3270 (Librairie EHLLAPI NS3270)

Déconnecte la session connectée et active.

Syntaxe	**DISCONNECT3270** *session*			
Paramètre	session	CSTRING	I	nom de la session à déconnecter

1. Pour déconnecter une session non active, il faut la rendre active en faisant appel à l'instruction CONNECT3270.
2. Au retour de cette instruction, plus aucune session n'est active : d'autres sessions peuvent par contre être connectées. Utilisez CONNECT3270 pour les rendre actives.
3. Il est conseillé de déconnecter explicitement les sessions qui ont été connectées, plutôt que de s'appuyer sur une déconnexion implicite lors de l'appel à l'instruction STOP3270.

Exemple :

```
INIT3270 ; Initialise la librairie EHLLAPI
CONNECT3270 "A" ; Connecte à la session A
; La session A est connectée et active
...
DISCONNECT3270 "A" ; Déconnecte la session connectée A
STOP3270 ; Termine les appels à EHLLAPI
```

Voir aussi CONNECT3270

Instruction ENTRY3270 (Librairie EHLLAPI NS3270)

Spécifie le nom de la DLL EHLLAPI à charger par INIT3270.

Syntaxe	**ENTRY3270** *nomDLL[.nomfonction]*			
Paramètre	nomDLL.nomfonction	CSTRING	I	nom de la DLL à charger, et optionnellement nom de la fonction à employer dans cette librairie.

1. Le nom de la DLL doit être spécifié sans son extension. Si le nom de la fonction d'entrée est également spécifiée, elle doit être séparée du nom de la DLL par un point (".").
2. Par défaut, lorsque cette instruction n'est pas employée, INIT3270 charge la librairie ACS3EHAP.DLL et utilise la fonction d'entrée HLLAPI.
3. Les noms de la DLL et de la fonction d'entrée sont fournis par le fabricant de la carte de télécommunication proposant un émulateur 3270.

Exemple :

```
; Utilisation par défaut : librairie ACS3EHAP.DLL et entrée HLLAPI donc
ENTRY3270 n'est pas nécessaire
INIT3270
...
STOP3270

; Strictement équivalent à :
ENTRY3270 "ACS3EHAP" ; ou "ACS3EHAP.HLLAPI"
INIT3270
...
STOP3270

; Utilisation librairie EHLLAPI.DLL, même entrée HLLAPI que librairie IBM
ENTRY3270 "EHLLAPI"
INIT3270
...
STOP3270

; Autre utilisation : librairie EHLLAPI.DLL et entrée API
ENTRY3270 "EHLLAPI.API"
```

Voir aussi INIT3270

Fonction GETSTATUS3270% (Librairie EHLLAPI NS3270)

Retourne le statut de la session.

Syntaxe	**GETSTATUS3270%**
Valeur retournée	INT(2)

La valeur du statut est une des constantes S3270_*%.

Exemple :

```
INIT3270
MOVE GETSTATUS3270% TO S%
IF (S% = S3270_DISCONNECTED%) OR (S% = S3270_UNOWNED%)
...; Erreur
ENDIF
```

Voir aussi Constantes S3270_*%

Instruction INIT3270 (Librairie EHLLAPI NS3270)

Initialise la librairie EHLLAPI.

Syntaxe	**INIT3270**

1. Cette instruction doit être appelée avant toute autre fonction ou instruction de la librairie EHLLAPI (sauf éventuellement ENTRY3270).
2. Cette instruction charge la librairie ACS3EHAP.DLL ou celle éventuellement spécifiée par ENTRY3270, et met à zéro le statut d'erreur.

Voir aussi ENTRY3270, STOP3270

Instruction LOCK3270 (Librairie EHLLAPI NS3270)

Bloque l'émulation de façon à ce que l'utilisateur ne puisse pas modifier directement l'écran 3270 en dehors de l'application conçue avec NatStar (ou NS-DK).

Syntaxe	**LOCK3270**

1. Après un LOCK3270, une application est donc assurée d'être la seule à pouvoir faire des entrées 3270.
2. Tant qu'aucun LOCK3270 n'a été fait, l'écran n'est pas bloqué : le mode UNLOCK3270 est sélectionné par défaut.

Voir aussi UNLOCK3270

Constantes S3270_*% (Librairie EHLLAPI NS3270)

Codes retour de la fonction GETSTATUS3270%.

Syntaxe	Déclaration interne	Description
S3270_APPLICATION%	2	La session est connectée et l'écran appartient à un programme applicatif.
S3270_CONTROL%	1	La session est connectée et l'écran appartient au SSCP.
S3270_DISCONNECTED%	0	La session est déconnectée. Généralement, cela signifie qu'il n'y a pas de connexion physique entre le terminal et le host.

S3270_UNOWNED%	3	La session est connectée au système mais l'écran n'appartient à aucun programme applicatif.

Voir aussi ***GETSTATUS3270%***

Instruction STOP3270 (Librairie EHLLAPI NS3270)

Fin des appels à la librairie EHLLAPI.

Syntaxe	**STOP3270**

1. Cette instruction doit être appelée à la fin de l'application avant de sortir. Elle déconnecte toutes les sessions connectées.
2. Aucune autre fonction ou instruction de EHLLAPI ne doit être appelée après STOP3270.

Voir aussi ***INIT3270***

Instruction UNLOCK3270 (Librairie EHLLAPI NS3270)

Débloque l'émulation de façon à ce que l'utilisateur puisse modifier directement l'écran 3270 en dehors de l'application conçue avec NatStar (ou NS-DK).

Syntaxe	**UNLOCK3270**

1. Après un UNLOCK3270, une application n'est donc plus assurée d'être la seule à pouvoir faire des entrées 3270.
2. Tant qu'aucun LOCK3270 n'a été fait, l'écran n'est pas bloqué : le mode UNLOCK3270 est sélectionné par défaut.

Voir aussi ***LOCK3270***

Gérer les erreurs

Constantes ERR3270_*% (Librairie EHLLAPI NS3270)

Valeurs possibles du code erreur rendu par la fonction ERROR3270%.

Syntaxe	Déclaration interne	Description
ERR3270_NOERROR%	0	Fonction exécutée avec succès.
ERR3270_DATAERROR%	6	Une erreur de donnée a été diagnostiquée et est due à un paramètre invalide.
ERR3270_INVALIDPARAM%	2	Paramètre spécifié erroné.
ERR3270_INVALIDPSPOS%	7	La position Presentation Space spécifiée est invalide.

ERR3270_NOTCONNECTED%	1	Nom de session invalide ou application non connectée.
ERR3270_NOTEXECUTED%	5	L'exécution de la fonction n'a pu être effectuée pour une raison différente de celle décrite dans ERR3270_SCREENBUSY%.
ERR3270_RESNOTAVAILABLE%	11	La ressource demandée n'est pas disponible (par exemple une autre application est déjà connectée à l'écran).
ERR3270_SCREENBUSY%	4	L'exécution de la fonction n'a pu être effectuée car l'écran est occupé (par exemple X[] ou XSYSTEM est affiché dans l'OIA).
ERR3270_STRNOTFOUND%	24	Chaîne non trouvée ou écran non formaté.
ERR3270_SYSTEMERROR%	9	Une erreur système est arrivée. Une émulation 3270 peut s'arrêter alors que le programme s'exécute.
ERR3270_TIMEOUT%	1	Paramètre invalide ou timeout.

Ces valeurs ne sont pas exhaustives, mais représentent les codes d'erreur communs aux fonctions d'interface EHLLAPI définies par IBM. Cela implique aussi qu'à une valeur peut correspondre plusieurs causes d'erreur selon le contexte.

Il est possible que, dans un contexte particulier, une erreur ayant une autre valeur que celles indiquées soit retournée. Nous vous conseillons donc de toujours tester si le code d'accès rendu n'est pas ERR3270_NOERROR% avant de comparer celui ci avec les autres constantes ERR3270_*%.

Voir aussi ***ERROR3270%***

Fonction ERROR3270% (Librairie EHLLAPI NS3270)

Retourne le code d'erreur et remet à zéro le code d'erreur interne.

Syntaxe	**ERROR3270%**
Valeur retournée	INT(2)

1. Il est conseillé de tester le code d'erreur après chaque appel à une fonction de la librairie EHLLAPI.
2. La plupart des codes d'erreur proviennent du code de retour original de EHLLAPI.
3. Les valeurs possibles pour le code d'erreur sont identifiées par les constantes ERR3270_*%.

Voir aussi ***ERR3270_*%***

Gérer l'Operator Information Area

Fonction GETOIA3270$ (Librairie EHLLAPI NS3270)

Retourne le contenu de la ligne système (Operator Information Area ou OIA).

Syntaxe	**GETOIA3270$**
Valeur retournée	CSTRING

1. La chaîne retournée a 80 caractères de long.
2. Cette chaîne peut être utilisée pour analyser les informations systèmes affichées dans la ligne système.
3. Référez vous à la documentation IBM EHLLAPI Programming Reference ou du fournisseur de l'émulateur 3270 pour la description du contenu de l'OIA.

Voir aussi EBCDIC2ASCII$ (librairie NSMISC)

Fonction GETX3270% (Librairie EHLLAPI NS3270)

Retourne la valeur du statut X de la ligne de statut (Operator Information Area ou OIA).

Syntaxe	**GETX3270%**
Valeur retournée	INT(2)

La valeur retournée est l'une des constantes X3270_*%.

Voir aussi Constantes X3270_*%

Constantes X3270_*% (Librairie EHLLAPI NS3270)

Codes retour de la fonction GETX3270%.

Syntaxe	Déclaration interne	Description
X3270_CLOCK%	1	Temps demandé par le host
X3270_COMM%	4	Problèmes avec la ligne de communication.
X3270_MACH%	3	Problèmes avec le terminal
X3270_NOFUNC%	5	Fonction non disponible. Pressez sur RESET
X3270_NOPRT%	9	L'imprimante associée au terminal ne fonctionne pas
X3270_NOSYMBOL%	11	Caractère graphique non disponible. Pressez sur RESET.
X3270_OVERFL%	6	Trop de données insérées dans le champ
X3270_PROTECT%	8	Un champ protégé ne peut être modifié
X3270_PRTBUSY%	10	L'imprimante associée au terminal est utilisée

X3270_READY%	0	Pas de X. Le host a donné la main au terminal.
X3270_SYSTEM%	2	Clavier bloqué par le host. Attendez ou pressez sur RESET
X3270_UNDERFL%	7	Pas assez de données insérées dans le champ
X3270_UNKNOWN%	99	Statut non interprété

Voir aussi ***GETX3270%***

Récupérer les dimensions de l'écran 3270

Fonction GETHEIGHT3270% (Librairie EHLLAPI NS3270)

Retourne la hauteur de l'écran 3270, en nombre de lignes.

Syntaxe	**GETHEIGHT3270%**
Valeur retournée	INT(2)

La ligne de statut n'est pas comptabilisée.
Exemple :

```
MOVE GETHEIGHT3270% TO I% ; I% peut valoir par exemple 24
```

Voir aussi ***GETWIDTH3270%***

Fonction GETWIDTH3270% (Librairie EHLLAPI NS3270)

Retourne la largeur de l'écran 3270, en nombre de colonnes.

Syntaxe	**GETWIDTH3270%**
Valeur retournée	INT(2)

Exemple :

```
MOVE GETWIDTH3270% TO I% ; I% peut valoir par exemple 80
```

Voir aussi ***GETHEIGHT3270%***

Gérer les lignes et les champs de l'écran 3270

Fonction GETFIELD3270$ (Librairie EHLLAPI NS3270)

Retourne une chaîne qui contient le texte d'un champ affiché à l'écran.

Syntaxe	**GETFIELD3270$** *(colonne, ligne)*			
Paramètres	colonne	INT(2)	I	numéro de colonne du champ

	ligne	INT(2)	I	numéro de ligne du champ
Valeur retournée	CSTRING			

Voir aussi ***GETFIELDATTR3270%, GETFIELDLENGTH3270%***

Fonction GETFIELDATTR3270% (Librairie EHLLAPI NS3270)

Retourne les attributs d'un champ affiché à l'écran.

Syntaxe	**GETFIELDATTR3270%** *(colonne, ligne)*			
Paramètres	colonne	INT(2)	I	numéro de colonne du champ
	ligne	INT(2)	I	numéro de ligne du champ
Valeur retournée	INT(2)			

La signification des bits significatifs de l'entier retourné est la suivante (le bit 0 étant le bit de poids faible) :

```
bit 2
0 = Non protégé, 1 = Protégé

bit 3
0 = Alphanumérique, 1 = Numérique

bits 4,5
00 = Affiché, non détectable par light-pen
01 = Affiché, détectable par light-pen
10 = Affiché en surbillance, détactable par light-pen
11 = Non affiché, non détectable

bit 7
0 = Champ non modifié, 1 = Champ modifié par opérateur
```

Exemple :

```
;En supposant que ATTR% soit l'entier retourné par la fonction
GETFIELDATTR3270% (colonne,ligne), le test des attributs peut se faire
ainsi :
if (ATTR% BAND 4) = 0 ;Test si champ non protégé
...
endif
if (ATTR% BAND 8) <> 0 ;Test si champ numérique
...
endif
if (ATTR% BAND 48) = 48 ;Test si champ non affiché
...
endif
if (ATTR% BAND 128) <> 0 ;Test si champ modifié
...
endif
```

Voir aussi ***GETFIELD3270$, GETFIELDLENGTH3270%***

Fonction GETFIELDLENGTH3270% (Librairie EHLLAPI NS3270)

Retourne la longueur d'un champ affiché à l'écran.

Syntaxe	**GETFIELDLENGTH3270%** *(colonne, ligne)*			
Paramètres	colonne	INT(2)	I	numéro de colonne du champ
	ligne	INT(2)	I	numéro de ligne du champ
Valeur retournée	INT(2)			

Voir aussi ***GETFIELD3270$***, ***GETFIELDATTR3270%***

Fonction GETFIRSTFIELD3270% (Librairie EHLLAPI NS3270)

Retourne le premier champ affiché à l'écran 3270.

Syntaxe	**GETFIRSTFIELD3270%**
Valeur retournée	INT(4)

La valeur retournée est un entier dont la partie basse (LOW) donne le numéro de colonne, et la partie haute (HIW) le numéro de ligne.
Exemple :

```
; Pour compter le nombre de champs :
MOVE 0 TO NB_FIELDS%
MOVE GETFIRSTFIELD3270% TO F%
WHILE F% <> 0
MOVE NB_FIELD%+1 TO NB_FIELD%
MOVE GETNEXTFIELD3270% (F%) TO F%
ENDWHILE

; Pour extraire la colonne et la ligne
MOVE GETFIRSTFIELD3270% TO F%
MOVE LOW F% TO COLONNE%
MOVE HIW F% TO LIGNE%
```

Voir aussi ***GETNEXTFIELD3270%***, ***GETPREVFIELD3270%***

Fonction GETLINE3270$ (Librairie EHLLAPI NS3270)

Retourne le texte de la ligne affichée à l'écran.

Syntaxe	**GETLINE3270$** *(ligne)*			
Paramètre	ligne	INT(2)	I	numéro de la ligne à lire
Valeur retournée	CSTRING			

Le numéro de ligne est compris entre 1 et généralement 24.

Fonction GETNEXTFIELD3270% (Librairie EHLLAPI NS3270)

Retourne les coordonnées du champ qui suit un champ donné de l'écran 3270.

Syntaxe	**GETNEXTFIELD3270%** *(coord-champ)*			
Paramètre	coord-champ	INT(4)	I	coordonnées du champ courant
Valeur retournée	INT(4) nulle : il n'y a pas de champ suivant non nulle : coordonnées du champ suivant			

1. coord-champ est obtenu par un appel précédent à GETFIRSTFIELD% ou GETNEXTFIELD%
2. La valeur retournée est un entier dont la partie basse (LOW) donne le numéro de colonne, et la partie haute (HIW) le numéro de ligne.

Exemple :

```
; Pour compter le nombre de champs
MOVE 0 TO NB_FIELDS%
MOVE GETFIRSTFIELD3270% TO F%
WHILE F% <> 0
MOVE NB_FIELD%+1 TO NB_FIELD%
MOVE GETNEXTFIELD3270% (F%) TO F%
ENDWHILE

; Pour extraire la colonne et la ligne
MOVE GETFIRSTFIELD3270% TO F%
MOVE LOW F% TO COLONNE%
MOVE HIW F% TO LIGNE%
```

Voir aussi GETPREVFIELD3270%, GETFIRSTFIELD3270%

Fonction GETPREVFIELD3270% (Librairie EHLLAPI NS3270)

Retourne les coordonnées du champ qui précède un champ donné de l'écran 3270.

Syntaxe	**GETPREVFIELD3270%** *(coord-champ)*			
Paramètre	coord-champ	INT(4)	I	coordonnées du champ courant
Valeur retournée	INT(4) nulle : il n'y a pas de champ précédent non nulle : coordonnées du champ précédent			

1. coord-champ est obtenu par un appel précédent à GETPREVFIELD%.
2. La valeur retournée est un entier dont la partie basse (LOW) donne le numéro de colonne, et la partie haute (HIW) le numéro de ligne.

Voir aussi GETNEXTFIELD3270%, GETFIRSTFIELD3270%

Chercher une chaîne de caractères

Fonction SEARCHSTRING3270% (Librairie EHLLAPI NS3270)

Recherche une chaîne de caractères dans l'écran 3270.

Syntaxe	**SEARCHSTRING3270%** *(chaîne)*			
Paramètre	chaîne	CSTRING	I	chaîne recherchée
Valeur retournée	INT(2) nulle : aucune chaîne n'a été trouvée non nulle : index de la première chaîne trouvée (1 pour la première chaîne, 2 pour la deuxième, etc...).			

1. La chaîne passée en paramètre peut contenir plusieurs chaînes, ce qui permet en un seul appel de chercher l'apparition d'une chaîne parmi plusieurs. Le format de la chaîne doit être : "colonne,ligne,chaîne^colonne,ligne,chaîne^..."
2. Si une colonne ou une ligne est nulle, ce critère colonne ou ligne n'est pas utilisé.
3. La chaîne cherchée au sein de la chaîne passée en paramètre peut aussi être de la forme #XXC (impérativement quatre caractères) afin de chercher une chaîne composée de XX caractères C. Ainsi, les deux lignes ci dessous sont rigoureusement équivalentes :

```
MOVE SEARCHSTRING3270% ("10,5,#05M") TO I%
MOVE SEARCHSTRING3270% ("10,5,MMMMM") TO I%
```

Exemple :

```
IF SEARCHSTRING3270% ("3,5,Coucou") = 1
MESSAGE "Trouvé !", "Coucou est affiché colonne 3, ligne 5"
ENDIF

MOVE SEARCHSTRING3270% ("12,13,Bonjour^0,5,32^0,0,Test") TO I%
; Si 0 est retourné, aucune chaîne n'a été trouvée
; Si 1, la chaîne "Bonjour" a été trouvée en col 12 ligne 13
; Si 2, la chaîne "Bonjour" n'a pas été trouvée, alors que "32" a été
trouvée en ligne 5 (colonne quelconque)
; Si 3, les chaînes "Bonjour" et "32" n'ont pas été trouvés, alors que
"Test" a été trouvé a un endroit quelconque de l'écran
```

Gérer le curseur 3270

Fonction GETXCURSOR3270% (Librairie EHLLAPI NS3270)

Retourne la colonne où est positionné le curseur.

Syntaxe	**GETXCURSOR3270%**
Valeur retournée	INT(2)

Le coin supérieur gauche de l'écran est à la colonne 1, ligne 1.

***Voir aussi** GETYCURSOR3270%, SETCURSOR3270, WAITFORCURSOR3270, WAITFORNOCURSOR3270*

Fonction GETYCURSOR3270% (Librairie EHLLAPI NS3270)

Retourne la ligne où est positionné le curseur.

Syntaxe	**GETYCURSOR3270%**
Valeur retournée	INT(2)

Le coin supérieur gauche de l'écran est à la colonne 1, ligne 1.

***Voir aussi** GETXCURSOR3270%, SETCURSOR3270, WAITFORCURSOR3270, WAITFORNOCURSOR3270*

Instruction SETCURSOR3270 (Librairie EHLLAPI NS3270)

Positionne le curseur à la ligne et colonne spécifiées.

Syntaxe	**SETCURSOR3270** *colonne, ligne*			
Paramètres	colonne	INT(2)	I	numéro de colonne (entre 1 et généralement 80)
	ligne	INT(2)	I	numéro de ligne (entre 1 et généralement 24)

***Voir aussi** GETXCURSOR3270%, GETYCURSOR3270%, WAITFORCURSOR3270, WAITFORNOCURSOR3270*

Entrer des données dans un buffer d'écran 3270

Instruction TYPESTRING3270 (Librairie EHLLAPI NS3270)

Simule la saisie d'une chaîne de caractères au clavier 3270.

Syntaxe	**TYPESTRING3270** *chaîne-saisie*			
Paramètre	chaîne-saisie	CSTRING	I	chaîne de caractères simulée

1. Les caractères sont entrés dans le buffer 3270 à la position courante du curseur, qui devrait être au début d'un champ.
2. Il est possible d'insérer dans la chaîne des touches de fonction en faisant précéder le code représentant la fonction par le caractère '@'. Voici l'ensemble des touches de fonctions disponibles :

Code	Touche de fonction
@@	@
@T	Tab
@C	Clear
@D	Delete
@E	Enter
@F	Erase EOF

@J	Jump to destination LT
@B	Backtab
@L	Cursor Left
@N	New Line
@P	Print Screen
@R	Reset
@$	Alternate cursor
@I	Insert
@U	Cursor Up
@V	Cursor Down
@Z	Cursor Right
@<	Backspace
@0	Home
@1	PF1
@2	PF2
...	...
@9	PF9
@a	PF10
@b	PF11
...	...
@o	PF24
@x	PA1
@y	PA2
@z	PA3
@A@F	Erase Input
@A@H	System Request
@A@J	Cursor Select
@A@1	Reset host colors
@A@Q	Attention
@A@L	Cursor left fast
@A@Z	Cursor right fast
@A@d	Red
@A@e	Pink
@A@f	Green

@S@x	Dup
@S@y	Field Mark
@A@b	Underscore
@A@c	Field highlight
@A@g	Yellow
@A@h	Blue
@A@i	Turquoise
@A@j	White
@A@y	Forward word tab
@A@z	Backwards word tab
@A@9	Reverse video

Exemple :

```
; Entre le texte 'Hello world' suivi de la touche ENTER
TYPESTRING3270 "Hello world@E"
```

Voir aussi ***PRESSENTER3270***, ***PRESSKEY3270***, ***PRESSPFKEY3270***, ***WRITEFIELD3270***

Instruction WRITEFIELD3270 (Librairie EHLLAPI NS3270)

Simule la frappe d'une chaîne de caractères au clavier 3270 dans un champ spécifié par sa position.

Syntaxe	**WRITEFIELD3270** *chaîne, colonne, ligne*			
Paramètres	chaîne	INT(2)	I	chaîne de caractères simulée
	colonne	INT(2)	I	numéro de colonne du champ
	ligne	INT(2)	I	numéro de ligne du champ

1. La position spécifiée par colonne et ligne peut correspondre à n'importe quelle partie d'un champ.
2. Contrairement à l'instruction TYPESTRING3270, il n'est pas nécessaire de positionner le curseur au début du champ. C'est pourquoi elle est très souvent utilisée en remplacement de cette instruction. Elle nécessite par contre d'avoir un écran formaté.

Exemple :

```
; Affiche PIERRE dans le champ à la colonne 12, ligne 15
WRITEFIELD3270 "PIERRE", 12, 15
```

Voir aussi ***TYPESTRING3270***

Simulations d'une frappe de touche

Constantes KEY3270_*% (Librairie EHLLAPI NS3270)

Touches prédéfinies 3270.

Syntaxe	Déclaration interne
KEY3270_ALTERNATE_CURSOR%	9
KEY3270_ATTENTION%	24
KEY3270_BACKSPACE%	15
KEY3270_BACKTAB%	5
KEY3270_CLEAR%	1
KEY3270_CURSOR_DOWN%	12
KEY3270_CURSOR_LEFT%	13
KEY3270_CURSOR_LEFT_FAST%	25
KEY3270_CURSOR_RIGHT%	14
KEY3270_CURSOR_RIGHT_FAST%	26
KEY3270_CURSOR_SELECT%	22
KEY3270_CURSOR_UP%	11
KEY3270_DELETE%	2
KEY3270_ENTER%	3
KEY3270_ERASE_EOF%	4
KEY3270_ERASE_INPUT%	20
KEY3270_HOME%	16
KEY3270_INSERT%	10
KEY3270_NEWLINE%	6
KEY3270_PA1%	17
KEY3270_PA2%	18
KEY3270_PA3%	19
KEY3270_PRINT_SCREEN%	7
KEY3270_RESET%	8
KEY3270_RESET_HOST_COLORS%	23
KEY3270_SYSTEM_REQUEST%	21
KEY3270_TAB%	0

***Voir aussi** PRESSKEY3270, TYPESTRING3270*

Instruction PRESSENTER3270 (Librairie EHLLAPI NS3270)

Simule l'appui de la touche ENTER du clavier 3270.

Syntaxe	**PRESSENTER3270**

Cette opération peut aussi être effectuée avec les instructions suivantes :

```
PRESSKEY3270 KEY3270_ENTER%
```

ou

```
TYPESTRING3270 "@E"
```

***Voir aussi** PRESSKEY3270, PRESSPFKEY3270, TYPESTRING3270*

Instruction PRESSKEY3270 (Librairie EHLLAPI NS3270)

Simule l'appui d'une touche de fonction spécifique au clavier 3270 comme les touches ENTER, TABULATION, CLEAR, etc.

Syntaxe	**PRESSKEY3270** *touche*			
Paramètre	touche	INT(2)	I	numéro de la touche simulée

1. Les touches disponibles sont celles décrites par les constantes KEY3270_*%
2. Il est possible d'appeler l'instruction TYPESTRING3270 pour faire la même opération.

***Voir aussi Constantes** KEY3270_*%, PRESSENTER3270, PRESSPFKEY3270, TYPESTRING3270*

Instruction PRESSPFKEY3270 (Librairie EHLLAPI NS3270)

Simule l'appui d'une touche PF.

Syntaxe	**PRESSPFKEY3270** *touche-PF*			
Paramètre	touche-PF	INT(2)	I	numéro de la touche PF simulée

1. L'expression passée est une expression entière prenant les valeurs 1 à 24.
2. Il est possible d'appeler l'instruction TYPESTRING3270 pour faire la même opération.

***Voir aussi** PRESSENTER3270, PRESSKEY3270, TYPESTRING3270*

Synchronisation

Instruction WAITFORCURSOR3270 (Librairie EHLLAPI NS3270)

Attend jusqu'à ce que le curseur soit positionné à une position donnée.

Syntaxe	**WAITFORCURSOR3270** *colonne, ligne, délai*			
Paramètres	colonne	INT(2)	I	numéro de colonne (entre 1 et

				généralement 80)
	ligne	INT(2)	I	numéro de ligne (entre 1 et généralement 24)
	délai	INT(4)	I	durée d'attente maximum

1. La durée d'attente maximum est exprimée en 1/10 de seconde.
2. Si délai vaut zéro, le délai est infini : l'instruction ne retourne pas tant que la condition d'attente n'est pas réalisée.

Exemple :

```
; Positionne le curseur à la colonne et ligne 1,1
SETCURSOR3270 1, 1

; Appui sur la touche ENTER pour changer d'écran
PRESSENTER3270

; Attend pour continuer que le curseur soit positionné en 10,10
WAITFORCURSOR3270 10, 10, 20

; Teste si la condition est bien vérifiée
IF ERROR3270% <> 0
... ; Time-out ...
ENDIF
```

***Voir aussi** WAITFORNOCURSOR3270, GETXCURSOR3270%, GETYCURSOR3270%, SETCURSOR3270*

Instruction WAITFORNOCURSOR3270 (Librairie EHLLAPI NS3270)

Attend jusqu'à ce que le curseur ne soit plus positionné à une position donnée.

Syntaxe	**WAITFORNOCURSOR3270** *colonne, ligne, délai*			
Paramètres	colonne	INT(2)	I	numéro de colonne (entre 1 et généralement 80)
	ligne	INT(2)	I	numéro de ligne (entre 1 et généralement 24)
	délai	INT(4)	I	durée d'attente maximum

1. La durée d'attente maximum est exprimée en 1/10 de seconde.
2. Si délai vaut zéro, le délai est infini : l'instruction ne retourne pas tant que la condition d'attente n'est pas réalisée.

Exemple :

```
; Positionne le curseur à la colonne et ligne 1,1
SETCURSOR3270 1, 1

; Appui sur la touche ENTER pour changer d'écran
PRESSENTER3270

; Attend pour continuer que le curseur ait changé de position
WAITFORNOCURSOR3270 1, 1, 20

; Teste si la condition est bien vérifiée
IF ERROR3270% <> 0
... ; Time out ...
ENDIF
```

***Voir aussi** WAITFORCURSOR3270, GETXCURSOR3270%, GETYCURSOR3270%, SETCURSOR3270*

Instruction WAITFORNOSTRING3270 (Librairie EHLLAPI NS3270)

Attend jusqu'à ce qu'une chaîne de caractères donnée ne soit plus affichée à l'écran.

Syntaxe	**WAITFORNOSTRING3270** *chaîne, colonne, ligne, délai*			
Paramètres	chaîne	CSTRING	I	chaîne recherchée
	colonne	INT(2)	I	numéro de colonne (entre 1 et généralement 80)
	ligne	INT(2)	I	numéro de ligne (entre 1 et généralement 24)
	délai	INT(4)	I	durée d'attente maximum

1. La durée d'attente maximum est exprimée en 1/10 de seconde.
2. Si délai vaut zéro, le délai est infini : l'instruction ne retourne pas tant que la condition d'attente n'est pas réalisée.

Exemple :

```
; Appui sur la touche ENTER pour changer d'écran
PRESSENTER3270

; Attend pour continuer que la chaîne ne soit plus affichée
WAITFORNOSTRING3270 "NOM", 1, 1, 20
```

```
; Teste si la condition est bien vérifiée
IF ERROR3270% <> 0
... ; Time-out
ENDIF
```

Voir aussi ***WAITFORSTRING3270***

Instruction WAITFORNOX3270 (Librairie EHLLAPI NS3270)

Attend que le statut X ait disparu pendant un certain temps de l'Operator Information Area (OIA).

Syntaxe	**WAITFORNOX3270** *délai, time out*			
Paramètres	délai	INT(4)	I	durée de disparition
	time out	INT(4)	I	durée d'attente maximum

1. délai et time out sont exprimés en 1/10s.
2. Si le X est absent pendant le délai délai, l'instruction retourne positivement (ERROR3270% est nul) sans attendre le time out.
3. L'instruction essaye le délai délai plusieurs fois, tant que le X est aperçu, mais au maximum jusqu'à time out. Si le time out a été atteint, ce qui signifie que le X a été aperçu au moins une fois à chaque délai essayé, l'instruction retourne un code d'erreur dans ERROR3270%.
4. Si time out vaut zéro, il y a un time out infini : l'instruction ne retourne pas tant que le X n'a pas disparu pendant un temps précisé par le délai.
5. Ces deux temps en paramètre sont nécessaires du fait que le statut X peut clignoter, ce qui peut rendre difficile la certitude de sa disparition. L'échantillonage des essais est toutefois assez rapide pour garantir que pendant délai indiqué, le statut X n'est jamais apparu.

Exemple :

```
; Appui sur la touche ENTER pour changer d'écran
PRESSENTER3270

; Attend pour continuer que le X statut disparaisse pendant au moins 2/10
de secondes
; La durée maximum de l'instruction est de 3 secondes
WAITFORNOX3270 2, 30

; Teste si la condition est bien vérifiée
IF ERROR3270% <> 0
... ; Time-out
ENDIF
```

Instruction WAITFORSTRING3270 (Librairie EHLLAPI NS3270)

Attend jusqu'à ce qu'une chaîne de caractères donnée soit affichée à une position donnée.

Syntaxe	**WAITFORSTRING3270** *chaîne, colonne, ligne, délai*			
Paramètres	chaîne	CSTRING	I	chaîne recherchée
	colonne	INT(2)	I	numéro de colonne (entre 1 et généralement 80)
	ligne	INT(2)	I	numéro de ligne (entre 1 et généralement 24)
	délai	INT(4)	I	durée d'attente maximum

1. La durée d'attente maximum est exprimée en 1/10 de seconde.
2. Si délai vaut zéro, le délai est infini : l'instruction ne retourne pas tant que la condition d'attente n'est pas réalisée.

Exemple :

```
; Appui sur la touche ENTER pour changer d'écran
PRESSENTER3270

; Attend pour continuer que la chaîne soit affichée
WAITFORSTRING3270 "NOM", 1, 1, 20

; Teste si la condition est bien vérifiée
IF ERROR3270% <> 0
... ; Time-out
ENDIF
```

Voir aussi ***WAITFORNOSTRING3270***

LIBRAIRIE DE COMMUNICATIONS NSCOMM

Cette librairie permet de gérer les communications entre applications via les queues, le presse papiers (clipboard) ou le DDE (Dynamic Data Exchange).

Installation

Déclarez NSCOMM.NCL dans les librairies nécessaires au développement de votre application.
Vérifier que le fichier NSxxCOMM.DLL est bien dans un des répertoires du PATH sous Windows.

> NSxxCOMM.DLL contient le code de deux librairies NCL : NSCOMM.NCL et NSAPPC.NCL.

Extensions du langage NCL pour les Queues

Les queues permettent de communiquer entre applications selon le schéma suivant :

- Une application unique crée la queue et reçoit les données (lecture de la queue). Cette application est nommé CREATEUR.
- Une ou plusieurs autres applications envoient les données (écriture dans la queue). Ces applications sont nommées EXPEDITEURS.

Pour être exact, le CREATEUR peut aussi écrire dans la queue, mais comme il est le seul à pouvoir y lire, cela est peu utilisé.
Les queues sont de type FIFO (First In First Out) : le CREATEUR lit l'élément le plus ancien de la queue, ce qui lui permet d'obtenir les éléments dans le même ordre qu'ils ont été écrits.
Pour que la communication se fasse correctement, il faut que le format des éléments de la queue soit identique entre CREATEUR et EXPEDITEUR(s). Le format des éléments doit donc être défini à l'aide d'un segment NCL.
Le rôle de chaque fonction et instruction de gestion des queues est le suivant :
INITQUEUE initialise et STOPQUEUE termine l'utilisation des queues.
CREATEQUEUE% crée la queue pour le CREATEUR, et OPENQUEUE% ouvre cette même queue pour un EXPEDITEUR.
CLOSEQUEUE ferme une queue.
WRITEONQUEUE écrit dans la queue (utilisable par EXPEDITEUR et CREATEUR).
READFROMQUEUE lit dans la queue (utilisable uniquement par CREATEUR).
PEEKFROMQUEUE% permet de connaître la taille du prochain élément de la queue, en attente de lecture (utilisable uniquement par CREATEUR).
ERRORQUEUE% retourne le code d'erreur de la dernière fonction ou instruction de queue effectuée.

Utilisation classique des queues, côté créateur

```
;Evénement INIT du CREATEUR
SEGMENT INDIVIDUAL
STRING LAST_NAME(31)
STRING FIRST_NAME(31)
...
ENDSEGMENT
INITQUEUE
MOVE CREATEQUEUE%("QUEUENAME") TO HQ%
NEW INDIVIDUAL, MYCLIENT%
;Evénement XXX du CREATEUR
READFROMQUEUE HQ%, MYCLIENT%, SIZEOF INDIVIDUAL
IF ERRORQUEUE% <> 0
MESSAGE "Error #" && ERRORQUEUE%, "READFROMQUEUE"
EXIT
ENDIF
MOVE INDIVIDUAL(MYCLIENT%).LAST_NAME TO ...
MOVE INDIVIDUAL(MYCLIENT%).FIRST_NAME TO ...
Evénement TERMINATE du CREATEUR
DISPOSE MYCLIENT%
CLOSEQUEUE HQ%
STOPQUEUE
```

Utilisation classique des queues, côté expéditeur

```
;Evénement INIT d'un EXPEDITEUR
; Same segment as the one used by the Owner
SEGMENT INDIVIDUAL
STRING LAST_NAME(31)
STRING FIRST_NAME(31)
...
ENDSEGMENT
INITQUEUE
; Same queue name as the Owner
MOVE OPENQUEUE%("QUEUENAME") TO HQ%
NEW INDIVIDUAL, MYCLIENT%
;Evénement XXX d'un EXPEDITEUR
MOVE ... TO INDIVIDUAL(MYCLIENT%).LAST_NAME
MOVE ... TO INDIVIDUAL(MYCLIENT%).FIRST_NAME
WRITEONQUEUE HQ%, MYCLIENT%, SIZEOF INDIVIDUAL
IF ERRORQUEUE% <> 0
MESSAGE "Error #" && ERRORQUEUE%, "WRITEONQUEUE"
ENDIF
;Evénement TERMINATE d'un EXPEDITEUR
DISPOSE MYCLIENT%
CLOSEQUEUE HQ%
STOPQUEUE
```

Conseils d'utilisation des queues

Le ou les EXPEDITEURS doivent être démarrés après le CREATEUR, afin d'empêcher des écritures intempestives dans une queue qui n'existerait pas encore.
Le ou les EXPEDITEURS doivent être arrêtés avant le CREATEUR, afin d'empêcher des écritures intempestives dans une queue qui n'existerait plus.

Précautions à prendre :

- READFROMQUEUE ne retourne pas tant qu'il n'y a rien dans la queue, ce qui peut bloquer l'application et même le système. Utiliser préalablement PEEKFROMQUEUE% pour connaître l'état de la queue.
- Tester des applications communiquant par queues est difficile à faire depuis l'éditeur de fenêtre, puisque ce dernier ne permet de démarrer qu'une seule fenêtre. De plus le testeur n'est pas multi instances : il n'est donc pas possible de le lancer plusieurs fois de suite pour tester simultanément EXPEDITEURS et CREATEUR. La seule solution consiste alors à générer tous les modules sauf celui qui doit être testé à l'aide du module de test.

Par exemple, pour tester un CREATEUR dont la fenêtre s'appelle CREATEUR.SCR et deux EXPEDITEURS dont les fenêtres s'appellent EXPED1.SCR et EXPED2.SCR, il faut :

1. Générez les modules EXPEDITEURS.
2. Démarrez CREATEUR.SCR depuis l'éditeur de fenêtre, ou tapez sur la ligne de commande obtenue à l'aide de la commande File/Run : NSTEST CREATEUR. Cette étape doit impérativement être faite avant les autres, pour que la queue soit correctement créée :
   a) Démarrez EXPED1 depuis la ligne de commande.
   b) Démarrez EXPED2 depuis la ligne de commande.

Tester des applications communiquant par queues est difficile à faire depuis l'éditeur de fenêtre, puisque ce dernier ne permet de démarrer qu'une seule fenêtre.

Pour cela, il faut directement utiliser le module de test. Par exemple, pour tester un CREATEUR dont la fenêtre s'appelle CREATEUR.SCR et deux EXPEDITEURS dont les fenêtres s'appellent EXPED1.SCR et EXPED2.SCR, il faut démarrer CREATEUR.SCR depuis l'éditeur de fenêtre et taper : NSTEST CREATEUR

Cette étape doit impérativement être faite avant les autres, pour que la queue soit correctement créée.

Extensions du langage NCL pour le DDE

Description générale du DDE

Le Dynamic Data Exchange, ou DDE, est un protocole d'échange de données entre applications, basé sur des événements spécifiques DDE_*.

Contrairement au Presse Papiers qui est un mécanisme "statique" demandant une action de l'utilisateur, le DDE est un mécanisme "dynamique" ne nécessitant pas l'action de l'utilisateur (sauf éventuellement pour démarrer l'échange) : ainsi lorsqu'une donnée change au sein d'une application serveur DDE, les applications client DDE peuvent être averties de ce changement et récupérer la nouvelle donnée de façon complètement automatique.

De nombreuses applications bureautiques supportent aujourd'hui ce protocole. Pour une meilleure compréhension du DDE, nous citerons souvent Excel qui est un excellent exemple d'application DDE, en tant que client comme en tant que serveur.

Client, Serveur et Conversation

Une "Conversation" DDE est un lien entre deux applications par lequel transite des événements respectant le protocole DDE.
L'application qui initie la conversation est le "client". L'application qui répond au client est le "serveur".
Une même application peut être engagée dans plusieurs conversations au même moment, et peut être client dans certaines conversations et serveur dans d'autres.

> Il ne peut y avoir qu'une seule conversation DDE entre deux fenêtres. Une application client ou serveur doit avoir une fenêtre différente pour chacune de ses conversations avec une même fenêtre de l'application serveur ou client correspondante. Pour être sûr de cette unicité, il suffit d'ouvrir une fenêtre cachée pour chaque conversation, le but de cette fenêtre étant uniquement de traiter les événements DDE.

Application, Topic et Item

Une conversation DDE est définie de façon unique par l'ensemble application + topic.
L'application est le nom de l'application serveur. Par exemple "Excel".
Le topic rassemble un ensemble de données. Pour des applications MDI comme Excel ou Word, le topic représente une des feuilles de calcul ou un des documents en cours d'édition. Par exemple "Sheet1".
L'item est une donnée (par exemple une cellule Excel) dont la valeur peut être échangée entre le client et le serveur. Par exemple "R1C1".

Topic System

Le protocole DDE recommande que tout serveur DDE ait, en plus de ses topics qui lui sont propres, un topic spécial nommé "System" permettant d'obtenir des informations générales sur certaines caractéristiques du serveur grâce aux items suivants :

- SysItems indique la liste des items du topic System.
- Topics indique la liste des topics du serveur,
- Status indique si le serveur est disponible ou indisponible (vaut "Ready" ou "Busy")
- Formats indique la liste des formats du presse papiers supportés par le serveur.

De plus, chaque serveur peut ajouter tout item système pouvant être utile à ses clients. Ainsi, Excel propose plusieurs items système supplémentaires, dont "Selection" qui indique le nom de la feuille active (topic) ainsi que les références de la ou des cellules actuellement sélectionnées (item) dans cette feuille.

Description générale d'une conversation DDE

Voici en quoi consiste une conversation DDE typique :

1. Le client initie la conversation et le serveur répond.

2. Des échanges de données sont effectués selon différentes méthodes :
   a) le serveur envoie une donnée au client, sur requête du client (Request),
   b) le client envoie une donnée au serveur (Poke), sans requête du serveur,
   c) après requête initiale du client (Advise), le serveur envoie automatiquement une donnée au client à chaque fois que sa valeur change (Data), sans requête supplémentaire du client ; cet échange est nommé "lien permanent",
   d) le client envoie une commande au serveur (Execute), que ce dernier doit exécuter.
3. La conversation peut se terminer à la demande du client ou à la demande du serveur.

Les événements, fonctions et instructions DDE

Selon que l'application est client DDE ou serveur DDE, les événements à traiter sont différents et les fonctions / instructions à utiliser sont différentes.
Dans le cas où une application désire pouvoir être à la fois client et serveur (c'est par exemple le cas d'Excel), tous les événements sont bien sûr à traiter et toutes les fonctions / instructions peuvent être utilisées.
Noter qu'il est généralement plus facile d'écrire une application client qu'une application serveur.

L'événement TIMER est reçu, avec PARAM1% à 0, par toute application faisant du DDE (client ou serveur), pour les besoins internes de la librairie NSComm. Afin d'éviter les conflits, ne pas utiliser ce même TIMER avec un identifieur 0.

Lors de l'utilisation d'un événement DDE, ne pas utiliser le TIMER 1, car ce dernier est utilisé par le protocole DDE.

Programmation d'un client DDE

Les événements à gérer pour un client sont :

- DDE_INITACK : réponse positive du serveur à une conversation demandée par le client.
- DDE_DATA : réception de donnée, uniquement si un lien permanent a été préalablement demandé par le client et accepté par le serveur.
- DDE_TERMINATE : fin de conversation lorsqu'elle est demandée par le serveur.

Les fonctions et instructions DDE à utiliser pour un client sont :

- INITIATEDDE% initialise une conversation DDE.
- REQUESTDDE% (ou REQUESTTXTDDE%) demande une donnée au serveur : lecture.
- POKEDDE% (ou POKETXTDDE%) envoie une donnée au serveur : écriture.
- EXECUTEDDE% (ou EXECUTETXTDDE%) demande au serveur d'exécuter une commande.
- ADVISEDDE% demande un lien permanent au serveur. UNADVISEDDE termine un lient permanent.

- TERMINATEDDE termine une conversation. Cette instruction est aussi utilisable par un serveur.

Programmation d'un serveur DDE

Evénements à gérer pour un serveur :

- DDE_INIT : conversation demandée par un client.
- DDE_REQUEST, DDE_POKE, DDE_EXECUTE : lecture de donnée demandée par le client, écriture de donnée, exécution de commande.
- DDE_ADVISE, DDE_UNADVISE : début de lien permanent demandé par le client, fin de lien permanent.
- DDE_TERMINATE : fin de conversation lorsqu'elle est demandée par le client.

Fonctions et instructions DDE à utiliser pour un serveur :

- RESPONDDDE% répond positivement à une demande de conversation.
- ACKDDE accepte une demande (Execute ou Poke) du client.
- NACKDDE refuse une demande (Execute, Poke ou Request).
- DATADDE% (ou DATATXTDDE%) envoie une donnée au client.
- TERMINATEDDE termine une conversation. Cette instruction est aussi utilisable par un client.

La majorité des fonctions / instructions ont pour effet de générer un événement DDE vers l'application distante. Chaque événement DDE porte un nom semblable à la fonction ou instruction qui l'a généré.

Exemple de conversation DDE

Initialisation de la conversation DDE

Application Client	Application Serveur
Evt XXX INITIATEDDE%	Evt DDE_INIT
Evt DDE_INITACK RETURN 0	RESPONDDDE% provoque la fin des fonctions INITIATEDDE% et RESPONDDDE% qui retournent un Handle de conversation DDE

Demande de lecture d'une donnée

Application Client	Application Serveur

Evt XXX REQUESTTXTDDE%	Evt DDE_REQUEST DATATXTDDE% provoque la fin de la fonction REQUESTTXTDDE% qui retourne TRUE%, et dans son dernier paramètre la donnée demandée.

Demande d'écriture d'une donnée

Application Client	Application Serveur
Evt XXX POKETXTDDE%	Evt DDE_POKE donnée dans segment DDEDATA pointé par PARAM34%. ACKDDE provoque la fin de la fonction POKETXTDDE% qui retourne TRUE%.

Demande d'exécution d'une commande

Application Client	Application Serveur
Evt XXX EXECUTETXTDDE%	Evt DDE_EXECUTE commande dans segment DDEDATA pointé par PARAM34%. ACKDDE provoque la fin de la fonction EXECUTETXTDDE% qui retourne TRUE%.

Fin de la conversation DDE

Application Client	Application Serveur
Evt XXX TERMINATEDDE	Evt DDE_TERMINATE

OU

Application Client	Application Serveur
Evt DDE_TERMINATE	Evt XXX TERMINATEDDE

Exemple de conversation DDE avec lien permanent

Demande de lien permanent sur un item

La conversation est supposée déjà établie, comme vu plus haut.

Application Client	Application Serveur
Evt XXX ADVISEDDE%	Evt XXX RETURN TRUE% provoque la fin de la fonction ADVISEDDE% qui retourne TRUE%.

Changements de l'item

Transmis par le serveur, réceptionnés par le client.

Application Client	Application Serveur
Evt XXX Evt DDE_DATA segment DDEDATA pointé par PARAM34%.	Evt XXX (Changement de la donnée) DATATXTDDE%
Evt DDE_DATA segment DDEDATA pointé par PARAM34%.	Evt XXX (Nouveau changement de la donnée) DATATXTDDE%

Fin de lien permanent sur un item

Application Client	Application Serveur
Evt XXX UNADVISEDDE	Evt DDE_UNADVISE

Extensions du langage NCL pour le Presse-Papiers

La programmation du presse papiers est extrêmement simple, avec seulement deux instructions et une fonction :

- READFROMCLIPBOARD lit le presse papiers et le stocke dans une variable NCL,
- WRITETOCLIPBOARD écrit dans le presse papiers à partir d'une variable NCL passée en paramètre,
- CLIPBOARDSIZE% retourne le nombre de caractères (= d'octets) contenus dans le presse papiers.

Rappelons que les raccourcis clavier du presse-papiers ([Shift]+[Del], [Ctrl]+[Ins], [Shift]+[Ins]) fonctionnent de façon automatique dans tous les contrôles Entry-Fields, CBE et MLE des fenêtres de classe Dialog, ainsi que dans l'aire client des fenêtres de classe Edit.

NSComm propose uniquement une gestion du mode texte du presse papiers. Ainsi, NSComm ne permet pas de copier du graphique dans le presse-papiers.

Autres méthodes de communication

D'autres méthodes de communication entre applications existent, utilisables en dehors de la librairie NSComm. Rappellons notamment :

- Mémoire partagée nommée, grâce au troisième paramètre (optionnel) de l'instruction NEW. ***Voir le manuel "Référence du langage NCL".***
- Envoi d'événements. ***Voir les Evénements Utilisateur dans le manuel "Référence du langage NCL", ainsi que la librairie NSWin (instruction POSTMESSAGE et fonction SENDMESSAGE%).***
- Attention cependant, ces fonctions sont fortement dépendantes de chaque système et réduisent la portabilité de votre application sur d'autres cibles.
- Fichiers disque. ***Voir la librairie NSMisc.***
- Communication APPC LU 6.2 pour applications distantes. ***Voir la librairie NSAPPC.***

Fichier NSCOMM.NCL

Les verbes de la librairie NSComm, décrits ci après, sont déclarés dans le fichier texte écrit en NCL, de nom NSCOMM.NCL. Ce fichier peut aussi contenir des verbes complémentaires (API non publique). Vous pouvez donc désirer consulter ce fichier pour avoir la référence exhaustive de la librairie.

Pour consulter le fichier NSCOMM.NCL :

1. Placez vous dans le répertoire <NATSTAR>\NCL ou <NSDK>\NCL

<NATSTAR> représente le répertoire que vous avez choisi au moment de l'installation de NatStar et <NSDK> celui de NS-DK.

2. Editez le fichier NSCOMM.NCL avec n'importe quel éditeur de texte.

Catégories fonctionnelles de la librairie NSComm

Gérer les files d'attente (queues)

Instruction CLOSEQUEUE (Librairie NSComm)

Ferme une queue.

Syntaxe	**CLOSEQUEUE** *handle queue*			
Paramètre	handle queue	INT(4)	I	handle de la queue à fermer

1. handle-queue est préalablement obtenu soit par CREATEQUEUE%, soit par OPENQUEUE%.
2. La queue est complètement effacée et détruite lorsque l'application qui effectue le CLOSEQUEUE a créé la queue par CREATEQUEUE%. Par contre, la queue existe toujours, et son contenu est inchangé, si l'application l'a simplement ouverte par OPENQUEUE%.
3. Suite à cette instruction, un éventuel code d'erreur peut être obtenu par ERRORQUEUE%.

4. Lorsque l'utilisation des queues est terminée au sein de l'application, c'est-à-dire après le dernier CLOSEQUEUE de l'application, il faut faire un STOPQUEUE.

Exemple 1 :

```
INITQUEUE
MOVE CREATEQUEUE% ("NOMQUEUE") TO HQ%
...
CLOSEQUEUE HQ%
IF ERRORQUEUE% <> 0
MESSAGE "Erreur n°" && ERRORQUEUE%, "CLOSEQUEUE"
ENDIF
STOPQUEUE
```

Exemple 2 :

```
INITQUEUE
MOVE OPENQUEUE% ("NOMQUEUE") TO HQ%
...
CLOSEQUEUE HQ%
IF ERRORQUEUE% <> 0
MESSAGE "Erreur n°" && ERRORQUEUE%, "CLOSEQUEUE"
ENDIF
STOPQUEUE
```

***Voir aussi** CREATEQUEUE%, OPENQUEUE%, ERRORQUEUE%, STOPQUEUE*

Fonction CREATEQUEUE% (Librairie NSComm)

Crée une queue et retourne son handle.

Syntaxe	**CREATEQUEUE%** *(nom-queue)*			
Paramètre	nom-queue	CSTRING	I	nom de la queue à créer
Valeur retournée	INT(4)			

1. Le handle retourné peut ensuite être utilisé pour lire dans la queue par READFROMQUEUE ou PEEKFROMQUEUE%.
2. Suite à cette fonction, un éventuel code d'erreur peut être obtenu par ERRORQUEUE%.
3. Avant le premier CREATEQUEUE% ou OPENQUEUE% de l'application, il est nécessaire d'activer l'utilisation des queues à l'aide de INITQUEUE.

Exemple :

```
INITQUEUE
MOVE CREATEQUEUE% ("NOMQUEUE") TO HQ%
IF ERRORQUEUE% <> 0
MESSAGE "Erreur n°" && ERRORQUEUE%, "CREATEQUEUE%"
ENDIF
...
READFROMQUEUE HQ%, ...
...
```

```
CLOSEQUEUE HQ%
STOPQUEUE
```

Voir aussi ***CLOSEQUEUE, READFROMQUEUE, PEEKFROMQUEUE%, OPENQUEUE%, INITQUEUE, ERRORQUEUE%***

Fonction ERRORQUEUE% (Librairie NSComm)

Retourne le code d'erreur de la dernière fonction ou instruction de queue effectuée.

Syntaxe	**ERRORQUEUE%**
Valeur retournée	INT(4)

1. Si la dernière fonction ou instruction s'était correctement effectuée, ERRORQUEUE% retourne 0.
2. Les fonctions ou instructions qui influent sur ERRORQUEUE% sont :
   - INITQUEUE, STOPQUEUE
   - CREATEQUEUE%, OPENQUEUE%, CLOSEQUEUE
   - PEEKFROMQUEUE%, READFROMQUEUE, WRITEONQUEUE
3. Les différents codes d'erreur retournés figurent dans le fichier NSCOMM.NCL.

Voir aussi ***INITQUEUE, STOPQUEUE, CREATEQUEUE%, OPENQUEUE%, CLOSEQUEUE, PEEKFROMQUEUE%, READFROMQUEUE, WRITEONQUEUE***

Instruction INITQUEUE (Librairie NSComm)

Initialise la gestion des queues au sein de l'application.

Syntaxe	**INITQUEUE**

1. Suite à cette instruction, un éventuel code d'erreur peut être obtenu par ERRORQUEUE%.
2. INITQUEUE est obligatoire pour initialiser l'utilisation des queues au sein de l'application. Elle doit être appelée avant le premier CREATEQUEUE% ou OPENQUEUE% de l'application.

Exemple :

```
INITQUEUE
IF ERRORQUEUE% <> 0
MESSAGE "Erreur n°" && ERRORQUEUE%, "INITQUEUE"
HALT
ENDIF
MOVE CREATEQUEUE%("QUEUE1") TO HQ1%
MOVE OPENQUEUE%("QUEUE2") TO HQ2%
...
CLOSEQUEUE HQ2%
CLOSEQUEUE HQ1%
MOVE CREATEQUEUE%("QUEUE3") TO HQ3%
...
CLOSEQUEUE HQ3%
STOPQUEUE
```

Voir aussi STOPQUEUE

Fonction OPENQUEUE% (Librairie NSComm)

Ouvre une queue et retourne son handle.

Syntaxe	**OPENQUEUE%** *(nom-queue)*			
Paramètre	nom-queue	CSTRING	I	nom de la queue à ouvrir
Valeur retournée	INT(4)			

1. Pour que cette fonction s'effectue correctement, il faut qu'une autre application ait préalablement fait un CREATEQUEUE% de même nom.
2. Suite à cette fonction, un éventuel code d'erreur peut être obtenu par ERRORQUEUE%.
3. Le handle retourné peut ensuite être utilisé pour écrire dans la queue par WRITEONQUEUE, mais jamais pour lire.
4. Avant le premier CREATEQUEUE% ou OPENQUEUE% de l'application, il est nécessaire d'activer l'utilisation des queues à l'aide de INITQUEUE.

Exemple :

```
INITQUEUE
MOVE OPENQUEUE% ("NOMQUEUE") TO HQ%
IF ERRORQUEUE% <> 0
MESSAGE "Erreur n°" && ERRORQUEUE%, "OPENQUEUE%"
ENDIF
...
WRITEONQUEUE HQ%, ...
...
CLOSEQUEUE HQ%
STOPQUEUE
```

Voir aussi CLOSEQUEUE, WRITEONQUEUE%, CREATEQUEUE%, ERRORQUEUE%, INITQUEUE

Fonction PEEKFROMQUEUE% (Librairie NSComm)

Retourne le nombre d'octets du prochain segment en attente de lecture dans une queue.

Syntaxe	**PEEKFROMQUEUE%** *(handle queue)*			
Paramètre	handle queue	INT(4)	I	handle de la queue à étudier
Valeur retournée	INT(4)			

1. Cette instruction est souvent utilisée avant un READFROMQUEUE pour s'assurer qu'il y a quelque chose à lire dans la queue. De plus, la taille retournée par

PEEKFROMQUEUE% peut être réutilisée en dernier paramètre de READFROMQUEUE.

2. Suite à cette fonction, un éventuel code d'erreur peut être obtenu par ERRORQUEUE%.

3. Seule l'application qui a fait CREATEQUEUE% peut utiliser PEEKFROMQUEUE%.

Exemple :

```
...
MOVE CREATEQUEUE%("NOMQUEUE") TO HQ%
...
MOVE PEEKFROMQUEUE% (HQ%) TO I%
IF ERRORQUEUE% <> 0
MESSAGE "Erreur n°" && ERRORQUEUE%, "PEEKFROMQUEUE%"
ELSEIF I% > 0
READFROMQUEUE HQ%, ..., I%
ELSE
MESSAGE "Peek", "Rien dans la queue"
ENDIF
```

Voir aussi READFROMQUEUE, ERRORQUEUE%

Instruction READFROMQUEUE (Librairie NSComm)

Lit dans une queue.

Syntaxe	**READFROMQUEUE** *handle queue, handle segment, taille segment*			
Paramètres	handle queue	INT(4)	I	handle de la queue dans laquelle lire
	handle segment	INT(4)	I	adresse d'un segment dans lequel sont lues les données
	taille segment	INTEGER	I	taille du segment

1. L'élément le plus ancien de la queue est enlevé pour être stocké dans la variable segment de handle handle-segment et de taille taille-segment.

2. Suite à cette instruction, un éventuel code d'erreur peut être obtenu par ERRORQUEUE%.

3. Sous Windows, cette instruction n'est jamais bloquante même lorsque la queue est vide. Dans ce cas, ERRORQUEUE% a pour valeur ERROR_QUEUE_ENTRY%.

Exemple :

```
SEGMENT INDIVIDU
STRING NOM(31)
STRING PRENOM(31)
...
ENDSEGMENT

INITQUEUE
MOVE CREATEQUEUE%("NOMQUEUE") TO HQ%
NEW INDIVIDU, MONCLIENT%
```

```
...
READFROMQUEUE HQ%, MONCLIENT%, SIZEOF INDIVIDU
IF ERRORQUEUE% <> 0
MESSAGE "Erreur n°" && ERRORQUEUE%, "READFROMQUEUE"
EXIT
ENDIF
MOVE INDIVIDU(MONCLIENT%).NOM TO ...
MOVE INDIVIDU(MONCLIENT%).PRENOM TO ...
...
DISPOSE MONCLIENT%
CLOSEQUEUE HQ%
STOPQUEUE
```

***Voir aussi** PEEKFROMQUEUE%, WRITEONQUEUE, ERRORQUEUE%*

Instruction STOPQUEUE (Librairie NSComm)

Termine la gestion des queues au sein de l'application.

Syntaxe	**STOPQUEUE**

1. Suite à cette instruction, un éventuel code d'erreur peut être obtenu par ERRORQUEUE%.
2. STOPQUEUE est obligatoire lorsque l'utilisation des queues est terminée au sein de l'application, c'est-à-dire après le dernier CLOSEQUEUE.
3. Cette instruction n'est obligatoire que sous Windows.

Exemple :

```
INITQUEUE
MOVE CREATEQUEUE%("QUEUE1") TO HQ1%
MOVE OPENQUEUE%("QUEUE2") TO HQ2%
...
CLOSEQUEUE HQ2%
CLOSEQUEUE HQ1%
MOVE CREATEQUEUE%("QUEUE3") TO HQ3%
...
CLOSEQUEUE HQ3%
STOPQUEUE
IF ERRORQUEUE% <> 0
MESSAGE "Erreur n°" && ERRORQUEUE%, "STOPQUEUE"
ENDIF
```

***Voir aussi** INITQUEUE*

Instruction WRITEONQUEUE (Librairie NSComm)

Ecrit dans une queue.

Syntaxe	**WRITEONQUEUE** *handle queue, handle segment, taille segment*			
Paramètres	handle queue	INT(4)	I	handle de la queue dans laquelle écrire

	handle segment	INT(4)	I	adresse d'un segment dans lequel sont les données
	taille segment	INTEGER	I	taille du segment

1. Un nouvel élément est stocké dans la queue, cet élément étant une copie de la variable segment dont le handle et la taille sont passés en paramètres.
2. Les données éventuellement déjà présentes dans la queue ne sont pas écrasées. La nouvelle donnée est ajoutée ("empilée") dans la queue.
3. Suite à cette instruction, un éventuel code d'erreur peut être obtenu par ERRORQUEUE%.
4. Cette instruction peut être utilisée par les applications qui ont fait OPENQUEUE%

Exemple :

```
SEGMENT INDIVIDU
STRING NOM(31)
STRING PRENOM(31)
...
ENDSEGMENT

INITQUEUE
MOVE OPENQUEUE%("NOMQUEUE") TO HQ%
NEW INDIVIDU, MONCLIENT%
...
MOVE ... TO INDIVIDU(MONCLIENT%).NOM
MOVE ... TO INDIVIDU(MONCLIENT%).PRENOM
WRITEONQUEUE HQ%, MONCLIENT%, SIZEOF INDIVIDU
IF ERRORQUEUE% <> 0
MESSAGE "Erreur n°" && ERRORQUEUE%, "WRITEONQUEUE"
ENDIF
...
DISPOSE MONCLIENT%
CLOSEQUEUE HQ%
STOPQUEUE
```

Voir aussi READFROMQUEUE, ERRORQUEUE%

Gérer le DDE

Segments de description de données

Segments DDEDATA, DDETXTDATA (Librairie NSComm)

Segments descriptifs des données échangées via les événements DDE_DATA, DDE_EXECUTE et DDE_POKE.

Syntaxe	SEGMENT DDETXTDATA CSTRING ITEM INT DATASIZE(2)

	CSTRING DATA ENDSEGMENT
	SEGMENT DDEDATA CSTRING ITEM INT DATASIZE(2) CHAR DATA(1024) ENDSEGMENT

1. La variable segment DDEDATA (ou DDETXTDATA) est pointée par le paramètre PARAM34% des événements DDE_DATA, DDE_EXECUTE et DDE_POKE. Elle est automatiquement allouée par les fonctions qui ont généré l'événement et remplie à l'aide des paramètres de ces fonctions :
   - DATADDE% (ou DATATXTDDE%)
   - EXECUTEDDE% (ou EXECUTETXTDDE%)
   - POKEDDE% (ou POKETXTDDE%)
2. La taille attribuée dans le segment DDEDATA au champ DATA ne signifie pas que la taille des données échangées par DDE est limitée à 1024 octets. Une application ne fait jamais de NEW d'un tel segment : sa déclaration sert uniquement à lire les données reçues avec les événements DDE DATA, DDE EXECUTE et DDE POKE. Cette déclaration est donc "fictive".

Ainsi si vous savez que votre serveur DDE est susceptible de recevoir plus de 1 Ko de données, par exemple 4 Ko, déclarez :

```
SEGMENT DDELONGDATA
CSTRING ITEM
INT DATASIZE(2)
CHAR DATA(4096)
ENDSEGMENT
```

et utilisez DDELONGDATA au lieu de DDEDATA pour lire les données. Dans tous les cas, lisez d'abord la valeur retournée dans DATASIZE pour connaître le nombre réel d'octets transférés, puis lisez le contenu du tableau DATA.

Par exemple, sur l'événement DDE EXECUTE, pour lire les 100 derniers octets reçus dans le bloc de 4 ko déclaré précédemment :

```
MOVE DDELONGDATA(PARAM34%).DATASIZE TO NB%
; NB% peut valoir 4096 ou moins (mais est supposé être toujours > à 100
pour cet exemple)
MOVE DDELONGDATA(PARAM34%).DATA((NB%-100)..NB%) TO S$
```

Exemple :

```
;Evénement DDE_POKE
evaluate ddetxtdata(param34%).item
where "R1C1"
move ddetxtdata(param34%).data to ...
...
endwhere
```

```
...
endevaluate
```

Voir aussi *DATADDE%*, *DATATXTDDE%*, *POKEDDE%*, *POKETXTDDE%*, ***Evénements*** *DDE_DATA*, *DDE_POKE*, *DDE_EXECUTE*

Du serveur au client

Instruction ACKDDE (Librairie NSComm)

Acceptation par le serveur DDE d'une demande DDE d'un client.

Syntaxe	**ACKDDE** *handle DDE, nom item*			
Paramètres	handle DDE	INT(4)	I	handle conversation DDE
	nom item	CSTRING	I	nom de l'item concerné

1. L'acceptation peut être retournée dans le traitement des événements DDE_ADVISE, DDE_EXECUTE ou DDE_POKE du serveur.
2. ACKDDE a pour effet de faire retourner TRUE% aux fonctions EXECUTEDDE% (ou EXECUTETXTDDE%) et POKEDDE% (ou POKETXTDDE%) employées par l'application client distante.

Exemple :

```
;Evénement DDE_POKE
...
ACKDDE PARAM12%, PARAM2$
```

Voir aussi *NACKDDE*, ***Evénements*** *DDE_ADVISE*, *DDE_EXECUTE*, *DDE_POKE*

Fonction DATADDE% (Librairie NSComm)

Envoi d'une mise à jour d'un item à un client dès le changement de valeur de l'item ou réponse à une demande de lecture d'un item.

Syntaxe	**DATADDE%** *(handle-DDE, nom-item, adresse-donnée, taille-donnée)*			
Paramètres	handle DDE	INT(4)	I	handle conversation DDE
	nom item	CSTRING	I	nom de l'item concerné
	adresse-donnée	INT(4)	I	adresse de la donnée transférée
	taille-donnée	INT(4)	I	taille de la donnée transférée
Valeur retournée	INT(1) TRUE% : la donnée a été transmise au client FALSE% : la donnée n'a pas été transmise au client			

1. Cette fonction est employée par un serveur DDE.
2. L'envoi d'une mise à jour d'un item au client, dès le changement de valeur de cet item, a pour effet de générer un événement DDE_DATA vers l'application client distante.
   Ce cas n'existe que si le serveur avait préalablement répondu positivement à un événement DDE_ADVISE sur cet item.
3. La réponse à une demande de lecture d'un item se fait sur occurrence de l'événement DDE_REQUEST. Cet événement est généré par un REQUESTDDE% du client.
   DATADDE% fait retourner TRUE% au REQUESTDDE% de l'application client distante.
4. Le nom de l'item doit être complété avec ".B" (par exemple, "DDEItem.B") s'il contient des données binaires. Si ce n'est pas le cas, le client DDE ne reçoit que les caractères précédant le premier octet nul rencontré dans la zone d'adresse adresse-donnée.

Exemple :

```
IF NOT DATADDE%(HDDE%, "R1C1", @VALUE$, LENGTH VALUE$)
MESSAGE "Erreur !", "La donnée R1C1 n'a pas été lue"
ENDIF/p>
```

***Voir aussi** DATATXTDDE%, ADVISEDDE%, UNADVISEDDE%, **Segments** DDEDATA, DDETXTDATA, **Evénements** DDE_DATA, DDE_REQUEST*

Fonction DATATXTDDE% (Librairie NSComm)

Envoi d'une mise à jour d'un item à un client dès le changement de valeur de l'item ou réponse à une demande de lecture d'un item.

Syntaxe	**DATATXTDDE%** *(handle-DDE, nom-item, chaîne-donnée)*			
Paramètres	handle DDE	INT(4)	I	handle conversation DDE
	nom item	CSTRING	I	nom de l'item concerné
	chaîne-donnée	CSTRING	I	chaîne transférée
Valeur retournée	INT(1) TRUE% : la donnée a été transmise au client FALSE% : la donnée n'a pas été transmise au client			

1. La fonction DATATXTDDE% est identique à DATADDE% : DATATXTDDE% est seulement une facilité de programmation lorsque la donnée à transmettre est une chaîne inférieure à 256 caractères.
2. Voir les commentaires de DATADDE%

Exemple :

```
IF NOT DATATXTDDE%(HDDE%, "R1C1", VALUE$)
MESSAGE "Erreur !", "La donnée R1C1 n'a pas été lue"
ENDIF
```

```
; Rigoureusement identique à
IF NOT DATADDE%(HDDE%, "R1C1", @VALUE$, LENGTH VALUE$)
MESSAGE "Erreur !", "La donnée R1C1 n'a pas été lue"
ENDIF
```

Voir aussi DATADDE%, ADVISEDDE%, UNADVISEDDE%, Segments DDEDATA, DDETXTDATA, Evénements DDE_DATA, DDE_REQUEST

Instruction NACKDDE (Librairie NSComm)

Refus d'une demande DDE d'un client.

Syntaxe	**NACKDDE** *handle DDE, nom-item*			
Paramètres	handle DDE	INT(4)	I	handle conversation DDE
	nom-item	CSTRING	I	nom de l'item concernée

1. Cette instruction peut être employée par un serveur lors du traitement d'un événement DDE_EXECUTE, DDE_POKE ou DDE_REQUEST.
2. NACKDDE a pour effet de faire retourner FALSE% aux fonctions EXECUTEDDE% (ou EXECUTETXTDDE%), POKEDDE% (ou POKETXTDDE%) et REQUESTDDE% (ou REQUESTTXTDDE%) employées par l'application client distante.

Exemple :

```
;Evénement DDE_REQUEST
...
NACKDDE PARAM12%, PARAM2$
```

Voir aussi ACKDDE, Evénements DDE_EXECUTE, DDE_POKE, DDE_REQUEST

Fonction RESPONDDDE% (Librairie NSComm)

Réponse positive à une demande de conversation DDE d'un client.

Syntaxe	**RESPONDDDE%** *(handle fenêtre, handle fenêtre dem, nom appli topic)*			
Paramètres	handle fenêtre	INT(4)	I	handle de la fenêtre (serveur DDE) acceptant le lien DDE
	handle fenêtre dem	INT(4)	I	handle de la fenêtre (client DDE) demandant le lien DDE
	nom appli topic	CSTRING	I/O	applications ou topics à lister
Valeur retournée	INT(4) 0 : la conversation DDE n'a pas été préalablement démarrée. non nul : handle de conversation DDE (cf commentaire 4)			

1. Cette fonction est employée par un serveur sur un événement DDE_INIT. Elle a pour effet de générer un événement DDE_INITACK vers l'application distante.
2. handle fenêtre est la fenêtre qui recevra les événements DDE_*.
3. handle fenêtre dem est égale à PARAM12% reçu avec DDE_INIT.
4. RESPONDDDE% retourne le handle conversation DDE si une conversation a été acceptée par un RETURN 0 sur DDE_INITACK. Ce handle doit ensuite être repassé en premier paramètre de chaque fonction ou instruction DDE. Ce handle est également repassé dans PARAM12% des événements DDE_*.
5. Si PARAM2$ de DDE_INIT vaut "" ou "appli", et si l'application réceptrice possède plusieurs topics disponibles, elle peut répondre par plusieurs RESPONDDDE% (un pour chaque topic) jusqu'à ce que l'une de ces fonctions retourne une valeur différente de zéro.
6. handle fenêtre est un handle au sens NatStar (ou NS-DK), alors que handle fenêtre dem est un handle au sens Windows ou PM. Un tel handle est égal au GETCLIENTHWND% (librairie NSWIN) d'un handle NatStar (ou NS-DK).

Exemple :

```
;Evénement DDE_INIT
IF (param2$ = "") OR (param2$ = appli$) OR (param2$ = "." & topic$) OR
(param2$ = appli$ & "." & topic$)
MOVE RESPONDDDE%(self%, param12%, appli$ & "." & topic$) TO hdde%
ENDIF
```

Voir aussi INITIATEDDE%, Evénements DDE_INIT, DDE_INITACK

Requêtes du client au serveur

Fonction ADVISEDDE% (Librairie NSComm)

Demande à l'application serveur distante de fournir les mises à jour d'une donnée (item) à chaque fois qu'elle change de valeur.

Syntaxe	**ADVISEDDE%** *(handle DDE, nom item)*			
Paramètres	handle DDE	INT(4)	I	handle conversation DDE
	nom item	CSTRING	I	nom de l'item concerné
Valeur retournée	INT(1) FALSE% : les mises à jour ne seront pas communiquées TRUE% : les mises à jour seront communiquées, un "lien permanent" est établi : le client reçoit alors des événements DDE_DATA à chaque changement de valeur de l'item du serveur.			

1. Cette fonction est activée par un client. Elle a pour effet de générer un événement DDE_ADVISE vers l'application serveur distante.
2. Selon le traitement de DDE_ADVISE effectué par le serveur, ADVISEDDE% retourne TRUE% ou FALSE%

Exemple :

```
IF ADVISEDDE%(HDDE%, "R1C1")
MESSAGE "Advise Accepté", "Des événements DDE_DATA seront maintenant reçus"
&& "à chaque changement de R1C1"
ELSE
MESSAGE "Advise Refusé", "Des événements DDE_DATA ne seront pas reçus" &&\
"à chaque changement de R1C1"
ENDIF
```

Voir aussi UNADVISEDDE, Evénements DDE_ADVISE, DDE_DATA

Fonction EXECUTEDDE% (Librairie NSComm)

Demande d'exécution d'une commande à un serveur.

Syntaxe	**EXECUTEDDE%** *(handle DDE, nom item, adresse commande, taille commande)*			
Paramètres	handle DDE	INT(4)	I	handle conversation DDE
	nom item	CSTRING	I	nom de l'item concerné
	adresse commande	INT(4)	I	adresse de la commande à exécuter
	taille commande	INT(4)	I	taille de la commande
Valeur retournée	INT(1) TRUE% : la commande a été acceptée par le serveur FALSE% : la commande a été refusée par le serveur			

Cette fonction est appelée par un client. Elle a pour effet de générer un événement DDE_EXECUTE vers l'application serveur distante.

Exemple :

```
IF NOT EXECUTEDDE%(HDDE%, "R1C1", @COMMAND$, LENGTH COMMAND$)
MESSAGE "Erreur !", "La commande n'a pas été acceptée"
ENDIF
```

Voir aussi Segments DDEDATA, DDETXTDATA, Evénements DDE_EXECUTE

Fonction EXECUTETXTDDE% (Librairie NSComm)

Demande d'exécution d'une commande à un serveur.

Syntaxe	**EXECUTETXTDDE%** *(handle DDE, nom item, commande)*			
Paramètres	handle DDE	INT(4)	I	handle conversation DDE
	nom item	CSTRING	I	nom de l'item concerné
	commande	CSTRING	I	nom de la commande

Valeur retournée	INT(1) TRUE% : la commande a été acceptée par le serveur FALSE% : la commande a été refusée par le serveur

La fonction EXECUTETXTDDE% est identique à EXECUTEDDE% : EXECUTETXTDDE% est seulement une facilité de programmation lorsque la commande à transmettre est une chaîne inférieure à 256 caractères.
Exemple :

```
IF NOT EXECUTETXTDDE%(HDDE%, "R1C1", COMMAND$)
MESSAGE "Erreur !", "La commande n'a pas été acceptée"
ENDIF

; Rigoureusement identique à
IF NOT EXECUTEDDE%(HDDE%, "R1C1", @COMMAND$, LENGTH COMMAND$)
MESSAGE "Erreur !", "La commande n'a pas été acceptée"
ENDIF
```

Voir aussi Segments DDEDATA, DDETXTDATA, Evénements DDE_EXECUTE

Fonction INITIATEDDE% (Librairie NSComm)

Demande la liste des serveurs DDE et de leurs topics, et permet d'établir une conversation DDE avec l'un d'entre eux.

