Manuel d'utilisation
Langage NCL

Table Of Contents

Ce manuel de référence contient toutes les informations nécessaires pour comprendre et utiliser du langage NCL dans les logiciels Nat System.

ELEMENTS DU LANGAGE

Introduction

NCL est un langage de programmation incluant un jeu complet d'instructions et fonctions, permettant d'écrire des programmes qui accèdent et manipulent l'interface utilisateur des fenêtres conçues

Généralement, une instruction tient en une ligne. Si une instruction est trop longue pour tenir en une seule ligne, il faut mettre un anti-slash (\) à la fin de la ligne pour étendre l'instruction à la ligne suivante.

> Attention, il ne doit y avoir aucun caractère après l'anti-slash.

L'instruction

```
MOVE 1 TO I%
```

peut être écrite

```
MOVE 1 \
TO \
I%
```

> Il ne doit y avoir aucun caractère après l'anti-slash.

Les commentaires du programme sont intéressants pour indiquer les parties spécifiques du programme qui effectuent quelque chose de particulier. Les commentaires peuvent être insérés en utilisant le point-virgule (;) pour indiquer le début du commentaire. Tout texte situé entre le point-virgule et la fin de la ligne est ignoré par le programme et considéré comme un commentaire.

```
; ceci est un commentaire et n'est pas exécuté
MOVE 1 TO I% ; ceci est un bon endroit
; pour un commentaire
```

Notes :

- Il est interdit de mettre un commentaire après un anti-slash (\).
- Il existe une instruction NCL, équivalente au point-virgule, permettant aussi de mentionner un commentaire : il s'agit de l'instruction COMMENT.
- Les espaces et tabulations peuvent être utilisés librement entre les symboles d'une instruction.

Littéraux

Entiers

Notation conventionnelle pour les entiers.
Exemple :

```
12, 1024 (base 10)
```

Entiers hexadécimaux avec le signe dollar ($) en premier caractère.
Exemple :

```
$C, $400 (base 16)
```

Entiers exprimés dans des bases entre 2 et 16. La base est spécifiée devant l'entier, les deux champs devant être séparés par le caractère dièse (#).
Exemples :

```
10#12, 10#1024
16#C, 16#400
2#1100,2#10000000000
```

L'entier minimum est -2147483647, et l'entier maximum est +2147483647. Il est toutefois possible de restreindre cet intervalle lors d'une déclaration explicite (cf LOCAL et GLOBAL).

Version 11
Si on utilise les nouveaux int(8) les valeurs minimum et maximum passent de -9223372036854775808 à 9223372036854775807.

Pour déclarer un littéral ou une constante au format INT(8) il faut rajouter le suffixe L au littéral.

exemples :

```
const BIGNUMBER 30000000000000000L

local i8% (8)
i8% = 30000000000000000L
```

Réels

Ils sont définis en notation conventionnelle. Pour différencier un réel d'un entier, il faut rajouter une partie fractionnaire ou un exposant.
Exemple :

```
1., 1.0, 1E+12, 1E-12, 1E12, 1.E+12
```

Les réels s'expriment entre +-3.4E-4932 et +-1.2E+4932 (plus, bien sûr, 0), avec environ 18 chiffres significatifs pour la mantisse. Il est toutefois possible de définir des réels ayant moins de chiffres significatifs et un exposant maximum plus petit lors d'une déclaration explicite (cf LOCAL et GLOBAL) : cela permet des économies de place mémoire, des calculs plus rapides, mais bien sûr avec moins de précision.

.1 n'est pas valide. Un réel doit commencer par un chiffre (0.1 par exemple)

Chaînes

Les chaînes sont définies entre apostrophes (') ou guillemets (").
Exemple :

```
'array' ou "array" ou "d'est" ou 'd"est'
```

Même si ce n'est pas nécessaire, pour entrer une apostrophe (ou un guillemet) à l'intérieur d'une chaîne définie avec des apostrophes (ou des guillemets), il suffit de doubler le caractère :

```
'd''est' ou "d""est"
```

Il est possible d'insérer des caractères non affichables dans une chaîne en utilisant le caractère accent circonflexe (^) suivi du caractère ASCII (pour définir un caractère de contrôle) ou le caractère dièse (#) suivi du code ASCII du caractère.

```
'array'^M^J'split in two' or
'array'#13#10'split in two'
```

Variables

Une variable permet de stocker une valeur qui pourra par la suite être utilisée dans une instruction. La valeur d'une variable est ce qui y est stocké au moment où elle est utilisée.

La valeur d'une variable est utilisable en employant son nom. Typiquement, la valeur change pendant l'exécution d'un programme alors que le nom de la variable reste le même.

La valeur d'une variable dépend de son type. NCL admet les types suivants : nombres (ou réels), entiers, chaînes de caractères et contrôles.

Conventions pour les noms de variable

Une variable est identifiée par son nom, appelé identifieur. Un identifieur doit commencer par un caractère alphabétique (A-Z ou a-z) et peut inclure n'importe quel nombre de caractères alphanumériques, y compris le tiret de soulignement (ou underscore).

Le nom des variables ne doit pas dépasser 31 caractères.

Les minuscules n'ont pas de signification particulière : elles sont automatiquement traduites en majuscules par l'interpréteur.

Une variable doit être terminée par un caractère spécial qui définit son type lorsque ce dernier n'est pas explicitement précisé lors de la déclaration de la variable. Cela est possible pour les types suivants :

caractère	type de la variable
$ (dollar)	chaînes de caractère (type CSTRING).
% (pour-cent)	entier (type INT ou POINTER)
£ (livre)	réel (type NUM)
# (dièse)	réel (type NUM)

Exemples de variables correctes	A% : une variable entière A1% : une autre variable entière Name$: une variable chaîne de caractères FIRST_NAME$: une autre variable chaîne PRICE£ : une variable réelle
Exemples de variables incorrectes	1NAME : doit commencer par une lettre NAME : doit être terminé par un caractère indiquant le type

Affectation de variables

Les variables peuvent stocker des chaînes, des entiers ou des nombres selon leur type.
L'instruction MOVE permet d'affecter des valeurs aux variables :

```
MOVE 1 TO A%   ; La variable A% vaut 1
MOVE 'Hello world' TO TEST$ ; La variable TEST$
     ; vaut 'Hello world'
MOVE 12.3E+12 TO R£  ; Affectation de R£
```

L'affectation peut également se faire à l'aide de l'opérateur "=" :

```
A% = 1    ; équivalent à MOVE 1 TO A%
```

L'utilisation du MOVE ou du "=" est indifférente. Utilisez celle qui vous est la plus naturelle en fonction des langages de programmation que vous connaissez déjà. Pour une meilleure lisibilité, il est cependant conseillé de ne conserver qu'une seule forme d'affectation dans le code d'une même application.
Une variable peut être utilisée au sein d'une instruction à chaque fois que sa valeur est nécessaire.

```
MOVE (A% + 3) * 4 TO B%
; Affectation de la variable B%
; ou encore
B% = (A% + 3) * 4
```

Localité ou globalité

Une variable peut être déclarée de façon locale grâce au mot-clé LOCAL. Cela signifie que son existence est uniquement liée à l'événement dans lequel elle est déclarée. En d'autres termes, une variable locale déclarée sur un événement ne pourra pas être utilisée depuis tout autre événement (du même objet ou d'un autre objet). Il en découle que des variables locales de même nom peuvent exister dans différents événements sans se gêner mutuellement, chacune ayant leur existence propre.

Si une variable est déclarée locale au sein d'une fonction ou instruction (FUNCTION et INSTRUCTION), son existence est limitée à la fonction ou à l'instruction.

A l'opposé, le mot-clé GLOBAL déclare une variable globale. La variable pourra être utilisée dans la totalité de l'application (dans le même événement, mais aussi dans tous les objets de la fenêtre, et même dans toutes les fenêtres de l'application).

Exemples :

```
LOCAL A% ; A% ne pourra être utilisée qu'au sein de l'événement
GLOBAL S$ ; S$ pourra être utilisée dans la totalité de l'application
```

Il est possible d'insérer des définitions multiples de variables dans la même ligne en insérant un séparateur virgule (,) entre chaque définition.

Exemple :

```
LOCAL A%, B%, N€, NOM$
GLOBAL S$, I%
```

Une déclaration GLOBAL doit être placée :

- soit dans une librairie .NCL,
- soit dans un événement INIT d'une fenêtre,
- soit dans une ressource .VAR.

Il peut parfois être intéressant de manipuler des variables locales à une fenêtre, notamment lorsqu'une même fenêtre doit être démarrée plusieurs fois dans la même application, et que chacune de ces fenêtres doit manipuler des données différentes. Dans ce cas, LOCAL et GLOBAL conviennent mal : employer plutôt SETDATA et GETDATA%, décrites dans le Chapitre "Référence du Langage".

Si une définition ne respecte pas cette règle (située sur l'événement EXECUTED d'un Push-Button par exemple), le testeur de syntaxe affiche le message d'erreur "Definition not

allowed here". De plus, lorsqu'elles sont situées dans un événement INIT, les déclarations GLOBAL et LOCAL doivent être regroupées IMPERATIVEMENT en tête de script : si tel n'est pas le cas, le message d'erreur "Invalid place for definition" apparaît.

Différences de fonctionnement entre le mode interprété et compilé :

- En mode interprété (test), les variables globales sont visibles et utilisables dans toute l'application sans restriction.
- En mode compilé, les variables globales définies dans une librairie NatStar (compilée sous forme d'une librairie dynamique ou DLL) sont visibles et utilisables dans toute cette librairie et seulement elle. Vous pouvez cependant exporter des variables globales tout comme des fonctions (en le décrivant dans le fichier .DEF), et ainsi les rendre visibles depuis une autre DLL.

Type

Une variable peut être de l'un des types suivants :

INT	entier
INTEGER	entier (taille native du système)
POINTER	pointeur
@POINTER	pointeur typé
NUM	réel
STRING	chaîne de caractères (identique aux chaînes Pascal)
CSTRING	chaîne de caractères (identique aux chaînes C)
DYNSTR	chaîne de caractères ou tableau de chaîne de caractères dynamique
CHAR	caractère
CONTROL	contrôle (Entry-Field, Push-Button, etc.)

Il est possible de ne pas définir le type d'une variable lors de sa déclaration. Dans ce cas, le nom de celle-ci doit se terminer par un caractère spécial comme indiqué dans la section "Variables Conventions".

Voir plus loin la définition d'un segment et d'une variable de type CONTROL.

Exemples :

```
LOCAL INT A% ; équivalent à LOCAL A%
LOCAL NUM N£ ; équivalent à LOCAL N£

GLOBAL CSTRING S$ ; équivalent à GLOBAL S$ ou même à une déclaration
; implicite
; puisque toute variable est GLOBAL par défaut
; et S$ de type CSTRING par défaut

LOCAL STRING C$
GLOBAL CHAR T$
GLOBAL IDENTITE CLIENT1 ; IDENTITE ayant été préalablement défini avec
; SEGMENT
Il est interdit de donner un type qui ne correspond pas au caractère spécial de la fin du
nom de la variable.
LOCAL INT A$ ; Erreur !
LOCAL NUM N% ; Erreur !
GLOBAL STRING S% ; Erreur !
LOCAL CSTRING C£ ; Erreur !
GLOBAL CHAR T% ; Erreur !
GLOBAL IDENTITE CLIENT1% ; Erreur !
```

Cas particulier : le pointeur typé

Un pointeur typé est une variable qui ne nécessite pas l'indication du type pour accéder aux valeurs pointées.
Exemple de code avec pointeurs non-typés :

```
Segment typeSeg
Pointer pNext
Int Qte
Num Prix
…
EndSegment ; typeSeg

Instruction ImprimerSeg Pointer pVarSeg
Imprimer "Quantité =" && \
typeSeg(pVarSeg).Qte
Imprimer "Prix =" && \
typeSeg(pVarSeg).Prix
…
EndInstruction ; ImprimerSeg

Instruction ImprimerListeSeg \
Pointer pVarSeg
; tant que le ptr n'est pas nul
While pVarSeg
ImprimerSeg pVarSeg
; passer au suivant
pVarSeg = \
typeSeg(pVarSeg).pNext
EndWhile
EndInstruction ; ImprimerListeSeg
Exemple de code avec pointeurs typés :
On n'indique pas le type à chaque accès aux champs du segment. Les fonctions vérifient le
type du paramètre.
Segment typeSeg
typeSeg@ pNext
Int Qte
Num Prix
…
EndSegment ; typeSeg

Instruction ImprimerSeg typeSeg@ pVarSeg
Imprimer "Quantité =" && \
pVarSeg.Qte
Imprimer "Prix =" && \
pVarSeg.Prix
…
EndInstruction ; ImprimerSeg

Instruction ImprimeListeSeg \
typeSeg@ pVarSeg
; tant que le pointeur n'est pas nul
While @pVarSeg
ImprimerSeg pVarSeg
; passer au suivant
@pVarSeg = pVarSeg.pNext

EndWhile
EndInstruction ; ImprimerListeSeg
```

Pour faire pointer un pointeur typé sur une nouvelle variable

Une syntaxe spécifique doit être utilisée pour faire pointer le pointeur typé sur une nouvelle variable.

```
@pVarTypé = autreVar
```

Pour passer un pointeur typé par adresse

Pour modifier le pointeur passé plutôt que ce qu'il pointe, il est nécessaire de passer par adresse le pointeur typé. Pour cela, le pointeur typé doit être encapsulé dans un segment.

Exemple :

```
Segment typeSeg
typeSeg@ pNext
CString name
…
EndSegment ; typeSeg
Segment typeRefSeg
typeSeg@ pSeg
EndSegment ; typeRefSeg
…
Function Rechercher%(typeRefSeg@ varRefSeg, name$) Return Int(1)
While @varRefSeg.pSeg
If varRefSeg.pSeg.name = name$
 Return True%
EndIf
@varRefSeg.pSeg = varRefSeg.pSeg.pNext ; avance pSeg
EndWhile
Return False%
EndFunction ; Rechercher
…
Local typeRefSeg varRefSeg
…
@varRefSeg.pSeg = listeSeg ; un début de liste typeSeg
If Rechercher%(varRefSeg, name$)
 ImprimerSeg varRefSeg.pSeg
EndIf
```

Pour transformer un pointeur typé en un pointeur non typé

Pour modifier un pointeur typé en un pointeur non typé, il faut utiliser le caractère "@" devant la définition de la variable.

Exemple :

Ici, typeSeg(p_VarSeg) et pVarSeg sont équivalents. Ces deux transformations sont utiles pour adapter une interface pour pointeurs typés à du code existant utilisant les pointeurs non typés, ou pour les traitements dans les fenêtres (avec GetData%, Param12% ou Param34%).

```
Local typeSeg@ pVarSeg, Pointer p_VarSeg
…
; transformation du pointeur typé en pointeur non-typé
p_VarSeg = @pVarSeg
; transformation du pointeur non-typé en pointeur typé
; particulièrement utile avec GetData%, Param12% ou Param34%
; à la place de p_VarSeg.
@pVarSeg = typeSeg(p_VarSeg)
```

Pour obtenir l'adresse d'une variable pointeur typé

Pour obtenir l'adresse d'une variable pointeur typé (et pas l'adresse qu'elle contient), il faut doubler le caractère "@", comme pour les variables de type CONTROL.

Exemple :

```
Local typeSeg@ pVarSeg, Pointer p_VarSeg
…
; transformation du pointeur typé en pointeur non-typé
p_VarSeg = @pVarSeg
; transformation du pointeur non-typé en pointeur typé
@pVarSeg = typeSeg(p_VarSeg)
; pour obtenir l'adresse de ces variables pointeurs (et non
; l'adresse qu'elle contiennent), il faut faire respectivement
; @@pVarSeg et @p_VarSeg (ce qui donne deux adresses distinctes
; puisqu'il s'agit de deux variables différentes).
```

Pour copier une variable

Les deux actions @pVarTypé = autreVar et pVarTypé = autreVar sont très différentes. La première action fait pointer le pointeur typé sur une variable. La deuxième action copie la variable dans celle que pointe déjà le pointeur typé (s'il n'est pas initialisé, il y a risque de plantage).

Exemple des deux cas de figures :

```
Segment typeSeg2
…
CString texte1
…
CString texte2
…
EndSegment ; typeSeg2

Local typeSeg2 varSeg2, @pTexte$
…
If IsDisabled%(C)
; pointeur typé sur texte1, pas de copie du texte
@pTexte$ = varSeg2.text1
Else
; pointeur typé sur texte2, pas de copie du texte
@pTexte$ = varSeg2.text2
EndIf
pTexte$ = Copy$(…) & … ; change varSeg2.text1 ou varSeg2.text2
```

Pour utiliser un pointeur typé avec un type simple

Utilisées avec un type simple, les pointeurs typés permettent de se passer d'un segment n'ayant qu'un champ de ce type pour accéder à la valeur. En effet, il n'est pas nécessaire d'indiquer le type lors de chaque accès à la valeur référencée.

Pour positionner un pointeur nul dans un pointeur typé

Il y a deux façons de positionner un pointeur nul dans un pointeur typé (fin de liste ou valeur non définie par exemple) :

- en positionnant un 0 :

```
Local @pTexte$
…
@pTexte$ = 0
```

- en positionnant une constante nulle :

```
Const Null 0 ; le nom (Null) n'est pas important
…
Local @pTexte$
…
@pTexte$ = Null
```

Pour initialiser un pointeur typé sur un nouveau bloc mémoire allouée

L'instruction NEW a été modifiée pour permettre d'initialiser un pointeur typé sur un nouveau bloc mémoire alloué.

Exemple :

```
Local typeSeg2@ pVarSeg2
…
New @pVarSeg2 {, nomPublic}
…
Dispose @pVarSeg2
est équivalent à :
Local Pointer pVarSeg2
…
New typeSeg2, pVarSeg2 {, nomPublic}
…
Dispose pVarSeg2
```

Pour retourner un pointeur typé par une fonction

Si une fonction doit retourner un pointeur typé, les expressions de ses RETURN doivent être des variables de même type qui ne doivent être ni un de ses paramètres, ni une de ses variables locales. Sauf si le paramètre ou la variable locale est un pointeur typé qui ne pointe pas elle-même sur une variable locale ou un paramètre. Le langage NCL ne peut pas vérifier si c'est le cas.
Exemple :

```
Global typeSeg defVarSeg

Function nouveauSeg(…) Return typeSeg@
Local typeSeg autreDefVarSeg, typeSeg@ pVarSeg

New @pVarSeg
If @pVarSeg ; allocation OK
pVarSeg.xxx = … ; initialiser ce que pointe pVarSeg
…
Return pVarSeg
EndIf

; Return suivant OK, mais alors il faut tester le
; retour de nouveauSeg dans l'appellant
; avant d'utiliser le résultat, voir plus bas !
Return 0 ; Pointeur nul

; Une alternative possible à la ligne ci-dessus serait
; de retourner une variable globale (dont le contenu
; aurait été convenablement initialisée au préalable)
Return defVarSeg ; jamais de pointeur nul !

; Par contre, la ligne suivante est prohibée (et bloquée
; par le langage) car autreDefVarSeg cesse d'exister
; lors du Return !
Return autreDefVarSeg

; Enfin, ceci est également prohibé bien
; qu'indétectable par le langage !
@pVarSeg = autreDefVarSeg
Return pVarSeg
EndFunction ; nouveauSeg
…
Local typeSeg@ pAutreVarSeg

@pAutreVarSeg = nouveauSeg(…)
If @pAutreVarSeg ; Pointeur non nul ?
… ; utilisation de pAutreVarSeg
EndIf
```

Taille de la variable

Il est possible de préciser la place mémoire, en octets, que prend la variable. Pour les entiers et réels, cela restreint ou élargit l'intervalle des nombres acceptés, ainsi que le nombre de chiffres significatifs.

type	tailles possibles
INT	1, 2, 4 ou 8 octets (4 étant la taille par défaut) Quelle que soit la taille, les entiers sont toujours signés
NUM	4, 8 ou 10 octets (8 étant la taille par défaut)
STRING	1 à 255 octets (255 étant la taille par défaut)
CSTRING	1 à 65000 octets (la valeur maximale de n dépendant de l'environnement: voir GLOBAL et LOCAL. 255 est la taille par défaut).
CHAR	1 à 65000 octets (1 étant la taille par défaut)

Aucune taille ne peut être définie pour une variable de type segment (voir plus loin, la définition d'un segment).

Exemple :

```
LOCAL INT A%(1) ; A% pourra varier entre -127 et +127
LOCAL NUM N£(10) ; plus gros nombres acceptés, et plus de
; précision
GLOBAL A$ ; type CSTRING par défaut et taille de 255 par
; défaut
GLOBAL STRING S$(32) ; 32 caractères maxi au lieu de 255

LOCAL CSTRING C$(32)
GLOBAL CHAR T$(32)

SEGMENT IDENTITE ; Cf définition d'un segment
...
ENDSEGMENT
GLOBAL IDENTITE CLIENT1(10); Erreur !
```

Valeur initiale d'une variable

Une seule règle à retenir pour s'assurer du même comportement d'une application en test et en généré : ne jamais faire de supposition quant à une valeur initiale par défaut des variables.

En d'autres termes, vous devez toujours initialiser toutes les variables que vous manipulez. Si vous ne prenez pas cette précaution, il y a une forte probabilité que l'exécutable final ne fonctionne pas ou fonctionne aléatoirement. En effet, en phase de test le module de test gère les variables dans une zone qui lui est propre et qu'il a initialisé à zéro mais cette initialisation n'est pas faite dans le code C généré tant pour les variables globales que pour les locales.

Dimension du tableau

Il est possible de déclarer un tableau de variables grâce aux caractères [et] entre lesquels se situe une expression entière ou une valeur entière qui représente le nombre d'éléments du tableau. Le premier élément a un index 0.

Pour accéder à un élément du tableau, il faut utiliser les crochets ([ ]) comme indiqué dans les exemples suivants :

```
LOCAL ARRAY1%[12]
 ; ARRAY1% est un tableau de 12 éléments entiers. ARRAY1%[0]
; accède au premier élément. ARRAY1%[11] accède au dernier
; élément.
GLOBAL ARRAY2£[3], ARRAY3$[5], ARRAY4$(10)[9]
 ; ARRAY2£ est un tableau de 3 éléments réels.
 ; ARRAY3$ est un tableau de 5 chaînes. Par défaut une chaîne peut
; contenir 255 caractères.
 ; ARRAY4$ est un tableau de 9 chaînes, chaque chaîne étant
; limitée à 10 caractères (au lien de 255 par défaut).
```

Il est aussi possible de déclarer des tableaux à plusieurs dimensions :

```
LOCAL ARRAY5%[2][3]
 ; ARRAY5% est un tableau d'entiers à deux dimensions dont les
; éléments sont au nombre de 6 (= 2 * 3)  :
ARRAY5%[0][0] ARRAY5%[0][1] ARRAY5%[0][2]
ARRAY5%[1][0] ARRAY5%[1][1] ARRAY5%[1][2]
```

Si vous soumettez un tableau sans taille (quand la première dimension du tableau est écrite sans indiquer la taille, par exemple I%[][12] ou S$[]), un SEGMENT terminé par un tableau sans taille, ou un tableau contenant des pointeurs typés à une requête SQL, le message d'erreur suivant apparaît : "Forbidden POINTER found (define environment var. NSSQLRELAXED=PTR or ALL to disable this check)". La variable d'environnement NSSQLRELAXED peut être positionné à ALL, ou à un ou plusieurs des trois éléments suivants séparés par le caractère "+" : PTR, REF et DIM0 (pour autoriser les POINTER, les pointeurs typés et/ou les tableaux sans taille). Si vous soumettez à une requête SQL un tableau contenant des Dynstr, l'application plante.

Segments

Un segment est une structure de données, qui se déclare avec l'instruction SEGMENT.
Exemple :

```
SEGMENT IDENTITE
STRING NOM
STRING PRENOM
INT ANNEE_NAISSANCE
ENDSEGMENT
```

Pour déclarer une variable de type segment, il suffit ensuite d'employer LOCAL, GLOBAL, ou NEW.
Exemple :

```
GLOBAL IDENTITE CLIENT1
```

Pour manipuler un champ du segment, il suffit d'employer le nom de la variable segment, suivi du caractère spécial ".", et suivi du nom du champ.
Exemple :

```
MOVE "Dupont" TO CLIENT1.NOM
```

L'adresse d'une variable segment peut être obtenue grâce au caractère spécial @ suivi du nom de la variable segment. Pour manipuler un champ du segment, il faut alors employer le nom du segment (et non le nom de la variable segment), suivi de l'adresse entre parenthèses, suivi du caractère spécial ".", et suivi du nom du champ.
Ainsi, le MOVE précédent est équivalent à :

```
GLOBAL I%
MOVE @CLIENT1 TO I%
MOVE "Dupont" TO IDENTITE(I%).NOM
```

Une déclaration SEGMENT doit être placée :

- soit dans une librairie .NCL,
- soit dans un événement INIT d'une fenêtre,
- soit dans une ressource .SEG.

Si une déclaration ne respecte pas cette règle (située sur l'événement EXECUTED d'un Push-Button par exemple), le testeur de syntaxe affiche le message d'erreur "Definition not allowed here". De plus, lorsqu'elles sont situées dans un événement INIT, les déclarations SEGMENT (ainsi que GLOBAL et LOCAL) doivent être regroupées

IMPERATIVEMENT en tête de script : si tel n'est pas le cas, le message d'erreur "Invalid place for definition" apparaît.

Si vous soumettez un SEGMENT terminé par un tableau sans taille, ou un SEGMENT contenant des pointeurs typés à une requête SQL, le message d'erreur suivant apparaît : "Forbidden POINTER found (define environment var. NSSQLRELAXED=PTR or ALL to disable this check)". La variable d'environnement NSSQLRELAXED peut être positionné à ALL, ou à un ou plusieurs des trois éléments suivants séparés par le caractère "+" : PTR, REF et DIM0 (pour autoriser les POINTER, les pointeurs typés et/ou les tableaux sans taille). Si vous soumettez à une requête SQL un SEGMENT contenant des Dynstr, l'application plante.

Constantes

Une constante permet de stocker une valeur lisible mais non modifiable par une instruction NCL. La valeur d'une constante est stockée au moment de sa définition.

Cette valeur est accessible en mode lecture uniquement au moyen du nom de la constante. Typiquement, ni la valeur, ni le nom d'une constante ne changent pendant l'exécution d'un programme.

Cependant des constantes peuvent également être définies à l'aide du mot-clé CONST, soit dans les scripts NCL des événements INIT des fenêtres, soit dans les librairies. Une définition de constante doit figurer en début de script ou de librairie et commence sur une nouvelle ligne comme toute instruction du langage.

La syntaxe est :

```
CONST nom-constante expression-constante
```

où nom-constante est l'identifieur de la constante.

Des définitions multiples de constantes peuvent être déclarées sur la même ligne en séparant chaque définition par une virgule (,).

Exemples :

```
CONST PI£ 3.14159, E£ 2.71828
CONST ONE% 1, ZERO% 0
CONST NAME$ 'Pierre DUPONT'
```

Seules les constantes déclarées à l'aide de l'éditeur de constantes peuvent intégrer la notion de Multi-Target.