Lors de l'utilisation d'un événement DDE, ne pas utiliser le TIMER 1, car ce dernier est utilisé par le protocole DDE.

Syntaxe	**INITIATEDDE%** *(handle fenêtre, nom-appli topic)*			
Paramètres	handle fenêtre	INT(4)	I	handle de la fenêtre (client DDE) désirant établir un lien DDE
	nom appli	CSTRING	I	applications ou topics à lister
Valeur retournée	INT(4) 0 : aucune conversation DDE n'a été démarrée. non nulle : handle de conversation DDE (cf commentaire 4)			

1. L'application qui fait l'INITIATEDDE% reçoit autant d'événements DDE_INITACK qu'il y a de topics correspondants (un serveur DDE pouvant avoir plusieurs topics).

Ces événements sont reçus et traités avant que la fonction ne retourne une valeur.

2. C'est la fenêtre handle fenêtre qui recevra les événements DDE_*.
3. nom-appli topic peut prendre quatre valeurs :
   a) "" pour lister la totalité des topics de toutes les applications serveurs DDE,
   b) "appli" pour lister la totalité des topics d'une application précise,
   c) ".topic" pour lister la totalité des applis ayant un topic précis,
   d) "appli.topic" pour lister uniquement un topic précis d'une application précise.

4. INITIATEDDE% retourne le handle conversation DDE si une conversation a été acceptée par un RETURN 0 sur DDE_INITACK. Ce handle doit ensuite être repassé en premier paramètre des fonctions ou instructions DDE. Ce handle est également repassé dans PARAM12% des événements DDE_*.
5. L'effet de INITIATEDDE% est de générer des événements DDE_INIT vers toutes les applications. Celles qui réagissent sur DDE_INIT en effectuant un RESPONDDDE% ont pour conséquence le retour d'un événement DDE_INITACK vers l'auteur de l'INITIATEDDE%.

Exemple 1 :

```
;Liste tous les topics DDE dans une list box sans établir de conversation
;Evénement INIT
DELETE FROM LISTBOX
MOVE INITIATEDDE%(SELF%, "") TO HDDE%
IF HDDE% <> 0
MESSAGE "Erreur DDE", "Conversation non demandée"
ENDIF
;Evénement DDE_INITACK
INSERT AT END PARAM2$ TO LISTBOX
RETURN 1
```

Exemple 2 :

```
;Etablit une liaison avec "Excel.Sheet1"
;Evénement INIT
MOVE INITIATEDDE%(SELF%, "Excel.Sheet1") TO HDDE%
IF HDDE% <> 0
MESSAGE "DDE OK", "Conversation établie avec Excel"
ENDIF
;Evénement DDE_INITACK
RETURN 0
;Evénement TERMINATE
TERMINATEDDE HDDE%
```

Exemple 3 :

```
;Idem exemple 2, établit une liaison avec "Excel.Sheet1"
;Evénement INIT
MOVE INITIATEDDE%(SELF%, "Excel") TO HDDE%
IF HDDE% <> 0
MESSAGE "DDE OK", "Conversation établie avec Excel"
ENDIF
;Evénement DDE_INITACK
IF PARAM2$ = "Excel.Sheet1"
RETURN 0
ELSE
RETURN 1
ENDIF
```

Voir aussi TERMINATEDDE, Evénement DDE_INITACK

Fonction POKEDDE% (Librairie NSComm)

Demande à un serveur de modifier une donnée.

Syntaxe	**POKEDDE%** *(handle DDE, nom item, adresse donnée, taille-donnée)*			
Paramètres	handle DDE	INT(4)	I	handle conversation DDE
	nom tem	CSTRING	I	nom de l'item concerné
	adresse donnée	INT(4)	I	adresse de la donnée transférée
	taille donnée	INT(4)	I	taille de la donnée transférée
Valeur retournée	INT(1) TRUE% : la donnée à modifier a été acceptée par le serveur FALSE% : la donnée à modifier a été refusée par le serveur			

Cette fonction est employée par un client. Elle a pour effet de générer un événement DDE_POKE vers l'application serveur distante.

Exemple :

```
IF NOT POKEDDE%(HDDE%, "R1C1", @VALUE$, LENGTH VALUE$)
MESSAGE "Erreur !", "La donnée R1C1 n'a pas été acceptée"
ENDIF
```

***Voir aussi** POKETXTDDE%, REQUESTDDE%, REQUESTTXTDDE%, Evénements DDE_POKE, Segments DDEDATA, DDETXTDATA*

Fonction POKETXTDDE% (Librairie NSComm)

Demande à un serveur de modifier une donnée.

Syntaxe	**POKETXTDDE%** *(handle DDE, nom item, chaîne donnée)*			
Paramètres	handle DDE	INT(4)	I	handle conversation DDE
	nom tem	CSTRING	I	nom de l'item concerné
	chaine donnée	INT(4)	I	chaîne transférée
Valeur retournée	INT(1) TRUE% : la donnée à modifier a été acceptée par le serveur FALSE% : la donnée à modifier a été refusée par le serveur			

La fonction POKETXTDDE% est identique à POKEDDE% : POKETXTDDE% est seulement une facilité de programmation lorsque la donnée à transmettre est une chaîne inférieure à 256 caractères.

Exemple :

```
IF NOT POKETXTDDE%(HDDE%, "R1C1", VALUE$)
MESSAGE "Erreur !", "La donnée R1C1 n'a pas été acceptée"
```

```
ENDIF

; Rigoureusement identique à
IF NOT POKEDDE%(HDDE%, "R1C1", @VALUE$, LENGTH VALUE$)
MESSAGE "Erreur !", "La donnée R1C1 n'a pas été acceptée"
ENDIF
```

***Voir aussi** POKEDDE%, REQUESTDDE%, REQUESTTXTDDE%, **Evénements** DDE_POKE, **Segments** DDEDATA, DDETXTDATA*

Fonction REQUESTDDE% (Librairie NSComm)

Demande la valeur d'une donnée à un serveur.

Syntaxe	**REQUESTDDE%** *(handle-DDE, nom item, adresse donnée, taille-maxi, variable-taille-retour)*			
Paramètres	handle DDE	INT(4)	I	handle conversation DDE
	nom-item	CSTRING	I	nom de l'item concerné
	adresse donnée	INT(4)	I	adresse de la donnée retournée
	taille maxi	INT(4)	I	taille maximum acceptée pour la donnée
	variable taille retour	INT(4)	I/O	taille effective de la donnée
Valeur retournée	INT(1) TRUE% : la donnée a été retournée par le serveur FALSE% : la donnée n'a pas été retournée			

Cette fonction est employée par un client. Elle a pour effet de générer un événement DDE_REQUEST vers l'application serveur distante.

Exemple :

```
LOCAL CHAR VALUE$(1024), INT I%(4)
IF NOT REQUESTDDE%(HDDE%, "R1C1", @VALUE$, 1024, I%)
MESSAGE "Erreur !", "La donnée R1C1 n'a pas été retournée"
ELSE
MESSAGE "La taille de R1C1 est", I%
MESSAGE "La valeur de R1C1 est", VALUE$
ENDIF
```

***Voir aussi** REQUESTTXTDDE%, POKEDDE%, POKETXTDDE%, **Evénements** DDE_REQUEST, **Segments** DDEDATA, DDETXTDATA*

Fonction REQUESTTXTDDE% (Librairie NSComm)

Demande la valeur d'une donnée à un serveur.

Syntaxe	**REQUESTTXTDDE%** *(handle DDE, nom item, var chaîne donnée)*			
Paramètres	handle DDE	INT(4)	I	handle conversation DDE
	nom-item	CSTRING	I	nom de l'item concerné
	var chaîne donnée	CSTRING	I/O	chaîne contenant la donnée retournée
Valeur retournée	INT(1) TRUE% : la donnée a été retournée par le serveur FALSE% : la donnée n'a pas été retournée			

La fonction REQUESTTXTDDE% est identique à REQUESTDDE% : REQUESTTXTDDE% est seulement une facilité de programmation lorsque la donnée à récupérer est une chaîne inférieure à 256 caractères.
Exemple :

```
LOCAL VALUE$
IF NOT REQUESTTXTDDE%(HDDE%, "R1C1", VALUE$)
MESSAGE "Erreur !", "La donnée R1C1 n'a pas été retournée"
ELSE
MESSAGE "La valeur de R1C1 est", VALUE$
ENDIF
```

Voir aussi REQUESTTXTDDE%, POKEDDE%, POKETXTDDE%, Evénements DDE_REQUEST, Segments DDEDATA, DDETXTDATA

Instruction TERMINATEDDE (Librairie NSComm)

Termine une conversation DDE en générant un DDE_TERMINATE vers l'autre application.

Syntaxe	**TERMINATEDDE** *handle DDE*			
Paramètre	handle DDE	INT(4)	I	handle conversation DDE

Cette instruction peut être employée par un client comme par un serveur.
Exemple :

```
;Evénement XXX (si Client)
MOVE INITIATEDDE%(SELF%, "appli.topic") TO HDDE%
;Evénement DDE_INIT (si Serveur)
MOVE RESPONDDDE%(SELF%, PARAM12%, "appli.topic") TO HDDE%
;Evénement YYY
TERMINATEDDE HDDE%
```

Voir aussi INITIATEDDE%, RESPONDDDE%, Evénement DDE_TERMINATE

Instruction UNADVISEDDE (Librairie NSComm)

Demande à l'application serveur distante de ne plus fournir les mises à jour d'une donnée (item).

Syntaxe	**UNADVISEDDE** *handle DDE, nom item*			
Paramètres	handle DDE	INT(4)	I	handle conversation DDE
	nom item	CSTRING	I	nom de l'item concerné

1. Cette instruction est employée par un client. Elle a pour effet de générer un événement DDE_UNADVISE vers l'application serveur distante.
2. Après cette instruction, le client ne reçoit plus d'événement DDE_DATA concernant cet item du serveur. Le "lien permanent" est rompu.

Exemple :

```
UNADVISEDDE HDDE%, "R1C1"
```

Voir aussi ADVISEDDE%, Evénement DDE_UNADVISE, DDE_DATA

Gestion du time-out

Fonction GETTIMEOUTDDE% (Librairie NSComm)

Retourne la valeur de la temporisation associée à un échange DDE.

Syntaxe	**GETTIMEOUTDDE%**
Valeur retournée	INTEGER

La temporisation est exprimée en millisecondes.

Voir aussi SETTIMEOUTDDE

Instruction SETTIMEOUTDDE (Librairie NSComm)

Modifie la temporisation associée à un échange DDE.

Syntaxe	**SETTIMEOUTDDE** *timeout*			
Paramètre	timeout	INTEGER	I	nouvelle temporisation

1. La temporisation est exprimée en millisecondes. La durée maximale est de 65000 ms.
2. Par défaut, la temporisation d'un échange DDE est de 10 millisecondes.
3. Une demande DDE échouant à cause d'une temporisation arrivant à terme est équivalent à une demande refusée. Les fonctions EXECUTEDDE%, POKEDDE% et REQUESTDDE% retournent alors FALSE%.

Exemple :

```
; La temporisation est passée à 20 millisecondes
SETTIMEOUTDDE 20
```

Voir aussi GETTIMEOUTDDE%

Gérer le presse-papiers

Fonction CLIPBOARDSIZE% (Librairie NSComm)

Retourne la taille du texte contenu dans le presse-papiers.

Syntaxe	**CLIPBOARDSIZE%**
Valeur retournée	INT(2)

La taille du texte est donnée en nombre de caractères.
Exemple :

```
MESSAGE "Le presse papiers contient", CLIPBOARDSIZE% && "caractères"
```

***Voir aussi** READFROMCLIPBOARD, WRITETOCLIPBOARD*

Instruction READFROMCLIPBOARD (Librairie NSComm)

Lit le texte contenu dans le presse papiers.

Syntaxe	**READFROMCLIPBOARD** *adresse variable, var taille lue, taille maxi*			
Paramètres	adresse variable	INT(4)	I	adresse d'une variable dans laquelle est copié le presse papier
	var taille lue	INT(2)	I/O	nombre de caractères copiés
	taille max	INT(2)	I	nombre maximum de caractères acceptés (correspond en général à la taille utile de la variable indiquée en premier paramètre)

Exemple :

```
LOCAL S$, I%(2)

READFROMCLIPBOARD @S$, I%, 255 ; 255 = (SIZEOF S$) - 1
MESSAGE "Le texte contenu dans le presse-papiers est", S$
MESSAGE "Le nombre de caractères lus est", I%

LOCAL CHAR S$(4096), I%(2)
READFROMCLIPBOARD @S$, I%, 4095
MESSAGE "Le texte contenu dans le presse-papiers est", S$
MESSAGE "Le nombre de caractères lus est", I%
```

***Voir aussi** CLIPBOARDSIZE%, WRITETOCLIPBOARD*

Instruction WRITETOCLIPBOARD (Librairie NSComm)

Ecrit du texte dans le presse-papiers.

Syntaxe	**WRITETOCLIPBOARD** *adresse variable, taille à écrire*			
Paramètres	adresse variable	INT(4)	I	adresse d'une variable dont le contenu sera copié dans le presse papiers
	taille à écrire	INT(2)	I	nombre de caractères à copier de la variable

Exemple :

```
LOCAL S$, I%(2)

MOVE "Bonjour" TO S$
WRITETOCLIPBOARD @S$, LENGTH S$

; Autre exemple : comment effacer le presse papiers
WRITETOCLIPBOARD 0, 0
```

***Voir aussi** CLIPBOARDSIZE%, READFROMCLIPBOARD*

Référence des événements DDE

Evénement DDE_ADVISE (Librairie NSComm)

Informe l'application serveur réceptrice qu'une application client demande à obtenir les mises à jour d'une donnée (item) chaque fois que cette dernière change de valeur. Le client demande un "lien permanent".

Syntaxe	**DDE_ADVISE**	
Paramètres	PARAM12%	Handle conversation DDE.
	PARAM2$	Nom de l'item demandé par ADVISEDDE%

1. Cet événement est automatiquement généré par l'application distante utilisant la fonction ADVISEDDE%.
2. L'application réceptrice doit répondre par RETURN TRUE% si elle accepte de fournir les mises à jour, ou par RETURN FALSE% si elle refuse. La fonction ADVISEDDE% génératrice de l'événement retourne TRUE% ou FALSE% en fonction de la réponse.
3. Si l'application réceptrice fait un RETURN TRUE%, c'est ensuite de sa responsabilité de faire un DATADDE% (ou DATATXTDDE%) à chaque fois que l'item change de valeur.

Exemple :

```
;Evénement DDE_ADVISE
if param12% = hdde%
evaluate param2$
where "R1C1"
move true% to memr1c1%
return true% ; acceptation
endwhere
...
else
return false% ; refus
endevaluate
endif
;Evénement changement de la donnée "R1C1"
if memr1c1%
if not datatxtdde%(hdde%, "R1C1", data$)
message "Erreur", "DataTxt refusé"
endif
endif
```

***Voir aussi Evénements** DDE_DATA, DDE_UNADVISE, **Fonctions** ADVISEDDE%, DATADDE% et DATATXTDDE%*

Evénement DDE_DATA (Librairie NSComm)

Indique à l'application client réceptrice (en "lien permanent" avec le serveur) qu'une donnée de l'application serveur a changé de valeur. La nouvelle valeur est passée avec l'événement.

Syntaxe	DDE_DATA	
Paramètres	PARAM12%	Handle conversation DDE.
	PARAM34%	Pointeur sur segment DDEDATA (ou DDETXTDATA)

1. Cet événement est automatiquement généré par l'application distante utilisant la fonction DATADDE% (ou DATATXTDDE%).
2. L'application réceptrice peut lire la donnée dans le segment dont le pointeur est passé dans PARAM34%.
3. Cet événement ne peut être reçu que si un ADVISEDDE% concernant cette donnée a été préalablement demandé par l'application client et accepté par l'application serveur.

Exemple :

```
;Evénement DDE_DATA
if param12% = hdde%
evaluate ddetxtdata(param34%).item
where "R1C1"
move ddetxtdata(param34%).data to datar1c1$
endwhere
...
endevaluate
endif
```

Voir aussi Evénements DDE_ADVISE, DDE_UNADVISE, Fonctions DATADDE% et DATATXTDDE%

Evénement DDE_EXECUTE (Librairie NSComm)

Demande à l'application serveur réceptrice d'exécuter une commande.

Syntaxe	DDE_EXECUTE	
Paramètres	PARAM12%	Handle conversation DDE.
	PARAM34%	Pointeur sur segment DDEDATA (ou DDETXTDATA)

1. Cet événement est automatiquement généré par l'application distante utilisant la fonction EXECUTEDDE% (ou EXECUTETXTDDE%).
2. L'application réceptrice doit répondre par l'instruction ACKDDE si elle accepte d'exécuter la commande, ou par l'instruction NACKDDE si elle refuse : la fonction EXECUTEDDE% (ou EXECUTETXTDDE%) génératrice de l'événement retourne alors TRUE% ou FALSE%.
3. L'application réceptrice peut lire la commande dans le segment dont le pointeur est passé dans PARAM34%.

Exemple :

```
;Evénement DDE_EXECUTE
if param12% = hdde%
evaluate ddetxtdata(param34%).data
where "R1C1"
move ddetxtdata(param34%).data to commandr1c1$
ackdde hdde%, ddetxtdata(param34%).data
endwhere
...
else
nackdde hdde%, ddetxtdata(param34%).data
endevaluate
endif
```

Voir aussi Evénement DDE_POKE, Fonctions EXECUTEDDE% et EXECUTETXTDDE%, Segments DDEDATA, DDETXTDATA

Evénement DDE_INIT (Librairie NSComm)

Informe l'application serveur réceptrice qu'une application client demande d'initialiser une conversation DDE.

Syntaxe	DDE_INIT	
Paramètres	PARAM12%	Handle fenêtre de l'application demandant une conversation. Ce paramètre vaut GETCLIENTHWND% du handle passé en premier paramètre du INITIATEDDE% générateur.
	PARAM2$	Nom du serveur demandé par INITIATEDDE%. Cette chaîne peut avoir quatre formes :

		"" lorsque le client demande à tous les serveurs tous leurs topics, "appli" lorsque le client demande à un serveur tous ses topics, ".topic" lorsque le client demande à tous les serveurs un topic précis, "appli.topic" lorsque le client demande à un serveur un topic précis.

1. Cet événement est automatiquement généré vers toutes les fenêtres principales par toute application utilisant la fonction INITIATEDDE%.
2. L'application réceptrice doit répondre par la fonction RESPONDDDE% si elle accepte la conversation. Cette fonction retourne alors soit 0(le client ne désire finalement pas établir la conversation), soit le Handle de Conversation DDE qui doit être ensuite réutilisé dans chaque PARAM12% des autres événements DDE_* ainsi qu'en premier paramètre des fonctions ou instructions DDE.
3. Si PARAM2$ vaut "" ou "appli", et si l'application réceptrice possède plusieurs topics disponibles, elle peut répondre par plusieurs RESPONDDDE% (un pour chaque topic) jusqu'à ce que l'une de ces fonctions retourne une valeur différente de zéro.

Exemple :

```
;Evénement DDE_INIT
if (param2$ = "") or (param2$ = appli$) or (param2$ = "." & topic$) or
(param2$ = appli$ & "." & topic$)
move responddde%(self%, param12%, appli$ & "." & topic$) to hdde%
endif
```

Voir aussi Evénements DDE_INITACK, DDE_TERMINATE, Fonctions INITIATEDDE%, RESPONDDDE%, Fonction GETCLIENTHWND% (librairie NSWIN)

Evénement DDE_INITACK (Librairie NSComm)

Informe l'application client réceptrice, qui avait préalablement fait un INITIATEDDE%, qu'un serveur DDE lui répond positivement.

Syntaxe	**DDE_INITACK**	
Paramètres	PARAM12%	Sans signification.
	PARAM2$	Nom du serveur "appli.topic" indiqué par RESPONDDDE%.

1. Cet événement est automatiquement généré par chaque réponse positive faite par les serveurs (RESPONDDDE% depuis l'événement DDE_INIT des serveurs).
2. Un RETURN 0 accepte la conversation avec le serveur DDE, alors qu'un RETURN 1 la refuse. La valeur rendue par défaut est 0.
3. Selon le dernier paramètre employé lors de INITIATEDDE% (par exemple, ""), cet événement peut être reçu plusieurs fois de suite. Cet événement n'est plus reçu dès qu'une conversation a été acceptée par RETURN 0.

4. Si aucun événement DDE_INITACK n'est accepté (aucun n'effectue un RETURN 0), alors INITIATEDDE%, comme RESPONDDDE%, retourne 0.

5. Si un événement DDE_INITACK est accepté (le premier qui effectue un RETURN 0), alors INITIATEDDE%, comme RESPONDDDE%, retourne le Handle de Conversation DDE qui sera ensuite réutilisé dans chaque PARAM12% des autres événements DDE_* ainsi qu'en premier paramètre des fonctions ou instructions DDE.

Exemple 1 :

```
;Evénement XXX pour liaison avec Excel
; SAUVEGARDE DE HDDE%
move initiatedde%(self%, "Excel.Sheet1") to hdde%
if hdde% <> 0
message "OK", "Conversation démarrée avec Excel"
else
message "Erreur", "Conversation impossible avec Excel"
endif
;Evénement DDE_INITACK
if param2$ <> "Excel.Sheet1"
return 0
else
return 1
endif
```

Exemple 2 :

```
;Evénement XXX pour lister les serveurs DDE
delete from listbox
if initiatedde%(self%, "") <> 0
message "Erreur", "Conversation acceptée mais non voulue"
endif
;Evénement DDE_INITACK
insert at end param2$ to listbox
return 1
```

Voir aussi Evénement DDE_INIT, DDE_TERMINATE, Fonctions INITIATEDDE%, RESPONDDDE%

Evénement DDE_POKE (Librairie NSComm)

Demande à l'application serveur réceptrice de modifier une donnée.

Syntaxe	**DDE_POKE**	
Paramètres	PARAM12%	Handle conversation DDE.
	PARAM34%	Pointeur sur segment DDEDATA (ou DDETXTDATA)

1. Cet événement est automatiquement généré par l'application distante utilisant la fonction POKEDDE% (ou POKETXTDDE%).

2. L'application réceptrice doit répondre par l'instruction ACKDDE si elle accepte de modifier la donnée, ou par l'instruction NACKDDE si elle refuse. La fonction POKEDDE% (ou POKETXTDDE%) génératrice de l'événement retourne alors TRUE% ou FALSE%.

3. L'application réceptrice peut lire la donnée dans le segment dont le pointeur est passé dans PARAM34%.

Exemple :

```
;Evénement DDE_POKE
if param12% = hdde%
evaluate ddetxtdata(param34%).item
where "R1C1"
move ddetxtdata(param34%).data to datar1c1$
ackdde hdde%, ddetxtdata(param34%).item
endwhere
...
else
nackdde hdde%, ddetxtdata(param34%).item
endevaluate
endif
```

Voir aussi Evénements DDE_REQUEST, DDE_EXECUTE, Fonctions POKEDDE% et POKETXTDDE%

Evénement DDE_REQUEST (Librairie NSComm)

Demande à l'application serveur réceptrice de fournir une donnée.

Syntaxe	DDE_REQUEST	
Paramètres	PARAM12%	Handle conversation DDE.
	PARAM2$	Nom de l'item demandé par REQUESTDDE% (ou REQUESTTXTDDE%).

1. Cet événement est automatiquement généré par l'application distante utilisant la fonction REQUESTDDE% (ou REQUESTTXTDDE%).
2. L'application réceptrice doit répondre par la fonction DATADDE% (ou DATATXTDDE%) si elle accepte de fournir la donnée, ou par l'instruction NACKDDE si elle refuse : la fonction REQUESTDDE% (ou REQUESTTXTDDE%) génératrice de l'événement retourne alors TRUE% ou FALSE%.
3. La donnée transmise dans le dernier paramètre de DATADDE% (ou DATATXTDDE%) est récupérée dans le dernier paramètre de REQUESTDDE% (ou REQUESTTXTDDE%).

Exemple :

```
Evénement DDE_REQUEST
if param12% = hdde%
evaluate param2$
where "R1C1"
if not datatxtdde%(hdde%, param2$, datar1c1$)
message "Erreur", "DataTxt refusé"
endif
endwhere
...
else
nackdde hdde%, param2$
endevaluate
endif
```

Voir aussi Evénement DDE_POKE, Fonctions REQUESTDDE% et REQUESTTXTDDE%

Evénement DDE_TERMINATE (Librairie NSComm)

Informe de la fin d'une conversation DDE.

Syntaxe	DDE_TERMINATE	
Paramètres	PARAM12%	Handle conversation DDE.
	PARAM2$	Sans signification. NE PAS UTILISER.

1. Cet événement peut être reçu par le client comme par le serveur.
2. Cet événement est automatiquement généré par l'application distante utilisant l'instruction TERMINATEDDE.
3. Cet événement ne peut être reçu que si une conversation DDE avait été démarrée, soit par RESPONDDDE% pour un serveur, soit par INITIATEDDE% pour un client.
4. Une application doit filtrer cet événement afin de détecter un abandon de la conversation par le process client/serveur avec lequel elle a été engagée. En vérifiant, comme dans l'exemple ci-dessous, la valeur de hdde%, il est possible de savoir si la conversation a toujours cours.

Exemple :

```
;Evénement DDE_TERMINATE
if param12% = hdde%
move 0 to hdde%
else
message "Erreur", "Demande d'arrêt" && "d'une conversation non démarrée"
endif
```

Voir aussi Evénements DDE_INIT, DDE_INITACK, Instruction TERMINATEDDE

Evénement DDE_UNADVISE (Librairie NSComm)

Informe l'application serveur réceptrice qu'une application client lui demande de ne plus lui fournir les mises à jour d'une donnée (item). Le client termine le "lien permanent".

Syntaxe	DDE_UNADVISE	
Paramètres	PARAM12%	Handle conversation DDE.
	PARAM2$	Nom de l'item précisé par UNADVISEDDE.

1. Cet événement est automatiquement généré par l'application distante utilisant l'instruction UNADVISEDDE.
2. Cet événement ne peut être reçu que si un DDE_ADVISE concernant cette donnée avait été préalablement reçu et accepté pendant la conversation.
3. C'est ensuite de la responsabilité de l'application réceptrice du DDE_UNADVISE de ne plus faire de DATADDE% (ou DATATXTDDE%) concernant cet item.

Exemple :

```
;Evénement DDE_UNADVISE
if param12% = hdde%
evaluate param2$
where "R1C1"
move false% to memr1c1%
endwhere
...
endevaluate
endif
;Evénement changement de la donnée "R1C1"
if memr1c1%
if not datatxtdde%(hdde%, "R1C1", datar1c1$)
message "Erreur", "DataTxt refusé"
endif
endif
```

Voir aussi Evénements DDE_DATA, DDE_ADVISE, Instruction UNADVISEDDE

LIBRAIRIE SEQUENTIELLE INDEXEE NSDB

NSDB est un gestionnaire de fichiers séquentiel indexé utilisable avec NatStar (ou NS-DK). Il permet à des applications orientées vers la gestion, d'organiser, d'insérer, modifier, détruire et consulter des données très diverses.

A propos de NSDB

La facilité de mise en oeuvre de multiples méthodes d'accès à base d'index multi clés permet l'échafaudage d'applications complexes offrant de très bonnes performances. NSDB peut être dans certaines applications une alternative séduisante par rapport à des systèmes de base de données plus lourds.

Plusieurs bases de données peuvent être ouvertes et utilisées en même temps. Une base est constituée de plusieurs types de segments (enregistrements) eux mêmes indexés sur plusieurs index indépendants.

NSDB permet de travailler en mode client serveur : accès simultané à la même base par plusieurs applications situées sur une même machine (fonctionnement standalone) ou sur différentes machines d'un même réseau local (fonctionnement réseau).

En résumé, les principales caractéristiques techniques de NSDB sont les suivantes :

- Multi BASES
- Multi TABLES
- Multi INDEX
- Multi CLES
- Mono ou Multi utilisateurs (mode client serveur)
- Standalone ou Réseau

La taille maximum d'une base NSDB utilisée dans une application NatStar (ou NS-DK) est de 134 Mo. De plus, Nat System propose en option un langage de type SQL, qui permet d'interroger les bases NSDB.

Installation

Librairies dynamiques

Le programme d'installation de NatStar (ou NS-DK) installe quatre librairies dynamiques nécessaires à NSDB. Ce sont :

- NSxx_DB.DLL : Interface entre NatStar (ou NS-DK) et NSDB.
- NSxxDB.DLL : NSDB version mono utilisateur
- NSxxDBCL.DLL : NSDB version multi utilisateurs (côté client)
- NSxxSCDB.DLL : Mise en œuvre en réseau

Librairie NCL

Contrairement à la majorité des librairies fournies avec NatStar (ou NS-DK), il n'y a aucun fichier .NCL associé à déclarer.

Fichier d'initialisation installé par défaut

Un fichier d'initialisation décrit les bases de données gérées par NSDB sur la machine. Ce fichier permet notamment de choisir le mode (client serveur ou autre) à mettre en oeuvre.
Le fichier d'initialisation utilisé par NSDB est désigné par la variable d'environnement NS DB :

```
SET NS-DB =C:\NATSTAR\NSDB\Nom_Fichier_Initialisation
```

Le nom du fichier d'initialisation peut être quelconque.
Le programme d'installation de NatStar (ou NS-DK) construit un premier fichier d'initialisation, appelé NSDB.INI et contenant des valeurs par défaut :

```
[System]
dbdir='C:\NATSTAR\NSDB' ; /* NSDB Working Directory */
Journal=YES ; /* YES (COMMIT and ROLLBACK in effect) or NO */
[User]
data='C:\NATSTAR\NSDB\data', CASESENS ; /* List of databases */
/* LogicalName='FullPathName' [, CASESENS | CASEINSENS] */
[End]
END_INIT ; /* End of the initialization file */
```

Ce fichier indique un répertoire de travail et définit une première base locale, appelée data, mono-utilisateur.
La syntaxe d'un fichier d'initialisation est décrite dans la section suivante.
La variable d'environnement NS-DB correspondante est déclarée automatiquement, lors de l'installation, dans le fichier AUTOEXEC.BAT sous Windows.

```
SET NS-DB=C:\NATSTAR\NSDB\NSDB.INI
```

Syntaxe du fichier d'initialisation

Le groupe [SYSTEM]

Ce groupe permet de définir les paramètres généraux de fonctionnement de NSDB (Répertoire NS-DB, fonctionnement client serveur, etc ...).

La commande DBDIR

La commande DBDIR définit le répertoire de travail NS-DB, où sont stockés les fichiers temporaires, les options de TRACE et de MONITORING ainsi que certaines caractéristiques générales de fonctionnement NS-DB.

Syntaxe	DBDIR='Drive:\Path' [,TRACE='NO' | 'TrFile' [,MONITOR='NO' | 'MnName' [,MAXTRANS=1..1024]]] ;

Où :

DBDIR	Le répertoire de travail précisé doit être préalablement créé. Il recevra à chaque initialisation un fichier temporaire de type NSWK000X.TMP généré automatiquement par l'optimiseur.
TRACE	La trace doit être principalement utilisée lorsqu'un "bug" NSDB inexpliqué survient dans votre application. Cette option permet de définir un nom de fichier d'une longueur maximum de 8 caractères dans lequel tous les appels à NSDB sont enregistrés. Ce fichier sera stocké dans le répertoire de travail NS-DB . En cas de problème NSDB inexpliqué : contactez Nat System pour une mise en oeuvre détaillée de la TRACE. Si TRACE='NO' la trace est inactive.
MONITOR	Définit le nom de la mémoire partagée utilisée entre le moteur NSDB et l'utilitaire DBMON.EXE. Si vous ne souhaitez pas mettre en oeuvre cet utilitaire, MONITOR peut être omis ou avoir la forme MONITOR='NO'.
MAXTRANS	Indique le nombre maximum de transactions NSDB qui peuvent être exécutées sur le noeud. Le nombre pris en compte par défaut si ce paramètre est omis est 50. Lorsque ce nombre devient insuffisant, l'erreur SQLEX 002 correspondant au message : TOO MANY TRANSACTIONS est renvoyée par NSDB lors de l'exécution d'un ordre DB_OPEN.

La commande SYSID

La commande SYSID identifie de façon unique chaque utilisateur client et/ou serveur de NSDB. Ce nom sera utilisé lors des accès client serveur.

Syntaxe	SYSID=NAME [,NETDLL=NETWORK DLL NAME [,NETLSTN='Drive:\Path\NetListener.EXE' [,MAXNAME=1..255 [,MAXCOMMAND=1..255 [,MAXSESSION=1..255]]]]] ;

Où :

NETDLL	Permet d'identifier le nom de la DLL de communication utilisée. Cette information est indispensable si l'une au moins des bases spécifiées dans le fichier d'initialisation est définie comme REMOTE. Elle indique au système le nom de la DLL réseau utilisée pour la communication client serveur.
NETLSTN	Cette information est utilisée par l'utilitaire DBSERV (serveur de database NSDB) lors de son démarrage. Elle indique au serveur DBSERV le nom du module de communication utilisé pour l'écoute des requêtes NSDB transmises par le réseau. Si ce paramètre est omis, DBSERV se considère comme serveur local et ne peut recevoir que des appels NSDB de clients résidant sur la même machine.

MAXNAME, MAXCOMMAND, MAXSESSION	Ces trois paramètres permettent respectivement de définir le nombre de maximum de noms, de commandes et de sessions NETBIOS utilisables par le NETBIOS LISTENER et par le serveur de base de données. Ces informations ne sont nécessaires que dans un contexte client serveur. Les valeurs utilisées par défaut par DBSERV sont : ▪ MAXNAME=16 ▪ MAXSESSION=16 ▪ MAXCOMMAND=16 Il peut être nécessaire d'augmenter ces valeurs lorsque le nombre de clients NETBIOS connectés simultanément au serveur de base de données augmente.

La commande JOURNAL

La commande Journal met en oeuvre la journalisation transactionnelle (possibilité d'annulation/validation des modifications par DB_ROLLBACK et DB_COMMIT).

Syntaxe	JOURNAL= YES | NO ;

Si JOURNAL=YES la journalisation transactionnelle est active.
Si JOURNAL=NO la journalisation transactionnelle est inopérante.

> Le fonctionnement de NSDB est plus rapide en mode journalisation, car les modifications de la base sont effectuées en mémoire et non sur disque : l'écriture sur disque n'est faiter que lors du DB_COMMIT.

Le groupe [USER]

Le groupe USER contient la liste des bases de données gérées par les applications sur la machine.
Le nom logique et le nom physique (avec sa localisation sur le disque) de chaque base de données sont spécifiés.

Syntaxe	Logical_Database_Name = 'Localisation' [,REMOTE | LOCAL] [,CASESENS | CASEINSENS] [,READONLY] [,BUFSIZE=taille buffer] ;

Où :

Logical_Database_Name	Représente le nom identifiant la base de données pour une application. Ce nom doit être utilisé par l'application pour identifier la base de données utilisée. Il ne peut excéder 8 caractères ALPHANUMERIQUES.
Localisation	Représente la localisation physique du PRIMARY TABLESPACE de la base de données. Cette localisation peut représenter un fichier disque ou un serveur distant.

CASESENS|CASEINSENS	signifie que les minuscules sont différenciées des majuscules. En spécifiant CASEINSENS, les minuscules ne sont pas différenciées des majuscules (les chaînes de caractères sont stockées en majuscules lors de chaque insertion ou mise à jour, et les recherches sont effectuées en majuscules). Le choix CASESENS ou CASEINSENS doit avoir un caractère permanent pour une base de données (Si une base est passée du mode CASESENS au mode CASEINSENS, les chaînes de caractères sont perdues). Ne rien préciser revient à spécifier CASESENS.
READONLY	indique que la base est partagée en lecture par les clients du réseau. Il doit être indiqué dans les fichiers d'initialisation de chaque poste client. La base ne sera donc jamais mise à jour. L'intérêt d'une telle déclaration consiste en une optimisation des accès réseau. Ce paramètre est incompatible avec les paramètres REMOTE ou LOCAL.

BUFSIZE	spécifie la taille d'un buffer de transfert. Un moyen d'optimiser les temps de transferts sur les bases distantes consiste en effet à transférer les données par paquets dans un buffer associé à la base Remote. Ce paramètre est surtout intéressant dans le cas de nombreux DB_NEXT ou DB_PREV successifs.
	Exemple :
	`/* Buffer constituant des paquets de 2048 octets */`
	`/* associé a la base REMOTE */`
	`BASE4='NOMSERV',REMOTE,BUFSIZE=2048 ;`

Le groupe [END]

Ce groupe doit simplement contenir :

```
END_INIT ;
```

Cette commande représente l'indicateur syntaxique de fin de fichier pour l'analyseur NSDB.

Un fichier d'initialisation supporte des commentaires. Comme en langage C, un commentaire commence par /* et se termine par */.

Exemple :

```
/* Ceci est un commentaire */
```

Les différentes commandes et les différents paramètres peuvent être saisis indifféremment en majuscules ou en minuscules.

Mise en oeuvre en mode client-serveur, standalone ou réseau

Présentation générale

En fonctionnement client serveur, des machines clientes accèdent à une même base de données située sur une machine serveur. En fait, il existe trois modes bien différents de fonctionnement client serveur :

1. le fonctionnement multi utilisateurs réseau
   a) Plusieurs applications disposées sur une machine cliente, ou plusieurs machines clientes, accèdent à la base distante.
   b) Chaque client NSDB est configuré en mode multi utilisateurs.
2. le fonctionnement mono utilisateur réseau
   a) Une seule application accède à la fois à la base distante.
   b) Ce fonctionnement est utilisé lorsqu'une application disposée sur un seul poste gère une base de données distante et non partagée.
3. le fonctionnement multi utilisateurs standalone
   a) Dans ce cas particulier, l'utilisateur dispose d'une seule machine qui sert à la fois de serveur et de client. Il s'agit quand même d'un fonctionnement client serveur parce que plusieurs applications disposées sur cette machine accèdent à la base locale.

Attention à la signification du terme "application" : il s'agit d'un process NSDB lancé par un utilisateur.
La mise en oeuvre de ces trois modes de fonctionnement est détaillée dans les paragraphes suivants.

Utilitaires de lancement et d'arrêt du serveur NSDB

L'utilitaire DBSERV.EXE lance le serveur NSDB. Il peut être démarré depuis une session plein écran.
Sa syntaxe est :

```
DBSERV [nom sysid [TRACE | NOTRACE [taille mémoire]]]
```

Les trois paramètres optionnels de DBSERV sont :

- le nom du SysId. Si ce paramètre est spécifié, il annule et remplace celui indiqué dans le fichier d'initialisation.
- la trace (TRACE ou NOTRACE). Ce paramètre n'a aucune influence en mode standalone.
- la taille de la mémoire partagée exprimée en octets correspondant à la taille maximum d'un "record" de table. Par défaut, cette taille est de 8192. Si la taille est spécifiée, employer au minimum 512 et au maximum 32768.

Au lancement du serveur, il s'affiche :

```
NatSys (R) DB server Version 1.00
Copyright (C) Nat System 1991,1992,1993 All rights reserved
SERVER DLL_LOAD : SUCCESSFULL DB CALL SQLEX 000
SERVER DB_INIT : SUCCESSFULL DB CALL SQLEX 000
SQLSRV+002 DB SCSH_SERVER waiting for client requestors
SERVER_ID IS * SysId *
```

Exemple :

```
[C:\] DBSERV NOMSRV NOTRACE 32768
```

Il est possible de démarrer plusieurs serveurs sur la même machine. Il faut à ce moment là créer autant de fichiers d'initialisation, un pour chaque SysId, et démarrer les serveurs avec :

```
[C:\] SET NS-DB =C:\NATSTAR\NSDB\OTHER.INI
[C:\] DBSERV OTHERSRV NOTRACE 32768
```

L'utilitaire SHUTDOWN.EXE arrête le serveur NSDB. Il peut être démarré depuis une session plein écran :

```
SHUTDOWN [nom sysid]
```

Le nom du SysId est optionnel. Spécifié, il annule et remplace celui indiqué dans le fichier d'initialisation.
A l'arrêt du serveur, il s'affiche :

```
NatSys (R) DB server Shutdown 1.00
Copyright (C) NAT SYSTEM 1991,1992,1993 All rights reserved
SQLSRV+002 SHUTDOWN Requests Transmitted to DB server
```

Et dans la session qui avait démarré le serveur :

```
SERVER DB_TERM : SUCCESSFUL DB CALL SQLEX 000
DB Server Ended ; duration : 130969 Ms
```

Exemple :

```
[C:\] SHUTDOWN NOMSRV
```

Fonctionnement réseau

Mise en place du serveur

Voici un exemple de fichier d'initialisation pour une machine serveur de données OS/2 :

```
[System]
dbdir='C:\NATSTAR\DB';
SysId='NOMSRV', NETDLL='ACSNETB',
NETLSTN='C:\NATSTAR\BIN\REMSRV.EXE';
```

```
Journal=YES ;

[User]
BASESRV='C:\NATSTAR\NSDB\MABASE.DAT', CASEINSENS;

[End]
END_INIT;
```

Le paramètre NETLSTN de SysId indique à NSDB que la machine demande à être serveur réseau de données.

Chaque machine du réseau (client ou serveur) doit avoir un SysId unique.

Paramètres

NETDLL :
Indiquer NB30 pour Microsoft Lan Manager 2.0 ou ACSNETB pour IBM Com Manager. Il est possible d'utiliser NB30 sur un serveur NSDB simultanément avec ACSNETB sur un client NSDB. Nat System fournit de plus une DLL NetBIOS pour Novell Netware, nommée NS02NWR.DLL.
Le paramètre NETDLL n'est pas exploité sous Windows car l'accès au NetBios se fait directement par l'interruption 5C du DOS.
NETLSTN :
Nat System fournit un programme d'écoute pour NetBIOS, REMSRV.EXE.

Il est possible de rediriger plusieurs fois une base sur un réseau. Le CLIENT peut définir une base BASE redirigée vers SERV1 en distant (BASE = 'SERV1', REMOTE;). SERV1 peut lui aussi redéfinir BASE redirigé vers SERV2 en distant (BASE = 'SERV2', REMOTE;) et ainsi de suite... jusqu'à ce que SERVn définisse BASE en local (BASE = 'C:\NATSTAR\NSDB\DATA.DAT', CASEINSENS;). Seule la dégradation des temps de réponse due à ces redirections en cascade limite ce type de pratique.

Le serveur doit être démarré avec DBSERV, et arrêté avec SHUTDOWN.

Mise en oeuvre côté client

Voici un exemple de fichier d'initialisation pour une machine cliente Windows :

```
[System]
dbdir='C:\NATSTAR\DB';
SysId='NOMCLI', NETDLL='ACSNETB';
Journal=YES;

[User]
BASESRV='NOMSRV', REMOTE;

[End]
END_INIT;
```

La ligne BASESRV='NOMSRV',REMOTE de la section [User] du client reprend le SysId ('NOMSRV') déclaré dans la section [System] du serveur et la base (BASESRV) déclarée dans la section [User] du serveur. Le paramètre REMOTE indique à NSDB que la machine demande à être client réseau.

Chaque machine du réseau (client ou serveur) doit avoir un SysId unique.

Fonctionnement multi utilisateurs

DBSERV doit être démarré sur chaque machine cliente lançant plusieurs process NSDB.
L'accès à la base distante se fait grâce à l'instruction DB_INIT en mode multi utilisateurs placée dans un script NCL. Le SysId du serveur local doit être indiqué derrière NS02DBCL :

```
DB_INIT "NS02DBCL.NOMCLI"
DB_OPEN HDB1%, "BASESRV",...
; accès base distante
```

Remarquer que pour la base BASESRV, le nom du serveur distant possédant physiquement cette base n'est pas spécifié dans le script NCL, ni dans DB_INIT, ni dans DB_OPEN : il est géré par le fichier d'initialisation.
La fin de l'utilisation de NSDB se programme dans un script NCL avec :

```
DB_STOP
```

SHUTDOWN doit être utilisé pour mettre fin à l'action de DBSERV.

Fonctionnement mono utilisateur

L'instruction DB_INIT suivante placée dans un script NCL permet à l'unique client d'accéder à la base distante en mode mono utilisateur :

```
DB_INIT "NS02DB"
DB_OPEN HDB1%, "BASESRV",...
; accès base distante
```

La fin de l'utilisation de NSDB se programme dans un script NCL avec :

```
DB_STOP
```

Il est bien sûr possible dans un fichier d'initialisation côté CLIENT d'indiquer une utilisation mixte de NSDB : une base étant distante (BASESRV) et une base étant locale (BASELOC).
Pour cela, il suffit de rajouter dans le groupe USER du fichier d'initialisation :

```
BASELOC='C:\NATSTAR\NSDB\LOCAL.DATA';
```

Et l'instruction suivante permet d'accéder à cette deuxième base :

```
DB OPEN HDB2%, "BASELOC",...
; accès base locale
```

Fonctionnement Standalone

Fichier(s) d'initialisation standalone

Il existe deux façons équivalentes de définir un fonctionnement multi utilisateurs standalone :

- avec un seul fichier d'initialisation
- avec deux fichiers d'initialisation sur la même machine, un pour les clients et un pour le serveur.

Premier cas : un fichier d'initialisation unique

Voici un exemple de fichier d'initialisation unique, autorisant un fonctionnement multi utilisateurs standalone :

```
[System]
dbdir='C:\NATSTAR\DB';
SysId='NOMSRV';
Journal=YES;

[User]
BASESRV='C:\NATSTAR\NSDB\LOCAL.DAT';

[End]
END_INIT;
```

Deuxième cas : un fichier d'initialisation client et un fichier d'initialisation serveur

Les deux fichiers doivent avoir des emplacements physiques différents.
Voici un exemple de fichier d'initialisation serveur :

```
[System]
dbdir='C:\NATSTAR\DB';
SysId='NOMSRV';
Journal=YES;

[User]
BASESRV='C:\NATSTAR\NSDB\LOCAL.DAT';

[End]
END_INIT;
```

L'emplacement de la base sur le disque et le nom du serveur sont indiqués dans ce fichier.
Le fichier d'initialisation client se présente ainsi :

```
[System]
dbdir='C:\NATSTAR\DB';
Journal=YES;

[User]
BASESRV='NOMSERV', LOCAL;

[End]
END_INIT;
```

Ce fichier d'initialisation indique que la base est sur le serveur local, de nom NOMSERV.
La ligne BASESRV='NOMSRV',LOCAL de la section [User] du client reprend le SysId ('NOMSRV') déclaré dans la section [System] du serveur et la base (BASESRV) déclarée dans la section [User] du serveur. Le paramètre LOCAL indique à NSDB que la machine demande à être client réseau.

Démarrage du serveur standalone

Premier cas : avec un fichier d'initialisation unique

Il faut démarrer le serveur standalone avec l'utilitaire DBSERV. Pour un client, soit donc toute application NatStar (ou NS-DK) fonctionnant sur la machine, il est maintenant possible d'accéder à ce serveur grâce à l'instruction DB_INIT en mode multi utilisateurs dans un script NCL. Le SysId du serveur doit être indiqué derrière NS02DBCL. Ainsi, le client peut faire :

```
DB_INIT "NS02DBCL.NOMSRV"
DB_OPEN HDB%, "BASESRV",...
```

Deuxième cas : avec un fichier d'initialisation client et un fichier d'initialisation serveur

DBSERV doit prendre en compte le fichier serveur, et non pas le fichier client. Pour ce faire, il faut préciser dans la session de lancement de DBSERV le fichier adéquat. Par exemple :

```
[C:\] SET NS-DB ="C:\NATSTAR\NSDB\NSDB2.INI"
[C:\] DBSERV
```

Un client accède ensuite à ce serveur grâce à une instruction DB_INIT mono utilisateur classique :

```
DB_INIT "NS02DB"
DB_OPEN HDB%, "BASESRV",...
```

Le fichier d'initialisation NSDB.INI client est utilisé.

Arrêt de NSDB

La fin de l'utilisation de NSDB se programme dans un script NCL avec :

```
DB_STOP
```

Le serveur multi utilisateurs est arrêté avec SHUTDOWN.

Langage NSDB

Présentation générale

DB_INIT démarre et DB_STOP clôt l'utilisation de la librairie NSDB.
DB_OPEN ouvre et DB_CLOSE ferme une base de données.
DB_OPENCURSOR% ouvre et DB_CLOSECURSOR ferme un curseur.
DB_INSERT insère, DB_UPDATE met à jour et DB_DELETE détruit une donnée.
DB_COMMIT valide et DB_ROLLBACK annule les dernières modifications dans la base.
DB_SEARCH recherche une donnée dans la base, puis DB_NEXT et DB_PREV permettent d'aller aux données suivantes et précédentes.
DB_ERROR% et DB_ERRMSG$ permettent d'obtenir un compte rendu sur les instructions précédentes.
INDEXES / ENDINDEXES permet de déclarer une table d'index à partir d'un SEGMENT préalablement déclaré.
La syntaxe des fonctions et instructions de NSDB est toujours accessible depuis la fenêtre Evénements de l'éditeur de fenêtre grâce à la ligne <NS-DB > qui apparaît en permanence dans la liste des librairies. Ces syntaxes sont contenues dans le fichier NSDB.SYS fourni avec NatStar (ou NS-DK).
Voici un exemple de fichier ressource INDIVIDU.SEG, contenant à la fois la description du segment et de la table d'index (ici un seul index, mais multi clés) :

```
SEGMENT INDIVIDU
STRING NOM(31)
STRING PRENOM(31)
ENDSEGMENT
INDEXES INDIVIDU
NOM,PRENOM
ENDINDEXES
```

Exemple d'utilisation

```
;Déclaration segment et index
; ---------------------------
SEGMENT INDIVIDU
STRING NOM(31)
STRING PRENOM(31)
ENDSEGMENT
INDEXES INDIVIDU
NOM, PRENOM
ENDINDEXES
```

```
GLOBAL INDIVIDU CL

; Ouverture base NSDB
; -------------------
DB_INIT "NS02DB" ; mode mono utilisateur
IF DB_ERROR%(0) <> 0
MESSAGE "Erreur NSDB", "DB_INIT"
EXIT
ENDIF

; La base "DATA" doit être déclarée dans le fichier d'initialisation
indiqué par SET NS-DB .
; ------------------------------------------------
DB_OPEN HDB%, "DATA", "CREATE" USING INDIVIDU
IF DB_ERROR%(HDB%) <> 0
MESSAGE "Erreur DB_OPEN", DB_ERRMSG$(HDB%)
DB_STOP
EXIT
ENDIF

; Ecriture base NSDB
; ------------------
MOVE "Dupont" TO CL.NOM
MOVE "Jean" TO CL.PRENOM
DB_INSERT HDB%, CL
IF DB_ERROR%(HDB%) <> 0
MESSAGE "Erreur DB_INSERT" && CL.NOM, DB_ERRMSG$(HDB%)
ENDIF

MOVE "Durand" TO CL.NOM
MOVE "Pierre" TO CL.PRENOM
DB_INSERT HDB%, CL
IF DB_ERROR%(HDB%) <> 0
MESSAGE "Erreur DB_INSERT" && CL.NOM, DB_ERRMSG$(HDB%)
ENDIF

; Lecture base NSDB
; -----------------
DB_SEARCH HDB%, CL WHERE NOM > " "
WHILE DB_ERROR%(HDB%) = 0
MESSAGE "Client", CL.NOM && CL.PRENOM
DB_NEXT HDB%, CL
ENDWHILE

; Fermeture base NSDB
; -------------------
DB_CLOSE HDB%
DB_STOP
```

Utilitaire de cohérence des bases

Utilisation

Un utilitaire de test est fourni :

```
DBCHCK.EXE
```

Il s'agit d'un outil de vérification d'état d'une base NSDB, qui affiche des erreurs SQLCCK XXX si la base est incorrecte. Cet outil est un exécutable mode caractères (DOS). Sa syntaxe est :

```
DBCHCK DATABASE_NAME
```

où DATABASE_NAME est le nom logique d'une base.
Si la vérification est bonne, l'utilitaire affiche :

```
NatSys (R) DB Physical Checker Version 1.00
Copyright (C) NAT SYSTEM 1991,1992,1993 All rights reserved
DATABASE open Time Elapse : 594
** SQLCCK+003 CHECKING : C:\NATSTAR\NSDB\DATA
** SQLCCK+004 KSDS SEQUENTIAL SCANNING NODE VERIFICATION **
UNUSED CI LIST(Must be empty) :
** SQLCCK+004 EMPTY LIST
TOTAL BAD SPACES : 0
** SQLCCK+000 !!! good news : PHYSICAL CHECK SUCCEED **
NS-DB Check Ended ; duration : 1906 Ms
```

Erreurs fatales

L'origine de ces problèmes graves peut être :

- Une coupure brutale d'alimentation de machine au moment d'un commit sur la base.
- Une erreur physique sur le support.
- Un bug (que nous espérons improbable !) dans le Space Manager.

Si une telle erreur est reportée, ne plus employer cette base, mais la remplacer par un backup valide.
Voici leur liste :

```
** SQLCCK 003 PHYSICAL ERRORS on  : Database_Space
** Please contact Nat System for informations
:
:
TOTAL BAD SPACES : X
NS-DB Check ended ; Duration : XXXXX Ms
```

Message de synthèse.

```
** SQLCCK 004 EMPTY CI FOUND  : No_Page
```

Une page ne contenant aucune donnée n'a pas été libérée par le space manager.

```
** SQLCCK 005 KEY NODE INVALID ORDER  : No Page
```

Classement incorrect des clés dans l'arbre (Déstructuration de l'arbre)

```
** SQLCCK 006 INVALID TREE LEVEL  : No_Page
```

N° de niveau invalide dans la page dans l'arbre balancé (Déstructuration de l'arbre)

```
** SQLCCK 007 CI INTERNAL INV KEY SEQUENCE  : No_Page
```

Classement incorrect des clés de la page

```
** SQLCCK 008 CI BAD INTERNAL STRUCTURE  : No_Page
```

Structure atypique d'une page physique.

```
UNUSED CI LIST(Must be empty)
No Page
:
:
No Page
```

Toutes les pages ne sont pas référencées dans l'arbre. Donc, perte d'une ou plusieurs branches.

```
** SQLCCK 009 SPACE OS DEPENDENT ERROR : Os_ErrNum
```

Ce message doit être analysé en fonction du contexte

Erreurs non fatales

Les messages suivants d'erreurs négatives ne remettent pas en cause la cohérence de la base.

```
** SQLCCK 001 PARAMETER MISSING
```

Oubli du paramètre database_Name.

```
** SQLCCK 002 BASIC DATABASE CATALOG ACCESS ERROR  :
```

Erreur d'ouverture de la database à scanner.

Catégories fonctionnelles de la librairie NSDB

Démarrer et terminer l'utilisation du module NSDB

Instruction DB_INIT (Librairie NSDB)

Initialise le module NSDB en mode mono-utilisateur ou en mode multi-utilisateurs.

Syntaxe	**DB_INIT** *nomDLL*			
Paramètre	nomDLL	STRING	I	nom de la DLL à utiliser

1. En mode mono-utilisateur, nomDLL doit contenir "NSxxDB".

2. En mode multi-utilisateurs, nomDLL doit contenir "NSxxDBCL" suivi du nom du serveur indiqué par la commande SysId de la section [System] du fichier d'initialisation. Les deux noms doivent être séparés par un point (".")
3. L'instruction DB_INIT doit être spécifiée avant toute instruction NSDB, en particulier avant l'instruction d'ouverture de la base par l'instruction DB_OPEN.
4. Il est important de tester DB_ERROR%(0) afin de vérifier si l'initialisation a pu être effectuée.

Ainsi l'erreur 100 "LIBRARY LOAD ERROR" indique que la librairie indiquée en paramètre n'a pu être chargée :

- soit le nom passé en paramètre est incorrect ("NS02DBB" par exemple),
- soit le nom est correct mais la librairie n'est pas trouvée (NS02DB.DLL ou NS02DBCL.DLL incorrectement installés).

L'erreur 101 "DB MODULE NOT LOADED" se produit lorsque le nom correspond à une DLL existante mais pas à NSDB ("NS02MATH" par exemple).

Exemple 1 : Mode mono utilisateur :

```
DB_INIT "NS02DB"
IF DB_ERROR%(0)<> 0
MESSAGE "Erreur DB_INIT mono utilisateur", DB_ERRMSG$(0)
EXIT
ENDIF

DB_OPEN hdb%, "base", "CREATE" USING INDIVIDU
...
DB_CLOSE hdb%

DB_STOP
```

Exemple 2 : Mode client / serveur multi utilisateurs :

```
Si dans le fichier d'initialisation, il y a dans la section [System]  :
SysId='NOMSRV';
alors, faire en NCL  :
DB_INIT "NS02DBCL.NOMSRV"
IF DB_ERROR%(0)<> 0
MESSAGE "Erreur DB_INIT multi utilisateurs", DB_ERRMSG$(0)
EXIT
ENDIF

DB_OPEN hdb%, "base", "CREATE" USING INDIVIDU
...
DB_CLOSE hdb%

DB_STOP
```

Voir aussi DB_OPEN, DB_STOP

Instruction DB_STOP (Librairie NSDB)

Clôt l'utilisation du module NSDB.

Syntaxe	DB_STOP

Exemple :

```
DB_INIT ...
DB_OPEN hdb%, ....
...
DB_CLOSE hdb%
DB_STOP
```

Voir aussi DB_CLOSE, DB_INIT

Ouvrir et fermer des bases de données

Instruction DB_CLOSE (Librairie NSDB)

Ferme une base de données.

Syntaxe	**DB_CLOSE** *handle base*			
Paramètre	handle base	INT(4)	I	handle de la base à fermer

1. DB_CLOSE ne doit pas être appelé si DB_OPEN a retourné un code d'erreur non nul.
2. DB_CLOSE effectue un commit implicite.
3. En cas d'arrêt normal de l'application (fermeture de la fenêtre), un DB_COMMIT et un DB_CLOSE sont automatiquement effectués sur chaque base restée ouverte.

Exemple :

```
DB_INIT ...
DB_OPEN hdb%, ...
...
DB_CLOSE hdb%
DB_STOP
```

Voir aussi DB_OPEN, DB_STOP

Instruction DB_OPEN (Librairie NSDB)

Ouvre une base de données NSDB.

Syntaxe	**DB_OPEN** *handle base, nom base, chaîne options USING table [, table [, ...]]*			
Paramètres	handle base	INT(4)	I/O	handle de la base
	nom base	CSTRING	I	nom de la base
	chaîne options	CSTRING	I	options d'ouverture de la base

	table	SEGMENT	I	tables d'index utilisées pour la session de travail

1. Cette instruction retourne le handle de la base dans la variable handle-base passée en premier paramètre. C'est ce handle qui doit ensuite être passé en premier paramètre des autres instructions DB_* afin de spécifier la base sur laquelle elles agissent.
2. Le handle retourné est nul si l'ouverture n'a pu être effectuée.
3. nom base est une chaîne de caractères indiquant le nom logique de la base à ouvrir, sans extension ni répertoire. Ce nom logique doit être spécifié dans le fichier d'initialisation indiqué par SET NS-DB , associé à un nom physique de fichier avec son répertoire de stockage dans le cas d'une base locale, ou associé à un nom de serveur NSDB dans le cas d'une base distante (fonctionnement en réseau).
4. chaîne options peut prendre trois valeurs
   a) "CREATE" crée la base si elle n'existe pas.
   b) "FORCECREATE" crée la base même si elle existe déjà.
   c) "" ne crée pas la base (retourne une erreur si la base n'existe pas).

Avec "FORCECREATE", si la base existait déjà, elle est toujours écrasée : les informations contenues sont donc définitivement perdues.

5. Chaque table précisé dans "USING table [, table [, ...]]" doit avoir été préalablement déclarée par SEGMENT et INDEXES.
6. Une session de travail est caractérisée par un groupe d'instructions NSDB encadrées entre deux instructions DB_OPEN et DB_CLOSE.
7. Le premier DB_OPEN doit être précédé de l'instruction d'initialisation du module NSDB, DB_INIT.
8. Il est important de vérifier le succès de l'ouverture de la base, avant toute manipulation de cette base, grâce à la fonction DB_ERROR%. Ainsi DB_ERROR% retourne 37 ("UNKNOWN DATABASE ID") si la base n'est pas déclarée dans le fichier d'initialisation.
9. Pour que plusieurs applications puissent ouvrir la même base, il faut que NSDB soit en mode multi utilisateurs. Voir DB_INIT.
10. Toute autre chaîne que "CREATE" ou "FORCECREATE" est équivalent à la chaîne vide ""."".
11. Avec "CREATE", il est possible de rajouter des tables à une base déjà existante. Ainsi, si la base a été initialement créée avec :

```
; Création de la base avec deux tables A et B
DB_OPEN ... "FORCECREATE" USING A, B
il est ensuite possible de rajouter une nouvelle table à cette base, tout
en conservant le contenu des tables existantes  :
; Ajout table C à la base existante
DB_OPEN ... "CREATE" USING A, B, C
```

```
Par contre, il est déconseillé d'ouvrir une base existante avec une
nouvelle table à créer, sans mentionner les tables existantes :
; Déconseillé ! (A, B, C devraient être
; mentionnées devant D)
DB_OPEN ... "CREATE" USING D
```

12. Lors du DB_OPEN, une vérification des segments correspondants aux tables est effectuée afin d'accepter ou refuser certains types ou tailles de champ. Deux types de champ sont refusés : CONTROL et segment imbriqué. Les tailles 2 et 4 de INT sont acceptés, la taille 1 est refusée. La taille 8 de NUM est acceptée, les tailles 4 et 10 sont refusées. Toutes les tailles de CHAR, STRING et CSTRING sont acceptées.

13. De nombreux noms de tables sont réservés par NSDB, et ne peuvent donc être employés après USING. Il s'agit de TABLE, KEY, INDEX, ORDER, SELECT, et plus généralement de tous les verbes du langage SQL.

Exemple :

```
SEGMENT INDIVIDUAL
...
ENDSEGMENT

SEGMENT invoice
...
ENDSEGMENT

INDEXES INDIVIDUAL
...
ENDINDEXES

INDEXES invoice
...
ENDINDEXES

; The database "DATA" must be declared in the
; initialization file specified by SET NS-DB.
DB_OPEN hdb%, "DATA", "CREATE" USING INDIVIDUAL, invoice
IF hdb% = 0 ; or IF DB_ERROR%(hdb%) <> 0
MESSAGE "DB_OPEN error", DB_ERRMSG$(0)
ELSE
... ; database opened successfully  !
...
DB_CLOSE hdb%
ENDIF
```

Voir aussi ***DB_INIT*****,** ***DB_CLOSE*****,** ***DB_STOP*****,** ***INDEXES***

Gérer le curseur

Instruction DB_CLOSECURSOR (Librairie NSDB)

Ferme un curseur.

Syntaxe	**DB_CLOSECURSOR** *handle curseur*			
Paramètre	handle curseur	INT(4)	I	handle du curseur

1. DB_CLOSECURSOR libère les ressources mémoires affectées à la gestion du curseur.
2. Deux méthodes sont possibles pour gérer les curseurs :
   a) Ouvrir, en début de traitement, autant de curseurs qu'il y aura d'opérations simultanées sur la base et les fermer juste avant la fermeture de la base,
   b) Ouvrir les curseurs en fonction des besoins et les fermer lorsqu'ils ne sont plus utilisés.

Exemple :

```
DB_INIT ...
DB_OPEN hdb%, ...
...
MOVE DB_OPENCURSOR%(hdb%) TO hcursor1%
MOVE DB_OPENCURSOR%(hcursor1%) TO hcursor2%
...
DB_CLOSECURSOR hcursor1%
DB_CLOSECURSOR hcursor2%
DB_CLOSE hdb%
DB_STOP
```

Voir aussi ***DB_OPENCURSOR%***

Fonction DB_OPENCURSOR% (Librairie NSDB)

Retourne un handle de curseur permettant de conserver plusieurs positions courantes lors d'opérations simultanées de recherche dans une même base.

Syntaxe	**DB_OPENCURSOR%** *(handle curseur)*			
Paramètre	handle curseur	INT(4)	I	handle de la base ou handle du curseur précédemment ouvert
Valeur retournée	INT(4)			

1. Le handle de curseur retourné est passé en paramètre aux fonctions DB_* à la place du handle de la base obtenu par DB_OPEN.
2. Le handle de la base de données doit être passé en paramètre de DB_OPENCURSOR% pour l'ouverture du premier curseur. L'ouverture des curseurs suivants s'obtient en passant en paramètre un handle de curseur déjà ouvert.
3. Des recherches simultanées peuvent également être effectuées en faisant plusieurs DB_OPEN de la même base. Les curseurs se prêtent mieux à cette opération car ils ne sont pas limités comme le handle rendu par DB_OPEN (50 au

maximum) et ne pénalisent pas le temps d'accès à l'inverse de DB_OPEN successifs.

Exemple :

```
GLOBAL INDIVIDUAL CL
GLOBAL COMPANY COMP

DB_OPEN hdb%, ...
MOVE DB_OPENCURSOR% (hdb%) TO hcursor1%
MOVE DB_OPENCURSOR% (hcursor1%) TO hcursor2%
DB_SEARCH hcursor1%, cl WHERE clientnum >= 0
WHILE DB_ERROR%(hcursor1%) = 0
DB_SEARCH hcursor2%, comp WHERE age > cl.age
WHILE DB_ERROR%(hcursor2%) = 0
...
DB_NEXT hcursor2%, soc
ENDWHILE
DB_NEXT hcursor1%, cl
ENDWHILE

DB_CLOSECURSOR hcursor1%
DB_CLOSECURSOR hcursor2%
```

Voir aussi DB_CLOSECURSOR

Chercher des enregistrements et naviguer

Instruction DB_NEXT (Librairie NSDB)

Recherche dans une base l'enregistrement suivant par rapport à la plus récente instruction DB_SEARCH, DB_PREV.

Syntaxe	**DB_NEXT** *handle base, variable segment*			
Paramètres	handle base	INT(4)	I	handle de la base
	variable segment	SEGMENT	I/O	enregistrement trouvé

1. L'enregistrement est retourné dans la variable passée en paramètre.
2. DB_NEXT permet donc de parcourir une liste en obtenant successivement les enregistrements de la liste triés selon l'index demandé.
3. Dès qu'il n'y a plus d'enregistrement correspondant aux critères initialement demandés lors du DB_SEARCH, l'instruction DB_NEXT fait retourner à DB_ERROR% l'erreur +2 ("DATABASE RECORD NOT FOUND").
4. Ne pas employer DB_NEXT sans au moins un DB_SEARCH préalable : cela provoque une erreur 29 ("SEQUENTIAL POS ERROR").
5. Le type de la variable passée en paramètre doit correspondre à un segment table d'index déclaré lors du DB_OPEN. Si ce n'est pas le cas, DB_ERROR% retourne 19 ("INEXISTING IN CATALOG").

Exemple :

```
DB_NEXT est typiquement utilisé dans des boucles de lecture séquentielle
indexée du type :
; Affichage d'une table dans une list box
DB_SEARCH hdb%, cl WHERE noclient >= 0
WHILE DB_ERROR%(hdb%) = 0
INSERT AT END cl.nom && cl.prenom TO liste
DB_NEXT hdb%, cl
ENDWHILE
```

Voir aussi DB_PREV, DB_SEARCH

Instruction DB_PREV (Librairie NSDB)

Recherche dans une base l'enregistrement précédent par rapport à la plus récente instruction DB_SEARCH, DB_NEXT.

Syntaxe	**DB_PREV** *handle base, variable segment*			
Paramètres	handle base	INT(4)	I	handle de la base
	variable segment	SEGMENT	I/O	enregistrement trouvé

1. L'enregistrement est retourné dans la variable passée en paramètre.
2. DB_PREV permet donc de "remonter" une liste en obtenant successivement les enregistrements de la liste triés selon l'index demandé.
3. Dès qu'il n'y a plus d'enregistrement correspondant aux critères initialement demandés lors du DB_SEARCH, l'instruction DB_PREV fait retourner à DB_ERROR% l'erreur +2 ("DATABASE RECORD NOT FOUND").
4. Ne pas employer DB_PREV sans au moins un DB_SEARCH et un DB_NEXT préalable. Un DB_PREV directement derrière un DB_SEARCH provoque l'erreur +2 ci dessus, alors qu'un DB_PREV sans DB_SEARCH provoque une erreur 29 ("SEQUENTIAL POS ERROR").
5. Le type de la variable passée en paramètre doit correspondre à un segment table d'index déclaré lors du DB_OPEN. Si ce n'est pas le cas, DB_ERROR% retourne 19 ("INEXISTING IN CATALOG").

Exemple :

```
DB_SEARCH hdb%, cl WHERE ...
DB_NEXT hdb%, cl ; Client X
DB_NEXT hdb%, cl ; Client Y
DB_NEXT hdb%, cl ; Client Z
DB_PREV hdb%, cl ; Return to client Y
```

Voir aussi DB_NEXT, DB_SEARCH

Instruction DB_SEARCH (Librairie NSDB)

Effectue une recherche dans une base en utilisant un ou plusieurs critères de recherche.

Syntaxe	**DB_SEARCH** *handle base, variable segment WHERE condition [, condition [, ...]]*			
Paramètres	handle base	INT(4)	I	handle de la base
	variable segment	SEGMENT	I/O	enregistrement trouvé

1. Le premier enregistrement trouvé correspondant à ce(s) critère(s) est retourné dans la variable passée en paramètre.
2. Chaque condition est composée d'un nom d'index (ou de champ non indexé) d'une table déclarée lors du DB_OPEN, suivi d'un opérateur (=, <, <=, >, >= ou ==), lui même suivi de l'expression à chercher.
3. DB_ERROR% retourne +2 ("DATABASE RECORD NOT FOUND") si aucun enregistrement ne correspond aux critères.
4. Les enregistrements suivants, selon les mêmes critères, s'obtiennent grâce à DB_NEXT.
5. L'opérateur = est un test d'inclusion, utilisable quand le type de l'index (ou du champ) est une chaîne de caractères. Une recherche du type nom='C' aboutit donc sur le premier nom commençant par la lettre C.
6. L'opérateur == est un test d'égalité stricte, utilisable quand le type de l'index (ou du champ) est une chaîne de caractères. Une recherche du type nom=='C' aboutit donc sur le premier nom strictement égal à C.
7. Les index servent uniquement à optimiser les performances d'accès. De manière générale, les recherches DB_SEARCH peuvent être faites sur n'importe quel champ du segment, indexé ou non. De plus, l'optimisation en fonction des critères multi clés est automatiquement faite par NSDB. Ainsi si "NOM, PRENOM" est un index multi clés, lors d'un "WHERE PRENOM=..., AGE=..., NOM=...", NSDB saura combiner les critères de façon à utiliser "NOM, PRENOM" comme index alors qu'il n'a pourtant pas été donné dans l'ordre par le WHERE.
8. Le type de la variable passée en paramètre doit correspondre à un segment table d'index déclaré lors du DB_OPEN. Si ce n'est pas le cas, DB_ERROR% retourne 19 ("INEXISTING IN CATALOG").
9. Par défaut, NSDB différencie les minuscules des majuscules pour les recherches de chaînes de caractères. Voir toutefois les paramètres CASESENS et CASEINSENS dans le fichier d'initialisation indiqué par SET NS DB.

Exemple 1 :

```
GLOBAL INDIVIDUAL CL

DB_SEARCH hdb%, cl WHERE clientnum >= 0
IF DB_ERROR%(hdb%) = 0
... ; first client found
ENDIF
```

Exemple 2 :

```
DB_SEARCH hdb%, cl WHERE last_name == 'SMITH', first_name == 'MICKEY'
IF DB_ERROR%(hdb%) = 0
... ; client MICKEY SMITH found
ENDIF
```

Exemple 3 :

```
DB_SEARCH hdb%, cl WHERE last_name = 'W'
IF (DB_ERROR%(hdb%) = 0
... ; there is at least one client whose last name
,...; begins with W
ELSE
MESSAGE "Warning !", "There are no last names beginning with W"
ENDIF
```

Exemple 4 :

```
DB_SEARCH hdb%, cl WHERE last_name >= 'W'
IF (DB_ERROR%(hdb%) = 0
... ; there is at least one client whose last name
... ; begins with W, X, Y or Z
ELSE
MESSAGE "Warning !", "There are no last names beginning with W, X, Y or Z"
ENDIF
```

***Voir aussi** DB_NEXT, DB_PREV, DB_OPEN, INDEXES*

Mettre à jour des enregistrements

Instruction DB_COMMIT (Librairie NSDB)

Valide toutes les modifications effectuées sur la base depuis le dernier DB_COMMIT, DB_ROLLBACK ou DB_OPEN.

Syntaxe	**DB_COMMIT** *handle base*			
Paramètre	handle base	INT(4)	I	handle de la base

1. Les modifications sur la base validées par DB_COMMIT sont celles effectuées par DB_INSERT, DB_DELETE et DB_UPDATE.
2. Pour que DB_COMMIT ait un sens, il faut que "Journal=YES ;" ait été précisé dans le fichier d'initialisation indiqué par SET NS-DB .
3. DB_COMMIT n'a aucun effet avec "Journal=NO ;" car les modifications sont toujours immédiatement prises en compte dans la base. Dans ce cas, aucun code d'erreur n'est retourné par DB_ERROR%.
4. DB_CLOSE effectue un commit implicite.
5. En cas d'arrêt normal de l'application (fermeture de la fenêtre), un DB_COMMIT et un DB_CLOSE sont automatiquement effectués sur chaque base restée ouverte.

Exemple :

```
DB_OPEN HDB%, ... ; Etat initial 0 de la base

DB_INSERT HDB%, ... ; Etat 1
DB_UPDATE HDB%, ... ; Etat 2

DB_COMMIT HDB%. ; Sauvegarde de l'état 2 de la base
; (ici un DB_ROLLBACK ferait revenir à l'état initial)

DB_UPDATE HDB%, ... ; Etat 3
DB_DELETE HDB%, ... ; Etat 4

DB_ROLLBACK HDB% ; Retour à l'état 2 de la base

DB_INSERT HDB%, ... ; Etat 5
DB_DELETE HDB%, ... ; Etat 6

DB_CLOSE HDB% ; Fermeture avec l'état 6
```

Voir aussi DB_ROLLBACK

Instruction DB_DELETE (Librairie NSDB)

Détruit un enregistrement.

Syntaxe	**DB_DELETE** *handle base, variable segment*			
Paramètres	handle base	INT(4)	I	handle de la base
	variable segment	SEGMENT	I	enregistrement à supprimer

1. DB_DELETE recherche l'enregistrement dont l'index primaire correspond à celui de la variable passée en paramètre et détruit l'enregistrement correspondant.
2. DB_DELETE entraîne la suppression de l'ensemble des index associés à ce segment.
3. Si aucun enregistrement de la base n'a cet index primaire, aucun effacement n'est effectué et DB_ERROR% retourne +2 ("DATABASE RECORD NOT FOUND").
4. DB_DELETE ne tient pas compte de la position courante maintenue par NSDB sur la base de données (DB_SEARCH, DB_NEXT, DB_PREV) et ne la modifie pas.
5. DB_DELETE ne tient pas compte des autres champs de la variable : l'enregistrement est effacé dès que l'index primaire correspond à celui de la variable, même si les autres index (ou champs non indexés) de cette variable ont des valeurs différentes.
6. Le type de la variable passée en paramètre doit correspondre à un segment table d'index déclaré lors du DB_OPEN. Si ce n'est pas le cas, DB_ERROR% retourne 19 ("INEXISTING IN CATALOG").

Exemple :

```
; Effacement du client numéro 1
MOVE 1 TO INDIVIDU(hseg%).noclient ; noclient est
; clé primaire
DB_DELETE hdb%, INDIVIDU(hseg%)
IF DB_ERROR%(hdb%) <> 0
MESSAGE "Erreur sur DB_DELETE", DB_ERRMSG$(hdb%)
ELSE
MESSAGE "OK sur DB_DELETE", "Enregistrement effacé."
ENDIF
```

***Voir aussi** DB_INSERT, DB_UPDATE*

Instruction DB_INSERT (Librairie NSDB)

Insère un enregistrement dans une base de données.