Sous-Chaînes

NCL a une capacité de manipulation directe des sous-chaînes grâce à la syntaxe :

```
variable-chaîne (entier..entier)
```

qui permet d'affecter ou de retourner les caractères de la variable-chaîne dont les positions sont comprises entre les deux bornes.

L'autre syntaxe :

```
variable-chaîne (entier)
```

permet d'affecter ou de retourner le caractère de la variable-chaîne dont la position est indiquée entre parenthèses. Le premier caractère est en position 0 pour les CSTRING et en position 1 pour les STRING.

Ainsi :

```
LOCAL CSTRING S$
MOVE "Nat System" TO S$
MOVE "S" TO S$(4) ; S$ vaut "Nat System"

LOCAL STRING T$
MOVE "Nat System" TO T$
MOVE "S" TO T$(5) ; T$ vaut "Nat System"
```

Ou :

```
LOCAL S$ ; CSTRING par défaut
MOVE "Nat System" TO S$
MOVE 4 TO I%
MOVE S$(I%..I%+2) TO T$ ; T$ vaut "Sys"
```

Ou encore :

```
LOCAL S$ ; CSTRING par défaut
MOVE "COURTE" TO S$
MOVE S$(0..2) TO S$(3..5) ; S$ vaut "COUCOU"
```

Expressions

Une expression est la représentation d'une valeur. Les exemples précédents utilisent des instructions d'affectation pour affecter une valeur (entière, numérique ou chaîne) à une variable. La valeur est alors une représentation de la variable. Ces expressions seront utilisées ensuite pour calculer des valeurs d'autres variables. Les expressions ne peuvent exister seules : elles sont utilisées au sein d'instructions.

Les expressions de NCL se conforment aux conventions classiques pour représenter des équations dans un langage de programmation. La plupart des opérateurs sont binaires et opèrent avec les opérandes de gauche et de droite. Les parenthèses et les lois de priorité définissent l'ordre d'évaluation.

Exemples d'expressions :

```
12.0
(12 * I%) - 8 / J%
"TEST" && NAME$
```

Voici la liste des opérateurs et leurs effets. (I pour entier, R pour réel, S pour chaîne)

Opérateur	Description	Gauche	Droite	Résultat	Exemple
-	Négation	I R	- -	I R	-12 -12.0
+	Addition	I R	I R	I R	1 + 12 1.0 + 12.0
-	Soustraction	I R	I R	I R	1 - 12 1.0 - 12.0
*	Multiplication	I R	I R	I R	1 * 12 1.0 * 12.0
/	Division	I R	I R	I R	1 / 12 1.0 / 12.0
%	Reste de la division	I	I	I	12 % 6
&	Concaténation de texte	S	S	S	"AA" & "BB"

&&	Concaténation de texte avec espace	S	S	S	"AA" && "BB"
=	Egal à	I R S	I R S	I I I	1 = 12 1.0 = 12.0 'AA' = 'BB'
<>	Différent de	I R S	I R S	I I I	1 <> 12 1.0 <> 12.0 'AA' <> 'BB'
<	Inférieur à	I R S	I R S	I R S	1 < 12 1.0 < 12.0 'AA' < 'BB'
>	Supérieur à	I R S	I R S	I I I	1 > 12 1.0 > 12.0 'AA' > 'BB'
<=	Inférieur ou égal à	I R S	I R S	I I I	1 <= 12 1.0 <= 12.0 'AA' <= 'BB'
>=	Supérieur ou égal à	I R S	I R S	I I I	1 >= 12 1.0 >= 12.0 'AA' >= 'BB'
AND	Les 2 opérandes sont-ils vrais ?	I	I	I	I% AND J%
OR	Un des opérandes est il vrai ?	I	I	I	I% OR J%
NOT	L'opérande est-il faux ?	I			NOT I%
BAND	AND bit à bit	I	I	I	I% BAND J%
BOR	OR bit à bit	I	I	I	I% BOR J%

BXOR	XOR bit à bit	I	I	I	I% BXOR J%
BNOT	NOT bit à bit	I			BNOT I%
ABS	valeur absolue	I R	- -	I R	ABS -12 ABS -12.0
NUM	Conversion en type numérique (s'il y a une erreur, 0.0 est retourné)	I R S	- - -	R R R	NUM 12 NUM 1.0 NUM "12"
INT	Conversion en type entier (s'il y a une erreur, 0 est retourné)	I R S	- - -	I I I	INT 12 INT 1.2 INT "12"
STRING	Conversion en type chaîne Pascal	I R S	- - -	S S S	STRING 12 STRING 1.2 STRING "12"
UPCASE	Conversion en majuscule	S	-	S	UPCASE "abc"
LOWCASE	Conversion en minuscule	S	-	S	LOWCASE "ABC"
SKIP	Saute les espaces de tête et de queue	S	-	S	SKIP " abc "
LSKIP	Saute les espaces de tête	S	-	S	LSKIP " abc"
RSKIP	Saute les espaces de queue	S	-	S	RSKIP "abc "
LOW	Rend la partie basse d'un entier (bits 0 à 15)	I	-	I	LOW 10000
HIW	Rend la partie haute d'un entier (bits 16 à 31)	I	-	I	HIW 10000
LOB	Rend la partie basse d'un mot (bits 0 à 7)	I	-	I	LOB 10000

HIB	Rend la partie haute d'un mot (bits 8 à 15)	I	-	I	HIB 10000
LENGTH	Rend la longueur d'une chaîne de caractères	S	-	I	LENGTH "abc"

La liste suivante décrit les priorités des opérateurs, par ordre croissant :

- AND, OR, BAND, BOR, BXOR
- =, <>, <, <=, >=, >
- addition, soustraction, &, &&
- *, /, %
- NOT, BNOT, signe +, signe -, UPCASE, LOWCASE,SKIP, LSKIP, RSKIP, NUM, INT, STRING, LOW, HIW, LOB, HIB, LENGTH, ()

Exemple 1 :

```
12 + 3 * 5
```

est strictement équivalent à

```
12 + (3 * 5)
```

Exemple 2 :

```
UPCASE SKIP " hello " && "WORLD"
```

est strictement équivalent à

```
(UPCASE (SKIP (" HELLO "))) && "WORLD" or "HELLO WORLD"
```

Les conversions sont implicites et basées sur les types des opérandes de gauche et de droite, et sur le type demandé en retour.
Quand un opérateur peut accepter un seul type (par exemple, l'opérateur de concaténation), les opérandes de gauche et de droite seront convertis dans les types espérés :

```
12 & -13 est converti en "12" & "-13"
"1000" % "12" est converti en 1000 % 12
```

Quand plus d'un type est autorisé (par exemple les opérateurs = < > ...), les opérandes sont convertis de la façon suivante, en supposant que tous les types sont valides pour les opérateurs (I = entier, R = réel, S = chaîne). :

Gauche	Droite	Type de l'opérateur

I I I	I R S	I R S
R R R	I R S	R R S
S S S	I R S	S S S

Plus spécifiquement, si un des opérandes est un opérande chaîne et si le type chaîne est valide pour l'opération, les deux opérandes seront convertis en chaînes.
Exemple :

```
12 < "13" signifie "12" < "13"
```

Si un des opérandes est une chaîne et l'autre non, et si l'opérateur ne peut pas être une chaîne, l'opérande chaîne est converti dans le type de l'autre opérande ou dans le type de l'opérateur si ce n'est pas possible.
Exemples :

```
12 + "13" signifie 12 + 13
12.0 + "13" signifie 12.0 + 13.0
12.0 % "13.0" signifie 12 % 13
```

S'il reste une ambiguïté, la chaîne est convertie en expression réelle.
Exemple :

```
+"12" devient +12.0
```

En général, c'est une mauvaise pratique de programmation d'utiliser des conversions implicites compliquées, particulièrement si le développeur ne les maîtrise pas complètement. Utiliser les opérateurs STRING, NUM et INT pour forcer les conversions.

```
NUM 12 + 12 force une opération réelle
INT 12.0 + 12 force une opération entière
```

Fenêtres

Chaque fenêtre a un nom unique qui permet de la définir sans ambiguïté.
La plupart des fenêtres sont ouvertes avec le langage NCL. Les instructions OPEN, OPENH, OPENS, OPENSH, CALL, CALLH, STROPEN, STROPENH, STROPENS, STROPENSH, STRCALL, STRCALLH sont utilisées pour ouvrir une nouvelle fenêtre.
Le nom d'une fenêtre ne peut être utilisé pour identifier une fenêtre d'une façon unique, puisqu'il est possible d'ouvrir plusieurs fois la même fenêtre au sein d'une même application (par exemple, un éditeur avec des fichiers multiples).
Une valeur entière (ou handle) est associée à chaque fenêtre ouverte et permettra de référencer la fenêtre, par exemple pour accéder à un contrôle spécifique d'une autre fenêtre.
Ce handle est retourné par les instructions NCL qui servent à ouvrir une fenêtre.
Exemple :

```
; Ouvre une fenêtre TEST. Stocke son handle dans HANDLE%.
; Cette fenêtre n'a pas de parent.
OPEN TEST,0, HANDLE%
; Affecte la valeur « DUPONT » au contrôle NAME de
; la fenêtre TEST avec handle HANDLE%
MOVE « DUPONT » TO TEST(HANDLE%).NAME
```

Deux fonctions NCL permettent de connaître directement un handle : SELF% qui retourne le handle de la fenêtre courante (celle qui contient le code qui effectue le SELF%), et MAINWINDOW% qui retourne le handle de la fenêtre initiale (celle avec laquelle l'application a démarré).

> Plusieurs fonctions et instructions NCL fonctionnent de la même façon sur les contrôles et fenêtres. On parle alors d'«objet » qui peut représenter indifféremment un contrôle ou une fenêtre. Voir le Chapitre Utilisation du Langage.

Contrôles

Un « contrôle » est une entité graphique complexe qui contient une valeur ou une liste de valeur. Ce contrôle est affiché dans la fenêtre et peut éventuellement y être modifié.

Les contrôles de base incluent des objets GUI comme static-text (texte statique), entry-fields (champs de saisie), check-boxes (cases à cocher), radio-buttons (boutons radio) , group-boxes (cadre), combo-boxes (boîtes à combinaison), bitmaps...

Comme une variable, un contrôle est une façon générale de stocker une valeur qui peut ensuite être utilisée au sein d'une expression. Du point de vue du développeur, la différence principale entre une variable et un contrôle est liée à leur domaine. Alors qu'une variable a un domaine global (la même variable peut être utilisée par tous les événements de l'application), un contrôle est stocké avec la fenêtre dans laquelle il est utilisé (le contrôle est utilisé par les événements de la fenêtre).

Il est possible d'accéder depuis une fenêtre à un contrôle d'une autre fenêtre, mais il est obligatoire dans ce cas de qualifier le nom du contrôle par une information fenêtre.

> Tous les contrôles sont considérés comme des chaînes.

Un contrôle est identifié par son nom, appelé identifieur. Un identifieur doit commencer avec un caractère alphabétique (A-Z ou a-z) et peut inclure tout nombre de caractères alphabétiques ou numériques, y compris le tiret de soulignement (_ ou underscore).

Les minuscules n'ont pas de signification particulière : elles sont automatiquement traduites en majuscules par l'interpréteur.

Contrôles corrects	A A1 EF_NOM FIRST_NAME LB_PRICE
Contrôles incorrects	1NAME : doit démarrer par une lettre NAME$: ne peut finir par un caractère de type

Déclaration et affectation de contrôles

Lors d'une tentative d'utilisation d'un contrôle qui n'existe pas, une erreur runtime est générée par l'interpréteur. La passe de compilation génère une erreur de compilation.

Normalement les contrôles sont directement modifiables par les actions de l'utilisateur. Toutefois, l'instruction MOVE (ou l'opérateur « = ») affecte des valeurs aux contrôles :

```
; Affecte 1 au contrôle A défini dans la fenêtre
; à laquelle appartient l'événement traité
MOVE 1 TO A
; ou encore
A=1
```

Le système est assez souple pour reconnaître les différents types de contrôle qui existent et pour afficher la valeur dans le format correct associé au contrôle.

Normalement, l'identifieur se réfère au contrôle défini dans la fenêtre dans laquelle le contrôle est utilisé : dans ce cas, le nom seul du contrôle suffit.

Pour accéder à un contrôle depuis une fenêtre ou depuis une fonction/instruction d'une librairie, le nom seul du contrôle ne suffit pas : il est obligatoire de qualifier le nom du contrôle en précisant la fenêtre à laquelle il appartient et son handle (qui doit impérativement être valide).

La syntaxe exacte est :

```
nom-fenêtre (handle-fenêtre).nom-contrôle
```

Pour accéder au contrôle NAME défini dans la fenêtre WINDOW, il faut entrer :

```
WINDOW(W%).NAME
```

où W% est une variable ou une expression entière contenant le handle de la fenêtre.

Pour accéder à un contrôle appartenant à un template, il n'y a aucun handle à préciser :

```
nom-template.nom-contrôle
```

Soit un Push-Button BUTTON00001 défini dans un template .TPL. Soit une fenêtre incorporant ce template à l'aide d'un contrôle template CONTROL00001.

Pour accéder au Push-Button depuis la fenêtre, il faut écrire :

```
CONTROL00001.BUTTON00001
```

Pour accéder à l'aire client d'une fenêtre de classe Edit ou List, et la manipuler comme s'il s'agissait d'un contrôle MLE ou List-Box, il faut préciser :

```
nom-fenêtre(handle-fenêtre).CLIENT
```

Lorsque la syntaxe accepte un « objet » (comme vu plus haut, ce terme est plus général que « contrôle »), le handle de la fenêtre peut être utilisé directement. C'est par exemple le cas de l'instruction LOAD.

Pour accéder aux barres de défilement associées à une fenêtre (style « Horz Scroll Bar » ou « Vert Scroll Bar » coché dans le volet de propriétés), et les manipuler comme s'il s'agissait de contrôles H.Scroll-Bar ou V.Scroll-Bar, il faut préciser :

```
nom-fenêtre(handle-fenêtre).HSCROLL
nom-fenêtre(handle-fenêtre).VSCROLL
```

CLIENT, HSCROLL et VSCROLL sont des mots-clés réservés par NCL.

Contrôles de base

Chaque contrôle de base peut stocker un jeu prédéfini de valeurs. Pour le langage NCL, tous les contrôles sont de type chaîne.
Nous définirons dans ce paragraphe la liste de tous les contrôles de base qui peuvent être manipulés par les outils NatSys. Les contrôles GUI sont supposés être connus.

Static-Text, Group-Box, Entry-Field, Push-Button

Les contrôles Static-Texts, Group-Boxes, Entry-Fields et Push-Buttons sont accessibles comme des variables chaîne qu'il est possible de lire ou d'écrire. Tout type de texte peut y être stocké.

```
MOVE 'Nom :' TO TEXT00001
; affectation d'une chaîne au static-text TEXT00001
MOVE 1 TO TEXT00001
; 1 est converti en chaîne puis affectation
MOVE 'Choix' TO GROUP00001
; affectation d'une chaîne au group-box GROUP00001
MOVE 1 TO GROUP00001
; 1 est converti en chaîne puis affectation
MOVE 'MARTIN' TO ENTRY00001
; 1 est converti en chaîne puis affectation
MOVE 'Suite' TO BUTTON00001
; affectation d'une chaîne au push-button BUTTON00001
MOVE 1 TO BUTTON00001
; 1 est converti en chaîne puis affectation
```

Un Entry-Field ayant « Spin-Button » coché dans son volet de propriétés est un cas particulier. Dans ce cas :

```
MOVE « 8 4 20 2 » TO ENTRY00001
; l'Entry-Field Spin-Button prend alors les caractéristiques suivantes :
; 8 est la valeur initiale qu'il affiche
; 4 est la valeur minimale
; 20 est la valeur maximale
; 2 est l'incrément
```

Deux remarques supplémentaires concernant un Entry-Field Spin-Button :

- L'instruction SETRANGE permet aussi de redéfinir la valeur initiale, ainsi que le minimum et le maximum, mais pas l'incrément.
- Le MOVE de lecture fonctionne comme pour un Entry-Field standard, et n'est donc pas réciproque du MOVE d'écriture que nous venons d'étudier : la valeur lue est celle affichée dans le Spin-Button, ce qui ne permet donc pas de connaître son minimum, son maximum ou son incrément.

Check-Box

Un contrôle Check box est accessible comme un variable qui ne peut contenir que trois valeurs :

- UNCHECKED% qui veut dire non-coché
- CHECKED% qui veut dire coché
- INDETERMINATE% qui veut dire indéterminé

La troisième valeur n'a de signification que si le bouton est défini à trois états (3 states check-box). En interne, une valeur entière est stockée avec le contrôle, même si le nom est considéré comme une variable chaîne. Toutefois, grâce aux conversions implicites, il suffit de taper :

```
MOVE CHECKED% TO BUTTON
```

Radio-Buttons

Un groupe de boutons radio est accessible comme une variable chaîne qui peut contenir les valeurs associées à chaque bouton radio.

Ainsi, supposons que nous ayons défini les deux boutons-radio OPTION1 et OPTION2 appartenant au même groupe, et associé l'entier 10 à OPTION1 et l'entier 20 à OPTION2. Alors, si OPTION1 est sélectionné, OPTION1 et OPTION2 retournent 10. Et si OPTION2 est sélectionné, OPTION1 et OPTION2 retournent 20.

Il ne faut pas affecter une valeur invalide à un bouton radio, sauf pour désélectionner tous les boutons. Dans l'exemple précédent, en affectant 10 à OPTION1 ou OPTION2, OPTION1 est sélectionné. En affectant 20 à OPTION1 ou OPTION2, OPTION2 est sélectionné. Toute autre valeur désélectionne OPTION1 et OPTION2.

List-Box, Combo-Box, CBE, MLE

Le nom d'une list-box ou d'une combo-box peut être utilisé et donne le texte de la ligne sélectionnée dans le contrôle. Il est possible d'indexer le nom pour obtenir le texte d'une ligne spécifique grâce aux crochets ([]).

Par exemple, si NAME est un contrôle list-box :

- NAME donne le texte de la ligne sélectionnée. Si aucune ligne n'est sélectionnée, NAME retourne une chaîne nulle. Il est impossible d'affecter une valeur à NAME si aucune ligne n'est sélectionnée.
- NAME[13] donne le texte de la 14ème ligne de la list-box. S'il n'y a pas 14 lignes, la variable retourne une chaîne nulle.

Il ne faut pas indexer pour ajouter une nouvelle ligne à une list-box. L'instruction INSERT est conçue dans ce but, et permet d'ajouter un élément dans la list-box.

Dans le cas d'une combo-box, le nom représente à la fois la ligne toujours visible du contrôle et la ligne associée dans la liste déroulante : contrairement à une CBE, ces deux textes sont toujours associés.

Dans le cas d'une combo-box avec entry-field, il est possible d'utiliser le nom pour accéder au texte défini dans l'entry-field, et les index pour accéder à une ligne de la list-box associée.

Dans le cas d'une MLE, le nom représente la ligne affichée à l'endroit du curseur clignotant, indépendamment d'une éventuelle sélection de texte.

> Se reporter au paragraphe « Listes » du chapitre « Utilisation du Langage » pour une description complète des listes.

Il est possible de formater chaque ligne d'une List Box : changer les couleurs, justifier, centrer, caler à gauche, caler à droite, encadrer ...

> Pour une description complète de ces formatages, se référer à l'annexe E Présentation des listes.

Rappelons que l'aire client d'une fenêtre de classe List peut se manipuler comme un contrôle List-Box, et l'aire client d'une fenêtre de classe Edit peut se manipuler comme un contrôle MLE. Il suffit d'utiliser CLIENT comme nom de contrôle afin de désigner l'aire client.

Contrôle Classeur

Le nom d'un contrôle Classeur peut être utilisé et donne le handle de la fenêtre à onglet sélectionnée dans le contrôle classeur.

Par exemple, si CC_TAB est le nom d'un contrôle Classeur, alors CC_TAB donne le handle de la fenêtre à onglet sélectionnée.

Il est possible d'indexer le nom pour obtenir le handle de la fenêtre rattachée à la fiche d'une fenêtre à onglet d'un contrôle Classeur avec la syntaxe : [ ].

Par exemple, si CC_TAB est le nom d'un contrôle Classeur, alors CC_TAB[0] donne le handle de la fenêtre rattachée à la fiche de la première fenêtre à onglet du contrôle Classeur. La numérotation correspond à l'ordre défini dans le volet de propriétés du contrôle Classeur.

> Se reporter au paragraphe « Contrôles Classeurs » du chapitre « Utilisation du Langage » pour une description complète des contrôles classeur.

Scroll-Bars Horizontales et Verticales

Le nom d'une scroll-bar donne la position courante de l'ascenseur qui est une valeur comprise entre le minimum et le maximum définis pour le contrôle. Pour positionner l'ascenseur, affecter une nouvelle valeur à ce contrôle.

Bitmaps

Gestion des Handle de Bitmap. Lecture, modification, affectation

Une bitmap est accessible comme une variable chaîne qui peut contenir un des éléments suivants :

1. Caractère « # » suivi d'un nombre représentant un pointeur (ou icône) système. Voir les constantes SPTR *%. Par exemple pour une icône point d'exclamation, la valeur est '#14'.
2. Un nombre représentant un handle bitmap retourné par la fonction CREATEBMP%. Un DELETEBMP est effectué automatiquement à la fermeture de la fenêtre contenant le contrôle bitmap.
3. Caractère « ^ » suivi d'un nombre représentant un handle pointeur retourné par la fonction CREATEPTR%. Un DELETEPTR est effectué automatiquement à la fermeture de la fenêtre contenant le contrôle bitmap.
4. Caractère « : » suivi d'un nombre. Comme dans le cas 2, le nombre représente aussi un handle bitmap retourné par la fonction CREATEBMP%, ou en provenance d'une autre bitmap.
   La différence est que la bitmap n'est pas désallouée, et reste donc utilisable, après la fermeture de la fenêtre contenant le contrôle bitmap.
5. Caractères « :^ » suivi d'un nombre. Comme dans le cas 3, le nombre représente aussi un handle pointeur retourné par la fonction CREATEPTR%. La différence est que le pointeur n'est pas désalloué, et reste donc utilisable, après la fermeture de la fenêtre contenant le contrôle bitmap.
6. Un nom de fichier bitmap ou pointeur. Ce nom de fichier ne doit pas commencer par les caractères #, ^, : ou un chiffre pour éviter une telle ambiguïté avec les cas précédents.

> CREATEBMP%, DELETEBMP, CREATEPTR%, DELETEPTR font partie de la librairie NSGraph.

Ces valeurs sont valables lorsque la bitmap est de type "icon" ou "Push-Button". Utiliser indifféremment MOVE, = , .TEXT ou GETTEXT$/SETTEXT.

Lorsque la bitmap est de type "Check-Box", sa valeur par MOVE est CHECKED% ou UNCHECKED% (comme un contrôle Check-Box). Utiliser GETTEXT$ ou SETTEXT pour manipuler les 6 cas ci-dessus.

Pour définir les trois bitmaps (Released, Pressed, Disable) associées à un contrôle bitmap, il suffit de séparer chaque définition par une virgule. Par exemple :

```
BITMAP00001 = "RELEA.BMP,#14,DISA.BMP"
```

Exemple :

Pour changer dynamiquement un contrôle bitmap et lui faire afficher successivement plusieurs images différentes, on peut :
- Ouvrir une fenêtre dans laquelle il y a les images stockées dans des bitmap (avec l'attribut preload de préférence)
- Puis affecter les différentes images à la bitmap désirée

```
local u%

for u% = 1 to 10
  BMP_TEST = ':' & getText$(BMP_ADD)
  wait 100
  BMP_TEST = ':' & getText$(BMP_CANCEL)
  wait 100
  BMP_TEST = ':' & getText$(BMP_CLOSE)
  wait 100
endFor

; ou bien, avec les qualificateurs dynamiques.
;
for u% = 1 to 10
  BMP_TEST.Text = ':' & BMP_ADD.Text
  wait 100
  BMP_TEST.Text = ':' & BMP_CANCEL.Text
  wait 100
  BMP_TEST.Text = ':' & BMP_CLOSE.Text
  wait 100
endFor
```

Icônes

Un contrôle icône ne peut contenir qu'un pointeur (ou icône) système, équivalent au 1 du contrôle bitmap ci-dessus mais sans le dièse (#). Ainsi, pour une icône point d'interrogation, le texte est « 14 ». Voir les constantes SPTR *%.

Templates

Le nom d'un template est lié aux événements GETVALUE et SETVALUE.
Par exemple :

```
MOVE 1 TO TEMPLATENAME
; Envoie un événement SETVALUE
; au template, avec 1 dans PARAM1%

MOVE TEMPLATENAME TO I%
; Envoie un événement GETVALUE au template.
; Si cet événement fait un RETURN 1, alors I%
; vaut 1.
```

Voir « Templates » dans le Chapitre « Utilisation du Langage » et « Evénements de Comportement » dans le Chapitre « Référence des Evénements » pour plus de détails.

Eléments de Menus

Le nom d'un élément de menu donne la valeur CHECKED% ou UNCHECKED% selon que l'élément de menu est coché ou non.

Par exemple :

```
MOVE CHECKED% TO ESTACTIF
; permet de cocher le menu ESTACTIF.
; La fonction GETTEXT$ lit le texte associé à un
; menu et l'instruction SETTEXT*
; change le texte associé à un menu.
```

Variables de type CONTROL

Comme vu précédemment dans les types, une variable NCL peut être de type CONTROL.

L'intérêt de telles variables est de pouvoir utiliser des fonctions et instructions NCL de manipulation de contrôle sans décrire de façon fixe le contrôle manipulé.

Par exemple :

```
GLOBAL CONTROL CTRL
MOVE BUTTON00001 TO @CTRL
HIDE CTRL
; Agit sur le Push-Button BUTTON00001
; L'instruction HIDE n'a pas encore été décrite.
; Elle permet de cacher un contrôle.
; Son utilisation directe serait :
; HIDE BUTTON00001
```

Il est à noter que l'affectation d'une variable contrôle doit être faite avec un « @ » précédant son nom.

Ce préfixe sert à pouvoir différencier la variable du contrôle associé.

Par exemple :

```
GLOBAL CONTROL CTRL
MOVE ENTRY00001 TO @CTRL
;la « variable » CTRL vaut ENTRY00001,
;c'est-à-dire agir sur CTRL est
;maintenant ;équivalent à agir sur ENTRY00001
MOVE ENTRY00002 TO CTRL
; Le contrôle associé à la variable CTRL prend la
;même valeur que le contrôle ENTRY00002 c'est-à-;dire le contenu de l'entry-field
ENTRY00002 est
;copié dans l'entry-field ENTRY00001.
```

Le type CONTROL prend surtout son importance pour des fonctions et instructions créées par le développeur et qui peuvent ainsi accepter des contrôles passés en paramètres.

Paramétrage dynamique des contrôles

La majorité des éléments de paramétrage peuvent être lus et modifiés en cours d'exécution d'une application à l'aide de qualificateurs dynamiques. Cette opération est dénommée « paramétrage dynamique ».

Le paramétrage dynamique d'un contrôle s'effectue en faisant suivre le nom du contrôle d'un qualificateur dynamique, le nom du contrôle et le qualificateur étant séparés par un point (« . ») :

```
Nom_objet.qualificateur_dynamique
```

Par exemple, pour indiquer de manière statique qu'une saisie est obligatoire dans un Entry-Field ENTRY1, il suffit de cocher « Required » dans son volet de propriétés.