Syntaxe	**DB_INSERT** *handle base, variable segment*			
Paramètres	handle base	INT(4)	I	handle de la base
	variable segment	SEGMENT	I	enregistrement à insérer

1. L'insertion est effectuée uniquement si l'index primaire de la variable ne correspond pas à celui d'un enregistrement déjà stocké dans la base.
2. Tous les index associés à ce segment sont automatiquement insérés dans la base de données dans le B+Tree respectif de chacun de ces index.
3. Si un enregistrement existe déjà dans la base avec le même index primaire, aucune insertion n'est effectuée et DB_ERROR% retourne 5 ("DB DUPLICATE").
4. DB_INSERT ne tient pas compte de la position courante maintenue par NSDB sur la base de données (DB_SEARCH, DB_NEXT, DB_PREV) et ne la modifie pas.
5. Le type de la variable passée en paramètre doit correspondre à un segment table d'index déclaré lors du DB_OPEN. Si ce n'est pas le cas, DB_ERROR% retourne 19 ("INEXISTING IN CATALOG").

Exemple :

```
MOVE 1 TO INDIVIDU(hseg%).noclient
MOVE 'DUPONT' TO INDIVIDU(hseg%).nom
DB_INSERT hdb%, INDIVIDU(hseg%)
IF DB_ERROR%(hdb%)<> 0
MESSAGE "Erreur DB_INSERT", DB_ERRMSG$(hdb%)
ENDIF
```

***Voir aussi** DB_UPDATE, DB_DELETE*

Instruction DB_ROLLBACK (Librairie NSDB)

Annule toutes les modifications sur une base depuis le dernier DB_COMMIT, DB_ROLLBACK ou DB_OPEN.

Syntaxe	**DB_ROLLBACK** *handle base*			
Paramètre	handle base	INT(4)	I	handle de la base

1. Les modifications sur la base annulées par DB_ROLLBACK sont celles effectuées par DB_INSERT, DB_DELETE et DB_UPDATE.
2. Pour que DB_ROLLBACK ait un sens, il faut que "Journal=YES ;" ait été précisé dans le fichier d'initialisation indiqué par SET NS-DB .
3. DB_ROLLBACK n'a aucun effet avec "Journal=NO ;". Dans ce cas, aucun code d'erreur n'est retourné par DB_ERROR%.
4. DB_ROLLBACK fait perdre la position courante de DB_SEARCH, DB_NEXT, DB_PREV.
5. En cas d'arrêt anormal de l'application, un rollback est transmis automatiquement au serveur en mode multi utilisateurs. En mode mono utilisateurs, aucune action n'est effectuée, ce qui est aussi équivalent à un rollback implicite.

Voir aussi DB_COMMIT

Instruction DB_UPDATE (Librairie NSDB)

Modifie un enregistrement dans une base.

Syntaxe	**DB_UPDATE** *handle base, variable segment*			
Paramètres	handle base	INT(4)	I	handle de la base
	variable segment	SEGMENT	I	enregistrement à modifier

1. Cette instruction recherche l'enregistrement dont l'index primaire correspond à celui de la variable passée en paramètre, et le modifie avec les autres champs de cette variable.
2. Si aucun enregistrement de la base n'a cet index primaire, aucune modification n'est effectuée et DB_ERROR% retourne +2 ("DATABASE RECORD NOT FOUND").
3. DB_UPDATE entraîne la modification de l'ensemble des index associés à ce segment.
4. DB_UPDATE ne tient pas compte de la position courante maintenue par NSDB sur la base de données (DB_SEARCH, DB_NEXT, DB_PREV) et ne la modifie pas.
5. Le type de la variable passée en paramètre doit correspondre à un segment table d'index déclaré lors du DB_OPEN. Si ce n'est pas le cas, DB_ERROR% retourne 19 ("INEXISTING IN CATALOG").

Exemple :

```
; Mise à jour du client numéro 1 (changement de nom)
MOVE 1 TO INDIVIDU(hseg%).noclient ; noclient est
; clé primaire
MOVE "Guillaume" TO INDIVIDU(hseg%).nom
DB_UPDATE hdb%, INDIVIDU(hseg%)
IF DB_ERROR%(hdb%) <> 0
```

```
MESSAGE "Erreur sur DB_UPDATE", DB_ERRMSG$(hdb%)
ELSE
MESSAGE "OK sur DB_UPDATE", "Enregistrement modifié."
ENDIF
```

***Voir aussi** DB_INSERT, DB_DELETE*

Gestion d'erreurs

Fonction DB_ERROR% (Librairie NSDB)

Retourne le message d'erreur de la dernière instruction NSDB effectuée sur une base.

Syntaxe	**DB_ERROR%** *(0 | db-handle)*			
Paramètre	db-handle	INT(4)	I	handle de la base
Valeur retournée	INT(4)			

1. DB_ERROR% respecte la norme SQL : la fonction retourne 0 s'il n'y a eu aucune erreur, retourne un nombre positif pour toute erreur non fatale (l'instruction a été exécutée, mais émet un "Warning"), ou retourne un nombre négatif pour toute erreur fatale (l'instruction n'a pu être exécutée).
2. Quand le handle n'a pas de signification (sur DB_INIT, DB_OPEN, DB_STOP, DB_CLOSE), la valeur 0 doit être passée en paramètre.
3. Une erreur SQLEX correspond à une erreur d'exécution, SQLSY correspond à une erreur de syntaxe et SQLLE correspond à une erreur lexicale.
4. Toutes les erreurs SQLSY et SQLLE (et quelques erreurs SQLEX) ne peuvent survenir qu'en cas de problème interne. Contacter alors le support Nat System.
5. Le code nul (pas d'erreur) correspond au message :

Syntaxe	Cause
SUCCESSFUL DB CALL SQLEX 000	retour normal d'un appel NSDB.

6. Les codes d'erreurs fatales (<0) et leurs messages sont :

Syntaxe	Cause	Remède
PRIMARY KEY UPDATE FORBIDDEN SQLEX 110	La modification de la Primary Key est interdite.	
TIMESTAMP CONFLICT OCCURS ON UPDATE SQLEX 109	Concurrence entre plusieurs opérations de mise à jour sur un même enregistrement.	Utiliser les transactions.
CURSOR POSITION LOST (ZCOM.RESERVED) SQLEX 108	Tentative d'utiliser "where current of cursor" pour un UPDATE ou un DELETE sur une table sans faire précéder l'opération par un SELECT.	Vérifier la programmation

INVALID DB FX ON REMOTE DB SQLEX 099	Tentative interne d'appel à la fonction DB_GETCATADRESS qui ne peut s'exécuter à distance.	Erreur interne contactez Nat System.
SQL WORKDB ERROR SQLEX 093	Erreur d'accès à l'INTERNAL WORK DATABASE SYSTEME du moteur SQL.	Si le message est INIT WORKDB ERROR, vérifiez que la base de données de travail SQL (SQLWKDB) est correctement définie dans le fichier NSDB.INI. Dans les autres cas contactez Nat System.
UNDEFINED CURSOR SQLEX 092	Tentative d'exécution d'une commande de recherche NEXT ou PREV à l'aide d'un curseur qui n'a pas été défini précédemment par OPENCURSOR + SEARCH.	Vérifiez la programmation de la recherche.
MONITOR INIT ERROR SQLEX 091	Erreur d'initialisation de l'utilitaire DBMON.	
UPGRADE OPERATION ON READONLY DB SQLEX 090	Une commande de mise à jour (INSERT, MODIF...) a été passée sur une base de données définie READONLY dans le fichier NSDB.INI.	Enlevez le paramètre READONLY pour la base de données dans le fichier NSDB.INI.
NET BUFFER OVERFLOW(INTERNAL) SQLEX 089	Erreur interne.	Contactez Nat System.
REMOTE BUFFER SIZE TOO SHORT SQLEX 088	Taille du buffer de communication insuffisante.	Contactez Nat System.
INVALID NULL INDICATOR TYPE SQLEX 086	Le type d'une variable indicateur de nullité doit toujours être un entier.	Fournir un entier comme indicateur de nullité.
INTERNAL STRUCT OVERFLOW SQLEX 080	Débordement de ressources internes à un module.	Contactez Nat System.
UNDEFINED FROM TABLE SQLEX 079	La table spécifiée dans la clause FROM n'existe pas dans le CATALOG.	Vérifier le nom de la table dans la clause FROM.
INDEX SIZE OVERFLOW SQLEX 078	La somme des tailles physiques des colonnes constituant l'index dépasse 250 octets.	Réduire la taille de l'index.

NO MORE CURSOR TO CLOSE ID SQLEX 077	Tentative de fermeture d'un curseur indéfini.	Vérifiez le contexte de l'ordre de fermeture du curseur.
TOO MANY OPENED CURSORS SQLEX 076	Dépassement de capacité de la table des curseurs.	Diminuez le nombre de curseurs ouverts.
CONVERSION CONDITION RAISED SQLEX 074	Tentative de conversion de données de types incompatibles	Analysez les conversions demandées dans le statement.
NOT IMPLEMENTED SQLSY 073	Fonction non implémentée.	
SQL INTERNAL TABLE OVERFLOW SQLEX 071	Débordement d'une table interne.	Contactez Nat System.
INT USAGE MSG2 SQLEX 070	Numéro de code erreur à usage interne ; il ne doit jamais être retourné à l'application appelante.	Contactez Nat System.
ALREADY DEFINED ON NETWORK SQLEX 069	Applications utilisant un même SYSID.	Vérifier le SYSID dans le fichier NSDB.INI.
COLUMN DATA TYPE MISMATCH SQLEX 068	Type incorrect.	Vérifier le type.
INT USAGE MSG1 SQLEX 067	Numéro de code erreur à usage interne ; il ne doit jamais être retourné à l'application appelante.	Contactez Nat System.
PKEY INDEX ALREADY DEFINED SQLEX 066	Le PRIMARY KEY INDEX identifiant une table ne doit être défini qu'une fois lors de la création de la table.	Enlevez l'option PRIMARY KEY de l'ordre CREATE INDEX.
POSITION INDEX DROPPED SQLEX 065	L'index utilisé par un curseur a été supprimé (DROP) par une transaction concurrente. Le positionnement devient inutilisable.	Repositionnez le curseur en redémarrant la recherche par un SEARCH.
TABLE PKEY INDEX MISSING SQLEX 064	Création d'une table sans définition du PRIMARY KEY INDEX qui doit être obligatoirement défini pour chaque table.	Relancez l'ordre de création en défnissant la PRIMARY KEY.
PKEY INDEX CANT BE BUILT SQLEX 063	Un BUILD INDEX a été demandé pour un PRIMARY KEY INDEX ce qui n'a pas de sens.	Relancez le build index sur un index qui n'est pas PRIMARY KEY.

DBSPACE INVALID PATH SQLEX 062	Le nom de fichier physique associé au TABLE SPACE a un chemin d'accès invalide.	Changez le chemin d'accès sur un ordre CREATE TABLESPACE et relancez l'ordre. Vérifiez que vous n'avez pas spécifier un mauvais chemin d'accès dans le fichier NSDB.INI sur l'ouverture d'une base existante. Utilisez l'accès Client/Serveur (NSDBCLI) pour accéder concurremment à une base de données.
CAN'T BE DROPPED SQLEX-060	La suppression de l'objet demandé est impossible (ex : DROP TABLE SYSTABLES ; tentative de suppression d'une table SYSTEME NSDB).	Vérifiez votre ordre DROP.
CLEANUP STARTING ERROR SQLEX 059	Erreur de démarrage lors de la commande REORG.	Contactez Nat System.
BUILD STARTING ERROR SQLEX 058	Erreur de démarrage lors de la commande BUILD.	Contactez Nat System.
PHYSICAL PAGE IN USE SQLEX 057	Une transaction tente de mettre à jour une page physique qui est déjà en cours de mise à jour par une transaction concurrente. Cette erreur ne peut être renvoyée que si JOURNAL=YES est spécifié dans le fichier d'initialisation NSDB.	Attendez le COMMIT ou le ROLLBACK de la transaction concurrente pour relancer l'ordre de mise à jour.
DATABASE RECORD TOO LONG SQLEX 056	Ce code retour est utilisé de manière interne par NSDB et ne doit jamais être retourné à l'appelant.	Contactez Nat System.
INFORMATION NOT FOUND ON CATALOG SQLEX 055	Tentative de DROP d'un objet inexistant au CATALOG.	Vérifiez le nom de l'objet que vous voulez supprimer.
DEST INTG INDEX DROP NOT ALLOWED SQLEX 054	Un FOREIGN KEY INDEX ne peut pas être supprimé par une commande DROP INDEX.	

PKEY INDEX DROP NOT ALLOWED SQLEX 053	Un PRIMARY KEY INDEX ne peut pas être supprimé par une commande DROP INDEX.	Utilisez la commande DROP TABLE si vous souhaitez supprimer la TABLE et le PKEY INDEX.
(* COLUMN NUMBER *) DATA OVERFLOW SQLEX 052	Débordement de la zone de données USER lors d'une insertion ou d'une modification d'une ligne d'une table.	Vérifiez la structure des données de votre zone d'écriture. Utilisez éventuellement l'utilitaire DCLGEN pour connaître la structure exacte de SEGMENT ou de RECORD associée à une table du catalogue NSDB.
NO MORE CLIENT TRANSACTION AVAILABLE SQLEX 051	Le nombre maximum de transactions actives a été atteint dans le module NSDB CLIENT (NS02DBCL).	Diminuer le nombre de DB_OPEN ou de "start transaction" simultanés dans votre application.
CLIENT INIT FAILURE SQLEX 050	Erreur d'initialisation CLIENT en mode CLIENT SERVEUR.	Analysez l'erreur NETBIOS ou l'erreur du module de communication associée à ce message.
UNMANAGED CATALOG TABLE SQLEX 049	Erreur d'intégrité du SYSTEM CATALOG.	Contactez Nat System.
COMMUNICATION MSG OVERFLOW SQLEX 048	Dépassement de capacité sur la taille du buffer de communication. Cette erreur peut survenir si vous définissez des structures de tables particulièrement importantes induisant des lignes de plus de 8192 octets.	Utilisez des structures plus petites ; une table bien construite (respectant les règles de normalisation relationnelles) doit avoir un nombre restreint de colonnes !
DB FX INVALID ON CATALOG SQLEX 047	Les ordres de mise à jour ne peuvent pas s'appliquer aux tables du SYSTEM CATALOG (SYSCOLUMNS,SYSINDEXES...).	Utilisez les commandes du langage de description des données (D.D.L) pour manipuler ces tables.
SERVER WAITING FAILURE SQLEX 046	Erreur de communication ATTENTE SERVEUR .	Analysez l'erreur NETBIOS ou l'erreur du module de communication associée à ce message.

SERVER NOT RESPONDING SQLEX 045	Erreur de communication : attente client.	Analysez l'erreur NETBIOS ou l'erreur du module de communication associée à ce message. Ce message peut notamment survenir en cas d'arrêt brutal du SERVEUR de base de données.
SERVER NOT READY SQLEX 044	Erreur de communication : connexion serveur.	Analysez l'erreur NETBIOS ou l'erreur du module de communication associée à ce message.
SERVER CALL ERROR SQLEX 043	Erreur de communication : appel serveur.	Analysez l'erreur NETBIOS ou l'erreur du module de communication associée à ce message. Ce message peut notamment survenir en cas d'arrêt brutal du serveur de base de données après le démarrage d'une transaction à distance.
CONNECTION FAILURE SQLEX 042	Erreur d'initialisation de la couche Client/Serveur.	Analysez l'erreur NETBIOS ou l'erreur du module de communication associée à ce message.
SEMAPHORE ERROR SQLEX 041	Erreur interne de gestion de sémaphore.	Contactez Nat System.
DB RECORD LOCKED SQLEX 040	Tentative d'accès à un enregistrement verrouillé. (réservé pour extensions).	Contactez Nat System.
TRANS JOURNALING ERROR SQLEX 039	Erreur interne de journalisation transactionnelle.	Contactez Nat System.
INIT FUNCTION NOT DONE SQLEX 038	Tentative d'exécution d'une fonction NSDB sans avoir passé la fonction d'initialisation DB_INIT.	Appelez DB_INIT avant toute autre fonction NSDB .
UNKNOWN DATABASE ID SQLEX 037	Démarrage d'une transaction NSDB sur une base qui n'est pas définie dans le fichier .INI.	Vérifiez que votre base est définie dans le fichier NSDB.INI.

ENVIR STRING MISSING SQLEX 036	La variable d'environnement NS-DB = qui définit la localisation du fichier d'initialisation NSDB n'a pas été positionnée.	Passer une commande SET NS-DB =Drive:\Path\NSDB.INI dans le fichier CONFIG.SYS et relancez le système.
INIT FILE READ FAILURE SQLEX 035	Le fichier d'initialisation NS-DB n'a pu être correctement ouvert.	Vérifiez que le drive, le répertoire et le fichier pointé par la variable d'environnement NS-DB existent.
LOGICAL NAME TOO LONG SQLEX 034	La longueur maximum du nom logique d'une base de données dans le fichier .INI est de 8 octets.	Modifiez le nom logique de la base de données dans le fichier .INI et dans les fonctions DB_OPEN ou DB_STARTRN des applications.
INIT FILE EMPTY SQLEX 033	Le fichier d'initialisation existant a une taille égale à zéro.	Renseignez le contenu du fichier .INI.
INIT FILE TOO BIG SQLEX 032	La taille du fichier d'initialisation ne peut dépasser 64 Ko.	
INIT FILE OPEN FAILURE SQLEX 031	Erreur d'ouverture sur le fichier NSDB.INI.	Vérifiez que la variable d'environnement NS-DB pointe sur un fichier existant avec un chemin d'accès et un drive valide.
INTERNAL DATA OVERFLOW SQLEX 030	Saturation de la taille d'un buffer interne de travail.	Contactez Nat System.
SEQUENTIAL POS ERROR SQLEX 029	Perte de positionnement dans une recherche avant (NEXT) ou arrière (PREV).	Vérifiez que vous utilisez le curseur qui a servi lors du DB_SEARCH.
INVALID COLUMN SQLEX 028	Erreur interne.	Contactez Nat System.
PHYSICAL RECORD MISMATCH SQLEX 027	Déphasage entre la longueur d'un élément dans l'enregistrement physique d'une table et la longueur définie au SYSTEM CATALOG pour cet élément.	Contactez Nat System.
INTERNAL INDEXING ERROR SQLEX 026	Perte de cohérence dans la structure des index d'une ligne d'une table.	Contactez Nat System.
UNEXPECTED KSDS FUNCTION SQLEX 025	Erreur interne.	Contactez Nat System.

(* REFERENTIAL INTEGRITY MESS IS DYNAMICALLY BUILD *) SQLEX 024	Violation d'intégrité référentielle.	Cette erreur doit être gérée applicativement en fonction du sens de la référence absente (ex : FACTURE définie pour CLIENT inexistant...) Pour plus de détail sur l'intégrité référentielle voir la commande CREATE INDEX Index_Referential_Integrity.
INV SPACE SQLEX 023	Erreur physique d'accès à un TABLESPACE dû à un drive ou un chemin d'accès invalide.	Vérifiez les noms et les localisations des objets physiques associés au TABLESPACE de la base de données.
UNCOMPLETE CATALOG DROP SQLEX 022	Toute la mémoire allouée pour la gestion du SYSTEM CATALOG n'a pu être libérée lors de la fermeture de la base de données.	Contactez Nat System.
CATALOG INTEGRITY PB SQLEX 021	Perte d'intégrité dans le SYSTEM CATALOG au moment de son chargement en mémoire.	Contactez Nat System.
EXCEEDED SQLSY 020	Dépassement de la capacité de l'objet associé au message.	
INEXISTING IN CATALOG SQLEX 019	L'objet associé n'a pas d'existence dans le SYSTEM CATALOG.	Vérifiez le nom de l'objet et, le cas échéant, le créer à l'aide des commandes du langage de description des données (D.D.L) CREATE TABLE, CREATE INDEX ...
INVALID CLU NUMBER SQLSY 018	La valeur d'un CLUSTERING NUMBER doit être comprise entre 1 et 254.	Respectez les valeurs ci contre.
EXISTING IN CATALOG SQLEX 017	Tentative de création d'un objet déjà défini dans le SYSTEM CATALOG.	Changez le nom de l'objet et soumettez à nouveau la demande.

TRANSACTION DATA BASE NOT OPENED SQLEX 016	Avant de pouvoir travailler sur une base de données (DB_SEARCH,DB_NEXT...) il faut d'abord avoir démarré une transaction sur la base de données à l'aide de la commande DB_OPEN (NCL) ou DB_STARTRN (L3G).	Exécutez l'ordre DB_STARTRN ou DB_OPEN puis soumettez votre requête à nouveau.
SQL IDENT EXCEEDS 18 BYTES SQLSY 015	Un identificateur SQL (nom de table , nom de colonne, etc...) dépasse 18 caractères.	Réduisez la taille de l'identificateur.
UNKNOWN DATABASE FUNCTION SQLEX 014	Le code FONCTION utilisé ne correspond pas à un code fonction NSDB.	(L3G) Vérifiez que le DB_FX du DB_CALL est valide.
UNEXPECTED END OF SOURCE SQLLE 013	Terminaison prématurée d'un source SQL lors d'une compilation d'une commande SQL.	Vérifiez le source SQL transmis au compilateur SQL.
LEXICAL ERROR SQLLE 012	Erreur de compilation SQL dans la phase lexicale.	Contactez Nat System.
INVALID CONSTANT SQLLE 011	Erreur dans la structure d'une constante lors de la phase lexicale (ex : Chaîne trop longue...).	
SOURCE GREATER 64 KO SQLSY 010	Une phrase SQL ne peut dépasser 64 Ko.	Vérifiez le STATEMENT passé au compilateur SQL.
INVALID SEPARATOR SQLSY 009	Séparateur invalide.	Vérifiez le STATEMENT passé au compilateur SQL.
END OF SOURCE STREAM SQLSY 008	Fin inattendue du SOURCE SQL transmis au compilateur.	Vérifier le STATEMENT passé au compilateur SQL.
(* DYNAMICALLY BUILD *) EXPECTED SQLSY 007	Le verbe précédent EXPECTED est attendu par le compilateur dans le STATEMENT SQL à la ligne et à la colonne spécifiée.	Corrigez le STATEMENT.
(* DYNAMICALLY BUILD *) SYNTAX ERROR SQLSY 006	Le verbe précédent SYNTAX ERROR constitue une erreur de syntaxe dans le statement provoquant la production de ce message.	Corrigez le STATEMENT.
DB DUPLICATE SQLEX 005	Une ligne d'une table contenant une PRIMARY KEY identique à celle que vous tentiez d'insérer existe déjà dans la base de données.	Changez la PRIMARY KEY de la ligne et retentez l'insertion.

KSDSERR SQLEX 004	Erreur d'accès dans le module de SPACE MANAGEMENT de la base de données.	Vérifiez qu'il reste suffisamment de place sur les unités physiques accueillant les TABLESPACES de la base de données, vérifiez en utilisant l'utilitaire DBCHCK que la structure physique de la base de données n'est pas endommagée.
INV CATALOG OBJECT TYPE SQLEX 003	Type d'objet inconnu stocké dans le SYSTEM CATALOG.	Contactez Nat System.
TOO MANY TRANSACTIONS SQLEX 002	Le nombre total de transactions actuellement géré par NSDB dépasse le maximum autorisé (50 Transactions).	Diminuez le nombre de transactions actives.
TOO MANY OPENED DATA BASES SQLEX 001	Le nombre total de bases de données ouvertes par NSDB dépasse le maximum autorisé (15).	Diminuez le nombre de bases ouvertes.

7. Les codes de warning (>0) et leurs messages sont :

Syntaxe	Cause	Remède
EMPTY DATABASE SQLEX+001	Base vide	
DATABASE RECORD NOT FOUND SQLEX+002	Lors d'une opération de recherche dans la base de données (SEARCH,NEXT,PREV), plus aucun enregistrement correspondant au(x) critère(s) de recherche n'existe dans la base de données.	
IGNORED SQLEX+003	Lors d'une opération NSDB l'élément superflu précédent IGNORED et figurant dans la requête utilisateur n'a pas à être pris en compte par le système.	
CATALOG POSITION LOST SQLEX+004	Suite à une opération concurrente sur le catalogue (typiquement DROP INDEX) la position courante précédemment acquise par un curseur NSDB est perdue.	

LOCKED RECORD READ SQLEX+005	Lecture d'un enregistrement verrouillé (non implémenté pour le moment).	
INTERNAL USAGE MSG SQLEX+006	Ce message ne doit jamais être renvoyé à l'entité appelante.	
INT USAGE MSG3 SQLEX+011	Ce message ne doit jamais être renvoyé à l'entité appelante.	
TRYING TO CLOSE UNDEFINED CURSOR SQLEX+012	Tentative de fermeture d'un curseur qui n'a pas été préalablement ouvert.	
INTERNAL USAGE MSG SQLEX+016	Ce message à usage interne ne doit jamais être retourné.	Contactez Nat System.
DATA TRUNCATION OCCURS MSG SQLEX+017	Lors d'un transfert de données, ou d'une conversion en format caractère, les données utilisateurs ou les données de la colonne lue ont été tronquées.	Vérifiez que vous n'insérez pas des colonnes à partir de zones trop grandes ou que vous ne lisez pas des colonnes dans des variables de longueur insuffisante

8. Les erreurs SQLEX 052, SQLEX 024, SQLSY 007 et SQLSY 006 ont des messages variables, car construits dynamiquement.
9. Voir aussi les codes d'erreur SQLCCK xxx rapportés par l'utilitaire DBCHCK.EXE.

Exemple :

```
DB_INSERT hdb%, segment(hseg%)
IF DB_ERROR%(hdb%) <> 0
MESSAGE "Erreur sur DB_INSERT", DB_ERRMSG$(hdb%)
ELSE
MESSAGE "OK sur DB_INSERT", "Enregistrement effectué."
ENDIF
```

Voir aussi DB_ERRMSG$

Fonction DB_ERRMSG$ (Librairie NSDB)

Retourne le message d'erreur de la dernière instruction NSDB effectuée sur une base.

Syntaxe	**DB_ERRMSG$** *(0 | db-handle)*

Paramètre	db-handle	INT(4)	I	handle de la base
Valeur retournée	CSTRING			

1. Quand le handle n'a pas de signification (sur DB_INIT, DB_OPEN, DB_STOP, DB_CLOSE), la valeur 0 doit être passée en paramètre.
2. Une erreur SQLEX correspond à une erreur d'exécution, SQLSY correspond à une erreur de syntaxe et SQLLE correspond à une erreur lexicale.
3. Toutes les erreurs SQLSY et SQLLE (et quelques erreurs SQLEX) ne peuvent survenir qu'en cas de problème interne à NatStar (ou NS-DK). Contactez alors le support Nat System.
4. Les erreurs SQLEX 052, SQLEX 024, SQLSY 007 et SQLSY 006 ont des messages variables, car construits dynamiquement.
5. Voir aussi les codes d'erreur SQLCCK xxx rapportés par l'utilitaire DBCHCK.EXE.
6. Voir DB_ERROR% pour une liste détaillée des messages et codes d'erreur.

Exemple :

```
DB_INSERT hdb%, segment(hseg%)
IF DB_ERROR%(hdb%) <> 0
MESSAGE "Erreur sur DB_INSERT", DB_ERRMSG$(hdb%)
ELSE
MESSAGE "OK sur DB_INSERT", "Enregistrement effectué."
ENDIF
```

Voir aussi DB_ERROR%

Tables d'index

Instruction INDEXES (Librairie NSDB)

Déclaration d'une table d'index associée à un segment.

Syntaxe	INDEXES nom segment champ segment [, champ segment [, ...]] champ segment [, champ segment [, ...]] ... ENDINDEXES

1. Le segment a préalablement été défini par l'instruction SEGMENT.
2. nom segment peut être ensuite employé avec DB_OPEN et s'appelle une table.
3. Chaque ligne correspond à un index qui peut être ensuite employé avec DB_SEARCH.

Un index peut être mono- clé (un seul champ du segment indiqué dans la ligne) ou multi- clés (plusieurs champs du segment sont indiqués dans la ligne, séparés par des virgules).
La première ligne désigne l'index primaire : tout enregistrement dans la base doit pouvoir être identifié de façon UNIQUE grâce à cet index.

4. Les déclarations INDEXES doit être placées soit dans une librairie .NCL soit dans une ressource .SEG. Si une définition ne respecte pas cette règle (située dans un événement par exemple), le testeur de syntaxe affiche le message d'erreur "Definition not allowed here". De plus ces définitions doivent être regroupées IMPERATIVEMENT en tête de script : si tel n'est pas le cas, le message d'erreur "Invalid place for definition" apparaît.
5. Les index servent uniquement à optimiser les performances d'accès. De manière générale, les recherches DB_SEARCH peuvent être faites sur n'importe quel champ du segment, indexé ou non.
6. Un même index peut appartenir à plusieurs tables. Cela s'appelle une "jointure". Voir l'exemple suivant : l'index NOFACTURE existe dans la table INDIVIDU et dans la table FACTURE. NSDB ne gère pas les jointures, car la base n'est pas "relationnelle".
7. Dans la terminologie classique des bases de données, un champ (indexé ou nom) d'un segment table d'index est nommé colonne.

Exemple :

```
SEGMENT INDIVIDUAL
INT CLIENTNUM(4)
STRING LAST_NAME(31)
STRING FIRST_NAME(31)
ENDSEGMENT

SEGMENT INVOICE
INT INVOICENUM(4)
INT CLIENTNUM(4)
INT AMOUNT
ENDSEGMENT

; Declaration of the INDIVIDUAL table consisting of three indexes
; - CLIENTNUM is the primary key (there cannot therefore be two clients in
the database with the same number)
; - LAST_NAME is a single-key index
; - LAST_NAME, FIRST_NAME a multiple-key index
INDEXES INDIVIDUAL
CLIENTNUM
LAST_NAME
LAST_NAME, FIRST_NAME
ENDINDEXES

; Declaration of the INVOICE table consisting of two indexes
; - INVOICENUM is the primary key (there cannot therefore be two invoices
in the database with the same number)
; - CLIENTNUM is a single-key index
INDEXES INVOICE
INVOICENUM
CLIENTNUM
ENDINDEXES

DB_INIT "NS02DB"
DB_OPEN hdb%, ... USING INDIVIDUAL, INVOICE
...
```

```
DB_SEARCH hdb%, INDIVIDUAL(hseg%) WHERE LAST_NAME==...
...
```

Voir aussi DB_OPEN, DB_SEARCH, Langage NCL : SEGMENT

LIBRAIRIE NSTAS

Cette librairie permet une interaction avec Windows.

Les fonctions TAS_* sont spécifiques à Windows.

Installation

1. Déclarez NSTAS.NCL dans les librairies nécessaires au développement de votre application.
2. Vérifiez que le fichier NSxxTAS.DLL est bien dans un des répertoires du PATH sous Windows.
3. Ajoutez NSTAS.LIB aux paramètres de l'éditeur de liens (menu Options/Setup, bouton Gen … pour NatStar et menu Project/Generation options setup, bouton Linker… pour NS-DK).

Fichier NSTAS.NCL

Les verbes de la librairie NSTAS, décrits ci après, sont déclarés dans le fichier texte écrit en NCL, de nom NSTAS.NCL. Ce fichier peut aussi contenir des verbes complémentaires (API non publique). Vous pouvez donc désirer consulter ce fichier pour avoir la référence exhaustive de la librairie.
Pour consulter le fichier NSTAS.NCL :

1. Placez-vous dans le répertoire <NATSTAR>\NCL ou <NSDK>\NCL.

<NATSTAR> représente le répertoire que vous avez choisi au moment de l'installation de NatStar et <NSDK> celui de NS-DK.

2. Editez le fichier NSTAS.NCL avec n'importe quel éditeur de texte.

Description des APIs

Fonction TAS_MAIL_SIMPLESEND (Librairie NSTAS)

Envoie un message e-mail en utilisant l'interface MAPI.

Syntaxe	**TAS_MAIL_SIMPLESEND** *(handle, subject, torecip, ccrecip, bccrecip, note, filename)*			
Paramètres	handle	INT(4)	I	handle de fenêtre (self%)
	subject	CSTRING	I	sujet de message
	torecip	CSTRING	I	destinataires du message

				(séparés par des points-virgules)
	ccrecip	CSTRING	I	destinataires d'une copie du message (séparés par des points-virgules)
	bccrecip	CSTRING	I	destinataires d'une copie discrète du message (séparés par des points-virgules)
	note	CSTRING	I	contenu du message
	filename	CSTRING	I	fichiers joints (séparés par des points-virgules)
Valeur de retour	INT(4) 0 si envoi réussi e_tas_apperr : erreur d'application (mauvais argument) e_tas_outres : ressources épuisées e_implem : non-implémenté			

Erreurs MAPI :

Syntaxe	Déclaration interne
MAPI_USER_ABORT	1
MAPI_E_USER_ABORT	MAPI_USER_ABORT
MAPI_E_FAILURE	2
MAPI_E_LOGON_FAILURE	3
MAPI_E_LOGIN_FAILURE	MAPI_E_LOGON_FAILURE
MAPI_E_DISK_FULL	4
MAPI_E_INSUFFICIENT_MEMORY	5
MAPI_E_ACCESS_DENIED	6
MAPI_E_TOO_MANY_SESSIONS	8

MAPI_E_TOO_MANY_FILES	9
MAPI_E_TOO_MANY_RECIPIENTS	10
MAPI_E_ATTACHMENT_NOT_FOUND	11
MAPI_E_ATTACHMENT_OPEN_FAILURE	12
MAPI_E_ATTACHMENT_WRITE_FAILURE	13
MAPI_E_UNKNOWN_RECIPIENT	14
MAPI_E_BAD_RECIPTYPE	15
MAPI_E_NO_MESSAGES	16
MAPI_E_INVALID_MESSAGE	17
MAPI_E_TEXT_TOO_LARGE	18
MAPI_E_INVALID_SESSION	19
MAPI_E_TYPE_NOT_SUPPORTED	20
MAPI_E_AMBIGUOUS_RECIPIENT	21
MAPI_E_AMBIG_RECIP	MAPI_E_AMBIGUOUS_RECIPIENT
MAPI_E_MESSAGE_IN_USE	22
MAPI_E_NETWORK_FAILURE	23
MAPI_E_INVALID_EDITFIELDS	24
MAPI_E_INVALID_RECIPS	25
MAPI_E_NOT_SUPPORTED	26

Exemple :

```
r% = tas_mail_simplesend (self%, "subject of the message",
"tol@company.com;to2@company.com:to3@compnay.com","ccl@company.com;cc2@comp
any.com",\
"bcc1@company.com""note of the
message","c:\dir1\file1.ext1;e:\dir2\file2.ext2")
```

Fonction TAS_DBOX_GETFILENAME (Librairie NSTAS)

Ouvre les boîtes de dialogue GetOpenFileName ou GetSaveFileName de Windows.

Syntaxe	**TAS_DBOX_GETFILENAME** *(handle, action, title, path, filter, filename)*			
Paramètres	handle	INT(4)	I	handle de fenêtre (self%)
	action	INT(4)	I	d_tas_dbox_getfilename_open% ou d_tas_dbox_getfilename_save%
	title	CSTRING	I	titre de la boîte de dialogue

	path	CSTRING	I	répertoire initial et fichier initial.
	filter	CSTRING	I	liste des paires de filtres (chaînes).
	filename	CSTRING	O	nom du fichier sélectionné
Valeur de retour	INT(4) 0 si réussi e_tas_dbox_cancel : touche d'annulation choisie pour quitter la boîte de dialogue.			

1. La première chaîne d'une paire décrit le filtre (par exemple, "Text Files"), et la deuxième spécifie le motif du filtre (par exemple, "*.TXT").
2. Pour spécifier les motifs d'un filtre multiple pour une seule chaîne, il faut les séparer avec des points-virgules (par exemple: "*.TXT;*.DOC;*.BAK").
3. Une chaîne de motifs peut contenir une combinaison de caractères de noms de fichiers valides et l'astérisque (*) - caractère joker.
4. Il ne faut pas mettre des espaces dans la chaîne de motifs.
5. On sépare des paires par le caractère #2. A l'intérieur d'une paire le séparateur est le caractère #1.

Par exemple:

```
"Text Files" & #1 & "*.TXT" & #2 &
"Doc files" & #1 & "*.DOC;*.DOT" & #2 &
"All" & #1 & "*.*"
```

6. Si on ne met pas la deuxième chaîne, la première chaîne est prise comme à la fois une chaîne d'écran et comme une chaîne de motifs (par exemple, "*.exe" & # 2 & "*.*" est identique à "*.exe" & #1 & "*.exe" & #2 & "*.*" & #1 & "*.*").

Exemple :

```
r% = tas_dbox_getfilename (self%,d_tas_dbox_getfilename_save%,"Save Text
File","c:\workspace\text1.txt","Text Files" & #1 & "*.txt",textfilename)
```

Voir aussi *TAS_DBOX_GETFILENAMEEX*

Fonction TAS_DBOX_GETFILENAMEEX (Librairie NSTAS)

Ouvre les boîtes de dialogue GetOpenFileName ou GetSaveFileName de Windows.

Syntaxe	**TAS_DBOX_GETFILENAMEEX** *(handle, action, title, path, filter, filename, defaultExtension)*			
Paramètres	handle	INT(4)	I	handle de fenêtre (self%)
	action	INT(4)	I	d_tas_dbox_getfilename_open% ou d_tas_dbox_getfilename_save%
	title	CSTRING	I	titre de la boîte de dialogue

	path	CSTRING	I	répertoire initial et fichier initial.
	filter	CSTRING	I	liste des paires de filtres (chaînes).
	filename	CSTRING	O	nom du fichier sélectionné
	defaultExtension	CSTRING	I	extension par défaut
Valeur de retour	INT(4) 0 si réussi e_tas_dbox_cancel : touche d'annulation choisie pour quitter la boîte de dialogue.			

1. La première chaîne d'une paire décrit le filtre (par exemple, "Text Files"), et la deuxième spécifie le motif du filtre (par exemple, "*.TXT").
2. Pour spécifier les motifs d'un filtre multiple pour une seule chaîne, il faut les séparer avec des points-virgules (par exemple: "*.TXT;*.DOC;*.BAK").
3. Une chaîne de motifs peut contenir une combinaison de caractères de noms de fichiers valides et l'astérisque (*) - caractère joker.
4. Il ne faut pas mettre des espaces dans la chaîne de motifs.
5. On sépare des paires par le caractère #2. A l'intérieur d'une paire le séparateur est le caractère #1.

Par exemple :

```
"Text Files" & #1 & "*.TXT" & #2 &
"Doc files" & #1 & "*.DOC;*.DOT" & #2 &
"All" & #1 & "*.*"
```

6. Si on ne met pas la deuxième chaîne, la première chaîne est prise comme à la fois une chaîne d'écran et comme une chaîne de motifs (par exemple, "*.exe" & # 2 & "*.*" est identique à "*.exe" & #1 & "*.exe" & #2 & "*.*" & #1 & "*.*").
7. La différence avec TAS_DBOX_GETFILENAME est la présence du paramètre defaultExtension qui permet de rajouter automatiquement l'extension du filtre sélectionné, ou si le filtre est "*.*" l'extension passée dans ce paramètre.

Exemple :

```
r% = tas_dbox_getfilenameEx (self%,d_tas_dbox_getfilename_save%,"Save Text
File","c:\workspace\text1.txt","Text Files" & #1 & "*.txt",textfilename,
"txt" )
```

***Voir aussi** TAS_DBOX_GETFILENAME*

Fonction TAS_DBOX_GETDIRNAME (Librairie NSTAS)

Ouvre la boîte de dialogue Select Folder de Windows.

Syntaxe	**TAS_DBOX_GETDIRNAME** *(handle, title, path, dirname)*

Paramètres	handle	pointer	I	handle de fenêtre (self%)
	title	CSTRING	I	titre de boîte de dialogue
	path	CSTRING	I	répertoire initial
	dirname	CSTRING	O	nom du répertoire sélectionné (limité à 255 caractères)
Valeur de retour	INT(4) 0 si réussi e_tas_dbox_cancel : touche d'annulation choisie pour quitter la boîte de dialogue.			

1. Si vous indiquez un répertoire dans le paramètre path, vous ne pourrez pas modifier le lecteur par la suite.
2. Si vous indiquez un répertoire inexistant dans le paramètre path, la boîte Select Folder de Windows ne s'ouvre pas et vous aurez l'erreur e_tas_dbox_cancel.

Exemple :

```
Local dirname$, path$

path$ = "d:\documents"

if tas_dbox_getdirname (Self%, "Select your directory",path$, dirname$)<>
E_TAS_DBOX_CANCEL
  message "selected directory is", dirname$
endif
```

Voir aussi Fonction_TAS_DBOX_GETDIRAMEEX,

Fonction TAS_DBOX_CHOOSEFONT (Librairie NSTAS)

Ouvre la boîte de dialogue Choose Font de Windows.

Syntaxe	**TAS_DBOX_CHOOSEFONT** *(handle, title, path, dirname)*			
Paramètres	handle	INT(4)	I	handle de fenêtre (self%)
	@name	CSTRING	I/O	nom de police
	@size	INT(2)	I/O	taille de police

	@sels	INT(1)	I/O	attributs de police
	@color	INT(2)	I/O	couleur de police
Valeur de retour	INT(4) 0 si réussi e_tas_dbox_cancel : touche d'annulation choisie pour quitter la boîte de dialogue.			

Exemple :

```
local r%, name$, size%(2), sels%(1), p%, color%(2)

;tfont is a control

name$ = tfont.fontname
p% = pos% (",", name$)
if (p% > 0)
  size% = int copy$ (name$, p% + 1, 255)
  name$ = copy$ (name$, 1, p% - 1)
  tfont.text = name$
endif
sels% = tfont.fontsels
color% = tfont.forecolor

r% = tas_dbox_choosefont (self%, name$, size%, sels%, color%)
if (r% = 0)
  tfont.text = name$
  tfont.fontname = name$ & "," & size%
  tfont.fontsels = sels%
  tfont.forecolor = color%
else
  message "error", r%
endif
```

LIBRAIRIE NSHTTP

Ce document présente la bibliothèque NSHTTP.NCL qui permet d'envoyer des requêtes HTTP.

> La bibliothèque NSHTTP.NCL remplace NSNSSV.NCL qui est conservée pour compatibilité ascendante mais n'est plus à utiliser. En effet, la bibliothèque NSNSSV.NCL, contrairement à la bibliothèque NSHTTP, n'est pas portable.

Introduction

Les outils de développement Nat System proposent une bibliothèque de fonctions permettant de questionner un site web à travers le protocole HTTP ou HTTPS.

HTTP	HTTP (Hypertext Transfer Protocol) est un protocole de communication client-serveur développé pour le web. Il est utilisé pour transférer les documents (document HTML, image, feuille de style, etc.) entre le serveur HTTP et le navigateur Web lorsqu'un visiteur consulte un site Web.
HTTPS	HTTPS est la variante sécurisée du protocole HTTP. Il permet au visiteur de vérifier l'identité du site auquel il accède grâce à un certificat d'authentification.

Quelques caractéristiques de la bibliothèque NSHTTP

- Ne supporte que le protocole TCP en mode bloquant.
- Un TIMEOUT (fixé par défaut à 180 secondes) peut être modifié.
- Permet d'effectuer des requêtes HTTP 1.1.
- Seules les méthodes HTTP suivantes sont utilisables : GET, POST et HEAD.
- La bibliothèque NSHTTP gère les réponses morcelées.
- A noter que les connexions persistantes ne sont pas supportées.

Utilisation de la librairie NSHTTP

L'utilisation des requêtes HTTP nécessite l'installation du service NSHTTP.NCL.

Installation du service

Pour NatStar :

1. Installez le service NSHTTP.NCL en activant le menu Options/Services.
2. Dans la zone Type, sélectionnez la case NCL Libraries.
3. Dans la zone Services, sélectionnez la bibliothèque NSHTTP et double-cliquez sur l'élément pour le placer dans la colonne Installed.

Pour NS-DK :

1. Installez le service NSHTTP.NCL en allant dans la vue NSDK Resources du browser de ressources. A l'item Libraries, activez la touche [Ins] ou le menu

contextuel Add libraries resources into project. La fenêtre Add libraries resources to project apparaît.

2. Dans la zone Public resources, sélectionnez la bibliothèque NSHTTP et activez le bouton Apply.

Modification de la configuration pour NatWeb et NatStar

1. Modifiez votre configuration en activant le menu Options/Configuration.
2. Activez le bouton Gen ..., la fenêtre C Generator Settings apparaît.
3. Ajoutez dans les groupes (.DLL) et (.EXE), au champ Libraries : NSHTTP.LIB
4. Validez en activant le bouton OK. La fenêtre Select Default Configuration réapparaît.

Modification de la configuration pour NS-DK

1. Modifiez votre configuration en activant le menu Options/Configurations.
2. Activez l'onglet Compiler/Linker.
3. Ajoutez dans les groupes (.DLL) et (.EXE), au champ Libraries : NSHTTP.LIB
4. Validez en activant le bouton OK. La fenêtre Project generation réapparaît.

Référence de la librairie NSHTTP

Fonction NS_HTTP_EOF (Librairie NSHTTP)

Indique si le flux HTTP a été complètement lu ou non.

Syntaxe	**NS_HTTP_EOF** *(pConnection)*			
Paramètre	pConnection	POINTER	I	handle de connexion
Valeur renvoyée	INT(1) TRUE% ou FALSE%			

Exemple :

```
...
WHILE (NS_HTTP_EOF(http%) = 0)
 I% = NS_HTTP_GETLINE(http%,@line$,SIZEOF line$)
 INSERT AT END line$ TO S2
ENDWHILE
...
```

Voir aussi *NS_HTTP_GETLINE*

Fonction NS_HTTP_ERROR (Librairie NSHTTP)

Indique si une erreur a eu lieu lors du dernier appel de fonction ou d'un appel précédent.

Syntaxe	NS_HTTP_ERROR
Valeur renvoyée	INT(4)

Exemple :

```
...
http% = NS_HTTP_NEW

; connect to the given site
NS_HTTP_CONNECT http%, URL

NS_HTTP_GET http%

IF (NS_HTTP_ERROR <> 0)
ERROR = NS_HTTP_GET_ERRORMSG
ENDIF
...
```

***Voir aussi** NS_HTTP_GET_ERRORMSG*

Fonction NS_HTTP_GET_ERRORMSG (Librairie NSHTTP)

Retourne la chaîne d'erreur.

Syntaxe	NS_HTTP_GET_ERRORMSG
Valeur renvoyée	CSTRING

Exemple :

```
...
http% = NS_HTTP_NEW

; connect to the given site
NS_HTTP_CONNECT http%, URL

NS_HTTP_GET http%

IF (NS_HTTP_ERROR <> 0)
ERROR = NS_HTTP_GET_ERRORMSG
ENDIF
...
```

***Voir aussi** NS_HTTP_ERROR*

Fonction NS_HTTP_GET_HEADERFIELD (Librairie NSHTTP)

Retourne la valeur du champ spécifié.

Syntaxe	**NS_HTTP_GET_HEADERFIELD** *(pConnection, fieldname$)*			
Paramètres	pConnection	POINTER	I	handle de connexion
	fieldname$	CSTRING	I	nom du champ
Valeur renvoyée	CSTRING			

1. Retourne "" si le champ spécifié n'existe pas dans la réponse HTTP.
2. Voir la liste des champs d'en-tête dans Tableau récapitulatif des champs du Header HTTP

Exemple :

```
; create HTTP connection object
http% = NS_HTTP_NEW

; connect to the given site
NS_HTTP_CONNECT http%, URL

NS_HTTP_GET http%

IF (NS_HTTP_ERROR <> 0)
ERROR = NS_HTTP_GET_ERRORMSG
ENDIF
; get all header fields and add them to the textarea
fieldValue$ = NS_HTTP_GET_HEADERFIELD (http%, "Content-type")
INSERT AT END "the Content-type is"&&fieldValue$ TO S1

; free the http connection object
NS_HTTP_DISPOSE(http%)
```

***Voir aussi** NS_HTTP_GET_HEADERFIELDN*

Fonction NS_HTTP_GET_HEADERFIELDCOUNT (Librairie NSHTTP)

Retourne le nombre de champs HTTP présents dans la réponse HTTP.

Syntaxe	**NS_HTTP_GET_HEADERFIELDCOUNT** *(pConnection)*			
Paramètre	pConnection	POINTER	I	handle de connexion
Valeur renvoyée	INT(4)			

Exemple :

```
...
http% = NS_HTTP_NEW
NS_HTTP_CONNECT http%, URL
NS_HTTP_GET http%
```

```
IF (NS_HTTP_ERROR <> 0)
ERROR = NS_HTTP_GET_ERRORMSG
ENDIF
; get all header fields and add them to the textarea
nbHeader% = NS_HTTP_GET_HEADERFIELDCOUNT(http%)
FOR I%=0 TO nbHeader%-1
 NS_HTTP_GET_HEADERFIELDN http%, I%, fieldName$, fieldValue$
 fieldNameValue$ = fieldName$ & ": " & fieldValue$
 INSERT AT END fieldNameValue$ TO S1
ENDFOR
...
```

***Voir aussi** NS_HTTP_GET_HEADERFIELDN*

Fonction NS_HTTP_GET_REASONPHRASE (Librairie NSHTTP)

Retourne le message de retour HTTP (Reason-Phrase). Ce message de retour Reason-Phrase est un court message destiné aux utilisateurs tandis que le Status-Code est un entier à 3-digits destiné aux automates. Il n'est pas obligatoire d'afficher ou de traiter la Reason-Phrase.

Syntaxe	**NS_HTTP_GET_REASONPHRASE** *(pConnection)*			
Paramètre	pConnection	POINTER	I	handle de connexion
Valeur renvoyée	CSTRING			

Exemple :

```
...
NS_HTTP_CONNECT http%, URL
NS_HTTP_GET http%
IF (NS_HTTP_ERROR <> 0)
ERROR = NS_HTTP_GET_ERRORMSG
ENDIF
INSERT AT END "Statut
="&&NS_HTTP_GET_STATUSCODE(http%)&&NS_HTTP_GET_REASONPHRASE(http%) to MLE.
...
```

***Voir aussi** NS_HTTP_GET_STATUSCODE*

Fonction NS_HTTP_GETLINE (Librairie NSHTTP)

Lit une ligne du flux HTTP et la copie dans pszBuff. Si une ligne complète n'a pas pu être lue (parce que plus longue que buffSize%), la fonction renvoie FALSE. La partie lue est copiée dans pszBuff et le reste de la ligne sera renvoyé lors du prochain appel.

Syntaxe	**NS_HTTP_GETLINE** *(pConnection, pszBuff, buffSize%)*

Paramètres	pConnection	POINTER	I	handle de connexion
	pszBuff	POINTER	I	buffer
	buffSize%	INT(4)	I	taille du buffer
Valeur renvoyée	INT(4)			

Exemple :

```
...
WHILE (NS_HTTP_EOF(http%) = 0)
 I% = NS_HTTP_GETLINE(http%,@line$,SIZEOF line$)
 INSERT AT END line$ TO S2
ENDWHILE
...
```

Voir aussi *NS_HTTP_EOF*

Fonction NS_HTTP_GETLINEEX (Librairie NSHTTP)

Renvoie une ligne du flux HTTP sous forme de DynStr.

Syntaxe	**NS_HTTP_GETLINEEX** *(pConnection)*			
Paramètre	pConnection	POINTER	I	handle de connexion
Valeur renvoyée	DynStr			

Exemple :

```
LOCAL POINTER http%
LOCAL I%
LOCAL Dynstr line$

S1 = ""
S2 = ""

; get HTTP connection object
http% = getdata%

; connect to the given site
NS_HTTP_CONNECT http%, URL

NS_HTTP_GET http%

IF (NS_HTTP_ERROR <> 0)
   ERROR = NS_HTTP_GET_ERRORMSG
ENDIF
INSERT AT END "Statut =
"&&NS_HTTP_GET_STATUSCODE(http%)&&NS_HTTP_GET_REASONPHRASE(http%) to MLE
S1.TEXT= NS_HTTP_GET_HEADEREX (http%)
; read the content of the http response and write it to the text area or to
a file
WHILE (NS_HTTP_EOF(http%) = 0)
```

```
    line$ = NS_HTTP_GETLINEEX(http%)
    INSERT AT END line$ TO S2
ENDWHILE

IF (NS_HTTP_ERROR <> 0)
   ERROR = NS_HTTP_GET_ERRORMSG
ENDIF
; free the http connection object later
```

***Voir aussi** NS_HTTP_GETLINE, NS_HTTP_GET_HEADEREX, NS_HTTP_POSTEX, NS_HTTP_READEX*

Fonction NS_HTTP_NEW (Librairie NSHTTP)

Crée un objet HTTP. Fonction à appeler avant toute autre fonction pour initialiser les échanges.

Syntaxe	**NS_HTTP_NEW**
Valeur renvoyée	POINTER Handle sur l'objet qui va gérer les échanges

Exemple :

```
LOCAL POINTER http%
LOCAL nbHeader%,I%
LOCAL fieldName$,fieldValue$,fieldNameValue$,line$(1000)

S1 = ""
S2 = ""

; create HTTP connection object
http% = NS_HTTP_NEW

; connect to the given site
NS_HTTP_CONNECT http%, URL ; http://www.perdu.com

NS_HTTP_GET http%

IF (NS_HTTP_ERROR <> 0)
ERROR.TEXT = NS_HTTP_GET_ERRORMSG
; EXIT
ENDIF

; get all header fields and add them to the textarea
nbHeader% = NS_HTTP_GET_HEADERFIELDCOUNT(http%)
FOR I%=0 TO nbHeader%-1
 NS_HTTP_GET_HEADERFIELDN http%, I%, fieldName$, fieldValue$
 fieldNameValue$ = fieldName$ & ": " & fieldValue$
 INSERT AT END fieldNameValue$ TO S1
ENDFOR

; read the content of the http response and write it to the text area ; or
to a file
WHILE (NS_HTTP_EOF(http%) = 0)
 I% = NS_HTTP_GETLINE(http%,@line$,SIZEOF line$)
 INSERT AT END line$ TO S2
```

```
ENDWHILE

IF (NS_HTTP_ERROR <> 0)
ERROR.TEXT = NS_HTTP_GET_ERRORMSG
ENDIF

; free the http connection object
NS_HTTP_DISPOSE(http%)
```

Fonction NS_HTTP_READ (Librairie NSHTTP)

Retourne le nombre d'octets lus.

Syntaxe	**NS_HTTP_READ** *(pConnection, pBuff, buffSize%)*			
Paramètres	pConnection	POINTER	I	handle de connexion
	pBuff	POINTER	I	buffer
	BuffSize%	INT	I	taille du buffer
Valeur renvoyée	INT(4)			

1. Cette fonction permet de lire en une seule fois le flux http. Plus besoin d'itérations
2. Pour pouvoir afficher une chaîne de caractères supérieure à 255 caractères dans un contrôle, utiliser l'attribut dynamique Text. Voir l'exemple.

Exemple :

```
LOCAL POINTER http%
LOCAL nbHeader%,I%
LOCAL fieldName$,fieldValue$,fieldNameValue$,line$(2000)

S1 = ""
S2 = ""

; create HTTP connection object
http% = NS_HTTP_NEW
; connect to the given site
NS_HTTP_CONNECT http%, URL
NS_HTTP_GET http%

IF (NS_HTTP_ERROR <> 0)
ERROR = NS_HTTP_GET_ERRORMSG
ENDIF
INSERT AT END "Statut =
"&&NS_HTTP_GET_STATUSCODE(http%)&&NS_HTTP_GET_REASONPHRASE(http%) to MLE
; get all header fields and add them to the textarea
nbHeader% = NS_HTTP_GET_HEADERFIELDCOUNT(http%)
FOR I%=0 TO nbHeader%-1
 NS_HTTP_GET_HEADERFIELDN http%, I%, fieldName$, fieldValue$
 fieldNameValue$ = fieldName$ & ": " & fieldValue$
 INSERT AT END fieldNameValue$ TO S1
ENDFOR
```

```
; read the content of the http response and write it to the text area
I% = NS_HTTP_READ(http%,@line$,SIZEOF line$)
; by using the .text attribute we could transfer a > than 255 characters
string to an MLE
S2.text = DELETE$ (line$,i%+1,SIZEOF line$ - i% )
IF (NS_HTTP_ERROR <> 0)
ERROR = NS_HTTP_GET_ERRORMSG
ENDIF
; free the http connection object
NS_HTTP_DISPOSE(http%)
```

Voir aussi ***NS_HTTP_GETLINE***, ***NS_HTTP_SAVETO***, ***NS_HTTP_READEX***

Fonction NS_HTTP_READEX (Librairie NSHTTP)

Retourne le nombre d'octets lus.

Syntaxe	**NS_HTTP_READEX** *(pConnection)*			
Paramètre	pConnection	POINTER	I	handle de connexion
Valeur renvoyée	DynStr			

1. Cette fonction permet de lire en une seule fois le flux HTTP. Plus besoin d'itérations.
2. Pour pouvoir afficher une chaîne de caractères supérieure à 255 caractères dans un contrôle, utiliser l'attribut dynamique Text. Voir l'exemple.

Exemple :

```
LOCAL POINTER http%
LOCAL DynStr ds

; S1 and S2 are two MLEs
S1 = ""
S2 = ""

; create HTTP connection object
http% = NS_HTTP_NEW
; connect to the given site
NS_HTTP_CONNECT http%, URL
NS_HTTP_GET http%

IF (NS_HTTP_ERROR <> 0)
ERROR = NS_HTTP_GET_ERRORMSG
ENDIF

; get all header fields and add them to the text area
S1.TEXT = NS_HTTP_GET_HEADEREX(http%)

; read the content of the http response and write it to the text area
ds = NS_HTTP_READEX(http%)
; by using the .text attribute we could transfer a string greater than ;
255 characters to an MLE
S2.text = ds
```

```
IF (NS_HTTP_ERROR <> 0)
ERROR = NS_HTTP_GET_ERRORMSG
ENDIF
; free the http connection object
NS_HTTP_DISPOSE(http%)
```

Voir aussi ***NS_HTTP_READ***

Instruction NS_HTTP_ADDREQUESTPROPERTY (Librairie NSHTTP)

Ajoute un champ HTTP à l'en-tête de la requête. A appeler avant les méthodes GET, POST ou HEAD.

Syntaxe	**NS_HTTP_ADDREQUESTPROPERTY** *pConnection, header$, value$*			
Paramètres	pConnection	POINTER	I	handle de connexion
	header$	CSTRING	I	nom du champ à ajouter
	value$	CSTRING	I	valeur à donner au champ

Voir les valeurs possibles des champs d'en-tête dans Tableau récapitulatif des champs du Header HTTP

Exemple :

```
LOCAL POINTER http%
LOCAL nbHeader%,I%
LOCAL fieldName$,fieldValue$,fieldNameValue$,line$(1000),buf$

; create HTTP connection object
http% = NS_HTTP_NEW

; connect to the given site
NS_HTTP_CONNECT http%, URL
NS_HTTP_ADDREQUESTPROPERTY http%,"Accept-Language","fr"
buf$ = "NOM="&NOM&"&B1=Envoyer"
NS_HTTP_POST http%,@buf$,LENGTH buf$, "application/x-www-form-urlencoded"
IF (NS_HTTP_ERROR <> 0)
ERROR = NS_HTTP_GET_ERRORMSG
ENDIF
```

Instruction NS_HTTP_CONNECT (Librairie NSHTTP)

Etablit la connexion à l'URL indiquée en 2ème paramètre.

Syntaxe	**NS_HTTP_CONNECT** *pConnection, url$*

Paramètres	pConnection	POINTER	I	handle de connexion
	url$	CSTRING	I	adresse URL

Exemple :

```
LOCAL POINTER http%
LOCAL nbHeader%,I%
LOCAL fieldName$,fieldValue$,fieldNameValue$,line$(1000)

S1 = ""
S2 = ""

; create HTTP connection object
http% = NS_HTTP_NEW

; connect to the given site
NS_HTTP_CONNECT http%, URL ; http://www.perdu.com

NS_HTTP_GET http%

IF (NS_HTTP_ERROR <> 0)
ERROR.TEXT = NS_HTTP_GET_ERRORMSG
; EXIT
ENDIF

; get all header fields and add them to the textarea
nbHeader% = NS_HTTP_GET_HEADERFIELDCOUNT(http%)
FOR I%=0 TO nbHeader%-1
 NS_HTTP_GET_HEADERFIELDN http%, I%, fieldName$, fieldValue$
 fieldNameValue$ = fieldName$ & ": " & fieldValue$
 INSERT AT END fieldNameValue$ TO S1
ENDFOR

; read the content of the http response and write it to the text area or to
a file
WHILE (NS_HTTP_EOF(http%) = 0)
 I% = NS_HTTP_GETLINE(http%,@line$,SIZEOF line$)
 INSERT AT END line$ TO S2
ENDWHILE

IF (NS_HTTP_ERROR <> 0)
ERROR.TEXT = NS_HTTP_GET_ERRORMSG
ENDIF

; free the http connection object
NS_HTTP_DISPOSE(http%)
```

Instruction NS_HTTP_DISCONNECT (Librairie NSHTTP)

Cette instruction permet de se déconnecter d'un site. Cette instruction est facultative puisqu'elle est appelée automatiquement lors de l'appel à NS_HTTP_DISPOSE ou lorsqu'un nouveau NS_HTTP_CONNECT est appelé.

Syntaxe	**NS_HTTP_DISCONNECT** *pConnection*			
Paramètre	pConnection	POINTER	I	handle de connexion

Exemple :

```
LOCAL POINTER http%
LOCAL nbHeader%,I%
LOCAL fieldName$,fieldValue$,fieldNameValue$,line$(1000)

; create HTTP connection object
http% = NS_HTTP_NEW

; connect to the site
NS_HTTP_CONNECT http%, "http://www.perdu.com"

NS_HTTP_GET http%

IF (NS_HTTP_ERROR <> 0)
ERROR = NS_HTTP_GET_ERRORMSG
ENDIF
; read the content of the http response
WHILE (NS_HTTP_EOF(http%) = 0)
 I% = NS_HTTP_GETLINE(http%,@line$,SIZEOF line$)
 INSERT AT END line$ TO S2
ENDWHILE
IF (NS_HTTP_ERROR <> 0)
ERROR = NS_HTTP_GET_ERRORMSG
ENDIF
; optional
NS_HTTP_DISCONNECT http%
; free the http connection object
NS_HTTP_DISPOSE(http%)
```

Instruction NS_HTTP_DISPOSE (Librairie NSHTTP)

Détruit l'objet créé préalablement par NS_HTTP_NEW.

Syntaxe	**NS_HTTP_DISPOSE** *pConnection*			
Paramètre	pConnection	POINTER	I	handle de connexion

Exemple :

```
LOCAL POINTER http%
LOCAL nbHeader%,I%
LOCAL fieldName$,fieldValue$,fieldNameValue$,line$(1000)

S1 = ""
S2 = ""

; create HTTP connection object
http% = NS_HTTP_NEW
```

```
; connect to the given site
NS_HTTP_CONNECT http%, URL ; http://www.perdu.com

NS_HTTP_GET http%

IF (NS_HTTP_ERROR <> 0)
ERROR.TEXT = NS_HTTP_GET_ERRORMSG
; EXIT
ENDIF

; get all header fields and add them to the textarea
nbHeader% = NS_HTTP_GET_HEADERFIELDCOUNT(http%)
FOR I%=0 TO nbHeader%-1
 NS_HTTP_GET_HEADERFIELDN http%, I%, fieldName$, fieldValue$
 fieldNameValue$ = fieldName$ & ": " & fieldValue$
 INSERT AT END fieldNameValue$ TO S1
ENDFOR

; read the content of the http response and write it to the text area ; or
to a file
WHILE (NS_HTTP_EOF(http%) = 0)
 I% = NS_HTTP_GETLINE(http%,@line$,SIZEOF line$)
 INSERT AT END line$ TO S2
ENDWHILE

IF (NS_HTTP_ERROR <> 0)
ERROR.TEXT = NS_HTTP_GET_ERRORMSG
ENDIF

; free the http connection object
NS_HTTP_DISPOSE(http%)
```

Instruction NS_HTTP_GET (Librairie NSHTTP)

Envoie une requête HTTP de type GET. La méthode GET peut être utilisée pour accéder à un document localisé par une URL.

Syntaxe	**NS_HTTP_GET** *pConnection*			
Paramètre	pConnection	POINTER	I	handle de connexion

Cette fonction doit être appelée après NS_HTTP_CONNECT.

Exemple :

```
LOCAL POINTER http%
LOCAL nbHeader%,I%
LOCAL fieldName$,fieldValue$,fieldNameValue$,line$(1000)

S1 = ""
S2 = ""

; create HTTP connection object
http% = NS_HTTP_NEW

; connect to the given site
```

```
NS_HTTP_CONNECT http%, URL ; http://www.perdu.com

NS_HTTP_GET http%

IF (NS_HTTP_ERROR <> 0)
ERROR.TEXT = NS_HTTP_GET_ERRORMSG
; EXIT
ENDIF

; get all header fields and add them to the textarea
nbHeader% = NS_HTTP_GET_HEADERFIELDCOUNT(http%)
FOR I%=0 TO nbHeader%-1
 NS_HTTP_GET_HEADERFIELDN http%, I%, fieldName$, fieldValue$
 fieldNameValue$ = fieldName$ & ": " & fieldValue$
 INSERT AT END fieldNameValue$ TO S1
ENDFOR

; read the content of the http response and write it to the text area or to
a file
WHILE (NS_HTTP_EOF(http%) = 0)
 I% = NS_HTTP_GETLINE(http%,@line$,SIZEOF line$)
 INSERT AT END line$ TO S2
ENDWHILE

IF (NS_HTTP_ERROR <> 0)
ERROR.TEXT = NS_HTTP_GET_ERRORMSG
ENDIF

; free the http connection object
NS_HTTP_DISPOSE(http%)
```

Voir aussi ***NS_HTTP_POST***

Instruction NS_HTTP_GET_HEADER (Librairie NSHTTP)

Récupère le Header d'une réponse http et l'affecte a un buffer passé en paramètre.

Syntaxe	**NS_HTTP_GET_HEADER** *pHttp, pszBuff, buffSize%*			
Paramètres	pHttp	POINTER	I	handle de connexion
	pszBuf	POINTER	I	buffer
	buffSize%	INT(4)	I	taille du buffer

1. Cette instruction est utile quand un traitement d'un élément du header en particulier n'est pas demandé.
2. Il serait judicieux d'utiliser l'attribut dynamique TEXT pour affecter le contenu du buffer à une MLE si on souhaite l'afficher au lieu de faire un traitement ligne par ligne.

Exemple :

```
...
LOCAL POINTER http%
LOCAL I%, fieldName$,fieldValue$,fieldNameValue$,line$(1000)

S1 = ""
S2 = ""

; get HTTP connection object
http% = getdata%
; connect to the given site
NS_HTTP_CONNECT http%, URL
NS_HTTP_OPTIONS http%
IF (NS_HTTP_ERROR <> 0)
ERROR = NS_HTTP_GET_ERRORMSG
ENDIF
INSERT AT END "Statut = \
"&&NS_HTTP_GET_STATUSCODE(http%)&&NS_HTTP_GET_REASONPHRASE(http%) to MLE
; get all header fields and add them to the textarea
NS_HTTP_GET_HEADER http%,@line$, SIZEOF line$
S1.TEXT = line$

; read the content of the http response and write it to the text area ; I%
is the number of byte read
I% = NS_HTTP_READ(http%,@line$,SIZEOF line$)
; by using .text attribute we could transfer > than 255 characters to ; an
MLE on the condition that word wrap is checked
S2.text = DELETE$ (line$,i%+1,SIZEOF line$ - i% )

IF (NS_HTTP_ERROR <> 0)
ERROR = NS_HTTP_GET_ERRORMSG
ENDIF
...
```

***Voir aussi** NS_HTTP_GET_HEADERFIELDCOUNT, NS_HTTP_GET_HEADERFIELDN*

Instruction NS_HTTP_GET_HEADEREX (Librairie NSHTTP)

Renvoie le header d'une réponse http sous forme de DynStr.

Syntaxe	**NS_HTTP_GET_HEADEREX** *pHttp*			
Paramètre	pHttp	POINTER	I	handle de connexion
Valeur renvoyée	DYNSTR			

1. Cette instruction est utile quand un traitement d'un élément du header en particulier n'est pas demandé.
2. Il est recommandé d'utiliser l'attribut dynamique TEXT pour affecter le contenu de la DynStr à une MLE si on souhaite l'afficher.

Exemple 1 :

```
LOCAL POINTER http%
LOCAL DynStr ds
```

```
; S1 and S2 are two MLEs
S1 = ""
S2 = ""

; create HTTP connection object
http% = NS_HTTP_NEW
; connect to the given site
NS_HTTP_CONNECT http%, URL
NS_HTTP_GET http%

IF (NS_HTTP_ERROR <> 0)
ERROR = NS_HTTP_GET_ERRORMSG
ENDIF

; get all header fields and add them to the text area
S1.TEXT = NS_HTTP_GET_HEADEREX(http%)

; read the content of the http response and write it to the text area
ds = NS_HTTP_READEX(http%)
; by using the .text attribute we could transfer a string greater than 255
characters to an MLE
S2.text = ds

IF (NS_HTTP_ERROR <> 0)
ERROR = NS_HTTP_GET_ERRORMSG
ENDIF
; free the http connection object
NS_HTTP_DISPOSE(http%)
```

Exemple 2 :

```
LOCAL POINTER http%
LOCAL DYNSTR buf$, DYNSTR line$

S1 = ""
S2 = ""

; get HTTP connection object
http% = getdata%

; connect to the given site
NS_HTTP_CONNECT http%, URL
NS_HTTP_ADDREQUESTPROPERTY http%,"Accept-Language","fr"
NS_HTTP_SET_FOLLOWREDIRECT http%, FALSE%
buf$ = "NOM="&NOM&"&B1=Envoyer"
NS_HTTP_POSTEX http%, buf$, "application/x-www-form-urlencoded"
IF (NS_HTTP_ERROR <> 0)
   ERROR = NS_HTTP_GET_ERRORMSG
ENDIF
INSERT AT END "Statut =
"&&NS_HTTP_GET_STATUSCODE(http%)&&NS_HTTP_GET_REASONPHRASE(http%) to MLE

; get all header fields and add them to the text area
S1.TEXT= NS_HTTP_GET_HEADEREX (http%)

; read the content of the http response and write it to the text area or to
a file
```

```
    S2.text = NS_HTTP_READEX(http%)
IF (NS_HTTP_ERROR <> 0)
   ERROR.TEXT = NS_HTTP_GET_ERRORMSG
ENDIF

; free the http connection object later
```

Voir aussi ***NS_HTTP_GET_HEADER*****,** ***NS_HTTP_GETLINEEX*****,** ***NS_HTTP_POSTEX*****,** ***NS_HTTP_READEX***

Instruction NS_HTTP_GET_HEADERFIELDN (Librairie NSHTTP)

Retourne le champ HTTP dont le numéro est précisé. Le nom du champ sera retourné dans fieldName$, la valeur du champ dans fieldValue$.

Syntaxe	**NS_HTTP_GET_HEADERFIELDN** *pConnection, n%, fieldName$, fieldValue$*			
Paramètres	pConnection	POINTER	I	handle de connexion
	n%	INT	I	numéro de champ HTTP
	fieldname$	CSTRING	O	nom du champ
	fieldValue$	CSTRING	O	valeur du champ

Exemple :

```
...
http% = NS_HTTP_NEW
NS_HTTP_CONNECT http%, URL
NS_HTTP_GET http%

IF (NS_HTTP_ERROR <> 0)
ERROR = NS_HTTP_GET_ERRORMSG
ENDIF
; get all header fields and add them to the textarea
nbHeader% = NS_HTTP_GET_HEADERFIELDCOUNT(http%)
FOR I%=0 TO nbHeader%-1
 NS_HTTP_GET_HEADERFIELDN http%, I%, fieldName$, fieldValue$
 fieldNameValue$ = fieldName$ & ": " & fieldValue$
 INSERT AT END fieldNameValue$ TO S1
ENDFOR
...
```

Voir aussi ***NS_HTTP_GET_HEADERFIELD*****,** ***NS_HTTP_GET_ HEADERFIELDCOUNT***

Instruction NS_HTTP_GET_STATUSCODE (Librairie NSHTTP)

Retourne le status-code de la réponse HTTP.

Syntaxe	**NS_HTTP_GET_STATUSCODE** *pConnection*			
Paramètre	pConnection	POINTER	I	handle de connexion

1. Le status code est un code numérique d'erreur pouvant être interprété par des automates. Il y a 5 catégories de status code :
   a) 1xx: Information -- requête arrivée, le traitement se poursuit.
   b) 2xx: Succès -- la requête a été bien reçue, comprise et acceptée.
   c) 3xx: Redirection – d'autres actions doivent suivrent pour compléter la requête. Par défaut, lorsque le status-code est un code de redirection, la requête est automatiquement redirigée. Il est possible de désactiver ce comportement.
   d) 4xx: Erreur client - la requête contient une mauvaise syntaxe ou ne peut pas être traitée.
   e) 5xx: Erreur serveur - Le serveur n'a pas pu traiter une requête apparemment valide.
2. Voir la liste exhaustive des Status code dans Tableau récapitulatif des status codes.

Exemple :

```
LOCAL POINTER http%
LOCAL nbHeader%,I%
LOCAL fieldName$,fieldValue$,fieldNameValue$,line$(1000)

S1 = ""
S2 = ""

; create HTTP connection object
http% = NS_HTTP_NEW

; connect to the given site
NS_HTTP_CONNECT http%, URL ; http://www.perdu.com

NS_HTTP_GET http%

IF (NS_HTTP_ERROR <> 0)
ERROR.TEXT = NS_HTTP_GET_ERRORMSG
; EXIT
ENDIF
INSERT AT END "Statut = " && NS_HTTP_GET_STATUSCODE(http%) &&
NS_HTTP_GET_REASONPHRASE(http%) to MLE
; get all header fields and add them to the textarea
nbHeader% = NS_HTTP_GET_HEADERFIELDCOUNT(http%)
FOR I%=0 TO nbHeader%-1
 NS_HTTP_GET_HEADERFIELDN http%, I%, fieldName$, fieldValue$
 fieldNameValue$ = fieldName$ & ": " & fieldValue$
 INSERT AT END fieldNameValue$ TO S1
ENDFOR

; read the content of the http response and write it to the text area or to
a file
WHILE (NS_HTTP_EOF(http%) = 0)
 I% = NS_HTTP_GETLINE(http%,@line$,SIZEOF line$)
 INSERT AT END line$ TO S2
```

```
ENDWHILE

IF (NS_HTTP_ERROR <> 0)
ERROR.TEXT = NS_HTTP_GET_ERRORMSG
ENDIF

; free the http connection object
NS_HTTP_DISPOSE(http%)
```

Voir aussi ***NS_HTTP_GET_REASONPHRASE***

Instruction NS_HTTP_HEAD (Librairie NSHTTP)

Envoie une requête HTTP de type HEAD.

Syntaxe	**NS_HTTP_HEAD** *pConnection*			
Paramètre	pConnection	POINTER	I	handle de connexion

1. La méthode HEAD est identique à GET sauf que le serveur ne doit pas renvoyer un message-body dans la réponse. La méthode HEAD est souvent utilisée pour tester les liens hypertexte quant à leur validité, accessibilité, ou récente modification.
2. Cette fonction doit être appelée après NS_HTTP_CONNECT.

Exemple :

```
LOCAL POINTER http%
LOCAL nbHeader%,I%
LOCAL fieldName$,fieldValue$,fieldNameValue$,line$(1000)

S1 = ""
S2 = ""

; create HTTP connection object
http% = NS_HTTP_NEW

; connect to the given site
NS_HTTP_CONNECT http%, URL ; http://www.perdu.com

NS_HTTP_HEAD http%

IF (NS_HTTP_ERROR <> 0)
ERROR.TEXT = NS_HTTP_GET_ERRORMSG
; EXIT
ENDIF

; get all header fields and add them to the textarea
nbHeader% = NS_HTTP_GET_HEADERFIELDCOUNT(http%)
FOR I%=0 TO nbHeader%-1
 NS_HTTP_GET_HEADERFIELDN http%, I%, fieldName$, fieldValue$
 fieldNameValue$ = fieldName$ & ": " & fieldValue$
 INSERT AT END fieldNameValue$ TO S1
ENDFOR
```

```
; free the http connection object
NS_HTTP_DISPOSE(http%)
```

Voir aussi ***NS_HTTP_POST*****,** ***NS_HTTP_GET***

Instruction NS_HTTP_OPTIONS (Librairie NSHTTP)

Envoie une requête HTTP de type OPTIONS. La méthode OPTIONS permet de déterminer les options ou les requis pour une ressource web ou les capacités d'un serveur sans impliquer une demande de fourniture de ressources a ce serveur.

Syntaxe	**NS_HTTP_OPTIONS** *pHttp*			
Paramètre	pHttp	POINTER	I	handle de connexion

Cette fonction doit être appelée après NS_HTTP_CONNECT.

Exemple :

```
...
LOCAL POINTER http%
LOCAL I%, fieldName$,fieldValue$,fieldNameValue$,line$(1000)

S1 = ""
S2 = ""

; get HTTP connection object
http% = getdata%
; connect to the given site
NS_HTTP_CONNECT http%, URL
; Methode OPTIONS
;The OPTIONS method represents a request for information about the
;communication ;options available on the request/response chain identified
by ;the Request-URI. ;This method allows the client to determine the
options ;and/or requirements ;associated with a resource, or the
capabilities of a ;server, without implying a resource action or initiating
a resource retrieval

NS_HTTP_OPTIONS http%

IF (NS_HTTP_ERROR <> 0)
ERROR = NS_HTTP_GET_ERRORMSG
ENDIF
INSERT AT END "Statut =
"&&NS_HTTP_GET_STATUSCODE(http%)&&NS_HTTP_GET_REASONPHRASE(http%) to MLE
; get all header fields and add them to the textarea
NS_HTTP_GET_HEADER http%,@line$, SIZEOF line$
S1.TEXT = line$

; read the content of the http response and write it to the text area I% is
; the number of byte read
I% = NS_HTTP_READ(http%,@line$,SIZEOF line$)
; by using .text attribute we could transfer > than 255 characters to an
MLE on ; the condition that word wrap is checked
S2.text = DELETE$ (line$,i%+1,SIZEOF line$ - i% )
```

```
IF (NS_HTTP_ERROR <> 0)
ERROR = NS_HTTP_GET_ERRORMSG
ENDIF
...
```

Voir aussi ***NS_HTTP_GET*****,** ***NS_HTTP_TRACE*****,** ***NS_HTTP_POST*****,** ***NS_HTTP_HEAD***

Instruction NS_HTTP_POST (Librairie NSHTTP)

Envoie une requête HTTP en utilisant la méthode POST. A la différence de la méthode GET, les données sont envoyées dans le corps (après l'en-tête CONTENT-LENGTH) de la requête et non avec l'URL.

Syntaxe	**NS_HTTP_POST** *pConnection, pbuff, bufflen%, contenttype$*			
Paramètres	pConnection	POINTER	I	handle de connexion
	pbuff	POINTER	I	pointe vers les données à envoyer
	bufflen%	INT(4)	I	taille des données à envoyer
	contenttype$	CSTRING	I	CONTENT-TYPE des données à envoyer. Par exemple "text/xml"

1. Cette instruction doit être appelée après NS_HTTP_CONNECT.
2. Pour une requête POST, les champs sont transmis derrière l'en-tête MIME CONTENT-LENGTH, ils sont séparés entre eux par le caractère & (à ne pas confondre avec le & du NCL qui sert à concaténer des chaînes), les espaces sont remplacés par des +, les éventuelles lettres accentuées et caractères &, / ... sont encodés et remplacés par leur code ANSI précédé du caractère % (exemple 'à' par %E0).
   En réceptionnant cette requête POST, le serveur HTTP ciblé passera le contrôle à la procédure citée en argument en lui transmettant les champs du formulaire sur son entrée standard. Ce procédé est propre à la méthode POST, dans le cas de la méthode GET les champs sont récupérés dans une variable d'environnement.

Exemple :

```
LOCAL POINTER http%
LOCAL nbHeader%,I%
LOCAL fieldName$,fieldValue$,fieldNameValue$,line$(1000),buf$
```

```
S1 = ""
S2 = ""

; create HTTP connection object
http% = NS_HTTP_NEW

; connect to a Natweb application at URL
;http://localhost/Scripts/cgifcgi.exe/nstest/LFORM.HFORM.1.103.17
NS_HTTP_CONNECT http%, URL
NS_HTTP_ADDREQUESTPROPERTY http%,"Accept-Language","fr"
buf$ = "NOM="&NOM&"&B1=Envoyer"
NS_HTTP_POST http%,@buf$,LENGTH buf$, "application/x-www-form-urlencoded"
IF (NS_HTTP_ERROR <> 0)
ERROR.TEXT = NS_HTTP_GET_ERRORMSG
ENDIF
Status = NS_HTTP_GET_STATUSCODE(http%)
; get all header fields and add them to the textarea
nbHeader% = NS_HTTP_GET_HEADERFIELDCOUNT(http%)
FOR I%=0 TO nbHeader%-1
 NS_HTTP_GET_HEADERFIELDN http%, I%, fieldName$, fieldValue$
 fieldNameValue$ = fieldName$ & ": " & fieldValue$
 INSERT AT END fieldNameValue$ TO S1
ENDFOR

; read the content of the http response and write it to the text area or to
a file
WHILE (NS_HTTP_EOF(http%) = 0)
 I% = NS_HTTP_GETLINE(http%,@line$,SIZEOF line$)
 INSERT AT END line$ TO S2
ENDWHILE

IF (NS_HTTP_ERROR <> 0)
ERROR.TEXT = NS_HTTP_GET_ERRORMSG
ENDIF

; free the http connection object
NS_HTTP_DISPOSE(http%)
```

***Voir aussi** NS_HTTP_GET, NS_HTTP_POSTEX*

Instruction NS_HTTP_POSTEX (Librairie NSHTTP)

Envoie une requête HTTP en utilisant la méthode POST. A la différence de la méthode GET, les données sont envoyées dans le corps (après l'en-tête CONTENT-LENGTH) de la requête et non avec l'URL.

Syntaxe	**NS_HTTP_POSTEX** *pConnection, ds, contenttype$*			
Paramètres	pConnection	POINTER	I	handle de connexion
	ds	DYNSTR	I	contient les données à envoyer

	contenttype$	CSTRING	I	CONTENT-TYPE des données à envoyer. Par exemple "text/xml"

1. Cette instruction doit être appelée après NS_HTTP_CONNECT.
2. Pour une requête POST, les champs sont transmis derrière l'en-tête MIME CONTENT-LENGTH, ils sont séparés entre eux par le caractère & (à ne pas confondre avec le & du NCL qui sert à concaténer des chaînes), les espaces sont remplacés par des +, les éventuelles lettres accentuées et caractères &, / ... sont encodées et remplacées par leur code ANSI précédé du caractère % (exemple 'à' par %E0). En réceptionnant cette requête POST, le serveur HTTP ciblé passera le contrôle à la procédure citée en argument en lui transmettant les champs du formulaire sur son entrée standard. Ce procédé est propre à la méthode POST, dans le cas de la méthode GET les champs sont récupérés dans une variable d'environnement.

Exemple :

```
Local dynstr URL$
local dynstr ds
Local dynstr dsRet
LOCAL pointer http%

URL$ = "http://nstest.natsys.fr/infos.php"
ds =
"Nom=NAT%20SYSTEM&Ville=LA%20ROCHELLE&Departement=17&Produit=NSDK&Version=V
5"

; create HTTP connection object
http% = NS_HTTP_NEW

; connect to the given site
NS_HTTP_CONNECT http%, URL$

; Send data with POST Method
NS_HTTP_POSTEX http%, ds, "application/x-www-form-urlencoded"

IF (NS_HTTP_ERROR <> 0)
 INSERT AT END NS_HTTP_GET_ERRORMSG TO MLELOG
else
 insert at end "data succesfully send" to MLELOG
ENDIF

; read the content of the http response and write it to the text area
dsRet = NS_HTTP_READEX(http%)

IF (NS_HTTP_ERROR <> 0)
 INSERT AT END NS_HTTP_GET_ERRORMSG TO MLELOG
else
 MLELOG.TEXT = dsRet
```

```
ENDIF

; free the http connection object
NS_HTTP_DISPOSE(http%)
```

Voir aussi ***NS_HTTP_CONNECT, NS_HTTP_POST***

Instruction NS_HTTP_SAVETO (Librairie NSHTTP)

Sauvegarde le fichier pointé par l'adresse URL dans le fichier indiqué.

Syntaxe	**NS_HTTP_SAVETO** *pConnection, pszFile$*			
Paramètres	pConnection	POINTER	I	handle de connexion
	pszFile$	CSTRING	I	nom du fichier
Valeur retournée	INT(1)			

A utiliser à la place de NS_HTTP_READ.

Voir aussi ***NS_HTTP_READ***

Instruction NS_HTTP_SET_FOLLOWREDIRECT (Librairie NSHTTP)

Indique si les redirections indiquées dans les headers HTTP doivent être suivies ou non.

Syntaxe	**NS_HTTP_SET_FOLLOWREDIRECT** *pConnection, follow%*			
Paramètres	pConnection	POINTER	I	handle de connexion
	follow%	INT(1)	I	True% ou False%

Exemple :

```
...
http% = NS_HTTP_NEW
NS_HTTP_SET_FOLLOWREDIRECT http%, FALSE%
NS_HTTP_CONNECT http%, URL
...
```

Instruction NS_HTTP_SET_PASSWORD (Librairie NSHTTP)

Permet d'inidquer le mot de passe pour l'authentification Basic ou Digest.
Le nom de l'utilisateur et le mot de passe peuvent également être définis au niveau de l'URL de la forme http://myuser:mypassword@host/path/ comme paramètre de NS_HTTP_CONNECT.

Syntaxe	**NS_HTTP_SET_PASSWORD** *pConnection, password$*			
Paramètres	pConnection	POINTER	I	handle de connexion
	password$	CSTRING	I	mot de passe de l'utilisateur

Voir aussi *NS_HTTP_SET_USERNAME*, *NS_HTTP_CONNECT*

Instruction NS_HTTP_SET_PROXY (Librairie NSHTTP)

Positionne un proxy.

Syntaxe	**NS_HTTP_SET_PROXY** *pConnection, proxy$*			
Paramètres	pConnection	POINTER	I	handle de connexion
	proxy$	CSTRING	I	nom du proxy

proxy$ doit être de la forme : http://…
Exemple :

```
...
http% = NS_HTTP_NEW
NS_HTTP_SET_PROXY http%, 'http://localhost:8081'
NS_HTTP_CONNECT http%, URL
...
```

Voir aussi *NS_HTTP_SET_PROXYUSERNAME*

Instruction NS_HTTP_SET_PROXYPASSWORD (Librairie NSHTTP)

Permet d'indiquer le mot de passe pour l'authentification Basic ou Digest du proxy.
Le nom de l'utilisateur et le mot de passe peuvent également être définis au niveau de l'URL du proxy de la forme http://myuser:mypassword@host/path/ comme paramètre de NS_HTTP_SET_PROXY.

Syntaxe	**NS_HTTP_SET_PROXYPASSWORD** *pConnection, password$*			
Paramètres	pConnection	POINTER	I	handle de connexion
	password$	CSTRING	I	mot de passe de l'utilisateur

Voir aussi *NS_HTTP_SET_PROXY*, *NS_HTTP_SET_PROXYUSERNAME*

Instruction NS_HTTP_SET_PROXYUSERNAME (Librairie NSHTTP)

Permet d'indiquer le nom d'utilisateur pour l'authentification Basic ou Digest du proxy. Le nom de l'utilisateur et le mot de passe peuvent également être définis au niveau de l'URL du proxy de la forme http://myuser:mypassword@host/path/ comme paramètre de NS_HTTP_SET_PROXY.

Syntaxe	**NS_HTTP_SET_PROXYUSERNAME** *pConnection, username$*			
Paramètres	pConnection	POINTER	I	handle de connexion
	username$	CSTRING	I	nom de l'utilisateur

***Voir aussi** NS_HTTP_SET_PROXY, NS_HTTP_SET_PROXYPASSWORD*

Instruction NS_HTTP_SET_ROOTCERTS (Librairie NSHTTP)

Permet d'indiquer le fichier contenant les certificats des autorités de certification. Ce fichier doit être au format PEM. Il est utilisé pour vérifier la validité du certificat envoyé par le serveur lors d'une requête HTTPS.
Par défaut, le fichier contenant les certificats des "certificates authorities" (CA) (fourni avec NS-DK, NatStar, NatWeb), utilisé pour les requêtes HTTPS, se nomme rootcerts.pem. Il est situé dans :

```
;(=EXE);(NS-CFG);(NS-INI);(PATH)"
```

Syntaxe	**NS_HTTP_SET_ROOTCERTS** *pConnection, CAFile$*			
Paramètres	pConnection	POINTER	I	handle de connexion
	CAFile$	CSTRING	I	fichier des CA au format PEM

Fonction NS_HTTP_GET_ROOTCERTS (Librairie NSHTTP)

Permet de récupérer le nom du fichier contenant les certificats des autorités de certification.

Syntaxe	**NS_HTTP_GET_ROOTCERTS** *(pConnection)*			
Paramètre	pConnection	POINTER	I	handle de connexion
Valeur renvoyée	CSTRING			

Instruction NS_HTTP_SET_TIMEOUT (Librairie NSHTTP)

Modifie la valeur du TIMEOUT.

Syntaxe	**NS_HTTP_SET_TIMEOUT** *pConnection, timeout%*			
Paramètres	pConnection	POINTER	I	handle de connexion
	timeout%	INT(4)	I	timeout en millisecondes.

Le TIMEOUT par défaut est de 180 secondes.
Exemple :

```
...
http% = NS_HTTP_NEW
NS_HTTP_SET_TIMEOUT http%, 120000 ; 2 minutes
NS_HTTP_CONNECT http%, URL
...
```

Instruction NS_HTTP_SET_USERNAME (Librairie NSHTTP)

Permet d'indiquer le nom d'utilisateur pour l'authentification Basic ou Digest.
Le nom de l'utilisateur et le mot de passe peuvent également être définis au niveau de l'URL de la forme http://myuser:mypassword@host/path/ comme paramètre de NS_HTTP_CONNECT.

Syntaxe	**NS_HTTP_SET_USERNAME** *pConnection, username$*			
Paramètres	pConnection	POINTER	I	handle de connexion
	username$	CSTRING	I	nom de l'utilisateur

***Voir aussi** NS_HTTP_SET_PASSWORD, NS_HTTP_CONNECT*

Instruction NS_HTTP_TRACE (Librairie NSHTTP)

Envoie une requête HTTP de type TRACE. La méthode TRACE permet au client au sens http du terme de savoir ce qui a été reçu du coté serveur et utiliser cette information à des fins de tests ou de diagnostiques.

Syntaxe	**NS_HTTP_TRACE** *pHttp*			
Paramètre	pHttp	POINTER	I	handle de connexion

Cette fonction doit être appelée après NS_HTTP_CONNECT.
Exemple :

```
...
LOCAL POINTER http%
LOCAL I%,fieldName$,fieldValue$,fieldNameValue$,line$(1000)

S1 = ""
S2 = ""

; get HTTP connection object
http% = getdata%
; connect to the given site
NS_HTTP_CONNECT http%, URL
;TRACE allows the client to see what is being received at the other ;end of
the ;request chain and use that data for testing or diagnostic
;information.
;The value of the Via header field (section 14.45) is of particular
;interest, since it acts as a trace of the request chain. Use of the ;Max-
Forwards header ;field allows the client to limit the length of ;the
request chain, which is ;useful for testing a chain of proxies ;forwarding
messages in an infinite loop.
;If the request is valid, the response SHOULD contain the entire ;request
message in the entity-body, with a Content-Type of ;"message/http".

NS_HTTP_TRACE http%

IF (NS_HTTP_ERROR <> 0)
ERROR = NS_HTTP_GET_ERRORMSG
ENDIF
INSERT AT END "Statut =
"&&NS_HTTP_GET_STATUSCODE(http%)&&NS_HTTP_GET_REASONPHRASE(http%)\
to MLE
; get all header fields and add them to the textarea
NS_HTTP_GET_HEADER http%,@line$, SIZEOF line$
S1.TEXT = line$

; read the content of the http response and write it to the text area ; I%
is the number of byte read
I% = NS_HTTP_READ(http%,@line$,SIZEOF line$)
; by using .text attribute we could transfer > than 255 characters to ; an
MLE on the condition that word wrap is checked
S2.text = DELETE$ (line$,i%+1,SIZEOF line$ - i% )

IF (NS_HTTP_ERROR <> 0)
ERROR = NS_HTTP_GET_ERRORMSG
ENDIF
...
```

***Voir aussi** NS_HTTP_GET, NS_HTTP_OPTIONS, NS_HTTP_POST, NS_HTTP_HEAD*

Tableau récapitulatif des champs du Header HTTP

Nom de l'en-tête	Description
Accept	Type de contenu accepté par le browser (par exemple text/html).
Accept-Charset	Jeu de caractères attendu par le browser.

Accept-Encoding	Codage de données accepté par le browser
Accept-Language	Langage attendu par le browser (anglais par défaut)
Authorization	Identification du browser auprès du serveur
Content-Encoding	Type de codage du corps de la requête
Content-Language	Type de langage du corps de la requête
Content-Length	Longueur du corps de la requête
Content-Type	Type de contenu du corps de la requête (par exemple text/html).
Date	Date de début de transfert des données
Forwarded	Utilisé par les machines intermédiaires entre le browser et le serveur
From	Permet de spécifier l'adresse e-mail du client
From	Permet de spécifier le renvoi du document s'il a été modifié depuis une certaine date
Link	Relation entre deux URLs
Orig-URL	URL d'origine de la requête
Referer	URL du lien à partir duquel la requête a été effectuée
User-Agent	Chaîne donnant des informations sur le client, comme le nom et la version du navigateur, du système d'exploitation

Tableau récapitulatif des status codes

Code	Message	Description
10x	Message d'information	Ces codes ne sont pas utilisés dans la version 1.0 du protocole.
20x	Réussite	Ces codes indiquent le bon déroulement de la transaction.
200	OK	La requête a été accomplie correctement.
201	CREATED	Elle suit une command POST, elle indique la réussite, le corps du reste du document est sensé indiquer l'URL à laquelle le document nouvellement créé devrait se trouver.
202	ACCEPTED	La requête a été acceptée, mais la procédure qui suit n'a pas été accomplie.
203	PARTIAL INFORMATION	Lorsque ce code est reçu en réponse à une commande GET, cela indique que la réponse n'est pas complète.

204	NO RESPONSE	Le serveur a reçu la requête mais il n'y a pas d'informations à renvoyer.
205	RESET CONTENT	Le serveur indique au navigateur de supprimer le contenu des champs d'un formulaire
205	PARTIAL CONTENT	Il s'agit d'une réponse à une requête comportant l'en-tête Range. Le serveur doit indiquer l'en-tête Content-Range.
30x	Redirection	Ces codes indiquent que la ressource n'est plus à l'emplacement indiqué.
301	MOVED	Les données demandées ont été transférées a une nouvelle adresse.
302	FOUND	Les données demandées sont à une nouvelle URL, mais ont cependant peut-être été déplacées depuis...
303	METHOD	Cela implique que le client doit essayer une nouvelle adresse, en essayant de préférence une autre méthode que GET.
304	NOT MODIFIED	Si le client a effectué une commande GET conditionnelle (en demandant si le document a été modifié depuis la dernière fois) et que le document n'a pas été modifié il renvoie ce code.
40x	Erreur due au client	Ces codes indiquent que la requête est incorrecte.
Code	Message	Description
400	BAD REQUEST	La syntaxe de la requête est mal formulée ou est impossible à satisfaire.
401	UNAUTHORIZED	Le paramètre du message donne les spécifications des formes d'autorisation acceptables. Le client doit reformuler sa requête avec les bonnes données d'autorisation.
402	PAYMENT REQUIRED	Le client doit reformuler sa demande avec les bonnes données de paiement.
403	FORBIDDEN	L'accès à la ressource est tout simplement interdit.
404	NOT FOUND	Le serveur n'a rien trouvé à l'adresse spécifiée.
50x	Erreur due au serveur	Ces codes indiquent qu'il y a eu une erreur interne du serveur.
500	INTERNAL ERROR	Le serveur a rencontré une condition inattendue qui l'a empêché de donner suite à la demande .
501	NOT IMPLEMENTED	Le serveur ne supporte pas le service demandé.

502	BAD GATEWAY	Le serveur a reçu une réponse invalide de la part du serveur auquel il essayait d'accéder en agissant comme une passerelle ou un proxy.
503	SERVICE UNAVAILABLE	Le serveur ne peut pas vous répondre à l'instant présent, car le trafic est trop dense.
504	GATEWAY TIMEOUT	La réponse du serveur a été trop longue vis à vis du temps pendant lequel la passerelle était préparée à l'attendre.

LIBRAIRIES NWXML ET NSTHXML

Ce chapitre décrit les fonctionnalités XML des outils NatStar et NS-DK.

Ce chapitre suppose la connaissance des langages XML et HTML.

Une application Nat System utilisant NWXML.DLL ne peut fonctionner que si le poste sur lequel elle est exécutée peut charger la DLL Microsoft MSVCRT.DLL de version supérieure ou égale à 6. Par conséquent, l'utilisateur a la responsabilité de mettre ce poste à jour en installant une version récente d'Internet Explorer par exemple ou en installant Vcredist.exe.

Vcredist.exe est contenu dans VC6RedistSetup.exe. Vous trouverez VC6RedistSetup.exe sur le CD dans le répertoire tools\vc. VC6RedistSetup.exe affiche le Contrat de Licence Utilisateur Final. Lorsque vous acceptez le contrat, le fichier Vcredist.exe est extrait.Vcredist.exe installe les fichiers de base, qui sont compris avec Visual Studio 6.0 SP4. Ces fichiers sont des dépendances de base pour la plupart des composants et des applications créées avec Visual C++ 6.0.

Introduction

XML s'impose de plus en plus comme le langage standard de diffusion d'information sur le Web et d'échanges de données entre les applications. Le langage XML n'étant pas basé sur une application, un outil, une base de données ou une version en particulier, la transmission et l'échange de données structurées s'en trouvent facilitées.

XML étant une technologie relativement récente, les différents browsers y réagissent différemment. Nat System vous recommande l'utilisation d'Internet Explorer 5.X qui offre actuellement de meilleurs résultats.

Le langage XML

NatStar et NS-DK permettent de traiter des documents XML, c'est-à-dire de les interroger, de les lire et de les analyser. Il permet également de produire des documents XML et de les transmettre à des navigateurs et à des applications capables d'interpréter ce langage.
Le langage XML a été élaboré pour décrire le contenu de documents et leurs données. Les données sont structurées par des balises qui représentent les nœuds de l'arbre XML.

Règles d'écriture

Les règles de balisage du langage XML, très apparentées à celles du langage HTML, servent à décrire les données correctement. Celles-ci incluent :

- L'imbrication correcte des balises.

Les balises ne peuvent se présenter comme suit : <A><B></A></B>. Elles doivent suivre le modèle <A><B></B></A>

- Les balises ouvrantes indiquant un champ vide comportent une barre oblique avant le crochet droit.

<PRENOM/> équivaut à <PRENOM></PRENOM>

- À toute balise ouvrante doit correspondre une balise fermante.

Contrairement au langage HTML, XML ne permet pas d'utiliser des balises fermantes isolées telles que </p>.

Le langage XML étant constitué d'un ensemble de normes à partir desquelles on crée ses propres balises, il rend possible l'attribution d'une structure et d'une forme à des concepts représentés par des données.

Il permet, par exemple, de créer une structure réunissant le nom, l'adresse et le numéro de téléphone d'un client en vue de décrire cette structure et les liens entre les données qui la composent. Cette information regroupée dans un document peut ensuite être partagée avec n'importe quelle application capable d'interpréter ce langage.

Exemple :

```
<CLIENT>
 <NOMCLIENT>CyberApplications</NOMCLIENT>
 <ADRESSE>
  <RUE> 18, rue de la Ferté St Jean </RUE>
  <CODEPOSTAL> 75 900 </CODEPOSTAL>
  <VILLE> PARIS </VILLE>
 </ADRESSE>
 <TELEPHONE> 01 42 66 99 77 </TELEPHONE>
</CLIENT>

<CLIENT>
 <NOMCLIENT>Wu Game</NOMCLIENT>
 <ADRESSE>
  <RUE> 42, rue Beauvais </RUE>
  <CODEPOSTAL> 75 563 </CODEPOSTAL>
  <VILLE> PARIS </VILLE>
 </ADRESSE>
 <TELEPHONE> 01 56 23 88 54 </TELEPHONE>
</CLIENT>
```

XML sert à décrire des données. Ses utilisateurs n'ayant pas à se préoccuper du formatage des documents, il leur évite d'avoir à deviner ce que signifient des balises telles que <OL>. Les balises XML sont habituellement faciles à lire et à comprendre. Par exemple, <NOMCLIENT> est une balise pour le nom du client.

Outre la création de balises, XML permet de joindre des attributs à des balises pour ainsi mieux maîtriser la description de l'information contenue dans les documents.

Exemple :

L'attribut REGION permet d'indiquer une information supplémentaire sur le client. Ici, l'attribut REGION de l'élément CLIENT a la valeur IdF pour Ile de France.

```
<CLIENT REGION = " IdF ">
  <NOMCLIENT>CyberApplications</NOMCLIENT>
</CLIENT>
```

Définition de types de documents

Un document XML est une arborescence qui décrit la structure du document. Cette structure peut être déclarée directement dans le corps du document XML ou dans un fichier à part selon une Définition de Type de Document (DTD) ou XSD.

Lorsqu'un document XML possède un fichier DTD ou XSD associé et la respecte, on dit qu'il est valide. Lorsqu'il respecte seulement les règles de la grammaire XML (balises fermées, correctement imbriquées…) on dit qu'il est bien formé. Les définitions de type de document constituent un ensemble de règles gouvernant la structure des documents XML. Les définitions de type de document spécifient :

- Les balises contenues dans les documents
- Les balises pouvant contenir d'autres balises
- Le nombre et la séquence des balises
- Les attributs joints aux balises
- La valeur des attributs

Bien qu'elles ne soient pas obligatoires, les définitions de type de document sont très utiles. Elles présentent, entre autres, les avantages suivants :

- L'exactitude du traitement et de l'analyse des documents.

Le processeur XML (ou le programme) détecte les anomalies et signale les erreurs à l'application.

- La prestation de valeurs par défaut.

La définition de type de document indique la valeur par défaut des attributs aux applications interprétant le langage XML.

- L'utilisation de documents XML par d'autres personnes.

L'association d'une définition de type de document à un document XML permet à n'importe quelle application et à n'importe quel navigateur d'interpréter celui-ci grâce à l'établissement de règles qui en décrivent le contenu et les attributs.

La présentation des documents XML

Le langage XML permet donc de structurer l'information qui devient clairement identifiable quelque soit le type de terminal utilisé. Par exemple, le numéro de téléphone de notre feuille XML sur les clients pourra être identifié par n'importe quelle application compatible XML.

Cette séparation entre le contenu et la présentation des documents devient nécessaire dans un contexte multi-support (navigateurs Internet, téléphones mobiles…). En effet, cette séparation permet de proposer une présentation spécifique adaptée aux capacités ergonomiques et techniques des différents supports de lecture.

Les normes XSL (eXtended Stylesheet Language) et XSLT (XSL Transformations) permettent de transformer un même document initial XML en plusieurs fichiers spécifiques : pages HTML, pages WML, fichier PDF, fichier texte…

Structure d'un document XML

Un document XML est composé de deux parties :

- Un prologue qui peut être constitué des éléments optionnels suivants :
  - Une déclaration XML,
  - Une ou plusieurs instructions de traitements,
  - Une partie déclarative (DTD ou XSD).
- Le corps du document qui contient le contenu balisé avec :
  - Un élément racine unique,
  - Des nœuds,
  - Des commentaires.

Exemple :

```
; DEBUT DU PROLOGUE
; déclaration XML
<?xml version="1.0"  encoding="ISO-8859-1" standalone="yes"?>
; nœud de type ProcessingInstruction
<?xml-stylesheet type="text/xsl" href="seminaire.xsl"?>
; nœud de type DocumentType
; DEBUT DE LA PARTIE DECLARATIVE
<!DOCTYPE seminaire [
; nœud de type Entity
 <!ENTITY organisme "Nat System S.A.">
 <!ELEMENT seminaire ( date, titre, societe, programme )>
 <!ELEMENT date ( #PCDATA )>
 <!ELEMENT titre ( #PCDATA )>
 <!ELEMENT societe ( #PCDATA )>
 <!ELEMENT programme ( chapitre+ )>
 <!ELEMENT chapitre ( #PCDATA )>
 <!ATTLIST chapitre numero NMTOKEN #REQUIRED>
]>
; FIN DE LA PARTIE DECLARATIVE
; FIN DU PROLOGUE
; DEBUT DU CORPS DU DOCUMENT
; nœud de type Element
<seminaire>
 <date/>
 <titre>
; nœud de type CDATA
  <![CDATA[Concepts de base des développements en <XML> ]]>
 </titre>
; "&organisme;" correspond à un nœud EntityReference
 <societe>Organisée par &organisme;</societe>
; nœud de type Comment
<!-- Programme du séminaire -->
 <programme>
; Les balises <chapitre numero="x"> correspondent à des nœuds de type
; Element
; Le contenu des balises <chapitre numero="x"> … </chapitre> correspond ; à
des nœuds de type Text
  <chapitre numero="1">Introduction</chapitre>
  <chapitre numero="2">Concepts de base</chapitre>
  <chapitre numero="3">Le langage XML</chapitre>
```

```
  <chapitre numero="4">Les espaces de nom</chapitre>
  <chapitre numero="5">La transformation par XSL</chapitre>
  <chapitre numero="6">Le modèle de document objet</chapitre>
  <chapitre numero="7">Les standards XML connexes</chapitre>
  <chapitre numero="8">Quelques standards complémentaires</chapitre>
  <chapitre numero="9">Les modèles d'architecture</chapitre>
  <chapitre numero="10">Les outils du marché</chapitre>
 </programme>
</seminaire>
; FIN DU CORPS DU DOCUMENT
```

Type de nœuds

DOM présente les documents XML sous forme de nœuds.
Il y a douze types de nœuds. Certains nœuds peuvent avoir des nœuds enfants. Les types possibles des nœuds enfants sont indiqués en italique.

- les nœuds racines (root ou Document) : Element ProcessingInstruction, Comment, DocumentType
- les nœuds d'éléments (Element) : Element, Text, Comment, ProcessingInstruction, CDATASection, EntityReference
- les nœuds de texte (Text)
- les nœuds attributs (Attribute) : Text, EntityReference
- les nœuds de section CDATA (CDATASection)
- les nœuds d'instructions de traitement (ProcessingInstruction)
- les nœuds commentaires (Comment)
- Les nœuds notation (Notation)
- Les nœuds entité (Entity) : Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
- Les nœuds d'entité référencée (EntityReference) : Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
- Les nœuds de fragment de document (DocumentFragment) : Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
- Les nœuds de type de document (DocumentType)

Gestion des flux XML

Parseur XML

NatStar et NS-DK intègrent un parseur C++ d'Apache qui permet de valider le contenu d'un fichier DTD ou XSD et de traiter les données XML provenant de ressources différentes (buffer, fichier système, URL).

Librairie DOM

NatStar et NS-DK proposent la librairie DOM pour manipuler des données XML.

Le parseur stocke un flux de données XML traité en mémoire sous forme d'une arborescence. Les APIs DOM permettent ensuite de le manipuler, comme par l'extraction des informations par une requête XML et la construction d'une réponse.
Le Document Object Model (DOM) est un modèle standard de présentation XML associé à une interface d'accès au modèle pour modifier les contenus, les structures et les styles d'un document :

- La création de documents XML, de ses éléments, de ses attributs
- La manipulation de documents XML, des éléments, des attributs
- La recherche d'information dans les documents
- L'export d'un document XML dans un buffer ou dans un fichier

Librairies XML

Ces librairies permettent de manipuler des fichiers XML.
Les fonctions et instructions du référentiel sont répertoriées dans le fichier NWXML.NCL.

Installation

NSDK : Installez NWXML.NCL dans les librairies du menu Project / Resources de NSDK.
NatStar : Installez les fichiers NWXML.NCL et NSTHXML.NCL en tant que service en exécutant les opérations suivantes :

1. Activez la commande Options \ Services. La boîte de dialogue Modify Services s'ouvre.
2. Sélectionnez NSTHXML et NWXML dans la colonne Available et activez la flèche vers la droite.
3. NSTHXML et NWXML apparaissent dans la colonne Installed.
4. Cliquez sur le bouton Close.

Catégories fonctionnelles de la librairie NWXML

Gestion du moteur XML

Les deux API suivantes sont indispensables pour assurer la portabilité de vos documents XML.

L'initialisation du moteur XML est obligatoire et s'effectue au moyen de la fonction NWX_INITIALIZE%. De même, il est nécessaire pour libérer les ressources allouées au moteur XML d'utiliser l'instruction NWX_TERMINATE.

Fonction NWX_INITIALIZE% (Librairie NWXML)

La fonction NWX_INITIALIZE% initialise le moteur XML. L'appel de cette fonction est obligatoire avant tout traitement XML ou XSL.

Syntaxe	**NWX_INITIALIZE%**
Valeur renvoyée	INT

1. La valeur retournée sera différente de 0 si une erreur s'est produite lors de l'initialisation.
2. L'espace mémoire allouée par cette fonction est désallouée par NWX_TERMINATE%.

Exemple 1 :

```
local pointer pDoc, pointer pParser
local ok%

ok% = NWX_INITIALIZE%

if ok%=0
pDoc =NWX_DOCUMENT_NEW ("XMLDoc")
pParser =NWX_PARSER_NEW

ok%= NWX_DOCUMENT_LOAD%(pDoc, "mydocxml.xml",pParser)
if ok%
;Traitement du document
endif
NWX_DOCUMENT_DISPOSE pDoc
NWX_PARSER_DISPOSE pParser
 NWX_ TERMINATE
Endif
```

Exemple 2 :

```
local ok%
ok% = NWX_INITIALIZE%
if ok%=0
NWX_TRANSFORMFILE "mydocxml.xml", "mydocxsl.xsl","mydochmtl.htm"
 NWX_TERMINATE
Endif
```

Voir aussi NWX_ TERMINATE

Instruction NWX_TERMINATE (Librairie NWXML)

La fonction NWX_TERMINATE libère les ressources utilisées par le moteur XML.

Syntaxe	**NWX_ TERMINATE**

Les ressources du moteur XML ont été allouées à l'appel de la fonction NWX_INITIALIZE%.

Exemple :

```
local pointer pDoc, pointer pParser
local ok%
```

```
ok% = NWX_INITIALIZE%
if ok%=0
pDoc =NWX_DOCUMENT_NEW ("XMLDoc")
pParser =NWX_PARSER_NEW
ok%= NWX_DOCUMENT_LOAD(pDoc, "myxml.xml",pParser)
if ok%
;Traitement du document
endif
NWX_DOCUMENT_DISPOSE pDoc
NWX_PARSER_DISPOSE pParser
 NWX_TERMINATE
endif
```

Voir aussi NWX_INITIALIZE%

Gestion du parseur

Instruction NWX_PARSER_DISPOSE (Librairie NWXML)

L'instruction NWX_PARSER_DISPOSE permet de fermer un parseur XML.

Syntaxe	**NWX_PARSER_DISPOSE** *pParser*			
Paramètre	pParser	POINTER	I/O	parseur à fermer

Exemple :

```
local pointer pParser
local pointer pPdoc
local ok%(1)
local ok2%

ok2%=NWX_INITIALIZE%

pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")

ok% = NWX_DOCUMENT_LOADFILE%(pDoc, "c:\XML\mydoc.XML", pParser)
if ok% = false%
  ; erreur
 else
  ; faire quelque chose avec pDoc
endif

NWX_PARSER_DISPOSE pParser
```

Voir aussi NWX_PARSER_NEW

Fonction NWX_PARSER_GETDOCUMENT (Librairie NWXML)

La fonction NWX_PARSER_GETDOCUMENT permet de retourner un pointeur vers le document courant. Cette fonction est utilisée pour faciliter l'analyse de plusieurs documents.

Syntaxe	**NWX_PARSER_GETDOCUMENT** *(pParser)*			
Paramètre	pParser	POINTER	I/O	pointeur du parseur
Valeur renvoyée	POINTER sur le document courant.			

Exemple :

```
local pointer pLast
local ok%

ok%=NWX_INITIALIZE%
pLast = NWX_PARSER_GETDOCUMENT (pParser)
```

Voir aussi NWX_PARSER_NEW

Fonction NWX_PARSER_ISVALID (Librairie NWXML)

La fonction NWX_PARSER_ISVALID permet de vérifier si un parseur est en mode de validation ou non. L'instruction NWX_PARSER_SETMODE nous permet de modifier ce mode.

Syntaxe	**NWX_PARSER_ISVALID** *(pParser)*			
Paramètre	pParser	POINTER	I/O	pointeur du parseur
Valeur renvoyée	INT(1) TRUE% ou FALSE%.			

Exemple :

```
local pointer pParser
local ok%

ok%=NWX_INITIALIZE%
;Nouveau parseur
pParser = NWX_PARSER_NEW
message "PARSEUR", NWX_PARSER_ISVALID (pParser)
;Le message affichera 0 car le parseur par défaut n'est pas ;en mode
validation.
```

Voir aussi NWX_PARSER_SETMODE

Fonction NWX_PARSER_NEW (Librairie NWXML)

La fonction NWX_PARSER_NEW permet de créer un parseur XML.

Syntaxe	**NWX_PARSER_NEW**

Valeur renvoyée	POINTER sur le parseur créé

1. Vous pouvez utiliser ce parseur pour charger plusieurs documents XML.
2. Vous devez fermer le parseur en utilisant NWX_PARSER_DISPOSE .

Exemple :

```
local pointer parser
local pointer doc
local ok%(1)
local ok2%

ok2%=NWX_INITIALIZE%

pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")

ok% = NWX_DOCUMENT_LOADFILE%(pDoc, "C:\XML\mydoc.XML", pParser)
if ok% = false%
  ; erreur
 else
  ;faire qqch avec pDoc
endif
NWX_PARSER_DISPOSE pParser
```

Voir aussi NWX_DOCUMENT_LOADFILE%, NWX_DOCUMENT_NEW, NWX_PARSER_DISPOSE

Instruction NWX_PARSER_SETCREATEENTREFNODES (Librairie NWXML)

L'instruction NWX_PARSER_SETCREATEENTREFNODES indique au parseur comment réagir lorsqu'il rencontre un nœud EntityReference dans l'arborescence DOM.

Syntaxe	**NWX_PARSER_SETCREATEENTREFNODES** *pParser, create%*			
Paramètres	pParser	POINTER	I/O	pointeur du parseur
	create%	INT(14)	I	état (TRUE% ou FALSE%)

1. La valeur par défaut est false%.
2. Si create%=true%, le parseur conserve le nœud EntityReference et intègre son contenu comme nœud enfant Text. Cependant, ce nœud Text est visible uniquement dans l'arbre DOM, pas en sortie.
3. Si create%=false%, le nœud EntityReference est remplacé par son contenu.
4. Le noeud EntityReference est utile pour insérer une référence à une entité sans qu'elle soit transformée par le parseur XML. Le résultat de l'insertion se traduit par la chaîne suivante : « & », le contenu du paramètre name$, suivi de « ; ».

Exemple :

```
local pointer pParser, pointer pDoc
local ok%(1), value$(2024)
local pointer pNode
local ok2%

ok2%=NWX_INITIALIZE%

;Nouveau parseur
pParser = NWX_PARSER_NEW
;Il faut d'abord mettre le parseur en mode Validation
NWX_PARSER_SETMODE pParser, 1
;Créer des noeuds d'entités
NWX_PARSER_SETCREATEENTREFNODES pParser,true%
;Nouveau document
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
;Charger le document XML
ok% = NWX_DOCUMENT_LOADFILE%(pDoc, "D:\XMLTest\auto.xml", pParser)

if NOT ok%
message 'erreur', ok%
endif

pNode =NWX_DOCUMENT_GETROOT (pDoc)
ok% = NWX_NODE_OUTPUT% (pNode,@value$, sizeof value$)

; Libère le parseur et le document
NWX_PARSER_DISPOSE pParser
NWX_DOCUMENT_DISPOSE pDoc
  Fichier source :
<?xml version="1.0" encoding= "ISO-8859-1" standalone = "yes"?>
<!DOCTYPE AUTO [
<!ENTITY modele "TOYOTA">
<!ELEMENT AUTO (MODELE,ANNEE)>
<!ELEMENT MODELE (#PCDATA)>
<!ELEMENT ANNEE (#PCDATA)> ]>

<AUTO>
<MODELE>
&modele; Camry
</MODELE>
<ANNEE>
1999
</ANNEE>
</AUTO>
  Résultat partiel :
; NWX_PARSER_SETCREATEENTREFNODES étant à true%, on
; conserve le nœud EntityReference et sa valeur

<AUTO>
<MODELE>
&modele; Camry
</MODELE>
<ANNEE>
1999
</ANNEE>
```

```
</AUTO>
  Résultat partiel :
; NWX_PARSER_SETCREATEENTREFNODES étant à false%,
; le nœud EntityReference est remplacé par sa valeur

<AUTO>
<MODELE>
TOYOTA Camry
</MODELE>
<ANNEE>
1999
</ANNEE>
</AUTO>
```

Voir aussi NWX_PARSER_NEW, NWX_ENTITYREFERENCE_NEW

Instruction NWX_PARSER_SETDONAMESPACES (Librairie NWXML)

Un document XML peut utiliser des éléments définis dans différentes DTD ou XSD. Pour différencier ces éléments, on peut préciser l'espace de noms auxquels ils appartiennent.
L'instruction NWX_PARSER_SETDONAMESPACES permet aux utilisateurs de définir si le parseur effectuera ou non le traitement des espaces de noms (NameSpaces).

Syntaxe	**NWX_PARSER_SETDONAMESPACES** *pParser, newState%*			
Paramètres	pParser	POINTER	I/O	pointeur du parseur
	newState%	INT(4)	I/O	état (TRUE% ou FALSE%)

1. Pour plus d'informations sur les espaces de noms (NameSpaces), reportez-vous au glossaire ou consultez les recommandations W3C (http://www.w3.org).
2. Un espace de noms est déclaré à l'aide de l'attribut xmlns : <séminaire xmlns="http://www.natsystem.com/seminaire">
3. La portée d'une déclaration est limitée au sous-arbre dans lequel la déclaration a été faite.
4. La valeur par défaut est false%.
5. Si newstate%=true% alors le parseur impose les contraintes et règles des NameSpaces tels que déterminés dans les spécifications XML.
6. Pour valider un fichier XML suivant un fichier XSD, positionner les deux instructions suivantes NWX_PARSER_SETDONAMESPACES et NWX_PARSER_SETMODE à TRUE%.

Exemple :

```
Local pointer pParser
local ok%

ok%=NWX_INITIALIZE%

pParser = NWX_PARSER_NEW
;Pour que le parseur tienne compte des espaces de noms
;il suffit de passer true% comme deuxième paramètre à cette
;fonction
NWX_PARSER_SETDONAMESPACES pParser,true%
```

Voir aussi NWX_PARSER_NEW

Instruction NWX_PARSER_SETEXITONFIRSTERROR (Librairie NWXML)

L'instruction NWX_PARSER_SETEXITONFIRSTERROR permet de déterminer le comportement du parseur quand il rencontre une erreur fatale.

Syntaxe	**NWX_PARSER_ SETEXITONFIRSTERROR** *pParser, newState%*			
Paramètres	pParser	POINTER	I/O	pointeur du parseur
	newState%	INT(4)	I/O	état (TRUE% ou FALSE%)

1. La valeur par défaut est true%
2. Si newstate%=true% alors à la première erreur fatale rencontrée, le parseur arrête son exécution.
3. Si newstate%=false% alors le parseur continue son exécution. Pour spécifier le fichier utilisé par le parseur pour enregistrer les warnings et erreurs, utiliser l'instruction NWX_PARSER_SETMSGFILE.

Exemple :

```
local pointer pParser
local pointer pDoc
local ok%
local pointer pChild
local ok2%

ok2%=NWX_INITIALIZE%

;Nouveau parseur vide
pParser = NWX_PARSER_NEW
;Mettre le parseur en mode de validation
NWX_PARSER_SETMODE pParser,1

NWX_PARSER_SETEXITONFIRSTERROR pParser,1
NWX_PARSER_SETMSGFILE pParser, "C:\Xml\ErreurNew.txt"
;Nouveau document vide
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
```

```
;Charger le document XML pour associer à pDoc et pParser
ok% = NWX_DOCUMENT_LOADFILE%(pDoc, "C:\Xml\mydoc.xml", pParser)
```

Voir aussi ***NWX_PARSER_SETMSGFILE***

Instruction NWX_PARSER_SETINCIGNWHITESPACE (Librairie NWXML)

En saisissant des documents XML, il est souvent nécessaire d'utiliser des blancs (espaces, tabulations et interlignes) pour faciliter la lecture. Cependant, tous ces blancs ne sont pas significatifs. L'instruction NWX_PARSER_SETINCIGNWHITESPACE (Set Include/Ignore Whitespaces) permet de spécifier si le parseur validé doit inclure les espaces blancs comme nœuds de type texte. Cela n'a aucun effet sur les parseurs non-validés qui incluent toujours du texte non-balisé.

Syntaxe	**NWX_PARSER_ SETINCIGNWHITESPACE** *pParser, include%*			
Paramètres	pParser	POINTER	I/O	pointeur du parseur
	include%	INT(1)	I	état (TRUE% ou FALSE%)

1. La valeur par défaut est true%
2. Si include%=true% alors les espaces blancs hors des balises sont conservés.
3. Si include%=false% alors les espaces blancs hors des balises ne sont pas conservés.

Exemple :

```
local pointer pParser
local pointer pDoc
local ok%
local value$
local ok2%

ok2%=NWX_INITIALIZE%

;Nouveau parseur
pParser = NWX_PARSER_NEW
;Il faut d'abord rendre le parseur en mode Validation
NWX_PARSER_SETMODE pParser, true%
;Il faut ensuite dire au parseur de ne plus tenir compte des
;espaces blancs
NWX_PARSER_SETINCIGNWHITESPACE pParser, false%

;Ensuite on peut déclarer un nouveau document
pDoc = NWX_DOCUMENT_NEW("XMLdoc")

;Charger le document XML
ok% = NWX_DOCUMENT_LOADFILE%(pDoc, "C:\Xml\mydoc.xml", pParser)
```

```
if ok%
ok% = NWX_DOCUMENT_OUTPUT% (pDoc, @value$, sizeof value$)
else
;Erreur
endif
```

Voir aussi NWX_PARSER_NEW, NWX_PARSER_SETMODE

Instruction NWX_PARSER_SETMODE (Librairie NWXML)

L'instruction NWX_PARSER_SETMODE spécifie si le parseur valide le buffer ou le fichier suivant une DTD donnée ou un XML Schéma (fichier XSD).

Syntaxe	**NWX_PARSER_SETMODE** *pParser, Mode%*			
Paramètres	pParser	POINTER	I/O	pointeur du parseur
	Mode%	INT(1)	I/O	mode (TRUE% ou FALSE%)

1. Par défaut, le parseur n'est pas en mode validation (FALSE%)
2. Si le parseur est en mode validation (TRUE%), le document .xml doit avoir une DTD ou un XSD associé.
3. Pour valider un fichier XML suivant un fichier XSD, positionner les deux instructions suivantes NWX_PARSER_SETDONAMESPACES et NWX_PARSER_SETMODE à TRUE%.
4. Si le document XML n'est pas valide et que le parseur est en mode validation, les erreurs seront enregistrées dans le fichier d'erreurs spécifié par la fonction NWX_PARSER_SETMSGFILE ou NWX_PARSER_SETOPTION.
5. Si le parseur est en mode validation et qu'aucune DTD ou XSD n'est spécifié, une erreur sera générée pour chacun des nœuds du document XML. Seulement, si la propriété NWX_PARSER_SETEXITONFIRSTERROR a été mise à TRUE%, alors seule la première erreur sera enregistrée.
6. Si le parseur est en mode validation et que la DTD ou le fichier XSD spécifié n'existe pas, une erreur sera générée pour cette ligne et le reste du document sera ignoré.

Exemple :

```
local pointer pParser
local ok%

ok%=NWX_INITIALIZE%
pParser = NWX_PARSER_NEW
NWX_PARSER_SETMODE pParser, true%
```

Voir aussi ***NWX_PARSER_NEW*****,** ***NWX_PARSER_DISPOSE*****,** ***NWX_PARSER_SETEXITONFIRSTERROR*****,** ***NWX_PARSER_SETOPTION*****,** ***NWX_PARSER_SETMSGFILE*****,** ***NWX_PARSER_SETDONAMESPACES***

Instruction NWX_PARSER_SETMSGFILE (Librairie NWXML)

L'instruction NWX_PARSER_SETMSGFILE spécifie le fichier utilisé par le parseur pour enregistrer les avertissements et erreurs.

Syntaxe	**NWX_PARSER_SETMSGFILE** *pParser, MsgFile$*			
Paramètres	pParser	POINTER	I/O	parseur XML
	MsgFile$	CSTRING	I	fichier de message

Par défaut, le parseur ne crée pas de fichier de message.

Exemple :

```
local pointer pParser
local ok%

ok%=NWX_INITIALIZE%
pParser = NWX_PARSER_NEW
NWX_PARSER_SETMSGFILE pParser, "c:\natstar\XML\msg.txt"
```

Voir aussi ***NWX_PARSER_NEW*****,** ***NWX_PARSER_DISPOSE***

Instruction NWX_PARSER_SETOPTION (Librairie NWXML)

L'instruction NWX_PARSER_SETOPTION spécifie un répertoire (D) et/ou un fichier (f) pour l'écriture des erreurs rencontrées par le parseur.

Nat System recommande plutôt l'utilisation de l'instruction NWX_PARSER_SETMSGFILE qui a les mêmes fonctionnalités.

Syntaxe	**NWX_PARSER_SETOPTION** *pParser, Opt$, Value$*			
Paramètres	pParser	POINTER	I/O	parseur XML
	Opt$	STRING	I/O	option
	Value$	CSTRING	I/O	valeur de l'option

1. Utilisez les constantes suivantes :
   a) NWX_DIRECTORY$: pour spécifier le répertoire du fichier d'erreur
   b) NWX_FILE$: pour spécifier le nom du fichier d'erreur
2. Quand vous utilisez un chemin d'accès relatif pour les fichiers d'erreurs, vérifier bien la présence de la variable d'environnement NWXML_DIR.

Exemple :

```
local pointer pParser
local ok%

ok%=NWX_INITIALIZE%
pParser = NWX_PARSER_NEW

NWX_PARSER_SETOPTION pParser, NWX_XML_DIRECTORY$, \
"c:\natstar\xml\pubtext"
NWX_PARSER_SETOPTION pParser, NWX_XML_FILE$, "errorfile"
```

Voir aussi ***NWX_PARSER_SETMSGFILE, NWX_PARSER_NEW, NWX_DIRECTORY$, NWX_FILE$***

Instruction NWX_PARSER_GETEXTERNALNONAMESPACESCHEMALOCATION (Librairie NWXML)

La fonction NWX_PARSER_GETEXTERNALNONAMESPACESCHEMALOCATION renvoie le schéma XML externe sans namespace.

Syntaxe	**NWX_PARSER_GETEXTERNALNONAMESPACESCHEMALOCATION** *pParser*			
Paramètre	pParser	POINTER	I/O	pointe du parse
Valeur retournée	DynStr Nom du fichier schéma XML s'il a été défini.			

Pour plus d'informations sur les espaces de noms, Nat System vous recommande de consulter le site suivant : http://xmlfr.org/documentations/faq/041217-0001.
Exemple :

```
Local int failed%, ok%
Local pointer pparser
Local pointer pdoc
Local dynstr ds
failed% = NWX_INITIALIZE%
pParser = NWX_PARSER_NEW
NWX_PARSER_SETMODE pparser, true%
NWX_PARSER_SETDONAMESPACES pparser, True%
NWX_PARSER_SETERRORCALLBACK pParser, @PARSER_ERROR_CALLBACK
IF CK_EXT
NWX_PARSER_SETEXTERNALNONAMESPACESCHEMALOCATION pParser, "mails.xsd"
ENDIF
message "External Schema Location",
NWX_PARSER_GETEXTERNALNONAMESPACESCHEMALOCATION(pParser)
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
ok% = NWX_DOCUMENT_LOADFILE%(pDoc, "mails.xml", pParser)
ds = NWX_DOCUMENT_OUT_DYNSTR(pdoc)
MLE1.text = ds
NWX_PARSER_DISPOSE pparser
NWX_DOCUMENT_DISPOSE pdoc
NWX_TERMINATE
```

Voir aussi ***NWX_PARSER_SETEXTERNALNONAMESPACESCHEMALOCATION, NWX_PARSER_GETEXTERNALSCHEMALOCATION, NWX_PARSER_SETEXTERNALSCHEMALOCATION***

Instruction NWX_PARSER_SETEXTERNALNONAMESPACESCHEMALOCATION (Librairie NWXML)

L'instruction NWX_ PARSER_SETEXTERNALNONAMESPACESCHEMALOCATION permet de spécifier un schéma XML externe sans namespace.

Syntaxe	**NWX_PARSER_SETEXTERNALNONAMESPACESCHEMALOCATION** *pParser, pathName*			
Paramètres	pParser	POINTER	I/O	pointeur du parseur
	pathName	CSTRING	I	nom du fichier schéma XML à utiliser

Pour plus d'informations sur les espaces de noms, Nat System vous recommande de consulter le site suivant : http://xmlfr.org/documentations/faq/041217-0001.

Exemple :

```
Local int failed%, ok%
Local pointer pparser
Local pointer pdoc
Local dynstr ds
failed% = NWX_INITIALIZE%
pParser = NWX_PARSER_NEW
NWX_PARSER_SETMODE pparser, true%
NWX_PARSER_SETDONAMESPACES pparser, True%
NWX_PARSER_SETERRORCALLBACK pParser, @PARSER_ERROR_CALLBACK
IF CK_EXT
NWX_PARSER_SETEXTERNALNONAMESPACESCHEMALOCATION pParser, "mails.xsd"
ENDIF
message "External Schema Location",
NWX_PARSER_GETEXTERNALNONAMESPACESCHEMALOCATION(pParser)
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
ok% = NWX_DOCUMENT_LOADFILE%(pDoc, "mails.xml", pParser)
ds = NWX_DOCUMENT_OUT_DYNSTR(pdoc)
MLE1.text = ds
NWX_PARSER_DISPOSE pparser
NWX_DOCUMENT_DISPOSE pdoc
NWX_TERMINATE
```

Voir aussi ***NWX_PARSER_GETEXTERNALNONAMESPACESCHEMALOCATION, NWX_PARSER_GETEXTERNALSCHEMALOCATION, NWX_PARSER_SETEXTERNALSCHEMALOCATION***

Instruction NWX_PARSER_GETEXTERNALSCHEMALOCATION (Librairie NWXML)

La fonction NWX_PARSER_ PARSER_GETEXTERNALSCHEMALOCATION permet de spécifier un schéma XML externe avec namespace.

Syntaxe	**NWX_PARSER_GETEXTERNALSCHEMALOCATION** *pParser*			
Paramètre	pParser	POINTER	I/O	pointeur du parseur
Valeur retournée	DynStr Nom du fichier schéma XML s'il a été défini.			

Exemple :

```
Local int failed%, ok%
Local pointer pparser
Local pointer pdoc
Local dynstr ds
failed% = NWX_INITIALIZE%
pParser = NWX_PARSER_NEW
NWX_PARSER_SETMODE pparser, true%
NWX_PARSER_SETDONAMESPACES pparser, True%
NWX_PARSER_SETERRORCALLBACK pParser, @PARSER_ERROR_CALLBACK
IF CK_EXT
NWX_PARSER_SETEXTERNALSCHEMALOCATION pParser,
"http://www.example.com/Report report.xsd"
ENDIF
message "External Schema Location",
NWX_PARSER_GETEXTERNALSCHEMALOCATION(pParser)
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
ok% = NWX_DOCUMENT_LOADFILE%(pDoc, "externalLocalisation.xml", pParser)
ds = NWX_DOCUMENT_OUT_DYNSTR(pdoc)
MLE1.text = ds
NWX_PARSER_DISPOSE pparser
NWX_DOCUMENT_DISPOSE pdoc
NWX_TERMINATE
```

Voir aussi ***NWX_PARSER_SETEXTERNALNONAMESPACESCHEMALOCATION***, ***NWX_PARSER_GETEXTERNALNONAMESPACESCHEMALOCATION***, ***NWX_PARSER_SETEXTERNALSCHEMALOCATION***

Instruction NWX_PARSER_SETEXTERNALSCHEMALOCATION (Librairie NWXML)

L'instruction NWX_PARSER_SETEXTERNALSCHEMALOCATION permet de spécifier un schéma XML externe avec namespace.

Syntaxe	**NWX_PARSER_SETEXTERNALNONAMESPACESCHEMALOCATION** *pParser*, *pathName*

Paramètres	pParser	POINTER	I/O	pointeur du parseur
	pathName	CSTRING	I	nom du fichier schéma XML à utiliser

Pour plus d'informations sur les espaces de noms, Nat System vous recommande de consulter le site suivant : http://xmlfr.org/documentations/faq/041217-0001

Exemple :

```
Local int failed%, ok%
Local pointer pparser
Local pointer pdoc
Local dynstr ds
failed% = NWX_INITIALIZE%
pParser = NWX_PARSER_NEW
NWX_PARSER_SETMODE pparser, true%
NWX_PARSER_SETDONAMESPACES pparser, True%
NWX_PARSER_SETERRORCALLBACK pParser, @PARSER_ERROR_CALLBACK
IF CK_EXT
NWX_PARSER_SETEXTERNALSCHEMALOCATION pParser,
"http://www.example.com/Report report.xsd"
ENDIF
message "External Schema Location",
NWX_PARSER_GETEXTERNALSCHEMALOCATION(pParser)
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
ok% = NWX_DOCUMENT_LOADFILE%(pDoc, "externalLocalisation.xml", pParser)
ds = NWX_DOCUMENT_OUT_DYNSTR(pdoc)
MLE1.text = ds
NWX_PARSER_DISPOSE pparser
NWX_DOCUMENT_DISPOSE pdoc
NWX_TERMINATE
```

Voir aussi ***NWX_PARSER_SETEXTERNALNONAMESPACESCHEMALOCATION*****,** ***NWX_PARSER_GETEXTERNALNONAMESPACESCHEMALOCATION*****,** ***NWX_PARSER_SETEXTERNALSCHEMALOCATION***

Instruction NWX_PARSER_SETERRORCALLBACK (Librairie NWXML)

L'instruction NWX_PARSER_SETERRORCALLBACK permet de spécifier une fonction appelée si le parseur rencontre des problèmes.

Syntaxe	**NWX_PARSER_SETERRORCALLBACK** *pParser, pCallback*			
Paramètres	pParser	POINTER	I/O	pointeur du parseur
	pCallback	POINTER	I/O	pointeur sur la

				fonction de callback

1. La fonction de callback doit respecter le prototype suivant : INSTRUCTION parser_error_callback level%, cstring DocumentId, LineNumber%, ColumnNumber%, cstring errMessage
2. DocumentId, LineNumber%, ColumnNumber% sont respectivement le document, la ligne et la colonne où se produisent le problème, errMessage est le message d'erreur associé.
3. level% est égale à l'une des constantes suivantes :
   a) NWX_PARSER_WARNING%
   b) NWX_PARSER_ERROR%
   c) NWX_PARSER_FATALERROR%

Exemple :

```
local pointer pParser
local ok%
ok%=NWX_INITIALIZE%
pParser = NWX_PARSER_NEW
NWX_PARSER_SETERRORCALLBACK pParser, @PARSER_ERROR_CALLBACK
...
INSTRUCTION parser_error_callback level%, cstring DocumentId, LineNumber%,
ColumnNumber%, cstring errMessage
Local msg$
msg$ = "Document :" && DocumentId & #10 & "Line :"&& LineNumber% && " -
Column :"&& ColumnNumber% & #10#10 & errMessage
Evaluate level%
Where NWX_PARSER_WARNING%
message "Warning", msg$
EndWhere
Where NWX_PARSER_Error%
message "Error", msg$
EndWhere
Where NWX_PARSER_FATALERROR%
message "Fatal Error", msg$
EndWhere
EndEvaluate
ENDINSTRUCTION
```

Import/export

Fonction NWX_DOCUMENT_LOAD% (Librairie NWXML)

Analyse le buffer et le charge dans le document visé par le pointeur pDoc.

Syntaxe	**NWX_DOCUMENT_LOAD%** *(pDoc, PARSEDBUF$, pParser)*			
Paramètres	pDoc	POINTER	I/O	pointeur d'élément

	PARSEDBUF$	CSTRING	I/O	buffer XML
	pParser$	POINTER	I	pointeur du parseur XML
Valeur renvoyée	INT(1) TRUE% ou FALSE%			

1. La fonction NWX_DOCUMENT_GETFROMBUFFER, plus facile d'utilisation, remplace NWX_DOCUMENT_LOAD%.
2. Si la valeur renvoyée est 0, une erreur est générée
3. Si la valeur renvoyée est 1, aucune erreur n'est générée
4. Utilisez cette instruction pour charger un buffer de format XML.
5. Si un fichier XML fait référence à une DTD ou un XSD externe et que le parseur ne la trouve pas, le fichier est tronqué (en mode validation ou pas).
6. Utilisez NWX_DOCUMENT_LOADFILE pour charger un fichier de format XML.

Exemple :

```
local pointer pDoc
local pointer pParser, pointer pRootNode
local value$, buffer$
local ok%
local ok2%

ok2%=NWX_INITIALIZE%

;Nouveau parseur
pParser = NWX_PARSER_NEW
;Nouveau document
pDoc = NWX_DOCUMENT_NEW("XMLdoc")

buffer$ ="<AUTO><MODELE>Toyota</MODELE><ANNEE>1953</ANNEE></AUTO>"

;Charger le buffer dans pDoc et pParser
ok% = NWX_DOCUMENT_LOAD%(pDoc, buffer$, pParser)

;Chercher le nœud racine du document
pRootNode = NWX_DOCUMENT_GETROOT (pDoc)
  ;Afficher ce nœud
ok% =NWX_NODE_OUTPUT% (pRootNode,@value$, sizeof value$)

MESSAGE "NOEUD RACINE", value$
```

Voir aussi NWX_DOCUMENT_GETFROMBUFFER, NWX_DOCUMENT_LOADFILE%, NWX_DOCUMENT_OUTPUTFILE%, NWX_DOCUMENT_OUTPUT , NWX_DOCUMENT_SETPRETTYPRINT

Fonction NWX_DOCUMENT_LOADFILE% (Librairie NWXML)

Analyse le fichier XML et le charge dans le document visé par le pointeur pDoc.

Syntaxe	**NWX_DOCUMENT_LOADFILE%** *(pDoc, name$, pParser$)*			
Paramètres	pDoc	POINTER	I/O	pointeur d'élément
	name$	CSTRING	I	nom de fichier
	pParser$	POINTER	I	pointeur du parseur XML
Valeur renvoyée	INT(1) TRUE% ou FALSE%. La valeur renvoyée est FALSE et le fichier XML n'est pas chargé si un des trois paramètres en entrée est null.			

1. La fonction NWX_DOCUMENT_GETFROMFILE, plus facile d'utilisation, remplace NWX_DOCUMENT_LOADFILE%.
2. Si la valeur renvoyée est FALSE%, une erreur est générée.
3. Si la valeur renvoyée est TRUE%, aucune erreur n'est générée.
4. Utilisez cette instruction pour charger le formatage XML d'un fichier.
5. Utilisez NWX_DOCUMENT_LOAD pour charger le formatage XML d'un buffer.
6. Si le fichier XML fait référence à une DTD ou un XSD cette dernière est également chargée.
7. Si un fichier XML fait référence à une DTD ou un XSD externe et que le parseur ne la trouve pas le fichier est tronqué (en mode validation ou pas).

Exemple :

```
local pointer pParser
local pointer pDoc
local ok%
local pointer pRootElem
local value$(1024)
local pointer pChild
local ok2%

ok2%=NWX_INITIALIZE%
; Nouveau parseur vide
pParser = NWX_PARSER_NEW

; Passe le parseur en mode validation
NWX_PARSER_SETMODE pParser, 1
; Afin que le parseur ne prennent pas en compte les espaces
; blancs
NWX_PARSER_SETINCIGNWHITESPACE pParser, 0

;Nouveau document vide
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
  ok% = NWX_DOCUMENT_LOADFILE%(pDoc, "C:\Xml\mydoc.xml", pParser)
```

```
if ok%
;Aller chercher le nœud racine
pRootElem = NWX_DOCUMENT_GETROOT(pDoc)
;Afficher le nœud
ok% = NWX_NODE_OUTPUT% (pRootElem,@value$,sizeof value$)
  MESSAGE "NOEUD RACINE",value$
else
MESSAGE "ERREUR", "Incapable de charger le document"
endif
```

Voir aussi ***NWX_DOCUMENT_GETFROMFILE***, ***NWX_DOCUMENT_LOAD%***, ***NWX_DOCUMENT_OUTPUTFILE%***, ***NWX_DOCUMENT_OUTPUT***, ***NWX_DOCUMENT_GETROOT***

Fonction NWX_DOCUMENT_OUTPUT% (Librairie NWXML)

Exporte le document XML dans un buffer.

Syntaxe	**NWX_DOCUMENT_OUTPUT%** *(pDoc, XMLBuff, Size%)*			
Paramètres	pDoc	POINTER	I/O	pointeur d'élément
	XMLBuff	POINTER	I	pointeur du buffer
	Size%	INT(4)	I	taille du buffer
Valeur renvoyée	INT(4) correspondant à la longueur de la chaîne passée au buffer.			

1. Pour la taille, vous devez ajouter 1 pour la terminaison "\0"
2. Si la taille de la sortie est plus longue que la valeur spécifiée, la chaîne est tronquée
3. Si XMLbuf ou Size% est égal à 0, la valeur renvoyée est la taille requise pour contenir toute la chaîne en sortie.
4. La valeur renvoyée est toujours la taille de la chaîne en sortie.

Exemple :

```
local pointer pParser
local pointer pDoc
local ok%(1)
local buffer$(1024)
local ok2%

ok2%=NWX_INITIALIZE%
pParser = NWX_PARSER_NEW

pDoc = NWX_DOCUMENT_NEW("XMLdoc")

ok% = NWX_DOCUMENT_LOADFILE%(pDoc, "c:\XML\mydoc.XML", \ pParser)
 if ok% = false%
  ; erreur
 else
```

```
  if NWX_DOCUMENT_OUTPUT% (pDoc,@buffer$, sizeof \ buffer$) > sizeof
buffer$
   ;buffer est trop court
  else
  ;on peut afficher le buffer
MESSAGE "DOCUMENT" , buffer
  endif
 endif
```

Voir aussi ***NWX_DOCUMENT_OUTPUTFILE%, NWX_DOCUMENT_OUTPUT, NWX_NODE_OUTPUTFILE%, NWX_NODE_OUTPUT%, NWX_DOCUMENT_LOADFILE, NWX_DOCUMENT_LOAD%***

Fonction NWX_DOCUMENT_OUTPUTFILE% (Librairie NWXML)

La fonction NWX_DOCUMENT_OUTPUTFILE% permet d'exporter au format XML le contenu du document dans un fichier.

Syntaxe	**NWX_DOCUMENT_OUTPUTFILE%** *(pDoc, FileName$)*			
Paramètres	pDoc	POINTER	I/O	pointeur d'élément
	FileName$	CSTRING	I	nom de fichier
Valeur renvoyée	INT(4) correspondant à la longueur de la chaîne contenue dans le fichier.			

Si le fichier contient déjà des informations, elles seront remplacées par le contenu du document

Exemple :

```
local pointer pParser
local pointer pDoc
local ok%
local XMLbuffer$
local ok2%

ok2%=NWX_INITIALIZE%

move "<book><title>Book's title</title><section>Book's
section</section></book>" to XMLbuffer$

pParser = NWX_PARSER_NEW

pDoc = NWX_DOCUMENT_NEW("XMLdoc")

ok% = NWX_DOCUMENT_LOAD%(pDoc, XMLbuffer$, pParser)

if ok% = false%
; erreur
else
ok% = NWX_DOCUMENT_OUTPUTFILE% (pDoc,"C:\Xml\mydoc.xml")
endif
```

Voir aussi ***NWX__DOCUMENT_OUTPUT, NWX_NODE_OUTPUTFILE%, NWX_NODE_OUTPUT%, NWX_DOCUMENT_LOADFILE, NWX_DOCUMENT_LOAD%***

Fonction NWX_NODE_OUTPUT% (Librairie NWXML)

Exporte le contenu d'un nœud vers un buffer avec tous les nœuds enfants qu'il contient.

Syntaxe	**NWX_NODE_OUTPUT%** *(pNode, pXMLBuff, Size%)*			
Paramètres	pNode	POINTER	I/O	pointeur du nœud
	pXMLBuff	POINTER	I/O	pointeur du buffer
	Size%	INT(4)	I/O	taille du buffer de sortie
Valeur renvoyée	INT(4) correspondant à la longueur de la chaîne passée dans le buffer.			

1. Pour la taille, vous devez ajouter 1 pour la terminaison "\0"
2. Si la taille de la sortie est plus longue que la valeur spécifiée, la chaîne est tronquée
3. Si pXMLbuf ou Size% est égal à 0, la valeur renvoyée est la taille requise pour contenir toute la chaîne en sortie.
4. La valeur renvoyée est toujours la taille de la chaîne en sortie.

Exemple

```
local pointer pDoc
local pointer pParser, pointer pRootNode
local value$, buffer$
local ok%
local ok2%

ok2%=NWX_INITIALIZE%

;Nouveau parseur
pParser = NWX_PARSER_NEW
;Nouveau document
pDoc = NWX_DOCUMENT_NEW("XMLdoc")

buffer$ ="<?xml version="1.0" encoding="ISO-8859-1" \
standalone="yes"?><AUTO><MODELE>Toyota</MODELE><ANNEE>1953</ANNEE></AUTO>"

;Charger le buffer dans pDoc et pParser
ok% = NWX_DOCUMENT_LOAD%(pDoc, buffer$, pParser)

;Chercher le noeud racine du document
pRootNode = NWX_DOCUMENT_GETROOT (pDoc)
```

```
;Afficher ce noeud
ok% = NWX_NODE_OUTPUT% (pRootNode,@value$, sizeof value$)

MLE1 = value$
```

Résultat :

```
<AUTO><MODELE>Toyota</MODELE><ANNEE>1953</ANNEE></AUTO>
```

Voir aussi ***NWX_DOCUMENT_OUTPUT*****,** ***NWX_DOCUMENT_OUTPUTFILE%*****,** ***NWX_NODE_OUTPUTFILE%*****,** ***NWX_DOCUMENT_LOADFILE*****,** ***NWX_DOCUMENT_LOAD%***

Fonction NWX_NODE_OUTPUTFILE% (Librairie NWXML)

La fonction NWX_NODE_OUTPUTFILE% permet d'exporter en format XML le contenu du nœud dans un fichier.

Syntaxe	**NWX_NODE_OUTPUTFILE%** *(pnode, FileName$)*			
Paramètres	pnode	POINTER	I/O	pointeur du nœud
	FileName$	CSTRING	I	nom de fichier
Valeur renvoyée	INT(4) correspondant à la longueur de la chaîne écrite dans le fichier.			

Si le fichier contient déjà de l'information, celle-ci sera remplacée par le contenu du document.
Exemple :

```
local pointer pParser
local pointer pDoc
local ok%(1)
local XMLbuffer$
local pointer pRoot
local pointer pChild
local ok2%

ok2%=NWX_INITIALIZE%

move "<book><title>Book's title</title><section>Book's
content</section></book>" to XMLbuffer$

pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")

ok% = NWX_DOCUMENT_LOAD%(pDoc, XMLbuffer$, pParser)

if ok% = false%
; erreur
else
```

```
pRoot =NWX_DOCUMENT_GETROOT(pDoc)
pChild =NWX_NODE_GETCHILD (pRoot)
; Écrira <title>Book's title</title> dans le fichier
; mydoc.xml
ok% = NWX_NODE_OUTPUTFILE% (pChild, "c:\XML\mydoc.xml")
endif
```

Voir aussi ***NWX_DOCUMENT_OUTPUT***, ***NWX_DOCUMENT_OUTPUTFILE%***, ***NWX_NODE_OUTPUTFILE%***, ***NWX_NODE_OUTPUT%***, ***NWX_DOCUMENT_LOADFILE***, ***NWX_DOCUMENT_LOAD%***

Fonction NWX_DOCUMENT_GETFROMBUFFER (Librairie NWXML)

Analyse le buffer et le charge dans le document référencé par le pointeur renvoyé en retour.

Syntaxe	**NWX_DOCUMENT_GETFROMBUFFER** *(PARSEDBUF$, pParser)*			
Paramètres	PARSEDBUF$	CSTRING	I/O	buffer XML
	pParser$	POINTER	I	pointeur du parseur XML
Valeur renvoyée	POINTER La valeur renvoyée est nulle si le chargement échoue, sinon c'est le pointeur sur le document chargé.			

Cette fonction remplace NWX_DOCUMENT_LOAD. Aucun appel n'est nécessaire à l'instruction NWX_DOCUMENT_NEW.

Exemple :

```
local pointer pDoc
local pointer pParser, pointer pRootNode
local DynStr value$, buffer$
local retCode%
retCode% = NWX_INITIALIZE%
if retCode% <> 0
exit
endif
;Nouveau parseur
pParser = NWX_PARSER_NEW
buffer$ ="<AUTO><MODELE>Toyota</MODELE><ANNEE>1953</ANNEE></AUTO>"
;Charger le buffer dans pDoc et pParser
pDoc = NWX_DOCUMENT_GETFROMBUFFER(buffer$, pParser)
;Chercher le noeud racine du document
pRootNode = NWX_DOCUMENT_GETROOT (pDoc)
;Afficher ce noeud
value$ = NWX_NODE_OUT_DYNSTR (pRootNode)
MESSAGE "NOEUD RACINE", value$
; liberer le document
NWX_DOCUMENT_DISPOSE pDoc
; liberer le parseur
```

```
NWX_PARSER_DISPOSE pParser
NWX_TERMINATE
```

***Voir aussi** NWX_DOCUMENT_LOADFILE%, NWX_DOCUMENT_LOAD.*

Fonction NWX_DOCUMENT_GETFROMFILE (Librairie NWXML)

Analyse le fichier XML et le charge dans le document référencé par le pointeur renvoyé en retour.

Syntaxe	**NWX_DOCUMENT_GETFROMFILE** *(name$, pParser$)*			
Paramètres	name$	CSTRING	I	nom de fichier
	pParser$	POINTER	I	pointeur du parseur XML
Valeur renvoyée	POINTER La valeur renvoyée est nulle si le chargement échoue, sinon renvoie le pointeur sur le document chargé.			

Cette fonction remplace NWX_DOCUMENT_LOADFILE%. Aucun appel n'est plus nécessaire à l'instruction NWX_DOCUMENT_NEW.

Exemple :

```
local pointer pParser
local pointer pDoc
local pointer pRootElem
local DynStr value$
local errCode%
errCode% = NWX_INITIALIZE%
if errcode% <> 0
exit
endif
; Nouveau parseur vide
pParser = NWX_PARSER_NEW
; Chargement du document
pDoc = NWX_DOCUMENT_GETFROMFILE ("C:\Xml\mydoc.xml", pParser)
if pDoc <> 0
;Aller chercher le noeud racine
pRootElem = NWX_DOCUMENT_GETROOT(pDoc)
;Afficher le noeud
value$ = NWX_NODE_OUT_DYNSTR(pRootElem)
MESSAGE "NOEUD RACINE",value$
else
MESSAGE "ERREUR", "Incapable de charger le document"
Endif
; liberer le document
NWX_DOCUMENT_DISPOSE pDoc
; liberer le parseur
NWX_PARSER_DISPOSE pParser
NWX_TERMINATE
```

Voir aussi ***NWX_DOCUMENT_LOADFILE%*****,** ***NWX_DOCUMENT_GETFROMBUFFER***

Gestion des documents XML

Instruction NWX_DOCUMENT_DISPOSE (Librairie NWXML)

Libère un document XML.