Pour lire ce paramètre dans le code NCL, il suffit d'employer le qualificateur dynamique .REQUIRED :

```
IF ENTRY.REQUIRED
MESSAGE « Test », « L'Entry-Field est Required »
ELSE
MESSAGE « Test », « L'Entry-Field n'est pas Required »
ENDIF
```

Pour modifier dynamiquement le comportement de l'Entry-Field, il suffit de faire :

```
; L'Entry-Field est Required
MOVE TRUE% TO ENTRY1.REQUIRED
```

Ou

```
; L'Entry-Field n'est pas Required
MOVE FALSE% TO ENTRY1.REQUIRED
```

La liste complète des qualificateurs dynamiques disponibles est donnée dans le chapitre « Référence du Langage ».

UTILISATION DU LANGAGE

Ce chapitre présente les différentes possibilités de NCL (instructions, fonctions, opérateurs, événements, ...), classées thématiquement par familles, ou domaines d'application.

Un même mot-clé peut apparaître dans différentes familles, et chaque mot-clé de NCL apparaît toujours au moins une fois.

Traitements numériques

Types

NCL permet de manipuler des nombres entiers et des nombres réels. Les instructions INT et NUM permettent de déclarer des tableaux de nombres, tout en permettant de plus de modifier le stockage des nombres (intervalle plus restreint de nombres acceptés, avec moins de chiffres significatifs).

Caractères spéciaux

Le type d'une variable numérique est déterminé par le dernier caractère de son nom : % pour les variables entières et £ ou # pour les variables réelles. Pour mémoire, $ est utilisé pour les chaînes de caractères.

Voir la partie suivante pour tous les traitements de chaîne possibles avec NCL.

Les crochets [ ] sont utilisés pour les tableaux de variables déclarés avec LOCAL ou GLOBAL.
Les nombres peuvent être exprimés dans d'autres bases grâce à $ (hexa) et à # (base quelconque).
Les réels peuvent employer les caractères ., E, +, - pour la partie décimale et l'exposant.
Les parenthèses ( ) servent à forcer les priorités dans les expressions numériques.
Les opérations classiques sont présentes : + pour l'addition, - pour la soustraction, * pour la multiplication, / pour la division, % pour le reste de la division (entre entiers).
Les tests classiques sont aussi présents : = pour égal, <> pour différent, < pour inférieur, > pour supérieur, <= pour inférieur ou égal, >= pour supérieur ou égal.

Opérateurs et fonctions

Outre les opérateurs notés avec des caractères spéciaux (voir ci-dessus), d'autres opérateurs sont disponibles.

AND, OR, NOT sont les opérateurs logiques "et", "ou", "non".

BAND, BOR, BXOR, BNOT sont les opérateurs bit à bit "et", "ou inclusif", "ou exclusif", "non".

ABS donne la valeur absolue, ROUND% l'arrondi, et TRUNC% la partie entière.

INT et NUM forcent une expression à être de type entier et réel, alors que ISINT% et ISNUM% vérifient que le contenu d'une chaîne représente un entier ou un réel.

Il est à noter que des fonctions ou opérateurs différents peuvent être strictement équivalents dans le cas de certains types.

Par exemple, INT et TRUNC% sont équivalents dans le cas d'un opérande réel.

STRING force un type chaîne (STRING 1 donne "1").

LOW et HIW extraient les mots bas et hauts d'un entier, alors que DWORD% fabrique un entier à partir d'un mot haut et d'un mot bas. Ici, entier = 4 octets, et mot = 2 octets.

LOB et HIB extraient les octets bas et hauts d'un mot, alors que WORD% fabrique un mot à partir d'un octet haut et d'un octet bas.

Instructions

Trois instructions aident le traitement : LOCAL et GLOBAL pour déclarer des variables et dimensionner des tableaux, et surtout MOVE pour affecter des variables. MOVE, avec son action à la fois dans les traitements numériques ou chaînes et les traitements sur les contrôles (radio-buttons, entry-field, etc...) est sûrement L'INSTRUCTION LA PLUS IMPORTANTE DE NCL !

Traitements de chaînes de caractères

Caractères spéciaux

Le type d'une variable chaîne est déterminé par le dernier caractère de son nom, qui doit être un $.

Les crochets [ ] sont utilisés pour les tableaux. Les instructions LOCAL et GLOBAL précisent le nombre de caractères maximum de chaque élément du tableau (chaque chaîne a par défaut une longueur maximum de 255 caractères).

Les caractères " et ' délimitent les chaînes, ^ permet d'obtenir un caractère de contrôle (^C par exemple) et # permet d'obtenir un caractère avec son code ASCII (#13 par exemple).

Les concaténations de deux chaînes se font grâce à & et &&, ce dernier opérateur rajoutant un espace entre les deux chaînes.

Les comparaisons entre chaînes se font avec =, <>, <, >, <=, >=. Ici les tests sont faits par ordre alphabétique. Par exemple "Z" est supérieur à "A", "AC" est supérieur à "ABC", etc...

Opérateurs

Outre les opérateurs notés avec des caractères spéciaux, d'autres opérateurs sont disponibles :

INT et NUM forcent une chaîne à être de type entier et réel. Par exemple, si S$ vaut "12", INT S$ vaut 12.

STRING force un nombre entier ou réel à être de type chaîne. Par exemple, si I% vaut 12, STRING I% vaut "12".

UPCASE et LOWCASE convertissent les caractères des chaînes en majuscules et minuscules.

SKIP, LSKIP et RSKIP permettent de sauter les espaces de tête et de queue des chaînes (respectivement tête+queue, tête uniquement, queue uniquement).

LENGTH indique le nombre de caractères que contient une chaîne.

Fonctions et Instructions

ASC% donne le code ASCII d'un caractère, alors que CHR$ donne le caractère à partir de son code ASCII.

COPY$, DELETE$, INSERT$ permettent de manipuler des parties de chaînes : respectivement extraction, effacement, insertion de caractères dans une chaîne. FILLER$ remplit une chaîne avec le même caractère, et POS% donne la position de caractères dans une chaîne.

> Voir aussi la partie "Sous-Chaînes" du chapitre "Eléments du Langage" pour l'emploi direct des sous-chaînes grâce à : VARIABLE-CHAINE(ENTIER..ENTIER)

ISINT%, ISNUM% vérifient que le contenu d'une chaîne représente un entier, un réel.

Comme pour les traitements numériques, trois instructions aident le traitement : LOCAL, GLOBAL et MOVE.

Voir aussi la librairie NSDate qui permet de manipuler des chaînes de date et heure, la fonction STRING$ de la librairie NSMisc qui permet de formater des chaînes de caractère.

Autres traitements de données

Segments

SEGMENT/ENDSEGMENT définissent des nouveaux types de variables sous la forme de structures de données. Des variables peuvent ensuite être définies ayant le type segment.
De plus, NEW et DISPOSE permettent d'allouer et de désallouer des zones mémoires : les variables segment sont alors manipulées par leur adresse.

Définitions de variables

Des définitions détaillées de données sont faites grâce à LOCAL et GLOBAL, qui permettent de préciser le domaine de validité des données (locales ou globales), le type, la taille, et les dimensions du tableau.

De plus, SETDATA et GETDATA% permettent de manipuler des données locales à une fenêtre.

Tailles de variables

L'opérateur SIZEOF permet de connaître la taille en octets d'une variable ou d'un segment.

Instructions LOCAL, GLOBAL, SEGMENT, INSTRUCTION et FUNCTION

Désormais le champ <Type> peut contenir un nom de type suivi par @. Ceci à pour effet de définir une variable pointeur sur le type mentionné.
Cette syntaxe peut être utilisée dans les différentes déclarations : local et global, ou les définitions de segments et de fonctions ou instructions.
L'intérêt de cette syntaxe est de permettre à l'analyseur syntaxique de vérifier les affectations. On ne peut plus faire pointer un pointeur d'entier vers une chaîne de caractère, ou affecter un pointeur sur un segment A à un pointeur sur un segment B, le contrôleur est aussi capable de valider la saisie des variables de segment .

> Comme c'est une définition de pointeurs il est obligatoire de les initialiser pour éviter les erreurs générales de protection.

```
Syntaxe  LOCAL <type[@]> variable-name <(size)> <[dimension] <[dimension] ...> >
GLOBAL <type[@]> variable-name <(size)> <[dimension] <[dimension] ...> >
SEGMENT segmentname
 <type[@]> field-name1 <(size)> <[dimension] <[dimension] ...> >
  <type[@]> field-name2 <(size)> <[dimension] <[dimension] ...> >
  ...
ENDSEGMENT
INSTRUCTION instrname [ [type[@]] [@]parameter1 [(size)] [,
 [type[@]] [@]parameter2 [(size)] [, ...]]]
FUNCTION functname [( [type[@]] [@]parameter1 [(size)] [,
 [type[@]] [@]parameter2 [(size)] [, ]]
 )] [RETURN type[@]]
 ...
[RETURN value]
```

Ici, < > et [] représentent des champs optionnels et [[]] représente des dimensions.

Une case à cocher Reference dans la boîte de dialogue Modify Description for Global permet de déclarer les variables typées.

```
GLOBAL TYPE@ VAR
;La checkbox Reference correspond au '@' ci-dessus.
Exemple 1
Segment Seg_EtatCivil
Prenom$(31)
Nom$(31)
DateDeNaissance%
CodePostal$(7)
Ville$(63)
EndSegment ; {Seg_EtatCivil}

Segment Seg_Adresse
No$(7)
NomRue$(35)
CodePostal$(7)
Ville$(63)
Pays$(15) ; unused
```

```
EndSegment ; {Seg_Adresse}

Segment SEG_Personne
Seg_EtatCivil@ EtatCivil
Seg_Adresse@ Adresse
SEG_Personne@ Suivante ; liste chaînée, remplace POINTER Suivante
EndSegment ; {SEG_Personne}

Segment s_Ville
Ville$(31)[410] ; kNbVille%
EndSegment ; {s_Ville}

Global s_Ville@ pVillePPB1
Exemple 2
LOCAL @I1%(1), @I2%(2), @I4%, @N4#(4), @N8#, @S80$(80), @S255$, I%(1)
I%=0
; initializing the pointer
@I1% =I%

message 'I1%', I1% ; displays 0

I1% = I1% + 1
message 'I1%', I1% ; displays 1

I% = I% + 1
message 'I1%', I1% ; displays 2
Exemple 3
Segment New_SEG_Personne
Prenom$(31)
Nom$(31)
Age%
Adresse$
New_SEG_Personne@ Suivante ; liste chaînée, remplace POINTER ;Suivante
EndSegment ; {New_SEG_Personne}

Instruction NewEcritTout Control LB, New_SEG_Personne@ Personne

Local New_SEG_Personne@ TmpPers

@TmpPers = Personne
While @TmpPers <> 0
Insert At End \
TmpPers.Prenom$ & #9 & \
TmpPers.Nom$ & #9 & \
TmpPers.Age% & #9 & \
TmpPers.Adresse$ \
To LB
@TmpPers = TmpPers.Suivante ; type vérifié à la compilation NSGEN
EndWhile
EndInstruction ; {NewEcritTout}
```

Remplissage et copie de zones mémoires

L'instruction FILL remplit une zone mémoire avec le même octet reproduit sur la longueur désirée.
L'instruction MOV copie une zone mémoire vers une autre zone mémoire sur la longueur désirée.

Librairies

Noter les librairies NSMisc, NSDB et NSSQL (Oracle, Sybase, ...) qui permettent de gérer des données évoluées, respectivement : fichiers séquentiels, fichiers séquentiels indexés ISAM, bases SQL.

Logique

Caractères spéciaux

Les seuls caractères spéciaux sont les =, <>, <, >, <=, >= qui sont les opérateurs classiques permettant de comparer des entiers, réels, ou chaînes.

Opérateurs

AND, OR, NOT, BAND, BOR, BNOT, BXOR sont les opérateurs déjà vus pour les traitements numériques et traitements de chaîne. Ils sont très utilisés lors des tests, tout comme les opérateurs caractères ci-dessus.

Vrai ou faux ?

Deux constantes ont été prédéfinies dans NCL dans le but de simplifier les tests : TRUE% = 1 (vrai) et FALSE% = 0 (faux).

Etats logiques IS*%

Il est facile de tester certains états grâce aux nombreuses fonctions IS*% de NCL. La majorité de ces fonctions (attention, pas toutes, cf. référence !) retournent TRUE% ou FALSE%.

Ainsi, ISDISABLED% ou ISHIDDEN% indiquent si un contrôle est désactivé ou un objet invisible. ISMAXIMISED% ou ISMINIMIZE% indiquent si une fenêtre est maximisée ou minimisée (c'est-à-dire iconisée). ISINT%, ISNUM%, ISDATE% (librairie NSDate), ISTIME% (librairie NSDate) indiquent si le contenu d'une chaîne représente un entier, un réel, une date, ou une heure aux formats acceptés par NCL.

De plus la fonction ISMOUSE% indique si une souris est branchée sur la machine au moment où l'application s'exécute.

Tests IF

Un test se fait grâce à l'instruction IF ("si" en français). Tout IF doit être conclu par une instruction ENDIF. Lorsque l'expression située juste après le IF, sur la même ligne, vaut TRUE%, les instructions suivantes seront exécutées jusqu'au ENDIF (cas simple, sans ELSE). Lorsque l'expression est différente de TRUE% (FALSE% par exemple), il y a un branchement direct après le ENDIF.

Dans le cas IF/ELSE/ENDIF, TRUE% fait exécuter les instructions entre le IF et le ELSE, et FALSE% fait exécuter les instructions entre le ELSE et le ENDIF. Dans les deux cas (TRUE% ou FALSE%), l'exécution reprend ensuite après le ENDIF.

Il existe de plus une instruction ELSEIF qui permet de faire des nouveaux tests à l'intérieur d'un IF. Ce sont des constructions IF/ELSEIF/ELSEIF/ ... /ELSEIF/ELSE/ENDIF, le ELSE pouvant être absent (si le ELSE est présent il doit être unique et sans ELSEIF entre lui et le ENDIF final).

Instructions LOOP REPEAT et WHILE

NCL permet des constructions de boucle grâce à LOOP/ENDLOOP, REPEAT/UNTIL et WHILE/ENDWHILE.

LOOP/ENDLOOP exécute les instructions entre les deux mots-clés indéfiniment.

REPEAT/UNTIL ("répéter jusqu'à") et WHILE/ENDWHILE ("tant que") exécutent les instructions entre les deux mots-clés jusqu'à ce que la condition soit TRUE% et tant que la condition est TRUE%. La condition est testée APRES chaque boucle dans le cas de REPEAT/UNTIL, alors qu'elle est testée AVANT chaque boucle dans le cas de WHILE/ENDWHILE. En conséquence, la différence essentielle est que les instructions entre REPEAT/UNTIL sont toujours exécutées au moins une fois, quelle que soit la condition.

BREAK et CONTINUE, qui peuvent s'employer uniquement au sein de constructions de type LOOP/ENDLOOP, REPEAT/UNTIL et WHILE/ENDWHILE, permettent de modifier le déroulement normal des boucles : BREAK arrête complètement la boucle, et CONTINUE fait passer à la boucle suivante.

Instruction EVALUATE

L'instruction EVALUATE / WHERE...ENDWHERE / ... / ENDEVALUATE permet d'exécuter une partie de code selon la valeur d'une expression. Cette instruction est équivalente au "case" du Pascal ou au "switch" du C, et évite une suite fastidieuse de IF.

Librairies de Fonctions et Instructions

FUNCTION/ENDFUNCTION et INSTRUCTION/ENDINSTRUCTION permettent de se définir ses propres fonctions et instructions, qui deviennent ensuite utilisables au même titre que les fonctions et instructions prédéfinies de NCL.

> Voir à la fin de ce chapitre des explications détaillées sur FUNCTION EXTERNAL et INSTRUCTION EXTERNAL qui permettent d'appeler des DLLs externes.

LOADDLL, UNLOADDLL, GETPROC, DYNAMIC permettent de gérer dynamiquement le chargement/déchargement d'une DLL et l'appel aux fonctions y figurant.

Nouvelle syntaxe d'appel d'une fonction

Nouvelle syntaxe d'appel d'une instruction

Instructions LOCAL, GLOBAL, SEGMENT, INSTRUCTION et FUNCTION

Temporisation

WAIT permet d'attendre un temps donné en milli-secondes. L'application est bloquée jusqu'à la fin de cette temporisation.

Arrêts d'exécution

EXIT permet de sortir du traitement de l'événement en cours, l'exécution étant reprise pour l'événement suivant, alors que HALT arrête complètement l'application.

BREAK permet de sortir d'une boucle. RETURN permet de clôturer le traitement d'une fonction, d'une instruction ou d'un événement.

Autres appels

Des instructions, qui ne sont pas des instructions de logique pure, ont pour conséquence des appels à d'autres parties de code de la même application.

C'est par exemple le cas de CALL qui démarre une nouvelle fenêtre : des événements vont ensuite circuler au sein de la fenêtre démarrée (et donc fort probablement du code NCL associé), et l'instruction derrière CALL ne sera exécutée que lorsque la fenêtre démarrée aura été fermée.

C'est aussi le cas de SEND qui envoie un événement : l'instruction derrière le SEND ne sera exécutée que lorsque le code associé à l'événement envoyé aura été exécuté.

C'est souvent le cas de nombreuses fonctions et instructions, notamment celles concernant le fenêtrage et les contrôles (cf. plus loin ces familles), ont pour effet indirect de faire générer des événements, et donc des exécution de code NCL. Par exemple INVALIDATE, SELECT, MINIMIZE, etc... et même MOVE !

Evénements

Paramètres

Chacun des événements décrits rapidement dans le premier chapitre, puis détaillés dans le chapitre "Référence des Evénements", est envoyé avec quatre paramètres qui sont PARAM1%, PARAM2%, PARAM3%, et PARAM4%. Leur signification dépend de chacun des événements. Par exemple :

- PARAM3% de CHARACTER donne le code ASCII de la touche appuyée,
- PARAM1% de MOUSEMOVE donne la position horizontale de la souris,
- PARAM1% de SELECTED donne
- pour une liste : l'index de l'élément sélectionné
- pour un contrôle Classeur : le numéro de la fenêtre à onglet sélectionnée.

PARAM12% et PARAM34% sont des facilités de programmation permettant d'obtenir deux entiers combinés à partir des quatre entiers précédents. PARAM1$ et PARAM2$ sont aussi des facilités de programmation permettant de travailler avec des chaînes dont l'adresse est stockée dans PARAM12% et PARAM34%.

Circulation

La circulation des événements est pilotée par le système d'exploitation qui envoie le bon événement à l'objet correspondant. Les événements sont souvents provoqués par l'utilisateur (déplacement de souris ou mouvement de fenêtre par exemple), par le système lui-même (timer par exemple), ou par l'application.

Dans ce dernier cas, une instruction spéciale, SEND, permet d'envoyer un événement directement à tout objet de l'application, sans passer par la circulation pilotée par le système d'exploitation.

Il faut également prendre en compte de nombreux effets de bord qui compliquent les choses. Ainsi un simple clic souris sur une list-box d'une fenêtre qui n'était ni active ni complètement visible, peut faire générer une multitude d'événements, comme par exemple un LOSEFOCUS à l'ancien objet ayant le focus, un PAINT à la fenêtre sélectionnée, un GETFOCUS et un SELECTED à la list-box, etc... Cf. "Autres Appels" de la famille "Logique".

Le multi-tâches

Une caractéristique importante du multi-tâches est à prendre en compte : lorsqu'un événement est traité, aucun autre événement n'est traité en parallèle, ce qui assure un fonctionnement "mono-tâche".

Les événements se déroulent toujours séquentiellement. Seuls SEND et CALL font dérouler un ou plusieurs autres événements, mais, même dans ce cas, il n'y a pas de déroulement de code en parallèle, puisque ces instructions ne retournent qu'après la fin de traitement de l'événement (ou de la fenêtre) appelé.

Surclasser

Il est possible de modifier le comportement standard d'un contrôle grâce à des événements ayant généralement le même nom que les fonctions et instructions de contrôles (SETTEXT, SETFORECOL, LOAD, DISABLE, ...) et appelés automatiquement dès que la fonction ou instruction associée est utilisée sur le contrôle.
Un nouveau comportement peut y être défini tout en utilisant éventuellement le comportement par défaut grâce à l'instruction PASS.
Voir aussi "Templates".

Timer

Un timer est un événement spécial, nommé TIMER, qui peut être reçu par les fenêtres (uniquement par elles, les contrôles ne pouvant le recevoir).

Lorsque l'instruction STARTTIMER est appelée, cet événement est envoyé régulièrement à un intervalle de temps déterminé par l'application. Cela peut par exemple permettre d'être assuré de remettre à jour régulièrement telle ou telle zone de la fenêtre, d'y afficher l'heure, etc...

En effet, à cause de l'architecture événementielle du système d'exploitation, il est impossible pour une application de garder la main sur un événement pour, par exemple, afficher l'heure en permanence dans un champ : la circulation d'événements serait alors stoppée, et donc en conséquence le fonctionnement multi-tâches du système d'exploitation. Les timers permettent de se sortir de ce genre de problème en permettant aux applications de demander au système de les "réveiller" régulièrement. Les applications n'ont ainsi plus besoin d'avoir des boucles infinies de traitement grâce à cette certitude d'être appelées régulièrement sur l'événement TIMER.

L'événement TIMER respecte lui-même l'architecture événementielle du système d'exploitation : il n'interrompt jamais un événement en cours de traitement. L'événement TIMER n'est aucunement prioritaire sur les autres événements, ce qui explique que le système d'exploitation ne garantisse pas que le délai demandé lors du démarrage du timer soit entièrement respecté. Le délai réel dépend des temps de traitement des événements.

L'instruction STOPTIMER arrête le timer.

Il est à noter qu'une même application peut démarrer plusieurs timers.

Il existe de plus une instruction WAIT qui permet de stopper provisoirement l'exécution pendant un délai donné en milli-secondes. Toujours pour les raisons de structure événementielle, il est déconseillé d'utiliser WAIT avec une temporisation trop longue : plus aucun événement ne circule au sein de l'application pendant cette temporisation !

Boîtes de messages

NCL permet de démarrer des boîtes de messages. Les boîtes de message sont des boîtes de dialogue simplifiées ne comportant qu'une barre de titre, un texte (le "message"), et un, deux, ou trois push-buttons pour préciser le choix. Chacun de ces boutons a pour effet immédiat de fermer la boîte. Ces boîtes sont modales : aucun autre objet de la même application ne peut avoir la main tant que la boîte reste ouverte.

Vu côté programmation, cette modalité se traduit par le fait que les trois fonctions ou instructions listées ici ne retournent que lorsque la boîte est fermée. Chacune d'elles comporte deux paramètres : le titre, et le message.

La fonction ASK3% démarre une boîte de messages avec les trois boutons Yes, No, Cancel, et retourne un entier identifiant le bouton enfoncé.

La fonction ASK2% démarre une boîte de messages avec les deux boutons Yes, No, et retourne un entier identifiant le bouton enfoncé.

L'instruction MESSAGE démarre une boîte de messages avec l'unique bouton OK. Ce n'est pas une fonction, puisque l'on est sûr de connaître le bouton enfoncé...

Les constantes prédéfinies YES%, NO% et CANCEL% permettent de tester facilement le retour de ASK2% et ASK3%. Il est à noter que ASK2% ne peut jamais retourner CANCEL%.

Objets: Fenêtres et Contrôles

Généralités

Plusieurs fonctions et instructions fonctionnent de la même façon sur les contrôles et les fenêtres : on parle alors d'"objet" qui peut représenter indifféremment un contrôle ou une fenêtre.

Dans ce cas, le paramètre spécifiant l'objet est optionnel : s'il n'est pas indiqué, la fonction ou instruction affecte la fenêtre courante.

Voici un exemple montrant toutes les façons d'employer SETFOCUS. Cet exemple est valable pour TOUTES les fonctions et instructions de ce paragraphe.

```
SETFOCUS ; Sur la fenêtre courante
SETFOCUS CONTROLNAME ; Sur le contrôle spécifié
SETFOCUS H% ; Sur la fenêtre de handle H%
SETFOCUS SCREENNAME(H%) ; Sur la fenêtre de nom SCREENNAME
 ; et de handle H%
SETFOCUS SCREENNAME(H%).CONTROLNAME
; Sur le contrôle appartenant à la fenêtre de nom SCREENNAME et de
; handle H%
SETFOCUS 2 * CONTROLNAME + 1
; Dès qu'il s'agit d'une expression, cela agit sur une fenêtre !
; Par exemple ici, si CONTROLNAME est un ; Entry-Field contenant 14,
; cela agit sur la fenêtre dont le handle vaut 29 !
; Attention, cet exemple est donné à titre d'illustration. Il est très ; dangereux de
considérer un handle comme valeur constante !
```

Fonctions et instructions

SHOW montre et HIDE cache un objet, alors que ISHIDDEN% permet de savoir si un objet est caché. Après un HIDE, l'objet existe toujours mais n'est plus visible : l'utilisateur ne peut donc plus le manipuler.

SETFORECOL et SETBACKCOL changent les couleurs d'avant-plan et d'arrière-plan d'un objet. GETFORECOL% et GETBACKCOL% permettent de connaître les couleurs d'avant-plan et d'arrière-plan d'un objet. Ces instructions et fonctions utilisent en paramètre les constantes COL *% spécifiant les couleurs.

SETFOCUS donne le focus à un objet.

SETTEXT modifie le texte d'un objet et GETTEXT$ permet de connaître le texte d'un objet. Par exemple, pour une fenêtre il s'agit de son titre, pour un entry-field il s'agit du texte saisi, pour un Static-Text du texte affiché. Dans certains cas (Static-Text, Entry-Field, ...), SETTEXT et GETTEXT$ sont équivalents à un MOVE sur l'objet.

SETPOS change la position d'un objet. GETXPOS% et GETYPOS% permettent de connaître la position d'un objet.

GETHEIGHT% et GETWIDTH% permettent de connaître la taille horizontale et verticale d'un objet.

Evénements

INIT est le premier événement qui arrive à un objet lors de sa création.

CHECK est un événement qui permet de vérifier l'objet avant la fermeture de la fenêtre.

GETFOCUS et LOSEFOCUS sont des événements qui indiquent l'obtention et la perte du focus. L'objet qui a le focus est l'objet vers lequel sont dirigées les actions clavier de l'utilisateur.

HELP est reçu dès que l'utilisateur appuie sur la touche [F1] par l'objet qui a le focus.

CHANGED est reçu par une fenêtre lorsque sa taille a changé, ou par un contrôle (Entry-Field ou CBE) lorsque son contenu est changé.

Evénements

Outre les événements que tous les objets peuvent recevoir (voir la partie précédente), voici les événements que les fenêtres sont les seules à recevoir, les contrôles ne les recevant jamais.

PAINT est reçu dès que la fenêtre est à repeindre. Il est surtout utilisé avec la librairie graphique NSGraph, et dans le cas de fenêtres de classe Window.

TERMINATE est reçu lors de la fermeture de la fenêtre.

HSCROLL et VSCROLL sont reçus lorsque l'utilisateur déroule les barres de défilement de fenêtres ayant "Horz Scroll-Bar" ou "Vert Scroll-Bar" dans le volet de propriétés de la fenêtre.

> Les événements DDE * sont reçus par les applications utilisant le DDE pour communiquer entre elles. Voir la librairie NSComm.

Les fenêtres reçoivent aussi des événements timer, clavier et souris.

Fonctions

MAINWINDOW% retourne le handle de la fenêtre principale de l'application (la fenêtre qui a été initialement démarrée), et SELF% retourne le handle de la fenêtre courante.

PARENTWINDOW% retourne le hande de la fenêtre mère d'une fenêtre.

SELF permet de qualifier un template.

GETCLIENTHEIGHT% et GETCLIENTWIDTH% retournent la hauteur et la largeur en pixels de la fenêtre.

ISMAXIMIZED% et ISMINIMIZED% permettent de savoir si la fenêtre est maximisée ou minimisée (= icônisée).

> Voir le paragraphe "Listes" pour une description complète des fenêtres de classe List et Edit.

Instructions

CALL, CALLH, STRCALL, STRCALLH, OPEN, OPENH, OPENS, OPENSH, STROPEN, STROPENH, STROPENH et STROPENSH ouvrent une fenêtre.

CLOSE ferme une fenêtre.

INVALIDATE invalide le contenu d'une fenêtre et génère un événement PAINT.

CHANGE change une fenêtre de position et de taille.

MAXIMIZE, MINIMIZE, RESTORE modifient aussi la taille d'une fenêtre : maximisation, minimisation, et restauration.

SETRANGE s'emploie dans le cas où le style "Horz Scroll Bar" ou "Vert Scroll Bar" est coché dans le volet de propriétés de la fenêtre, pour préciser le minimum et maximum ainsi que la nouvelle position de l'ascenseur.

Contrôles

Evénements

Outre les événements que tous les objets peuvent recevoir (voir plus haut), voici les deux seuls événements que les contrôles sont les seuls à recevoir, les fenêtres ne les recevant jamais.

SELECTED est généré :

- lorsqu'un élément de menu est en cours de sélection,
- lorsque l'utilisateur sélectionne un élément d'une liste, une fenêtre à onglets d'un contrôle Classeur ou un bouton (focus sur Push-Button ou clic sur Check-Box, Radio-Button),
- lorsque l'utilisateur déroule un contrôle ascenseur.

EXECUTED est généré :

- lorsqu'un élément de menu est choisi,
- lorsque l'utilisateur sélectionne un élément de liste ou un bouton (clic sur Push-Button ou double-clic sur Check-Box et Radio-Button).

Fonctions et Instructions

L'instruction MOVE est la plus employée puisqu'elle permet de connaître le contenu du contrôle ou d'en modifier le contenu.

Voir aussi...

- le paragraphe suivant, "Listes", pour une description complète des contrôles listes (List-Box, Combo-Box, CBE, MLE).
- le paragraphe "Contrôles de base" dans le chapitre "Eléments du Langage" pour une description complète de tous les contrôles.

SETRANGE s'emploie dans le cas d'ascenseur pour préciser un nouveau minimum et maximum (lesquels peuvent être initialisés grâce aux champs Minimum et Maximum de leur volet de propriétés).

ENABLE active et DISABLE désactive un contrôle, alors que ISDISABLED% permet de savoir si un contrôle est désactivé. Par exemple, faire DISABLE sur un Push-Button le grise et empêche l'utilisateur de l'enfoncer : l'événement EXECUTED ne peut donc plus lui arriver.

LOCK et UNLOCK s'emploient pour bloquer et débloquer les entrées utilisateur : par exemple LOCK rend impossible toute saisie dans un contrôle Entry-Field, CBE et MLE. ISLOCKED% permet de savoir si un contrôle est bloqué.

Surclasser

Il est possible de modifier le comportement standard d'un contrôle grâce à des événements ayant généralement le même nom que les fonctions et instructions de contrôles (SETTEXT, SETFORECOL, LOAD, DISABLE, ...) et appelés automatiquement dès que la fonction ou instruction associée est utilisée sur l'objet.

Un nouveau comportement peut y être défini tout en utilisant éventuellement le comportement par défaut grâce à l'instruction PASS.

Voir aussi plus loin "Templates".

Paramétrage dynamique

Il est possible de connaître et de modifier dynamiquement le paramétrage des contrôles. Voici des exemples de paramétrages "dynamiques" : .BACKCOLOR, .FILLTEXT, .FONTNAME, .JUSTIFICATION, .RELIEF, .WORDWRAP, ...

Des constantes permettent de manipuler certains paramétrages : JUST *%, REL *%, TYP *%.

Paramétrage dynamique des contrôles

La majorité des éléments de paramétrage peuvent être lus et modifiés en cours d'exécution d'une application à l'aide de qualificateurs dynamiques. Cette opération est dénommée "paramétrage dynamique".

La paramétrage dynamique d'un contrôle s'effectue en faisant suivre le nom du contrôle d'un qualificateur dynamique, le nom du contrôle et le qualificateur étant séparés par un point (".") :

```
nom_objet.qualificateur_dynamique
```

Par exemple, pour indiquer de manière statique qu'une saisie est obligatoire dans un Entry-Field ENTRY1, il suffit de cocher "Required" dans son volet de propriétés.

Pour lire ce paramètre dans le code NCL, il suffit d'employer le qualificateur dynamique .REQUIRED :

```
IF ENTRY1.REQUIRED
MESSAGE "Test", "L'Entry-Field est Required"
ELSE
MESSAGE "Test", "L'Entry-Field n'est pas Required"
ENDIF
```

Pour modifier dynamiquement le comportement de l'Entry Field, il suffit de faire :

```
; L'Entry-Field est Required
MOVE TRUE% TO ENTRY1.REQUIRED
```

ou :

```
; L'Entry-Field n'est pas Required
MOVE FALSE% TO ENTRY1.REQUIRED
```

> La liste complète des qualificateurs dynamiques disponibles est donnée dans le chapitre "Référence du Langage".

Listes

Généralités

Une "liste" peut être un contrôle :

- List-Box
- Combo-Box
- CBE
- MLE

ou une fenêtre de classe :

- Edit
- List

Les fonctionnalités standards des List box et des Combo box sont décrites dans l'annexe E de ce manuel.

Fenêtre de classe Edit ou List

L'aire client d'une fenêtre de classe List peut se manipuler comme un contrôle List-Box, et l'aire client d'une fenêtre de classe Edit peut se manipuler comme un contrôle MLE. Il suffit d'utiliser CLIENT comme nom de contrôle afin de désigner l'aire client.
Ainsi, pour connaître le texte de la ligne sélectionnée d'une fenêtre SCREEN.SCR de classe List et de handle H%, il suffit de faire :

```
MOVE SCREEN(H%).CLIENT TO S$
```

ou plus simplement si le MOVE est effectué depuis la fenêtre elle-même :

```
MOVE CLIENT TO S$
```

Toutes les manipulations décrites ci-après fonctionnent de la même façon sur ces fenêtres et sur les contrôles listes.

Chargement et sauvegarde d'un fichier

Pour remplir une liste avec un fichier texte, il suffit d'indiquer le nom du fichier dans le champ Init de son le volet de propriétés (et en dégrisant la case Preload), ou d'utiliser l'instruction LOAD.

Pour sauver dans un fichier le contenu d'une liste, utiliser l'instruction SAVE.

Pour charger ou sauvegarder le fichier en ordre alphabétique normal ou inverse, préciser ASCENDING ou DESCENDING derrière LOAD ou SAVE.

Modification du contenu de la liste

Si NAME est le nom de la liste, et qu'une ligne est sélectionnée, alors MOVE S$ TO NAME remplace le texte de la ligne sélectionnée par la chaîne S$.

INSERT insère une ligne en milieu de liste (INSERT AT position chaîne TO objet), en fin de liste (INSERT AT END chaîne TO objet), dans l'ordre alphabétique normal (INSERT ASCENDING chaîne TO objet) ou dans l'ordre alphabétique inverse (INSERT DESCENDING chaîne TO objet).

DELETE supprime une ligne (DELETE AT position FROM objet) ou toutes les lignes (DELETE FROM objet).

Chaque INSERT ou DELETE est effectué directement à l'affichage, ce qui a pour effet désagréable de faire clignoter la liste lors de nombreuses modifications successives. Pour éviter cela, NOUPDATE bloque la mise à jour d'une liste. Un UPDATE ultérieur débloque la liste en la mettant complètement à jour.

Nombre de lignes

LINECOUNT% retourne le nombre de lignes d'une liste.

Sélection d'une ligne

Il suffit à l'utilisateur de cliquer sur une ligne pour la sélectionner. Il est aussi possible de le faire en NCL: SELECT sélectionne une ligne et UNSELECT désélectionne une ligne. Le champ Init du volet de propriétés sélectionne une ligne dès l'apparition de la liste.

SELECTION% retourne le numéro de la ligne sélectionnée. Pour sauvegarder ce numéro lors de la fermeture de la fenêtre, indiquer le nom d'une variable NCL dans le champ Output du volet de propriétés. La constante NOSELECTION% indique qu'aucune ligne n'a été sélectionnée.

Dans le cas de list-box à sélection multiple, la fonction ISSELECTED% permet de savoir si une ligne est sélectionnée.

Que ce soit par action de l'utilisateur (simple-clic) ou par programmation (instruction SELECT), dès qu'une ligne est sélectionnée, un événement SELECTED est généré. De même, lors d'un double-clic sur la liste, un événement EXECUTED est généré. Pour ces deux événements, PARAM1% indique directement le numéro de la ligne sélectionnée.

Contenu d'une ligne

Pour connaître le texte affiché dans une ligne de la liste, il faut indexer le nom de l'objet. Par exemple, si NAME est le nom de la liste, NAME[0] retourne le texte de la première ligne.

Le nom de l'objet, sans index, retourne le texte de la ligne sélectionnée, ou la chaîne vide lorsqu'aucune ligne est sélectionnée.

Formatage d'une liste

Il est possible de formater chaque ligne d'une List Box ou d'une Combo box : changer les couleurs, justifier, centrer, caler à gauche, caler à droite, encadrer, etc...

Voir aussi l'annexe E contient une description détaillée du formatage des List Box et des Combo box, les paramétrages dynamiques qui permettent de gérer les listes : .TABULATIONS, .SEPARATORS.

Contrôles Classeurs

Généralités

Un contrôle Classeur est un contrôle qui permet de simuler dans une fenêtre le fonctionnement d'un classeur qui regroupe des fenêtres à onglet composées de fiche. Chaque fiche présente plusieurs contrôles qui peuvent être de n'importe quel type (Entry Field, Radio Button, ...). Une fiche rassemble des contrôles de façon thématique.
Un contrôle Classeur a toujours une fenêtre à onglet sélectionnée : celle qui est affichée au premier plan.

Nombre de fenêtres à onglet dans un contrôle classeur

LINECOUNT% retourne le nombre de fenêtres à onglet dans un contrôle classeur.

Sélection d'une fenêtre à onglet dans un contrôle classeur

Il suffit à l'utilisateur de cliquer sur un onglet pour sélectionner la fenêtre à onglets correspondante.
Il est aussi possible de le programmer en langage NCL avec les verbes suivants :

- SELECT : sélectionne une fenêtre à onglet.
- SELECTION% : retourne le numéro de la fenêtre à onglet sélectionnée.

> Se reporter au chapitre "Référence du Langage" pour une description complète de ces APIs.

Que ce soit par action de l'utilisateur (simple-clic) ou par programmation (instruction SELECT), dès qu'une fenêtre à onglet est sélectionnée, un événement SELECTED est généré.

Menus

Dès qu'une fenêtre a Menu Bar coché dans son volet de propriétés, elle possède une barre de menus.
Ces menus sont considérés comme des contrôles : les fonctions et instructions vues plus haut fonctionnent donc également sur eux.
Voici toutefois un résumé sur la façon de les gérer.

Fonctions et instructions

Quelques fonctions et instructions suffisent à traiter complètement les éléments de menus :

```
MOVE CHECKED% TO MENU ; coche un élément de menu
MOVE UNCHECKED% TO MENU ; décoche un élément de menu
MOVE MENU TO A%  ; A% vaut CHECKED% si le menu est coché
DISABLE MENU ; désactive (= grise) un élément de menu
ENABLE MENU ; active un élément de menu
MOVE GETTEXT$(MENU) TO S$ ; lit le texte du menu
SETTEXT S$ TO MENU ; modifie le texte du menu
HIDE MENU ; cache un élément de menu
SHOW MENU ; montre un élément de menu
```

Lorsque DISABLE / ENABLE ou HIDE / SHOW est appliqué à un menu déroulant, cela affecte également la totalité des éléments qu'il contient.

Evénements

Les événements sont au nombre de quatre :

- INIT lors du premier déroulement du menu
- HELP lorsque [F1] est enfoncé
- SELECTED lorsque le menu est en cours de sélection
- EXECUTED lorsque le menu a été réellement sélectionné

L'événement SELECTED peut par exemple servir à afficher une chaîne de caractères en bas de la fenêtre expliquant la signification des menus au fur et à mesure de leur sélection, comme le font par exemple Microsoft Excel ou Microsoft Word.

Affectation

Pour pouvoir accéder par MOVE à un contrôle Template, il est nécessaire de traiter les événements GETVALUE et SETVALUE du Template.

Fonctions et Instructions

Pour qu'un contrôle Template sache réagir sur les instructions NCL de manipulation de contrôles (comme SETTEXT, SETPOS, HIDE, etc ...), il est nécessaire de traiter les événements portant généralement le même nom (comme SETTEXT, SETPOS, HIDE, etc...).
Pour qu'un contrôle Template sache réagir sur les fonctions NCL de manipulation de contrôles (comme GETTEXT$, LINECOUNT%, ISDISABLED%, etc ...), il est nécessaire de traiter les événements portant généralement le même nom mais sans le caractère de terminaison (comme GETTEXT, LINECOUNT, ISDISABLED, etc...), et de les terminer par un RETURN afin de retourner une valeur à la fonction NCL appelée.

Surclasser les templates

Il est important de noter que tous les événements de comportement d'un template sont doublés : il en existe au niveau du template lui-même (fichier .TPL) ainsi qu'au niveau du contrôle template. Tout événement est d'abord envoyé au contrôle template, puis éventuellement transmis au template lui-même.

Prenons un exemple. Supposons une instruction SETTEXT effectuée sur un contrôle template (nommé CTRLT) depuis n'importe quel script de la fenêtre (nommé SCREEN.SCR) incorporant le template (nommé TEMPL.TPL) :

```
; Evénement quelconque d'un objet de SCREEN
SETTEXT "Bonjour" TO CTRLT
```

Un événement SETTEXT est d'abord envoyé au contrôle CTRLT. Puis deux possibilités :

1. Si cet événement n'est pas traité, il est automatiquement transmis à TEMPL dont le script de l'événement SETTEXT est alors exécuté.
2. Si cet événement est traité, son script est exécuté. Le script de l'événement SETTEXT de TEMPL n'est exécuté que si PASS ou RETURN DEFRET% est employé au sein du script de CTRLT.

Pour résumer, traiter les événements du template lui-même définit son comportement "standard". Si les événements au niveau du contrôle template sont traités, le comportement est "modifié" : c'est ce que signifie "surclasser".

Gestion du clavier

Clavier pour les contrôles

Le clavier est automatiquement géré pour les contrôles. Par exemple, l'application n'a pas à gérer le clavier sur un entry-field : automatiquement, le système d'exploitation se charge d'y afficher les caractères au fur et à mesure de la saisie, d'y gérer le backspace, les flèches de direction, le presse-papiers, etc... De même, le Tab est géré pour passer le focus d'un contrôle au contrôle suivant, la barre d'espace sur un check-box, les flèches de direction sur les contrôles ascenseur, sur les groupes de radio-buttons ou sur les menus. Et ainsi de suite pour tous les contrôles...

Des événements peuvent toutefois être générés vers les contrôles dans certains cas : CHANGED dès qu'une touche est enfoncée dans un Entry-Field, SELECTED dès qu'une touche curseur est enfoncée dans un List-Box, EXECUTED dès que la touche [Return] est enfoncée sur un Push-Button, HELP dès que la touche [F1] est enfoncée sur tout type de contrôle, etc...

Clavier pour les fenêtres

Une application a uniquement à gérer le clavier pour les fenêtres. Et encore..., ce n'est pas toujours nécessaire : une application composée uniquement de boîtes de dialogue classiques (c'est-à-dire une application où tout est géré par contrôles : fenêtres de classe Dialog) n'a aucunement besoin de gérer le clavier.

Contrairement à une application écrite de manière classique, il n'y a aucune fonction ou instruction permettant de savoir à un instant donné s'il y a une touche enfoncée. En conséquence de la structure événementielle du système d'exploitation, une application ne peut avoir une boucle infinie attendant l'appui d'une touche, cela bloquerait le multi-tâches.

La méthode élégante choisie par le système d'exploitation est de gérer le clavier sur un seul et unique événement : CHARACTER. Cet événement est automatiquement envoyé à la fenêtre qui a le focus. Une application n'a donc pas à programmer l'attente d'une touche : elle est automatiquement prévenue dès le moindre appui grâce à l'événement CHARACTER.

Evénement CHARACTER

Cet événement est en fait généré dès qu'il y a un changement d'état sur le clavier (appui ou relâchement d'une touche ou même d'une combinaison de touches), mais aussi dès qu'une touche reste enfoncée (repeat count). Avec cet événement, les quatre paramètres ont chacun une signification particulière.

Le paramètre PARAM4% contient une constante VK * lorsque la touche est "virtuelle". Les touches virtuelles correspondent essentiellement aux touches qui n'ont pas de code ASCII (Caps Lock, touches de fonction, Home, Shift, ...).

Le paramètre PARAM3% contient le code ASCII (lorsqu'il existe) de la touche.

Le paramètre PARAM2% contient le "repeat count" qui, lorsqu'il est différent de 1, indique que la touche a été laissée appuyée un certain temps.

Et le paramètre PARAM1% contient le "type" de la touche, codée avec une combinaison des constantes KF *. Par exemple, il permet d'indiquer si PARAM3% et PARAM4% sont significatifs (touche virtuelle et touche avec code ASCII), s'il s'agit d'une enfoncement de touche ou d'une remontée, etc...

L'événement CHARACTER est peut-être le plus complet des événements, où chaque paramètre a son importance. De par sa richesse, il peut s'avérer complexe à traiter avec notamment les nombreuses combinaisons possibles de PARAM1%. Une façon simplifiée de le traiter est donnée en exemple dans le chapitre "Référence des Evénements".

Gestion de la souris

Souris pour les contrôles

Notez les nombreuses similitudes de fonctionnement de la souris et du clavier sur les contrôles : ces deux périphériques y sont automatiquement gérés, facilitant notablement le travail du programmeur.

La souris est automatiquement gérée pour les contrôles. Par exemple, l'application n'a pas à gérer la souris sur un entry-field : automatiquement, le système d'exploitation se charge de positionner le curseur clignotant de saisie à l'endroit du clic. De même, un clic sur un Push-Button provoque automatiquement son enfoncement, un clic sur un élément d'une list-box provoque sa sélection, un clic sur un check-box le coche s'il n'était pas coché ou le décoche s'il était coché, un drag sur un contrôle Scroll-Bar déplace l'ascenseur, et ainsi de suite pour de nombreux contrôles... De même pour les menus, un clic sur un menu déroulant fait automatiquement dérouler le menu. De nombreux exemples peuvent ainsi être cités montrant que les contrôles réagissent automatiquement à la souris sans avoir à recourir à la moindre programmation.

Des événements sont souvent générés vers les contrôles, simultanément aux actions déjà citées : GETFOCUS dès qu'un clic a été fait sur un contrôle qui ne possédait pas préalablement le focus, EXECUTED dès l'enfoncement d'un Push-Button, SELECTED dès la sélection d'un élément de list-box, SELECTED dès l'appui sur un check-box, encore SELECTED dès le déroulement d'un scroll-bar, etc...

Souris pour les fenêtres

Une application a uniquement à gérer la souris pour les fenêtres. Cela n'est cependant pas systématique : une application composée uniquement de boîtes de dialogue classiques (c'est-à-dire une application où tout est géré par contrôles : fenêtres de classe Dialog) n'a aucunement besoin de gérer la souris.

Contrairement à une application écrite de manière classique, il n'y a aucune fonction ou instruction permettant de savoir à un instant donné si la souris est manipulée. En conséquence de la structure événementielle de Windows, une application ne peut avoir une boucle infinie attendant l'appui d'un bouton de souris, cela bloquerait le multi-tâches.

La méthode élégante choisie par le système d'exploitation est de gérer la souris sur quelques événements : MOUSEMOVE (déplacement souris), BUTTONDOWN (bouton souris enfoncé), BUTTONUP (bouton souris relâché), BUTTONDBLCLK (double-clic). Ces événements sont automatiquement envoyés à la fenêtre qui est située physiquement sous le pointeur souris. Une application n'a donc pas à programmer l'attente d'un clic : elle est automatiquement prévenue dès la moindre manipulation de la souris grâce aux événements cités. La constante MOU *% permet de connaître le type de bouton enfoncé lors de l'occurrence de l'un des événements BUTTON*.

ISMOUSE% permet de savoir si une souris est connectée à la machine sur laquelle l'application est en train de tourner. En effet, même si elle est recommandée, la souris reste un périphérique optionnel sur des systèmes comme Windows. Il est donc conseillé de toujours proposer des équivalents clavier à la totalité des actions souris de façon à ne pas pénaliser l'utilisateur qui n'en possède pas ou ... qui n'aime pas s'en servir. Heureusement, comme nous l'avons déjà vu, le clavier est automatiquement géré sur la totalité des contrôles facilitant ainsi ce travail d'équivalence.

NBBUTTONS% permet de connaître le nombre de boutons de la souris. Généralement, les souris ont deux boutons : c'est le cas des souris IBM et Microsoft. Mais attention, il existe des souris à un bouton et à trois boutons.

Par défaut, le pointeur souris est représenté par une flèche pointant vers le haut gauche. NatStar permet de changer ce pointeur grâce à l'instruction SETPTR utilisée avec la fonction GETSPTR% et une constante SPTR *% représentant le pointeur choisi. Il est par exemple conseillé, pour le confort de l'utilisateur, de changer le pointeur en sablier dès qu'un traitement peut être long.

Enfin, il est possible de "capturer" les événements souris grâce à l'instruction CAPTURE : ils sont alors tous dirigés vers la fenêtre spécifiée et non plus vers la fenêtre située physiquement sous le pointeur souris. Cette capture se termine grâce à l'instruction UNCAPTURE.

> Un exemple d'utilisation de ces fonctions est donnée dans le chapitre sur le librairie NSGraph du Manuel "Référence des Librairies de Services".

Divers

Voici des fonctions, instructions et événements difficiles à classer dans les familles préalablement listées.

- BEEP effectue un ou plusieurs bips sonores.
- SCREENHEIGHT% retourne la hauteur de l'écran physique.
- SCREENWIDTH% retourne la largeur de l'écran physique.

Des événements sont fortement liés à des librairies : PAINT avec la librairie NSGraph, HELP avec la librairie NSHelp, et DDE_* avec la librairie NSComm.

Nouvelle syntaxe d'appel d'une fonction

Avec les nouvelles versions il est possible d'appeler une fonction sans mettre les parenthèses autour des paramètres et sans préfixer l'appel par une variable et le signe d'égalisation.

Syntaxe : [Ret =]FunctionName [(] param1, param2, ...[)]

Exemple

```
;definition
Function Compare%(@I%, J%)

If I% < J%
Return I%
EndIf
I% = J%
Return J%
EndFunction

;Call
Compare% I%, J%.
```

Nouvelle syntaxe d'appel d'une instruction

Avec les nouvelles versions, il est possible d'appeler une instruction en mettant des parenthèses autour des paramètres.
Syntaxe : InstructionName [(] param1, param2, ...[)]
Exemple

```
;definition
Instruction Affect I%, J%, @Res%
If I% < J%
Res% = I%
Else
Res% = J%
EndIf
EndInstruction

;Call
Local Res%
Affect (123, 456, Res%)
```

Librairies

NatStar est fourni avec de nombreuses librairies qui s'emploient directement depuis le langage NCL, sous la forme d'extensions : nouvelles fonctions, nouvelles instructions, nouvelles constantes, ... Citons quelques librairies : NSSQL (Oracle, Sybase, Ingres,...), NS3270 (EHLLAPI), NSGraph (Graphiques).

Liaisons avec les langages C et Pascal

Correspondance des types NCL avec les autres langages

La correspondance des types est la suivante :

NCL	C 32 bits	C 64 bits
INT(1)	char	char
INT(2)	short	short
INT ou INT(4)	int	int
INT(8)	long long	long long
NUM(4)	float	float
NUM ou NUM(8)	double	double
NUM(10)	long double	long double
STRING	char [256] (*)	char [256] (*)
STRING(n)	char [n+1] (*)	char [n+1] (*)
CSTRING	char [256] (**)	char [256] (**)
CSTRING(n)	char [n+1] (**)	char [n+1] (**)
CHAR	char [1]	char [1]
CONTROL	struct (***)	struct (***)
FUNCTION EXTERNAL	<type> __stdcall function	<type> function
INSTRUCTION EXTERNAL	void __stdcall function	void function

(*) STRING = chaîne Pascal. Le premier octet indique la longueur, et le deuxième octet correspond au premier caractère.

(**) CSTRING = chaîne C. Le premier octet correspond au premier caractère, et le dernier caractère est suivi par un octet nul.

(***) voir la structure NS_CONTROL dans le fichier NSLIB.H.

Insertion de code C natif dans du code NCL

NCL permet d'insérer du code spécifique C.

Les variables NCL peuvent être récupérées facilement dans le code natif, avec la règle de conversion suivante :

NCL	C
Var$	szVar
Var%	iVar
Var£	rVar

Si la variable a été déclarée sans suffixe ($, % ou £) par GLOBAL ou LOCAL, son nom est intégralement conservé, sans aucun préfixe (sz, i, ou r).

Par exemple, pour appeler à l'aide du C la fonction MessageBeep de Windows, il est possible de faire :

```
LOCAL TYPE%(2)
MOVE 0 TO TYPE%

; Inclusion de source natif C
%
{
/* Déclaration de la fonction Windows en externe */
extern void __stdcall MessageBeep(WORD wType);

/* Appel de MessageBeep */
MessageBeep(iTYPE);
}
%
; Fin de l'inclusion de source natif C
```

Il est bien évident que ce code natif en ligne ne sera pas exécuté par le module de test, mais sera par contre directement inclus et pris en compte par le module de génération.

Noter aussi l'instruction INCLUDE qui permet d'inclure un fichier source NCL, qui lui même peut inclure un source C de la même façon que celle que nous venons de décrire.

Dans NatStar, le code C natif ne doit pas comporter d'ordre #include. Si ce code a besoin d'inclure un fichier .h, il est nécessaire de créer une fonction ou une instruction EXTERNAL. Pour plus d'informations, reportez-vous à la rubrique FUNCTION EXTERNAL et INSTRUCTION EXTERNAL.

FUNCTION EXTERNAL et INSTRUCTION EXTERNAL

Le code défini au sein de FUNCTION et INSTRUCTION peut être du code NCL comme décrit plus haut, mais peut aussi être du code situé dans une DLL externe en les déclarant avec EXTERNAL.