Syntaxe	**NWX_DOCUMENT_DISPOSE** *pDoc*			
Paramètre	pDoc	POINTER	I/O	pointeur du document

Utilisez cette instruction pour libérer les documents créés avec NWX_DOCUMENT_NEW .

Exemple :

```
local pointer pParser
local pointer pDoc
local ok%(1)
local ok2%

ok2%=NWX_INITIALIZE%

pParser = NWX_PARSER_NEW

pDoc = NWX_DOCUMENT_NEW("XMLdoc")

ok% = NWX_DOCUMENT_LOADFILE%(pDoc, "c:\XML\mydoc.XML", pParser)
if ok% = false%
  ; erreur
 else
  ; faire qqch avec pDoc
  NWX_DOCUMENT_DISPOSE pDoc
endif
```

Voir aussi ***NWX_DOCUMENT_NEW***

Fonction NWX_DOCUMENT_GETELEMSBYTAGNAME (Librairie NWXML)

Retourne une liste d'éléments d'un document avec un nom de balise spécifié.

Syntaxe	**NWX_DOCUMENT_GETELEMSBYTAGNAME** *(pDoc, tagname$)*			
Paramètres	pDoc	POINTER	I/O	pointeur du document
	tagname$	CSTRING	O	nom de la balise
Valeur renvoyée	POINTER sur une liste de nœuds.			

1. Utilisez NWX_NODELIST_GETFIRST , NWX_NODELIST_GETNEXT pour interroger la liste.
2. Pour libérer la liste, utilisez NWX_NODELIST_DISPOSE .
3. Pour plus d'informations sur les nœuds Document, reportez-vous au glossaire situé à la fin du document.

Exemple :

```
local pointer pParser
local pointer pDoc
local ok%(1)
local value$(1024)
local pointer pNodeList
local pointer pNode
local ok2%

ok2%=NWX_INITIALIZE%

;Nouveau parseur
pParser = NWX_PARSER_NEW

;Nouveau document
pDoc = NWX_DOCUMENT_NEW("XMLdoc")

;Charger le document XML
ok% = NWX_DOCUMENT_LOADFILE%(pDoc, "musique.xml", pParser)

pNodeList = NWX_DOCUMENT_GETELEMSBYTAGNAME(pDoc, "titre")
if pNodeList
  pNode = NWX_NODELIST_GETFIRST(pNodeList)
 ok% =  NWX_NODE_OUTPUT%(pNode, @value$, Sizeof value$)
else
   value$ = "pas de liste!!!"
endif
INSERT AT END  value$(0..length value$) TO MLE1
while pNode
   ok% =  NWX_NODE_GETVALUE%(pNode, @value$, Sizeof value$)
   INSERT AT END  value$(0..length value$) TO MLE1
   pNode = NWX_NODELIST_GETNEXT(pNodeList,pNode)
endwhile
; dispose la liste
NWX_NODELIST_DISPOSE
; dispose le document
NWX_DOCUMENT_DISPOSE pDoc
; Libère le parseur
NWX_PARSER_DISPOSE pParser

Fichier source musique.xml :
<basedd_mp3>
 <enregistrement format="CD" type="rock">
  <groupe>Aha</groupe>
  <titre>Take on me</titre>
 </enregistrement>
 <enregistrement format="Vinyl" type="pop">
  <groupe>The Beach Boys</groupe>
  <titre>Surfin' USA</titre>
  <titre>California Girls</titre>
```

```
  <titre>Barbara-Ann</titre>
 </enregistrement>
 <enregistrement format="CD" type="classique">
  <groupe>Orchestre de Berlin</groupe>
  <titre>Les 4 saisons de Vivaldi</titre>
 </enregistrement>
</basedd_mp3>
```

Résultat :

```
<titre>Take on me</titre>
Take on me
Surfin' USA
California Girls
Barbara-Ann
Les 4 saisons de Vivaldi
```

Voir aussi ***NWX_ELEMENT_GETELEMSBYTAGNAME*****,** ***NWX_NODELIST_GETFIRST*****,** ***NWX_NODELIST_GETNEXT*****,** ***NWX_NODELIST_DISPOSE***

Fonction NWX_DOCUMENT_GETROOT (Librairie NWXML)

Retourne le nœud racine du document XML. Le nœud racine du document XML correspond à la première ligne du corps du document qui ne coïncide pas toujours avec la première ligne du document. En effet, un document XML est constitué de deux parties : un prologue (optionnel) et le corps du document (qui contient le contenu balisé).

Pour plus d'informations sur la structure d'un document XML, reportez-vous à la section Structure d'un document XML.

Syntaxe	**NWX_DOCUMENT_GETROOT** *(pDoc)*			
Paramètre	pDoc	POINTER	I/O	pointeur du document
Valeur renvoyée	POINTER sur un document.			

Exemple :

```
local pointer pParser
local pointer pDoc
local ok%(1)
local value$(2048)
local node%
local pointer pNode
local CONTROL ctrl
local ok2%

ok2%=NWX_INITIALIZE%
  ;Nouveau parseur
pParser = NWX_PARSER_NEW

;Mode validation
```

```
NWX_PARSER_SETMODE pParser, TRUE%

;Sans espaces
NWX_PARSER_SETINCIGNWHITESPACE pParser, FALSE%
;Faire en sorte que le parseur sorte à la rencontre d'une
;erreur fatale
NWX_PARSER_SETEXITONFIRSTERROR pParser, TRUE%

;Fichier d'erreurs
NWX_PARSER_SETMSGFILE pParser, EF_DOC2

;Nouveau document
pDoc = NWX_DOCUMENT_NEW("XMLdoc")

;Charger le document XML
ok% = NWX_DOCUMENT_LOADFILE%(pDoc, EF_DOC, pParser)

pNode =  NWX_DOCUMENT_GETROOT(pDoc)
ok% =  NWX_NODE_OUTPUT%(pNode, @value$, Sizeof value$)
  ;Afficher le texte dans la MLE
@ctrl = wXML(self%).MLE1
ok% =  ToMLE% (value$, ctrl)
; Dispose le document
NWX_DOCUMENT_DISPOSE pDoc
; Libère le parseur
NWX_PARSER_DISPOSE pParser
Fichier xml source :
Le nœud racine du document XML suivant correspond au nœud <REPERTOIRE>
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<REPERTOIRE>
 <CLIENT>
  <NOMCLIENT>Cyberé Globe</NOMCLIENT>
  <ADRESSE>
   <RUE>rue Elle</RUE>
   <CODEPOSTAL>95 000</CODEPOSTAL>
   <VILLE>Anjou</VILLE>
  </ADRESSE>
  <TELEPHONE>01 55 66 77 99</TELEPHONE>
 </CLIENT>
</REPERTOIRE>
Résultat :
<REPERTOIRE>
 <CLIENT>
  <NOMCLIENT>Cyberé Globe</NOMCLIENT>
  <ADRESSE>
   <RUE>rue Elle</RUE>
   <CODEPOSTAL>95 000</CODEPOSTAL>
   <VILLE>Anjou</VILLE>
  </ADRESSE>
  <TELEPHONE>01 55 66 77 99</TELEPHONE>
 </CLIENT>
</REPERTOIRE>
```

Voir aussi NWX_NODE_DETACH%, Structure d'un document XML

Fonction NWX_DOCUMENT_NEW (Librairie NWXML)

Crée un pointeur sur un document XML.

Syntaxe	**NWX_DOCUMENT_NEW** *(docname$)*			
Paramètre	docname$	CSTRING	O	nom du document
Valeur renvoyée	POINTER sur un document.			

1. A la création, les documents n'ont pas de nœuds.
2. Pour remplir le document, utilisez par exemple :
   a) NWX_DOCUMENT_LOAD%
   b) NWX_DOCUMENT_LOADFILE%
   c) NWX_ELEMENT_NEW
3. Quand vous avez terminé, libérez le document avec : NWX_DOCUMENT_DISPOSE

.

Exemple :

```
local pointer pDoc
local ok%(1)
local ok2%

ok2%=NWX_INITIALIZE%

pDoc = NWX_DOCUMENT_NEW("ofxdoc")

ok% = NWX_DOCUMENT_LOADFILE%(pDoc, "C:\Xml\mydoc.xml",\ pParser)
```

Voir aussi NWX_DOCUMENT_DISPOSE, NWX_DOCUMENT_LOAD%, NWX_DOCUMENT_LOADFILE%, NWX_ELEMENT_NEW

Fonction NWX_DOCUMENT_OUT_DYNSTR (Librairie NWXML)

Exporte le contenu d'un document XML vers une chaîne dynamique.

Syntaxe	**NWX_DOCUMENT_OUT_DYNSTR** *(pDoc)*			
Paramètre	pDoc	POINTER	I/O	pointeur du document
Valeur renvoyée	DYNSTR chaîne représentant le document.			

Voir aussi NWX_NODE_OUT_DYNSTR

Instruction NWX_DOCUMENT_SETPRETTYPRINT (Librairie NWXML)

Imprime un document XML en générant des retours chariot pour que le document soit plus lisible.

Syntaxe	**NWX_DOCUMENT_SETPRETTYPRINT** *pDoc, val*			
Paramètres	pDoc	POINTER	I/O	pointeur vers le

				document XML
	val	INT(1)	I	génération (true) ou pas (false) de retour chariot

Exemple :

```
local Dynstr ds
Local pointer pParser
Local pointer pXML
Local pointer pXSL
Local pointer pResultDoc
Local ok%
Local ok2%
ok2%=NWX_INITIALIZE%
pParser = NWX_PARSER_NEW
ds =NWX_XSLTRANSFORMFILESTODOM ("C:\Widget.scrxml",
"c:\NatStar2NatJet.xsl", pResultDoc)
if ds <> ""
  message "erreur", ds
endif
; exporter le résultat dans un fichier ou un buffer en
NWX_DOCUMENT_SETPRETTYPRINT (pResultDoc, true%)
ok% = NWX_DOCUMENT_OUTPUTFILE%(pResultDoc, "c:\Resultat.xml")
NWX_DOCUMENT_DISPOSE pResultDoc
NWX_PARSER_DISPOSE pParser
NWX_TERMINATE
```

Voir aussi ***NWX_DOCUMENT_OUTPUTFILE%***, ***NWX_DOCUMENT_OUTPUT***, ***NWX_NODE_OUTPUTFILE%***, ***NWX_NODE_OUTPUT%***, ***NWX_DOCUMENT_LOADFILE***, ***NWX_DOCUMENT_LOAD%*** , ***NWX_DOCUMENT_DISPOSE***, ***NWX_DOCUMENT_OUT_DYNSTR***

Gestion des noeuds XML

Fonction NWX_CDATASECTION_NEW (Librairie NWXML)

Crée un nœud XML de type CDATASection contenant la chaîne de caractères spécifiée.

Syntaxe	**NWX_CDATASECTION_NEW** *(pDoc, data$)*			
Paramètres	pDoc	POINTER	I/O	pointeur du document
	data$	CSTRING	I	chaîne de caractères de la

				section CDATA
Valeur renvoyée	POINTER vers le nouveau nœud créé.			

Cette fonction crée le nœud mais ne l'attache pas à l'arbre. Pour ce, utilisez les fonctions NWX_NODE_ATTACHBEFORE% et NWX_NODE_ATTACHAFTER%.

Exemple :

```
local pointer pDoc
local pointer pParser
local sValue$
local ok%
local pointer pNewCData
local pointer pRootNode
local pointer pNode
local ok2%

ok2%=NWX_INITIALIZE%
pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
move "<AUTO><MODELE>Toyota</MODELE><ANNEE>1951</ANNEE></AUTO>" to sValue$
pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)
pRootNode = NWX_DOCUMENT_GETROOT (pDoc)
pNode = NWX_NODE_GETCHILD (pRootNode)
pNewCData = NWX_CDATASECTION_NEW (pDoc, "Document <XML>")
ok%= NWX_NODE_ATTACHBEFORE% (pNode,pNewCData)
ok% = NWX_DOCUMENT_OUTPUT% (pDoc,@sValue$,sizeof sValue$)\
delete from MLE1
INSERT AT END sValue$ TO MLE1
; dispose le document
NWX_DOCUMENT_DISPOSE pDoc
; Libère le parseur
NWX_PARSER_DISPOSE pParser
```

Résultat :

```
<AUTO>
<![CDATA[Document <XML>]]>
<MODELE>Toyota</MODELE>
<ANNEE>1951</ANNEE>
</AUTO>
```

Voir aussi NWX_NODE_NEW, NWX_NODE_ATTACHBEFORE%, NWX_NODE_ATTACHAFTER%

Fonction NWX_COMMENT_NEW (Librairie NWXML)

Crée un nœud XML de type Comment contenant la chaîne de caractères spécifiée.

Syntaxe	**NWX_COMMENT_NEW** *(pDoc, comment$)*

Paramètres	pDoc	POINTER	I/O	pointeur du document
	comment$	CSTRING	I	le commentaire
Valeur renvoyée	POINTER vers le nouveau nœud créé.			

Cette fonction crée le nœud mais ne l'attache pas à l'arbre. Pour ce, utilisez les fonctions NWX_NODE_ATTACHBEFORE% et NWX_NODE_ATTACHAFTER%.

Exemple :

```
local pointer pDoc
local pointer pParser
local sValue$
local ok%
local pointer pNewElement
local pointer pRootNode
local pointer pNode
local ok2%

ok2%=NWX_INITIALIZE%

pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")

move "<AUTO><MODELE>Toyota</MODELE><ANNEE>1951</ANNEE></AUTO>" to sValue$

pParser = NWX_PARSER_NEW

pDoc = NWX_DOCUMENT_NEW("XMLdoc")

ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)
pRootNode = NWX_DOCUMENT_GETROOT (pDoc)
pNode = NWX_NODE_GETCHILD (pRootNode)
pNode = NWX_NODE_GETNEXT (pNode)

pNewElement = NWX_ELEMENT_NEW (pDoc, "TYPE")

ok%=NWX_NODE_SETVALUE% (pNewElement,"Berline")

ok%= NWX_NODE_ATTACHBEFORE% (pNode,pNewElement)

pNewElement = NWX_COMMENT_NEW (pDoc, "ceci est un commentaire")
ok%= NWX_NODE_ATTACHBEFORE% (pNode,pNewElement)

ok% = NWX_DOCUMENT_OUTPUTFILE%( pDoc, EF_DOC2)

ok% = NWX_DOCUMENT_OUTPUT% (pDoc,@sValue$,sizeof sValue$) delete from MLE1
INSERT AT END sValue$ TO MLE1
; dispose le document
NWX_DOCUMENT_DISPOSE pDoc
; Libère le parseur
NWX_PARSER_DISPOSE pParser
```

```
Résultat :
<AUTO>
<MODELE>Toyota</MODELE>
<TYPE>Berline</TYPE>
<!--ceci est un commentaire-->
<ANNEE>1951</ANNEE>
</AUTO>
```

***Voir aussi** NWX_NODE_NEW, NWX_NODE_ATTACHBEFORE%, NWX_NODE_ATTACHAFTER%*

Instruction NWX_DOCUMENTFRAGMENT_DISPOSE (Librairie NWXML)

Libère le nœud DocumentFragment spécifié.

Syntaxe	**NWX_DOCUMENTFRAGMENT_DISPOSE** *pDoc*			
Paramètre	pDoc	POINTER	I/O	pointeur du nœud DocumentFragment

Exemple :

```
local pointer pDoc
local pointer pParser
local sValue$
local ok%
local pointer pNewElement
local pointer pRootNode
local pointer pNode
local pointer pDocFrag
local ok2%

ok2%=NWX_INITIALIZE%
pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
move "<AUTO><MODELE>Toyota</MODELE><ANNEE>1951</ANNEE></AUTO>" \ to sValue$
pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)
pRootNode = NWX_DOCUMENT_GETROOT (pDoc)
pNode = NWX_NODE_GETCHILD (pRootNode)
pNode = NWX_NODE_GETNEXT (pNode)
pDocFrag=NWX_DOCUMENTFRAGMENT_NEW (pDoc)
pNewElement = NWX_ELEMENT_NEW (pDoc, "TYPE")
ok%=NWX_NODE_SETVALUE% (pNewElement,"Berline")
ok%= NWX_NODE_ATTACHCHILD% (pDocFrag,pNewElement)
pNewElement = NWX_COMMENT_NEW (pDoc, "ceci est un commentaire")
ok%= NWX_NODE_ATTACHCHILD% (pDocFrag,pNewElement)
ok%= NWX_NODE_ATTACHBEFORE% (pNode,pDocFrag)
ok% = NWX_DOCUMENT_OUTPUTFILE%( pDoc, EF_DOC2)
ok% = NWX_DOCUMENT_OUTPUT% (pDoc,@sValue$,sizeof sValue$) delete from MLE1
INSERT AT END  sValue$ TO MLE1
; dispose le nœud DocumentFragment
NWX_DOCUMENTFRAGMENT_DISPOSE pDocFrag
; dispose le document
NWX_DOCUMENT_DISPOSE pDoc
; Libère le parseur
NWX_PARSER_DISPOSE pParser
```

Résultat :

```
<AUTO>
<MODELE>Toyota</MODELE>
<TYPE>Berline</TYPE>
<!--ceci est un commentaire-->
<ANNEE>1951</ANNEE>
</AUTO>
```

Voir aussi NWX_ DOCUMENTFRAGMENT_NEW

Fonction NWX_DOCUMENTFRAGMENT_NEW (Librairie NWXML)

La fonction NWX_DOCUMENTFRAGMENT_NEW permet de créer un nœud DocumentFragment vide.

Syntaxe	**NWX_DOCUMENTFRAGMENT_NEW** *(pDoc)*			
Paramètre	pDoc	POINTER	I/O	pointeur du document
Valeur renvoyée	POINTER vers le nœud DocumentFragment créé.			

1. Cette fonction crée le nœud mais ne l'attache pas à l'arbre. Pour ce, utilisez les fonctions NWX_NODE_ATTACHBEFORE% et NWX_NODE_ATTACHAFTER%.
2. L'objet DocumentFragment hérite tout de l'objet nœud (méthodes, propriétés, fonctions …). On peut donc effectuer sur l'objet DocumentFragment toutes les opérations qu'on effectue sur un nœud. De plus, l'objet DocumentFragment se manipule comme un nœud et non comme un Document.
3. L'insertion d'un DocumentFragment est intégrale. On ne peut pas insérer un ou des nœuds enfants du DocumentFragment. Cependant, seuls ses nœuds enfants sont transférés et non le DocumentFragment lui-même.
4. L'attachement d'un DocumentFragment à un Document transfert ses nœuds enfants au Document et le nœud DocumentFragment se vide. La fonction NWX_NODE_GETCHILD renvoie 0.

Exemple :

```
local pointer pDoc
local pointer pParser
local sValue$
local ok%
local pointer pNewElement
local pointer pRootNode
local pointer pNode
local pointer pDocFrag
local ok2%

ok2%=NWX_INITIALIZE%
pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
```

```
move "<AUTO><MODELE>Toyota</MODELE><ANNEE>1951</ANNEE></AUTO>" \ to sValue$
pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)
pRootNode = NWX_DOCUMENT_GETROOT (pDoc)
pNode = NWX_NODE_GETCHILD (pRootNode)
pNode = NWX_NODE_GETNEXT (pNode)
pDocFrag = NWX_DOCUMENTFRAGMENT_NEW (pDoc)
pNewElement = NWX_ELEMENT_NEW (pDoc, "TYPE")
ok%=NWX_NODE_SETVALUE% (pNewElement,"Berline")
ok%= NWX_NODE_ATTACHCHILD% (pDocFrag,pNewElement)
pNewElement = NWX_COMMENT_NEW (pDoc, "ceci est un commentaire")
ok%= NWX_NODE_ATTACHCHILD% (pDocFrag,pNewElement)
ok%= NWX_NODE_ATTACHBEFORE% (pNode,pDocFrag)
ok% = NWX_DOCUMENT_OUTPUTFILE%( pDoc, EF_DOC2)
ok% = NWX_DOCUMENT_OUTPUT% (pDoc,@sValue$,sizeof sValue$)
delete from MLE1
INSERT AT END  sValue$ TO MLE1
; dispose le nœud DocumentFragment
NWX_DOCUMENTFRAGMENT_DISPOSE pDocFrag
; dispose le document
NWX_DOCUMENT_DISPOSE pDoc
; Libère le parseur
NWX_PARSER_DISPOSE pParser
Résultat :
<AUTO>
<MODELE>Toyota</MODELE>
<TYPE>Berline</TYPE>
<!--ceci est un commentaire-->
<ANNEE>1951</ANNEE>
</AUTO>
```

Voir aussi ***NWX_NODE_NEW*****,** ***NWX_NODE_ATTACHBEFORE%*****,** ***NWX_NODE_ATTACHAFTER%***

Fonction NWX_DOCUMENTTYPE_NEW_ID (Librairie NWXML)

La fonction NWX_DOCUMENTTYPE_NEW_ID permet de créer un nœud de référence au fichier de validation (DTD ou XSD).

Syntaxe	**NWX_DOCUMENTTYPE_NEW_ID** *(pDoc, sName, sPublicId, sSystemId)*			
Paramètres	pDoc	POINTER	I/O	pointeur d'élément
	Sname	CSTRING	I	nom du Document type
	sPublicId	CSTRING	I	identificateur publique
	sSystemId	CSTRING	I	identificateur système
Valeur renvoyée	POINTER sur le nœud créé			

1. Cette fonction crée le nœud DocumentType mais ne l'attache pas à l'arbre. Pour ce, utilisez les fonctions NWX_NODE_ATTACHBEFORE% et NWX_NODE_ATTACHAFTER%.
2. Pour créer un nœud de référence ayant un identificateur système, il suffit de passer un paramètre vide comme sPublicId.
3. Si les deux identificateurs sont passés en paramètre, l'identificateur public sera prioritairement généré.

Exemple :

```
local pointer pDoc, pointer pDocType
local ok%

ok%=NWX_INITIALIZE%
pDoc =NWX_DOCUMENT_NEW ("XMLDoc")
pDocType = NWX_DOCUMENTTYPE_NEW_ID (pDoc,"client","","client2.dtd")
; le nœud ainsi créé = < !DOCTYPE client SYSTEM "client2.dtd" >

pDocType = NWX_DOCUMENTTYPE_NEW_ID (pDoc,"client","client2.dtd","")
; le nœud ainsi créé = < !DOCTYPE client PUBLIC "client2.dtd" >
```

***Voir aussi** NWX_DOCUMENT_NEW, NWX_NODE_ATTACHBEFORE%, NWX_NODE_ATTACHAFTER%*

Fonction NWX_ENTITYREFERENCE_NEW (Librairie NWXML)

La fonction NWX_ENTITYREFERENCE_NEW permet de créer un nœud de type EntityReference.

Syntaxe	**NWX_ENTITYREFERENCE_NEW** *(pDoc, Name$)*			
Paramètres	pDoc	POINTER	I/O	pointeur de l'élément
	name$	CSTRING	I	nom de l'entité à référencer
Valeur renvoyée	POINTER vers le nœud EntityReference créé.			

1. Le nœud EntityReference est utile pour insérer une référence à une entité sans qu'elle soit transformée par le parseur XML. Le résultat de l'insertion se traduit par la chaîne suivante : « & », le contenu du paramètre name$, suivi de « ; ». Ainsi NWX_ENTITYREFERENCE_NEW (pDoc,"modele") donne &modele;
2. Le nœud EntityReference doit être attaché comme nœud enfant de l'élément qui contient le texte à référencer.

```
<CLIENT>Nom &compagnie ;</CLIENT>
```

Dans cet exemple le nœud EntityReference &compagnie; est un nœud enfant de CLIENT.

3. Tous les descendants du nœud EntityReference sont en lecture seule.

Exemple :

```
local pointer pParser, pointer pDoc
local ok%(1)
local pointer pNode, pointer pRootNode
local pointer pEntRefNode
local ok2%

ok2%=NWX_INITIALIZE%

;Nouveau parseur
pParser = NWX_PARSER_NEW

;Il faut d'abord rendre le parseur en mode Validation
NWX_PARSER_SETMODE pParser, 1
NWX_PARSER_SETINCIGNWHITESPACE pParser, false%

;Nouveau document
pDoc = NWX_DOCUMENT_NEW("XMLdoc")

;Charger le document XML
ok% = NWX_DOCUMENT_LOADFILE%(pDoc, "D:\XMLTest\auto.xml", pParser)

pRootNode =NWX_DOCUMENT_GETROOT (pDoc)
pNode =NWX_NODE_GETCHILD (pRootNode)
;Aller chercher le nœud enfant afin de pouvoir insérer
;devant le nœud text existant dans le fichier
pNode =NWX_NODE_GETCHILD (pNode)

;Créer le noeud Entity Reference
pEntRefNode = NWX_ENTITYREFERENCE_NEW (pDoc,"modele")

;L'attacher devant le noeud text
ok%= NWX_NODE_ATTACHBEFORE% (pNode,pEntRefNode)

ok% = NWX_DOCUMENT_OUTPUTFILE% (pDoc,"D:\XMLTest\auto2.xml")

; Libère le parseur et le document
NWX_PARSER_DISPOSE pParser
NWX_DOCUMENT_DISPOSE pDoc
```

Fichier source :

```
<?xml version="1.0" encoding= "ISO-8859-1" standalone = "yes"?>
<!DOCTYPE AUTO [
<!ENTITY modele "TOYOTA">
<!ELEMENT AUTO (MODELE,ANNEE)>
<!ELEMENT MODELE (#PCDATA)>
<!ELEMENT ANNEE (#PCDATA)>
]>
<AUTO>
<MODELE>
Camry
</MODELE>
<ANNEE>
```

```
1999
</ANNEE>
</AUTO>
```

Résultat :

```
<!DOCTYPE AUTO [
<!ENTITY modele "TOYOTA">
<!ELEMENT AUTO (MODELE,ANNEE)>
<!ELEMENT MODELE (#PCDATA)*>
<!ELEMENT ANNEE (#PCDATA)*>
]><AUTO><MODELE>&modele;
Camry
</MODELE><ANNEE>
1999
</ANNEE></AUTO>
```

Voir aussi ***NWX_NODE_NEW***, ***NWX_NODE_ATTACHBEFORE%***, ***NWX_NODE_ATTACHCHILD%***, ***NWX_NODE_ATTACHAFTER%***, ***NWX_PARSER_SETCREATEENTREFNODES***

Fonction NWX_NODE_ADDVALUE% (Librairie NWXML)

Ajoute une nouvelle valeur à un nœud.

Syntaxe	**NWX_NODE_ADDVALUE%** *(pNode, Value$)*			
Paramètres	pNode	POINTER	I/O	pointeur du nœud
	Value$	CSTRING	I/O	nouvelle valeur
Valeur renvoyée	INT(1)			

1. La taille de Value$ peut être supérieur à 255 caractères.
2. Cette fonction sert uniquement pour les nœuds de type COMMENT, CDATA_SECTION, ATTRIBUTE et TEXT.

Exemple :

```
local pointer pElem
local ok%
local ok2%

ok2%=NWX_INITIALIZE%
pElem = NWX_ELEMENT_NEW(pDoc, "title")
NWX_NODE_SETVALUE%(pElem, "a xml document test")
ok% = NWX_NODE_ADDVALUE%(pElem, "(suite)")
```

Résultat :

```
<title> a xml document test (suite) </title>
```

Voir aussi ***NWX_NODE_SETVALUE%***, ***NWX_NODE_GETVALUE%***

Fonction NWX_NODE_ATTACHAFTER% (Librairie NWXML)

Insère un nouveau nœud frère après le nœud spécifié.

Syntaxe	**NWX_NODE_ATTACHAFTER%** *(pNode, pNewNode)*			
Paramètres	pNode	POINTER	I/O	pointeur du nœud
	pNewNode	POINTER	I/O	pointeur du nouveau nœud
Valeur renvoyée	INT(1)			

Le nœud doit être préalablement créé.
Exemple :

```
local pointer pDoc,pointer pParser
local sValue$
local ok%
local pointer pNewElement
local pointer pRootNode
local pointer pNode
local ok2%

ok2%=NWX_INITIALIZE%

pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")

move "<AUTO><MODELE>Toyota</MODELE><ANNEE>1951</ANNEE></AUTO>"  to sValue$

pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)
pRootNode = NWX_DOCUMENT_GETROOT (pDoc)

;Positionner le pointeur sur le nœud <MODELE>
pNode = NWX_NODE_GETCHILD (pRootNode)

;Créer un nouveau noeud
pNewElement = NWX_ELEMENT_NEW (pDoc, "COULEUR")
ok%=NWX_NODE_SETVALUE% (pNewElement,"Grise")

;Attache le nouveau nœud devant le nœud <ANNEE> et après le
;nœud <MODELE>
ok%= NWX_NODE_ATTACHAFTER% (pNode,pNewElement)
```

Voir aussi NWX_NODE_ATTACHBEFORE%, NWX_NODE_ATTACHCHILD%, NWX_NODE_DETACH%

Fonction NWX_NODE_ATTACHBEFORE% (Librairie NWXML)

Insère un nouveau nœud frère avant le nœud spécifié.

Syntaxe	**NWX_NODE_ATTACHBEFORE%** *(pNode, pNewNode)*			
Paramètres	pNode	POINTER	I/O	pointeur du nœud
	pNewNode	POINTER	I/O	pointeur du nouveau nœud
Valeur renvoyée	INT(1)			

Le nœud doit être préalablement créé.

Exemple :

```
local pointer pDoc,pointer pParser
local sValue$
local ok%
local pointer pNewElement
local pointer pRootNode
local pointer pNode
local ok2%

ok2%=NWX_INITIALIZE%

pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")

move "<AUTO><MODELE>Toyota</MODELE><ANNEE>1951</ANNEE></AUTO>"  to sValue$

pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)
pRootNode = NWX_DOCUMENT_GETROOT (pDoc)

pNode = NWX_NODE_GETCHILD (pRootNode)
;Positionne le pointeur sur le noeud <ANNEE>
pNode = NWX_NODE_GETNEXT (pNode)

;Crée un nouveau noeud
pNewElement = NWX_ELEMENT_NEW (pDoc, "COULEUR")
ok%=NWX_NODE_SETVALUE% (pNewElement,"Grise")

;Attache le nouveau noeud devant le noeud <ANNEE>
ok% = NWX_NODE_ATTACHBEFORE% (pNode,pNewElement)
```

Voir aussi NWX_NODE_ATTACHAFTER%, NWX_NODE_ATTACHCHILD%, NWX_NODE_DETACH%

Fonction NWX_NODE_ATTACHCHILD% (Librairie NWXML)

Insère un nouveau nœud comme enfant du nœud spécifié.

Syntaxe	**NWX_NODE_ATTACHCHILD%** *(pNode, pChildNode)*			
Paramètres	pNode	POINTER	I/O	pointeur du nœud
	pChildNode	POINTER	I/O	pointeur du nœud enfant
Valeur renvoyée	INT(1)			

1. Le nœud doit être préalablement créé.
2. Si l'élément père a déjà des nœuds enfant, le nouveau nœud est inséré à la fin.

Exemple :

```
local pointer pDoc
local pointer pParser
local sValue$
local ok%
local pointer pNewElement
local pointer pRootNode
local pointer pNode
local ok2%

ok2%=NWX_INITIALIZE%

pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")

move "<AUTO><MODELE></MODELE><ANNEE>1951</ANNEE></AUTO>" to sValue$

ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)
pRootNode = NWX_DOCUMENT_GETROOT (pDoc)

pNode = NWX_NODE_GETCHILD (pRootNode)

pNewElement = NWX_ELEMENT_NEW (pDoc, "TYPE")

ok%=NWX_NODE_SETVALUE% (pNewElement,"Berline")

ok% = NWX_NODE_ATTACHCHILD% (pNode,pNewElement)

pNewElement = NWX_ELEMENT_NEW (pDoc, "MARQUE")
ok%=NWX_NODE_SETVALUE% (pNewElement,"TOYOTA")
ok%= NWX_NODE_ATTACHCHILD% (pNode,pNewElement)

ok% = NWX_DOCUMENT_OUTPUTFILE%( pDoc, EF_DOC2)
ok% = NWX_DOCUMENT_OUTPUT% (pDoc,@sValue$,sizeof sValue$)
delete from MLE1
INSERT AT END  sValue$ TO MLE1
; dispose le document
NWX_DOCUMENT_DISPOSE pDoc
```

```
; Libère le parseur
NWX_PARSER_DISPOSE pParser
```

Résultat :

```
<AUTO>
    <MODELE>
        <TYPE>Berline</TYPE>
        <MARQUE>TOYOTA</MARQUE>
    </MODELE>
    <ANNEE>1951</ANNEE>
</AUTO>
```

Voir aussi ***NWX_NODE_ATTACHBEFORE%*** **,** ***NWX_NODE_ATTACHAFTER%*****,** ***NWX_NODE_DETACH%***

Fonction NWX_NODE_CLONENODE (Librairie NWXML)

La fonction NWX_NODE_CLONENODE permet de reproduire un nœud. Cette fonction est utilisée comme constructeur générique pour copier des nœuds. Le nœud dupliqué n'a pas de parent (parentNode retourne null).

Syntaxe	**NWX_NODE_CLONENODE** *(pNode, deep%)*			
Paramètres	pNode	POINTER	I/O	pointeur du nœud
	deep%	INT(1)	I	état (TRUE% ou FALSE%)
Valeur renvoyée	POINTER vers le nœud dupliqué.			

1. Quand deep%=true%, la fonction copie récursivement le sous-arbre sous le nœud spécifié.
2. Quand deep%=false%, seul le nœud lui-même est cloné sans la valeur qu'il contient (avec ses attributs, si c'est un Element).
3. La duplication d'un nœud Element entraîne la copie de tous les attributs et leurs valeurs. Les attributs générés par le processeur XML sont également copiés pour représenter les attributs ayant des valeurs par défaut. Mais cette méthode ne copie pas la valeur du nœud si deep%=false.

Exemple :

```
local pointer pDoc, pointer pParser
local pointer pClonedNode, pointer pNode,pointer pRoot
local sValue$
local ok% , deep%
local ok2%

ok2%=NWX_INITIALIZE%

pParser = NWX_PARSER_NEW
```

```
pDoc = NWX_DOCUMENT_NEW("XMLdoc")

move "<BOOK><TITLE>Book's title</TITLE></BOOK>" to sValue$

pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")

ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)
pRoot = NWX_DOCUMENT_GETROOT (pDoc)

pNode = NWX_NODE_GETCHILD (pRoot)

deep% =1
pClonedNode = NWX_NODE_CLONENODE (pNode,deep%)

ok% = NWX_DOCUMENT_OUTPUT% (pClonedNode,@sValue$,sizeof sValue$)

;sValue contiendra <TITLE>Book's title</TITLE>
```

Voir aussi NWX_NODE_REPLACECHILD, NWX_NODE_GETCHILD

Fonction NWX_NODE_DETACH% (Librairie NWXML)

Retire d'un document le nœud spécifié.

Syntaxe	**NWX_NODE_DETACH%** *(pNode)*			
Paramètre	pNode	POINTER	I/O	pointeur du nœud
Valeur renvoyée	INT(1)			

1. Cette fonction enlève le nœud, mais le nœud n'est pas libéré. Pour libérer un nœud, utilisez NWX_NODE_DISPOSE .
2. Si le nœud retiré a des nœuds enfant, ceux-ci seront également retirés mais non libérés

Exemple :

```
local pointer pDoc, pointer pParser
local sValue$
local ok%
local pointer pRootNode,pointer pNode
local ok2%

ok2%=NWX_INITIALIZE%

pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")

move "<AUTO><MODELE>Toyota</MODELE><ANNEE>1920</ANNEE></AUTO>" \ to sValue$

pParser = NWX_PARSER_NEW

pDoc = NWX_DOCUMENT_NEW("XMLdoc")
```

```
ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)
pRootNode = NWX_DOCUMENT_GETROOT (pDoc)
;Se positionner sur le noeud <MODELE>
pNode = NWX_NODE_GETCHILD (pRootNode)
;Détacher ce nœud
ok%=NWX_NODE_DETACH% (pNode)
NWX_NODE_DISPOSE(pNode)
```

Résultat :

```
;pDoc contient maintenant
<AUTO>
<ANNEE>1920</ANNEE>
</AUTO>
```

Voir aussi NWX_NODE_DISPOSE

Instruction NWX_NODE_DISPOSE (Librairie NWXML)

Libère le nœud spécifié.

Syntaxe	**NWX_NODE_DISPOSE** *pNode*			
Paramètre	pNode	POINTER	I/O	pointeur du noeud

Si le nœud n'est pas détaché, cette instruction le détache puis le libère.
Exemple :

```
local pointer pDoc, pointer pParser
local sValue$
local ok%
local pointer pRootNode,pointer pNode
local ok2%

ok2%=NWX_INITIALIZE%
pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")

move "<AUTO><MODELE>Toyota</MODELE><ANNEE>1920</ANNEE></AUTO>" to sValue$

pParser = NWX_PARSER_NEW

pDoc = NWX_DOCUMENT_NEW("XMLdoc")

ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)
pRootNode = NWX_DOCUMENT_GETROOT (pDoc)
;Se positionner sur le noeud <MODELE>
pNode = NWX_NODE_GETCHILD (pRootNode)
;Détacher ce nœud
ok%=NWX_NODE_DETACH% (pNode)
NWX_NODE_DISPOSE(pNode)
```

Résultat :

```
;pDoc contient maintenant
<AUTO>
<ANNEE>1920</ANNEE>
</AUTO>
```

Voir aussi ***NWX_NODE_DETACH%, NWX_DOCUMENT_OUTPUTFILE%, NWX_DOCUMENT_OUTPUT, NWX_NODE_OUTPUTFILE%, NWX_NODE_OUTPUT%, NWX_DOCUMENT_LOADFILE, NWX_DOCUMENT_LOAD%, NWX_DOCUMENT_DISPOSE, NWX_DOCUMENT_OUT_DYNSTR***

Instruction NWX_NODELIST_DISPOSE (Librairie NWXML)

Libère la liste spécifiée.

Syntaxe	**NWX_NODELIST_DISPOSE** *pNodeList*			
Paramètre	pNodeList	POINTER	I/O	pointeur de liste

1. pNodeList est une liste de nœuds qui ont tous la même balise.
2. Vous devez fermer la liste ouverte quand vous utilisez NWX_ELEMENT_GETELEMSBYTAGNAME ou NWX_DOCUMENT_GETELEMSBYTAGNAME

Exemple :

```
local pointer pDoc
local pointer pParser
local sValue$
local ok%
local pointer pNodeList
local pointer pNode
local pointer ptext
local ok2%

ok2%=NWX_INITIALIZE%
delete from MLE1
pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
NWX_PARSER_SETCREATEENTREFNODES pParser, TRUE%
move '<AUTO><MODELE>Toyota</MODELE><MODELE>Peugeot</MODELE> \
<MODELE>Jaguar</MODELE></AUTO>' to sValue$

ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)

;I do not deal with errors
pNodeList = NWX_DOCUMENT_GETELEMSBYTAGNAME (pDoc,'MODELE')

pNode = NWX_NODELIST_GETFIRST (pNodeList)
WHILE pnode
 ok% = NWX_NODE_OUTPUT%(pNode,@svalue$, sizeof svalue$)
 INSERT AT END  sValue$ TO MLE1
 pNode = NWX_NODELIST_GETNEXT (pNodeList,pNode)
ENDWHILE
; dispose list
NWX_NODELIST_DISPOSE pNodeList
; dispose document
```

```
NWX_DOCUMENT_DISPOSE pDoc
; free parser
NWX_PARSER_DISPOSE pParser
```

Résultat :

```
<MODELE>Toyota</MODELE>
<MODELE>Peugeot</MODELE>
<MODELE>Jaguar</MODELE>
```

Voir aussi ***NWX_ELEMENT_GETELEMSBYTAGNAME***, ***NWX_DOCUMENT_GETELEMSBYTAGNAME***

Fonction NWX_NODE_GETCHILD (Librairie NWXML)

Retourne un pointeur sur le nœud enfant du nœud passé en paramètre.

Syntaxe	**NWX_NODE_GETCHILD** *(pNode)*			
Paramètre	pNode	POINTER	I/O	pointeur du nœud
Valeur renvoyée	POINTER sur le nœud enfant			

Exemple :

```
local pointer pDoc, pointer pParser
local sValue$
local ok%
local pointer pNewElement, pointer pRootNode
local pointer pChildNode
local ok2%

ok2%=NWX_INITIALIZE%
pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")

move "<AUTO><MODELE>Toyota</MODELE><ANNEE>1951</ANNEE></AUTO>"  to sValue$

ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)

pRootNode = NWX_DOCUMENT_GETROOT (pDoc)
pChildNode = NWX_NODE_GETCHILD (pRootNode)
```

Résultat :

```
; pChild pointe sur le nœud suivant
<MODELE>Toyota</MODELE>
```

Voir aussi ***NWX_NODE_GETPREV***, ***NWX_NODE_GETNEXT***, ***NWX_NODE_GETPARENT***

Fonction NWX_NODE_GETPARENT (Librairie NWXML)

Retourne un pointeur sur le nœud père.

Syntaxe	**NWX_NODE_GETPARENT** *(pNode)*			
Paramètre	pNode	POINTER	I/O	pointeur du nœud
Valeur renvoyée	POINTER sur le nœud.			

Exemple

```
local pointer pDoc
local pointer pParser
local sValue$
local ok%
local pointer pNewElement
local pointer pRootNode
local pointer pNode
local ok2%

ok2%=NWX_INITIALIZE%

pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")

move "<AUTO><MODELE><TYPE>TOYOTA \
Berline</TYPE></MODELE><ANNEE>1951</ANNEE></AUTO>" to sValue$

ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)
pRootNode = NWX_DOCUMENT_GETROOT (pDoc)
  pNode = NWX_NODE_GETCHILD (pRootNode)
pNode = NWX_NODE_GETNEXT (pNode)    ;Annee
pNode = NWX_NODE_GETPARENT (pNode)  ;Auto

ok% = NWX_NODE_OUTPUT% (pNode,@sValue$,sizeof sValue$)\
delete from MLE1
INSERT AT END  sValue$ TO MLE1
; dispose le document
NWX_DOCUMENT_DISPOSE pDoc
; Libère le parseur
NWX_PARSER_DISPOSE pParser
```

Résultat :

```
<AUTO>
<MODELE>
<TYPE>TOYOTA Berline</TYPE>
</MODELE>
<ANNEE>1951</ANNEE>
</AUTO>
```

Voir aussi ***NWX_NODE_GETCHILD***, ***NWX_NODE_GETPREV***, ***NWX_NODE_GETNEXT***

Fonction NWX_NODE_GETNEXT (Librairie NWXML)

Retourne un pointeur sur le nœud frère suivant.

Syntaxe	**NWX_NODE_GETNEXT** *(pNode)*			
Paramètre	pNode	POINTER	I/O	pointeur du nœud
Valeur renvoyée	POINTER sur le nœud frère suivant			

Exemple :

```
local pointer pDoc
local pointer pParser
local sValue$
local ok%
local pointer pNewElement
local pointer pRootNode
local pointer pNode
local ok2%

ok2%=NWX_INITIALIZE%
pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")

move "<AUTO><MODELE><TYPE>TOYOTA \
Berline</TYPE></MODELE><ANNEE>1951</ANNEE></AUTO>" to sValue$

ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)
pRootNode = NWX_DOCUMENT_GETROOT (pDoc)
  pNode = NWX_NODE_GETCHILD (pRootNode)
pNode = NWX_NODE_GETNEXT (pNode)    ;Année

ok% = NWX_NODE_OUTPUT% (pNode,@sValue$,sizeof sValue$)delete from MLE1
INSERT AT END  sValue$ TO MLE1
; dispose le document
NWX_DOCUMENT_DISPOSE pDoc
; Libère le parseur
NWX_PARSER_DISPOSE pParser
Résultat :
<ANNEE>1951</ANNEE>
```

Voir aussi ***NWX_NODE_GETCHILD***, ***NWX_NODE_GETPREV***, ***NWX_NODE_GETPARENT***

Fonction NWX_NODE_GETPREV (Librairie NWXML)

Retourne un pointeur sur le nœud frère précédent.

Syntaxe	**NWX_NODE_GETPREV** *(pNode)*			
Paramètre	pNode	POINTER	I/O	pointeur du nœud
Valeur renvoyée	POINTER			

Exemple :

```
local pointer pDoc
local pointer pParser
local sValue$
local ok%
local pointer pNewElement
local pointer pRootNode
local pointer pNode
local ok2%

ok2%=NWX_INITIALIZE%

pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")

move "<AUTO><MODELE><TYPE>TOYOTA \
Berline</TYPE></MODELE><ANNEE>1951</ANNEE></AUTO>" to sValue$

ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)
pRootNode = NWX_DOCUMENT_GETROOT (pDoc)
  pNode = NWX_NODE_GETCHILD (pRootNode)
pNode = NWX_NODE_GETNEXT (pNode)    ;Annee
pNode = NWX_NODE_GETPREV (pNode)    ;MODELE

ok% = NWX_NODE_OUTPUT% (pNode,@sValue$,sizeof sValue$)delete from MLE1
INSERT AT END  sValue$ TO MLE1
; dispose le document
NWX_DOCUMENT_DISPOSE pDoc
; Libère le parseur
NWX_PARSER_DISPOSE pParser
```

Résultat :

```
<MODELE><TYPE>TOYOTA Berline</TYPE></MODELE>
```

Voir aussi NWX_NODE_GETCHILD, NWX_NODE_GETNEXT, NWX_NODE_GETPARENT

Fonction NWX_NODE_GETTYPE% (Librairie NWXML)

La fonction NWX_NODE_GETTYPE% retourne le type d'un nœud.

Syntaxe	**NWX_NODE_GETTYPE%** *(pNode)*			
Paramètre	pNode	POINTER	I/O	pointeur du nœud
Valeur renvoyée	INT(1)			

Les valeurs renvoyées peuvent être :

- NWX_ELEMENT%
- NWX_TEXT%
- NWX_CDATA_SECTION%
- NWX_PROCESSING_INSTRUCTION%
- NWX_COMMENT%

- NWX_NOTATION%
- NWX_ATTRIBUTE%
- NWX_ENTITY_REFERENCE%
- NWX_ENTITY%
- NWX_DOCUMENT_TYPE%
- NWX_DOCUMENT_FRAGMENT%
- NWX_DOCUMENT%

Exemple :

```
local pointer pDoc
local ok%(1)
local ok2%

ok2%=NWX_INITIALIZE%

; Nouveau document
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
ok% = NWX_NODE_GETTYPE% (pDoc)
; Retourne 12 pour un nœud de type Document
```

Voir aussi NWX_DOCUMENT_NEW

Fonction NWX_NODE_GETUSERDATA (Librairie NWXML)

La fonction NWX_NODE_GETUSERDATA retourne le pointeur de la donnée utilisateur.

Syntaxe	**NWX_NODE_GETUSERDATA** *(pNode)*			
Paramètre	pNode	POINTER	I/O	pointeur du nœud
Valeur renvoyée	POINTER vers la donnée utilisateur.			

Ne s'applique pas aux nœuds de type Document.

Exemple :

```
local pointer pDoc
local pointer pParser
local sValue$
local ok%,i%,h%
local pointer pNode
local sVal$(101)
local ok2%

ok2%=NWX_INITIALIZE%

pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")

move "<AUTO><MODELE>Toyota</MODELE><ANNEE>1951</ANNEE></AUTO>" \ to sValue$
```

```
ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)
pNode = NWX_DOCUMENT_GETROOT (pDoc)

new 1000, h%
fill h%, 100, 0 ; \0
fill h%, 99, 65;  A

; association du handle avec nœud
NWX_NODE_SETUSERDATA pNode, h%

; récupération du handle du nœud
i% = NWX_NODE_GETUSERDATA (pNode)

mov i%, @sval$, 100

delete from MLE1
INSERT AT END sval$ TO MLE1; affiche 99 fois A
; dispose la mémoire allouée avec new
dispose h%
; dispose le document
NWX_DOCUMENT_DISPOSE pDoc
; Libère le parseur
NWX_PARSER_DISPOSE pParser
```

Voir aussi ***NWX_NODE_SETUSERDATA***

Fonction NWX_NODE_GETVALUE% (Librairie NWXML)

Retourne la valeur et la taille d'un nœud.

Syntaxe	**NWX_NODE_GETVALUE%** *(pNode, pValue, Size%)*			
Paramètres	pNode	POINTER	I/O	pointeur du nœud
	pValue	POINTER	I/O	pointeur du buffer avec valeur
	Size%	INT(4)	I/O	taille du buffer
Valeur renvoyée	INT(4)			

1. Pour la taille, vous devez ajouter 1 pour la terminaison "\0"
2. Si la taille de la sortie est plus longue que la valeur spécifiée, la chaîne est tronquée. Pour pallier les problèmes de taille, utiliser plutôt la fonction NWX_NODE_GETVALUE qui utilise une chaîne de taille variable de type DynStr pour la sortie.

3. Si XMLbuf ou Size% est égal à 0, la valeur renvoyée est la taille requise pour contenir toute la chaîne en sortie.
4. La valeur renvoyée est toujours la taille de la chaîne en sortie.

Exemple :

```
local pointer pDoc,pointer pParser
local sValue$
local ok%
local pointer pNewElement, pointer pRootNode
local pointer pNode
local ok2%

ok2%=NWX_INITIALIZE%

pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")

move "<AUTO><MODELE>Toyota</MODELE><ANNEE>1951</ANNEE></AUTO>"  to sValue$

ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)

pRootNode = NWX_DOCUMENT_GETROOT (pDoc)
pNode = NWX_NODE_GETCHILD (pRootNode)

ok% = NWX_NODE_GETVALUE%(pNode, @sValue$, sizeof sValue$)

MESSAGE "Modele", sValue$
;sValue$ contiendra Toyota
```

***Voir aussi** NWX_NODE_GETVALUE, NWX_NODE_SETVALUE%, NWX_NODE_ADDVALUE%*

Fonction NWX_NODE_GETVALUE (Librairie NWXML)

Retourne la valeur d'un nœud.

Syntaxe	**NWX_NODE_GETVALUE%** *(pNode)*			
Paramètre	pNode	POINTER	I/O	pointeur du nœud
Valeur renvoyée	DYNSTR			

A ne pas confondre avec la fonction NWX_NODE_GETVALUE% qui retourne la valeur et la taille d'un nœud. Par ailleurs, avec la fonction NWX_NODE_GETVALUE%, si la taille de la sortie est plus longue que la valeur spécifiée, la chaîne est tronquée. Pour pallier les problèmes de taille, utiliser plutôt la fonction NWX_NODE_GETVALUE qui utilise une chaîne de taille variable de type DynStr pour la sortie.

Exemple :

```
local pointer pDoc
local pointer pParser
local sValue$
local pointer pRootNode
local pointer pNode
```

```
local ret%
local dynstr ds
ret% = NWX_INITIALIZE%
IF ret% <> 0
EXIT
ENDIF
pParser = NWX_PARSER_NEW
sValue$ = "<AUTO><MODELE>Toyota</MODELE><ANNEE>1951</ANNEE></AUTO>"
pDoc = NWX_DOCUMENT_GETFROMBUFFER(sValue$, pParser)
pRootNode = NWX_DOCUMENT_GETROOT (pDoc)
pNode = NWX_NODE_GETCHILD (pRootNode)
ds = NWX_NODE_GETVALUE(pNode)
MESSAGE "Modele", ds
;ds contiendra Toyota
```

***Voir aussi** NWX_NODE_GETVALUE%, NWX_NODE_SETVALUE%, NWX_NODE_ADDVALUE%*

Fonction NWX_NODE_NEW (Librairie NWXML)

Crée un nœud avec le type spécifié.

Syntaxe	**NWX_NODE_NEW** *(pDoc, NodeType%, NodeName$)*			
Paramètres	pDoc	POINTER	I/O	pointeur du document
	NodeType%	CONST	I/O	type de nœud
	NodeName$	CSTRING	I/O	nom de nœud
Valeur renvoyée	POINTER			

1. Si le nœud est NWX_ELEMENT% , NodeName$ est le nom de la balise de l'élément.
2. Les documents XML se présentent sous forme arborescente avec les nœuds. Un nœud peut être un élément, commentaire, ou une instruction de process...
3. Les valeurs possibles de NodeType% sont :
   a) NWX_ELEMENT%
   b) NWX_TEXT%
   c) NWX_CDATA_SECTION%
   d) NWX_COMMENT%
   e) NWX_NOTATION%
   f) NWX_ATTRIBUTE%
   g) NWX_ENTITY_REFERENCE%
   h) NWX_ENTITY%
   i) NWX_DOCUMENT_FRAGMENT%

4. Les nœuds de types Document, Document Type, et Processing Instruction doivent être créer respectivement avec NWX_DOCUMENT_NEW, NWX_DOCUMENTTYPE_NEW_ID, et NWX_PROCESSINGINSTUCTION_NEW.
5. Il est impossible de modifier la valeur des nœuds de type Notation et Entity au niveau actuel des spécifications DOM. Il est donc peu utile de créer des nœuds de ce type.
6. Les nœuds créés, attacher-les à l'arbre avec les fonctions NWX_NODE_ATTACH*.

Exemple :

```
local pointer pDoc
local pointer pParser, pointer pRootNode
local pointer pNodeNew, pointer pNode
local value$, buffer$
local ok%
local ok2%

ok2%=NWX_INITIALIZE%
;Nouveau parseur
pParser = NWX_PARSER_NEW
;Nouveau document
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
buffer$ ="<?xml version='1.0' encoding='ISO-8859-1' \
standalone='yes'?><AUTO><MODELE>Toyota</MODELE><ANNEE>1953</ANNEE></AUTO>"
;Charger le buffer dans pDoc et pParser
ok% = NWX_DOCUMENT_LOAD%(pDoc, buffer$, pParser)
;Chercher le noeud racine du document
pRootNode = NWX_DOCUMENT_GETROOT (pDoc)
pNode = NWX_NODE_GETCHILD (pRootNode)
pNodeNew = NWX_NODE_NEW (pDoc,NWX_TEXT%,"DESCRIPTION")
ok% = NWX_NODE_ATTACHBEFORE% (pNode,pNodeNew)
;Afficher ce noeud
ok% = NWX_NODE_OUTPUT% (pRootNode,@value$, sizeof value$)
MLE1 = value$
```

Résultat :

```
<AUTO>DESCRIPTION
<MODELE>Toyota</MODELE>
<ANNEE>1953</ANNEE>
</AUTO>
```

Voir aussi ***NWX_NODE_ATTACHBEFORE%, NWX_NODE_ATTACHAFTER%, NWX_NODE_ATTACHCHILD%, NWX_DOCUMENT_NEW, NWX_DOCUMENTTYPE_NEW_ID, Constantes***

Fonction NWX_NODE_REPLACECHILD (Librairie NWXML)

Cette fonction permet de remplacer un nœud enfant poldChild par un nouveau nœud enfant pnewChild. La fonction retourne le pointeur au nœud poldChild.

Syntaxe	**NWX_NODE_REPLACECHILD** *(pnewChild, poldChild)*

Paramètres	pnewChild	POINTER	I/O	pointeur du nouveau nœud enfant
	poldChild	POINTER	I/O	pointeur de l'ancien nœud enfant
Valeur renvoyée	POINTER vers l'ancien nœud enfant.			

Si newChild est un nœud DocumentFragment, poldChild est remplacé par tous les fragments DocumentFragment enfants, qui sont insérés dans le même ordre. Si pnewChild est déjà dans l'arbre, alors il est déplacé

Exemple :

```
local pointer pDoc,pointer pParser
local sValue$
local ok%
local pointer pNewElement,pointer pRootNode, pointer pNode
local ok2%

ok2%=NWX_INITIALIZE%
pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")

move "<AUTO><MODELE>Toyota</MODELE></AUTO>" to sValue$

pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)
pRootNode = NWX_DOCUMENT_GETROOT (pDoc)

pNode = NWX_NODE_GETCHILD (pRootNode)
;Créer un nouveau noeud
pNewElement = NWX_ELEMENT_NEW (pDoc, "MARQUE")
;Lui donner une valeur
ok%=NWX_NODE_SETVALUE% (pNewElement,"Toyota")
;Remplacer le noeud <MODELE> par le nouveau noeud <MARQUE>
pNewElement =NWX_NODE_REPLACECHILD (pNewElement,pNode)
```

Voir aussi NWX_NODE_CLONENODE

Instruction NWX_NODE_SETUSERDATA (Librairie NWXML)

L'instruction NWX_NODE_SETUSERDATA permet de spécifier des informations supplémentaires pour un nœud. Ces informations n'ont aucun impact sur le document généré. Elles sont à usage interne et peuvent être récupérés par la fonction NWX_NODE_GETUSERDATA.

Syntaxe	**NWX_NODE_SETUSERDATA** *pNode, pUserData*			
Paramètres	pNode	POINTER	I/O	pointeur du nœud
	pUserData	POINTER	I/O	pointeur de l'utilisateur

Ne s'applique pas aux nœuds de type Document.
Exemple :

```
local sValue$
local ok%,i%,h%
local pointer pNode
local sVal$(101)
local ok2%

ok2%=NWX_INITIALIZE%
pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")

move "<AUTO><MODELE>toyota</MODELE><ANNEE>1951</ANNEE></AUTO>" to sValue$

pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")

ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)
pNode = NWX_DOCUMENT_GETROOT (pDoc)

new 1000, h%
fill h%, 100, 0 ; \0
fill h%, 99, 65;  A
; association du handle avec noeud
NWX_NODE_SETUSERDATA pNode, h%
; récupération du handle du noeud
i% = NWX_NODE_GETUSERDATA (pNode)
mov i%, @sval$, 100
delete from MLE1
INSERT AT END sval$ TO MLE1; affiche 99 fois A
; dispose la mémoire allouée avec new
dispose h%
; dispose le document
NWX_DOCUMENT_DISPOSE pDoc
; Libère le parseur
NWX_PARSER_DISPOSE pParser
```

Voir aussi NWX_NODE_GETUSERDATA

Fonction NWX_NODE_SETVALUE% (Librairie NWXML)

Définit la valeur d'un nœud.

Syntaxe	**NWX_NODE_SETVALUE%** *(pNode, Value$)*

Paramètres	pNode	POINTER	I/O	pointeur du nœud
	Value$	CSTRING	I/O	valeur du nœud
Valeur renvoyée	INT(1)			

1. Les types de nœuds possibles sont :
   a) NWX_ATTRIBUTE%
   b) NWX_ELEMENT%
   c) NWX_TEXT%
   d) NWX_CDATA_SECTION%
   e) NWX_COMMENT%
2. Les autres types de nœud ont une valeur égale à NULL qui ne peut être modifié.
3. La taille de Value$ peut être supérieure à 255 caractères.

Exemple :

```
local pointer pElem
local ok%

ok%=NWX_INITIALIZE%
pElem = NWX_ELEMENT_NEW(pDoc, "title")
NWX_NODE_SETVALUE%(pElem, "a XML document test")
```

Résultat :

```
<title> a xml document test </title>
```

Voir aussi NWX_NODE_ADDVALUE%, NWX_NODE_GETVALUE%

Fonction NWX_NODELIST_GETFIRST (Librairie NWXML)

Retourne le premier nœud de la liste.

Syntaxe	**NWX_NODELIST_GETFIRST** *(pNodeList)*			
Paramètre	pNodeList	POINTER	I/O	pointeur de liste
Valeur renvoyée	POINTER			

pNodeList est une liste de nœuds qui ont la même balise.

Exemple :

```
local pointer pDoc
local pointer pParser
local sValue$
local ok%
```

```
local pointer pNodeLists
local pointer pNode
local pointer ptext
local ok2%

ok2%=NWX_INITIALIZE%

delete from MLE1
pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
NWX_PARSER_SETCREATEENTREFNODES pParser, TRUE%
move
'<AUTO><MODELE>Toyota</MODELE><MODELE>Peugeot</MODELE><MODELE>Jaguar</MODEL
E></AUTO>' to sValue$

ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)

;I do not deal with errors
pNodeList = NWX_DOCUMENT_GETELEMSBYTAGNAME (pDoc,'MODELE')
pNode = NWX_NODELIST_GETFIRST (pNodeList)
WHILE pnode
 ok% = NWX_NODE_OUTPUT%(pNode,@svalue$, sizeof svalue$)
 INSERT AT END  sValue$ TO MLE1
 pNode = NWX_NODELIST_GETNEXT (pNodeList,pNode)
ENDWHILE

; dispose list
NWX_NODELIST_DISPOSE pNodeList
; dispose  document
NWX_DOCUMENT_DISPOSE pDoc
; free  parser
NWX_PARSER_DISPOSE pParser
```

Résultat :

```
<MODELE>Toyota</MODELE>
<MODELE>Peugeot</MODELE>
<MODELE>Jaguar</MODELE>
```

Voir aussi ***NWX_ELEMENT_GETELEMSBYTAGNAME*** , ***NWX_ELEMENT_GETTAGNAME%***, ***NWX_DOCUMENT_GETELEMSBYTAGNAME*** , ***NWX_NODELIST_GETNEXT***

Fonction NWX_NODELIST_GETNEXT (Librairie NWXML)

Retourne le nœud suivant sur la liste.

Syntaxe	**NWX_NODELIST_GETNEXT** *(pNodeList, pNode)*			
Paramètres	pNodeList	POINTER	I/O	pointeur de liste
	pNode	POINTER	I/O	pointeur du nœud

Valeur renvoyée	POINTER

pNodeList est une liste de nœuds qui ont la même balise.
Exemple :

```
local pointer pDoc
local pointer pParser
local sValue$
local ok%
local pointer pNodeList
local pointer pNode
local pointer ptext
local ok2%

ok2%=NWX_INITIALIZE%

delete from MLE1
pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
NWX_PARSER_SETCREATEENTREFNODES pParser, TRUE%
move
'<AUTO><MODELE>Toyota</MODELE><MODELE>Peugeot</MODELE><MODELE>Jaguar</MODEL
E></AUTO>' to sValue$

ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)

;I do not deal with errors
pNodeList = NWX_DOCUMENT_GETELEMSBYTAGNAME (pDoc,'MODELE')

pNode = NWX_NODELIST_GETFIRST (pNodeList)
WHILE pnode
 ok% = NWX_NODE_OUTPUT%(pNode,@svalue$, sizeof svalue$)
 INSERT AT END  sValue$ TO MLE1
 pNode = NWX_NODELIST_GETNEXT (pNodeList,pNode)
ENDWHILE
; dispose list
NWX_NODELIST_DISPOSE pNodeList
; dispose  document
NWX_DOCUMENT_DISPOSE pDoc
; free  parser
NWX_PARSER_DISPOSE pParser
```

Résultat :

```
<MODELE>Toyota</MODELE>
<MODELE>Peugeot</MODELE>
<MODELE>Jaguar</MODELE>
```

Voir aussi ***NWX_ELEMENT_GETELEMSBYTAGNAME***, ***NWX_ELEMENT_GETTAGNAME%***, ***NWX_DOCUMENT_GETELEMSBYTAGNAME***, ***NWX_NODELIST_GETFIRST***

Fonction NWX_PROCESSINGINSTRUCTION_NEW (Librairie NWXML)

La fonction NWX_PROCESSINGINSTRUCTION_NEW permet de créer un nœud de type ProcessingInstruction à partir du nom et des données spécifiées.

Syntaxe	**NWX_PROCESSINGINSTRUCTION_NEW** *(pDoc, name$, data$)*			
Paramètres	pDoc	POINTER	I/O	pointeur du document
	name$	CSTRING	I/O	nom du nœud
	data$	CSTRING	I/O	données du nœud
Valeur renvoyée	POINTER vers le nœud ProcessingInstruction créé.			

1. Le nœud créé, attachez-le à l'arbre avec les fonctions NWX_NODE_ATTACH*.
2. Vous pouvez vous servir de cette fonction pour créer une déclaration XML.

Exemple :

```
local pointer pDoc,pointer pParser
local pointer pXmlDecl, pointer pNode
local pointer pRootNode, pointer pProcInst
local sValue$
local ok%
local ok2%

ok2%=NWX_INITIALIZE%

pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")

;Créer la déclaration XML
pXmlDecl = NWX_PROCESSINGINSTRUCTION_NEW (pDoc,"xml","version='1.0'
encoding='ISO-8859-1' standalone='yes'")
;Attacher comme premier de la racine du document
ok%=NWX_NODE_ATTACHCHILD% (pDoc,pXmlDecl)

;Créer le noeud Processing instruction
pProcInst = NWX_PROCESSINGINSTRUCTION_NEW (pDoc,"xml-
stylesheet","type='text/xsl' href='auto.xsl'")

;Attacher après la déclaration XML
ok%=NWX_NODE_ATTACHCHILD% (pDoc,pProcInst)

;Créer le noeud racine de l'arbre XML
pRootNode =NWX_NODE_NEW (pDoc,NWX_ELEMENT%,"AUTO")

;Attacher au document
ok%=NWX_NODE_ATTACHCHILD% (pDoc,pRootNode)

;Créer d'autres noeuds
pNode =NWX_NODE_NEW (pDoc,NWX_ELEMENT%,"MODELE")
ok%=NWX_NODE_SETVALUE% (pNode,"Peugeot")
```

```
;Attacher au noeud racine de l'arbre XML
ok%=NWX_NODE_ATTACHCHILD% (pRootNode,pNode)

ok%= NWX_NODE_OUTPUT% (pDoc,@sValue$,sizeof sValue$)
MLE1 = sValue$
NWX_PARSER_DISPOSE pParser
NWX_DOCUMENT_DISPOSE pDoc
```

Résultat :

```
<?xml version= '1.0' encoding= 'ISO-8859-1' standalone= 'yes'?>
<?xml-stylesheet type='text/xsl' href='auto.xsl'?>
<AUTO>
<MODELE>Peugeot</MODELE>
</AUTO>
```

Voir aussi ***NWX_NODE_NEW*****,** ***NWX_NODE_ATTACHBEFORE%*****,** ***NWX_NODE_ATTACHAFTER%*****,** ***NWX_NODE_ATTACHCHILD%*****.**

Fonction NWX_TEXTNODE_NEW (Librairie NWXML)

La fonction NWX_TEXTNODE_NEW permet de créer un nœud de type TEXT avec la chaîne de caractères spécifié.

Syntaxe	**NWX_TEXTNODE_NEW** *(pDoc, data$)*			
Paramètres	pDoc	POINTER	I/O	pointeur du document
	data$	CSTRING	I/O	données du nœud
Valeur renvoyée	POINTER vers le nœud Text créé.			

1. Le nœud créé, attachez-le à l'arbre avec les fonctions NWX_NODE_ATTACH*.
2. Un nœud de type texte ne nécessite pas de balises.

Exemple :

```
local pointer pDoc
local pointer pParser
local sValue$
local ok%
local pointer pRootNode, pointer pNode
local pointer pTextNode
local ok2%

ok2%=NWX_INITIALIZE%
pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
move
'<AUTO><MODELE>Toyota</MODELE><MODELE>Peugeot</MODELE><MODELE>Jaguar</MODEL
E></AUTO>' to sValue$
ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)
```

```
;<AUTO>
pRootNode = NWX_DOCUMENT_GETROOT (pDoc)
;<MODELE>
pNode = NWX_NODE_GETCHILD (pRootNode)
pTextNode = NWX_TEXTNODE_NEW (pDoc,"Les modèles")
;Attacher avant <MODELE>
ok%= NWX_NODE_ATTACHBEFORE% (pNode,pTextNode)
ok%= NWX_DOCUMENT_OUTPUT% (pDoc,@sValue$,sizeof sValue$)
MLE1 = sValue$
NWX_PARSER_DISPOSE pParser
NWX_DOCUMENT_DISPOSE pDoc
```

Résultat :

```
<AUTO>Les modèles
<MODELE>Toyota</MODELE>
<MODELE>Peugeot</MODELE>
<MODELE>Jaguar</MODELE>
</AUTO>
```

***Voir aussi** NWX_DOCUMENT_NEW, NWX_NODE_ATTACHBEFORE%, NWX_NODE_ATTACHAFTER%, NWX_NODE_ATTACHCHILD%.*

Fonction NWX_NODE_OUT_DYNSTR (Librairie NWXML)

Exporte le contenu d'un nœud vers une chaine dynamique.

Syntaxe	**NWX_NODE_OUT_DYNSTR** *(pNode)*			
Paramètre	pNode	POINTER	I/O	pointeur du nœud
Valeur renvoyée	DYNSTR chaîne représentant le noeud.			

Exemple :

```
local pointer pDoc
local pointer pParser, pointer pRootNode
local buffer$
local DynStr dsNode
local ret%
; Initialize NWXML
ret% = NWX_INITIALIZE%
;New parseur
pParser = NWX_PARSER_NEW
; load the buffer in pDoc and pParser
buffer$ = '<?xml version="1.0" encoding="ISO-8859-1"
standalone="yes"?><AUTO><MODELE>Toyota</MODELE><ANNEE>1953</ANNEE></AUTO>'
pDoc = NWX_DOCUMENT_GETFROMBUFFER(buffer$, pParser)
; Search the root node of the document
pRootNode = NWX_DOCUMENT_GETROOT (pDoc)
; display the node
dsNode = NWX_NODE_OUT_DYNSTR(pRootNode)
MLE1.TEXT = dsNode
```

```
; free the document
NWX_DOCUMENT_DISPOSE pDoc
; free the parser
NWX_PARSER_DISPOSE pParser
; free NWXML
NWX_TERMINATE
```

Résultat :

```
<AUTO><MODELE>Toyota</MODELE><ANNEE>1953</ANNEE></AUTO>
```

Voir aussi ***NWX_DOCUMENT_OUT_DYNSTR***

Gestion des éléments XML

Fonction NWX_ELEMENT_GETATTRIBUTE (Librairie NWXML)

Retourne l'attribut du nœud Element spécifié.

Syntaxe	**NWX_ELEMENT_GETATTRIBUTE** *(pElem, attrname$)*			
Paramètres	pElem	POINTER	I/O	pointeur d'élément
	attrname$	CSTRING	O	nom de l'attribut
Valeur renvoyée	POINTER sur un attribut.			

1. Utilisez NWX_ATTRIBUTE_GETVALUE% pour retourner la valeur de l'attribut.
2. Pour retourner tous les attributs d'un élément, utilisez NWX_ELEMENT_GETATTRIBUTELIST.

Exemple :

```
local pointer pParser
local pointer pDoc
local ok%(1)
local attValue$(100)
local pointer pNodeList
local pointer pAttList
local pointer pNode
local pointer pAttr
local ok2%

ok2%=NWX_INITIALIZE%

pParser = NWX_PARSER_NEW

pDoc = NWX_DOCUMENT_NEW("XMLdoc")

ok% = NWX_DOCUMENT_LOADFILE%(pDoc, EF_DOC, pParser)

pNodeList = NWX_DOCUMENT_GETELEMSBYTAGNAME(pDoc, "enregistrement")
 if pNodeList
   pNode = NWX_NODELIST_GETFIRST(pNodeList)
   pAttr = NWX_ELEMENT_GETATTRIBUTE(pNode, "format")
```

```
   ok% = NWX_ATTRIBUTE_GETVALUE% (pAttr,@attValue$,sizeof  attValue$)

   MLE1 = attValue$
endif
```

Fichier source xml :

```
<basedd_mp3>
 <enregistrement format="CD" type="rock">
  <groupe>Aha</groupe>
  <titre>Take on me</titre>
 </enregistrement>
</basedd_mp3>
```

Résultat :

```
"CD"
```

Voir aussi ***NWX_ELEMENT_SETATTRIBUTE%***, ***NWX_ELEMENT_GETATTRIBUTELIST***, ***NWX_ATTRIBUTE_GETVALUE%***

Fonction NWX_ELEMENT_GETATTRIBUTELIST (Librairie NWXML)

Retourne tous les attributs d'un élément spécifié dans un ordre arbitraire.

Syntaxe	**NWX_ELEMENT_GETATTRIBUTELIST***(pElem)*			
Paramètre	pElem	POINTER	I/O	pointeur d'élément
Valeur renvoyée	POINTER sur une liste d'attributs.			

1. Utilisez NWX_ATTRIBUTELIST_GETFIRST , NWX_ATTRIBUTELIST_GETNEXT pour interroger la liste.
2. Pour retourner un attribut, utilisez NWX_ELEMENT_GETATTRIBUTE%.
3. Il faut libérer la mémoire en utilisant la fonction NWX_ATTRIBUTELIST_DISPOSE.

Exemple :

```
local pointer pParser
local pointer pDoc
local ok%(1)
local attName$(1024)
local attValue$(1024)
local node%
local pointer pNodeList
local pointer pAttList
local pointer pNode
local pointer pAttr
local ok2%

ok2%=NWX_INITIALIZE%

pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
```

```
ok% = NWX_DOCUMENT_LOADFILE%(pDoc, EF_DOC, pParser)

pNodeList = NWX_DOCUMENT_GETELEMSBYTAGNAME(pDoc, "enregistrement")

if pNodeList
   pNode = NWX_NODELIST_GETFIRST(pNodeList)

   while pNode
     pAttList = NWX_ELEMENT_GETATTRIBUTELIST(pNode)
     pAttr = NWX_ATTRIBUTELIST_GETFIRST (pAttList)
  WHILE pAttr
       ok% =  NWX_ATTRIBUTE_GETNAME% (pAttr,@attName$,sizeof \ attName$)
       ok% =  NWX_ATTRIBUTE_GETVALUE% (pAttr,@attValue$,sizeof \ attValue$)
       INSERT AT END attName$ & "  " & attValue$ TO MLE1
       pAttr = NWX_ATTRIBUTELIST_GETNEXT (pAttList,pAttr)
  ENDWHILE

     pNode = NWX_NODELIST_GETNEXT(pNodeList,pNode)
   endwhile
endif
NWX_ATTRIBUTELIST_DISPOSE
NWX_NODELIST_DISPOSE
; dispose le document
NWX_DOCUMENT_DISPOSE pDoc
; Libère le parseur
NWX_PARSER_DISPOSE pParser
```

Fichier source xml :

```
<basedd_mp3>
 <enregistrement format="CD" type="rock">
  <groupe>Aha</groupe>
  <titre>Take on me</titre>
 </enregistrement>
 <enregistrement format="Vinyl" type="pop">
  <groupe>The Beach Boys</groupe>
  <titre>Surfin' USA</titre>
  <titre>California Girls</titre>
  <titre>Barbara-Ann</titre>
 </enregistrement>
 <enregistrement format="CD" type="classique">
  <groupe>Orchestre de Berlin</groupe>
  <titre>Les 4 saisons de Vivaldi</titre>
 </enregistrement>
</basedd_mp3>
```

Résultat :

```
type  rock
format  CD
type  pop
format  Vinyl
type  classique
format  CD
```

Voir aussi ***NWX_ELEMENT_GETATTRIBUTE*****,** ***NWX_ATTRIBUTELIST_GETFIRST*****,** ***NWX_ATTRIBUTELIST_GETNEXT***

Fonction NWX_ELEMENT_GETELEMSBYTAGNAME (Librairie NWXML)

Retourne une liste de tous les éléments parmi les descendants d'un nœud avec le nom de balise spécifié.

Syntaxe	**NWX_ELEMENT_GETELEMSBYTAGNAME** *(pElem, tagname$)*			
Paramètres	pElem	POINTER	I/O	pointeur d'élément
	tagname$	CSTRING	O	nom de la balise
Valeur renvoyée	POINTER sur une liste d'éléments.			

1. Si pElem a la même balise, l'élément fait partie de la liste renvoyée.
2. Utilisez NWX_NODELIST_GETFIRST, NWX_NODELIST_GETNEXT pour interroger la liste.
3. Pour libérer la liste, utilisez NWX_NODELIST_DISPOSE .

Exemple :

```
local pointer pParser
local pointer pDoc
local ok%(1)
local attName$(1024)
local attValue$(1024)
local name$
local node%
local pointer pNodeList
local pointer pAttList
local pointer pNode
local pointer pAttr
local pointer pElement
local ok2%

ok2%=NWX_INITIALIZE%
pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
ok% = NWX_DOCUMENT_LOADFILE%(pDoc, EF_DOC, pParser)

pElement = NWX_DOCUMENT_GETROOT (pDoc)
pNodeList = NWX_ELEMENT_GETELEMSBYTAGNAME(pElement,"enregistrement")
if pNodeList
pNode = NWX_NODELIST_GETFIRST(pNodeList)
while pNode
ok% = NWX_ELEMENT_GETTAGNAME%(pNode, @name$, sizeof name$)
INSERT AT END name$ TO MLE1
pAttList = NWX_ELEMENT_GETATTRIBUTELIST(pNode)
pAttr = NWX_ATTRIBUTELIST_GETFIRST (pAttList)
WHILE pAttr
ok% = NWX_ATTRIBUTE_GETNAME% (pAttr,@attName$,sizeof attName$)
ok% = NWX_ATTRIBUTE_GETVALUE% (pAttr,@attValue$,sizeof attValue$)
INSERT AT END " " & attName$ & " " & attValue$ TO MLE1
pAttr = NWX_ATTRIBUTELIST_GETNEXT (pAttList,pAttr)
```

```
ENDWHILE
pNode = NWX_NODELIST_GETNEXT(pNodeList,pNode)
endwhile
endif
; disposes the document
NWX_DOCUMENT_DISPOSE pDoc
; frees the parser
NWX_PARSER_DISPOSE pParser
```

Résultat :

```
enregistrement
  type  rock
  format  CD
enregistrement
  type  pop
  format  Vinyl
enregistrement
  type  classique
  format  CD
```

Voir aussi ***NWX_DOCUMENT_GETELEMSBYTAGNAME, NWX_ELEMENT_GETTAGNAME, NWX_NODELIST_GETFIRST, NWX_NODELIST_GETNEXT, NWX_NODELIST_DISPOSE***

Fonction NWX_ELEMENT_GETTAGNAME% (Librairie NWXML)

Retourne dans un buffer le nom de la balise d'un élément.

Syntaxe	**NWX_ELEMENT_GETTAGNAME%** *(pElem, pName, Size%)*			
Paramètres	pElem	POINTER	I/O	pointeur d'élément
	pName$	POINTER	I/O	pointeur du buffer
	Size%	CSTRING	O	taille du buffer
Valeur renvoyée	INT(4)			

1. Pour la taille, vous devez ajouter 1 pour la terminaison "\0"
2. Si la taille de la sortie est plus longue que la valeur spécifiée, la chaîne est tronquée. Pour pallier les problèmes de taille, utiliser plutôt la fonction NWX_ELEMENT_GETTAGNAME qui utilise une chaîne de taille variable de type DynStr pour la sortie.
3. Si pName ou Size% est égal à 0, la valeur renvoyée est la taille requise pour contenir toute la chaîne en sortie.
4. La valeur renvoyée est toujours la taille de la chaîne en sortie.
5. Il est préférable d'initialiser le buffer avant d'utiliser cette fonction.

Exemple :

```
local pointer pParser
local pointer pDoc
local ok%(1)
local attName$(1024)
local attValue$(1024)
local name$
local node%
local pointer pNodeList
local pointer pAttList
local pointer pNode
local pointer pAttr
local ok2%

ok2%=NWX_INITIALIZE%

pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
ok% = NWX_DOCUMENT_LOADFILE%(pDoc, EF_DOC, pParser)
pNodeList = \ NWX_DOCUMENT_GETELEMSBYTAGNAME(pDoc,"enregistrement")

if pNodeList
   pNode = NWX_NODELIST_GETFIRST(pNodeList)

   while pNode
     ok% = NWX_ELEMENT_GETTAGNAME%(pNode, @name$, sizeof name$)
     INSERT AT END  name$ TO MLE1
     pAttList = NWX_ELEMENT_GETATTRIBUTELIST(pNode)
     pAttr = NWX_ATTRIBUTELIST_GETFIRST (pAttList)
  WHILE pAttr
       ok% =  NWX_ATTRIBUTE_GETNAME% (pAttr,@attName$,sizeof attName$)
       ok% =  NWX_ATTRIBUTE_GETVALUE% (pAttr,@attValue$,sizeof attValue$)
       INSERT AT END  "  " & attName$ & "  " & attValue$ TO MLE1
       pAttr = NWX_ATTRIBUTELIST_GETNEXT (pAttList,pAttr)
  ENDWHILE

     pNode = NWX_NODELIST_GETNEXT(pNodeList,pNode)
   endwhile
endif
; disposes the document
NWX_DOCUMENT_DISPOSE pDoc
; frees the parser
NWX_PARSER_DISPOSE pParser
```

Fichier source :

```
<basedd_mp3>
 <enregistrement format="CD" type="rock">
  <groupe>Aha</groupe>
  <titre>Take on me</titre>
 </enregistrement>
 <enregistrement format="Vinyl" type="pop">
  <groupe>The Beach Boys</groupe>
  <titre>Surfin' USA</titre>
  <titre>California Girls</titre>
  <titre>Barbara-Ann</titre>
 </enregistrement>
 <enregistrement format="CD" type="classique">
```

```
  <groupe>Orchestre de Berlin</groupe>
  <titre>Les 4 saisons de Vivaldi</titre>
 </enregistrement>
</basedd_mp3>
```

Résultat :

```
enregistrement
  type  rock
  format  CD
enregistrement
  type  pop
  format  Vinyl
enregistrement
  type  classique
  format  CD
```

Voir aussi ***NWX_ELEMENT_GETTAGNAME*****,** ***NWX_ELEMENT_GETELEMSBYTAGNAME*****,** ***NWX_DOCUMENT_GETELEMSBYTAGNAME***

Fonction NWX_ELEMENT_GETTAGNAME (Librairie NWXML)

Retourne le nom de la balise d'un élément.

Syntaxe	**NWX_ELEMENT_GETTAGNAME%** *(pElem)*			
Paramètre	pElem	POINTER	I/O	pointeur d'élément
Valeur renvoyée	DYNSTR			

A ne pas confondre avec la fonction NWX_ELEMENT_GETTAGNAME%. Avce cette dernière, si la taille de la sortie est plus longue que la valeur spécifiée, la chaîne est tronquée. Pour pallier les problèmes de taille, utiliser plutôt la fonction NWX_ELEMENT_GETTAGNAME qui utilise une chaîne de taille variable de type DynStr pour la sortie.
Exemple :

```
local pointer pParser
local pointer pDoc
local dynstr attName$
local dynstr attValue$
local dynstr name$

local pointer pNodeList
local pointer pAttList
local pointer pNode
local pointer pAttr
local ret%
ret% = NWX_INITIALIZE%
IF ret% <> 0
EXIT
ENDIF
pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_GETFROMFILE(EF_DOC, pParser)
pNodeList = NWX_DOCUMENT_GETELEMSBYTAGNAME(pDoc,"enregistrement")
```

```
IF pNodeList
pNode = NWX_NODELIST_GETFIRST(pNodeList)
WHILE pNode
name$ = NWX_ELEMENT_GETTAGNAME(pNode)
INSERT AT END name$ TO MLE1
pAttList = NWX_ELEMENT_GETATTRIBUTELIST(pNode)
pAttr = NWX_ATTRIBUTELIST_GETFIRST(pAttList)
WHILE pAttr
attName$ =  NWX_ATTRIBUTE_GETNAME(pAttr)
attValue$ =  NWX_ATTRIBUTE_GETVALUE(pAttr)
INSERT AT END  "  " & attName$ & "  " & attValue$ TO MLE1
pAttr = NWX_ATTRIBUTELIST_GETNEXT (pAttList,pAttr)
ENDWHILE
pNode = NWX_NODELIST_GETNEXT(pNodeList,pNode)
ENDWHILE
ENDIF
; disposes the document
NWX_DOCUMENT_DISPOSE pDoc
; frees the parser
NWX_PARSER_DISPOSE pParser

Fichier source :
<basedd_mp3>
 <enregistrement format="CD" type="rock">
  <groupe>Aha</groupe>
  <titre>Take on me</titre>
 </enregistrement>
 <enregistrement format="Vinyl" type="pop">
  <groupe>The Beach Boys</groupe>
  <titre>Surfin' USA</titre>
  <titre>California Girls</titre>
  <titre>Barbara-Ann</titre>
 </enregistrement>
 <enregistrement format="CD" type="classique">
  <groupe>Orchestre de Berlin</groupe>
  <titre>Les 4 saisons de Vivaldi</titre>
 </enregistrement>

</basedd_mp3>
Résultat :
enregistrement
  type  rock
  format  CD
enregistrement
  type  pop
  format  Vinyl
enregistrement
  type  classique
  format  CD
```

Voir aussi ***NWX_ELEMENT_GETTAGNAME%*****,** ***NWX_ELEMENT_GETELEMSBYTAGNAME*****,** ***NWX_DOCUMENT_GETELEMSBYTAGNAME***

Fonction NWX_ELEMENT_NEW (Librairie NWXML)

Crée un nouvel élément.

Syntaxe	**NWX_ELEMENT_NEW** *(pDoc, Elename$)*			
Paramètres	pElem	POINTER	I/O	pointeur d'élément
	Elename$	CSTRING	I	nom de l'élément
Valeur renvoyée	POINTER sur le nœud.			

Utiliser NWX_NODE_ATTACHBEFORE% ou NWX_NODE_ATTACHAFTER% pour attacher l'élément créé.

Exemple :

```
local pointer pDoc
local pointer pParser
local sValue$
local ok%
local pointer pNewElement
local pointer pRootNode
local pointer pNode
local ok2%

ok2%=NWX_INITIALIZE%

pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")

move "<AUTO><MODELE>Toyota</MODELE><ANNEE>1951</ANNEE></AUTO>"  to sValue$

pParser = NWX_PARSER_NEW

pDoc = NWX_DOCUMENT_NEW("XMLdoc")

ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)
pRootNode = NWX_DOCUMENT_GETROOT (pDoc)

pNode = NWX_NODE_GETCHILD (pRootNode)
pNode = NWX_NODE_GETNEXT (pNode)

pNewElement = NWX_ELEMENT_NEW (pDoc, "TYPE")

ok%=NWX_NODE_SETVALUE% (pNewElement,"Berline")

ok%= NWX_NODE_ATTACHBEFORE% (pNode,pNewElement)

ok% = NWX_DOCUMENT_OUTPUTFILE%(pDoc, EF_DOC2)

ok% = NWX_DOCUMENT_OUTPUT% (pDoc,@sValue$,sizeof sValue$)
delete from MLE1
INSERT AT END  sValue$ TO MLE1
; dispose le document
NWX_DOCUMENT_DISPOSE pDoc
; Libère le parseur
NWX_PARSER_DISPOSE pParser
```

Résultat :

```
<AUTO>
<MODELE>Toyota</MODELE>
<TYPE>Berline</TYPE>
<ANNEE>1951</ANNEE>
</AUTO>
```

Voir aussi ***NWX_NODE_NEW*** ***,NWX_NODE_ATTACHCHILD%,*** ***NWX_NODE_ATTACHBEFORE%,*** ***NWX_NODE_ATTACHAFTER%,*** ***NWX_DOCUMENT_NEW***

Fonction NWX_ELEMENT_SETATTRIBUTE% (Librairie NWXML)

Permet de créer un attribut pour un nœud Element.

Syntaxe	**NWX_ELEMENT_SETATTRIBUTE%** *(pElem, sAttrName, sAttrValue)*			
Paramètres	pDoc	POINTER	I/O	pointeur de l'élément
	sAttrName	CSTRING	I	nom de l'attribut
	sAttrValue	CSTRING	I	valeur de l'attribut
Valeur renvoyée	INT(1) TRUE% ou FALSE%.			

1. La valeur renvoyée est FALSE% lorsque :
   a) Un des paramètres en entrée est null.
   b) Le nom de l'attribut contient des caractères spéciaux (&, <,> …)
   c) Le nœud Element est en lecture seule
2. Si l'attribut existe déjà pour cet élément alors la valeur de celui-ci est remplacée par la nouvelle valeur.

Exemple :

```
local pointer pDoc,pointer pParser
local ok%(1)
local buffer$, sValue$
local pointer pRootNode, pointer pElem
local ok2%

ok2%=NWX_INITIALIZE%

pDoc = NWX_DOCUMENT_NEW("XMLdoc")
pParser = NWX_PARSER_NEW

sValue$ = "<DIRECTORY><CUSTOMER>Smith</CUSTOMER></DIRECTORY>"

ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)
```

```
;Aller chercher le noeud customer
pRootNode =NWX_DOCUMENT_GETROOT (pDoc)
pElem = NWX_NODE_GETCHILD (pRootNode)
;Ajout d'un nouvel attribut à l'élément customer
ok% = NWX_ELEMENT_SETATTRIBUTE% (pElem,"number","1")
;Le nœud customer aura donc l'attribut number avec comme ;valeur 1
<CUSTOMER number ="1" >
```

Voir aussi ***NWX_ELEMENT_GETATTRIBUTE***, ***NWX_ELEMENT_GETATTRIBUTELIST***, ***NWX_ATTRIBUTE_GETVALUE%***

Gestion des attributs XML

Fonction NWX_ATTRIBUTE_ATTACH% (Librairie NWXML)

Attache un attribut à un élément.

Syntaxe	**NWX_ATTRIBUTE_ATTACH%** *(pElem, pAttr)*			
Paramètres	pElem	POINTER	I/O	pointeur d'élément
	pAttr	POINTER	I/O	pointeur de l'attribut
Valeur renvoyée	INT(1)			

Pour créer un élément, utilisez NWX_ATTRIBUTE_NEW.

Exemple

```
local pointer pDoc
local pointer pParser
local sValue$
local attName$
local attValue$
local ok%
local pointer pAttList
local pointer pNode
local pointer pAttr
local pointer pAttrNew
local pointer pNodeList
local ok2%

ok2%=NWX_INITIALIZE%

delete from MLE1

pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
move
'<AUTO><MODELE>Toyota</MODELE><MODELE>Peugeot</MODELE><MODELE>Jaguar</MODEL
E></AUTO>' to sValue$

ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)
```

```
pNode = NWX_DOCUMENT_GETROOT (pDoc)
pNodeList = NWX_DOCUMENT_GETELEMSBYTAGNAME(pDoc, "MODELE")
; First <MODELE>  (Toyota)
pNode = NWX_NODELIST_GETFIRST(pNodeList)

;Ajout d'un nouvel attribut à l'élément
pAttrNew= NWX_ATTRIBUTE_NEW(pDoc, "Annee")
ok%= NWX_ATTRIBUTE_ATTACH%(pNode, pAttrNew)
ok%= NWX_ATTRIBUTE_SETVALUE% (pAttrNew, "2001")
pAttrNew= NWX_ATTRIBUTE_NEW(pDoc, "Type")
ok%= NWX_ATTRIBUTE_ATTACH%(pNode, pAttrNew)
ok%= NWX_ATTRIBUTE_SETVALUE% (pAttrNew, "Berline")

;Liste des attributs de Toyota
pAttList = NWX_ELEMENT_GETATTRIBUTELIST(pNode)
pAttr = NWX_ATTRIBUTELIST_GETFIRST (pAttList)
pAttr = NWX_ATTRIBUTELIST_GETNEXT (pAttList,pAttr)
ok% =  NWX_ATTRIBUTE_GETNAME% (pAttr,@attName$,sizeof attName$) ;Année
ok% =  NWX_ATTRIBUTE_GETVALUE% (pAttr,@attValue$,sizeof \ attValue$);2001

INSERT AT END attName$ & "  " & attValue$ TO MLE1;Annee 2001
INSERT AT END  "  " TO MLE1; ligne blanche

ok% = NWX_DOCUMENT_OUTPUT% (pDoc,@sValue$,sizeof sValue$)
INSERT AT END  sValue$ TO MLE1
; dispose Attribute list
NWX_NODELIST_DISPOSE pAttList
; dispose Node list
NWX_NODELIST_DISPOSE pNodeList
; dispose le document
NWX_DOCUMENT_DISPOSE pDoc
; Libère le parseur
NWX_PARSER_DISPOSE pParser
```

Résultat :

```
Annee 2001
<AUTO><MODELE Type = "Berline" Annee = "2001">Toyota</MODELE>
<MODELE>Peugeot</MODELE><MODELE>Jaguar</MODELE></AUTO>
```

Voir aussi ***NWX_ATTRIBUTE_DETACH%***

Fonction NWX_ATTRIBUTE_DETACH% (Librairie NWXML)

Enlève l'attribut spécifié d'un élément.

Syntaxe	**NWX_ATTRIBUTE_DETACH%** *(pElem, pAttr)*			
Paramètres	pElem	POINTER	I/O	pointeur d'élément
	pAttr	POINTER	I/O	pointeur de l'attribut

Valeur renvoyée	INT(1)

Cette fonction retire l'attribut, mais ne le libère pas. Pour libérer un attribut, utilisez NWX_ATTRIBUTE_DISPOSE

Exemple :

```
; enlever l'attribut "type"
local pointer pAttrList
local pointer pAttr
local name$
local ok%

ok%=NWX_INITIALIZE%

pAttrList = NWX_ELEMENT_GETATTRIBUTELIST(pElemBook)
if pAttrList
 pAttr = NWX_ATTRIBUTELIST_GETFIRST(pAttrList)
 while pAttr
  NWX_ATTRIBUTE_GETNAME(pAttr, @name$, sizeof name$)
   if name = "type" NWX_ATTRIBUTE_DETACH%(pElem, pAttr)
   else
    pAttr = \ NWX_ATTRIBUTELIST_GETNEXT(pAttrList, pAttr)
   endif
 endwhile
NWX_ATTRIBUTELIST_DISPOSE pAttrList
endif
```

Voir aussi NWX_ATTRIBUTE_ATTACH%, NWX_ATTRIBUTE_DISPOSE

Instruction NWX_ATTRIBUTE_DISPOSE (Librairie NWXML)

Libère l'attribut.

Syntaxe	**NWX_ATTRIBUTE_DISPOSE** *pAttr*			
Paramètre	pAttr	POINTER	I/O	pointeur de l'attribut

L'attribut doit être libéré d'un élément. Pour libérer les attributs, utilisez NWX_ATTRIBUTE_DISPOSE.

Exemple :

```
; enlever et libérer un attribut "type"
local pointer pAttrList
local pointer pAttr
local name$
local ok%

ok%=NWX_INITIALIZE%

pAttrList = NWX_ELEMENT_GETATTRIBUTELIST(pElemBook)
if pAttrList
 pAttr = NWX_ATTRIBUTELIST_GETFIRST(pAttrList)
```

```
 while pAttr
  NWX_ATTRIBUTE_GETNAME%(pAttr, @name$, sizeof name$)
   if name = "type"
    if NWX_ATTRIBUTE_DETACH%(pElem, pAttr)
     NWX_ATTRIBUTE_DISPOSE pAttr
    endif
    break
   else
    pAttr = NWX_ATTRIBUTELIST_GETNEXT(pAttrList, pAttr)
   endif
 endwhile
endif
```

Voir aussi ***NWX_ATTRIBUTE_DETACH%***

Fonction NWX_ATTRIBUTE_GETNAME% (Librairie NWXML)

Retourne le nom d'un attribut.

Syntaxe	**NWX_ATTRIBUTE_GETNAME%** *(pAttr, pName, Size%)*			
Paramètres	pAttr	POINTER	I/O	pointeur de l'attribut
	pName	POINTER	I/O	pointeur du buffer
	Size%	INT(4)	I/O	taille du buffer
Valeur renvoyée	INT(4)			

1. Pour la taille, vous devez ajouter 1 pour la terminaison "\0"
2. Si la taille de la sortie est plus longue que la valeur spécifiée, la chaîne est tronquée
3. Si pName ou Size% est égal à 0, la valeur renvoyée est la taille requise pour contenir toute la chaîne en sortie.
4. La valeur renvoyée est toujours la taille de la chaîne en sortie.

Exemple :

```
local pointer pDoc
local pointer pParser
local sValue$
local attName$
local attValue$
local ok%
local pointer pAttList
local pointer pNode
local pointer pAttr
local pointer pAttrNew
local pointer pNodeList
local ok2%
```

```
ok2%=NWX_INITIALIZE%

delete from MLE1

pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
move
'<AUTO><MODELE>Toyota</MODELE><MODELE>Peugeot</MODELE><MODELE>Jaguar</MODEL
E></AUTO>' to sValue$

ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)

pNode = NWX_DOCUMENT_GETROOT (pDoc)
pNodeList = NWX_DOCUMENT_GETELEMSBYTAGNAME(pDoc, "MODELE")
; First <MODELE>  (Toyota)
pNode = NWX_NODELIST_GETFIRST(pNodeList)

;Ajout d'un nouvel attribut à l'élément
pAttrNew= NWX_ATTRIBUTE_NEW(pDoc, "Annee")
ok%= NWX_ATTRIBUTE_ATTACH%(pNode, pAttrNew)
ok%= NWX_ATTRIBUTE_SETVALUE% (pAttrNew, "2001")
pAttrNew= NWX_ATTRIBUTE_NEW(pDoc, "Type")
ok%= NWX_ATTRIBUTE_ATTACH%(pNode, pAttrNew)
ok%= NWX_ATTRIBUTE_SETVALUE% (pAttrNew, "Berline")

;Liste des attributs de Toyota
pAttList = NWX_ELEMENT_GETATTRIBUTELIST(pNode)
pAttr = NWX_ATTRIBUTELIST_GETFIRST (pAttList)
pAttr = NWX_ATTRIBUTELIST_GETNEXT (pAttList,pAttr)
ok% =  NWX_ATTRIBUTE_GETNAME% (pAttr,@attName$,sizeof attName$) ;Année
ok% =  NWX_ATTRIBUTE_GETVALUE% (pAttr,@attValue$,sizeof \ attValue$);2001

INSERT AT END attName$ & "  " & attValue$ TO MLE1;Annee 2001
INSERT AT END  "  " TO MLE1; ligne blanche

ok% = NWX_DOCUMENT_OUTPUT% (pDoc,@sValue$,sizeof sValue$)
INSERT AT END  sValue$ TO MLE1

; dispose Attribute list
NWX_ATTRIBUTELIST_DISPOSE pAttList

; dispose Node list
NWX_NODELIST_DISPOSE pNodeList
; dispose le document
NWX_DOCUMENT_DISPOSE pDoc
; Libère le parseur
NWX_PARSER_DISPOSE pParser
```

Résultat :

```
Annee 2001
<AUTO><MODELE Type = "Berline" Annee = "2001">Toyota</MODELE>
<MODELE>Peugeot</MODELE><MODELE>Jaguar</MODELE></AUTO>
```

Voir aussi NWX_ATTRIBUTE_NEW

Fonction NWX_ATTRIBUTE_GETNAME (Librairie NWXML)

Retourne le nom d'un attribut.

Syntaxe	**NWX_ATTRIBUTE_GETNAME** *(pAttr)*			
Paramètre	pAttr	POINTER	I/O	pointeur de l'attribut
Valeur renvoyée	DYNSTR			

La fonction NWX_ATTRIBUTE_GETNAME permet de renvoyer des chaînes de type DYNSTR.

Voir aussi ***NWX_ATTRIBUTE_GETNAME%*****,** ***NWX_ATTRIBUTE_NEW***

Fonction NWX_ATTRIBUTE_GETVALUE% (Librairie NWXML)

Retourne la valeur d'un attribut.

Syntaxe	**NWX_ATTRIBUTE_GETVALUE%** *(pAttr, pValue, Size%)*			
Paramètres	pAttr	POINTER	I/O	pointeur de l'attribut
	pValue	POINTER	I/O	pointeur de l'attribut value
	Size%	INT(4)	I/O	taille du buffer
Valeur renvoyée	INT(4)			

1. Pour la taille, vous devez ajouter 1 pour la terminaison "\0"
2. Si la taille de la sortie est plus longue que la valeur spécifiée, la chaîne est tronquée
3. Si pName ou Size% est égal à 0, la valeur renvoyée est la taille requise pour contenir toute la chaîne en sortie.
4. La valeur renvoyée est toujours la taille de la chaîne en sortie.

Exemple :

```
local pointer pDoc
local pointer pParser
local sValue$
local attName$
local attValue$
local ok%
local pointer pAttList
local pointer pNode
```

```
local pointer pAttr
local pointer pAttrNew
local pointer pNodeList
local ok2%

ok2%=NWX_INITIALIZE%

delete from MLE1

pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
move '<AUTO><MODELE>Toyota</MODELE><MODELE>Peugeot</MODELE>
<MODELE>Jaguar</MODELE></AUTO>' to sValue$

ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)

pNode = NWX_DOCUMENT_GETROOT (pDoc)
pNodeList = NWX_DOCUMENT_GETELEMSBYTAGNAME(pDoc, "MODELE")
;First <MODELE>  (Toyota)
pNode = NWX_NODELIST_GETFIRST(pNodeList)

;Ajout d'un nouvel attribut à l'élément
pAttrNew= NWX_ATTRIBUTE_NEW(pDoc, "Annee")
ok%= NWX_ATTRIBUTE_ATTACH%(pNode, pAttrNew)
ok%= NWX_ATTRIBUTE_SETVALUE% (pAttrNew, "2001")
pAttrNew= NWX_ATTRIBUTE_NEW(pDoc, "Type")
ok%= NWX_ATTRIBUTE_ATTACH%(pNode, pAttrNew)
ok%= NWX_ATTRIBUTE_SETVALUE% (pAttrNew, "Berline")

;Liste des attributs de Toyota
pAttList = NWX_ELEMENT_GETATTRIBUTELIST(pNode)
pAttr = NWX_ATTRIBUTELIST_GETFIRST (pAttList)
pAttr = NWX_ATTRIBUTELIST_GETNEXT (pAttList,pAttr)
ok% =  NWX_ATTRIBUTE_GETNAME% (pAttr,@attName$,sizeof attName$) ;Année
ok% =  NWX_ATTRIBUTE_GETVALUE% (pAttr,@attValue$,sizeof \ attValue$);2001

INSERT AT END attName$ & "  " & attValue$ TO MLE1;Annee 2001
INSERT AT END  "  " TO MLE1; ligne blanche

ok% = NWX_DOCUMENT_OUTPUT% (pDoc,@sValue$,sizeof sValue$)
INSERT AT END  sValue$ TO MLE1

; dispose Attribute list
NWX_ATTRIBUTELIST_DISPOSE pAttList
; dispose Node list
NWX_NODELIST_DISPOSE pNodeList
; dispose le document
NWX_DOCUMENT_DISPOSE pDoc
; Libère le parseur
NWX_PARSER_DISPOSE pParser
```

Résultat :

```
Annee 2001
<AUTO><MODELE Type = "Berline" Annee = "2001">Toyota</MODELE>
<MODELE>Peugeot</MODELE><MODELE>Jaguar</MODELE></AUTO>
```

Voir aussi ***NWX_ATTRIBUTE_SETVALUE%***

Fonction NWX_ATTRIBUTE_GETVALUE (Librairie NWXML)

Retourne la valeur d'un attribut.

Syntaxe	**NWX_ATTRIBUTE_GETVALUE** *(pAttr)*			
Paramètre	pAttr	POINTER	I/O	pointeur de l'attribut
Valeur renvoyée	DYNSTR			

La fonction NWX_ATTRIBUTE_GETVALUE permet de renvoyer des chaînes de type DYNSTR.

Voir aussi ***NWX_ATTRIBUTE_GETVALUE%***, ***NWX_ATTRIBUTE_SETVALUE%***

Instruction NWX_ATTRIBUTELIST_DISPOSE (Librairie NWXML)

Libère la liste d'attributs spécifiée.

Syntaxe	**NWX_ATTRIBUTELIST_DISPOSE** *pNode*			
Paramètre	pNode	POINTER	I/O	pointeur du nœud

Si le nœud n'est pas détaché, cette instruction le détache puis le libère.

Exemple :

```
; enlever l'attribut "type"
local pointer pAttrList
local pointer pAttr
local name$
local ok%

ok%=NWX_INITIALIZE%

pAttrList = NWX_ELEMENT_GETATTRIBUTELIST(pElemBook)
if pAttrList
 pAttr = NWX_ATTRIBUTELIST_GETFIRST(pAttrList)
 while pAttr
  NWX_ATTRIBUTE_GETNAME(pAttr, @name$, sizeof name$)
   if name = "type" NWX_ATTRIBUTE_DETACH%(pElem, \ pAttr)
   else
    pAttr = \ NWX_ATTRIBUTELIST_GETNEXT(pAttrList, pAttr)
   endif
 endwhile
NWX_ATTRIBUTELIST_DISPOSE pAttrList
endif
```

Voir aussi ***NWX_ATTRIBUTE_DETACH%***

Fonction NWX_ATTRIBUTE_NEW (Librairie NWXML)

Crée un attribut avec le nom spécifié qu'il faudra attacher à un élément.

Syntaxe	**NWX_ATTRIBUTE_NEW** *(pDoc, AttrName$)*			
Paramètres	pDoc	POINTER	I/O	pointeur du document
	AttrName$	CSTRING	I/O	nom de l'attribut
Valeur renvoyée	POINTER			

Pour associer un attribut à un élément, utilisez NWX_ATTRIBUTE_ATTACH% .
Exemple :

```
local pointer pDoc
local pointer pParser
local sValue$
local attName$
local attValue$
local ok%
local pointer pAttList
local pointer pNode
local pointer pAttr
local pointer pAttrNew
local pointer pNodeList
local ok2%

ok2%=NWX_INITIALIZE%
delete from MLE1

pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
move '<AUTO><MODELE>Toyota</MODELE><MODELE>Peugeot</MODELE>\
<MODELE>Jaguar</MODELE></AUTO>' to sValue$

ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)

pNode = NWX_DOCUMENT_GETROOT (pDoc)
pNodeList = NWX_DOCUMENT_GETELEMSBYTAGNAME(pDoc, "MODELE")
; First <MODELE>  (Toyota)
pNode = NWX_NODELIST_GETFIRST(pNodeList)

;Ajout d'un nouvel attribut à l'élément
pAttrNew= NWX_ATTRIBUTE_NEW(pDoc, "Annee")
ok%= NWX_ATTRIBUTE_ATTACH%(pNode, pAttrNew)
ok%= NWX_ATTRIBUTE_SETVALUE% (pAttrNew, "2001")
pAttrNew= NWX_ATTRIBUTE_NEW(pDoc, "Type")
ok%= NWX_ATTRIBUTE_ATTACH%(pNode, pAttrNew)
ok%= NWX_ATTRIBUTE_SETVALUE% (pAttrNew, "Berline")
;Liste des attributs de Toyota
pAttList = NWX_ELEMENT_GETATTRIBUTELIST(pNode)
pAttr = NWX_ATTRIBUTELIST_GETFIRST (pAttList)
pAttr = NWX_ATTRIBUTELIST_GETNEXT (pAttList,pAttr)
ok% =  NWX_ATTRIBUTE_GETNAME% (pAttr,@attName$,sizeof attName$) ; Année
```

```
ok% =  NWX_ATTRIBUTE_GETVALUE% (pAttr,@attValue$,sizeof attValue$);2001

INSERT AT END attName$ & "  " & attValue$ TO MLE1 ; Année 2001
INSERT AT END  "  " TO MLE1 ; ligne blanche

ok% = NWX_DOCUMENT_OUTPUT% (pDoc,@sValue$,sizeof sValue$)
INSERT AT END  sValue$ TO MLE1
; dispose Attribute list
NWX_ATTRIBUTELIST_DISPOSE pAttList
; dispose Node list
NWX_NODELIST_DISPOSE pNodeList
; dispose le document
NWX_DOCUMENT_DISPOSE pDoc
; Libère le parseur
NWX_PARSER_DISPOSE pParser
```

Résultat :

```
Annee 2001
<AUTO><MODELE Type = "Berline" Annee = "2001">Toyota</MODELE>
<MODELE>Peugeot</MODELE><MODELE>Jaguar</MODELE></AUTO>
```

Voir aussi ***NWX_ATTRIBUTE_GETNAME%***

Fonction NWX_ATTRIBUTE_SETVALUE% (Librairie NWXML)

Définit la valeur d'un attribut.

Syntaxe	**NWX_ATTRIBUTE_SETVALUE%** *(pAttr, Value$)*			
Paramètres	pAttr	POINTER	I/O	pointeur de l'attribut
	Value$	CSTRING	O	valeur
Valeur renvoyée	INT(1)			

1. Si l'attribut a déjà une valeur, la valeur devient la valeur définie par Value$.
2. La taille de Value$ peut être supérieur à 255 caractères. Utilisez la terminaison "\0".

Exemple :

```
local pointer pDoc
local pointer pParser
local sValue$
local attName$
local attValue$
local ok%
local pointer pAttList
local pointer pNode
local pointer pAttr
local pointer pAttrNew
local pointer pNodeList
local ok2%
```

```
ok2%=NWX_INITIALIZE%

delete from MLE1

pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
move '<AUTO><MODELE>Toyota</MODELE><MODELE>Peugeot</MODELE>\
<MODELE>Jaguar</MODELE></AUTO>' to sValue$

ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)

pNode = NWX_DOCUMENT_GETROOT (pDoc)
pNodeList = NWX_DOCUMENT_GETELEMSBYTAGNAME(pDoc, "MODELE")
; First <MODELE>  (Toyota)
pNode = NWX_NODELIST_GETFIRST(pNodeList)

;Ajout d'un nouvel attribut à l'élément
pAttrNew= NWX_ATTRIBUTE_NEW(pDoc, "Annee")
ok%= NWX_ATTRIBUTE_ATTACH%(pNode, pAttrNew)
ok%= NWX_ATTRIBUTE_SETVALUE% (pAttrNew, "2001")
pAttrNew= NWX_ATTRIBUTE_NEW(pDoc, "Type")
ok%= NWX_ATTRIBUTE_ATTACH%(pNode, pAttrNew)
ok%= NWX_ATTRIBUTE_SETVALUE% (pAttrNew, "Berline")

;Liste des attributs de Toyota
pAttList = NWX_ELEMENT_GETATTRIBUTELIST(pNode)
pAttr = NWX_ATTRIBUTELIST_GETFIRST (pAttList)
pAttr = NWX_ATTRIBUTELIST_GETNEXT (pAttList,pAttr)
ok% =  NWX_ATTRIBUTE_GETNAME% (pAttr,@attName$,sizeof attName$) ;Année
ok% =  NWX_ATTRIBUTE_GETVALUE% (pAttr,@attValue$,sizeof \ attValue$);2001

INSERT AT END attName$ & "  " & attValue$ TO MLE1 ;Année 2001
INSERT AT END  "  " TO MLE1 ; ligne blanche

ok% = NWX_DOCUMENT_OUTPUT% (pDoc,@sValue$,sizeof sValue$)
INSERT AT END  sValue$ TO MLE1
; dispose Attribute list
NWX_NODELIST_DISPOSE pAttList
; dispose Node list
NWX_NODELIST_DISPOSE pNodeList
; dispose le document
NWX_DOCUMENT_DISPOSE pDoc
; Libère le parseur
NWX_PARSER_DISPOSE pParser
```

Résultat :

```
Annee 2001
<AUTO><MODELE Type = "Berline" Annee = "2001">Toyota</MODELE>
<MODELE>Peugeot</MODELE><MODELE>Jaguar</MODELE></AUTO>
```

Voir aussi NWX_ATTRIBUTE_GETVALUE%

Fonction NWX_ATTRIBUTELIST_GETFIRST (Librairie NWXML)

Retourne le premier attribut d'une liste d'attribut.

Syntaxe	**NWX_ATTRIBUTELIST_GETFIRST** *(pAttrList)*			
Paramètre	pAttrList	POINTER	I/O	pointeur de liste
Valeur renvoyée	POINTER sur un nœud Attribut.			

Utilisez NWX_ELEMENT_GETATTRIBUTELIST pour créer une liste d'attributs.
Exemple :

```
local pointer pAttrList
local pointer pAttr
local ok%

ok%=NWX_INITIALIZE%

pAttrList = NWX_ELEMENT_GETATTRIBUTELIST(pElemBook)
 if pAttrList
  pAttr = NWX_ATTRIBUTELIST_GETFIRST(pAttrList)
  while pAttr
   ;faire qqch avec pAttr
   pAttr = NWX_ATTRIBUTELIST_GETNEXT(pAttrList, pAttr)
  endwhile
 endif
```

Voir aussi NWX_ATTRIBUTELIST_GETNEXT, NWX_ELEMENT_GETATTRIBUTELIST

Fonction NWX_ATTRIBUTELIST_GETNEXT (Librairie NWXML)

Retourne l'attribut suivant sur la liste.

Syntaxe	**NWX_ATTRIBUTELIST_GETNEXT** *(pAttrList, pAttr)*			
Paramètres	pAttrList	POINTER	I/O	pointeur de liste
	pAttr	POINTER	I/O	pointeur de l'attribut
Valeur renvoyée	POINTER			

Utilisez NWX_ELEMENT_GETATTRIBUTELIST pour créer une liste d'attributs.
Exemple :

```
local pointer pAttrList
local pointer pAttr
local ok%

ok%=NWX_INITIALIZE%

pAttrList = NWX_ELEMENT_GETATTRIBUTELIST(pElemBook)
```

```
if pAttrList
 pAttr = NWX_ATTRIBUTELIST_GETFIRST(pAttrList)
 while pAttr
  ;faire qqch avec pAttr
  pAttr = NWX_ATTRIBUTELIST_GETNEXT(pAttrList, pAttr)
 endwhile
endif
```

***Voir aussi** NWX_ATTRIBUTELIST_GETFIRST, NWX_ELEMENT_GETATTRIBUTELIST*

Processus XSLT

Fonction NWX_XSLTRANSFORM (Librairie NWXML)

La fonction NWX_XSLTRANSFORM permet de transformer un document DOM en un autre document DOM en utilisant une feuille XSL.

Syntaxe	**NWX_XSLTRANSFORM** *(pDocXML, pDocXSL)*			
Paramètres	pDocXML	POINTER	I	pointeur du fichier XML
	pDocXSL	POINTER	I	pointeur du fichier XSL
Valeur renvoyée	POINTER vers le document DOM obtenu.			

1. L'entête de la feuille de style devrait être :

```
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"> et non pas
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/TR/WD-xsl" version="1.0">
```

2. Il est important d'utiliser comme espace de nommage l'adresse du W3C pour la recommandation candidate ainsi que d'indiquer le numéro de la version.

Exemple :

```
Local pointer pParser
Local pointer pXML
Local pointer pXSL
Local pointer pResultDoc
Local ok%
Local ok2%

ok2%=NWX_INITIALIZE%

pParser = NWX_PARSER_NEW
pXML = NWX_DOCUMENT_NEW(« xmldoc »)
```

```
ok% = NWX_DOCUMENT_LOADFILE(pXML, « catalogue.xml », pParser)

pXSL = NWX_DOCUMENT_NEW(« xsldoc »)
ok% = NWX_DOCUMENT_LOADFILE(pXSL, « catalogue.xsl », pParser)

pResultDoc = NWX_XSLTRANSFORM (pXML, pXSL)
; exporter le résultat dans un fichier ou un buffer en
; utilisant NWX_DOCUMENT_OUTPUT ou NWX_DOCUMENT_OUTPUTFILE%
ok% = NWX_DOCUMENT_OUTPUTFILE%(pResultDoc, « catalogue.xml »)
; voir fichiers catalogue.xml et catalogue.xsl dans la fonction
NWX_XSLTRANSFORMFILE
```

Voir aussi ***NWX_XSLTRANSFORMFILE*****,** ***NWX_XSLTRANSFORM_ERRMSG*****,** ***NWX_XSLTRANSFORMFILE_ERRMSG*****,** ***NWX_XSLTRANSFORMFILESTODOM***

Instruction NWX_XSLTRANSFORMFILE (Librairie NWXML)

L'instruction NWX_XSLTRANSFORMFILE permet de transformer un fichier XML en un autre fichier (XML, HTML, texte …) suivant la feuille de style XSL.

Syntaxe	**NWX_XSLTRANSFORMFILE** *xmlfile, xslfile, outfile*			
Paramètres	xmlfile	CSTRING	I	fichier XML
	xslfile	CSTRING	I	fichier XSL
	outfile	CSTRING	O	fichier de sortie

1. L'entête de la feuille de style devrait être :

```
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"> et non pas
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/TR/WD-xsl" version="1.0">
```

2. Il est important d'utiliser comme espace de nommage l'adresse du W3C pour la recommandation candidate ainsi que d'indiquer le numéro de la version.

Exemple :

```
local ok2%

ok2%=NWX_INITIALIZE%
; transformer le fichier catalogue.xml en catalogue.html en
; utilisant catalogue.xsl
NWX_XSLTRANSFORMFILE(« CATALOGUE.xml », « CATALOGUE.xsl », \
« CATALOGUE.html »)
Fichier CATALOGUE.XML :
<?xml version="1.0" encoding= "ISO-8859-1" standalone = "yes"?>
<REPERTOIRE>
<VOITURE>
<MARQUE>HONDA</MARQUE>
```

```
<MODELE>CIVIC</MODELE>
<COULEUR>ROUGE</COULEUR>
<PORTES>4</PORTES>
<OPTIONS ToitOuvrant="Oui" Climatiseur="Non" Automatique="Non"/>
</VOITURE>
<VOITURE>
<MARQUE>FORD</MARQUE>
<MODELE>ESCORT</MODELE>
<COULEUR>BLEUE</COULEUR>
<PORTES>2</PORTES>
<OPTIONS ToitOuvrant="Non" Climatiseur="Non" Automatique="Oui"/>
</VOITURE>
</REPERTOIRE>
Fichier CATALOGUE.XSL :
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:template match="/">
<html>
<body>
<table border="2" >
<tr>
<th>MARQUE</th>
<th>MODELE</th>
<th>COULEUR</th>
<th>PORTES</th>
<th>TOIT OUVRANT</th>
<th>CLIMATISEUR</th>
<th>AUTOMATIQUE</th>
</tr>
<xsl:for-each select="REPERTOIRE/VOITURE">
<tr>
<td><xsl:value-of select="MARQUE"/></td>
<td><xsl:value-of select="MODELE"/></td>
<td><xsl:value-of select="COULEUR"/></td>
<td><xsl:value-of select="PORTES"/></td>
<td><xsl:value-of select="OPTIONS/@ToitOuvrant"/></td>
<td><xsl:value-of select="OPTIONS/@Climatiseur"/></td>
<td><xsl:value-of select="OPTIONS/@Automatique"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
Fichier résultat :
<html>
<body>
<table border="2">
<tr>
<th>MARQUE</th>
<th>MODELE</th>
<th>COULEUR</th>
<th>PORTES</th>
<th>TOIT OUVRANT</th>
<th>CLIMATISEUR</th>
<th>AUTOMATIQUE</th>
```

```
</tr>
<tr>
<td>HONDA</td>
<td>CIVIC</td>
<td>ROUGE</td>
<td>4</td>
<td>Oui</td>
<td>Non</td>
<td>Non</td>
</tr>
<tr>
<td>FORD</td>
<td>ESCORT</td>
<td>BLEUE</td>
<td>2</td>
<td>Non</td>
<td>Non</td>
<td>Oui</td>
</tr>
</table>
</body>
</html>
```

***Voir aussi** NWX_XSLTRANSFORM , NWX_XSLTRANSFORMFILESTODOM*

Fonction NWX_XSLTRANSFORM_ERRMSG (Librairie NWXML)

La fonction NWX_XSLTRANSFORM_ERRMSG permet de transformer un document DOM en un autre document DOM en utilisant une feuille XSL.

Syntaxe	**NWX_XSLTRANSFORM** *(pDocXML, pDocXSL, pDocXMLOut)*			
Paramètres	pDocXML	POINTER	I	pointeur du fichier XML
	pDocXSL	POINTER	I	pointeur du fichier XSL
	pDocXMLOut	POINTER	O	pointeur vers le document DOM obtenu.
Valeur renvoyée	DynStr vide si pas d'erreur dans la transformation du document, sinon le texte de l'erreur			

1. L'entête de la feuille de style devrait être :

```
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"> et non pas
```

```
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/TR/WD-xsl" version="1.0">
```

2. Il est important d'utiliser comme espace de nommage l'adresse du W3C pour la recommandation candidate ainsi que d'indiquer le numéro de la version.

Voir aussi ***NWX_XSLTRANSFORMFILE***, ***NWX_XSLTRANSFORM***, ***NWX_XSLTRANSFORMFILE_ERRMSG***, ***NWX_XSLTRANSFORMFILESTODOM***

Fonction NWX_XSLTRANSFORMFILE_ERRMSG (Librairie NWXML)

La fonction NWX_XSLTRANSFORMFILE_ERRMSG permet de transformer un fichier XML en un autre fichier (XML, HTML, texte …) suivant la feuille de style XSL.

Syntaxe	**NWX_XSLTRANSFORMFILE_ERRMSG** *(xmlfile, xslfile, outfile)*			
Paramètres	xmlfile	CSTRING	I	fichier XML
	xslfile	CSTRING	I	fichier XSL
	outfile	CSTRING	O	fichier de sortie
Valeur retournée	DynStr vide si pas d'erreur dans la transformation du document, sinon le texte de l'erreur			

1. L'entête de la feuille de style devrait être :

```
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"> et non pas
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/TR/WD-xsl" version="1.0">
```

2. Il est important d'utiliser comme espace de nommage l'adresse du W3C pour la recommandation candidate ainsi que d'indiquer le numéro de la version.

Exemple :

```
local Dynstr ds
ferase EF_PATH&"\"&EF_FILE&".xjt"
ds = NWX_XSLTRANSFORMFILE_ERRMSG (EF_PATH&"\"&EF_FILE&".xml",
EF_PATH&"\NatStar2NatJetbug.xsl",EF_PATH&"\"&EF_FILE&".xjt")
if ds <> ""
  message "erreur", ds
endif
LOAD EF_PATH&"\"&EF_FILE&".xjt" TO MLE1
```

Voir aussi ***NWX_XSLTRANSFORM_ERRMSG***, ***NWX_XSLTRANSFORM***, ***NWX_XSLTRANSFORMFILE***, ***NWX_XSLTRANSFORMFILESTODOM***

Fonction NWX_XSLTRANSFORMFILESTODOM (Librairie NWXML)

La fonction NWX_XSLTRANSFORMFILESTODOM produit une arborescence DOM à partir d'un fichier XML et d'une feuille de style XSL.

Syntaxe	**NWX_XSLTRANSFORMFILESTODOM** *(xmlfile$, xslfile$, pDocXMLOut)*			
Paramètres	xmlfile$	CSTRING	I	fichier XML
	xslfile$	CSTRING	I	fichier XSL
	pDocXMLOut	POINTER	O	pointeur vers un arbre DOM
Valeur renvoyée	DYNSTR non vide si erreur.			

Exemple :

```
local Dynstr ds
Local pointer pParser
Local pointer pXML
Local pointer pXSL
Local pointer pResultDoc
Local ok% Local ok2%
ok2%=NWX_INITIALIZE%
pParser = NWX_PARSER_NEW
ds =NWX_XSLTRANSFORMFILESTODOM ("C:\Widget.scrxml",
"C:\NatStar2NatJet.xsl", pResultDoc)
if ds <> ""
  message "erreur", ds
endif
; exporter le résultat dans un fichier ou un buffer
NWX_DOCUMENT_SETPRETTYPRINT (pResultDoc, true%)
ok% = NWX_DOCUMENT_OUTPUTFILE%(pResultDoc, "c:\Resultat.xml")
NWX_DOCUMENT_DISPOSE pResultDoc
NWX_PARSER_DISPOSE pParser
NWX_TERMINATE
```

Voir aussi ***NWX_XSLTRANSFORMFILE***, ***NWX_XSLTRANSFORM***, ***NWX_XSLTRANSFORM_ERRMSG***, ***NWX_XSLTRANSFORMFILE_ERRMSG***

Constantes

Constante NWX_ATTRIBUTE% (Librairie NWXML)

Nœud de type ATTRIBUTE

Syntaxe	**NWX_ATTRIBUTE%**

Constante NWX_CDATA_SECTION% (Librairie NWXML)

Nœud de type CDATASection

Syntaxe	NWX_CDATA_SECTION%

Constante NWX_COMMENT% (Librairie NWXML)

Nœud de type COMMENT.

Syntaxe	NWX_COMMENT%

Constante NWX_DIRECTORY$ (Librairie NWXML)

Option de type DIRECTORY. Cette constante est à utiliser avec l'instruction NWX_PARSER_SETOPTION

Syntaxe	NWX_DIRECTORY$

Constante NWX_DOCUMENT% (Librairie NWXML)

Nœud de type DOCUMENT.

Syntaxe	NWX_DOCUMENT%

Constante NWX_DOCUMENT_FRAGMENT% (Librairie NWXML)

Nœud de type DOCUMENT_FRAGMENT.

Syntaxe	NWX_DOCUMENT_FRAGMENT%

Constante NWX_DOCUMENT_TYPE% (Librairie NWXML)

Nœud de type DOCUMENT_TYPE.

Syntaxe	NWX_DOCUMENT_TYPE%

Constante NWX_ELEMENT% (Librairie NWXML)

Nœud de type ELEMENT.

Syntaxe	NWX_ELEMENT%

Constante NWX_ENTITY% (Librairie NWXML)

Nœud de type ENTITY.

Syntaxe	NWX_ENTITY%

Constante NWX_ENTITY_REFERENCE% (Librairie NWXML)

Nœud de type ENTITY_REFERENCE.

Syntaxe	NWX_ENTITY_REFERENCE%

Constante NWX_NOTATION% (Librairie NWXML)

Nœud de type NOTATION.

Syntaxe	NWX_NOTATION%

Constante NWX_PROCESSING_INSTRUCTION% (Librairie NWXML)

Nœud de type PROCESSING INSTRUCTION.

Syntaxe	NWX_PROCESSING_INSTRUCTION%

Constante NWX_TEXT% (Librairie NWXML)

Nœud de type TEXT.

Syntaxe	NWX_TEXT%

Constante NWX_XMLFILE$ (Librairie NWXML)

Option de type FILE. Cette constante est à utiliser avec l'instruction NWX_PARSER_SETOPTION

Syntaxe	NWX_ XMLFILE$

Catégories fonctionnelles de la librairie NSTHXML

La librairie NSTHXML est intégrée uniquement à l'outil NatStar.

Récupération des instances

La fonction suivante est spécifique à l'outil NatStar et permet de récupérer les caractéristiques d'instances, chargées en mémoire dans un fichier XML.

Fonction THINGS_DUMP_XML% (Librairie NSTHXML)

La fonction THINGS_DUMP_XML% permet d'exporter les caractéristiques d'instances chargées en mémoire dans un fichier XML.

Syntaxe	**THINGS_DUMP_XML%** *(Hinst%, File_Out$, Cst%)*			
Paramètres	Hinst%	POINTER	I	Handle d'instance
	File_Out$	CSTRING(255)	O	Fichier de sortie
	Cst%	INT(1)	I	Type de recherche
Valeur renvoyée	INT(1) 0 : L'exécution s'est bien déroulée Autre : Erreur			

Le paramètre Cst% peut prendre la valeur XML_THING%. Cette constante indique que l'on souhaite récupérer les caractéristiques de l'instance HINST% ainsi que ses instances liées avec les liens SetOf et Refto.

Le handle d'instance ne peut pas être nul.

Exemple :

```
local ok%
local ok2%

ok2%=NWX_INITIALIZE%
ok% = THINGS_DUMP_XML%(ldata1,'C:\CXMLREF.xml', XML_THING%)
```

Voir aussi Constante XML_THING%, Description des principales balises générées

Constante XML_THING% (Librairie NSTHXML)

XML_THING% indique que l'on souhaite exporter les caractéristiques de l'instance HINST% de la fonction THINGS_DUMP_XML% ainsi que ses instances liées.

Syntaxe	**XML_THING%**

Description des principales balises générées (Librairie NSTHXML)

Balise	Définition
<Thing_root></Thing_root>	Balise principale du document.

<Thing_info></Thing_info>	Les informations contenues dans la balise <Thing_info> sont celles de chaque instance : NameClass : nom de la classe ShortDescInt : description courte de l'instance
<Var></Var>	Balise recensant toutes les informations des variables de l'instance : Name : nom de la variable <Value></Value> : la valeur de la variable <IDType> </IDType> : type de la variable (les valeurs possibles sont INT, STRING, CSTRING, NUM, CHAR, BYTE, SEGMENT, REFTO, SETOF) <TypeOf> </TypeOf> : Dans le cas d'une variable de type SETOF ou REFTO, cette balise contient le nom de la classe pointée. Pour les autres cas elle contient le typedef de la variable <ValuePrimaryKey/> : valeur de la clé primaire pour la classe d'instance referencée par la variable RefTo <LongDescInst> </LongDescInst> : description longue de l'instance référencée par la variable RefTo. <SetofCount></SetOfCount> : contient (dans le cas d'une variable de type SETOF) le nombre d'éléments de la liste d'instances associées à la variable. <Status>< /Status> : indique (dans le cas d'une variable de type SETOF) l'état des instances référencées (0 : la variable ne référence aucune instance ; 1 : les instances référencées par la variable sont chargée en mémoire)

Exemple :

```
<Var Name="ADDRESS" HClass="269404418">
 <Value>1</Value>
 <Index>0</Index>
 <Size>14</Size>
 <IDType>REFTO</IDType>
 <TypeOf>ADDRESS</TypeOf>
 <DBName/>
```

```
 <BackRef/>
 <Flag>0</Flag>
 <HinstRefTO>269431102</HinstRefTO>
 <ValuePrimaryKey/>
 <LongDescInst>ADDRESS 1</LongDescInst>
</Var>
```

Glossaire

Attribut

Un attribut XML permet de spécifier les caractéristiques d'un élément. Par exemple, on peut définir les caractéristiques d'une voiture par sa marque et sa couleur sous la forme :

```
<Voiture Marque="Toyota" Couleur="Rouge"/>
```

Syntaxe	<nom_élément nom_attribut1= "valeur de l'attribut1" nom_attribut2= "valeur de l'attribut2">

CDATASection

Les sections CDATA sont utilisées pour protéger des caractères de balisage dans des portions de textes. Le seul délimiteur reconnu dans une section CDATA est la chaîne "]]>" qui ferme les sections CDATA. Les sections CDATA ne peuvent pas être imbriquées les unes dans les autres. L'objectif premier est d'inclure tel quel des fragments XML, sans avoir à protéger tous les délimiteurs propres au langage.

Document

Un nœud Document représente tout le document XML et constitue donc la racine de l'arbre.

DocumentFragment

Un nœud DocumentFragment représente un fragment du Document.

DocumentType

Chaque Document a un attribut doctype dont la valeur est soit null soit un nœud DocumentType. Le DocumentType permet de définir le type du Document.

DOM

DOM (Document Object Model) correspond à un langage standard d'interface (API) permettant à un programme de naviguer dans un arbre XML et d'en lire ou d'en modifier le contenu. DOM impose de construire un arbre en mémoire contenant l'intégralité des éléments du document en mémoire.

DTD

La DTD (Document type Definition) est un composant optionnel d'un document XML. Il permet de différencier un document bien formé d'un document valide. Un document bien formé respecte les règles syntaxiques du langage XML. Un document valide est conforme à la grammaire définie par la DTD. Ainsi, associer une DTD à un document XML permet de vérifier sa validité.
La DTD spécifie :

- Les balises contenues dans les documents,
- Les balises pouvant contenir d'autres balises,
- Le nombre et la séquence des balises,
- Les attributs joints aux balises,
- La valeur des attributs.

Element

Un nœud Element correspond à l'objet de base d'un document XML.

Syntaxe	<nom_élément nom_attribut1= "valeur de l'attribut1" nom_attribut2= "valeur de l'attribut2"> données …</nom_élément>

Entity

Un nœud Entity représente n'importe quelle entité d'un document XML qu'elle soit analysable ou non par le parseur. Notez que l'interface modélise l'entité elle-même et non sa déclaration. La modélisation des déclarations d'entités sera intégrée dans une prochaine spécification de DOM.

Entity reference

Les nœuds EntityReference correspondent à des abréviations pour des données répétitives. Elles sont référencées dans le document XML et délimitées par les caractères & et ; .
Exemple de déclaration d'entité :

```
<!ENTITY xml " Extensible Markup Language ">
```

Exemple d'utilisation dans un document XML : Le langage XML (&xml ;) …
Résultat : Le langage XML (Extensible Markup Language) …

Feuille de style

Une feuille de style contient un ensemble de règles de présentation d'un document.

Namespaces ou espaces de noms

Un espace de noms est un mécanisme utilisé pour lever les éventuelles ambiguïtés des balises.
Par exemple, la balise <NOM> peut désigner le nom d'une personne ou d'un produit. Les espaces de noms permettent de les différencier en utilisant un préfixe : « perso:nom » et « produits:nom »
Exemple de déclaration des espaces de noms :

```
<BIBLIO xmlns= « http://www.purchase.com »
xmlns:perso= « http://www.noms.org »>
```

Les balises non préfixées se rapporteront à l'espace de noms par défaut « http://www.purchase.com ». Les balises préfixées par « perso : » se rapporteront à l'URL « http://www.noms.org ».

Noeud

Un nœud correspond au point d'un arbre d'où partent une ou plusieurs branches.

Notation

Un nœud Notation sert, soit pour déclarer par un nom, le format d'une entité non contrôlée par le parseur, soit pour des déclarations formelles de cibles d'instructions de traitement. Un nœud Notation n'a aucun parent.
Syntaxe pour définir le type de données non XML utilisé par une entité externe :

```
< !NOTATION NomNotation SYSTEM|PUBLIC " ValeurNotation ">
< !ENTITY NomEntiteExterne SYSTEM " FichierNonXML " NDATA NomNotation>
```

Exemple :

```
< !NOTATION JPEG SYSTEM " PaintShopPro.exe " >
< !ENTITY logo SYSTEM " Nat System.jpg " NDATA JPEG >
```

Parseur

Un parseur XML analyse les flux XML et stocke les contenus en mémoire sous forme de DOM (Document Object Model) pour des traitements ultérieurs.

Processing Instruction

Un nœud ProcessingInstruction représente une instruction de traitement, utilisée en XML comme un moyen pour conserver des informations spécifiques à un processeur dans le texte même du document. Les instructions de traitement commençant par xml ont un usage réservé par le standard XML.

Syntaxe	< ? Nom_Instruction ContenuLibre ?>

Exemple de la déclaration XML :

```
< ?xml version= " 1.0 " encoding= " UTF-8 " standalone= " yes" ?>
```

SAX

SAX (Simple API pour XML) est une API basée sur un modèle événementiel, cela signifie que SAX permet de déclencher des événements au cours de l'analyse du document XML.
Une application basée sur SAX peut gérer uniquement les éléments dont elle a besoin sans avoir à construire en mémoire une structure contenant l'intégralité du document.

W3C

Organisme de proposition et de normalisation des protocoles et langages du Web. Site du W3C : http://www.w3.org

XML

XML est un langage de balisage extensible, c'est-à-dire que les développeurs peuvent créer leurs propres balises en fonction du type d'information à stocker et/ou à échanger. XML distingue bien les données et leur présentation. La présentation des données étant prise en charge par XSL.

XSD

XSD (XML Schema Definition) est un langage de description qui définit de façon structurée le type de contenu, la syntaxe et la sémantique d'un document XML. Il est également utilisé pour valider un document XML, c'est-à-dire vérifier si le document XML respecte les règles décrites dans le document XSD. Contrairement à la norme DTD, XSD respecte le format XML.

XSL

XSL correspond au langage de feuilles de style XML. XSL définit différentes règles de présentation d'un document XML selon le support de lecture.

XSLT

Le processus XSLT permet de transformer un document XML en appliquant la feuille de style adéquate selon le type de terminal connecté.

LIBRAIRIE NSSAXXML

Ce chapitre présente la librairie NSSAXXML intégrée aux outils de développement Nat System. Elle permet de gérer l'analyse du document XML.

Installation

Déclarez NsSaxXml.NCL dans les librairies nécessaires au développement de votre application.
Vérifiez que les fichiers NWXML.DLL et NWXMLLIB sont bien dans un des répertoires du PATH sous Windows.

Fichier NSSAXXML.NCL

Les verbes de la librairie NSSAXXML, décrits ci après, sont déclarés dans le fichier texte écrit en NCL, de nom NsSaxXml.NCL. Ce fichier peut aussi contenir des verbes complémentaires (API non publique). Vous pouvez donc désirer consulter ce fichier pour avoir la référence exhaustive de la librairie.
Pour consulter le fichier NsSaxXml.NCL :

1. Placez-vous dans le répertoire <NATSTAR>\NCL.

<NATSTAR> représente le répertoire que vous avez choisi au moment de l'installation de NatStar. La procédure est identique pour NatWeb et NS-DK.

2. Editez le fichier NsSaxXml.NCL avec n'importe quel éditeur de texte.

Catégories fonctionnelles de la librairie NSSAXXML

Démarrer et terminer l'utilisation de la librairie

Fonction NSAX_INITIALIZE% (Librairie NsSaxXml)

Crée une connexion avec une session donnée.

Syntaxe	**NSAX_INITIALIZE%** *(UNREPFLAGS%,EXPANDNAMESSPACES%)*			
Paramètres	UNREPFLAGS%	INT(4)	I	flag
	EXPANDNAMESSPACES%	INT(1)	I	extension ou non des alias

1. Le paramètre UNREPFLAGS% indique au parseur ce qu'il doit faire lorsqu'il ne peut représenter un caractère dans le jeu de caractère de l'appelant (charset) :

Syntaxe	Déclaration interne	Description
cSAX_CharRef%	0	affiche le caractère comme une référence.
cSAX_fail%	1	l'opération échoue.

cSAX_replace%	2	remplace le caractère avec un caractère de remplacement.

2. Le paramètre EXPANDNAMESSPACES% indique si l'on doit étendre les alias du namespace avec leurs URI. Si cette option est sélectionnée, il faut valider le traitement des "espaces nommés" via l'instruction NSAX_PARSERSETFEATURE avec l'option cSAX_fgSAX2CoreNameSpaces.

Voir aussi *constantes cSAX_*, NSAX_TERMINATE, NSAX_PARSERSETFEATURE*

Instruction NSAX_TERMINATE (Librairie NsSaxXml)

Libère les ressources allouées par la librairie SAX.

Syntaxe	**NSAX_TERMINATE**

Voir aussi *NSAX_INITIALIZE*

Gestion du parseur

Fonction NSAX_CREATEPARSER (Librairie NsSaxXml)

Crée un objet "lecteur" associé à l'analyseur pour permettre l'analyse d'un document.

Syntaxe	**NSAX_CREATEPARSER** *(PDATA, LOCATION)*			
Paramètres	PDATA	POINTER	I	pointeur fourni par l'appelant
	LOCATION	POINTER	I/O	retourne dans LOCATION un pointeur
Valeur retournée	INT			

1. Le pointeur PDATA est fourni par l'appelant et sera enregistré dans l'objet créé puis transmis dans chacune des méthodes de call back. Ainsi, l'appelant peut associer un contexte au parseur.
2. Le paramètre LOCATION retourne dans lui-même un pointeur qui pourra être utilisé pendant l'analyse du document avec les fonctions suivantes :
   a) NSAX_GETLINENUMBER%
   b) NSAX_GETLINENUMBER%
   c) NSAX_GETPUBLICID
   d) NSAX_GETSYSTEMID

Exemple :

```
Local PDATA%

INITIALIZE_PARSER Selection%(CB_CHARS),CHK_EXPANDNAMESSPACES=CHECKED%
New S_PRIVDATA,PDATA%

Insert At End "*** Create Parser result: " &
NSAX_CREATEPARSER(PDATA%,P_LOC) & " ***" TO LERR
```

```
S_PRIVDATA(PDATA%).LOCATOR = P_LOC
NSAX_PARSERSETFEATURE cSAX_fgSAX2CoreValidation,
CHK_COREVALIDATION=CHECKED%
;Check box CHK_COREVALIDATION checked

Insert At End "*** Parse result : " & NSAX_STARTPARSER(CB_XMLFILE) & " ***"
TO LERR

NSAX_RELEASEPARSER
Dispose PDATA%

NSAX_TERMINATE
```

Voir aussi ***NSAX_RELEASEPARSER***, ***NSAX_PARSERSETFEATURE***, ***NSAX_STARTPARSER***

Instruction NSAX_RELEASEPARSER (Librairie NsSaxXml)

Détruit l'objet utilisé pour l'analyse du document.

Syntaxe	**NSAX_RELEASEPARSER**

Exemple :

```
Local PDATA%

INITIALIZE_PARSER Selection%(CB_CHARS),CHK_EXPANDNAMESSPACES=CHECKED%
New S_PRIVDATA,PDATA%

Insert At End "*** Create Parser result: " &
NSAX_CREATEPARSER(PDATA%,P_LOC) & " ***" TO LERR

S_PRIVDATA(PDATA%).LOCATOR = P_LOC
NSAX_PARSERSETFEATURE cSAX_fgSAX2CoreValidation,
CHK_COREVALIDATION=CHECKED%
;Check box CHK_COREVALIDATION checked

Insert At End "*** Parse result : " & NSAX_STARTPARSER(CB_XMLFILE) & " ***"
TO LERR

NSAX_RELEASEPARSER
Dispose PDATA%

NSAX_TERMINATE
```

Voir aussi ***NSAX_CREATEPARSER***, ***NSAX_PARSERSETFEATURE***, ***NSAX_STARTPARSER***

Instruction NSAX_PARSERSETFEATURE (Librairie NsSaxXml)

Positionne différents comportements de l'analyseur syntaxique créés au préalable par NSAX_CREATEPARSER.

Syntaxe	**NSAX_PARSERSETFEATURE** *(FEATURE%, VAL%)*			
Paramètres	FEATURE%	INT(4)	I	une constante cSAX_*%
	VAL%	INT(4)	I	TRUE% ou FALSE%

Exemple :

```
Local PDATA%

INITIALIZE_PARSER Selection%(CB_CHARS),CHK_EXPANDNAMESSPACES=CHECKED%

New S_PRIVDATA,PDATA%


Insert At End "*** Create Parser result: " &
NSAX_CREATEPARSER(PDATA%,P_LOC) & " ***" TO LERR

S_PRIVDATA(PDATA%).LOCATOR = P_LOC
NSAX_PARSERSETFEATURE cSAX_fgSAX2CoreValidation,
CHK_COREVALIDATION=CHECKED%
;Check box CHK_COREVALIDATION checked

Insert At End "*** Parse result : " & NSAX_STARTPARSER(CB_XMLFILE) & " ***"
TO LERR

NSAX_RELEASEPARSER
Dispose PDATA%

NSAX_TERMINATE
```

Voir aussi constantes cSAX_*%, NSAX_CREATEPARSER, NSAX_RELEASEPARSER, NSAX_STARTPARSER

Constantes cSAX_* (Librairie NsSaxXml)

Comportements de l'analyseur syntaxique.

Syntaxe	Déclaration interne	Description
cSAX_fgSAX2CoreValidation	101	Positionné à vrai, le document doit spécifier une grammaire de validation. Par défaut : VRAI
cSAX_fgXercesDynamic	102	Positionné à vrai, l'analyseur validera le document seulement si une grammaire est fournie. Par défaut : FAUX
cSAX_fgSAX2CoreNameSpaces	103	Positionné à vrai, le document doit contenir une grammaire qui supporte les espaces nommés. Par défaut : VRAI

cSAX_fgXercesSchema	104	Validation du traitement des "schema". S'il est positionné à vrai, la constante cSAX_fgSAX2CoreNameSpaces doit l'être également. Par défaut : VRAI
cSAX_fgXercesSchemaFullChecking	105	Ce dispositif examine la grammaire de schéma pour déceler les erreurs additionnelles qui sont longues ou qui utilisent intensivement de la mémoire. Cela n'affecte pas le niveau de vérification exécuté sur les instances de document qui emploient des schémas de grammaire. Par défaut : FAUX
cSAX_fgSAX2CoreNameSpacePrefixes	106	Positionné à vrai. Rapporte les noms et les attributs préfixés originaux utilisés pour des déclarations de namespace. Par défaut : FAUX
cSAX_CharRef%	0	Utilisées avec la fonction NSAX_INITIALIZE%, elles indiquent au parseur ce qu'il doit faire lorsqu'il ne peut représenter un caractère dans le jeu de caractère de l'appelant (charset). Voir la fonction NSAX_INITIALIZE%.
cSAX_fail%	1	
cSAX_replace%	2	

Fonction NSAX_STARTPARSER (Librairie NsSaxXml)

Démarre l'analyse du document spécifié en paramètre.

Syntaxe	**NSAX_STARTPARSER** *(xmlfile$)*			
Paramètre	*xmlfile$*	CSTRING	I	nom d'un fichier XML
Valeur retournée	INT			

Exemple :

```
Local PDATA%
```

```
INITIALIZE_PARSER Selection%(CB_CHARS),CHK_EXPANDNAMESSPACES=CHECKED%
New S_PRIVDATA,PDATA%

Insert At End "*** Create Parser result: " &
NSAX_CREATEPARSER(PDATA%,P_LOC) & " ***" TO LERR

S_PRIVDATA(PDATA%).LOCATOR = P_LOC
NSAX_PARSERSETFEATURE cSAX_fgSAX2CoreValidation,
CHK_COREVALIDATION=CHECKED%
;Check box CHK_COREVALIDATION checked

Insert At End "*** Parse result : " & NSAX_STARTPARSER(CB_XMLFILE) & " ***"
TO LERR

NSAX_RELEASEPARSER
Dispose PDATA%

NSAX_TERMINATE
```

Voir aussi ***NSAX_CREATEPARSER***, ***NSAX_RELEASEPARSER***, ***NSAX_PARSERSETFEATURE***

Méthodes d'analyse d'attributs de nœuds

Fonction NSAX_ATTRIBUTES_GETLENGTH% (Librairie NsSaxXml)

Retourne le nombre d'attribut.

Syntaxe	**NSAX_ATTRIBUTES_GETLENGTH%** *(P)*			
Paramètre	*P*	POINTER	I/O	pointeur sur un attribut

Voir aussi ***NSAX_ATTRIBUTE_GETNAME%***, ***NSAX_ATTRIBUTE_GETTYPE%***, ***NSAX_ATTRIBUTE_GETVALUE%***

Fonction NSAX_ATTRIBUTE_GETNAME% (Librairie NsSaxXml)

Retourne le nom d'attribut sous la forme d'un pointeur sur une DynStr.

Syntaxe	**NSAX_ATTRIBUTE_GETNAME%** *(P, INDEX, EXPANDNAMESSPACES%)*			
Paramètres	P	DYNSTR	I/O	Retourne le nom sous la forme d'un pointer sur une DynStr. Ce pointeur est un DynStr. S'il est différent de NULL, il doit être libéré ensuite avec NSAX_DISPOSE.
	INDEX	INT	I	index d'attribut
	EXPANDNAMESSPACES%	INT(1)	I	flag indiquant l'extension ou non des

				alias des namespaces avec leurs URI

Voir aussi ***NSAX_DISPOSE*****,** ***NSAX_ATTRIBUTES_GETLENGTH%*****,** ***NSAX_ATTRIBUTES_GETTYPE%*****,** ***NSAX_ATTRIBUTES_GETVALUE%***

Fonction NSAX_ATTRIBUTE_GETTYPE% (Librairie NsSaxXml)

Retourne le type d'attribut sous la forme d'un pointeur sur une DynStr.

Syntaxe	**NSAX_ATTRIBUTE_GETTYPE%** *(P, INDEX)*			
Paramètres	P	DYNSTR	I/O	Retourne le nom sous la forme d'un pointer sur une DynStr. Ce pointeur est un DynStr. S'il est différent de NULL, il doit être libéré ensuite avec NSAX_DISPOSE.
	INDEX	INT	I	index d'attribut

Les types possibles sont :

"NOTATION"	Une suite de noms de notation séparés par un trait vertical. Chaque nom doit correspondre exactement à un nom de notation déclaré dans la DTD. NOTATION (val1 | val2 | ..)
"CDATA"	Accepte n'importe quelle chaîne de caractères. <!ATTLIST nom_elmt nom_attrib CDATA>
"ENTITY"	Définit comme valeur d'attribut une entité non parsée déclarée dans une DTD. <!ATTLIST nom_elmt nom_attrib ENTITY #option>
"ENTITIES"	Définit comme valeur d'attribut plusieurs entités (séparées par des espaces) non parsées déclarées dans une DTD. <!ATTLIST nom_elmt nom_attrib ENTITIES #option>
"ID"	Indique que l'attribut possède une valeur unique pour chaque élément. La valeur peut contenir une ou plusieurs lettres, des chiffres, des points (.), des tirets hauts (-) ou bas (_), et un double point (:). <!ATTLIST nom_elmt nom_attrib ID #option>
"IDREF"	Indique que la valeur de l'attribut se réfère à un ID d'un autre attribut. <!ATTLIST nom_elmt nom_attrib IDREF #option>
"IDREFS"	Identique à IDREF excepté que la valeur de l'attribut peut se référer à plusieurs ID, chaque valeur étant séparée par un espace. <!ATTLIST nom_elmt nom_attrib IDREFS #option>

"NMTOKEN"	Indique que la valeur de l'attribut est une chaîne de caractères pouvant contenir une ou plusieurs lettres, des chiffres, des points (.), des tirets hauts (-) ou bas (_), et un double point (:). <!ATTLIST nom_elmt nom_attrib NMTOKEN #option>
"NMTOKENS"	Identique à NMTOKEN excepté que l'attribut peut avoir plusieurs valeurs séparées par des espaces. <!ATTLIST nom_elmt nom_attrib NMTOKENS #option>

Voir aussi ***NSAX_DISPOSE*****,** ***NSAX_ATTRIBUTES_GETLENGTH%*****,** ***NSAX_ATTRIBUTE_GETNAME%*****,** ***NSAX_ATTRIBUTE_GETVALUE%***

Fonction NSAX_ATTRIBUTES_GETVALUE% (Librairie NsSaxXml)

Retourne la valeur éventuelle d'un attribut sous la forme d'un pointer sur une DynStr.

Syntaxe	**NSAX_ATTRIBUTES_GETVALUE%** *(P, INDEX)*			
Paramètres	P	DYNSTR	I/O	Retourne le nom sous la forme d'un pointer sur une DynStr. Ce pointeur est un DynStr. S'il est différent de NULL, il doit être libéré ensuite avec NSAX_DISPOSE.
	INDEX	INT	I	index d'attribut

Voir aussi ***NSAX_DISPOSE*****,** ***NSAX_ATTRIBUTES_GETLENGTH%*****,** ***NSAX_ATTRIBUTE_GETNAME%*****,** ***NSAX_ATTRIBUTE_GETTYPE%***

Méthodes à utiliser avec un LOCATOR

Le LOCATOR permet de localiser le curseur pendant le traitement du flux. Ainsi, par exemple on peut connaître le numéro de ligne et de colonne en cours d'analyse.

Fonction NSAX_GETCOLUMNNUMBER% (Librairie NsSaxXml)

Retourne le numéro de colonne.

Syntaxe	**NSAX_GETCOLUMNNUMBER%** *(P)*			
Paramètre	*P*	POINTER	I	Pointeur sur un "Locator"

Voir aussi ***NSAX_GETLINENUMBER%*****,** ***NSAX_GETPUBLICID*****,** ***NSAX_GETSYSTEMID***

Fonction NSAX_GETLINENUMBER% (Librairie NsSaxXml)

Retourne le numéro de ligne.

Syntaxe	**NSAX_GETLINENUMBER%** *(P)*			
Paramètre	*P*	POINTER	I	Pointeur sur un "Locator"

Voir aussi ***NSAX_GETCOLUMNNUMBER%*****,** ***NSAX_GETPUBLICID*****,** ***NSAX_GETSYSTEMID***

Fonction NSAX_GETPUBLICID (Librairie NsSaxXml)

Retourne un pointeur sur l'identificateur public (URN).

Syntaxe	**NSAX_GETPUBLICID** *(P)*			
Paramètre	*P*	DYNSTR	I	Pointeur sur un "Locator"
Valeur retournée	POINTER			

Ce pointeur est un DynStr. S'il est différent de NULL, il doit être libéré ensuite avec NSAX_DISPOSE.

Voir aussi *NSAX_GETCOLUMNNUMBER%*, *NSAX_GETLINENUMBER%*, *NSAX_GETSYSTEMID*

Fonction NSAX_GETSYSTEMID (Librairie NsSaxXml)

Retourne un pointeur sur l'identificateur système (URL).