Ces DLL sont des librairies dynamiques du système d'exploitation. Elles peuvent être écrites en n'importe quel langage (C, Pascal, ...) dont le compilateur peut générer du code compatible.

Les principaux intérêts de cette méthode par rapport à l'insertion directe de code C sont :

- la DLL est toujours réellement exécutée, y compris par le module de test. Ce n'est pas le cas de l'insertion de code natif, le module de test ne sachant pas "interpréter" les langages C, Pascal ou autre.
- la DLL peut être réemployée au sein de différentes applications sans duplication de code

Meilleure modularité du développement

Rappelons que c'est d'ailleurs la méthode utilisée par Nat System pour produire ses librairies standards fournies avec NatStar comme NS3270, NSMisc, NSGraph.
Le développeur qui a choisi cette méthode doit fournir deux fichiers :

- le fichier .DLL
- le fichier .NCL décrivant l'interface de la DLL à l'aide de FUNCTION EXTERNAL et INSTRUCTION EXTERNAL.

Ecriture d'une DLL en Microsoft C

Création du fichier MYDLL.C

Quand une fonction est entière, elle retourne directement le résultat entier. Voir "MySubstract".

Par contre, quand une fonction est de type réel ou chaîne, le run-time l'appelle avec un paramètre supplémentaire qui est un pointeur sur une zone déjà allouée (de la taille correspondant au type retourné) : la fonction doit alors retourner ce même pointeur après avoir stocké le résultat dans la zone pointée.

Ce premier paramètre supplémentaire n'apparaît pas dans la liste des paramètres déclarés côté NCL avec FUNCTION EXTERNAL. Voir "InvertString" et "MyRealSum".

Une instruction ne retourne rien. Elle doit donc être déclarée de type void en C. Voir "MyBeep".

```
/* MYDLL.C */
/*
  This sample shows the differents types of NCL Call:

  mySubstract  : FUNCTION wich return an INT, Substact of two integer arguments
  MyRealSum    : FUNCTION wich return a NUM, Sum of two reals arguments
  InvertString : FUNCTION wich return a CSTRING, invert the argument
  IncFields    : INSTRUCTION wich modify a SEGMENT. The handle of the segment
                 is an argument. Increase the values of the segment fields.
  MyBeep       : INSTRUCTION wich call directly the Windows API

*/

#include <windows.h>
#ifdef WIN64
#  define NS_EXPENTRY
#else
#  ifdef WIN32
#    define NS_EXPENTRY __stdcall
#  endif
#endif

/* NCL Declaration :
  SEGMENT POINT
     INT X
     INT Y
     INT Z
  ENDSEGMENT
  GLOBAL POINT MYPOINT
  INSTRUCTION INCFIELDS POINT @MYPOINT */
```

```
struct point {
  long x;
  long y;
  long z;
};

/*************************************************************/
// NCL Declaration : FUNCTION mySubstract(A%, B%)
long  NS_EXPENTRY mySubstract (long a, long b)
{
  return a - b;
}

/*************************************************************/
// NCL Declaration : FUNCTION MYREALSUM#(A#, B#)
double* NS_EXPENTRY MyRealSum (double *result, double a, double b)
{
  *result = a+b;
  return result;
}

/*************************************************************/
// NCL Declaration : FUNCTION INVERTSTRING$(A$, int l);
char * NS_EXPENTRY InvertString (char *result, char a[256], long l)
{
  int i;
  for (i = 0; i < l; i++)
     *(result+i) = a[l-i-1] ;
  *(result+l) = 0;
  return result;
}

/*************************************************************/
// NCL Declaration : instruction incfields point @mypoint
void NS_EXPENTRY IncFields(struct point *mypoint)
{
  mypoint->x++;
  mypoint->y++;
  mypoint->z++;
}

/*************************************************************/
// NCL Declaration : instruction mybeep n%
void NS_EXPENTRY MyBeep (int n)
{
 int i;
 for (i=0; i<n; i++)
   MessageBeep(0);
}
```

Création du fichier MYDLL.DEF

```
EXPORTS
 mySubstract
 MyRealSum
 InvertString
 IncFields
 MyBeep
```

Compilation de MYDLL.C

En 32 bits

```
CL.EXE /c /W3 /Gs /Zp /DWIN32 /DWINDOWS_IGNORE_PACKING_MISMATCH ..\MYDLL.C
```

remarque : le flag de compilation "WINDOWS_IGNORE_PACKING_MISMATCH" est positionné, car avec certains compilateurs, Le Zp1 n'est pas supporté par défaut si le source C inclut Windows.h

En 64 bits

```
CL.EXE /c /W3 /Gs /DWIN64 ..\MYDLL.C
```

Génération de MYDLL.DLL

en 32 bits

```
LINK.EXE /OUT:MYDLL.DLL /SUBSYSTEM:WINDOWS /DLL /DEF:..\MYDLL.DEF /IMPLIB:MYDLL.LIB
MYDLL.OBJ USER32.LIB
```

en 64 bits

```
LINK.EXE /OUT:MYDLL.DLL /SUBSYSTEM:WINDOWS /MACHINE:X64 /DLL /DEF:..\MYDLL.DEF
/IMPLIB:MYDLL.LIB MYDLL.OBJ USER32.LIB
```

Ne pas oublier de placer MYDLL.DLL dans un des répertoires spécifiés dans le PATH

Interface NCL/ C avec MYDLL.NCL

Pour créer un fichier MYDLL.NCL décrivant l'interface NCL avec MYDLL.DLL, saisir le code NCL suivant :

```
segment point
  int x
  int y
  int z
endSegment ; point

function mySubstract%(a%, b%) external 'MYDLL.mySubstract'
function myRealSum#(a#, b#) external 'MYDLL.MyRealSum'
function invertString$(a$, l%(2)) external 'MYDLL.InvertString'
instruction myBeep n%  external 'MYDLL.MyBeep'
instruction incFields point @mypoint external 'MYDLL.IncFields'
```

Bien respecter la casse des noms de fonctions.

Appel des fonctions et instructions

Maintenant, depuis tout événement de toute application, il est possible d'appeler ces fonctions et instructions.

Par exemple :

```
local i%, n#,s$

i% = MySubstract(3, 2) TO I% ; I% vaut 1
s$ = invertString$("NatSys") TO S$ ; S$ vaut "SyStaN"

n# = myRealSum#(2.3, 4.5) ; N# vaut 5.8


MyBeep (2) ; 2 Bip
```

Voir le sample NS-DK fourni **testDLL**

REFERENCE DES EVENEMENTS

Référence des événements

Ce chapitre de référence contient la totalité des événements que peuvent recevoir les objets, classés par ordre alphabétique.

Les six catégories d'événements

Liste des Evénements

Les six catégories d'événements

Il existe six catégories d'événements associés à un objet :

1. Evénements prédéfinis appelés automatiquement selon la famille du contrôle (par exemple EXECUTED sur un Push-Button).
2. Evénements prédéfinis non appelés automatiquement (par exemple EXECUTED sur un Entry-Field).
3. Evénements de comportement permettant de modifier le comportement standard d'un contrôle, ou de définir le comportement d'un template (par exemple GETVALUE et SETVALUE).
4. Evénements DDE permettant de gérer l'échange dynamique de données entre applications
5. Evénements utilisateurs USER0 à USER15 permettant d'ajouter des événements spécifiques appellables au sein de l'application comme depuis des applications externes.
6. Autres événements utilisateurs, ayant des noms quelconques, différents des noms imposés des cinq catégories précédentes, permettant d'ajouter des événements spécifiques appellables uniquement au sein de l'application.

Nombre d'événements par objet

Ces catégories représentent un potentiel de 90 événements par objet ! Heureusement, il n'y a aucune obligation de les gérer tous... Ce nombre se décompose ainsi :

- 19 pour la somme des événements (1) et (2)
- 34 pour les événements (3)
- 9 pour les événements (4)
- 16 pour les événements (5)
- 16 pour les événements (6)

Les 19 événements (1) et (2) peuvent être classés en trois sous-catégories :

1. Les événements de base que sont les événements clavier, souris et timer :
   a) CHARACTER quand une touche clavier est appuyée ou relâchée
   b) MOUSEMOVE quand la souris est bougée
   c) BUTTONDOWN quand un des boutons de la souris est appuyé
   d) BUTTONUP quand un des boutons de la souris est relâché
   e) BUTTONDBLCK quand un double-clic est effectué sur un de ces boutons
   f) TIMER quand la période de temporisation est écoulée
2. Des événements plus sophistiqués sont générés par le GUI ou le développeur pour réagir à ces événements de base. Par exemple, quand l'utilisateur demande la fermeture de la fenêtre, le GUI enverra un événement TERMINATE :
   a) ACTIVATE quand une fenêtre est activée
   b) INIT quand un objet GUI (fenêtre ou contrôle) est initialisé
   c) TERMINATE quand une fenêtre se ferme
   d) EXECUTED quand un push-button est appuyé, un élément de menu choisi, quand l'utilisateur double clique sur un contrôle
   e) SELECTED quand un élément de menu ou de List box est sélectionné
   f) LOSEFOCUS quand un objet GUI perd le focus
   g) GETFOCUS quand un objet GUI obtient le focus
   h) CHANGED quand la taille ou la position de la fenêtre est changée, quand le texte d'un entry-field est modifié
   i) CHECK quand un push-button de la même fenêtre est appuyé, ou quand un utilisateur demande la fermeture de la fenêtre afin d'accepter ou non la fermeture ou avant un changement de focus
   j) PAINT quand l'aire client de la fenêtre doit être redessinée

k) VSCROLL quand l'utilisateur change la position de l'ascenseur dans une barre de défilement verticale

l) HSCROLL quand l'utilisateur change la position de l'ascenseur dans une barre de défilement horizontale

m) HELP quand l'utilisateur appuie sur la touche [F1] pour demander une aide

3. Le développeur peut augmenter le jeu d'événements en rajoutant ceux prédéfinis mais non appelés automatiquement. Le nombre d'événements (1) dépend de l'objet.

Reportez vous à l'Annexe « Liste des Evénements par Objet » pour leur liste exacte.

Les événements (2) sont les événements restants des 19 événements moins les événements (1).

Appel automatique des événements

Les événements (1) et (3) sont appelés automatiquement lors de toute manipulation de l'objet, par l'utilisateur de l'application ou par programmation NCL.

Les événements (4) sont appelés automatiquement uniquement si l'objet est une fenêtre , c'est-à-dire si ce n'est pas un contrôle, et si une communication DDE est demandée par l'application courante ou une application externe.

Les événements (2) (5) (6), ainsi que les événements (4) ne correspondant pas aux conditions citées, NE SONT JAMAIS APPELES AUTOMATIQUEMENT. Ils doivent donc être appelés par l'instruction SEND ou POST au sein de l'application. Dans le cas (5), ils peuvent de plus être appelés par la fonction WinPostMsg depuis tout code écrit en C, Pascal, ou COBOL.

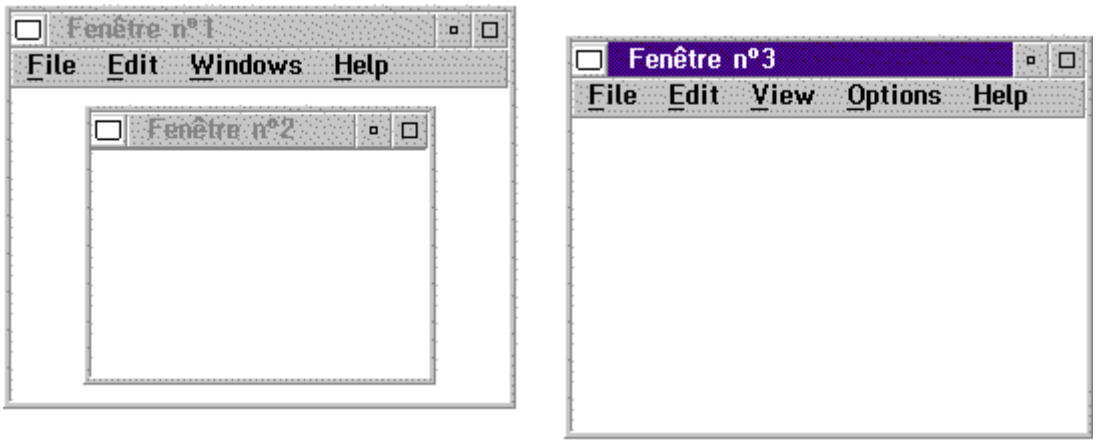
Liste des Evénements

Evénement ACTIVATE

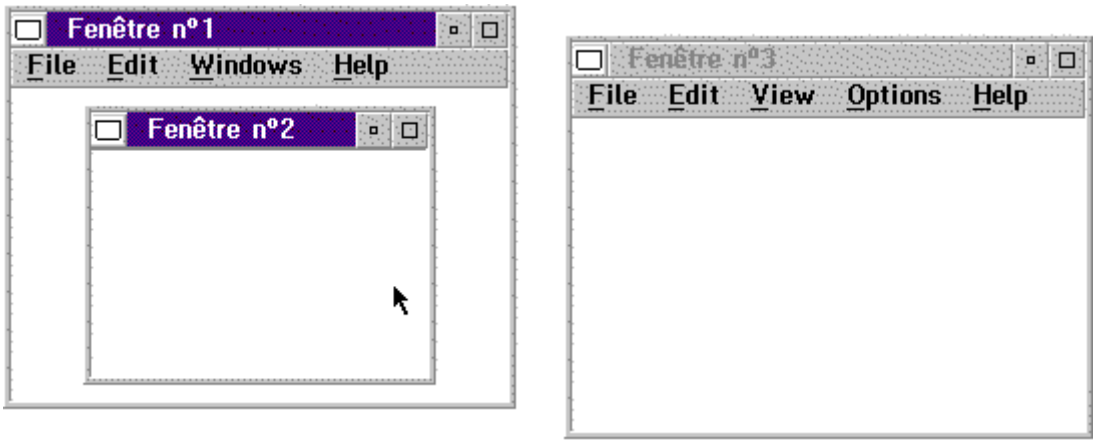
Cet événement est généré lorsque le focus fenêtre change, c'est-à-dire lorsqu'une fenêtre est activée (elle reçoit le focus ou l'une de ses fenêtres filles reçoit le focus), ou désactivée (elle perd le focus).

Paramètres	PARAM1%	Vaut 1 si la fenêtre est activée, 0 si elle est désactivée.
	PARAM2%	Sans signification
	PARAM3%	Sans signification
	PARAM4%	Sans signification

1. Graphiquement, l'activation d'une fenêtre est visible par le changement de couleur de sa barre de titre. Cette couleur est alors celle spécifiée pour le paramètre "Barre de titre active" dans le Panneau de Configuration du système.
2. Illustration de la génération de l'événement ACTIVATE :
   a) Le focus est sur la fenêtre n°3 :

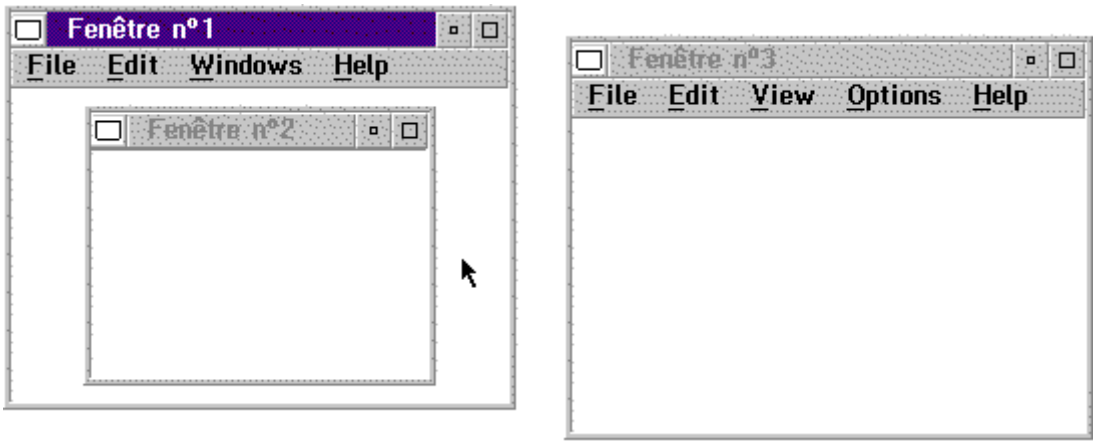


   b) En cliquant sur la fenêtre n°2, trois événements ACTIVATE sont générés :
      i. sur la fenêtre n°1, avec PARAM1% égal à 1,
      ii. sur la fenêtre n°2, avec PARAM1% égal à 1,
      iii. sur la fenêtre n°3, avec PARAM1% égal à 0.



c) En cliquant ensuite sur la fenêtre n°1, un seul événement ACTIVATE est généré :

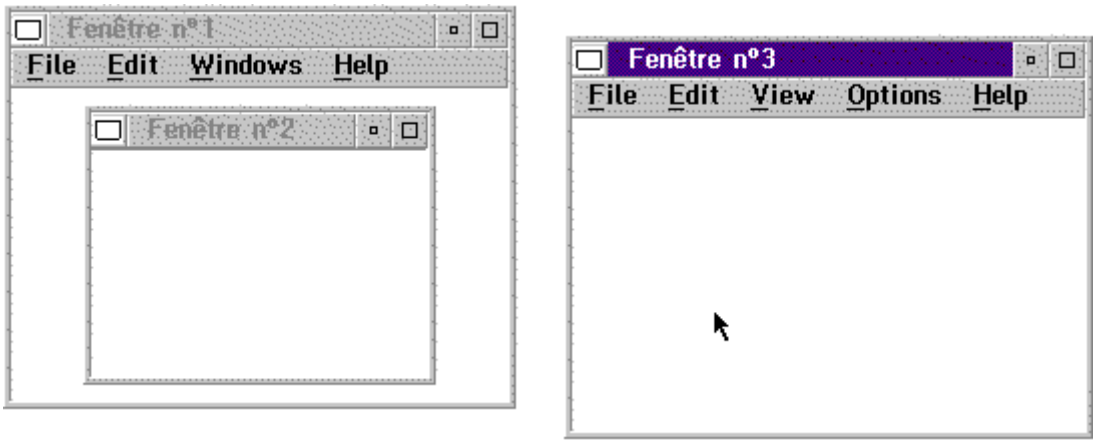
i. sur la fenêtre n°2, avec PARAM1% égal à 0.



d) Enfin, en cliquant sur la fenêtre n°3, deux événements ACTIVATE sont générés :

i. sur la fenêtre n°1, avec PARAM1% égal à 0,

ii. sur la fenêtre n°3, avec PARAM1% égal à 1.



Objets	Window (toutes classes sauf Report)

Evénement ADJUSTSIZEORPOS

Cet événement est généré lorsque l'utilisateur désire modifier la position ou la taille d'une fenêtre. Il permet d'intercepter cette demande et de changer par programme les valeurs des nouvelles positions et tailles.

Paramètres	PARAM1%	Vaut ADJUST_SIZE%, ADJUST_MOVE% ou ADJUST_SIZE% bor ADJUST_MOVE%
	PARAM34%	Pointeur sur un segment de type SEGSIZEORPOS de la librairie NSMISC qui contient les nouvelles positions et dimensions de la fenêtre.

```
;Exemple
Evénement ADJUSTSIZEORPOS de la fenêtre

local int x, int w, int h, int y

x = SEG_SIZEORPOS (param34%).posX
y = SEG_SIZEORPOS (param34%).posY
w = SEG_SIZEORPOS (param34%).Width
h = SEG_SIZEORPOS (param34%).Height

; La fenêtre ne peut pas être déplacée à moins de
; 100 pixels du bord gauche inférieur de l'écran.
if param1% band ADJUST_MOVE%
if x < 100
x = 100
endif
if y < 100
y = 100
endif
endif

; La taille de la fenêtre ne peut pas être inférieure à
; 200 pixels sur 200 pixels.
if param1% band ADJUST_SIZE%
if w < 200
w = 200
endif
if h < 200
h = 200
endif
endif

; mise à jour du segment
SEG_SIZEORPOS (param34%).posX = x
SEG_SIZEORPOS (param34%).posY = y
SEG_SIZEORPOS (param34%).Width = w
SEG_SIZEORPOS (param34%).Height = h
```

Voir aussi ADJUST_*%, SEG_ SIZEORPOS de la librairie NSMISC.

Evénement BUTTONDBLCLK

Cet événement est généré lorsque l'utilisateur double clique sur la souris. Cet événement est envoyé à la fenêtre située sous le pointeur souris.

Paramètres	PARAM1%	Position horizontale du pointeur souris (en pixels par rapport au bord gauche de la fenêtre)
	PARAM2%	Position verticale du pointeur souris (en pixels par rapport au bord bas de la fenêtre)
	PARAM3%	Bouton sur lequel l'action a été faite (MOU_PRIMARY% ou MOU_SECONDARY%)
	PARAM4%	Sans signification

1. Attention : la fenêtre reçoit auparavant BUTTONDOWN et BUTTONUP lors du premier clic. BUTTONDBLCLK remplace le second BUTTONDOWN et un BUTTONUP suit le BUTTONDBLCLK.
2. Le délai de déclenchement de cet événement dépend du paramétrage souris effectué dans le Control Panel
3. Cet événement est également généré lorsque le pointeur souris est situé au-dessus d'un contrôle dans une fenêtre de classe Dialog. Pour être traité, il doit alors être rajouté à la liste des événements de base du contrôle.

Objets	Window (toutes classes sauf Report)

Voir aussi :
Evénements BUTTONDOWN, BUTTONUP, MOUSEMOVE
Instructions CAPTURE, UNCAPTURE
Fonctions ISMOUSE%, NBBUTTONS%
Constantes MOU_*%

Evénement BUTTONDOWN

Cet événement se classe en deux catégories :

1. Il est généré lorsque l'utilisateur appuie sur l'un des boutons de la souris. Cet événement est envoyé à la fenêtre située sous le pointeur souris.
2. Il est généré lorsque l'utilisateur appuie sur le bouton d'apparition de la liste d'une Combo-Box ou d'une CBE.

Paramètres	**Catégorie 1**	
	PARAM1%	Position horizontale du pointeur souris (en pixels par rapport au bord gauche de la fenêtre)
	PARAM2%	Position verticale du pointeur souris (en pixels par rapport au bord inférieur de la fenêtre)
	PARAM3%	Bouton sur lequel l'action a été faite (MOU_PRIMARY% ou MOU_SECONDARY%)
	PARAM4%	Sans signification
	Catégorie 2	
	PARAM1%	Sans signification
	PARAM2%	Sans signification
	PARAM3%	0
	PARAM4%	Sans signification

1. Cet événement n'est pas envoyé en permanence : il est envoyé une seule fois lors du changement d'état (bouton haut -> bouton bas). Il n'est plus envoyé tant que le bouton reste en position basse.
   a) Pour la catégorie 1
      i. Si une fenêtre a demandé de capturer la souris (par l'instruction CAPTURE), l'événement est envoyé à cette fenêtre même si la souris n'est pas au-dessus de son aire client.
      ii. Cet événement est également généré lorsque le pointeur souris est situé au-dessus d'un contrôle dans une fenêtre de classe Dialog. Pour être traité, il doit alors être rajouté à la liste des événements de base du contrôle.
   b) Pour la catégorie 2.

i. Cet événement permet de remplir la liste avant son apparition, afin d'éviter les traitements inutiles si l'utilisateur ne déroule jamais la Combo-Box ou la CBE.

ii. Il peut être précédé d'un autre événement BUTTONDOWN de catégorie 1 lors de l'appui sur le bouton de la souris afin de cliquer sur le bouton de la Combo-Box. Ce n'est cependant pas le comportement par défaut pour ce contrôle. Pour que cela puisse se produire, il faut positionner à False l'option FilterButtonDowns dans le fichier NSLIB.INI.

iii. Il n'existe pas d'événement BUTTONUP associé.

2. Il peut s'avérer nécessaire d'utiliser l'instruction PASS qui permet d'appeler le traitement par défaut de l'événement courant.

Objets	Window (toutes classes sauf Report) Combo-Box CBE-Field

Voir aussi :
Evénements ***BUTTONUP***, ***BUTTONDBLCLK***, ***MOUSEMOVE***
Instructions ***CAPTURE***, ***UNCAPTURE*** ***PASS***
Fonctions ***ISMOUSE%***, ***NBBUTTONS%***
Constantes ***MOU_*%***

Evénement BUTTONUP

Cet événement est généré lorsque l'utilisateur relâche un des boutons de la souris. Cet événement est envoyé à la fenêtre située sous le pointeur souris.

Paramètres	PARAM1%	Position horizontale du pointeur souris (en pixels par rapport au bord gauche de la fenêtre)
	PARAM2%	Position verticale du pointeur souris (en pixels par rapport au bord bas de la fenêtre)
	PARAM3%	Bouton sur lequel l'action a été faite (MOU_PRIMARY% ou MOU_SECONDARY%)
	PARAM4%	Sans signification

1. Cet événement n'est pas envoyé en permanence, tant que le bouton est en position haute : il est envoyé une seule fois lors du changement d'état (bouton bas => bouton haut).
2. Si une fenêtre a demandé à capturer la souris (par l'instruction CAPTURE), l'événement est envoyé à cette fenêtre même si la souris n'est pas au-dessus de son aire client.
3. Cet événement est également généré lorsque le pointeur souris est situé au-dessus d'un contrôle dans une fenêtre de classe Dialog. Pour être traité, il doit alors être rajouté à la liste des événements de base du contrôle.
4. Il peut s'avérer nécessaire d'utiliser l'instruction PASS qui permet d'appeler le traitement par défaut de l'événement courant.

Objets	Window (toutes classes sauf Report)

Voir aussi :
***Evénements** BUTTONUP, BUTTONDBLCLK, MOUSEMOVE*
***Instructions** CAPTURE, UNCAPTURE PASS*
***Fonctions** ISMOUSE%, NBBUTTONS%*
***Constantes** MOU_*%*
Instruction PASS

Evénement CHANGED

Cet événement se classe en deux catégories distinctes :

1. Il est généré lorsque l'utilisateur change la taille de la fenêtre.
2. Il est généré lorsque le champ d'édition d'un Entry-Field, d'un CBE-Field ou d'un MLE-Field est modifié par l'utilisateur.

Paramètres	**Catégorie 1**	
	PARAM1%	Ancienne taille horizontale de la fenêtre (en pixels)
	PARAM2%	Ancienne taille verticale de la fenêtre (en pixels)
	PARAM3%	Nouvelle taille horizontale de la fenêtre (en pixels)
	PARAM4%	Nouvelle taille verticale de la fenêtre (en pixels)
	Catégorie 2	
	PARAM1%	Longueur du texte saisi (sans signification pour une MLE)
	PARAM2%	Sans signification
	PARAM3%	Sans signification
	PARAM4%	Sans signification

1. Pour la catégorie 1

Si « Size Redraw » est coché dans le volet de propriétés de la fenêtre de classe Window, un événement PAINT est automatiquement généré après tout événement CHANGED.

CHANGED est aussi reçu lors d'une minimisation ou maximisation d'une fenêtre.

2. Pour la catégorie 2

CHANGED peut être utilisé pour faire une vérification sur le contenu du champ qui vient d'être changé.

```
;Exemple
;Pour qu'un push-button BUTTON00001 soit invalide (et donc ne
;puisse être enfoncé) lorsque le champ d'édition ENTRY00001 est
;vide, il suffit de faire sur l'événement CHANGED de ce même ;champ  :

IF PARAM1% = 0
; équivalent à IF ENTRY00001 = ""
DISABLE BUTTON00001
ELSE
ENABLE BUTTON00001
ENDIF
```

```
;Et ne pas oublier de faire un DISABLE du bouton sur l'événement
;INIT si le champ est initialement vide (ou cocher "Disable"
;dans le volet de propriétés du bouton).
```

Objets	Window (toutes classes sauf Report) Entry-Field CBE-Field MLE-Field

Voir aussi

Evénement PAINT (commentaire 7)

Instruction CHANGE

Si catégorie 1 :

Instructions SETPOS, MINIMIZE, MAXIMIZE, RESTORE

Fonctions GETHEIGHT%, GETWIDTH%

Fonctions GETCLIENTHEIGHT%, GETCLIENTWIDTH%

Evénement CHARACTER

Cet événement est généré lorsque l'utilisateur a enfoncé ou relâché une touche (ou une combinaison de touches) du clavier. Il est envoyé à la fenêtre qui a le focus.

Paramètres	PARAM1%	Type de la touche enfoncée ou relâchée.
	PARAM2%	Indique le « repeat count » de la touche.
	PARAM3%	Code ASCII, s'il existe, du caractère associé à la touche (uniquement si KF_CHAR% est positionné dans PARAM1%).
	PARAM4%	Code virtuel, s'il existe, de la touche (uniquement si KF_VIRTUALKEY% est positionné dans PARAM1%).

1. PARAM1% contient une combinaison des masques KF_*% suivants :

   a) KF_CHAR%
   signifie que PARAM3% est valide

   b) KF_VIRTUALKEY%
   signifie que PARAM4% est valide

   c) KF_KEYUP%
   signifie qu'il s'agit d'une remontée de touche. Si ce masque n'y est pas, cela signifie qu'il s'agit d'une touche enfoncée.

   d) KF_PREVDOWN%
   signifie que la touche était préalablement en position basse. Si ce masque n'y est pas, cela signifie que la touche était préalablement en position haute. Ce masque est positionné lorsqu'on continue à appuyer sur la même touche. A partir d'une fréquence donnée, l'événement va être envoyé avec ce masque.

   e) KF_LONEKEY%
   signifie que la touche a été enfoncée et relâchée sans que, entre ces deux événements, aucune autre touche n'ait été enfoncée ou relâchée.

   f) KF_SHIFT%
   signifie qu'une touche [Shift] est appuyée simultanément.

   g) KF_ALT%
   signifie qu'une touche [Alt] est appuyée simultanément.

   h) KF_CTRL%
   signifie qu'une touche [Ctrl] est appuyée simultanément.

Les constantes KF_*% sont toutes des puissances de 2, ce qui signifie qu'elles ne contiennent toujours qu'un seul bit positionné à 1. Ainsi si PARAM1% comprend trois 1 lorsqu'il est écrit en binaire, cela signifie que trois KF_*% sont positionnés. Voir les opérateurs binaires BAND, BOR, BNOT, BXOR qui permettent de tester aisément les bits d'un entier.

2. PARAM2% vaut en général 1. Il est supérieur à 1 si l'utilisateur envoie plus vite des caractères que l'application ne peut en traiter : à ce moment-là, le système d'exploitation combine les événements claviers consécutifs de chaque touche en un seul événement CHARACTER.

3. PARAM4% contient une des constantes VK *% suivantes associées aux touches de même nom :

a) VK_LEFT%, VK_RIGHT%, VK_UP%, VK_DOWN%,

b) VK_PAGEUP%, VK_PAGEDOWN%,

c) VK_HOME%, VK_END%,

d) VK_INSERT%, VK_DELETE%,

e) VK_SPACE%, VK_TAB%, VK_BACKSPACE%, VK_BACKTAB%,

f) VK_NEWLINE%, VK_ENTER%,

g) VK_ESCAPE%, VK_BREAK%, VK_PAUSE%,

h) VK_PRINTSCRN%, VK_SYSRQ%,

i) VK_CAPSLOCK%, VK_NUMLOCK%, VK_SCROLLLOCK%,

j) VK_SHIFT%, VK_CTRL%, VK_ALT%,

k) VK_F1%, VK_F2%, VK_F3%, ..., VK_F24%

4. Une application doit vérifier PARAM2% si elle ne veut pas perdre de caractères. Cependant, dans la plupart des cas, ce paramètre est ignoré car il peut rendre complexe le traitement.

5. Un appui normal sur une touche génère deux événements CHARACTER : un premier lorsque la touche est enfoncée, puis un second lorsque la touche est relâchée. Souvent, il est possible de ne gérer que le premier événement, ce qui peut se traduire par coder le début du traitement de CHARACTER comme dans l'exemple ci-dessous.

6. Les masques KF_CHAR% et KF_VIRTUALKEY% ne sont pas incompatibles. Par exemple, pour un appui sur la barre d'espace, le caractère ' ' sera stocké dans PARAM3%, et VK_SPACE% dans PARAM4%. En fonction de la logique de l'application, on pourra utiliser l'un ou l'autre.

7. Cet événement est également généré pour les contrôles d'une fenêtre de classe Dialog. Pour être traité, il doit alors être rajouté à la liste des événements de base du contrôle.

```
;Exemple
IF (PARAM1% BAND KF_KEYUP%) <> 0 ; cf. commentaire 5
EXIT
ENDIF
TOUCHE$ = FILLER$(PARAM3%, PARAM2%)
```

Objets	Window (toutes classes sauf Report) Entry-Field CBE-Field MLE-Field

Voir aussi Evénement HELP (touche [F1])

Evénement CHECK

Cet événement se classe en 3 catégories :

1. Il est généré lorsque l'utilisateur a enfoncé un Push-Button, avant l'événement EXECUTED. L'événement est reçu par tous les contrôles appartenant à la même fenêtre que le push-button. Il n'est pas reçu par la fenêtre elle-même (objet Window) sauf si le push-button demande de fermer la fenêtre (voir catégorie 2).
   Le premier contrôle qui retourne un entier différent de zéro stoppe l'envoi des événements CHECK destinés aux autres contrôles, stoppe la fermeture de la fenêtre (si elle avait été demandée), et positionne le focus sur lui.
2. Il est généré lorsque la fermeture d'une fenêtre est demandée (par exemple par l'instruction CLOSE ou par double clic sur la boîte du menu système), avant l'événement TERMINATE. L'événement est reçu par la fenêtre elle-même (objet Window). Il n'est pas reçu par les contrôles appartenant à la fenêtre. Si la fenêtre retourne un entier différent de zéro, la fenêtre n'est pas fermée.
3. Il est généré à la place d'un événement LOSEFOCUS lorsqu'un objet perd le focus si le « Check Mode » a été activé pour la fenêtre courante (l'instruction SETCHECKMODE de la librairie NSMisc permet d'activer ou de désactiver le « Check Mode »).
   Il n'est cependant pas possible de savoir si la perte de focus provient d'un clic souris, d'un appui sur la touche Tab ou sur Alt+accélérateur clavier

Paramètres	**Catégorie 1**	
	PARAM1%	Précise la cause du CHECK : Vaut CHK_PUSHBUTTON%
	PARAM2%	Sans signification
	PARAM3%	Sans signification
	PARAM4%	Sans signification
	Catégorie 2	
	PARAM1%	Précise la cause du CHECK : Vaut CHK_LOSEFOCUS% ou CHK_ENDSESSION%. Voir dernier alinéa du commentaire 3.
	PARAM2%	Sans signification
	PARAM3%	Sans signification

	PARAM4%	Sans signification
	Catégorie 3	
	PARAM1%	Précise la cause du CHECK : Vaut CHK_LOSEFOCUS%
	PARAM2%	Sans signification
	PARAM3%	Sans signification
	PARAM4%	Sans signification

1. Les trois constantes indiquées pour PARAM1% sont définies dans la librairie NSMisc.
2. Lorsqu'un Push-Button a « Close on execution » coché dans son volet de propriétés, ou effectue un CLOSE dans le script EXECUTED, les 2 premières catégories sont concernées : l'événement CHECK est d'abord reçu par les contrôles, puis par la fenêtre.
3. Pour la catégorie 1 :
   a) Aucun événement CHECK n'est généré lorsque le Push-Button enfoncé a « No checking » coché dans son volet de propriétés.
   b) Il ne faut faire aucune supposition sur l'ordre de réception des événements CHECK par les contrôles : ni Z-Order, ni cheminement du focus. Mais il est bien sûr garanti que tous les contrôles en reçoivent un.
   c) Un RETURN 1 refuse la fermeture de la fenêtre.
   d) Un RETURN 0 génère un nouvel événement CHECK envoyé au contrôle suivant. S'il n'y a pas de contrôle suivant, la fenêtre est fermée.
4. Pour la catégorie 2 :
   a) Un RETURN 1 refuse la fermeture de la fenêtre : l'événement TERMINATE n'est pas généré.
   b) Un RETURN 0 ferme la fenêtre : l'événement TERMINATE est alors généré
   c) Lorsqu'une fenêtre possède plusieurs fenêtres filles, l'ordre de traitement des événements CHECK et TERMINATE est, par défaut, le suivant :
   d) événement CHECK de la fenêtre mère,
   e) si le CHECK de la fenêtre mère retourne 0, événement CHECK de la première fenêtre fille,
   f) si le CHECK de cette fenêtre fille retourne 0, événement CHECK de la fenêtre fille suivante,
   g) idem pour toutes les autres fenêtres filles,
   h) si le CHECK de la dernière fenêtre fille retourne 0, événement TERMINATE de la fenêtre mère,

i) événement TERMINATE de la première fenêtre fille,

j) puis événement TERMINATE de la fenêtre fille suivante,

k) idem pour toutes les autres fenêtres filles.

l) L'ordre d'empilement des fenêtres détermine l'ordre de réception des événements CHECK, la première fenêtre fille à recevoir l'événement CHECK étant celle située en avant plan dans la fenêtre mère, la seconde étant celle situé «en dessous », etc.

m) Si une fenêtre fille retourne 1 dans un événement CHECK, plus aucun CHECK n'est envoyé aux filles suivantes. De ce fait, aucune fenêtre ne reçoit d'événement TERMINATE : le refus de fermeture d'une fenêtre fille bloque la fermeture de toutes les autres fenêtres.

n) Il est néanmoins possible de forcer l'occurrence de l'événement TERMINATE d'une fenêtre fille juste à la suite de l'événement CHECK. Pour cela, il faut forcer la fermeture de la fenêtre en codant un CLOSE en fin de traitement de l'événement CHECK.

o) Ainsi, si le CHECK de l'une des fenêtre filles retourne 0 après la fermeture forcée, l'événement TERMINATE de cette même fenêtre est généré avant d'envoyer un CHECK à la fenêtre fille suivante.

p) Si la valeur retournée est égale à 1, plus aucune fenêtre fille ne reçoit d'événement CHECK, et par conséquent d'événement TERMINATE. Seule la fenêtre mère et les fenêtres filles n'ayant pas reçu de CHECK restent alors ouvertes.

q) Si l'arrêt du système d'exploitation est demandé pendant l'exécution d'une application, celle-ci reçoit l'événement CHECK avec CHK_ENDSESSION% pour valeur de PARAM1%. Il est possible d'interdire l'arrêt de Windows en codant RETURN 1 dans l'événement CHECK.

5. Pour la catégorie 3

   a) Un RETURN 1 empêche le contrôle recevant le CHECK de perdre le focus suite à une action de l'utilisateur visant à donner le focus à un autre objet de la même application.

6. Pour les trois catégories, lorsqu'aucun traitement n'est fait sur l'événement CHECK, cela est équivalent à faire un RETURN 0. Par défaut, la fermeture est donc acceptée.

```
;Exemple Catégorie 1
; Coder ce script sur l'événement CHECK
; d'un Entry-Field est équivalent à cocher
; "Required" dans son volet de propriétés
IF ENTRY00001 = ""
RETURN 1 ; Tant que l'Entry-Field est vide,
; la fermeture de la fenêtre est
; refusée et le focus se positionne
```

```
; automatiquement sur lui
ELSE ; Si quelque chose a été saisi
RETURN 0 ; dans l'Entry-Field, la fermeture
; de la fenêtre est acceptée (envoi
; d'un CHECK au contrôle suivant).
ENDIF

;Exemple Catégorie 2
; Affichage d'une boîte de messages demandant
; la confirmation de fermeture
IF ASK2%("Close", "Voulez-vous réellement fermer l'application ?") = NO%
RETURN 1 ; Si appui sur <No>,
; la fenêtre n'est pas fermée.
ELSE ; Si appui sur <Yes>
RETURN 0 ; la fenêtre est fermée et
; l'événement TERMINATE est généré.
ENDIF

;Exemple Catégorie 3
; Test de la valeur de l'entry field
; Refus de donner le focus pour une valeur
; <100.
; On suppose que SETCHECKMODE TRUE% a été codé
; dans l'événement INIT de la fenêtre
; principale.
IF PARAM1% = CHK_PUSHBUTTON%
IF ENTRY00001 < 100
BEEP
RETURN 1
ENDIF
ENDIF
```

Objets	Tous, sauf Menu-Item : Window (toutes classes sauf Report) Entry-Field Scroll-Bar horizontale Scroll-Bar verticale Radio-Button Check-box List-Box Combo-box CBE-Field MLE-Field Bitmap

Voir aussi

Evénement TERMINATE, Instruction CLOSE

Fonction GETCHECKMODE%, instruction SETCHECKMODE et constantes CHK_*% (librairie NSMisc).

Evénements DDE_*

Ces événements sont à utiliser pour gérer l'échange de données entre applications : cela permet par exemple de communiquer avec Microsoft Excel.

1. DDE signifie Dynamic Data Exchange.
2. Ces événements sont au nombre de neuf :
   a) DDE_ADVISE
   b) DDE_DATA
   c) DDE_EXECUTE
   d) DDE_INIT
   e) DDE_INITACK
   f) DDE_POKE
   g) DDE_REQUEST
   h) DDE_TERMINATE
   i) DDE_UNADVISE

Objets	Window (toutes classes sauf Report)

Voir aussi Description complète de ces événements dans la librairie NSComm

Evénement ENDMENU

Cet événement est reçu par une fenêtre lorsqu'un des menus de sa barre de menus est fermé, soit en appuyant sur la touche Esc soit en cliquant en dehors du menu.

Paramètres	PARAM1%	Sans signification
	PARAM2%	Sans signification
	PARAM3%	Sans signification
	PARAM4%	Sans signification

1. Cet événement ne figure pas dans les événements de base d'une fenêtre, pour être reçu, il doit être ajouté aux événements de cette dernière.
2. Typiquement, cet événement permet de maintenir à jour une aide dans une barre d'état : l'action pouvant être effectuée grâce au menu sélectionné est affichée lors de l'occurrence de l'événement SELECTED du menu, elle est effacée (et, éventuellement, remplacée par l'information sur l'aide elle-même) lors de l'occurrence de ENDMENU.

Objets	Window (toutes classes sauf Report)

Evénement EXECUTED

Cet événement se classe en 5 catégories distinctes :

1. Il est généré lorsque l'utilisateur a choisi un élément de menu.
2. Il est généré lorsque l'utilisateur a choisi un élément de liste (List-Box, Combo-Box, CBE-Field, Window de classe List) par double-clic souris ou appui sur Return.
3. Il est généré lorsque l'utilisateur a enfoncé un push-button.
4. Il est généré lorsque l'utilisateur a enfoncé une bitmap de type « Push-Button » ou double-cliqué sur une bitmap de type « Check-Box ».
5. Il est généré lorsque l'utilisateur a validé un radio-button ou un check-box par double clic souris ou appui sur Return.

Paramètres	**Catégorie 1,3,4 et 5**	
	PARAM1%	Sans signification
	PARAM2%	Sans signification
	PARAM3%	Sans signification
	PARAM4%	Sans signification
	Catégorie 2	
	PARAM1%	Numéro d'ordre de l'élément choisi (0 pour le premier de la liste)
	PARAM2%	Sans signification
	PARAM3%	Sans signification
	PARAM4%	Sans signification

1. Pour les catégories 1, 2 et 5

Un événement SELECTED est préalablement reçu.

2. Pour les catégories 1 et 3

Les champs « Call » et « Close on execution » des volets de propriétés des Push-Buttons et Menu-Items sont invalidés dès que le script de l'événement EXECUTED est traité.

3. Pour la catégorie 3

Des événements CHECK sont générés vers les autres contrôles de la fenêtre si « No checking » n'est pas coché dans le volet de propriétés du Push-Button.

Objets	Window de classe List Menu-Item Push-Button Radio-Button Check-Box List-Box Combo-Box CBE-Field Bitmap style Check-Box Bitmap style Push-Button

Voir aussi Evénement SELECTED

Evénement GETFOCUS

Cet événement est généré lorsqu'un objet reçoit le focus. Le changement de focus se fait à l'aide de la souris (par exemple en cliquant sur une autre fenêtre) ou à l'aide du clavier (par exemple en appuyant sur [Tab] pour aller au contrôle suivant d'une boîte de dialogue).

Paramètres	PARAM1%	Sans signification
	PARAM2%	Sans signification
	PARAM3%	Sans signification
	PARAM4%	Sans signification

1. Il y a toujours un moyen visuel de savoir qui a le focus : il peut se visualiser par un encadré en pointillé (par exemple pour un check-box), par une barre de titre foncée (pour une fenêtre), par un point d'insertion clignotant (pour un entry-field), etc...
2. Si OBJET1 est l'objet qui avait initialement le focus, et OBJET2 est celui qui le reçoit, un LOSEFOCUS est d'abord reçu et traité par OBJET1 avant que GETFOCUS soit reçu et traité par OBJET2.
3. Tout objet qui a reçu un GETFOCUS peut être assuré de recevoir ensuite (bien sûr, délai inconnu) un LOSEFOCUS.
4. Sur un GETFOCUS, il est important de ne faire aucun traitement ayant pour conséquence un nouveau changement de focus.
5. Par exemple, n'employez pas d'instruction SETFOCUS ou d'instruction MESSAGE.
6. En programmant un Custom control, ne codez pas de PASS sur l'événement GETFOCUS. Pour des raisons techniques, ce PASS n'aurait pas d'effet.

Objets	Tous, sauf Menu-Item : Window (toutes classes sauf Report) Entry-field Push-button Scroll-bar horizontale Scroll-bar verticale Radio-button Check-box List-box Combo-box

	CBE-Field MLE-Field Bitmap

Voir aussi Evénement LOSEFOCUS, Instruction SETFOCUS

Evénement HELP

Cet événement est généré lors de l'appui sur la touche [F1] afin d'obtenir de l'aide.

Paramètres	PARAM1%	Sans signification
	PARAM2%	Sans signification
	PARAM3%	Sans signification
	PARAM4%	Sans signification

1. Lorsqu'un HLPINITIALIZE a été préalablement fait (par exemple sur l'événement INIT de la fenêtre), NatStar effectue automatiquement un HLPOPEN de l'item nommé SCREENNAME__CONTROLNAME (nom de la fenêtre et nom du contrôle séparés par deux underscores) si un contrôle a le focus, ou de l'item SCREENNAME si une fenêtre a le focus.

Par exemple, si [F1] est appuyé alors que le focus est situé sur l'entry-field nommé ENTRY de la fenêtre SAMPLE.SCR, l'item SAMPLE__ENTRY sera ouvert. Ce HLPOPEN automatique fonctionne en interprété, avec le module de test. Mais :

Pour que ce HLPOPEN automatique fonctionne aussi en généré, il faut IMPERATIVEMENT utiliser l'option /HELPS avec le module de génération, ou avoir coché l'option « Help info » dans les paramètres de génération.

2. Le fonctionnement indiqué ci-dessus n'est pas effectué lorsqu'aucun HLPINITIALIZE n'a été fait, ou dès que l'événement HELP est traité. Ainsi, même en ayant fait un HLPINITIALIZE, un simple commentaire présent dans l'événement HELP annule le fonctionnement cité précédemment. Avec le même principe, il est possible d'ouvrir un item différent de SCREENNAME__CONTROLNAME en codant un HLPOPEN dans l'événement HELP.
3. En programment un Custom control, ne codez pas de PASS sur l'événement HELP. Pour des raisons techniques, ce PASS n'aurait pas d'effet.

Objets	Tous : Window (toutes classes sauf Report) Menu-Item Entry-field Push-button Scroll-bar horizontale Scroll-bar verticale

	Radio-button Check-box List-box Combo-box CBE-Field MLE-Field Bitmap

Voir aussi Librairie NSHelp (Instructions HLPINITIALIZE, HLPOPEN, HLPTERMINATE)

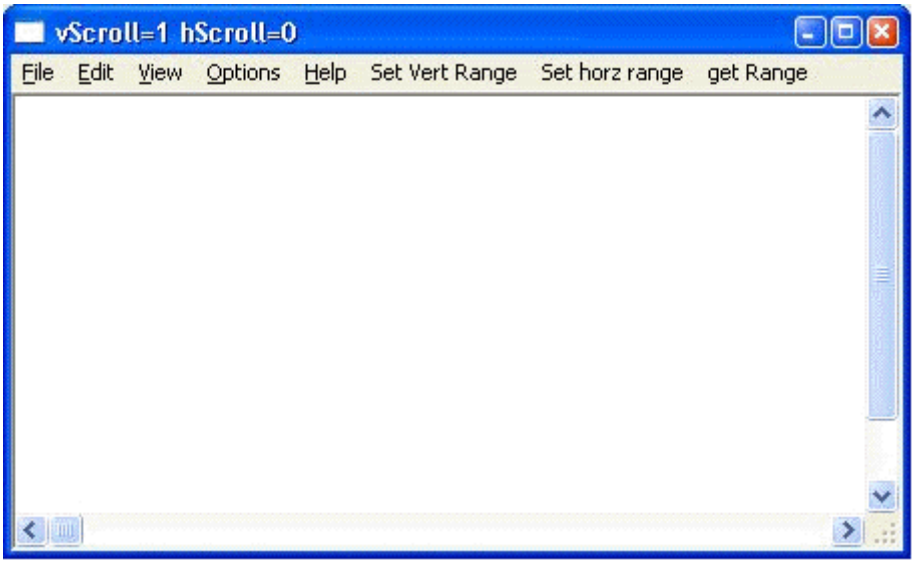
Evénement HSCROLL

Cet événement est généré lorsque l'utilisateur agit sur la barre horizontale de défilement d'une fenêtre (seulement si cette dernière a été déclarée avec le style "Horz Scroll-Bare).

Paramètres	PARAM1%	Position actuelle de l'ascenseur
	PARAM2%	Position minimum de l'ascenseur (nombre de lignes minimum à faire défiler)
	PARAM3%	Position maximum de l'ascenseur (nombre de lignes maximum à faire défiler)
	PARAM4%	Nombre de lignes visibles à l'écran

1. PARAM1% contient une des constantes SB_*% suivantes :
   a) SB_LINELEFT%, SB_LINERIGHT% (appui respectivement sur la flèche gauche ou la flèche droite de la barre de défilement)
   b) SB_PAGELEFT%, SB_PAGERIGHT% (appui entre l'ascenseur et une des flèches, respectivement du côté gauche ou du côté droit)
   c) SB_SLIDERTRACK%, (ascenseur sélectionné et directement déplacé)
   d) SB_SLIDERPOSITION% (ascenseur déplacé puis relâché)
   e) SB_ENDSCROLL% (fin de l'appui sur une des flèches de l'ascenseur)
2. Ne pas confondre une fenêtre "Window avec style Horz Scroll-Bar", et un contrôle "Horizontal Scroll-Bar". Ce dernier contrôle ne reçoit jamais HSCROLL : il reçoit SELECTED
3. Une application bien conçue doit, lors des événements CHARACTER, s'envoyer à elle-même des HSCROLL (via l'instruction SEND) avec le bon PARAM1% de façon à ce que les touches [Left], [Right], [PageUp], [PageDown], et leur combinaison avec [Ctrl], soient équivalentes aux actions souris.
4. La gestion de l'ascenseur et du clavier est automatiquement gérée pour les fenêtres de classe List et Edit.

```
;Exemple
VSCROLL='1,1,6,20'
HSCROLL='0,1,6,20'
```



Objets	Window (toutes classes sauf Report)

Voir aussi Evénements SELECTED, VSCROLL, Instruction SETRANGE

Evénement INIT

Cet événement est généré lors de l'ouverture d'une fenêtre. Il est reçu par la fenêtre puis par tous ses contrôles.

Paramètres	PARAM1%	Sans signification sauf si USING est utilisé avec OPEN*, CALL*, STROPEN*,STRCALL*, dans ce cas elle dépend du développeur.
	PARAM2%	Idem PARAM1%
	PARAM3%	Idem PARAM1%
	PARAM4%	Idem PARAM1%

1. INIT est reçu lors de la création logique de l'objet, et donc AVANT son apparition réelle à la fenêtre.
2. Il ne faut faire aucune supposition sur l'ordre de réception des événements INIT par les contrôles : ni Z-Order, ni cheminement du focus. Mais il est garanti deux choses :
   a) le premier INIT est destiné à la fenêtre
   b) tous les contrôles reçoivent un INIT.
3. L'événement INIT des Menu-Items est particulier :
   a) si le Menu-Item est un « Pulldown Menu », l'événement INIT n'est pas reçu lors de l'ouverture de la fenêtre. Il est reçu lors de chaque déroulement du menu effectué par l'utilisateur.
   b) si le Menu-Item n'est pas un « Pulldown Menu », l'événement INIT n'est JAMAIS RECU !
4. Nat System recommande de ne pas faire de déclarations dans l'événement INIT des fenêtres. La génération fonctionnera correctement, mais le vérificateur de syntaxe retournera des erreurs.

Objets	Tous : Window (toutes classes sauf Report) Menu-Item Entry-field Push-button Scroll-bar horizontale Scroll-bar verticale Radio-button

	Check-box List-box Combo-box CBE-Field MLE-Field Bitmap

Voir aussi Evénement TERMINATE (pour Window), USING

Evénement INITCONTEXT

Cet événement est généré lors du chargement d'un template ou d'un contrôle utilisateur dans la fenêtre qui le contient.

Paramètres	PARAM1%	Sans signification
	PARAM2%	Sans signification
	PARAM1$	Contient la chaîne de contexte du template ou du contrôle utilisateur.
	PARAM3%	Largeur du template s'il est retaillable.
	PARAM4%	Hauteur du template s'il est retaillable.
Objets	Templates Contrôles utilisateurs	

Voir aussi ***TERMINATECONTEXT***

Evénement LOSEFOCUS

Cet événement est généré lorsqu'un objet perd le focus. Le changement de focus se fait à l'aide de la souris (par exemple en cliquant sur une autre fenêtre) ou à l'aide du clavier (par exemple en appuyant sur [Tab] pour aller au contrôle suivant d'une boîte de dialogue).