Syntaxe	**NSAX_GETSYSTEMID** *(P)*			
Paramètre	*P*	DYNSTR	I	Pointeur sur un "Locator"
Valeur retournée	POINTER			

Ce pointeur est un DynStr. S'il est différent de NULL, il doit être libéré ensuite avec NSAX_DISPOSE.

Voir aussi *NSAX_GETCOLUMNNUMBER%*, *NSAX_GETLINENUMBER%*, *NSAX_GETPUBLICID*

Méthodes à utiliser avec un handle d'exception

Fonction NSAX_PE_GETCOLUMNNUMBER% (Librairie NsSaxXml)

Retourne le numéro de colonne.

Syntaxe	**NSAX_PE_GETCOLUMNNUMBER%** *(P)*			
Paramètre	*P*	POINTER	I/O	Pointeur sur un handle d'exception

Voir aussi *NSAX_PE_GETLINENUMBER%*, *NSAX_PE_GETPUBLICID*, *NSAX_PE_GETSYSTEMID*, *NSAX_PE_GETMESSAGE*, *NSAX_DISPOSE*

Fonction NSAX_PE_GETLINENUMBER% (Librairie NsSaxXml)

Retourne le numéro de ligne.

Syntaxe	**NSAX_PE_GETLINENUMBER%** *(P)*			
Paramètre	*P*	POINTER	I/O	Pointeur sur un handle d'exception

Voir aussi *NSAX_PE_GETCOLUMNNUMBER%*, *NSAX_PE_GETPUBLICID*, *NSAX_PE_GETSYSTEMID*, *NSAX_PE_GETMESSAGE*, *NSAX_DISPOSE*

Fonction NSAX_PE_GETPUBLICID (Librairie NsSaxXml)

Retourne un pointeur sur l'identificateur public (URN).

Syntaxe	**NSAX_PE_GETPUBLICID** *(P)*			
Paramètre	*P*	DYNSTR	I/O	Pointeur sur un handle d'exception

Ce pointeur est un DynStr. S'il est différent de NULL, il doit être libéré ensuite avec NSAX_DISPOSE.

Voir aussi ***NSAX_PE_GETCOLUMNNUMBER%***, ***NSAX_PE_GETLINENUMBER%***, ***NSAX_PE_GETSYSTEMID***, ***NSAX_PE_GETMESSAGE***, ***NSAX_DISPOSE***

Fonction NSAX_PE_GETSYSTEMID (Librairie NsSaxXml)

Retourne un pointeur sur l'identificateur système (URL).

Syntaxe	**NSAX_PE_GETSYSTEMID** *(P)*			
Paramètre	*P*	DYNSTR	I/O	Pointeur sur un handle d'exception

Ce pointeur est un DynStr. S'il est différent de NULL, il doit être libéré ensuite avec NSAX_DISPOSE.

Voir aussi ***NSAX_PE_GETCOLUMNNUMBER%***, ***NSAX_PE_GETLINENUMBER%***, ***NSAX_PE_GETPUBLICID***, ***NSAX_PE_GETMESSAGE***, ***NSAX_DISPOSE***

Fonction NSAX_PE_GETMESSAGE (Librairie NsSaxXml)

Retourne un message d'erreur.

Syntaxe	**NSAX_PE_GETMESSAGE** *(P)*			
Paramètre	*P*	DYNSTR	I/O	Pointeur sur un handle d'exception

Ce pointeur est un DynStr. S'il est différent de NULL, il doit être libéré ensuite avec NSAX_DISPOSE.

Voir aussi ***NSAX_PE_GETCOLUMNNUMBER%***, ***NSAX_PE_GETLINENUMBER%***, ***NSAX_PE_GETPUBLICID***, ***NSAX_PE_GETSYSTEMID***, ***NSAX_DISPOSE***

Instruction NSAX_DISPOSE (Librairie NsSaxXml)

Permet de libérer la mémoire allouée pour les DynStr retournés par les méthodes NSAX_GETPUBLICID, NSAX_GETSYSTEMID, NSAX_PE_GETPUBLICID, NSAX_PE_GETSYSTEMID et NSAX_PE_GETMESSAGE.

Syntaxe	**NSAX_DISPOSE** *P*			
Paramètre	P	DYNSTR	I/O	Pointeur sur un handle d'exception

Voir aussi ***NSAX_PE_GETCOLUMNNUMBER%***, ***NSAX_PE_GETLINENUMBER%***, ***NSAX_PE_GETPUBLICID***, ***NSAX_PE_GETSYSTEMID***, ***NSAX_PE_GETMESSAGE***

Prototypes des méthodes de callback

L'analyseur syntaxique appelle automatiquement une méthode lorsqu'un événement est détecté dans le fichier XML. Sept événements sont détectés par la librairie :

détection d'une balise de début (SAXH_STARTELEMENT), d'une balise de fin (SAXH_ENDELEMENT), de données entre deux balises (SAXH_CHARACTERS), début du traitement d'un document XML (SAXH_STARTDOCUMENT), fin du traitement (SAXH_ENDDOCUMENT), détection d'un commentaire (SAXH_COMMENT), détection d'une instruction de traitement.
Ces fonctions peuvent être utilisées en les enregistrant dans l'analyseur syntaxique XML avec l'instruction NSAX_INIFNT.

Instruction NSAX_INIFNT (Librairie NsSaxXml)

Informe la librairie des différentes méthodes à appeler en callback.

Syntaxe	**NSAX_INIFNT** *TY, P*			
Paramètres	TY	INTEGER	I	constante cSAXH_* indiquant la finalité de la méthode
	P	POINTER	I	pointeur sur la fonction SAXH_* à implémenter

Exemple :

```
Instruction INITIALIZE_PARSER UNREPFLAGS%,EXPANDNAMESSPACES%(1)
Local Int RET

  NSAX_INIFNT cSAXH_WRITE,@SAXH_WRITE

NSAX_INIFNT cSAXH_STARTDOCUMENT,@SAXH_STARTDOCUMENT
  NSAX_INIFNT cSAXH_ENDDOCUMENT,@SAXH_ENDDOCUMENT

  NSAX_INIFNT cSAXH_STARTELEMENT,@SAXH_STARTELEMENT
  NSAX_INIFNT cSAXH_ENDELEMENT,@SAXH_ENDELEMENT

NSAX_INIFNT cSAXH_PROCESSINGINSTRUCTION,@SAXH_PROCESSINGINSTRUCTION
  NSAX_INIFNT cSAXH_COMMENT,@SAXH_COMMENT
  NSAX_INIFNT cSAXH_CHARACTERS,@SAXH_CHARACTERS

  NSAX_INIFNT cSAXH_ERROR,@SAXH_ERROR
  NSAX_INIFNT cSAXH_FATALERROR,@SAXH_FATALERROR
  NSAX_INIFNT cSAXH_WARNING,@SAXH_WARNING

RET = NSAX_INITIALIZE%(UNREPFLAGS%,EXPANDNAMESSPACES%)

EndInstruction
```

Voir aussi constantes ***cSAX_**, SAXH_STARTDOCUMENT, SAXH_ENDDOCUMENT, SAXH_STARTELEMENT, SAXH_ENDELEMENT, SAXH_PROCESSINGINSTRUCTION, SAXH_COMMENT, SAXH_CHARACTERS***

Constantes cSAXH_* (Librairie NsSaxXml)

Permet d'indiquer la finalité de la méthode appelée en callback par l'instruction NSAX_INIFNT.

Syntaxe	Déclaration interne	Description
cSAXH_WRITE	1	Texte – Réservé
cSAXH_STARTDOCUMENT	8	Début/Fin de document
cSAXH_ENDDOCUMENT	9	
cSAXH_STARTELEMENT	10	Eléments
cSAXH_ENDELEMENT	11	
cSAXH_PROCESSINGINSTRUCTION	12	Instruction
cSAXH_COMMENT	13	Commentaire
cSAXH_CHARACTERS	14	Caractère
cSAXH_ERROR	20	Gestion des Erreurs
cSAXH_FATALERROR	21	
cSAXH_WARNING	22	

Exemple :

```
NSAX_INIFNT cSAXH_STARTDOCUMENT,@SAXH_STARTDOCUMENT
```

Voir aussi ***NSAX_INIFNT, SAXH_STARTDOCUMENT, SAXH_ENDDOCUMENT, SAXH_STARTELEMENT, SAXH_ENDELEMENT, SAXH_PROCESSINGINSTRUCTION, SAXH_COMMENT, SAXH_CHARACTERS***

Fonction SAXH_STARTDOCUMENT (Librairie NsSaxXml)

Appelé au démarrage de l'analyse syntaxique.
Cette méthode est appelée par le parseur (une fois uniquement) au démarrage de l'analyse du flux XML. Elle est appelée avant toutes les autres méthodes de l'interface, à l'exception unique, évidemment, de la méthode setDocumentLocator. Cet événement devrait vous permettre d'initialiser tout ce qui doit l'être avant le début du parcours du document

Syntaxe	**SAXH_STARTDOCUMENT** *(PDATA)*			
Paramètre	PDATA	POINTER	I	Contexte donné via NSAX_CREATEPARSER
Valeur retournée	INT			

Exemple :

```
Global Control C_LST

FUNCTION SAXH_STARTDOCUMENT(POINTER PDATA) RETURN INT
Insert At End " *** SAXH_STARTDOCUMENT *** " TO C_LST
Return TRUE%
ENDFUNCTION
```

Voir aussi ***NSAX_CREATEPARSER***, ***NSAX_INIFNT***, ***SAXH_ENDDOCUMENT***, ***SAXH_STARTELEMENT***, ***SAXH_ENDELEMENT***, ***SAXH_PROCESSINGINSTRUCTION***, ***SAXH_COMMENT***, ***SAXH_CHARACTERS***

Fonction SAXH_ENDDOCUMENT (Librairie NsSaxXml)

Appelé à la fin de l'analyse syntaxique.
Cette méthode est appelée à la fin du parcours du flux après toutes les autres méthodes. Il peut alors être utile à ce moment de notifier d'autres objets du fait que le travail est terminé.

Syntaxe	**SAXH_ENDDOCUMENT** *(PDATA)*			
Paramètre	PDATA	POINTER	I	Contexte donné via NSAX_CREATEPARSER
Valeur retournée	INT			

Exemple :

```
Global Control C_LST

FUNCTION SAXH_ENDDOCUMENT(POINTER PDATA) RETURN INT
Insert At End " *** SAXH_ENDDOCUMENT *** " TO C_LST
Return TRUE%
ENDFUNCTION
```

Voir aussi ***NSAX_CREATEPARSER***, ***NSAX_INIFNT***, ***SAXH_STARTDOCUMENT***, ***SAXH_STARTELEMENT***, ***SAXH_ENDELEMENT***, ***SAXH_PROCESSINGINSTRUCTION***, ***SAXH_COMMENT***, ***SAXH_CHARACTERS***

Fonction SAXH_STARTELEMENT (Librairie NsSaxXml)

Appelé au début de l'analyse d'un élément XML, lors de la détection d'une balise de début.

Syntaxe	**SAXH_STARTELEMENT** *(PDATA, NAME, ATTRIBUTES)*			
Paramètres	PDATA	POINTER	I	contexte donné via NSAX_CREATEPARSER
	NAME	CSTRING	I	nom de l'élément
	ATTRIBUTES	POINTER	I/O	pointeur sur la liste des attributs des éléments

Le paramètre ATTRIBUTES est à utiliser avec les fonctions NSAX_ATTRIBUTE_* pour lister l'ensemble des attributs et leurs valeurs.
Exemple :

```
Global Control C_LST
Global Control C_LERR
Global POINTER P_LOC
Global Line$(2048)
```

```
Segment S_PRIVDATA
 POINTER LOCATOR
EndSegment

Segment S_POINTER
 Pointer P
EndSegment
Segment S_CSTRING
 CSTRING S
EndSegment

Segment S_DYNSTR
 DYNSTR DS
EndSegment

FUNCTION SAXH_STARTELEMENT(POINTER PDATA,CSTRING NAME,POINTER \
@ATTRIBUTES) RETURN INT
Local LINE$,I%,N%,Pointer P,ILINE%

 ILINE% = NSAX_GETLINENUMBER%(S_PRIVDATA(PDATA).LOCATOR)
 LINE$ = "Line=" & ILINE% && "=<" & NAME
 Insert At End LINE$ TO C_LST

 N% = NSAX_ATTRIBUTES_GETLENGTH%(ATTRIBUTES)-1
If N% >= 0
 For I%=0 TO N%
  LINE$ = " id:" & I%
  P = NSAX_ATTRIBUTE_GETNAME%(ATTRIBUTES,I%,TRUE%)
  If P
   LINE$ = LINE$ && "NAME=" & S_CSTRING(P).S
   NSAX_DISPOSE P
  Else
   LINE$ = LINE$ && "NAME=''"
  EndIf
  P = NSAX_ATTRIBUTE_GETTYPE%(ATTRIBUTES,I%)
  If P
   LINE$ = LINE$ && "TYPE=" & S_CSTRING(P).S
   NSAX_DISPOSE P
  Else
   LINE$ = LINE$ & "TYPE=''"
  EndIf
  P = NSAX_ATTRIBUTE_GETVALUE%(ATTRIBUTES,I%)
  If P
   LINE$ = LINE$ && "VALUE=" & S_CSTRING(P).S
   NSAX_DISPOSE P
  Else
   LINE$ = LINE$ & "VALUE=''"
  EndIf
  Insert At End LINE$ TO C_LST
 EndFor
 Insert At End "..." & NAME && ">" TO C_LST
EndIf

 Return TRUE%

ENDFUNCTION
```

Voir aussi ***NSAX_CREATEPARSER, NSAX_INIFNT, SAXH_STARTDOCUMENT, SAXH_ENDDOCUMENT, SAXH_ENDELEMENT, SAXH_PROCESSINGINSTRUCTION, SAXH_COMMENT, SAXH_CHARACTERS NSAX_ATTRIBUTE_****

Fonction SAXH_ENDELEMENT (Librairie NsSaxXml)

Appelé à la fin de l'analyse d'un élément XML, lors de la détection d'une balise de fin.

Syntaxe	**SAXH_ENDELEMENT** *(PDATA, NAME)*			
Paramètres	PDATA	POINTER	I	contexte donné via NSAX_CREATEPARSER
	NAME	CSTRING	I	nom de l'élément
Valeur retournée	INT			

Exemple :

```
Global Control C_LST

FUNCTION SAXH_ENDELEMENT(POINTER PDATA,CSTRING NAME) RETURN INT
Insert At End "</"&NAME&">" TO C_LST
 Return TRUE%
ENDFUNCTION
```

Voir aussi ***NSAX_CREATEPARSER, NSAX_INIFNT, SAXH_STARTDOCUMENT, SAXH_ENDDOCUMENT, SAXH_STARTELEMENT, SAXH_PROCESSINGINSTRUCTION, SAXH_COMMENT, SAXH_CHARACTERS***

Fonction SAXH_PROCESSINGINSTRUCTION (Librairie NsSaxXml)

Nœud d'instruction de traitement dans la sortie.
Cet événement est appelé pour chaque instruction de fonctionnement rencontrée. Ces instructions sont celles que vous trouvez hors de l'arbre XML lui-même.

Syntaxe	**SAXH_PROCESSINGINSTRUCTION** *(PDATA, TARGET, DATA)*			
Paramètres	PDATA	POINTER	I	contexte donné via NSAX_CREATEPARSER
	TARGET	DYNSTR	I/O	cible
	DATA	DYNSTR	I/O	données

Exemple :

```
Global Control C_LST

FUNCTION SAXH_PROCESSINGINSTRUCTION(POINTER PDATA,DynStr @TARGET,DynStr
@DATA) RETURN INT
  Insert At End "*** PROCESSINGINSTRUCTION Target:" && TARGET && "Data:" &&
DATA TO C_LST
```

```
  Return TRUE%
ENDFUNCTION
```

Voir aussi ***NSAX_CREATEPARSER, NSAX_INIFNT, SAXH_STARTDOCUMENT, SAXH_ENDDOCUMENT, SAXH_ENDELEMENT, SAXH_STARTELEMENT, SAXH_COMMENT, SAXH_CHARACTERS***

Fonction SAXH_COMMENT (Librairie NsSaxXml)

Appelé lorsqu'un commentaire est trouvé, c'est-à-dire un texte compris entre les balises "<!—" et "-->". Par exemple <!-- Edited with XML Spy v2007 (http://www.altova.com) -->.

Syntaxe	**SAXH_COMMENT** *(PDATA, ds)*			
Paramètres	PDATA	POINTER	I	contexte donné via NSAX_CREATEPARSER
	ds	DYNSTR	I/O	commentaire

Exemple :

```
Global Control C_LST
Global Control C_LERR
Global POINTER P_LOC
Global Line$(2048)

Segment S_PRIVDATA
 POINTER LOCATOR
EndSegment

Segment S_CSTRING
 CSTRING S
EndSegment

FUNCTION SAXH_COMMENT(POINTER PDATA,DynStr @ds) RETURN INT
Local P%
 Insert At End "Line=" & NSAX_GETLINENUMBER%(S_PRIVDATA(PDATA).LOCATOR) &&
" Comment:" && ds TO C_LST
 P% = NSAX_GETSYSTEMID(S_PRIVDATA(PDATA).LOCATOR)
 If P%
  Insert At End "NSAX_GETSYSTEMID" && S_CSTRING(P%).S TO C_LST
  NSAX_DISPOSE P%
 EndIf
 P% = NSAX_GETPUBLICID(S_PRIVDATA(PDATA).LOCATOR)
 If P%
  Insert At End "NSAX_GETPUBLICID" && S_CSTRING(P%).S TO C_LST
  NSAX_DISPOSE P%
 EndIf
 Return TRUE%
ENDFUNCTION
```

Voir aussi ***NSAX_CREATEPARSER, NSAX_INIFNT, SAXH_STARTDOCUMENT, SAXH_ENDDOCUMENT, SAXH_ENDELEMENT, SAXH_STARTELEMENT, SAXH_PROCESSINGINSTRUCTION , SAXH_CHARACTERS***

Fonction SAXH_CHARACTERS (Librairie NsSaxXml)

La détection de données entre deux balises déclenche l'appel de cet événement.

Syntaxe	**SAXH_CHARACTERS** *(PDATA, src)*			
Paramètres	PDATA	POINTER	I	contexte donné via NSAX_CREATEPARSER
	src	DYNSTR	I/O	source
Valeur retournée	INT			

En général, cet événement est appelé tout simplement par la présence de texte entre la balise d'ouverture et la balise de fermeture, comme dans l'exemple suivant:

```
<maBalise>un peu de texte</maBalise>
```

Dans l'exemple ci-dessous (chars.xml), SAXH_CHARACTERS sera invoqué trois fois, pour "un peu" puis " de texte" et enfin " éparpillé ".

```
<maBalise>un peu
<baliseImbriquee nom="coucou"/> de texte<baliseImbriquee nom="un
non"/>éparpillé
</maBalise>
```

Exemple :

```
Global Control C_LST

FUNCTION SAXH_CHARACTERS(POINTER PDATA,DynStr @src) RETURN INT
Local P%, DynStr dst, Char C, I%

If Length(src) > 0

dst = ''
For I%=1 To Length(src)
C = copy$(src,I%,1)
If ASC%(C) = 10
dst = dst & '<RC>'
ElseIf ASC%(C) = 9
dst = dst & '<TAB>'
ElseIf ASC%(C) < 32
dst = dst & '<' & ASC%(C) &'>'
Else
dst = dst & C
EndIf
EndFor

Insert At End "Line=" & NSAX_GETLINENUMBER%(S_PRIVDATA(PDATA).LOCATOR) &&
'Length:' && Length(src) && " Characters:" && dst TO C_LST
 EndIf

Return TRUE%

ENDFUNCTION
```

Voir aussi NSAX_CREATEPARSER, NSAX_INIFNT, SAXH_STARTDOCUMENT, SAXH_ENDDOCUMENT, SAXH_ENDELEMENT, SAXH_STARTELEMENT, SAXH_PROCESSINGINSTRUCTION, SAXH_COMMENT

Gestion des erreurs

Fonction SAXH_ERROR% (Librairie NsSaxXml)

Retourne un code d'erreur.

Syntaxe	**SAXH_ERROR%** *(PDATA, E)*			
Paramètres	PDATA	POINTER	I	contexte donné via NSAX_CREATEPARSER
	E	POINTER	I/O	Handle à utiliser avec les méthodes NSAX_PE_*

Si retourne TRUE%, le système remet à zéro le code d'erreur.
Exemple :

```
Global Control C_LERR

Instruction ErrorMsg POINTER @E
Local Pointer P
 P = NSAX_PE_GETSYSTEMID(E)
 If P
  Insert At End "NSAX_PE_GETSYSTEMID:"&&S_CSTRING(P).S TO C_LERR
  NSAX_DISPOSE P
 EndIf
 P = NSAX_PE_GETPUBLICID(E)
 If P
  Insert At End "NSAX_PE_GETPUBLICID:"&&S_CSTRING(P).S TO C_LERR
  NSAX_DISPOSE P
 EndIf
 P = NSAX_PE_GETMESSAGE(E)
 If P
  Insert At End "NSAX_PE_GETMESSAGE:"&&S_CSTRING(P).S TO C_LERR
  NSAX_DISPOSE P
 EndIf
EndInstruction

FUNCTION SAXH_ERROR(POINTER PDATA, POINTER @E) RETURN INT
 Insert At End "SAXH_ERROR"&&"Col:"&NSAX_PE_GETCOLUMNNUMBER%(E)&"
Line:"&NSAX_PE_GETLINENUMBER%(E) TO C_LERR
 ErrorMsg(E)
 Return TRUE%
```

*Voir aussi NSAX_CREATEPARSER, SAXH_FATALERROR%, SAXH_WARNING%, NSAX_PE_**

Fonction SAXH_FATALERROR% (Librairie NsSaxXml)

Retourne une erreur fatale.

Syntaxe	**SAXH_FATALERROR%** *(PDATA, E)*

Paramètres	PDATA	POINTER	I	contexte donné via NSAX_CREATEPARSER
	E	POINTER	I/O	Handle à utiliser avec les méthodes NSAX_PE_*

Si retourne TRUE%, le système remet à zéro le code d'erreur.
Exemple :

```
Global Control C_LERR

Instruction ErrorMsg POINTER @E
Local Pointer P
 P = NSAX_PE_GETSYSTEMID(E)
 If P
  Insert At End "NSAX_PE_GETSYSTEMID:"&&S_CSTRING(P).S TO C_LERR
 NSAX_DISPOSE P
 EndIf
 P = NSAX_PE_GETPUBLICID(E)
 If P
  Insert At End "NSAX_PE_GETPUBLICID:"&&S_CSTRING(P).S TO C_LERR
 NSAX_DISPOSE P
 EndIf
 P = NSAX_PE_GETMESSAGE(E)
 If P
  Insert At End "NSAX_PE_GETMESSAGE:"&&S_CSTRING(P).S TO C_LERR
 NSAX_DISPOSE P
 EndIf
EndInstruction

FUNCTION SAXH_FATALERROR(POINTER PDATA, POINTER @E) RETURN INT
 Insert At End "SAXH_FATALERROR"&&"Col:"&NSAX_PE_GETCOLUMNNUMBER%(E)&"
Line:"&NSAX_PE_GETLINENUMBER%(E) TO C_LERR
 ErrorMsg(E)
 Return TRUE%
ENDFUNCTION
```

Voir aussi NSAX_CREATEPARSER, SAXH_ERROR%, SAXH_WARNING%, NSAX_PE_*

Fonction SAXH_WARNING% (Librairie NsSaxXml)

Retourne un avertissement.

Syntaxe	**SAXH_WARNING%** *(PDATA, E)*			
Paramètres	PDATA	POINTER	I	contexte donné via NSAX_CREATEPARSER
	E	POINTER	I/O	Handle à utiliser avec les méthodes NSAX_PE_*

Si retourne TRUE%, le système remet à zéro le code d'erreur.
Exemple :

```
Global Control C_LERR
```

```
Instruction ErrorMsg POINTER @E
Local Pointer P
 P = NSAX_PE_GETSYSTEMID(E)
 If P
  Insert At End "NSAX_PE_GETSYSTEMID:"&&S_CSTRING(P).S TO C_LERR
 NSAX_DISPOSE P
 EndIf
 P = NSAX_PE_GETPUBLICID(E)
 If P
  Insert At End "NSAX_PE_GETPUBLICID:"&&S_CSTRING(P).S TO C_LERR
 NSAX_DISPOSE P
 EndIf
 P = NSAX_PE_GETMESSAGE(E)
 If P
  Insert At End "NSAX_PE_GETMESSAGE:"&&S_CSTRING(P).S TO C_LERR
 NSAX_DISPOSE P
 EndIf
EndInstruction

FUNCTION SAXH_WARNING(POINTER PDATA, POINTER @E) RETURN INT
 Message "SAXH_WARNING","Col:"&NSAX_PE_GETCOLUMNNUMBER%(E)&"
Line:"&NSAX_PE_GETLINENUMBER%(E)
 ErrorMsg(E)
 Return TRUE%
ENDFUNCTION
```

Voir aussi NSAX_CREATEPARSER, SAXH_ERROR%, SAXH_FATALERROR%, NSAX_PE_*

LIBRAIRIE NWXSEC

Nat System propose la nouvelle librairie NWXSEC qui permet de sécuriser vos documents XML en autorisant l'utilisation de signatures numériques et la mise en forme canonique des documents XML.

Introduction

La librairie NWXSEC permet de sécuriser vos documents XML en autorisant l'utilisation de signatures numériques et la mise en forme canonique des documents XML.
La signature numérique permet d'authentifier l'auteur d'un document et de garantir son intégrité.
La canonisation consiste à convertir des données vers un format standard unique. Par exemple, des balises peuvent contenir des espaces sans que cela ait un impact sur la syntaxe XML. <Elem> équivaut à < Elem >. Cependant, les algorithmes de comparaison logique utilisés pour la signature numérique ne supportent pas ces différences. Il est donc nécessaire de mettre en forme canonique les documents XML avant de les signer.

Installation

Déclarez NWXSEC.NCL dans les librairies nécessaires au développement de votre application.
Vérifiez que le fichier NWXSEC.DLL est bien dans un des répertoires du PATH sous Windows.

Fichier NWXSEC.NCL

Le verbe de la librairie NWXSEC, décrit ci après, est déclaré dans le fichier texte écrit en NCL, de nom NWXSEC.NCL. Ce fichier peut aussi contenir des verbes complémentaires (API non publique). Vous pouvez donc désirer consulter ce fichier pour avoir la référence exhaustive de la librairie.
Pour consulter le fichier NWXSEC.NCL :

1. Placez-vous dans le répertoire <NATSTAR>\NCL.

<NATSTAR> représente le répertoire que vous avez choisi au moment de l'installation de NatStar. La procédure est identique pour NS-DK et NatWeb.

2. Editez le fichier NWXSEC.NCL avec n'importe quel éditeur de texte.

Référence de la librairie NWXSEC.NCL

Fonction NWX_NODE_CANONICAL_OUTPUT (Librairie NWXSEC)

Effectue la transformation canonique d'un nœud donné.

Syntaxe	**NWX_NODE_CANONICAL_OUTPUT** *(pNode, printComments, exclusive)*

Paramètres	pNode	POINTER	I/O	pointeur du nœud
	printComments	INT(1)	I	indique si l'on garde les commentaires
	exclusive	INT(1)	I	indique si l'on utilise la canonisation exclusive
Valeur retournée	DynStr contenant le XML transformé sous forme canonique.			

Il faut utiliser NWX_PARSER_SETDONAMESPACES pour activer la prise en compte des espaces de nommage.
Exemple :

```
local pointer pDoc
local pointer pParser
local sValue$(500)
local ok%
local dynstr ds
local pointer pRootNode
local pointer pNode
delete from MLE1
ok% = NWX_INITIALIZE%
ok% = NWXSEC_INITIALIZE%
pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
NWX_PARSER_SETCREATEENTREFNODES pParser, TRUE%
NWX_PARSER_SETDONAMESPACES pParser,true%
sValue$ = '<?xml version="1.0" encoding="ISO-8859-1"?><n1:racine
xmlns:n1="http://abc.def.com/n1"
xmlns:n2="http://abc.def.com/n2"><n2:element><n1:element Z="7"
A="8"><n2:truc>xéz<!--***--
></n2:truc></n1:element></n2:element></n1:racine>'
ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)
pRootNode = NWX_DOCUMENT_GETROOT (pDoc)
pNode = NWX_NODE_GETCHILD (pRootNode)
WHILE pnode <> 0
ok% = NWX_NODE_OUTPUT%(pNode,@svalue$, sizeof svalue$)
INSERT AT END 'origine ->'&sValue$ TO MLE1
ds = NWX_NODE_CANONICAL_OUTPUT(pNode, CK_COMMENTS,CK_EXCLUSIVE)
INSERT AT END 'canon   ->'&ds TO MLE1
pNode = NWX_NODE_GETCHILD (pNode)
ENDWHILE
; dispose  document
NWX_DOCUMENT_DISPOSE pDoc
; free  parser
NWX_PARSER_DISPOSE pParser
NWXSEC_TERMINATE
NWX_TERMINATE
```

Voir aussi NWX_PARSER_SETDONAMESPACES (Librairie NWXML), NWX_DOCUMENT_CANONICAL_OUTPUT, NWX_NODE_CANONICAL_OUTPUTFILE%, NWX_DOCUMENT_CANONICAL_OUTPUTFILE%

Fonction NWX_NODE_CANONICAL_OUTPUTFILE% (Librairie NWXSEC)

Effectue la transformation canonique d'un nœud donné, et stocke le résultat dans un fichier.

Syntaxe	**NWX_NODE_CANONICAL_OUTPUTFILE%** *(pNode, FileName$, printComments, exclusive)*			
Paramètres	pNode	POINTER	I/O	pointeur du nœud
	FileName$	CSTRING	I	nom du fichier de sortie
	printComments	INT(1)	I	indique si l'on garde les commentaires
	exclusive	INT(1)	I	indique si l'on utilise la canonisation exclusive
Valeur retournée	TRUE% si succès, FALSE% sinon			

Il faut utiliser NWX_PARSER_SETDONAMESPACES pour activer la prise en compte des espaces de nommage.

Exemple :

```
local pointer pDoc
local pointer pParser
local sValue$(500)
local ok%
local pointer pRootNode, pointer pNode
local cstring outDir
local index%
ok% = NWX_INITIALIZE%
ok% = NWXSEC_INITIALIZE%
pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
NWX_PARSER_SETCREATEENTREFNODES pParser, TRUE%
NWX_PARSER_SETDONAMESPACES pParser,true%
sValue$ = '<?xml version="1.0" encoding="ISO-8859-1"?><n1:racine
xmlns:n1="http://abc.def.com/n1"
xmlns:n2="http://abc.def.com/n2"><n2:element><n1:element Z="7"
A="8"><n2:truc>xéz<!--***--
></n2:truc></n1:element></n2:element></n1:racine>'
ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)
outDir = e_outdir
pRootNode = NWX_DOCUMENT_GETROOT (pDoc)
pNode = NWX_NODE_GETCHILD (pRootNode)
index% = 0
WHILE pnode <> 0
IF NWX_NODE_CANONICAL_OUTPUTFILE%(pNode, outDir &
'\scanonNode'&index%&'.xml', CK_COMMENTS,CK_EXCLUSIVE)
message "CANON -> FILE", "OK"
ELSE
message "CANON -> FILE", "Failed"
ENDIF
pNode = NWX_NODE_GETCHILD (pNode)
index% = index% + 1
```

```
ENDWHILE
; dispose  document
NWX_DOCUMENT_DISPOSE pDoc
; free  parser
NWX_PARSER_DISPOSE pParser
NWXSEC_TERMINATE
NWX_TERMINATE
```

Voir aussi NWX_PARSER_SETDONAMESPACES (Librairie NWXML), NWX_DOCUMENT_CANONICAL_OUTPUT, NWX_NODE_CANONICAL_OUTPUT, NWX_DOCUMENT_CANONICAL_OUTPUTFILE%

Fonction NWX_DOCUMENT_CANONICAL_OUTPUT (Librairie NWXSEC)

Effectue la transformation canonique d'un document.

Syntaxe	**NWX_DOCUMENT_CANONICAL_OUTPUT** *(pDoc, printComments, exclusive)*			
Paramètres	pDoc	POINTER	I/O	pointeur du document
	printComments	INT(1)	I	indique si l'on garde les commentaires
	exclusive	INT(1)	I	indique si l'on utilise la canonisation exclusive
Valeur retournée	DynStr contenant le XML transformé sous forme canonique			

Il faut utiliser NWX_PARSER_SETDONAMESPACES pour activer la prise en compte des espaces de nommage.

Exemple :

```
local pointer pDoc
local pointer pParser
local sValue$(500)
local ok%
local dynstr ds
delete from MLE1
ok% = NWX_INITIALIZE%
ok% = NWXSEC_INITIALIZE%
pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
NWX_PARSER_SETCREATEENTREFNODES pParser, TRUE%
NWX_PARSER_SETDONAMESPACES pParser,true%
sValue$ = '<?xml version="1.0" encoding="ISO-8859-1"?><n1:racine
xmlns:n1="http://abc.def.com/n1"
xmlns:n2="http://abc.def.com/n2"><n2:element><n1:element Z="7"
A="8"><n2:truc>xéz<!--***--
></n2:truc></n1:element></n2:element></n1:racine>'
ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)
ds = NWX_DOCUMENT_CANONICAL_OUTPUT(pDoc, CK_COMMENTS,CK_EXCLUSIVE)
MLE1.TEXT = ds
; dispose document
NWX_DOCUMENT_DISPOSE pDoc
```

```
; free parser
NWX_PARSER_DISPOSE pParser
NWXSEC_TERMINATE
NWX_TERMINATE
```

Voir aussi NWX_PARSER_SETDONAMESPACES (Librairie NWXML), NWX_NODE_CANONICAL_OUTPUT, NWX_NODE_CANONICAL_OUTPUTFILE%, NWX_DOCUMENT_CANONICAL_OUTPUTFILE%

Fonction NWX_DOCUMENT_CANONICAL_OUTPUTFILE% (Librairie NWXSEC)

Effectue la transformation canonique d'un document, et stocke le résultat dans un fichier.

Syntaxe	**NWX_DOCUMENT_CANONICAL_OUTPUTFILE%** *(pDoc, FileName$, printComments, exclusive)*			
Paramètres	pDoc	POINTER	I/O	pointeur du document
	FileName$	CSTRING	I	nom du fichier de sortie
	printComments	INT(1)	I	indique si l'on garde les commentaires
	exclusive	INT(1)	I	indique si l'on utilise la canonisation exclusive
Valeur retournée	TRUE% si succès, FALSE% sinon			

Il faut utiliser NWX_PARSER_SETDONAMESPACES pour activer la prise en compte des espaces de nommage.
Exemple :

```
local pointer pDoc, pointer pParser
local sValue$(500)
local ok%
local cstring outDir
ok% = NWX_INITIALIZE%
ok% = NWXSEC_INITIALIZE%
pParser = NWX_PARSER_NEW
pDoc = NWX_DOCUMENT_NEW("XMLdoc")
NWX_PARSER_SETCREATEENTREFNODES pParser, TRUE%
NWX_PARSER_SETDONAMESPACES pParser,true%
sValue$ = '<?xml version="1.0" encoding="ISO-8859-1"?><n1:racine
xmlns:n1="http://abc.def.com/n1"
xmlns:n2="http://abc.def.com/n2"><n2:element><n1:element Z="7"
A="8"><n2:truc>xéz<!--***--
></n2:truc></n1:element></n2:element></n1:racine>'
ok% = NWX_DOCUMENT_LOAD%(pDoc, sValue$, pParser)
outDir = e_outdir
IF NWX_DOCUMENT_CANONICAL_OUTPUTFILE%(pDoc, outDir & '\scanon.xml',
CK_COMMENTS,CK_EXCLUSIVE)
message "CANON -> FILE", "OK"
```

```
ELSE
message "CANON -> FILE", "Failed"
ENDIF
; dispose  document
NWX_DOCUMENT_DISPOSE pDoc
; free  parser
NWX_PARSER_DISPOSE pParser
NWXSEC_TERMINATE
NWX_TERMINATE
```

Voir aussi NWX_PARSER_SETDONAMESPACES (Librairie NWXML), NWX_NODE_CANONICAL_OUTPUT, NWX_NODE_CANONICAL_OUTPUTFILE%, NWX_DOCUMENT_CANONICAL_OUTPUT

LIBRAIRIE NSUNICOD

Ce chapitre présente l'utilisation de l'UTF-8 dans NatStar.

Les pages de codes

ASCII

L'American Standard Code for Information Exchange ou ASCII est un codage utilisant 7 ou 8 bits et qui associe des valeurs numériques à 256 caractères au plus. Ce codage représente des lettres, des chiffres, des signes de ponctuation, des caractères de commande, et d'autres symboles. L' ASCII a été développée en 1968 pour normaliser la transmission de données entre un matériel et des logiciels disparates. L'ASCII est implémenté dans la plupart des mini-ordinateurs et tous les ordinateurs individuels.
Le code ASCII étendu est subdivisé en deux parties :

- le code ASCII proprement dit, qui inclut les 128 premiers caractères alphanumériques,
- IBM Character Set comprenant 128 autres caractères pour les graphiques et les dessins de lignes.

ANSI

La page de code ANSI est la représentation de caractère la plus répandue employée par les ordinateurs individuels (Windows 95/98). La représentation ANSI emploie seulement un octet simple pour représenter chaque caractère, elle est limitée à un maximum de 256 codes de caractère et de ponctuation. Bien que ce soit adapté pour le français, la page de code ANSI ne sied pas à beaucoup d'autres langues. Les premiers 127 caractères de cette page de codes sont les mêmes en ASCII.

UNICODE

L'Unicode est un codage défini par l'organisme de normalisation international (ISO). Il permet d'intégrer tous les alphabets en même temps.
Il en existe plusieurs implémentations. UCS-2 et UTF-8 sont deux façons courantes de stocker les configurations de bits qui représentent des caractères Unicode.
L'UCS-2 définit un nombre dans l'intervalle de 0 à 65.535 (216 - 1) pour tout caractère ou symbole de chaque langue (plus quelques espaces vides pour les évolutions futures).
Microsoft Windows NT, SQL-Server et SQL-Server ODBC représentent en interne les données avec le format UCS-2.

UTF-8

L'UTF-8 est une page de code qui emploie une chaîne d'octets pour représenter une chaîne de texte ASCII (codes < = 127 Hexa 7F). Ces caractères demeurent sans changement sur un octet simple, les caractères Unicode -UCS2-128 (ou hexa 80) à 2047 (ou hexa 7ff) (y compris le latin, le grec, le cyrillique, l'hébreu, et l'arabe) est converti en une séquence de 2-octets, et 2048 (ou hexa 800) à 65535 (ou hexa ffff) (chinois, japonais, coréen, et autres) deviennent une séquence de 3-octets. L'avantage est que la plupart des textes ASCII restent inchangés et presque tous les éditeurs peuvent le lire, y compris l'éditeur NCL.

Cependant, l'UTF-8 n'est pas un codage valide pour des arguments de ligne de commande pour Windows NT 4.0 ou 5.0, et il n'est pas supporté sous Windows 95/98.

Exemple :
La Lettre A en ANSI ou ASCII est représenté par 41 en Hexa (65 Déc). En UCS-2 elle est représentée sur 2 octets 41 pour le poids faible et 00 pour le poids fort.
En UTF-8 la lettre A est représenté par 65 (ou Hexa 41) sur un octet comme pour l'ASCII ou l'ANSI.

Les pages de codes étendues

Restrictions

NatStar et NS-DK sous les systèmes d'exploitations Windows NT, 2000 et XP supporte nativement l'UTF-8 à partir de la version 2.61 et l'UCS-2 avec quelques restrictions.
Il est capable de lire ou écrire un fichier dans la page de codes UTF-8, transformer une chaîne de caractères UTF-8 en ANSI ou en UCS-2 et inversement. Les fenêtres et contrôles NatStar acceptent des chaînes UTF-8, et il vous est possible de saisir des chaînes UTF-8 littérales dans l'éditeur NCL sans qu'elles apparaissent correctement dans ce dernier.
Il n'est pas possible de lire des fichiers UCS-2 ni d'afficher des chaînes UCS-2 dans les fenêtres et contrôles.
Comme l'éditeur de source ne gère pas l'Unicode (il affiche les caractères Unicode encodés en UTF-8 autres que les caractères ASCII/ANSI dont le code est inférieur à 128 comme une séquence de plusieurs caractères), un nouvel élément du menu contextuel (Modify Unicode text...) permet, quand NS-Design ou NatStar est lancé en mode Unicode, d'éditer la sélection (il faut faire le clic bouton droit dans la zone sélectionnée pour la modifier) dans un Entry Field (qui supporte correctement l'Unicode) à condition qu'elle ne soit pas à cheval sur plusieurs lignes et qu'elle ne dépasse pas 255 caractères. Si le début et/ou la fin de la sélection commence/se termine à l'intérieur d'une tabulation, celle(s)-ci n'en fera/feront pas partie, par contre, si elle commence/se termine au milieu d'un caractère UTF-8 composite, celui/ceux-ci sera/seront complètement pris en compte. Après validation du dialogue de saisie, la sélection est mise à jour pour inclure tout le texte modifié. S'il n'y a pas de sélection lors de l'ouverture du dialogue de saisie, le texte saisi est inséré (et n'est pas sélectionné), mais si le curseur était au milieu d'un caractère UTF-8 composite, l'insertion est faite avant le premier octet de celui-ci.

Un raccourci clavier permet d'ouvrir le dialogue de saisie: [Ctrl]+*.

Mise en œuvre

Pour créer une application supportant l'UTF-8 avec NatStar et NS-DK.

1. Positionner UnicodeWndClass à True sous la rubrique [Miscellaneous] du fichier NSLIB.INI.

```
[Miscellaneous]
UnicodeWndClass=True
```

Si UnicodeWndClass n'est pas fixé à True, on ne peut pas avoir de caractères étendus dans le titre des fenêtres.

2. Démarrer NatStar ou NS-DK avec le paramètre /UNICODE ou /U sur un WorkSpace vierge.
3. Importer un fichier export ou démarrer l'application à partir de 0.
4. Mettre les fichiers NSUNICOD.NCL et NSGRAPH.NCL dans les services.

Le dossier GLOB est défini en mode Unicode (ou non) à sa création selon les paramètres de NatStar ou NS-DK. Ce dossier GLOB ne pourra pas être modifié par la suite. Pour vérifier qu'on est en mode Unicode, activez le menu Help/About, la version doit indiquer U1.30.

Référence de la librairie NSUNICOD

Fonction AnsiToUTF8 (Librairie NSUNICOD)

Convertit une chaîne ANSI (ISO 8859/1) à un octet par caractère en chaîne UTF-8.

Syntaxe	**AnsiToUTF8** *(Ansi$, WideChCnt%, pUTF8Str, UTF8Size%, bForcedTerm%)*			
Paramètres	Ansi$	CSTRING	I	la chaîne ANSI, peut être > à 255 caractères
	WideChCnt%	INTEGER	I	nombre de caractères à convertir de la chaîne ANSI ou -1 pour la longueur

				totale de la chaîne.
	pUTF8str	POINTER	I	adresse de la chaîne UTF8
	UTF8Size%	INTEGER	I	taille maximale en octet de la chaîne UTF8 de sortie.
	bForcedTerm%	INT(1)	I	True force le dernier caractère de la chaîne UTF8 à 0 binaire (\0) même si la chaîne est trop courte, False sinon.
Valeur retournée	INTEGER taille de la chaîne retournée en octet : le minimum de (WideChCnt%, UTF8Size%).			

Exemple :

```
local utf8$, size%
fill @utf8$, sizeof utf8$, 0
size% = AnsiToUTF8 ("Antoine", -1, @utf8$, 10, TRUE%)
INSERT AT END "UTF8 string of size "&&size% to LB1 ;8
INSERT AT END utf8$ to LB1 ;Antoine
fill @utf8$, sizeof utf8$, 0
size% = AnsiToUTF8 ("Antoine", -1, @utf8$, 10, FALSE%)
INSERT AT END "UTF8 string of size "&&size% to LB1;8
INSERT AT END utf8$ to LB1;Antoine
fill @utf8$, sizeof utf8$, 0
size% = AnsiToUTF8 ("Antoine", 6, @utf8$, 10, FALSE%)
INSERT AT END "UTF8 string of size "&&size% to LB1 ;6
INSERT AT END utf8$ to LB1;Antoin
fill @utf8$, sizeof utf8$, 0
size% = AnsiToUTF8 ("Antoine", 4, @utf8$, 10, TRUE%)
INSERT AT END "UTF8 string of size "&&size% to LB1 ; 4
```

```
INSERT AT END utf8$ to LB1 ; Anto
fill @utf8$, sizeof utf8$, 0
size% = AnsiToUTF8 ("Antoine", -1, @utf8$, 7, TRUE%)
INSERT AT END "UTF8 string of size "&&size% to LB1 ; 7
INSERT AT END utf8$ to LB1 ; Antoin
; here the 'e' is overwritten by the Binary 0
fill @utf8$, sizeof utf8$, 0
size% = AnsiToUTF8 ("Antoine", -1, @utf8$, 7, FALSE%)
INSERT AT END "UTF8 string of size "&&size% to LB1 ; 7
INSERT AT END utf8$ to LB1 ; Antoine
```

***Voir aussi** UTF8ToUnicode, UnicodeToUTF8, CharsToUTF8*

Fonction AnsiToUTF8$ (Librairie NSUNICOD)

Convertit une chaîne ANSI (ISO 8859/1) à un octet par caractère en chaîne UTF-8 de taille maximale 255 caractères + le 0 binaire terminal.

Syntaxe	**AnsiToUTF8$** *(Ansi$)*			
Paramètre	Ansi$	CSTRING	I	la chaîne ANSI
Valeur retournée	CSTRING la chaîne UTF8.			

Exemple :

```
local utf8$ , Ansi$
ansi$ = "&éàçêîêâôoe"
INSERT AT END "Length ansi$"&& length ansi$ to LB1 ; 11
utf8$ = AnsiToUTF8$ (ansi$)
INSERT AT END utf8$&&"Length ="&& length utf8$ to LB1 ; 19
```

***Voir aussi** AnsiToUTF8*

Fonction CharsToUTF8 (Librairie NSUNICOD)

Convertit une chaîne de caractère à un octet par caractères en chaîne UTF-8. La conversion se fait en utilisant le code ASCII de chaque caractère comme indice de la table de conversion passée en paramètre. Autrement dit si on met le code Unicode de la lettre ‘A’ à la 65ème position de la table le ‘A’ sera converti par un ‘A’.

Syntaxe	**CharsToUTF8** *(pWideChStr, WideChCnt%, pUTF8Str, UTF8Size%, bForcedTerm%, table)*			
Paramètres	pWideChStr	POINTER	I	adresse de la chaîne de caractères.
	WideChCnt%	INTEGER	I	nombre de caractères dans la chaîne

				pointée par pWideChStr ou –1 pour la longueur totale de la chaîne pointée.
	pUTF8str	POINTER	I	adresse de la chaîne UTF8
	UTF8Size%	INTEGER	I	taille en octet de la chaîne UTF8 de sortie.
	bForcedTerm%	INT(1)	I	TRUE force le dernier caractère de la chaîne UTF8 à 0 binaire (\0) même si la chaîne est trop courte, FALSE sinon.
	table	INT(2)[256]	I	table de translation en Unicode des caractères de la chaîne en entrée. Au caractère n de la chaîne doit correspondre le nieme élément du tableau.
Valeur retournée	INTEGER taille de la chaîne retournée en octet.			

Cette fonction peut être très utile pour implémenter un code page non fourni.
Exemple :

```
local utf8$, size%, I% ,MyCodePage$, Ansi$
local INT unicode(2)[256]
;I fill my unicode table which has to be in this format INT(2)[256]
fill @unicode, sizeof unicode, 0
utf8$ = ' !"#$%&'
utf8$ = utf8$&"'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPKRSTUVWXYZ"
utf8$ = utf8$&"[\]^_`abcdefghijklmnopqrstuvwxyz{|}";ASCII char set
size% = UTF8ToUnicode (@UTF8$, -1, @unicode, 256, FALSE%)

;Now i use the unicode I filled previously to convert my Ansi string into
My Code Page
Ansi$ = "ABCDEFGH"
size% = CharsToUTF8(@Ansi$,-1,@MyCodePage$,10, TRUE%, unicode)
INSERT AT END "(@utf8$,-1,@MyCodePage$,10, TRUE%,unicode) string
size"&&size% to LB1
INSERT AT END "MyCodePage$"&&MyCodePage$ to LB1;ABCDEFGH
;(because I reproduced the ASCII char set)
```

***Voir aussi** UTF8ToUnicode, UnicodeToUTF8, AnsiToUTF8*

Fonction UnicodeToUTF8 (Librairie NSUNICOD)

Convertit une chaîne Unicode à deux octets par caractère (UCS-2) en chaîne UTF-8.

Syntaxe	**UnicodeToUTF8** *(pWideChStr, WideChCnt%, pUTF8Str, UTF8Size%, bForcedTerm%)*			
Paramètres	pWideChStr	POINTER	I	adresse de la chaîne Unicode
	WideChCnt%	INTEGER	I	nombre de caractères dans la chaîne pointée par pWideChStr ou –1 pour la longueur totale de la chaîne pointée.
	pUTF8str	POINTER	I	adresse de la chaîne UTF8
	UTF8Size%	INTEGER	I	taille en octet de la chaîne

				UTF8 de sortie.
	bForcedTerm%	INT(1)	I	True force le dernier caractère de la chaîne UTF8 à 0 binaire (\0) même si la chaîne est trop courte, False sinon.
Valeur retournée	INTEGER taille de la chaîne retournée en octet			

Exemple :

```
local utf8$, size%, I% ,oututf8$
local INT unicode(2)[100]
utf8$ = "Antoine"
fill @unicode, sizeof unicode, 0
size% = UTF8ToUnicode (@UTF8$, -1, @unicode, 10, TRUE%)
size% = UnicodeToUTF8 (@unicode,-1,@oututf8$,10, TRUE%)
INSERT AT END " (@unicode,-1,@oututf8$,10, TRUE%) UTF8 string size"&&size%
to LB1 ;8
INSERT AT END "oututf8$"&&oututf8$ to LB1 ;
utf8$ = ENTRY1 ;7 caracteres Arabes
fill @unicode, sizeof unicode, 0
fill @oututf8$, sizeof oututf8$, 0
size% = UTF8ToUnicode (@UTF8$, -1, @unicode, 10, TRUE%)
size% = UnicodeToUTF8 (@unicode,-1,@oututf8$,10, TRUE%)
INSERT AT END " (@unicode,-1,@oututf8$,10, TRUE%) UTF8 string size"&&size%
to LB1 ;9
INSERT AT END "oututf8$"&&oututf8$ to LB1 ;
fill @oututf8$, sizeof oututf8$, 0
size% = UnicodeToUTF8 (@unicode,-1,@oututf8$,13, TRUE%)
INSERT AT END "(@unicode,-1,@oututf8$,13, TRUE%) UTF8 string size"&&size%
to LB1 ;12
INSERT AT END "oututf8$"&&oututf8$ to LB1 ;
fill @oututf8$, sizeof oututf8$, 0
size% = UnicodeToUTF8 (@unicode,-1,@oututf8$,13, FALSE%)
INSERT AT END "(@unicode,-1,@oututf8$,13, FALSE%) UTF8 string size"&&size%
to LB1 ;12
INSERT AT END "oututf8$"&&oututf8$ to LB1 ;
size% = UnicodeToUTF8 (@unicode,-1,@oututf8$,14, TRUE%)
INSERT AT END "(@unicode,-1,@oututf8$,14, TRUE%) UTF8 string size"&&size%
to LB1 ;13
INSERT AT END "oututf8$"&&oututf8$ to LB1 ;
fill @oututf8$, sizeof oututf8$, 0
size% = UnicodeToUTF8 (@unicode,-1,@oututf8$,14, FALSE%)
INSERT AT END "(@unicode,-1,@oututf8$,14, FALSE%) UTF8 string size"&&size%
to LB1 ;14
INSERT AT END "oututf8$"&&oututf8$ to LB1 ;
```

```
fill @oututf8$, sizeof oututf8$, 0
size% = UnicodeToUTF8 (@unicode,-1,@oututf8$,15, FALSE%)
INSERT AT END "(@unicode,-1,@oututf8$,15, FALSE%) UTF8 string size"&&size%
to LB1 ;15
INSERT AT END "oututf8$"&&oututf8$ to LB1 ;
fill @oututf8$, sizeof oututf8$, 0
size% = UnicodeToUTF8 (@unicode,-1,@oututf8$,15, TRUE%)
INSERT AT END "(@unicode,-1,@oututf8$,15, TRUE%) UTF8 string size"&&size%
to LB1 ;15
INSERT AT END "oututf8$"&&oututf8$ to LB1 ;
```

***Voir aussi** UTF8ToUnicode, AnsiToUTF8, CharsToUTF8*

Fonction UTF8ChCnt (Librairie NSUNICOD)

Retourne le nombre de caractères UTF8 entre le début d'une chaîne UTF8 et une position passée en paramètre.

Syntaxe	**UTF8ChCnt** *(UTF8$, ByteCnt%)*			
Paramètres	UTF8$	POINTER	I	chaîne UTF8.
	ByteCnt%	INTEGER	I	le nombre d'octets à compter.
Valeur retournée	INTEGER			

Le nombre de caractères UTF8 entre le début de la chaîne et la position ByteCnt% si la position tombe à l'intérieur d'un caractère multi-octets, ce dernier n'est pas compté.
Exemple :

```
local utf8$ , i%
UTF8$ = ENTRY1 ; 7 characters 2 bytes each
i% = UTF8ChCnt (UTF8$, 5) ; before the 5th byte
INSERT AT END UTF8$&&"Number of characters before byte 5 ="&& i% to LB1 ; 2
```

***Voir aussi** UTF8Strlen, UTF8StrEnd*

Fonction UTF8Copy (Librairie NSUNICOD)

Si Start% est > 0 garde la partie limitée par Start% et la fin de la chaîne d'une chaîne UTF8 pointée par pWideChStr si Len% le permet sinon tronque la partie en trop.
Si Start% est < 0 alors Start% indique le rang du 1er caractère à partir de la fin et Len% les caractères avant ce dernier. Ne pas compter un caractère de plus dans Len% pour le \0 dans ce cas-là.

Syntaxe	**UTF8Copy** *(pWideChStr, Start%,Len%)*			
Paramètres	pWideChStr	POINTER	I	adresse de la chaîne UTF8.

	Start%	INTEGER	I	le rang du 1er caractère à garder dans la chaîne pointée par pWideChStr. Les indices commencent à partir de 1. Pour les valeurs négatives, Start% indique la position à partir de la fin.
	Len%	INTEGER	I	le nombre de caractères que contiendra la chaîne resultante. Le \0 est compté si Start% est > 0.
Valeur retournée	POINTER Le pointeur retourné est égal à pWideChStr mais la chaîne est modifiée.			

Exemple :

```
local cstring utf8$ ,INTEGER i% ,POINTER H%, K%
Local cstring SubStr$, cstring SubStr2$
Local POINTER pH%
UTF8$ = "Nat System"
pH%=@UTF8$
k% = 6
i% = 1
h% = UTF8Copy (pH%,i%, K%); copy$ ; 5 + \0 = 6
INSERT AT END "UTF8$"&&UTF8$&& "Length"&&Length UTF8$ to LB1; Cogni
UTF8$ = "Nat System"
pH%=@UTF8$
k% = 5
i% = 6
h% = UTF8Copy (pH%,i%, K%); copy$
INSERT AT END "UTF8$"&&UTF8$&& "Length"&&Length UTF8$ to LB1; Case
```

```
UTF8$ = "Nat System"
pH%=@UTF8$
k% = 6
i% = -5
h% = UTF8Copy (pH%,i%, K%); copy$
INSERT AT END "UTF8$"&&UTF8$&& "Length"&&Length UTF8$ to LB1; Cogni
UTF8$ = "Nat System"
pH%=@UTF8$
k% = 1
i% = -5
h% = UTF8Copy (pH%,i%, K%); copy$
INSERT AT END "UTF8$"&&UTF8$&& "Length"&&Length UTF8$ to LB1; i
UTF8$ = "Nat System"
pH%=@UTF8$
k% = 1
i% = 5
h% = UTF8Copy (pH%,i%, K%); copy$
INSERT AT END "UTF8$"&&UTF8$&& "Length"&&Length UTF8$ to LB1
; empty string "" because Start is > 0
```

Voir aussi Copy$, UTF8CopyEx, UTF8Insert, UTF8Delete

Fonction UTF8CopyEx (Librairie NSUNICOD)

Copie dans la chaîne de destination l'intervalle déterminé par Start% et Len% de la chaîne source. Maxsize% est diminué de Len% également.

Syntaxe	**UTF8CopyEx** *(pszUTF8Dest, pszUTF8Src, Start%,Len%, MaxSize%)*			
Paramètres	pszUTF8Dest	POINTER	I	adresse de la chaîne UTF8 de destination.
	pszUTF8Src	POINTER	I	adresse de la chaîne UTF8 source.
	Start%	INTEGER	I	le rang du 1er caractère à copier dans la chaîne pointée par pszUTF8Dest. Pour les valeurs négatives, Start% indique la position à partir de la fin et Len% le

				nombre de caractères à inclure avant cette position.
	Len%	INTEGER	I	le nombre de caractères à copier + un \0 terminal si Start% est > 0. Pour les valeurs négatives, Len% indique le nombre de caractères à inclure avant la position Start%.
	MaxSize%	INTEGER	I/O	après retour Maxsize% sera diminué du nombre de caractères copiés. Le \0 est compté dans Maxsize% (Maxsize%>0)
Valeur retournée	POINTER pointeur sur le 0 binaire terminal (\0) de la chaîne destination.			

Les appels aux fonctions UTF8xxxEx peuvent être imbriqués en utilisant le pointeur retourné par n'importe quelle fonction d'entre elles comme pszUTF8Dest pour la suivante. On initialise le paramètre MaxSize% avec la taille en octet de la chaîne destination qu'on aimerait avoir en finale (celle pointé par pszUTF8Dest) et on le passe à tous les appels de fonctions UTF8xxxEx imbriqués.

Exemple :

```
local cstring utf8$(17) , INTEGER i% , POINTER H% , INTEGER Maxsize%
Local cstring Dest$
Local POINTER HSrc%, POINTER HDest%
Maxsize% = 9
UTF8$ = "Nat System"
HDest% = @Dest$
HSrc%=@UTF8$
INSERT AT END "UTF8$"&&UTF8$ to LB1; Nat System
h% = UTF8CopyEx (HDest%, HSrc%, 1, 6, Maxsize%)
INSERT AT END "Dest$ "&&Dest$ && "Maxsize ="&&Maxsize% to LB1; Cogni
```

```
UTF8$ = "Nat System Inc"
Maxsize% = 13
h% = UTF8CopyEx (HDest%, HSrc%, 6, 5, Maxsize%)
INSERT AT END "Dest$ "&&Dest$ && "Maxsize ="&&Maxsize% to LB1; Case
```

Voir aussi Copy$, UTF8Copy, UTF8InsertEx, UTF8DeleteEx

Fonction UTF8Delete (Librairie NSUNICOD)

Supprime le nombre de caractères (contenu dans le paramètre Len%) d'une chaîne UTF-8 à partir du caractère de rang Start%. Le rang peut être négatif, dans ce cas la position est calculée à partir de la fin.

Syntaxe	**UTF8Delete** *(pWideChStr, Start%,Len%)*			
Paramètres	pWideChStr	POINTER	I	adresse de la chaîne UTF8.
	Start%	INTEGER	I	le rang du 1er caractère à supprimer dans la chaîne pointée par pWideChStr. Les indices commencent à partir de 1. Pour les valeurs négatives Start% indique la position à partir de la fin et Len% le nombre de caractères à supprimer avant cette position.
	Len%	INTEGER	I	le nombre de caractères à supprimer si Start% est > 0 prévoir un caractère de plus pour \0.
Valeur retournée	POINTER le pointeur retourné est égal à pWideChStr mais la chaîne est modifiée.			

Exemple :

```
local cstring utf8$(17) ,INTEGER i% ,POINTER H%, K%
Local POINTER pH%
UTF8$ = "Nat System Inc"
pH%=@UTF8$
i% = 1
k% = 10 ; 9 char + 1 for the \0
h% = UTF8Delete (pH%,i%, k%)
INSERT AT END "UTF8$"&&UTF8$ to LB1; Inc
UTF8$ = "Nat System Inc"
pH%=@UTF8$
i% = -1
k% = 3 ; 3 char no need to count the \0 in this case start <0
h% = UTF8Delete (pH%,i%, k%)
INSERT AT END "UTF8$"&&UTF8$ to LB1; Inc
```

Voir aussi Delete$, UTF8Copy, UTF8Insert, UTF8DeleteEx

Fonction UTF8DeleteEx (Librairie NSUNICOD)

Copie dans la destination la chaîne source en :

- retirant l'intervalle déterminé par Start% et Len%,
- vérifiant que la chaîne de destination ne dépasse pas Maxsize%.

Syntaxe	**UTF8DeleteEx** *(pszUTF8Dest, pszUTF8Src, Start%,Len%, MaxSize%)*			
Paramètres	pszUTF8Dest	POINTER	I	adresse de la chaîne UTF8 de destination.
	pszUTF8Src	POINTER	I	adresse de la chaîne UTF8 source.
	Start%	INTEGER	I	le rang du 1er caractère à ne pas inclure dans la chaîne pointée par pszUTF8Dest. Pour les valeurs négatives, Start% indique la position à partir de la fin.
	Len%	INTEGER	I	le nombre de caractères à ne pas inclure de la chaîne pointé par pszUTF8Src. Il faut prévoir un caractère de plus pour le \0 quand start% est >0. Pour les valeurs négatives, Len% indique le nombre de caractères à ne pas inclure

				avant la position Start%.
	MaxSize%	INTEGER	I/O	la taille maximale en octets de la chaîne destination. Cette taille est mise à jour en sortie et diminuée de la taille en octets des caractères copiés.
Valeur retournée	POINTER Pointeur sur le 0 binaire terminal (\0) de la chaîne destination.			

Les appels aux fonctions UTF8xxxEx peuvent être imbriqués en utilisant le pointeur retourné par n'importe quelle fonction d'entre elles comme pszUTF8Dest pour la suivante. On initialise le paramètre MaxSize% avec la taille en octet de la chaîne destination qu'on aimerait avoir en finale (celle pointé par pszUTF8Dest) et on le passe à tous les appels de fonctions UTF8xxxEx imbriqués. Voir l'exemple 2 ci-dessous.

Exemple 1 :

```
local cstring utf8$(17) , INTEGER i% , POINTER H% , INTEGER Maxsize%
Local cstring SubStr$
Local POINTER HSrc%, POINTER HDest%
Substr$ =""
HDest% = @Substr$
; 012345678901234567890123456789012345678901234567890
UTF8$ = "Nat System Inc" ;13 characters + binary 0 = 14
HSrc%=@UTF8$
i% = 10
Maxsize%=20 ; for example
h% = UTF8DeleteEx (HDest%, HSrc%, i%, 5, Maxsize%);4 characters " Inc" +
Binary 0 = 5
INSERT AT END "Destination"&&Substr$ && "Maxsize"&&Maxsize% to LB1
; Nat System ; Maxsize 11
UTF8$ = "Nat System Inc" ;13 characters + binary 0 = 14
HSrc%=@UTF8$
i% = -1
Maxsize%=15 ; for example
; 4 characters " Inc" because start is < 0 we do not count the closing
binary 0
h% = UTF8DeleteEx (HDest%, HSrc%, i%, 4, Maxsize%)
; Nat System ; Maxsize 6
INSERT AT END "Destination"&&Substr$ && "Maxsize"&&Maxsize% to LB1
```

Exemple 2 :

```
local cstring utf8$(17) , INTEGER i% , POINTER H% , INTEGER Maxsize%
Local cstring Dest$, End$, IT$
Local POINTER HSrc%, POINTER HDest% ,POINTER HEnd% ,POINTER HIT%
It$ = "an IT Cie "
UTF8$ = "Nat System is "
End$ = "specialized in Web applications"
Dest$ = ""
HDest% = @Dest$
HSrc%=@UTF8$
HIT%=@IT$
HEnd%=@End$
Maxsize% = 42
h% = UTF8DeleteEx(UTF8InsertEx (UTF8InsertEx (HDest%, HSrc%,Maxsize%, 1,
14), \ HIT%,Maxsize%, 14,11),HEnd%, 19, 14, Maxsize%)
INSERT AT END "Destination"&&Dest$ && "Maxsize"&&Maxsize% to LB1;
;"Nat System is an IT Cie specialized in Web" Maxsize = 1
```

Voir aussi Delete$, UTF8CopyEx, UTF8InsertEx, UTF8Delete

Fonction UTF8FindInterval (Librairie NSUNICOD)

Trouve des pointeurs sur des caractères dans une chaîne UTF-8 à partir du rang et d'un nombre de caractères.

Syntaxe	**UTF8FindInterval** *(pWideChStr, Offset%,Len%)*			
Paramètres	pWideChStr	POINTER	I/O	pointeur sur l'adresse de la chaîne UTF8 ce pointeur sera mis à jour en foction du paramètre Offset%, il est positionné sur le 1er octet du caractère pointé.
	Offset%	INTEGER	I	le rang du caractère à trouver dans la chaîne pointée par pWideChStr. Sa valeur peut être entre 1 (dans ce cas le pointeur retourné est égal à pWideChStr) et UTF8StrLen(pszUTF8) (dans ce cas ou le pointeur retourné pointe vers le \0 terminal), ou entre –1(dans ce cas ou le pointeur retourné pointe vers le \0 terminal) et

				- UTF8StrLen(pszUTF8)-1 (dans ce cas le pointeur retourné est égal à pWideChStr).
	Len%	INTEGER	I	le nombre de caractères
Valeur retournée	POINTER le pointeur retourné est tout au plus (si la chaîne est assez longue). Len% caractères après le caractère d'Offset (ou avant si Offset% < 0).			

Exemple :

```
local cstring utf8$(17) ,INTEGER i% ,POINTER H% , K%
Local cstring SubStr$, cstring SubStr2$
Local POINTER pH%
fill @SubStr$, sizeof SubStr$ , 0
UTF8$ = "Nat System"
pH%=@UTF8$
k% = 6
i% = 1
h% = UTF8FindInterval (pH%,i%, k%)
fill @SubStr$, sizeof SubStr$ , 0
mov h%, @SubStr$, 5 ;
fill @SubStr2$, sizeof SubStr2$ , 0
mov ph%, @SubStr2$, 4
INSERT AT END "SubStr$"&&SubStr$&&"SubStr2$"&&SubStr2$to LB1; Cogni & case
```

***Voir aussi** UTF8FindOffset, UTF8ChCnt, UTF8StrEnd, UTF8StrLen*

Fonction UTF8FindOffset (Librairie NSUNICOD)

Trouve un caractère à partir de son rang dans une chaîne UTF-8. L'index peut être positif (à partir du début de la chaîne) ou négatif (à partir de l'extrémité de la chaîne).

Syntaxe	**UTF8FindOffset** *(pWideChStr, Offset%)*			
Paramètres	pWideChStr	POINTER	I	adresse de la chaîne UTF8
	Offset%	INTEGER	I/O	le rang du caractère à trouver dans la chaîne pointée par pWideChStr. Cet index est mis à jour au rang réel du caractère avant le retour de la fonction si la chaîne est plus petite que le rang demandé. Sa valeur peut être entre 1 (dans ce cas le pointeur retourné est égale à pWideChStr) et UTF8StrLen(pszUTF8) (dans ce

				cas ou le pointeur retourné pointe vers le \0 terminal), ou entre –1(dans ce cas ou le pointeur retourné pointe vers le \0 terminal) et -UTF8StrLen(pszUTF8)-1 (dans ce cas le pointeur retourné est égal à pWideChStr).
Valeur retournée	POINTER pointeur sur le 1er octet du caractère			

1. Si l'Offset% = 2 et pWideChStr est longue d'au moins un caractère, l'Offset reste inchangé et le pointeur retourné est sur le 1er octet du 2ème caractère de la chaîne pWideChStr (sur le \0 si la longueur de la chaîne= 1).
2. Si l'Offset% = 4 et pWideChStr est long de deux caractères, l'Offset est placé à 3, et l'indicateur retourné est sur le 0 terminal de pWideChStr.
3. Si l'Offset% = -2 et pWideChStr est longue d'au moins un caractère, l'Offset reste sans changement et le pointeur retourné est sur le 1er octet du dernier caractère de la chaîne pWideChStr (le premier caractère si longueur = 1).
4. Si l'Offset% = -4 et pWideChStr est long de deux caractères, l'Offset est placé à -3 et l'indicateur retourné est sur le 1er octet du 1er caractère de pWideChStr.

Exemple :

```
local cstring utf8$(17) ,INTEGER i% ,INTEGER j%,POINTER H%
Local cstring Substring$
fill @Substring$, sizeof Substring$ , 0
UTF8$ = ENTRY1 ; 7 characters 2 bytes each
For j% = -8 to 8 step 1
 i% = j%
 h% = UTF8FindOffset (@UTF8$, i%)
 fill @Substring$, sizeof Substring$ , 0
 mov h%, @Substring$, 16 - (i% * 2)
 INSERT AT END "Substring$"&&Substring$&&"i%" && i% to LB1
ENDFOR
```

***Voir aussi** UTF8FindInterval, UTF8ChCnt, UTF8StrEnd, UTF8StrLen*

Fonction UTF8GetCh (Librairie NSUNICOD)

Renvoie le caractère Unicode (UCS-2) correspondant au caractère UTF-8 passé.

Syntaxe	**UTF8GetCh** *(pWideChStr)*			
Paramètre	pWideChStr	POINTER	I	adresse de la chaîne UTF8 contenant le

				caractère à convertir
Valeur retournée	INT(2) le caractère Unicode (UCS-2) correspondant.			

Exemple :

```
local utf8$
local int i%(2)
UTF8$ = "A" ; 1 character
i% = UTF8GetCh (@UTF8$)
INSERT AT END "Low Byte" && LOB i% && "High Byte" && HIB i% to LB1
```

***Voir aussi** UTF8Strlen, UTF8ChCnt, UTF8StrEnd*

Fonction UTF8Insert (Librairie NSUNICOD)

Copie dans la destination, l'intervalle déterminé par Start% et Len% de la chaîne source.

Syntaxe	**UTF8Insert** *(pszUTF8Dest, pszUTF8Src, MaxSize%, Start%, Len%)*			
Paramètres	pszUTF8Dest	POINTER	I	adresse de la chaîne UTF8 de destination .
	pszUTF8Src	POINTER	I	adresse de la chaîne UTF8 source.
	MaxSize%	INTEGER	I	nombre d'octet maximum de la chaîne pszUTF8Dest. Maxsize% n'inclus pas le '\0' terminal.
	Start%	INTEGER	I	le rang du 1er caractère à inclure dans la chaîne pointée par pszUTF8Dest. Pour les valeurs négatives

				start% indique la position à partir de la fin.
	Len%	INTEGER	I	le nombre de caractères à garder de la chaîne destination + le \0 si start est > 0. Si la chaîne destination est plus courte que Len% elle sera complètement remplacée par la chaîne source.
Valeur retournée	POINTER pointeur pszUTF8Dest inchangé.			

1. Les rangs commencent par 1.
2. Si Maxsize% le permet la chaîne entière sera insérée à l'endroit indiqué par Start% sinon elle sera tronquée en fonction de la place restante.

Exemple :

```
local cstring utf8$ , POINTER H%
Local cstring SubStr$
Local POINTER HSrc%, POINTER HDest%
; 12345678901234567890123456789O
SubStr$="Natsystems is an It cie"
UTF8$ = "Nat System"
HDest% = @Substr$
HSrc%=@UTF8$
h% = UTF8Insert (HDest%, HSrc%, 23, 1, 11)
INSERT AT END "Substr$ "&&Substr$ to LB1; "Nat System is an It cie"
SubStr$="Natsystems is an It cie"
UTF8$ = "Nat System"
HDest% = @Substr$
HSrc%=@UTF8$
h% = UTF8Insert (HDest%, HSrc%, 23, -14, 10)
; 10 instead of 11 becase start is <0
INSERT AT END "Substr$ "&&Substr$ to LB1; "Nat System is an It cie"
```

Voir aussi Insert$, UTF8Delete, UTF8Copy, UTF8InsertEx

Fonction UTF8InsertEx (Librairie NSUNICOD)

Copie dans la destination l'intervalle déterminé par Start% et Len% de la chaîne source. Maxsize% est diminué de la valeur contenue dans Len% également.

Syntaxe	**UTF8InsertEx** *(pszUTF8Dest, pszUTF8Src, MaxSize%, Start%, Len%)*			
Paramètres	pszUTF8Dest	POINTER	I	adresse de la chaîne UTF8 de destination.
	pszUTF8Src	POINTER	I	adresse de la chaîne UTF8 source.
	MaxSize%	INTEGER	I/O	au retour Maxsize% est diminuée du nombre d'octets que contient la chaîne pointé par pszUTF8Src.
	Start%	INTEGER	I	le rang du 1er caractère à inclure dans la chaîne pointée par pszUTF8Dest. Pour les valeurs négatives Start% indique la position à partir de la fin.
	Len%	INTEGER	I	le nombre de caractères à garder de la chaîne destination. Si la chaîne destination est plus courte que Len% elle sera complètement remplacée par la chaîne source. Si Start% est > 0 prévoir un caractère de plus pour Len%
Valeur retournée	POINTER pointeur sur le 0 binaire terminal (\0) de la chaîne destination..			

1. Les appels aux fonctions UTF8xxxEx peuvent être imbriqués en utilisant le pointeur retourné par n'importe quelle fonction d'entre elles comme pszUTF8Dest pour la suivante. On initialise le paramètre MaxSize% avec la taille en octet de la chaîne destination qu'on aimerait avoir en finale (celle pointé par pszUTF8Dest) et on le passe à tous les appels de fonctions UTF8xxxEx imbriqués.
2. Les rangs commencent par 1.
3. Si Maxsize% le permet, la chaîne entière sera insérée à l'endroit indiqué par Start% sinon elle sera tronquée en fonction de la place restante.

Exemple :

```
local cstring utf8$(17) , INTEGER i% , POINTER H% , INTEGER Maxsize%
Local cstring SubStr$
Local POINTER HSrc%, POINTER HDest%
SubStr$="Natsystems is an It cie"
Maxsize% = 23
UTF8$ = "Nat System"
```

```
HDest% = @Substr$
HSrc%=@UTF8$
INSERT AT END "Substr$ "&&Substr$ && "Maxsize ="&&Maxsize% to LB1;
h% = UTF8InsertEx (HDest%, HSrc%,Maxsize%, 1, 11 )
;"Nat System is an It cie" Maxsize 13
INSERT AT END "Substr$ "&&Substr$ && "Maxsize ="&&Maxsize% to LB1
HSrc%=@""
h% = UTF8InsertEx (HDest%, HSrc%,Maxsize%, 1, 0 )
;"Nat System is an It cie" Maxsize 12 (- the \0)
INSERT AT END "Substr$ "&&Substr$ && "Maxsize ="&&Maxsize% to LB1
```

Voir aussi Insert$, UTF8DeleteEx, UTF8CopyEx, UTF8Insert

Fonction UTF8NextChIndex (Librairie NSUNICOD)

Retourne le rang du 1er octet du caractère UTF8 suivant.

Syntaxe	**UTF8NextChIndex** *(pszUTF8, Pos%, Len%)*			
Paramètres	pszUTF8	POINTER	I	adresse de la chaîne UTF8.
	Pos%	INTEGER	I	le rang de départ d'un octet
	Len%	INTEGER	I	taille de la chaîne UTF8 en octets
Valeur retournée	INTEGER rang du premier octet du caractère précédent.			