Paramètres	PARAM1%	Sans signification
	PARAM2%	Sans signification
	PARAM3%	Sans signification
	PARAM4%	Sans signification

1. Il y a toujours un moyen visuel de savoir qui a le focus : il peut se visualiser par un encadré en pointillé (par exemple pour un check-box), par une barre de titre foncée (pour une fenêtre), par un point d'insertion clignotant (pour un entry-field), etc...
2. Si OBJET1 est l'objet qui avait initialement le focus, et OBJET2 est celui qui le reçoit, un LOSEFOCUS est d'abord reçu et traité par OBJET1 avant que GETFOCUS soit reçu et traité par OBJET2.
3. Tout objet qui a reçu un GETFOCUS peut être assuré de recevoir ensuite (bien sûr, délai inconnu) un LOSEFOCUS.
4. Sur un LOSEFOCUS, il est important de ne faire aucun traitement ayant pour conséquence un nouveau changement de focus. Par exemple, n'employez pas d'instruction SETFOCUS ou d'instruction MESSAGE. Par contre, il est possible d'empêcher un contrôle de perdre le focus à l'aide de l'instruction SETCHECKMODE de la librairie NSMisc.
5. En programmant un Custom control, ne codez pas de PASS sur l'événement LOSEFOCUS. Pour des raisons techniques, ce PASS n'aurait pas d'effet.

Objets	Tous, sauf Menu-Item : Window (toutes classes sauf Report) Entry-field Push-button Scroll-bar horizontale Scroll-bar verticale Radio-button Check-box

	List-box Combo-box CBE-Field MLE-Field Bitmap

Voir aussi :
Evénements CHECK (3ème catégorie), GETFOCUS, SETFOCUS
Fonction GETCHECKMODE%, instructions FOCUSCONTROL et SETCHECKMODE (librairie NSMisc).

Evénement MOUSEMOVE

Cet événement est généré lorsque l'utilisateur déplace la souris. Cet événement est envoyé à la fenêtre située sous la nouvelle position du pointeur souris.

Paramètres	PARAM1%	Nouvelle position horizontale du pointeur souris (en pixels par rapport au bord gauche de la fenêtre)
	PARAM2%	Nouvelle position verticale du pointeur souris (en pixels par rapport au bord bas de la fenêtre)
	PARAM3%	Sans signification
	PARAM4%	Sans signification

1. Cet événement arrive généralement en « rafale ». Théoriquement, il est généré à chaque déplacement de un pixel du pointeur souris : ainsi, par exemple, le déplacement du pointeur souris du coin bas gauche de la fenêtre jusqu'au coin haut droit peut générer jusqu'à plusieurs centaines de MOUSEMOVE. Pratiquement, il est généré dès que l'événement précédent a fini d'être traité : cela dépend donc de la vitesse de déplacement de la souris, du temps passé pour le traitement des actions faites sur chaque MOUSEMOVE, ainsi que des actions faites sur tout autre événement qui surviendrait entre deux MOUSEMOVE.

2. Si la souris a été capturée (instruction CAPTURE), TOUS les déplacements seront capturés : la position de la souris peut donc prendre des valeurs en dehors de l'aire client de la fenêtre, et en particulier des valeurs négatives.

3. Pour gérer la souris d'une façon particulière, il est nécessaire de le signaler de manière visuelle en changeant la forme du curseur souris. Cela se fait sur cet événement, grâce à l'instruction SETPTR.

```
SETPTR GETSPTR%(SPTR_ILLEGAL%)
```

4. Cet événement est également généré lorsque le pointeur souris est situé au-dessus d'un contrôle dans une fenêtre de classe Dialog. Pour être traité, il doit alors être rajouté à la liste des événements de base du contrôle.

Objets	Window (toutes classes sauf Report)

Voir aussi :
Evénements ***BUTTONDBLCLK***, ***BUTTONDOWN***, ***BUTTONUP***
Instructions ***CAPTURE***, ***UNCAPTURE***, ***SETPTR***
Fonctions ***ISMOUSE%***, ***NBBUTTONS%***, ***GETSPTR%***
Constantes ***SPTR_*%***

Evénement PAINT

Cet événement est généré dès que le système d'exploitation estime que le contenu (tout ou partie) de la fenêtre doit être redessiné.

Paramètres	PARAM1%	Poids-faible du handle Presentation Space, qui sert ensuite à dessiner dans la fenêtre (ce handle est demandé par les fonctions et instructions de la librairie NSGraph).
	PARAM2%	Poids-fort du handle Presentation Space.
	PARAM3%	Sans signification
	PARAM4%	Sans signification

1. Cet événement est généré pour toutes les fenêtres, y compris les fenêtres de classe Dialog. Mais dans ce cas, il est peu intéressant de le gérer, car NatStar sait automatiquement redessiner la totalité des contrôles composant la fenêtre.
2. Le système d'exploitation ne mémorise pas les dessins faits par les fenêtres de classe Window. Ainsi, soit une fenêtre WINDOW1 située en avant-plan : la totalité de la fenêtre est donc visible. Soit une autre fenêtre WINDOW2 qui vient ensuite chevaucher partiellement WINDOW1 : une partie de WINDOW1 devient donc invisible. Maintenant le seul fait de fermer WINDOW2 découvrira des zones de WINDOW1 qui avaient été cachées et qui ne sont plus valides : le système d'exploitation n'ayant pas mémorisé le contenu de WINDOW1 demandera à cette dernière de se redessiner et le signalera précisément grâce à l'événement PAINT.
3. Le commentaire précédent implique un point important dans le développement d'applications avec NatStar : une fenêtre de classe Window doit toujours être capable de se redessiner à n'importe quel moment.
4. Chaque contrôle élémentaire (Push-button, list-box, etc...) a aussi ce problème : en fait, NatStar possède en interne des routines qui permettent de les redessiner. Le redessin des contrôles contenus dans une fenêtre ou une boîte de dialogue n'a donc pas à être fait par l'application, NatStar s'en charge automatiquement.
5. Lors de la création d'une fenêtre, deux événements sont toujours envoyés dans le même ordre : INIT (fenêtre pas encore visible), puis PAINT (fenêtre visible avec sa frame, mais contenu de l'aire client encore vierge).

6. Par défaut, lorsqu'aucun script n'est présent, le fond de la fenêtre est peint en COL_BACKGROUND% (paramétrable dans le Control Panel, généralement équivalent à la couleur blanche). Le seul fait de mettre un « ; » dans le script annule ce fonctionnement par défaut, et le fond de la fenêtre n'est plus peint (donne l'impression de « transparence » à la fenêtre).
7. Si « Size Redraw » est coché dans le volet de propriétés de la fenêtre, et si « Size » est sélectionné dans le champ « Border » du même volet de propriétés, un événement PAINT est automatiquement généré à chaque changement de taille de la fenêtre, après l'événement CHANGED.
8. Si « Save Bits » est coché dans le volet de propriétés de la fenêtre, les images affichées dans les fenêtres cachées par cette fenêtre seront sauvegardées automatiquement : ces autres fenêtres ne recevront donc pas de PAINT lors du déplacement de la fenêtre « Save Bits » qui les cachait. Ce comportement peut toutefois changer selon le système d'exploitation utilisé : il est recommandé de développer sans faire de supposition sur cette sauvegarde automatique.
9. Comme le handle Presentation Space est passé « éclaté » en deux parties, mot de poids-faible dans PARAM1% et mot de poids-fort dans PARAM2%, utilisez la fonction PARAM12% qui permet de les combiner aisément.

```
MOVE DWORD% (PARAM2%, PARAM1%) TO PS%
; ou bien
MOVE PARAM12% TO PS%
```

Objets	Window (toutes classes sauf Report)

Voir aussi :
Instruction INVALIDATE
Fonctions GETHEIGHT%, GETWIDTH%
Fonctions GETCLIENTHEIGHT%, GETCLIENTWIDTH%
Librairie NSGraph
Fonctions PARAM*%, PARAM*$

Evénement SELECTED

Cet événement se classe en 5 catégories :

1. Il est généré lorsqu'un élément de menu est en cours de sélection
2. Il est généré lorsque l'utilisateur a sélectionné un élément de liste (List-Box, Combo-box, CBE-Field, Window de classe List) ou une fenêtre à onglets du contrôle classeur par clic souris ou appui sur les flèches de direction
3. Il est généré lorsque l'utilisateur déplace le curseur clignotant au sein d'un MLE-Field ou Window de classe Edit.
4. Il est généré lorsque l'utilisateur sélectionne un bouton (radio-button, check-box, ou bitmap de style check-box)
5. Il est généré lorsque l'utilisateur modifie la position d'un ascenseur (contrôles scroll-bar horiz, scroll-bar vert)

Paramètres	**Catégorie 1 et 4**	
	PARAM1%	Sans signification
	PARAM2%	Sans signification
	PARAM3%	Sans signification
	PARAM4%	Sans signification
	Catégorie 2	
	PARAM1%	Numéro d'ordre de l'élément choisi (0 pour le premier de la liste ou pour la première fenêtre à onglets du contrôle Classeur)
	PARAM2%	Sans signification
	PARAM3%	Sans signification
	PARAM4%	Sans signification
	Catégorie 3	
	PARAM1%	Sans signification
	PARAM2%	Sans signification
	PARAM3%	Sans signification
	PARAM4%	Sans signification
	Catégorie 5	

	PARAM1%	Nouvelle position de l'ascenseur (position comprise entre le minimum et le maximum).
	PARAM2%	Sans signification
	PARAM3%	Valeur minimum de l'ascenseur.
	PARAM4%	Valeur maximum de l'ascenseur.

1. Les contrôles concernés (sauf les Scroll-bars) reçoivent ensuite un événement EXECUTED s'ils sont réellement choisis et validés.

2. Ne pas confondre une fenêtre « Window avec style Horz Scroll-Bar ou Vert Scroll-Bar », et un contrôle « Horizontal ou Vertical Scroll-Bar ». Les fenêtres (sauf de classe List) ne reçoivent jamais SELECTED : elles reçoivent HSCROLL et VSCROLL.

3. Pour la catégorie 3

4. La valeur initiale, ainsi que le minimum et le maximum, peuvent être définis statiquement dans le volet de propriétés de l'ascenseur, ou redéfinis dynamiquement par l'instruction SETRANGE.

Objets	Window classe List et Edit Menu-Item Scroll-bar horizontale Scroll-bar verticale Radio-button Check-box List-box Combo-box CBE-Field MLE-Field Bitmap style Check-box

Voir aussi :

Evénements EXECUTED, HSCROLL, VSCROLL

Instructions SELECT, UNSELECT, SETRANGE

Fonctions ISSELECTED%, SELECTION%

Evénement TERMINATE

Cet événement est généré lorsque la fermeture de la fenêtre vient d'être effectuée par l'instruction CLOSE, par double clic sur la boîte du menu système, par sélection de « Close » du menu système, ou par l'appui sur [Alt]+[F4].

Paramètres	PARAM1%	Sans signification
	PARAM2%	Sans signification
	PARAM3%	Sans signification
	PARAM4%	Sans signification

1. Contrairement à TERMINATE, l'événement CHECK arrive AVANT la fermeture réelle de la fenêtre. Ainsi, CHECK sert souvent à demander une confirmation de fermeture à l'utilisateur par l'intermédiaire d'une boîte de messages : TERMINATE n'est pas généré si CHECK retourne une valeur différente de zéro.
2. Le fait de fermer une fenêtre n'implique pas l'arrêt de l'application.
3. Les templates contenus dans une boîte de dialogue reçoivent l'événement TERMINATE, avant exécution de l'événement TERMINATE de la boîte de dialogue.
4. Les fenêtres filles reçoivent l'événement TERMINATE lors de la fermeture de la fenêtre mère.

Objets	Window (toutes classes sauf Report)

V***oir aussi Evénements INIT, CHECK, Instruction CLOSE***

Evénement TERMINATECONTEXT

Cet événement est généré lors du déchargement d'un template ou d'un contrôle utilisateur dans la fenêtre qui le contient.

Paramètres	PARAM1%	Sans signification
	PARAM2%	Sans signification
	PARAM3%	Sans signification
	PARAM4%	Sans signification
Objets	Templates Contrôles utilisateurs	

Voir aussi INITCONTEXT

Evénement TIMER

Cet événement est généré uniquement si l'instruction STARTTIMER a été appelée : le système d'exploitation envoie alors régulièrement cet événement à la fenêtre qui a démarré le timer.

Paramètres	PARAM1%	Numéro du timer tel qu'il avait été passé à STARTTIMER.
	PARAM2%	Sans signification
	PARAM3%	Sans signification
	PARAM4%	Sans signification

1. Cet événement est généré régulièrement toutes les n milli-secondes (n'ayant été défini au démarrage du timer) jusqu'à ce qu'il y ait un STOPTIMER demandé (ou bien sûr si la fenêtre n'existe plus).
2. Si plusieurs timers ont été démarrés par la même fenêtre, ce PARAM1% sert à les distinguer entre eux : le même événement est généré quelque soit le timer, seul PARAM1% diffère.
3. Comme tous les autres événements, le délai d'arrivée de cet événement n'est pas garanti. Il dépend essentiellement du temps de traitement de l'événement en cours de traitement lorsque le timer arrive à échéance : TIMER n'arrivera que lorsque le ou les événements précédents auront été traités.
4. Lors de l'utilisation d'un événement DDE, ne pas utiliser le TIMER 1, car ce dernier est utilisé par le protocole DDE.

Objets	Window (toutes classes sauf Report)

Voir aussi Instructions STARTTIMER, STOPTIMER

Evénement VSCROLL

Cet événement est généré lorsque l'utilisateur agit sur la barre verticale de défilement d'une fenêtre (seulement si cette dernière a été déclarée avec le style "Vert Scroll-Bar").

Paramètres	PARAM1%	Position actuelle de l'ascenseur
	PARAM2%	Position minimum de l'ascenseur (nombre de lignes minimum à faire défiler)
	PARAM3%	Position maximum de l'ascenseur (nombre de lignes maximum à faire défiler)
	PARAM4%	Nombre de lignes visibles à l'écran

1. PARAM1% contient une des constantes SB_*% suivantes :
   a) SB_LINEUP%, SB_LINEDOWN% (appui respectivement sur la flèche haute ou la flèche basse de la barre de défilement)
   b) SB_PAGEUP%, SB_PAGEDOWN% (appui entre l'ascenseur et l'une des flèches, respectivement côté haut ou côté bas)
   c) SB_SLIDERTRACK%, (ascenseur sélectionné et directement déplacé)
   d) SB_SLIDERPOSITION% (ascenseur déplacé puis relâché)
   e) SB_ENDSCROLL% (fin de l'appui sur une des flèches de l'ascenseur)
2. Ne pas confondre une fenêtre "Window avec style Vert Scroll-Bar", et un contrôle "Vertical Scroll-Bar". Ce dernier contrôle ne reçoit jamais VSCROLL : il reçoit SELECTED.
3. Une application bien conçue doit, lors des événements CHARACTER, s'envoyer à elle-même des VSCROLL (via l'instruction SEND) avec le bon PARAM1% de façon à ce que les touches [Up], [Down], [PageUp], [PageDown], et leur combinaison avec [Ctrl], soient équivalentes aux actions souris.
4. La gestion de l'ascenseur et du clavier est automatiquement gérée pour les fenêtres de classe List et Edit.

```
;Exemple
VSCROLL='1,1,6,20'
HSCROLL='0,1,6,20'
```

Objets	Window (toutes classes sauf Report)

Voir aussi Evénements HSCROLL, SELECTED, Instruction SETRANGE

Evénements Utilisateur

La signification de ces événements dépend du développeur. Ils peuvent être appelés grâce à l'instruction SEND de NCL.

Paramètres	PARAM1%	Signification selon le développeur, passé par SEND.
	PARAM2%	Signification selon le développeur, passé par SEND.
	PARAM3%	Signification selon le développeur, passé par SEND.
	PARAM4%	Signification selon le développeur, passé par SEND.

1. Ces événements peuvent avoir n'importe quel nom, sauf les noms d'événements prédéfinis comme INIT, CHECK, DDE *, GETVALUE,...
2. Il n'y a de règles strictes sur les noms de ces événements que lorsqu'ils doivent être appelés depuis une autre application graphique externe écrite en C ou avec NatStar. Pour cela, il faut :
   a) du côté de l'application NatStar réceptrice, nommer ces événements USER0, USER1, ..., USER15
   b) du côté de l'application expéditrice, si elle est écrite en C, envoyer ces événements par la fonction système PostMessage sous Windows.
   c) du côté de l'application expéditrice, si elle est écrite avec NatStar, envoyer ces événements par l'instruction POSTMESSAGE de la librairie NSWin avec WM_USER%, WM_USER%+1, ..., WM_USER%+15 (WM_USER% étant une constante prédéfinie de la librairie NSWin sous Windows).
3. Le paramètre « mp1 » de PostMessage ou WinPostMsg (= PARM1% de POSTMESSAGE du NCL) est reçu dans PARAM12%, et « mp2 » (= PARM2%) dans PARAM34%.
4. La fonction GETCLIENTHWND% de la librairie NSWin permet d'obtenir le handle fenêtre Windows, utilisable par PostMessage ou POSTMESSAGE (du NCL), à partir d'un handle fenêtre NatStar.
5. Il peut y avoir au maximum sur le même objet :
   a) 16 événements utilisateur USER0 à USER15
   b) 16 événements utilisateur ayant un nom quelconque
6. Un événement de la catégorie (2), vue en introduction de ce chapitre, ajouté à un contrôle peut également être considéré comme un « événement utilisateur » car il ne peut être généré que par un SEND. Ce serait par exemple le cas d'un événement EXECUTED ajouté sur un Entry-Field.
7. Avec cette extension, il peut donc y avoir plus de 32 événements utilisateur.

8. La réception d'un événement USER0 fait sortir immédiatement d'une instruction WAIT, sans attendre la fin du délai.

Objets	Tous les objets peuvent en avoir

Voir aussi :

Instruction SEND

Instruction POSTMESSAGE (Librairie NSWin sous Windows)

Constante WM_USER% (Librairie NSWin sous Windows)

Evénements de Comportement

Ces événements sont directement appelés par les fonctions et instructions NCL de manipulation d'objets, et permettent donc de définir le comportement d'un template ou de modifier le comportement standard d'un objet.

Paramètres	PARAM1%	Signification selon l'événement.
	PARAM2%	Signification selon l'événement.
	PARAM3%	Signification selon l'événement.
	PARAM4%	Signification selon l'événement.

1. Il existe 31 événements de comportement :

DELETE	HIDE	SELECTION
DELETEALL	INSERT	SETBACKCOL
DISABLE	ISDISABLED	SETFOCUS
GETBACKCOL	ISHIDDEN	SETFORECOL
GETFORECOL	ISLOCKED	SETPOS
GETHEIGHT	ISSELECTED	SETRANGE
GETTEXT	LINECOUNT	SETTEXT
GETVALUE	LOAD	SETVALUE
GETWIDTH	LOCK	UPDATE
GETXPOS	SAVE	
GETYPOS	SELECT	

2. Le tableau suivant détaille chacun de ces événements :

   a) Les colonnes « PARAM1% », « PARAM2% » et « PARAM2$ » précisent les paramètres passés avec chacun de ces événements. Rappelons que PARAM2$ permet de manipuler la chaîne pointée par DWORD%(PARAM4%, PARAM3%).

   b) La colonne « Appelé par NCL » indique la fonction ou instruction NCL qui déclenche l'événement avec ses paramètres.

   c) La colonne « Retour » indique quelle valeur doit retourner l'événement, uniquement dans le cas où le déclencheur de l'événement est une fonction.

Evénement	PARAM1%	PARAM2%	PARAM2$	Appelé par NCL (d)	Retour
DELETE	/	position	/	DELETE AT position FROM objet	/
DELETEALL	/	/	/	DELETE FROM objet	/
DISABLE	TRUE%	/	/	DISABLE objet	/
	FALSE%	/	/	ENABLE objet	/
GETBACKCOL	/	/	/	GETBACKCOL% (objet)	RETURN COL *%
GETFORECOL	RETURN COL *%	/	/	/	/
GETHEIGHT	/	/	/	GETHEIGHT% (objet)	RETURN hauteur
GETTEXT	/	/	/	GETTEXT$ (objet)	MOVE texte TO PARAM2$
GETVALUE	DEFRET%	0	/	(b) MOVE objet TO xx	MOVE valeur TO PARAM2$
	index (a)	0	/	(b) MOVE objet[index] TO xx	MOVE valeur TO PARAM2$
	(e)	(e)	/	(b) MOVE objet.paramdyn TO xx	MOVE valeur TO PARAM2$
GETWIDTH	/	/	/	GETWIDTH% (objet)	RETURN largeur
GETXPOS	/	/	/	GETXPOS% (objet)	RETURN posx
GETYPOS	/	/	/	GETYPOS% (objet)	RETURN posy
HIDE	TRUE%	/	/	HIDE objet	/

	FALSE%	/	/	SHOW objet	/
INSERT	0	position	chaîne	INSERT AT position chaîne TO objet	/
	1	/	chaîne	INSERT AT END chaîne TO objet	/
	2	/	chaîne	INSERT ASCENDING chaîne TO objet	/
	3	/	chaîne	INSERT DESCENDING chaîne TO objet	/
ISDISABLED	/	/	/	ISDISABLED% (objet)	RETURN TRUE% | FALSE%
ISHIDDEN	/	/	/	ISHIDDEN% (objet)	RETURN TRUE% | FALSE%
ISLOCKED	/	/	/	ISLOCKED% (objet)	RETURN TRUE% | FALSE%
ISSELECTED	position	/	position	ISSELECTED% (objet, position)	RETURN TRUE% | FALSE%
LINECOUNT	/	/	/	LINECOUNT% (objet)	RETURN nblignes
LOAD	1	/	nomfich	LOAD nomfich TO objet	/
	2	/	nomfich	LOAD ASCENDING nomfich TO objet	/
	3	/	nomfich	LOAD DESCENDING nomfich TO objet	/
LOCK	TRUE%	/	/	LOCK objet	/
	FALSE%	/	/	UNLOCK objet	/

SAVE	/	/	nomfich	SAVE objet TO nomfich	/
SELECT	TRUE%	position	/	SELECT position FROM objet	/
	FALSE%	position	/	UNSELECT position FROM objet	/
SELECTION	/	/	/	SELECTION% (objet)	RETURN sélection
SETBACKCOL	couleur	/	/	SETBACKCOL couleur TO objet	/
SETFOCUS	/	/	/	SETFOCUS objet	/
SETFORECOL	couleur	/	/	SETFORECOL couleur TO objet	/
SETPOS	posx	posy	/	SETPOS posx, posy TO objet	/
SETRANGE	/	position	(c)	SETRANGE position, min, max TO objet	/
SETTEXT	/	/	texte	SETTEXT texte TO objet	/
SETVALUE	DEFRET%	0	valeur	MOVE valeur TO objet	/
	index (a)	0	valeur	MOVE valeur TO objet[index]	/
	(e)	(e)	valeur	MOVE valeur TO objet.paramdyn	/
UPDATE	TRUE%	/	/	UPDATE objet	/
	FALSE%	/	/	NOUPDATE objet	/

(a) L'index est utilisable dans le cas d'un objet liste (List-Box, Combo-Box, CBE, MLE, fenêtre de classe List) et dans le cas d'un template.

(b) MOVE ou toute instruction demandant la valeur de l'objet (IF, EVALUATE, ...).

(c) SETRANGE passe le mini dans PARAM3% et le maxi dans PARAM4%.

(d) Certains paramétrages dynamiques génèrent aussi des événements de comportement. Se référer à la note 6, ainsi qu'au tableau des paramétrages dynamiques dans le Chapitre « Référence du Langage ».

(e) Le paramétrage dynamique « paramdyn » est passé dans PARAM1$ (dans ce cas, PARAM2% est différent de zéro)

Rappel : TRUE% vaut 1, FALSE% vaut 0, DEFRET% vaut 32767.

3. Les événements ont généralement le même nom que la fonction ou instruction NCL. Les seules exceptions sont :

Fonction / Instruction	Evénement de comportement
DELETE FROM	DELETEALL
ENABLE	DISABLE avec FALSE% dans PARAM1%
MOVE	GETVALUE ou SETVALUE
SHOW	HIDE avec FALSE% dans PARAM1%
UNLOCK	LOCK avec FALSE% dans PARAM1%
UNSELECT	SELECT avec FALSE% dans PARAM1%
NOUPDATE	UPDATE avec FALSE% dans PARAM1%

4. Comme tous les événements standards, ne pas clôturer le script par RETURN est équivalent à faire un RETURN 0.

5. RETURN DEFRET% employé au sein d'un script d'un événement de comportement effectue un PASS implicite. La différence est que PASS permet de continuer le script alors que RETURN finit son traitement.

a) Dans le cas d'un contrôle quelconque, l'instruction PASS (ou RETURN DEFRET%) permet d'appeler le traitement par défaut interne à NatStar.

b) Dans le cas d'un contrôle template, l'instruction PASS (ou RETURN DEFRET%) permet d'appeler le script NCL de ce même événement dans le fichier template .TPL.

c) Comme indiqué dans le tableau, les paramétrages dynamiques génèrent des événements de comportement GETVALUE et SETVALUE. De plus, pour certains paramétrages dynamiques, le fonctionnement par défaut de GETVALUE et SETVALUE est de générer d'autres événements de comportement. Il s'agit de :

Paramètre Dynamique	Evénement de comportement
.BACKCOULEUR	GETFORECOL ou SETFORECOL
.DISABLED	ISDISABLED ou DISABLE
.FORECOULEUR	GETFORECOL ou SETFORECOL
.HEIGHT	GETHEIGHT
.HIDDEN	ISHIDDEN ou HIDE

.LOCKED	ISLOCKED ou LOCK
.TEXT	GETTEXT ou SETTEXT
.UPDATED	UPDATE
.VALEUR	GETVALUE ou SETVALUE
.WIDTH	GETWIDTH
.X	GETXPOS ou SETPOS
.Y	GETYPOS ou SETPOS

```
;Exemple 1
;Soit un contrôle CONTROLNAME. Voici comment intercepter et
;gérer tous les MOVE.

; Evénement SETVALUE de CONTROLNAME
MOVE PARAM2$ TO MYVAL$

; Evénement GETVALUE de CONTROLNAME
MOVE "NatSys" && MYVAL$ TO PARAM2$

; Tout événement de tout objet
MOVE "Bonjour" TO CONTROLNAME
; génère un SETVALUE
MOVE CONTROLNAME TO S$
; génère un GETVALUE
; et S$ vaut "NatSys Bonjour"

;Exemple 2
;Soit un contrôle CONTROLNAME. Voici comment intercepter tous
;les ENABLE et DISABLE.

; Evénement DISABLE de CONTROLNAME
IF PARAM1% = TRUE%
MESSAGE "Disable", "Début de traitement"
ELSE
MESSAGE "Enable", "Début de traitement"
ENDIF
RETURN DEFRET% ; Appel du traitement par
; défaut et fin du traitement

;Exemple 3
;Même exemple que le précédent, mais en employant PASS afin de
;pouvoir reprendre le traitement spécifique après le traitement
;par défaut.

; Evénement DISABLE de CONTROLNAME
IF PARAM1% = TRUE%
MESSAGE "Disable", "Début de traitement"
ELSE
MESSAGE "Enable", "Début de traitement"
ENDIF
PASS PARAM1%, PARAM2%, PARAM3%, PARAM4%
; Appel du traitement par défaut
; et reprise du traitement spécifique
MESSAGE "Disable/Enable", "Fin de traitement"Objets
```

Objets	Tous les objets peuvent les traiter

Voir aussi :
Instructions PASS, RETURN
Toutes fonctions et instructions citées dans le tableau.
Paramétrages dynamiques

REFERENCE DU LANGAGE

Introduction au NCL

Devant chacun des noms de mot-clé, acceptés par NCL, figure sa famille. Voici les sept familles existantes de NCL, avec quelques exemples de mots-clés pour chacun :

- Paramétrage dynamique (.DISABLED, .FONTNAME, .X)
- Opérateur (AND, INT, LENGTH)
- Constante prédéfinie (TRUE%, CHECKED%, VK_SPACE%)
- Séparateur (AT, FROM, TO)
- Fonction (ASK2%, ISDISABLED%, PARAM1%)
- Instruction (MAXIMIZE, IF, MOVE)
- Type (CHAR, CONTROL, CSTRING)

> Il est à noter que les opérateurs qui s'écrivent avec des caractères non alphabétiques (<, =, >, &, &&, ...) ne sont pas listés. La description de ces opérateurs figure dans le chapitre "Eléments du Langage".