Exemple :

```
local cstring utf8$ ,INTEGER i% ,POINTER pH% , INTEGER K%
UTF8$ = "Nat System"
i% = 4
K% = UTF8PrevChIndex (UTF8$, i%)
INSERT AT END "K%"&&K% to LB1 ; 3
K% = UTF8NextChIndex (UTF8$,i%, sizeof UTF8$)
INSERT AT END "K%"&&K% to LB1 ; 5
```

Voir aussi UTF8PtrPrevCh, UTF8PtrNextCh, UTF8PrevChIndex, UTF8ScanNextCh, UTF8ScanPrevCh

Fonction UTF8NextWordIndex (Librairie NSUNICOD)

Si bSel% est FALSE, cherche le premier caractère du mot suivant sinon le dernier caractère du mot (au sens linguistique) présent ou suivant.

Syntaxe	**UTF8NextWordIndex** *(UTF8$, Pos%, Len%, bSel%)*			
Paramètres	UTF8$	CSTRING	I	chaîne UTF8.
	Pos%	INTEGER	I	rang du premier octet d'un caractère UTF8.
	Len%	INTEGER	I	Taille en octets de la chaîne UTF8.
	bSel%	INT(1)	I	Si FALSE cherche le premier caractère du mot suivant sinon le dernier caractère du mot (au sens linguistique) présent ou suivant.
Valeur retournée	INTEGER le rang du premier octet d'un caractère UTF8. Les rangs commencent par 0.			

Exemple :

```
local cstring utf8$ ,INTEGER i% ,POINTER pH% , INTEGER K%
UTF8$ = "Nat System is an IT Cie"
pH%=@UTF8$
i% = 0
K% = UTF8NextWordIndex (UTF8$,i%, sizeof UTF8$, FALSE%)
INSERT AT END "K%"&&K% to LB1 ; 10
K% = UTF8NextWordIndex (UTF8$,K%, sizeof UTF8$, FALSE%)
INSERT AT END "K%"&&K% to LB1 ; 13
K% = UTF8NextWordIndex (UTF8$,K%, sizeof UTF8$, FALSE%)
INSERT AT END "K%"&&K% to LB1 ; 16
K% = UTF8NextWordIndex (UTF8$,K%, sizeof UTF8$, FALSE%)
INSERT AT END "K%"&&K% to LB1 ; 19

K% = UTF8PrevWordIndex (UTF8$, K%)
INSERT AT END "K%"&&K% to LB1 ; 16
K% = UTF8PrevWordIndex (UTF8$, K%)
INSERT AT END "K%"&&K% to LB1 ; 13
K% = UTF8PrevWordIndex (UTF8$, K%)
INSERT AT END "K%"&&K% to LB1 ; 10
K% = UTF8PrevWordIndex (UTF8$, K%)
INSERT AT END "K%"&&K% to LB1 ; 0
```

***Voir aussi** UTF8PtrPrevCh, UTF8PtrNextCh, UTF8PrevChIndex, UTF8NextChIndex, UTF8ScanNextCh*

Fonction UTF8PrevChIndex (Librairie NSUNICOD)

Retourne le rang du 1er octet du caractère UTF8 précédent.

Syntaxe	**UTF8PrevChIndex** *(pszUTF8, Pos%)*			
Paramètres	pszUTF8	POINTER	I	adresse de la chaîne UTF8.

	Pos%	INTEGER	I	le rang de départ
Valeur retournée	INTEGER rang du premier octet du caractère précédent.			

Exemple :

```
local cstring utf8$ ,INTEGER i% ,POINTER pH% , INTEGER K%
UTF8$ = "Nat System"
i% = 4
K% = UTF8PrevChIndex (UTF8$, i%)
INSERT AT END "K%"&&K% to LB1 ; 3
K% = UTF8NextChIndex (UTF8$,i%, sizeof UTF8$)
INSERT AT END "K%"&&K% to LB1 ; 5
```

Voir aussi ***UTF8PtrPrevCh, UTF8PtrNextCh, UTF8NextChIndex , UTF8ScanNextCh, UTF8ScanPrevCh***

Fonction UTF8PrevWordIndex (Librairie NSUNICOD)

Retourne le rang du premier octet du premier caractère UTF8 du mot au sens linguistique contenant ou précédant le caractère UTF8 dont le premier octet se trouve à Pos%.

Syntaxe	**UTF8PrevWordIndex** *(UTF8$, Pos%)*			
Paramètres	UTF8$	CSTRING	I	chaîne UTF8.
	Pos%	INTEGER	I	rang du premier octet d'un caractère UTF8.
Valeur retournée	INTEGER le rang du premier octet du premier caractère du mot (au sens linguistique) contenant ou précédant le caractère UTF8, dont le premier octet se trouve à Pos%. Les rangs commencent à 0.			

Exemple :

```
local cstring utf8$ ,INTEGER i% ,POINTER pH% , INTEGER K%
UTF8$ = "Nat System is an IT Cie"
pH%=@UTF8$
i% = 0
K% = UTF8NextWordIndex (UTF8$,i%, sizeof UTF8$, FALSE%)
INSERT AT END "K%"&&K% to LB1 ; 10
K% = UTF8NextWordIndex (UTF8$,K%, sizeof UTF8$, FALSE%)
INSERT AT END "K%"&&K% to LB1 ; 13
K% = UTF8NextWordIndex (UTF8$,K%, sizeof UTF8$, FALSE%)
INSERT AT END "K%"&&K% to LB1 ; 16
```

```
K% = UTF8NextWordIndex (UTF8$,K%, sizeof UTF8$, FALSE%)
INSERT AT END "K%"&&K% to LB1 ; 19

K% = UTF8PrevWordIndex (UTF8$, K%)
INSERT AT END "K%"&&K% to LB1 ; 16
K% = UTF8PrevWordIndex (UTF8$, K%)
INSERT AT END "K%"&&K% to LB1 ; 13
K% = UTF8PrevWordIndex (UTF8$, K%)
INSERT AT END "K%"&&K% to LB1 ; 10
K% = UTF8PrevWordIndex (UTF8$, K%)
INSERT AT END "K%"&&K% to LB1 ; 0
```

Voir aussi *UTF8PtrPrevCh, UTF8PtrNextCh, UTF8PrevChIndex, UTF8NextChIndex, UTF8ScanNextCh*

Fonction UTF8PtrNextCh (Librairie NSUNICOD)

Retourne un pointeur sur le caractère UTF8 suivant.

Syntaxe	**UTF8PtrNextCh** *(pszUTF8)*			
Paramètre	pszUTF8	POINTER	I	adresse de la chaîne UTF8.
Valeur retournée	POINTER pointeur sur le 1er octet du caractère suivant.			

Exemple :

```
local cstring utf8$ ,INTEGER i% ,POINTER H% , K%
Local cstring SubStr$, cstring SubStr2$
Local POINTER pH%, POINTER Hprev%, POINTER HNext%
fill @SubStr$, sizeof SubStr$ , 0
UTF8$ = "Nat System"
pH%=@UTF8$
k% = 6
i% = 1
h% = UTF8FindInterval (pH%,i%, k%)
fill @SubStr$, sizeof SubStr$ , 0
mov ph%, @SubStr$, 4
INSERT AT END "SubStr$"&&SubStr$to LB1; case
Hprev% = UTF8PtrPrevCh (ph%)
fill @SubStr2$, sizeof SubStr2$ , 0
mov Hprev%, @SubStr2$, 5
INSERT AT END "SubStr2$"&&SubStr2$to LB1; icase
HNext% = UTF8PtrNextCh (h%); start address
fill @SubStr2$, sizeof SubStr2$ , 0
mov HNext%, @SubStr2$, 8
INSERT AT END "SubStr2$"&&SubStr2$to LB1; ognicase
```

Voir aussi *UTF8PtrPrevCh, UTF8NextChIndex, UTF8PrevChIndex, UTF8ScanNextCh, UTF8ScanPrevCh*

Fonction UTF8PtrPrevCh (Librairie NSUNICOD)

Retourne un pointeur sur le caractère UTF8 précédent.

Syntaxe	**UTF8PtrPrevCh** *(pszUTF8)*			
Paramètre	pszUTF8	POINTER	I	adresse de la chaîne UTF8.
Valeur retournée	POINTER pointeur sur le 1er octet du caractère précédent.			

Exemple :

```
local cstring utf8$ ,INTEGER i% ,POINTER H% , K%
Local cstring SubStr$, cstring SubStr2$
Local POINTER pH%, POINTER Hprev%, POINTER HNext%
fill @SubStr$, sizeof SubStr$ , 0
UTF8$ = "Nat System"
pH%=@UTF8$
k% = 6
i% = 1
h% = UTF8FindInterval (pH%,i%, k%)
fill @SubStr$, sizeof SubStr$ , 0
mov ph%, @SubStr$, 4
INSERT AT END "SubStr$"&&SubStr$to LB1; case
Hprev% = UTF8PtrPrevCh (ph%)
fill @SubStr2$, sizeof SubStr2$ , 0
mov Hprev%, @SubStr2$, 5
INSERT AT END "SubStr2$"&&SubStr2$to LB1; icase
HNext% = UTF8PtrNextCh (h%); start address
fill @SubStr2$, sizeof SubStr2$ , 0
mov HNext%, @SubStr2$, 8
INSERT AT END "SubStr2$"&&SubStr2$to LB1; ognicase
```

Voir aussi *UTF8PtrNextCh, UTF8NextChIndex, UTF8PrevChIndex, UTF8ScanNextCh, UTF8ScanPrevCh*

Fonction UTF8ScanNextCh (Librairie NSUNICOD)

Retourne le rang du premier octet du caractère UTF8 suivant et retourne l'équivalent en Unicode.

Syntaxe	**UTF8ScanNextCh** *(pszUTF8, Pos%, Len%)*			
Paramètres	pszUTF8	POINTER	I	adresse de la chaîne UTF8.
	Pos%	INTEGER	I/O	rang du premier octet d'un caractère

				UTF8 mise à jour pour indiquer le premier octet du caractère suivant.
	Len%	INTEGER	I	taille en octets de la chaîne UTF8.
Valeur retournée	INT(2) Le caractère Unicode équivalent.			

Permet de parcourir une chaîne UTF8 caractère par caractère en convertissant en Unicode chaque caractère.

Exemple :

```
local cstring utf8$ ,INTEGER i% ,POINTER pH% , INT K%(2)
UTF8$ = ENTRY1 ; Made of 2 byte characters
pH%=@UTF8$
i% = 4
K% = UTF8ScanPrevCh (UTF8$, i%)
INSERT AT END "Low Byte" && LOB K% && "High Byte" && HIB K%&&"i%"&&i% to
LB1
; i= 2
K% = UTF8ScanPrevCh (UTF8$, i%)
INSERT AT END "Low Byte" && LOB K% && "High Byte" && HIB K%&&"i%"&&i% to
LB1
; i= 0
K% = UTF8ScanNextCh (UTF8$,i%, sizeof UTF8$)
INSERT AT END "Low Byte" && LOB K% && "High Byte" && HIB K%&&"i%"&&i% to
LB1
; i= 2
K% = UTF8ScanNextCh (UTF8$,i%, sizeof UTF8$)
INSERT AT END "Low Byte" && LOB K% && "High Byte" && HIB K%&&"i%"&&i% to
LB1
; i= 4
```

***Voir aussi** UTF8PtrPrevCh, UTF8PtrNextCh, UTF8PrevChIndex, UTF8NextChIndex, UTF8ScanPrevCh*

Fonction UTF8ScanPrevCh (Librairie NSUNICOD)

Retourne le rang du premier octet du caractère UTF8 précédent et retourne l'équivalent en UCS-2.

Syntaxe	**UTF8PtrScanPrevCh** *(pszUTF8, Pos%)*

Paramètres	pszUTF8	POINTER	I	adresse de la chaîne UTF8.
	Pos%	INTEGER	I/O	rang du premier octet d'un caractère UTF8 mise à jour pour indiquer le premier octet du caractère précédent.
Valeur retournée	INT(2) le caractère Unicode équivalent.			

Permet de parcourir une chaîne UTF8 caractère par caractère en convertissant en UCS-2 chaque caractère.
Exemple :

```
local cstring utf8$ ,INTEGER i% ,POINTER pH% , INT K%(2)
UTF8$ = ENTRY1 ; Made of 2 byte characters
pH%=@UTF8$
i% = 4
K% = UTF8ScanPrevCh (UTF8$, i%)
INSERT AT END "Low Byte" && LOB K% && "High Byte" && HIB K%&&"i%"&&i% to
LB1
; i= 2
K% = UTF8ScanPrevCh (UTF8$, i%)
INSERT AT END "Low Byte" && LOB K% && "High Byte" && HIB K%&&"i%"&&i% to
LB1
; i= 0
K% = UTF8ScanNextCh (UTF8$,i%, sizeof UTF8$)
INSERT AT END "Low Byte" && LOB K% && "High Byte" && HIB K%&&"i%"&&i% to
LB1
; i= 2
K% = UTF8ScanNextCh (UTF8$,i%, sizeof UTF8$)
INSERT AT END "Low Byte" && LOB K% && "High Byte" && HIB K%&&"i%"&&i% to
LB1
; i= 4
```

***Voir aussi** UTF8PtrPrevCh, UTF8PtrNextCh, UTF8PrevChIndex, UTF8NextChIndex, UTF8ScanNextCh*

Fonction UTF8StrEnd (Librairie NSUNICOD)

Renvoie un pointeur sur le 0 binaire terminal de la chaîne UTF8 dont l'adresse est passée en paramètre. Ne fonctionne pas correctement avec les chaînes UTF8 mal formées.

Syntaxe	**UTF8StrEnd** *(pWideChStr)*			
Paramètre	pWideChStr	POINTER	I	adresse de la chaîne UTF8
Valeur retournée	POINTER Pointeur sur le 0 binaire terminal de la chaîne.			

Exemple :

```
local utf8$ , POINTER i%, char a
UTF8$ = ENTRY1 ; 7 characters 2 bytes each
i% = UTF8StrEnd (@UTF8$)
mov i%, @ a, 1
insert at end "asc%(a)"&& asc% (a) to LB1
```

***Voir aussi** UTF8Strlen, UTF8ChCnt*

Fonction UTF8StrLen (Librairie NSUNICOD)

Retourne la taille en caractères d'une chaîne UTF8.

Syntaxe	**UTF8StrLen** *(UTF8$)*			
Paramètre	UTF8$	POINTER	I	chaîne UTF8
Valeur retournée	INTEGER le nombre de caractères UTF8.			

Exemple :

```
local utf8$ , i%
UTF8$ = ENTRY1 ; 7 characters 2 bytes each
i% = UTF8Strlen (UTF8$)
INSERT AT END UTF8$&&"Number of charcaters ="&& i% to LB1 ; 7
```

***Voir aussi** UTF8ChCnt, UTF8StrEnd*

Fonction UTF8ToAnsi (Librairie NSUNICOD)

Convertit une chaîne UTF8 en une chaîne ANSI (ISO 8859/1).

Syntaxe	**UTF8ToAnsi** *(UTF8$, WideChCnt%, pAStr, ASize%, bForcedTerm%)*			
Paramètres	UTF8$	CSTRING	I	chaîne UTF8.

	WideChCnt%	INTEGER	I	nombre d'octets dans la chaîne UTF8$ ou -1 pour la longueur totale de la chaîne.
	pAstr	POINTER	I	adresse de la chaîne ANSI.
	ASize%	INTEGER	I	taille en octet de la chaîne ANSI de sortie.
	bForcedTerm%	INT(1)	I	TRUE force le dernier caractère de la chaîne Unicode à 0 binaire (\0) même si la chaîne est trop courte, FALSE sinon.
Valeur retournée	INTEGER nombre d'octets écrit dans la chaîne ANSI.			

Exemple :

```
local utf8$, size%, Ansi$
utf8$ = "Antoine"
size% = UTF8ToAnsi (UTF8$, -1, @Ansi$, 20, TRUE%)
INSERT AT END "(UTF8$, -1, @Ansi$, 20, TRUE%) Size" && size% to LB1
INSERT AT END "Ansi$" && Ansi$ to LB1
```

***Voir aussi** UTF8ToAnsi$, AnsiToUTF8, UnicodeToUTF8, CharsToUTF8, UTF8ToUnicode*

Fonction UTF8ToAnsi$ (Librairie NSUNICOD)

Convertit une chaîne UTF8 en une chaîne ANSI (ISO 8859/1) la chaîne en sortie est limitée à 255 caractères + le 0 binaire.

Syntaxe	**UTF8ToAnsi$** *(UTF8$)*			
Paramètre	UTF8$	CSTRING	I	chaîne UTF8.
Valeur retournée	CSTRING la chaîne ANSI			

Exemple :

```
local utf8$ , Ansi$
UTF8$ = "Antoine"
Ansi$ = UTF8ToAnsi$ (UTF8$)
INSERT AT END Ansi$&&"Length ="&& length Ansi$ to LB1
```

***Voir aussi** UTF8ToAnsi*

Fonction UTF8ToUnicode (Librairie NSUNICOD)

Convertit une chaîne UTF8 en une chaîne Unicode à 2 octets par caractère (UCS-2).

Syntaxe	**UTF8ToUnicode** *(pWideChStr, WideChCnt%, pUStr, USize%, bForcedTerm%)*			
Paramètres	pWideChStr	POINTER	I	adresse de la chaîne UTF8
	WideChCnt%	INTEGER	I	nombre d'octets dans la chaîne pointée par pWideChStr ou -1 pour la longueur totale de la chaîne pointée.
	pUstr	POINTER	I	adresse du tableau d'INT(2) pour la chaîne Unicode.
	USize%	INTEGER	I	taille en caractères de la chaîne Unicode de sortie.
	bForcedTerm%	INT(1)	I	TRUE force le dernier caractère de la chaîne Unicode à 0 binaire (\0)

				même si la chaîne est trop courte, FALSE sinon.
Valeur retournée	INTEGER nombre de caractères Unicode (sur 2 octets) : le minimum de (WideChCnt%, USize%)			

Exemple :

```
local utf8$, size%, I%
local INT unicode(2)[100]
utf8$ = ENTRY1
INSERT AT END utf8$ to LB1
fill @unicode, sizeof unicode, 0
size% = UTF8ToUnicode (@UTF8$, -1, @unicode, 20, TRUE%)
INSERT AT END "Size" && size% to LB1
FOR i% = 0 To size% - 1
INSERT AT END "Low Byte" && LOB unicode[i%] && "High Byte" && HIB
unicode[i%] to LB1
ENDFOR

utf8$ = "Antoine"
fill @unicode, sizeof unicode, 0
size% = UTF8ToUnicode (@UTF8$, -1, @unicode, 10, TRUE%)
INSERT AT END "(@UTF8$, -1, @unicode, 10, TRUE%) UCS-2 string size"&&size%
to LB1 ;8
FOR i% = 0 To size%
INSERT AT END "Low Byte" && LOB unicode[i%] && "High Byte" && HIB
unicode[i%] to LB1
ENDFOR
fill @unicode, sizeof unicode, 0
size% = UTF8ToUnicode (@UTF8$, -1, @unicode, 10, FALSE%)
INSERT AT END "(@UTF8$, -1, @unicode, 10, FALSE%) UCS-2 string size"&&size%
to LB1;8
FOR i% = 0 To size%
INSERT AT END "Low Byte" && LOB unicode[i%] && "High Byte" && HIB
unicode[i%] to LB1
ENDFOR
fill @unicode, sizeof unicode, 0
size% = UTF8ToUnicode (@UTF8$, 6, @unicode, 10, FALSE%)
INSERT AT END "(@UTF8$, 6, @unicode, 10, FALSE%) UCS-2 string size"&&size%
to LB1;6
FOR i% = 0 To size%
INSERT AT END "Low Byte" && LOB unicode[i%] && "High Byte" && HIB
unicode[i%] to LB1
ENDFOR
fill @unicode, sizeof unicode, 0
size% = UTF8ToUnicode (@UTF8$, 4, @unicode, 10, TRUE%)
INSERT AT END "(@UTF8$, 4, @unicode, 10, TRUE%) UCS-2 string size"&&size%
to LB1;4
FOR i% = 0 To size%
INSERT AT END "Low Byte" && LOB unicode[i%] && "High Byte" && HIB
unicode[i%] to LB1
```

```
ENDFOR
fill @unicode, sizeof unicode, 0
size% = UTF8ToUnicode (@UTF8$, -1, @unicode, 7, TRUE%)
INSERT AT END "(@UTF8$, -1, @unicode, 7, TRUE%) UCS-2 string size"&&size%
to LB1;7
FOR i% = 0 To size%
INSERT AT END "Low Byte" && LOB unicode[i%] && "High Byte" && HIB
unicode[i%] to LB1
ENDFOR
fill @unicode, sizeof unicode, 0
size% = UTF8ToUnicode (@UTF8$, -1, @unicode, 7, FALSE%)
INSERT AT END "(@UTF8$, -1, @unicode, 7, FALSE%) UCS-2 string size"&&size%
to LB1;7
FOR i% = 0 To size%
INSERT AT END "Low Byte" && LOB unicode[i%] && "High Byte" && HIB
unicode[i%] to LB1
ENDFOR
```

***Voir aussi** AnsiToUTF8, UnicodeToUTF8, CharsToUTF8*

Fonction UTF8ToUnicodeAlloc (Librairie NSUNICOD)

Convertit une chaîne UTF8 en une chaîne Unicode à deux octets par caractère (UCS-2).

Syntaxe	**UTF8ToUnicodeAlloc** *(pWideChStr, WideChCnt%, pUStr, USize%, bForcedTerm%)*			
Paramètres	pWideChStr	POINTER	I	adresse de la chaîne UTF8.
	WideChCnt%	INTEGER	I	nombre d'octets dans la chaîne pointée par pWideChStr ou -1 pour la longueur totale de la chaîne pointée.
	pUstr	POINTER	O	adresse d'un buffer mémoire alloué contenant la chaîne Unicode

				qu'on récupère.
	bForcedTerm%	INT(1)	I	TRUE force le dernier caractère de la chaîne Unicode à 0 binaire (\0) même si la chaîne est trop courte, FALSE sinon.
Valeur retournée	INTEGER nombre de caractères Unicode retourné dans le buffer mémoire (sur 2 octets).			

Exemple :

```
local utf8$, size%, I%
local POINTER h%, INT Unicode(2)[256]
utf8$ = "Antoine"
size% = UTF8ToUnicodeAlloc (@UTF8$, -1, h%, TRUE%)
mov h%, @Unicode, size% * 2
INSERT AT END "Size" && size% to LB1
FOR i% = 0 To size% - 1
INSERT AT END "Low Byte" && LOB Unicode[i%] && "High Byte" && HIB
Unicode[i%]\
to LB1
ENDFOR
UTF8Free h%
```

***Voir aussi** UTF8ToUnicode, UTF8Free*

Instruction UTF8Free (Librairie NSUNICOD)

Libère la mémoire allouée pour le buffer de sortie de la fonction UTF8ToUnicodeAlloc.

Syntaxe	**UTF8Free** *pWideChStr*			
Paramètre	pWideChStr	POINTER	I	adresse du buffer à libérer.

***Voir aussi** UTF8ToUnicodeAlloc*

Instruction UTF8LowcaseBuf (Librairie NSUNICOD)

Convertit en minuscule la chaîne UTF8 passée.

Syntaxe	**UTF8LowcaseBuf** *pszUTF8, Len%*

Paramètres	pszUTF8	POINTER	I	adresse de la chaîne UTF8.
	Len%	INTEGER	I	taille de la chaîne.

Exemple :

```
local utf8$, size%, I%
utf8$ = ENTRY1 ; Arabic characters
INSERT AT END utf8$ to LB1
UTF8UpcaseBuf @UTF8$, sizeof UTF8$
INSERT AT END utf8$ to LB1
UTF8LowcaseBuf @UTF8$, sizeof UTF8$
INSERT AT END utf8$ to LB1

utf8$ = "Antoine"
INSERT AT END utf8$ to LB1
UTF8UpcaseBuf @UTF8$, sizeof UTF8$
INSERT AT END utf8$ to LB1
UTF8LowcaseBuf @UTF8$, sizeof UTF8$
INSERT AT END utf8$ to LB1
```

***Voir aussi** UTF8UpcaseBuf*

Instruction UTF8UpcaseBuf (Librairie NSUNICOD)

Convertit en majuscule la chaîne UTF8 passée (si l'alphabet de la chaîne passée le permet ce qui n'est pas le cas de l'arabe par exemple).

Syntaxe	**UTF8UpcaseBuf** *pszUTF8, Len%*			
Paramètres	pszUTF8	POINTER	I	adresse de la chaîne UTF8.
	Len%	INTEGER	I	taille de la chaîne.

Exemple :

```
local utf8$, size%, I%
utf8$ = ENTRY1 ; Arabic characters
INSERT AT END utf8$ to LB1
UTF8UpcaseBuf @UTF8$, sizeof UTF8$
INSERT AT END utf8$ to LB1
UTF8LowcaseBuf @UTF8$, sizeof UTF8$
INSERT AT END utf8$ to LB1
```

```
utf8$ = "Antoine"
INSERT AT END utf8$ to LB1
UTF8UpcaseBuf @UTF8$, sizeof UTF8$
INSERT AT END utf8$ to LB1
UTF8LowcaseBuf @UTF8$, sizeof UTF8$
INSERT AT END utf8$ to LB1
```

Voir aussi UTF8LowcaseBuf

LIBRAIRIE D'AUTOMATION NSAUTOM

La librairie d'automation NSAUTOM comprend des fonctions et instructions d'automation qui permettent d'appeler un serveur d'automation OLE (par la suite, nous dirons plus simplement "serveur d'automation") et d'exécuter ses méthodes.

Pour plus d'informations sur les OLE, reportez-vous pour NatStar au Guide de Développement, et pour NS-DK au Manuel d'Utilisation.

Introduction (Librairie d'automation NSAUTOM)

L'automation permet à des applications d'échanger des informations par l'intermédiaire d'appels de fonctions. Une application est un serveur d'automation si elle possède des fonctions, objets ou propriétés que des conteneurs OLE peuvent appeler via l'automation.

En général, un serveur d'automation propose un ou plusieurs types d'objets. Chaque type d'objet possède ses propres méthodes (ou fonctions) et des propriétés (variables spécifiques à l'objet).

Word sous Windows est un serveur d'automation car vous pouvez appeler des instructions ou macros Word à partir d'un conteneur OLE. Word possède les types d'objets document, modèle de document, etc.

Chaque type d'objet possède des méthodes.

Le type d'objet document possède les méthodes NouveauDocument, FermerDocument, ChargerFichier, etc …

Grâce à l'automation, vous pouvez développer des applications capables d'interagir avec des applications existantes. En effet, l'automation fournit des commandes inter-applications qui permettent à des applications d'interagir sans intervention de l'utilisateur. Ces commandes peuvent avoir des paramètres, comme lors d'un appel de fonction. L'automation est une extension des messages DDE.

Des fonctions et instructions de la librairie NS-OLE vous permettent :

- de réaliser des conteneurs d'automation.
- de créer des objets d'automation OLE,
- d'appeler des méthodes,
- de spécifier des propriétés,
- de détruire les objets d'automation OLE créés.

Contrairement à un objet OLE qui est incorporé à un contrôle OLE, l'automation OLE est une instance d'un serveur d'automation OLE. Elle permet ainsi à un client OLE d'utiliser les fonctions d'un serveur d'automation OLE.

Elle permet de consulter ou mettre à jour un document en appelant un serveur OLE.

L'interaction entre le conteneur que vous développez et le serveur s'effectue de manière précise :

- le conteneur OLE (votre application Nat System) crée un objet d'automation OLE ce qui a pour effet de lancer le serveur d'automation OLE,
- le conteneur OLE appelle les méthodes et les propriétés de d'automation OLE,

- lorsque les échanges désirés ont eu lieu, le conteneur OLE détruit l'objet d'automation OLE.

Fonctions et instructions d'automation OLE

Fonctions OLE_CALL_METHOD*% (Librairie d'automation NSAUTOM)

Appelle une méthode d'un objet d'automation OLE.

Syntaxe	**OLE_CALL_METHOD*%** *(ObjectHandle%, MethodName$, VarType%, @Ret%, Param1Type%, @Param1$, Param2Type%, @Param2$, ...)*			
Paramètres	ObjectHandle%	INT(4)	I	Handle de l'objet d'automation
	MethodName$	CSTRING	I	Nom de la méthode
	VarType%	INT(2)	O	Type de la variable retournée
	@Ret%	ADDRESS	O	Adresse de la variable retournée
	Param1Type%	INT(2)	I	Type du premier paramètre
	@Param1$	ADDRESS	I/O	Adresse du premier paramètre
	Param2Type%	INT(2)	I	Type du second paramètre
	@Param2$	ADDRESS	I/O	Adresse du second paramètre
	etc.			
Valeur retournée	INT(2)			

1. Retourne zéro si l'appel de la méthode a échoué (par exemple si le nom de la méthode n'est pas reconnu, ou si le nombre de paramètres est incorrect), ou une valeur différente de 0 sinon.
2. Le type de la variable retournée et de chaque paramètre est l'une des constantes TYPE_*%.
3. Le nom de la fonction à utiliser dépend du nombre de paramètres attendus par la méthode. Utilisez OLE_CALL_METHOD0% pour appeler une méthode sans paramètres, OLE_CALL_METHOD1% pour appeler une méthode avec un paramètre, etc... Le nombre maximum de paramètres pouvant être utilisés par la méthode est 15.

4. Le serveur reconnaît les paramètres de la méthode en fonction de leur ordre. Aussi, si vous voulez modifier un paramètre de rang n, vous devez spécifier tous les paramètres de rang 1 à rang n-1.
5. Pour connaître la liste des méthodes appelables d'un serveur d'automation, consultez la documentation de ce serveur.
6. Ne passez jamais des constantes en paramètres mais toujours des adresses de variables.

Exemple :

```
; Ouverture d'un nouveau fichier dans WinWord
; L'instruction Word FichierNouveau n'a pas de paramètre
ret% = OLE_CALL_METHOD0%(ObjHandle%, "FichierNouveau", 0, 0)
if ret% = 0
message "Erreur", "FichierNouveau"
endif

; Ouverture du fichier autoexec.bat, avec confirmation de la conversion de
format (confirme%=1) en lecture seule (lectseule%=1)
; L'instruction Word FichierOuvrir admet trois paramètres
nom$="C:\AUTOEXEC.BAT"
confirme%=1
lectseule%=1
ret% = OLE_CALL_METHOD3%(ObjHandle%, "FichierOuvrir", 0, 0, TYPE_STRING,
@nom$, TYPE_INT2,@confirme%, TYPE_INT2,@lectseule%)
```

Voir aussi *TYPE_*%*

Fonction OLE_CREATE_AUTOMATION_OBJECT% (Librairie d'automation NSAUTOM)

Crée un objet automation OLE et lance le serveur OLE.

Syntaxe	**OLE_CREATE_AUTOMATION_OBJECT%** *(ObjectName$)*			
Paramètre	ObjectName$	CSTRING	I	nom de l'objet automation
Valeur retournée	INT(4)			

1. Retourne le handle de l'objet d'automation en cas de succès, 0 sinon. Ce handle doit être utilisé par l'instruction OLE_DELETE_AUTOMATION_OBJECT ainsi que les fonctions permettant de mettre à jour les méthodes et les propriétés de l'objet d'automation.
2. Vous devez appeler cette fonction après la fonction OLE_INIT_AUTOM% et avant toute fonction modifiant les propriétés et les méthodes des objets d'automation OLE.
3. Les documentation des applications qui supportent l'automation décrivent le nom des objets d'automation.
4. Les principaux cas d'erreurs sont les suivants :

a) le serveur d'automation n'est pas disponible,
b) le nom de l'objet automation est incorrect,
c) vous appelez OLE_CREATE_AUTOMATION_OBJECT% avant la fonction OLE_INIT_AUTOM%.

Exemple :

```
; Début de l'événement EXECUTED d'un bouton permettant de créer un objet
d'automation OLE et de lancer Word
ObjHandle% = OLE_CREATE_AUTOMATION_OBJECT%("word.basic")
if ObjHandle% = 0
message "Erreur","Création objet d'automation"
endif
```

Voir aussi OLE_DELETE_AUTOMATION_OBJECT

Instruction OLE_DELETE_AUTOMATION_OBJECT (Librairie d'automation NSAUTOM)

Détruit un objet d'automation OLE et ferme le serveur.

Syntaxe	**OLE_DELETE_AUTOMATION_OBJECT** *(ObjectHandle%)*			
Paramètre	ObjectHandle%	INT(4)	I	Handle de l'objet d'automation

Vous devez appeler cette instruction après toute fonction modifiant les propriétés et les méthodes des objets d'automation OLE et avant l'instruction OLE_TERM_AUTOM.

Exemple :

```
; Dernière ligne d'un script permettant de travailler à partir d'un objet
d'automation OLE.
OLE_DELETE_AUTOMATION_OBJECT(ObjectHandle%)
```

Voir aussi OLE_CREATE_AUTOMATION_OBJECT%

Fonction OLE_GET_PROPERTY% (Librairie d'automation NSAUTOM)

Retourne l'état d'une propriété d'un objet d'automation OLE.

Syntaxe	**OLE_GET_PROPERTY%** *(ObjectHandle%, PropertyName$, PropertyType%, @RetVal%)*			
Paramètres	ObjectHandle%	INT(4)	I	Handle de l'objet d'automation
	PropertyName$	CSTRING	I	Nom de la propriété

	PropertyType%	INT(2)	I	Type de la propriété
	@RetVal%	INT(4)	I	Adresse de la variable contenant la valeur de la propriété
Valeur retournée	INT(2)			

1. Retourne 0 si l'appel a échoué (la propriété n'existe pas par exemple), ou 1 sinon. Dans ce dernier cas, la variable dont l'adresse @val% a été passée contient la valeur de la propriété.
2. Le type de la propriété est l'une des constantes TYPE_*%.
3. Pour connaître la liste des propriétés des objets d'un serveur d'automation, consultez la documentation de ce serveur.

Exemple :

```
; Retourne l'état de la propriété MaPropriété de l'objet identifié par le
handle ObjHandle%
ret% = OLE_GET_PROPERTY%(ObjHandle%, "MaPropriété", TYPE_BOOL, @Val%)
if ret% = 0
message "Erreur", "MaPropriété"
else
message "valeur de MaPropriété",Val%
endif
```

***Voir aussi** OLE_SET_PROPERTY%, TYPE_*%*

Fonctions OLE_GET_PROPERTYx% (Librairie d'automation NSAUTOM)

Récupère la valeur d'une propriété automation comme le fait OLE_GET_PROPERTY% mais en passant un, deux, trois ou NbParams% paramètres supplémentaires. Ces fonctions sont utiles avec les propriétés indexées (comme les cellules d'un tableau Excel par exemple).

Syntaxe	**OLE_GET_PROPERTY1%** *(object%, PropertyName, typeRet%, pvalRet, type1%, pval1)* **OLE_GET_PROPERTY2%** *(object%, PropertyName, typeRet%, pvalRet, type1%, pval1, type2%, pval2)* **OLE_GET_PROPERTY3%** *(object%, PropertyName, typeRet%, pvalRet, type1%, pval1, type2%, pval2, type3%, pval3)* **OLE_GET_PROPERTYN%** *(object%, PropertyName, typeRet%, pvalRet, NbParams%, ptypes, pvals)*			
Paramètres	object%	INT(4)	I	Handle de l'objet d'automation
	PropertyName	CSTRING	I	Nom de la propriété

	TypeRet%	INT(2)	I	Type de la propriété
	pValret%	INT(4)	I	Adresse de la variable contenant la valeur de la propriété
	type1%	INT(2)	I	Type sur la valeur du paramètre
	pval1	POINTER	I	Pointeur sur la valeur du paramètre
	type2%	INT(2)	I	Pointeur sur un tableau d'entiers (de 2 octets, donnant les types des NbParams% paramètres)
	pval2	POINTER	I	Pointeur sur un tableau de pointeurs (pointeurs sur les valeurs des paramètres).

Voir aussi ***OLE_GET_PROPERTY%***

Fonction OLE_GET_SET_DISP_MSGBOX% (Librairie d'automation NSAUTOM)

Active ou désactive l'affichage de boites de message d'erreur lors des appels aux méthodes et propriété automation. La valeur retournée est l'état avant l'appel à la fonction, les valeurs passé au paramètre add% ajoutent des boites d'erreur alors que celles passées à remove% en retirent. Les valeurs possibles sont : OLE_RET_ERR_MSGBOX% pour les erreurs OLE qui ne sont pas des exceptions et OLE_EXCEPTION_MSGBOX% pour les exceptions OLE (seules celles-ci sont affichées par défaut).

Syntaxe	**OLE_GET_SET_DISP_MSGBOX%** *(add%, remove%)*			
Paramètres	add%	INT	I	Ajoute des boites d'erreur
	remove%	INT	I	Retire des boites d'erreur

1. Pour lire l'état courant :

```
status%=OLE_GET_SET_DISP_MSGBOX%(0, 0)
```

2. Pour restaurer un état antérieur :

```
status%=OLE_GET_SET_DISP_MSGBOX%(status%, BNOT status%).
```

3. Pour désactiver tous les messages d'erreur :

```
status%=OLE_GET_SET_DISP_MSGBOX%(0,BNOT 0).
```

Fonction OLE_INIT_AUTOM% (Librairie d'automation NSAUTOM)

Charge le module d'automation OLE et initialise OLE2. Retourne le handle d'automation.

Syntaxe	**OLE_INIT_AUTOM%**
Valeur retournée	INT(2)

1. Retourne le handle d'automation en cas de succès, 0 sinon. Ce handle doit être utilisé par l'instruction OLE_TERM_AUTOM.
2. Vous devez nécessairement appeler cette fonction pour utiliser l'automation dans une application NatStar. Nous vous conseillons de la mettre dans l'événement INIT de la fenêtre principale de votre application.
3. Les principaux cas d'erreurs sont les suivants :
   a) vous ne pouvez utiliser OLE 2 car il n'est pas installé dans votre environnement,
   b) vous n'avez plus de mémoire.

Exemple :

```
; Evénement INIT de la fenêtre principale d'un conteneur d'automation :
InitHandle% contient le handle d'automation.
InitHandle% = OLE_INIT_AUTOM%
if InitHandle% = 0
message "Erreur","Init"
endif
```

***Voir aussi** OLE_TERM_AUTOM*

Fonction OLE_SET_PROPERTY% (Librairie d'automation NSAUTOM)

Met à jour une propriété d'un objet d'automation OLE.

Syntaxe	**OLE_SET_PROPERTY%** *(ObjectHandle%, PropertyName$, PropertyType%, @val%)*			
Paramètres	ObjectHandle%	INT(4)	I	Handle de l'objet d'automation
	PropertyName$	CSTRING	I	Nom de la propriété
	PropertyType%	INT(2)	I	Type de la propriété
	@val%	INT(4)	I	Adresse de la variable contenant la

				valeur de la propriété
Valeur retournée	INT(2)			

1. Retourne 0 si l'appel a échoué (la propriété n'existe pas par exemple), ou 1 sinon. Dans ce dernier cas, la propriété a été mise à jour à partir du contenu de la variable dont l'adresse @val% a été passée.
2. Le type de la propriété est l'une des constantes TYPE_*%.
3. Pour connaître la liste des propriétés des objets d'un serveur d'automation, consultez la documentation de ce serveur.

Exemple :

```
; Affectation de TRUE% à la propriété MaPropriété de l'objet identifié par
le handle ObjHandle%
val%=TRUE;
ret% = OLE_SET_PROPERTY%(ObjHandle%, "MaPropriété", TYPE_BOOL, @Val%)
if ret% = 0
message "Erreur", "MaPropriété"
endif
```

***Voir aussi** OLE_GET_PROPERTY%, TYPE_*%*

Fonctions OLE_SET_PROPERTYx% (Librairie d'automation NSAUTOM)

Change la valeur d'une propriété automation comme le fait OLE_SET_PROPERTY% mais en passant un, deux, trois ou NbParams% paramètres supplémentaires. Ces fonctions sont utiles avec les propriétés indexées (comme les cellules d'un tableau Excel par exemple).

Syntaxe	**OLE_SET_PROPERTY1%** *(object%, PropertyName, typeRet%, pvalRet, type1%, pval1)* **OLE_SET_PROPERTY2%** *(object%, PropertyName, typeRet%, pvalRet, type1%, pval1, type2%, pval2)* **OLE_SET_PROPERTY3%** *(object%, PropertyName, typeRet%, pvalRet, type1%, pval1, type2%, pval2, type3%, pval3)* **OLE_SET_PROPERTYN%** *(object%, PropertyName, typeRet%, pvalRet, NbParams%, ptypes, pvals)*			
Paramètres	object%	INT(4)	I	Handle de l'objet d'automation
	PropertyName	CSTRING	I	Nom de la propriété
	TypeRet%	INT(2)	I	Type de la propriété
	pValret%	INT(4)	I	Adresse de la variable contenant la valeur de la propriété
	type1%	INT(2)	I	Type sur la valeur du paramètre
	pval1	POINTER	I	Pointeur sur la valeur du paramètre

	type2%	INT(2)	I	Pointeur sur un tableau d'entiers (de 2 octets, donnant les types des NbParams% paramètres)
	pval2	POINTER	I	Pointeur sur un tableau de pointeurs (pointeurs sur les valeurs des paramètres).

Voir aussi *OLE_SET_PROPERTY%*

Instruction OLE_TERM_AUTOM (Librairie d'automation NSAUTOM)

Libère le module d'automation OLE.

Syntaxe	**OLE_TERM_AUTOM** *(h%)*			
Paramètre	h%	INT(2)	I	sans signification

Vous devez appeler cette instruction pour arrêter l'automation dans une application NatStar. Nous vous conseillons de la mettre dans l'événement TERMINATE de la fenêtre principale de votre application.
Exemple :

```
; Evénement TERMINATE de la fenêtre principale d'un conteneur d'automation
: paramètre obligatoire mais sans signification
OLE_TERM_AUTOM(0)
```

Voir aussi *OLE_INIT_AUTOM%*

Constantes TYPE_*% (Librairie d'automation NSAUTOM)

Types utilisés pour les fonctions OLE_CALL_METHOD*%, OLE_SET_PROPERTY% et OLE_GET_PROPERTY%.

Syntaxe	Déclaration interne	Description
TYPE_INT2	2	entier sur deux octets
TYPE_INT4	3	entier sur quatre octets
TYPE_FLOAT4	4	nombre flottant sur quatre octets
TYPE_FLOAT8	5	nombre flottant sur huit octets
TYPE_STRING	8	chaîne de caractère
TYPE_BOOL	11	nombre booléen (TRUE = 1, FALSE = 0).
TYPE_BYREF	16384	pour passer un objet par référence (valeur modifiée lors de l'appel), à utiliser en combinaison avec les constantes précédentes.

Voir aussi *OLE_CALL_METHOD*%, OLE_SET_PROPERTY% et OLE_GET_PROPERTY%*

Fonction OLE_ALLOC_STRING% (Librairie d'automation NSAUTOM)

Retourne une valeur BSTR à partir d'une chaîne unicode. La chaîne unicode peut avoir été crée par l'instruction OLE_ANSI_TO_UNICODE.

Syntaxe	**OLE_ALLOC_STRING%** *(unicode_str)*			
Paramètre	unicode_str	INT(4)		handle de la chaîne unicode
Valeur retournée	Valeur BSTR			

Cette fonction n'est nécessaire que si une méthode d'automation a besoin d'un BSTR qui doit être passé par référence. Si une méthode d'automation nécessite un BTSR qui doit être passé par valeur (c'est le cas le plus fréquent), NSAUTOM convertit la chaîne d'ANSI en unicode de manière transparente, et alloue et libère la valeur BSTR automatiquement.

Voir aussi *OLE_ANSI_TO_UNICODE*

Instruction OLE_ANSI_TO_UNICODE (Librairie d'automation NSAUTOM)

Convertit une chaîne ANSI en chaîne unicode dans un buffer unicode.

Syntaxe	**OLE_ANSI_TO_UNICODE** *(hstring, unicode_buffer, taille_buffer)*			
Paramètres	hstring	CSTRING	I	handle de la chaîne ANSI
	unicode_buffer	INT(4)	I	adresse de buffer pour chaîne unicode
	taille_buffer	INT(4)	I	taille de buffer pour chaîne unicode

Cette instruction n'est pas nécessaire quand les chaînes sont passées par valeur plutôt que par référence, puisque NSAUTOM convertit les chaînes d'ANSI en Unicode automatiquement.

Voir aussi *OLE_ALLOC_STRING%*

Fonction OLE_FREE_STRING% (Librairie d'automation NSAUTOM)

Libère une chaîne qui a été allouée par OLE_ALLOC_STRING%, ou par un serveur d'automation.

Syntaxe	**OLE_FREE_STRING%** *(hstring)*			
Paramètre	hstring	INT(4)	I	Handle de la chaîne unicode

Voir aussi *OLE_ALLOC_STRING%*

Fonction OLE_INSERT_CONTROL% (Librairie d'automation NSAUTOM)

Insère un objet OLE dans un contenant OLE vide. Retourne TRUE% ou FALSE%.

Syntaxe	**OLE_INSERT_CONTROL%** *(hOLEcontrol, type_objet)*			
Paramètres	hOLEcontrol	INT(4)	I	handle d'un contrôle OLE NatStar vide
	type_objet	CSTRING	I	type d'objet OLE (comme "Microsoft Wordart 2.0" ou "NSGrid control")
Valeur retournée	INT(4) FALSE% : l'objet n'a pas pu être inséré TRUE% : l'objet a été inséré			

Fonction OLE_UNICODE_TO_ANSI$ (Librairie d'automation NSAUTOM)

Convertit une chaîne unicode en une chaîne ANSI. La chaîne unicode peut avoir été retournée par une méthode d'automation, ou créée par appel à la fonction OLE_ANSI_TO_UNICODE.

Syntaxe	**OLE_UNICODE_TO_ANSI$** *(unicode_str)*			
Paramètre	unicode_str	INT(4)	I	handle de la chaîne unicode

Voir aussi *OLE_ANSI_TO_UNICODE*

LIBRAIRIE NSOCX

Nat System propose la librairie NSOCX qui permet d'intégrer dans vos applications des contrôles basés sur les ActiveX que vous trouvez dans le commerce

Pour plus d'informations sur les OCX et ActiveX, reportez-vous pour NatStar au Guide de Développement, et pour NS-DK au Manuel d'Utilisation.

Introduction

Un ActiveX est un type de contrôle prédéfini, commercialisé par un éditeur de logiciel et utilisable pour développer des applications Windows.

Les ActiveX répondent à des besoins fonctionnels plus spécialisés que les types de contrôles standards de Windows. Produits par des éditeurs de logiciels, ils sont disponibles dans le commerce sous forme de fichier portant l'extension .OCX. Ces fichiers exécutables sont comparables à des librairies dynamiques (DLL). Un éditeur d'ActiveX peut choisir d'implémenter un ou plusieurs ActiveX par fichier.

Il existe toutes sortes d'ActiveX :

- tableur,
- accès à une base de données,
- générateur de rapport,
- etc.

Un ActiveX se comporte comme une partie d'application que vos applications Windows peuvent utiliser pour éditer des données et les présenter selon une ergonomie qui a été définie par le concepteur de l'ActiveX. A la différence d'une application classique, vous ne pouvez pas lancer l'ActiveX seul. Un ActiveX est une "mini application", un composant applicatif complet et réutilisable, mais pas autonome.

Un ActiveX fonctionne comme un serveur OLE. Vos applications, se comportant comme des conteneurs OLE, peuvent l'activer afin d'éditer des données. L'ActiveX gère entièrement la présentation, l'édition et la sauvegarde de ses données. Avec un ActiveX, vos applications ne peuvent utiliser que l'inclusion d'objets, pas la liaison – ce qui est normal puisque l'ActiveX ne peut être lancé seul.

Catégories fonctionnelles de la librairie NSOCX

Fonction d'appel d'une méthode d'ActiveX

Fonction OCX_CALL$ (Librairie NSOCX)

Appelle une méthode d'un OCX en lui passant les paramètres attendus, et retourne la valeur de retour de cette méthode convertie en chaîne de caractères.

Syntaxe	**OCX_CALL$** *(OCXControl, MethodName$, noParams%, pParams%)*			
Paramètres	OCXControl	CONTROL	I	Nom du contrôle

				OCX pour lequel exécuter la méthode
	MethodName$	CSTRING	I	Nom de la méthode à appeler
	noParams%	INT(4)	I	Nombre de paramètres à passer à la méthode
	pParams%	POINTER	I	Adresse du segment des paramètres
Valeur retournée	CSTRING			

Si vous appelez OCX_CALL$ avec un nom de méthode qui n'existe pas pour l'OCX, aucune méthode n'est exécutée et OCX_CALL$ retourne une chaîne vide. Cependant si vous appelez une méthode qui ne retourne pas de valeur, vous recevez également une chaîne vide en retour. Attention donc à bien vérifier le nom de la méthode que vous passez en paramètre.

1. OCXControl : identifiant du contrôle OCX en mode design. Cette valeur permet d'identifier l'OCX auquel la méthode appelée doit s'appliquer.
2. MethodName$: nom de la méthode que vous voulez appeler ; cette méthode doit exister pour l'OCX concerné. Pour connaître la liste des méthodes appelables d'un OCX, consultez sa documentation.
3. noParams% : nombre de paramètres passés à la méthode appelée.
4. pParams% : pointeur sur le segment des paramètres à passer à la méthode appelée. C'est l'adresse d'un segment de type OCXPARAM ; ce type est déclaré dans le fichier NSOCX.NCL comme une structure contenant des adresses de variables.

```
Pointer Param1%
Pointer Param2%
...
Pointer Param20%
```

Un segment de type OCXPARAM permet de stocker les adresses de 20 variables à passer en paramètres. Vous ne pouvez donc appeler que des méthodes ayant un nombre de paramètres inférieur ou égal à 20.

5. Avant d'appeler OCX_CALL$, vous devez allouer en mémoire le segment d'adresses des paramètres. Pour cela vous utilisez le type OCXPARAM et l'instruction NEW du langage NCL. NEW retourne un pointeur (adresse) sur le

segment alloué ; vous passerez ce pointeur dans le paramètre pParams% après avoir rempli le segment.

6. Une fois le segment alloué, vous devez alimenter ses champs (Param1, Param2, etc.) à partir des adresses des variables à passer en paramètres à la méthode. Pour obtenir l'adresse d'une variable, vous utilisez la syntaxe NCL @ : @MYVAR représente l'adresse de la variable de nom MYVAR. Attention, vous devez remplir les champs du segment par les variables dans l'ordre où la méthode les attend.
7. Alimentez autant de champs du segment que la méthode attend de paramètres, et indiquez ce nombre dans noParams%. Si la méthode que vous appelez ne prend pas de paramètre, mettez 0 (zéro) dans noParams%. Dans ce cas, le paramètre pParams% ne sera pas pris en compte ; il est inutile d'allouer un segment de paramètres et vous pouvez passer également 0 dans pParams%.
8. Au retour de l'appel de OCX_CALL$, la méthode a modifié la valeur des variables qui lui ont été passées comme des paramètres par adresse.

Avant la fin de votre application, vous devrez désallouer le segment de paramètres pour libérer la mémoire qu'il occupe. Pour cela, vous utilisez l'instruction DISPOSE du langage NCL.

Exemple :

```
; Exemple d'appel d'une méthode RandomInitialize d'un contrôle OCX
d'identifiant OCXGRID
; La méthode attend 2 paramètres entiers
; Déclaration des variables locales (dont les paramètres)
LOCAL Ret$
LOCAL MaxValue%
LOCAL Recalc%
LOCAL pParams%

; Initialisation des paramètres à passer à la méthode
MaxValue% = 100
Recalc% = TRUE%

; Allocation d'une zone en mémoire pour le segment de paramètre
NEW OCXPARAM, pParams%
; Indication de l'adresse des variables paramètres dans le segment de
paramètres
OCXPARAM(pPARAMS%).param1 = @BorneMax%
OCXPARAM(pPARAMS%).param2 = @Recalc%

Ret$ = OCX_CALL$ (OCXGRID, "RandomInitialize", 2, pParams%)
; Inutile de tester Ret$ : le seul cas d'erreur est si MaxValue% est
erroné, et nous avons passé 100, qui est une valeur OK
; Libération en mémoire du segment de paramètre
DISPOSE pParams%
```

Segment OCXPARAM (Librairie NSOCX)

Le segment OCXPARAM est une structure contenant des adresses de variables.

```
Pointer Param1%
Pointer Param2%
...
Pointer Param20%
```

Un segment de type OCXPARAM permet de stocker les adresses de 20 variables à passer en paramètre. Vous ne pouvez donc lire que des propriétés ayant un nombre de paramètres inférieur ou égal à 20.

Syntaxe	SEGMENT **OCXPARAM** INT PARAM1 (4) INT PARAM2 (4) INT PARAM3 (4) INT PARAM4 (4) INT PARAM5 (4) INT PARAM6 (4) INT PARAM7 (4) INT PARAM8 (4) INT PARAM9 (4) INT PARAM10 (4) INT PARAM11 (4) INT PARAM12 (4) INT PARAM13 (4) INT PARAM14 (4) INT PARAM15 (4) INT PARAM16 (4) INT PARAM17 (4) INT PARAM18 (4) INT PARAM19 (4) INT PARAM20 (4) ENDSEGMENT

Fonctions spécifiques au navigateur Web Microsoft

Instruction OCX_ACTIVE_TAB (Librairie NSOCX)

Active le déplacement par tabulation dans l'ActiveX Navigateur Web Microsoft.

Syntaxe	**OCX_ACTIVE_TAB**

Voir aussi *OCX_INACTIVE_TAB*

Instruction OCX_INACTIVE_TAB (Librairie NSOCX)

Désactive le déplacement par tabulation dans l'ActiveX Navigateur Web Microsoft.

Syntaxe	OCX_INACTIVE_TAB

Option par défaut.

Voir aussi *OCX_ACTIVE_TAB*

Instruction OCX_ACTIVE_KEY_MESSAGES (Librairie NSOCX)

Active le déclenchement des événements KeyUP et KeyDown dans l'ActiveX Navigateur Web Microsoft.

Syntaxe	OCX_ACTIVE_KEY_MESSAGES

Voir aussi *OCX_INACTIVE_KEY_MESSAGES*

Instruction OCX_INACTIVE_KEY_MESSAGES (Librairie NSOCX)

Désactive le déclenchement des événements KeyUP et KeyDown dans l'ActiveX Navigateur Web Microsoft.

Syntaxe	OCX_INACTIVE_KEY_MESSAGES

Option par défaut.

Voir aussi *OCX_ACTIVE_KEY_MESSAGES*

Programmation des propriétés d'un contrôle ActiveX

Au cours de l'exécution d'une application intégrant des contrôles ActiveX, vous pouvez récupérer ou modifier une propriété d'un contrôle ActiveX si la fenêtre qui le contient est affichée, même si le contrôle est caché. Cela correspond à l'intervalle de temps compris entre le début du traitement de son événement INIT et la fin du traitement de son événement TERMINATE.
Le nom de la propriété dépend de l'ActiveX et de l'information que vous voulez récupérer ou modifier. La documentation de l'ActiveX doit vous indiquer la liste de ses propriétés, et pour chaque propriété, son type et la signification de ses valeurs.

Programmation d'une propriété de type simple

Ces propriétés sont les plus simples à manipuler : vous y accédez comme s'il s'agissait de champs d'un segment NCL.

Pour récupérer la valeur d'une propriété de type simple

Syntaxe	<Variable> = <ControlName>.<PropertyName>

où :

- <Variable> nom de la variable réceptrice
- <ControlName> nom du contrôle ActiveX dans la fenêtre ou le template ; c'est le contenu du champ Name du volet de propriétés
- <PropertyName> nom de la propriété à récupérer

Exemple :

Récupération, dans la variable entière NbLines%, de la valeur de la propriété NbLines du contrôle ActiveX appelé OCXCTRL.

```
NbLines% = OCXCTRL.NbLines
```

La syntaxe <ControlName>.<PropertyName> retourne la valeur de la propriété désirée. Il n'est pas obligatoire de récupérer cette valeur dans une variable ; vous pouvez tout aussi bien l'utiliser dans un calcul ou la passer à une fonction.

Pour modifier dynamiquement une propriété de type simple

Syntaxe	<ControlName>.<PropertyName> = <Valeur>

où :

- <ControlName> nom du contrôle ActiveX dans la fenêtre ou le template ; c'est le contenu du champ Name du volet de propriétés
- <PropertyName> nom de la propriété à modifier
- <Valeur> valeur à affecter à la propriété

Exemple :
Modification (passage à la valeur 8) de la propriété entière NbLines du contrôle ActiveX appelé OCXCTRL.

```
OCXCTRL.NbLines = 8
```

Les valeurs que vous pouvez affecter aux propriétés dépendent du type de la propriété, comme indiqué ci-dessous.

Le résultat de la modification de la valeur d'une propriété dépend de l'ActiveX. Toutes les modifications ne sont peut être pas possibles ; c'est la documentation de l'ActiveX qui vous indique lesquelles sont licites.

Pour modifier les propriétés d'un contrôle ActiveX :

Pour modifier une propriété de type entier	Affectez lui un entier (INT%). Les valeurs admissibles dépendent de la propriété.
Pour modifier une propriété de type réel	Affectez lui un réel (NUM#). Les valeurs admissibles dépendent de la propriété.
Pour modifier une propriété de type chaîne de caractères	Affectez lui une chaîne de caractères (CSTRING$). Les valeurs admissibles dépendent de la propriété.
Pour modifier une propriété de type exclusif (booléen)	Affectez lui TRUE% ou FALSE%.

Pour modifier une propriété de type tristate	Affectez lui un entier (INT%) positif de valeur 0,1 ou 2.
Pour modifier une propriété de type couleur	Affectez lui une chaîne de 6 caractères décrivant une couleur. Chaque caractère représente une valeur hexadécimale : chiffre de 0 (zéro) à 9 ou lettre comprise entre 'a' et 'f'. La chaîne comprend 3 groupes de 2 caractères. Le premier groupe représente le niveau de bleu dans la couleur, le deuxième le niveau de vert, et le troisième le niveau de rouge. Un groupe représente une valeur hexadécimale codée sur un octet, et peut donc prendre une valeur entre '00' (0 = absence de couleur) et 'ff' (255 = saturation de la couleur). Par exemple '000000' représente le noir, 'FFFFFF' représente le noir, 'FF0000' représente la couleur bleue, etc.

Vous ne pouvez pas modifier dynamiquement une propriété de type image.

Exemples de programmation :

Les deux exemples ci-dessous illustrent davantage la programmation des propriétés d'un contrôle ActiveX :

Le contrôle OcxGrid possède la propriété BackColor qui définit sa couleur de fond. La ligne de code suivante affiche le contrôle ActiveX sur fond rouge :

```
OcxGrid.BackColor = '0000ff'
```

Le contrôle OcxGrid (sorte de tableur) possède la propriété CellHeight qui définit la hauteur de ses cellules. La ligne de code suivante ajoute 10 à cette hauteur :

```
Ocxgrid.CellHeight = Ocxgrid.CellHeight + 10
```

Programmation d'une propriété paramétrée

Certaines propriétés des ActiveX sont un peu plus complexes : pour accéder à leur valeur, vous devez fournir un ou plusieurs paramètres qui permettent à l'ActiveX d'atteindre la valeur adéquate. Vous disposez de deux APIs pour consulter et modifier la valeur d'une propriété de ce type.

Voici un exemple d'une telle propriété : supposons un ActiveX se présentant comme un tableau, qui permet de mémoriser des chaînes de caractères dans des cellules identifiées par un couple de coordonnées (Ligne, Colonne). L'ActiveX pourra par exemple avoir une propriété paramétrée CellText. Pour lire ou écrire cette propriété, vous devrez fournir les valeurs des deux paramètres Ligne et Colonne.

A la création d'un contrôle ActiveX, Nat System génère automatiquement un script dans l'événement INIT de sa boîte d'événements. Ce script contient des exemples

d'appel de ces APIs pour toutes les propriétés paramétrées de l'ActiveX ; ces appels sont mis en commentaire (précédés du caractère ";"). Pour une propriété donnée, vous pouvez copier l'exemple d'appel là où vous le désirez et l'adapter à vos besoins. Pensez à substituer le paramètre indiquant le nom du contrôle ActiveX concerné.

Fonction OCX_GET_PROPERTY$ (Librairie NSOCX)

Retourne la valeur d'une propriété paramétrée d'un OCX, à partir des valeurs des paramètres nécessaires pour y accéder.

Syntaxe	**OCX_GET_PROPERTY$** *(OCXControl, PropertyName$, noParams%, pParams%)*			
Paramètres	OCXControl	CONTROL	I	Nom du contrôle OCX concerné
	PropertyName$	CSTRING	I	Nom de la propriété concernée
	noParams%	INT(4)	I	Nombre de paramètres de la propriété
	pParams%	POINTER	I	Adresse du segment contenant la valeur des paramètres pour accéder à la propriété
Valeur retournée	CSTRING Valeur de la propriété consultée, sous forme d'une chaîne de caractères.			

1. OCXControl : identifiant du contrôle OCX en mode design. Cette valeur permet d'identifier l'OCX dont vous voulez consulter une propriété.
2. PropertyName$: nom de la propriété que vous voulez lire ; cette propriété doit exister pour l'OCX concerné. Pour connaître la liste des propriétés paramétrées d'un OCX, consultez sa documentation.
3. noParams% : nombre de paramètres de la propriété paramétrée.
4. pParams% : pointeur sur le segment contenant les valeurs des paramètres. C'est l'adresse d'un segment de type OCXPARAM ; ce type est déclaré dans le fichier NSOCX.NCL comme une structure contenant des adresses de variables.

```
Pointer Param1%
Pointer Param2%
...
Pointer Param20%
```

Un segment de type OCXPARAM permet de stocker les adresses de 20 variables à passer en paramètres. Vous ne pouvez donc lire que des propriétés ayant un nombre de paramètres inférieur ou égal à 20.

5. Avant d'appeler OCX_GET_PROPERTY$, vous devez allouer en mémoire le segment d'adresses des paramètres. Pour cela vous utilisez le type OCXPARAM et l'instruction NEW du langage NCL. NEW retourne un pointeur (adresse) sur le segment alloué ; vous passerez ce pointeur dans le paramètre pParams% après avoir rempli le segment.
6. Une fois le segment alloué, vous devez alimenter ses champs (Param1, Param2, etc.) à partir des adresses des variables à passer en paramètres à la propriété. Pour obtenir l'adresse d'une variable, vous utilisez la syntaxe NCL @ : @MYVAR représente l'adresse de la variable de nom MYVAR. Attention, vous devez remplir les champs du segment par les variables dans l'ordre où la propriété les attend.
7. Alimentez autant de champs du segment que la propriété attend de paramètres, et indiquez ce nombre dans noParams%.

Avant la fin de votre application, vous devrez désallouer le segment de paramètres pour libérer la mémoire qu'il occupe. Pour cela, vous utilisez l'instruction DISPOSE du langage NCL.

Exemple :

```
; Exemple de lecture d'une propriété CellText (texte de cellule) d'un
contrôle OCX de type tableau, d'identifiant OCXCONTROL
; La propriété est paramétrée par 2 paramètres entiers :
; - LigNo = n° de ligne
; - ColNo = n° de colonne

; Déclaration des variables locales (dont les paramètres)
LOCAL ColWidth%
LOCAL LigNo%
LOCAL ColNo%
LOCAL pParams%

; Initialisation des paramètres pour accéder à la propriété
LigNo% = 3 ; cellule à la 3ème ligne
ColNo% = 4 ; cellule à la 4ème colonne

; Allocation d'une zone en mémoire pour le segment de paramètre
NEW OCXPARAM, pParams%
; Indication de l'adresse des variables paramètres dans le segment de
paramètres
OCXPARAM(pPARAMS%).param1 = @LigNo%
OCXPARAM(pPARAMS%).param2 = @ColNo%

ColWidth% = OCX_GET_PROPERTY$ (OCXCONTROL, "CellText", 2, pParams%)
; Le NCL convertit automatiquement le retour de la fonction en entier
```

```
; Libération en mémoire du segment de paramètre
DISPOSE pParams%
```

Instruction OCX_SET_PROPERTY (Librairie NSOCX)

Change la valeur d'une propriété paramétrée d'un OCX, à partir des valeurs des paramètres nécessaires pour y accéder et de la nouvelle valeur à affecter à la propriété.

Syntaxe	**OCX_SET_PROPERTY** *(OCXControl, PropertyName$, PropertyValue$, noParams%, pParams%)*			
Paramètres	OCXControl	CONTROL	I	Nom du contrôle OCX concerné
	PropertyName$	CSTRING	I	Nom de la propriété concernée
	PropertyValue$	CSTRING	I	Nouvelle valeur à affecter à la propriété concernée
	noParams%	INT(4)	I	Nombre de paramètres de la propriété
	pParams%	POINTER	I	Adresse du segment contenant la valeur des paramètres pour accéder à la propriété

1. PropertyValue$: nouvelle valeur à affecter à la propriété.
2. Pour l'explication des autres paramètres et le mode d'emploi de la fonction, référez vous aux commentaires de la fonction OCX_GET_PROPERTY$. La seule différence est que l'instruction OCX_SET_PROPERTY prend en plus le paramètre PropertyValue$, et ne retourne rien.

Exemple :

```
; Exemple d'écriture d'une propriété CellText (texte de cellule) d'un
contrôle OCX de type tableau, d'identifiant OCXCONTROL
```

```
; La propriété est paramétrée par 2 paramètres entiers :
; - LigNo = n° de ligne
; - ColNo = n° de colonne

; Déclaration des variables locales (dont les paramètres)
LOCAL ColWidth%
LOCAL LigNo%
LOCAL ColNo%
LOCAL pParams%

; Initialisation des paramètres pour accéder à la propriété
LigNo% = 3 ; cellule à la 3ème ligne
ColNo% = 4 ; cellule à la 4ème colonne

; Allocation d'une zone en mémoire pour le segment de paramètre
NEW OCXPARAM, pParams%
; Indication de l'adresse des variables paramètres dans le segment de
paramètres
OCXPARAM(pPARAMS%).param1 = @LigNo%
OCXPARAM(pPARAMS%).param2 = @ColNo%

; On met la chaîne "New text" dans la cellule désignée
OCX_SET_PROPERTY OCXCONTROL, "CellText", "New text", 2, pParams%

; Libération en mémoire du segment de paramètre
DISPOSE pParams%
```

Programmation d'une propriété d'un sous-objet

Certains ActiveX contiennent des objets, qui contiennent eux-mêmes des propriétés. Vous disposez de deux APIs pour consulter et modifier la valeur d'une propriété d'un sous- objet d'un ActiveX.

Fonction OCX_GET_SUBPROPERTY$ (Librairie NSOCX)

Retourne la valeur d'une propriété d'un sous objet d'un OCX.

Syntaxe	**OCX_GET_SUBPROPERTY$** *(OCXControl, ObjectPropertyName$, Type%)*			
Paramètres	OCXControl	CONTROL	I	Nom du contrôle OCX concerné
	ObjectPropertyName$	CSTRING	I	Nom de la propriété concernée
	Type%	INT(4)	I	Type de la propriété à récupérer

Valeur retournée	CSTRING C'est la valeur de la propriété consultée, sous forme d'une chaîne de caractères.

1. OCXControl : identifiant du contrôle OCX en mode design. Cette valeur permet d'identifier l'OCX dont vous voulez consulter une propriété de sous objet.
2. ObjectPropertyName$: nom de la propriété que vous voulez lire ; le sous objet et sa propriété doivent exister pour l'OCX concerné. Pour connaître la liste des sous objets d'un OCX et de leurs propriétés, consultez la documentation de l'OCX.

La syntaxe de la chaîne à passer est <ObjectName>.<PropertyName>, avec :

- <ObjectName> nom du sous objet
- <PropertyName> nom de la propriété du sous objet

3. Type% : Type de la propriété à récupérer. C'est l'une des valeurs constantes T_* qui sont définies dans NSOCX.NCL :

```
T_INT2 pour un entier codé sur 2 caractères
T_INT4 pour un entier codé sur 4 caractères
T_FLOAT4 pour un réel codé sur 4 caractères
T_FLOAT8 pour un réel codé sur 8 caractères
T_STRING pour une chaîne de caractères
T_OBJECT pour un objet
T_BOOL pour un booléen
= entier codé sur 2 caractères
(TRUE = 1, FALSE = 0)
```

Exemple :

```
; Un contrôle OCX d'identifiant OCXCONTROL a un sous-objet de nom Sheet.
Celui-ci a une propriété SheetName de type STRING
; Lecture de la propriété SheetName du sous-objet Sheet

LOCAL SName$
SName$ = OCX_GET_SUBPROPERTY$ (OCXCONTROL, "Sheet.SheetName", T_STRING)
```

Voir aussi ***OCX_SET_SUBPROPERTY***

Instruction OCX_SET_SUBPROPERTY (Librairie NSOCX)

Modifie la valeur d'une propriété d'un sous objet d'un OCX.

Syntaxe	**OCX_SET_SUBPROPERTY** *(OCXControl, ObjectPropertyName$, ObjectPropertyValue$, Type%)*			
Paramètres	OCXControl	CONTROL	I	Nom du contrôle OCX concerné
	ObjectPropertyName$	CSTRING	I	Nom de la propriété concernée

	ObjectPropertyValue$	CSTRING	I	Valeur de la propriété concernée
	Type%	INT(4)	I	Type de la propriété à modifier

1. ObjectPropertyValue$: nouvelle valeur à affecter à la propriété.
2. Pour l'explication des autres paramètres et le mode d'emploi de la fonction, référez vous aux commentaires de la fonction OCX_GET_SUBPROPERTY$. La seule différence est que l'instruction OCX_SET_SUBPROPERTY prend en plus le paramètre ObjectPropertyValue$, et ne retourne rien.

Exemple :

```
; Un contrôle OCX d'identifiant OCXCONTROL a un sous-objet de nom Sheet.
Celui-ci a une propriété SheetName de type STRING
; Modification de la propriété SheetName du sous-objet Sheet (on y met le
texte "New text")
OCX_SET_SUBPROPERTY OCXCONTROL, "Sheet.SheetName", "New text", T_STRING
```

Voir aussi OCX_GET_SUBPROPERTY$

GLOSSARY

A

Attribut: Un attribut XML permet de spécifier les caractéristiques d'un élément. Par exemple, on peut définir les caractéristiques d'une voiture par sa marque et sa couleur sous la forme : <Voiture Marque="Toyota" Couleur="Rouge"/> Syntaxe : <nom_élément nom_attribut1= "valeur de l'attribut1" nom_attribut2= "valeur de l'attribut2">

C

CDATASection: Les sections CDATA sont utilisées pour protéger des caractères de balisage dans des portions de textes. Le seul délimiteur reconnu dans une section CDATA est la chaîne "]]>" qui ferme les sections CDATA. Les sections CDATA ne peuvent pas être imbriquées les unes dans les autres. L'objectif premier est d'inclure tel quel des fragments XML, sans avoir à protéger tous les délimiteurs propres au langage.

D

Document: Un nœud Document représente tout le document XML et constitue donc la racine de l'arbre.

DocumentFragment: Un nœud DocumentFragment représente un fragment du Document.

DocumentType: Chaque Document a un attribut doctype dont la valeur est soit null soit un nœud DocumentType. Le DocumentType permet de définir le type du Document.

DOM: DOM (Document Object Model) correspond à un langage standard d'interface (API) permettant à un programme de naviguer dans un arbre XML et d'en lire ou d'en modifier le contenu. DOM impose de construire un arbre en mémoire contenant l'intégralité des éléments du document en mémoire.

DTD: La DTD (Document type Definition) est un composant optionnel d'un document XML. Il permet de différencier un document bien formé d'un document valide. Un document bien formé respecte les règles syntaxiques du langage XML. Un document valide est conforme à la grammaire définie par la DTD. Ainsi, associer une DTD à un document XML permet de vérifier sa validité. La DTD spécifie : "Les balises contenues dans les documents, "Les balises pouvant contenir d'autres balises, "Le nombre et la séquence des balises, "Les attributs joints aux balises, "La valeur des attributs.

E

Element: Un nœud Element correspond à l'objet de base d'un document XML. Syntaxe : <nom_élément nom_attribut1= "valeur de l'attribut1" nom_attribut2= "valeur de l'attribut2"> données …</nom_élément>

Entity: Un nœud Entity représente n'importe quelle entité d'un document XML qu'elle soit analysable ou non par le parseur. Notez que l'interface modélise l'entité elle-même et non sa déclaration. La modélisation des déclarations d'entités sera intégrée dans une prochaine spécification de DOM.

Entity reference: Les nœuds EntityReference correspondent à des abréviations pour des données répétitives. Elles sont référencées dans le document XML et délimitées par les caractères & et ; . Exemple de déclaration d'entité : <!ENTITY xml " Extensible Markup Language "> Exemple d'utilisation dans un document XML : Le langage XML (&xml ;) … Résultat : Le langage XML (Extensible Markup Language) …

Espace de noms: Un espace de noms est un mécanisme utilisé pour lever les éventuelles ambiguïtés des balises. Par exemple, la balise <NOM> peut désigner le nom d'une personne ou d'un produit. Les espaces de noms permettent de les différencier en utilisant un préfixe : " perso:nom " et " produits:nom ". Exemple de déclaration des espaces de noms : <BIBLIO xmlns= " http://www.purchase.com " xmlns:perso= " http://www.noms.org "> Les balises non préfixées se rapporteront à l'espace de noms par défaut " http://www.purchase.com " . Les balises préfixées par " perso : " se rapporteront à l'URL " http://www.noms.org ".

F

Feuille de style: Une feuille de style contient un ensemble de règles de présentation d'un document.

N

Namespaces: Un espace de noms est un mécanisme utilisé pour lever les éventuelles ambiguïtés des balises. Par exemple, la balise <NOM> peut désigner le nom d'une personne ou d'un produit. Les espaces de noms permettent de les différencier en utilisant un préfixe : " perso:nom " et " produits:nom " Exemple de déclaration des espaces de noms : <BIBLIO xmlns= " http://www.purchase.com " xmlns:perso= " http://www.noms.org "> Les balises non préfixées se rapporteront à l'espace de noms par défaut " http://www.purchase.com " . Les balises préfixées par " perso : " se rapporteront à l'URL " http://www.noms.org ".

Noeud: Un nœud correspond au point d'un arbre d'où partent une ou plusieurs branches.

Notation: Un nœud Notation sert, soit pour déclarer par un nom, le format d'une entité non contrôlée par le parseur, soit pour des déclarations formelles de cibles d'instructions de traitement. Un nœud Notation n'a aucun parent. Syntaxe pour définir le type de données non XML utilisé par une entité externe : < !NOTATION NomNotation SYSTEM|PUBLIC " ValeurNotation "> < !ENTITY NomEntiteExterne SYSTEM " FichierNonXML " NDATA NomNotation> Exemple : < !NOTATION JPEG SYSTEM " PaintShopPro.exe " > < !ENTITY logo SYSTEM " Nat System.jpg " NDATA JPEG >

P

Parseur: Un parseur XML analyse les flux XML et stocke les contenus en mémoire sous forme de DOM (Document Object Model) pour des traitements ultérieurs.

Processing Instruction: Un nœud ProcessingInstruction représente une instruction de traitement, utilisée en XML comme un moyen pour conserver des informations spécifiques à un processeur dans le texte même du document. Les instructions de traitement commençant par xml ont un usage réservé par le standard XML. Syntaxe : < ? Nom_Instruction ContenuLibre ?> Exemple de la déclaration XML : < ?xml version= " 1.0 " encoding= " UTF-8 " standalone= " yes" ?>

S

SAX: SAX (Simple API pour XML) est une API basée sur un modèle événementiel, cela signifie que SAX permet de déclencher des événements au cours de l'analyse du document XML. Une application basée sur SAX peut gérer uniquement les éléments dont elle a besoin sans avoir à construire en mémoire une structure contenant l'intégralité du document.

W

W3C: Organisme de proposition et de normalisation des protocoles et langages du Web. Site du W3C : http://www.w3.org

X

XML: XML est un langage de balisage extensible, c'est-à-dire que les développeurs peuvent créer leurs propres balises en fonction du type d'information à stocker et/ou à échanger. XML distingue bien les données et leur présentation. La présentation des données étant prise en charge par XSL.

XSL: XSL correspond au langage de feuilles de style XML. XSL définit différentes règles de présentation d'un document XML selon le support de lecture.

XSLT: Le processus XSLT permet de transformer un document XML en appliquant la feuille de style adéquate selon le type de terminal connecté.

INDEX