La liste complète des mots-clés est donnée en annexe (Quick Reference).

Différences essentielles entre fonctions et instructions

- Une fonction retourne une valeur, alors qu'une instruction ne retourne rien.
- Lorsqu'il y a des paramètres, ils doivent syntaxiquement être spécifiés entre parenthèses dans le cas d'une fonction, sans parenthèse dans le cas d'une instruction.
- Le nom d'une fonction se termine par un caractère spécial indiquant le type de la valeur retournée : % pour entier, £ ou # pour nombre réel, $ pour chaîne. Le nom d'une instruction ne se termine par aucun caractère spécial.

Typiquement, voici la façon de les utiliser :

```
; Utilisation de fonctions
MOVE ASK2% ("Titre", "Message") TO RET%
; Appel de MAFONCTION%
MOVE MAFONCTION%(PAR1%, PAR2%) TO RET%

; Utilisation d'instructions
MESSAGE "Titre", "Message"
MONINSTRUCTION PAR1%, PAR2% ; appel de MONINSTRUCTION
Comme le Pascal, mais contrairement au langage C, il est interdit d'ignorer le retour
d'une fonction  :
ASK2%("Titre", "Message") ; Syntax error !
MAFONCTION%(PAR1%, PAR2%) ; Syntax error !
```

Dans quelques cas, une variable passée en paramètre à une fonction ou instruction peut être modifiée par l'appel, comme par exemple avec le dernier paramètre (optionnel) des instructions CALL et OPEN. Cela correspond aussi au "@" des paramètres de FUNCTION et INSTRUCTION.

Caractères spéciaux pour les syntaxes

Dans les syntaxes, les crochets [ ] encadrent des champs optionnels, et le caractère | signifie "ou".
Ainsi, par exemple :

```
LOAD [ASCENDING|DESCENDING] nom-fichier TO objet
```

indique que les syntaxes acceptées sont :

```
LOAD nom-fichier TO objet
LOAD ASCENDING nom-fichier TO objet
LOAD DESCENDING nom-fichier TO objet
```

Champs contrôle, fenêtre, objet

Entier, Réel, Chaîne

A chaque fois que entier est spécifié dans une syntaxe, cela signifie une expression de type entier, comme par exemple 2 ou I% ou encore (I%+2)*3. De plus, lorsque cela s'avère nécessaire, une conversion automatique est faite pour rendre entier le résultat de l'expression.

Ainsi si S$ vaut "12", BEEP S$ fera 12 bips.

Le même raisonnement s'applique pour ce qui est réel ou chaîne.

Ainsi l'opérateur STRING dans l'instruction suivante est superflu, la conversion en chaîne étant automatique :

```
MESSAGE "Valeur de I%", STRING I%
```

De même, certains contrôles (entry-field, CBE field, ...) se manipulent directement.

Par exemple :

```
MESSAGE "Contenu de l'entry-field", ENTRY00001
MESSAGE "Contenu de l'entry-field",\
TEST(HANDLE%).ENTRY00001
```

Handle-fenêtre

Dans tout ce qui suit, handle-fenêtre signifie une expression entière, par exemple H%, désignant de façon unique une fenêtre. Ce handle est retourné, dès l'ouverture des fenêtres, par le dernier paramètre des instructions CALL et OPEN.
Le handle de la première fenêtre d'une application est connu par MAINWINDOW%.
Le handle de la fenêtre courante est connu par SELF%.

Contrôle

A chaque fois que contrôle est employé, cela désigne un élément de base tel que check-box, push-button, menu-item, etc.
La syntaxe de contrôle est :

```
[nom-fenêtre (handle-fenêtre).] nom-contrôle
```

ou si le contrôle appartient à un template :

```
nom-template.nom-contrôle
```

Si nom-contrôle est utilisé seul, le contrôle désigné appartient à la fenêtre courante.
Cela peut aussi signifier l'aire client d'une fenêtre de classe Edit ou List avec :

```
[nom-fenêtre (handle-fenêtre).]CLIENT
```

Cela peut aussi signifier l'ascenseur associé à une fenêtre ayant le style "Horz Scroll-Bar" ou "Vert Scroll-Bar" coché dans le volet de propriétés de la fenêtre. La syntaxe est alors :

```
[nom-fenêtre (handle-fenêtre).]HSCROLL
[nom-fenêtre (handle-fenêtre).]VSCROLL
```

Objet

A chaque fois que objet est employé, cela signifie :

```
contrôle | nom fenêtre | nom-fenêtre(handle-fenêtre)
```

(où contrôle et fenêtre ont la signification vue ci-dessus)

Notes :

- nom-fenêtre(handle-fenêtre) doit être employé dès que NatStar a besoin de connaître le nom de la fenêtre pour pouvoir accéder à un contrôle. L'existence d'un contrôle est lié au nom de la fenêtre dans laquelle il a été conçu via l'éditeur de fenêtres.
- De plus, si deux fenêtres de même nom ont été démarrées, les "mêmes" contrôles existeront dans chacune de ces deux fenêtres. Ce qui explique qu'il faille aussi préciser le handle-fenêtre pour accéder à un contrôle précis.

Stockage de données

NS-DK permet de sauvegarder des données directement dans une fenêtre (ou un contrôle). Cette fonctionnalité évite de passer par une multitude de contrôles cachés pour stocker des données locales.

Syntaxe

Stockage de données :

- <Nom de la fenêtre>(handle%).CLIENT[indice] pour une fenêtre
- <Nom du contrôle>[indice]pour un contrôle

Les données sont stockées dans un tableau et sont créées automatiquement lors d'une tentative d'écriture d'un élément inexistant.

Suppression du tableau :

- DELETE FROM CLIENT pour une fenêtre
- DELETE FROM <Nom du contrôle> pour un contrôle

Insertion de lignes dans le tableau : INSERT AT <line> TO

Suppression de lignes dans un tableau : DELETE <line> FROM

Pour des contrôles de type ListBox, MLE, Bitmap ou ComboBox (avec ou sans EntryField), les données stockées sont visibles dans le contrôle et sont détruites avec le contrôle. Il n'est donc pas conseillé d'utiliser les syntaxes définies précédemment avec ces contrôles.

Nat System recommande l'utilisation du CLIENT plutôt que du <Nom du contrôle> pour des raisons de performance. Cette option est en effet essentiellement destinée aux fenêtres ou aux boîtes de dialogue.

La fonction GETDATA% et l'instruction SETDATA restent valables pour associer un segment.

Le paramètre dynamique .CLIPCHILDREN avec CLIENT permet lors de la repeinture de la zone cliente d'une fenêtre d'exclure les zones correspondant aux contrôles de la fenêtre.

Exemple :

```
local h%
OPEN LISTW, 0, h%

INSERT AT END "UN" TO LISTW(h%).CLIENT
INSERT AT END "DEUX" TO LISTW(h%).CLIENT
INSERT AT END "TROIS" TO LISTW(h%).CLIENT
MESSAGE "LISTW(h%).CLIENT[3]",LISTW(h%).CLIENT[3]
```

Référence du langage NCL

Paramétrage dynamique .*

Lit ou modifie dynamiquement le paramétrage d'un contrôle

Syntaxe	**.ANCHOR** **.AUTOTAB** **.AUTOSCROLL** **.AUTOSIZE** **.AUTOTAB** **.BACKCOLOR** **...** **.WORDWRAP** **.X** **.Y**

1. Le paramétrage d'un contrôle est aussi définissable de façon statique dans son volet de propriétés.
2. L'effet des paramétrages sur les contrôles est détaillé dans la description du volet de propriétés de chaque contrôle.
3. Le testeur de syntaxe, de même que les modules de test et de génération, ne font aucune vérification de validité du paramétrage dynamique. Un MOVE 50 TO ENTRY00001.WWIDTH (avec 2 W) ne provoque aucune erreur : son effet est de générer un événement de comportement SETVALUE vers ENTRY00001, avec PARAM1$ valant WWIDTH et PARAM2$ valant 50.
4. Il n'y a pas de paramétrage dynamique .NAME (équivalent du champ "Name" des volets de propriétés). Voir toutefois la fonction CONTROLNAME$ de la librairie NSMISC.

Exemple 1 :

```
; Changement de la police d'un Entry-Field
MOVE "Times New Roman,14" TO ENTRY00001.FONTNAME
```

Exemple 2 :

```
; Elargissement de 100 pixels d'un Entry-Field
MOVE ENTRY00001.WIDTH + 100 TO ENTRY00001.WIDTH
```

Exemple 3 :

```
; Equivalent de ENABLE BUTTON00001
MOVE FALSE% TO BUTTON00001.DISABLED
```

Paramétrage dynamique	Valeur	Fonction équivalente (1)	Instruction équivalente (1)
.ANCHOR (22)	NS_AS_*		
.AppendLineAutomatiquely (46)	TRUE% | FALSE%		
.APPENDTITLELINE (47)			
.AUTOSCROLL	TRUE% | FALSE%		
.AUTOSIZE	TRUE% | FALSE%		
.AUTOTAB (24)	TRUE% | FALSE%		
.AUTOWIDTH	TRUE% | FALSE%		
.BACKCOLOR (55)	COL_*% , TRANSP_*%	GETBACKCOL%	SETBACKCOL
.BOLD	TRUE% | FALSE%		
.BORDER (43)	TRUE% | FALSE%		
.CASE (2)	TRUE% | FALSE%		
.CANRIGHTCLICK (36)	TRUE% | FALSE%		
.CHARACTERS	« caractères acceptés », par exemple « 0..9 »		
.CLIPCHILDREN (23)	TRUE% | FALSE%		
.COLOREDFRAME(56)	TRUE% | FALSE%		
.COLUMN (3)	entier		
.COLUMNSTYPE (26)	"N" = tri numérique "C" = tri caractère		
.ColumnSwappingEnabled (35)	TRUE% | FALSE%		
.DISABLE_x_TAB	TRUE% | FALSE%		
.DISABLED	TRUE% | FALSE%	ISDISABLED%	DISABLE | ENABLE
.DRAGDROP (34)	TRUE% | FALSE%		

.EDITABLE (52)	TRUE% | FALSE%		
.ERASERECT	TRUE% | FALSE%		
.FILLTEXT (4)	TRUE% | FALSE%		
.FONTNAME	« nom_fonte, taille_fonte », par exemple « Tms Rmn,12 »		
.FONTSELS (5)	Combinaison de constantes GFS_*%		
.FORECOLOR (25)	COL_*%, TRANSP_*%	GETFORECOL%	SETFORECOL
.FORCECHILD (40)	TRUE% | FALSE%		
.FORMAT	« format Microsoft_Excel », par exemple		
.HALFTONE	TRUE% | FALSE%		
.HANDLE	pointer		
.HEIGHT	entier	GETHEIGHT%	/
.HIDDEN (6)	TRUE% | FALSE%	ISHIDDEN%	HIDE | SHOW
.HIDETEXT	TRUE% | FALSE%		
.HILITESELECTION (42)	TRUE% | FALSE%		
.HORZSCROLLBAR	TRUE% | FALSE%		
.HOTTRACKED (33)	TRUE% | FALSE%		
.INTERNALVALUE (15)			
.IsTreeBox (53)	TRUE% | FALSE%		
.JUSTIFICATION	Constantes JUST_*%		
.KIND (7)	Constante DI_*%		
.LINE (3)	entier		
.LISTWIDTH (17)	entier		
.LOCKED	TRUE% | FALSE%		
.MARGIN	TRUE% | FALSE%		

.MAXLEN (9)	entier		
.MNEMONIC (6) (10)	TRUE% | FALSE%		
.MOUSEPOINTER (6) (18)	Handle pointeur système		
.MULTIPLESEL	TRUE% | FALSE%		
.NB_COLUMNS (38)			
.NBLINES (15)			
.NOADJUSTPOS	TRUE% | FALSE%		
.NOARROW (13)	TRUE% | FALSE%		
.NOBLANKS	TRUE% | FALSE%		
.NOCHARACTER (44)	TRUE% | FALSE%		
.NOCHECKING	TRUE% | FALSE%		
.NOPOINTERFOCUS	TRUE% | FALSE%		
.ORIENT (21)	integer		
.PARENTID			
.PICTURE (19)	integer	GETMENU ITEMPICTURE%	SETMENU ITEMPICTURE
.R*C*_BACKCOLOR (32)	COL_*%		
.R*C*_FORECOLOR (32)	COL_*%		
.REALTIMESCROLL	TRUE% | FALSE%		
.RELIEF (11)	Constante Rel_*%		
.REQUIRED	TRUE% | FALSE%		
.SEARCHABLE (37)	SEARCH_*		
.SEPARATORS	"car1,car2,&. "		
.SHEETBITMAPSWIDTH (50)	TRUE% | FALSE%		
.SHOWTITLE (41)	TRUE% | FALSE%		
.ShowDottedLines (49)	TRUE% | FALSE%		
.ShowLeavesDottedLines (48)	TRUE% | FALSE%		
.SORTORDER (27)	chaîne de caractères		

.SORTUIENABLED (28)	TRUE% | FALSE%		
.SORTFUNC (45)	POINTEUR de fonction		
.SKIPBLANKS	TRUE% | FALSE%		
.SUBKIND (8)	DI_TDATA% DI_TDSRCH% DI_TLDATA% DI_TLITER%		
.TAB_IDENT	TRUE% | FALSE%		
.TABULATIONS	" tab1,tab2,&. "		
.TEXT (57)	" texte "	GETTEXT$	SETTEXT
.TEXTSEL			
.TITLE (39)			
.TITLEBACKCOLOR (29)	COL_*%		
.TITLEFORECOLOR (30)	COL_*%		
.TOPLINE (14)	entier		
.TOOLTIP (20)	"text"		
.TooltipSeparator (54)	TRUE% | FALSE%		
.TRANSPARENCY (31)	COL_*% ou TRANSP%		
.TreeBoxColumnWidth (51)	TRUE% | FALSE%		
.TYPE	Constante TYP_*%		
.UPDATED (6)	TRUE% | FALSE%		
.VALUE	entier, réel ou chaîne de caractères	MOVE (11)	MOVE
.VERTSCROLLBAR	TRUE% | FALSE%		
.WIDTH	entier	GETXPOS%	SETPOS
.WORDWRAP	TRUE% | FALSE%	GETYPOS%	SETPOS
.X	entier	GETXPOS%	SETPOS
.Y	entier	GETYPOS%	SETPOS

1. Les paramétrages dynamiques qui ont des fonctions ou instructions NCL équivalentes génèrent les mêmes événements de comportement que ces

fonctions ou instructions. Ainsi un MOVE CTRL.DISABLED TO I% génère un appel à ISDISABLED vers le contrôle CTRL.

2. .CASE correspond au "Upcase" du volet de propriétés.
3. .COLUMN et .LINE permettent de connaître ou forcer la position du caret clignotant au sein d'un champ d'édition. Dans un contrôle SheetBox, .LINE positionne ou retourne toutes les colonnes avec leurs séparateurs.
4. .FILLTEXT correspond au "Force Fill" du volet de propriétés.
5. Voir la signification des GFS_*% dans la librairie NSGraph.
6. Ces paramétrages dynamiques nont pas de correspondant statique dans les volets de propriétés.
7. .KIND ne doit être utilisé qu'en lecture. Il permet de connaître la famille du contrôle : Entry-Field, Push-Button, etc... Voir la signification des DI_*% dans la librairie NSMisc.
8. .SUBKIND ne doit être utilisé qu'en lecture. Il permet de connaître la famille du contrôle : Data template, Ldata template, etc... Pour plus dinformations sur les constantes DI_*% consulter la librairie NSTHINGS :
   a) DI_TDATA% (Data template, déclaré 200 en interne)
   b) DI_TDSRCH% (D_SRCH service Data template, déclaré 201 en interne)
   c) DI_TLDATA% (LData template, déclaré 202 en interne)
   d) DI_TLITER% (Iterator template, déclaré 203 en interne)
9. .MAXLEN correspond au "Length" du volet de propriétés.
10. Par défaut, le paramétrage dynamique .MNEMONIC vaut TRUE%, ce qui a pour effet de souligner tout caractère du texte qui suit un tilde "~", afin de désigner un accélérateur clavier. Lorsque .MNEMONIC est positionné à FALSE%, les tildes "~" du texte sont affichés comme les autres caractères, sans avoir aucun effet sur le caractère suivant.
11. .RELIEF correspond au "Shadow" du volet de propriétés.
12. MOVE ou toute instruction demandant la valeur du contrôle (IF, EVALUATE, etc.)
13. Cache (si mis à TRUE%) ou affiche (si mis à FALSE%) la flèche à droite d'une Combo Box ou d'une CBE lorsque le contrôle n'a pas le focus. Pour un contrôle SheetBox, permet de récupérer ou de définir lhandle de la première ligne visible.
14. Indique (lecture) ou modifie (écriture) l'index de la 1ère ligne. Dans le cas d'une List box, ne concerne que les lignes pouvant défiler.
15. Indique (lecture uniquement) le nombre de lignes affichées dans la liste. Dans le cas d'une List box, ne concerne que les lignes pouvant défiler.

16. Indique (lecture uniquement) la valeur associée au Radio bouton (valeur définie dans son volet de propriétés).

17. .LISTWIDTH indique (lecture) ou modifie (écriture) la largeur de la liste associée à une Combo box ou une CBE (la valeur, exprimée en pixels, doit être comprise entre 2 et 1024). Si la valeur de .LISTWIDTH est supérieur à la largeur de la Combo Box (ou de la CBE), la liste s'affiche avec la valeur de .LISTWIDTH. Si la valeur de .LISTWIDTH est inférieure à la largeur de la Combo Box (ou de la CBE), la liste s'affiche avec la largeur de la Combo Box.

Par ailleurs, si le paramètre .LISTWIDTH n'a pas été positionné en écriture, sa lecture renvoie toujours la largeur de la Combo Box. Une fois .LISTWIDTH positionné par programmation, sa lecture renvoie toujours la valeur définie par programmation.

18. Affecte à un contrôle ou une fenêtre un pointeur souris différent qui saffiche lors du passage du pointeur souris sur ce contrôle ou cette fenêtre. La valeur à passer est le handle dun pointeur système obtenu par GETSPTR%. Inversement, il est possible de connaître le pointeur système associé à un contrôle ou à une fenêtre.

19. .PICTURE est supporté uniquement pour les items de menu. Affecte une image au menuItem. Les constantes SMIP_*% pour les images prédéfinies des items de menu sont intégrées dans le fichier NSMISC.NCL.

```
mnu_close.picture = SMIP_ExitApp% ; a predefined picture to exit the application
mnu_del.picture = SMIP_EditDelete% ; the picture of the Delete item of the Edit menu
```

Pour plus d'informations, voir WIL_APPENDFROMBMP%.

20. .TOOLTIP fonctionne avec tous les types de contrôles. Il permet d'insérer des bulles d'aide.

```
EF_NOM.TOOLTIP ="Saisissez votre nom"
```

Quand le pointeur de la souris passe au dessus du contrôle, une bulle jaune s'affiche pour quelques secondes dans laquelle l'utilisateur peut voir le texte du Tooltip.

21. Pour surcharger la configuration courante de l'imprimante avec l'orientation du papier de votre choix. A la lecture, le paramètre dynamique .ORIENT retourne les valeurs suivantes :

a) 2 (l'orientation papier est forcée à paysage par PO_LANDSCAPE%)

b) 1 (l'orientation papier est forcée à portrait par PO_PORTRAIT%)

c) 0 (forcée à la configuration courante de l'imprimante)

d) -1 (la configuration courante de l'imprimante et l'orientation portrait)

e) -2 (la configuration courante de l'imprimante et l'orientation paysage)

Valeurs possibles du paramètre ORIENT :

a) RP_PRINT.ORIENT = 0: (orientation courante)

b) RP_PRINT.ORIENT = PO_PORTRAIT%: (orientation portrait)

c) RP_PRINT.ORIENT = PO_LANDSCAPE%: (orientation paysage)

Ce paramètre doit être défini avant l'exécution du bouton Print, par exemple en mettant le code correspondant dans l'événement INIT de la fenêtre contenant le bouton Print.

22. .ANCHOR est en écriture seule.

23. .CLIPCHILDREN permet lors du réaffichage de la zone cliente dune fenêtre dexclure les zones correspondant aux contrôles de la fenêtre. Ce paramètre est utilisable uniquement avec le pseudo-contrôle CLIENT d'une fenêtre de dialogue.

```
CLIENT.CLIPCHILDREN = FALSE% ; enlève l'état ClipChildren.
CLIENT.CLIPCHILDREN = TRUE% ; positionne l'état ClipChildren.
```

Dans le cas où on utilise l'ancrage dans une boîte de dialogue retaillable et que certains de ses contrôles se repeignent mal (par exemple des custom controls), on peut corriger le problème en mettant le flag CLIENT.CLIPCHILDREN à FALSE%.

```
TANCRA4(self%).Client.CLIPCHILDREN=FALSE%
```

Ce paramètre permet essentiellement de résoudre les problèmes de compatibilité avec les versions précédentes. Par défaut en version 5, ce flag est positionné à vrai automatiquement pour les boîtes de dialogue retaillables avec l'option Size Redraw cochée.

24. .AUTOTAB est utilisé pour des contrôles Entry-Field et des CBE destinés à saisir du texte de taille fixe. Une fois le nombre de caractères fixe renseigné, une tabulation est déclenchée automatiquement sur le contrôle suivant.

25. .FORECOLOR permet de positionner la couleur d'avant-plan et la couleur de transparence des bitmaps.

```
BMP.FORECOLOR=TRANSP_TOPLEFT%
```

26. .COLUMNSTYPE indique le type de tri voulu pour les colonnes de la SheetBox. Le type de chacune des colonnes est séparé par une virgule. Le type est défini par "N" si on veut que les colonnes soient triées en tri numérique, ou "C" si on veut un tri de type "caractère ". Par défaut, le tri est de type C.

27. .SORTORDER permet de préciser lordre des colonnes sur lequel doit porter le tri, de majeur à mineur. La première colonne est comptée comme colonne

n°1. Si on veut trier une colonne par ordre décroissant, on indiquera une valeur négative.

28. .SORTUIENABLED permet dactiver ou de désactiver le tri. Si le tri est activé, il est déclenché alors par un simple clic sur le titre de la colonne. Un premier clic permet de trier dans un sens (ascendant ou descendant), le deuxième clic trie dans le sens inverse.

29. .TITLEBACKCOLOR permet de modifier la couleur de fond du titre du contrôle SheetBox.

30. .TITLEFORECOLOR permet de modifier la couleur de la police du titre du contrôle SheetBox.

31. .TRANSPARENCY permet de modifier ou lire la couleur de transparence des bitmaps du contrôle SheetBox.

32. .R*C*_FORECOLOR et .R*C*_BACKCOLOR permettent de définir une répétition de couleurs sur les lignes et les colonnes successives.

33. .HOTTRACKED permet d'encadrer la cellule au moment du déplacement de la souris.

34. .DRAGDROP permet d'activer loption glisser/déposer des nSuds du contrôle SheetBox avec le bouton droit de la souris.

35. .ColumnSwappingEnabled permet de changer la place des colonnes en glissant-déposant le titre de la colonne dans un contrôle SheetBox avec le bouton droit de la souris.

36. .CANRIGHTCLICK permet dactiver la sélection dune ligne dans un contrôle SheetBox par un clic-droit et ainsi denvoyer lévénement SELECTED. Utile pour intégrer par exemple des menus contextuels.

37. .SEARCHABLE permet deffectuer une recherche sur les éléments dune colonne.

38. .NB_COLUMNS permet de récupérer ou de définir le nombre de colonnes dun contrôle SheetBox.

39. .TITLE permet de modifier le titre dune colonne dun contrôle SheetBox.

40. .FORCECHILD affiche ou retire le '+' devant un nSud fils terminal dans un contrôle SheetBox.

41. .SHOWTITLE permet de cacher la ligne de titre du contrôle SheetBox.

42. .HILITESELECTION permet de ne plus afficher de curseur et d'interdire toute sélection dans un contrôle SheetBox.

43. .BORDER positionne (ou retire) l'attribut du contrôle en style bordure fine dans un contrôle SheetBox.

44. .NOCHARACTER permet d'interdire la recherche qui a lieu sur la première colonne en saisissant un caractère.

45. .SORTFUNC permet de spécifier la fonction de tri des colonnes de la SheetBox.
46. .AppendLineAutomatiquely permet dajouter une ligne automatiquement en mode Edit in place dans un contrôle SheetBox.
47. .APPENDTITLELINE ajoute une nouvelle ligne de titre qui a les mêmes attributs que la ligne de titre de la SheetBox.
48. .ShowLeavesDottedLines quand il n'y a pas de bitmap associée dans un contrôle SheetBox, ce paramètre retire les pointillés correspondant à la largeur de la bitmap.
49. .ShowDottedLines retire les pointillés des lignes dans un contrôle SheetBox.
50. .SHEETBITMAPSWIDTH permet de récupérer la largeur des bitmaps d'un contrôle SheetBox.
51. .TreeBoxColumnWidth permet de récupérer la largeur des colonnes d'un contrôle SheetBox.
52. .EDITABLE autorise la saisie dans le contrôle SheetBox.
53. .IsTreeBox permet de spécifier le style du contrôle SheetBox en TreeBox ou en ListBox.
54. .TooltipSeparator indique le séparateur utilisé pour les bulles d'aide permettant ainsi d'insérer une ligne complète avec bulle d'aide : <texte><séparateur de bulle d'aide><texte de la bulle d'aide><séparateur de colonne>.
55. .BACKCOLOR permet de définir la couleur de remplacement de la couleur « transparente » des bitmaps.
56. .COLOREDFRAME ne s'applique qu'aux Group-Boxes et permet de donner la couleur de foreground au cadre.
57. .TEXT permet de changer dynamiquement un contrôle bitmap en lui affectant le handle d'une autre bitmap. Voir le chapitre Gestion des Handle de Bitmap

Paramètrage dyn.	Bmp	Classifier	ChkB	ComB	CBE	EntF (*)	GrpB	HScB	Ico	LisB	MLE	MnuItem	PshB	R
.ANCHOR	O	O	O	O	O	O	O	O	O	O	O		O	
.AppendLineAutomatiquely														
.APPENDTITLELINE														
.AUTOSCROLL					O	O								
.AUTOSIZE			O											
.AUTOTAB					O	O								

.AUTOWIDTH		O											
.BACKCOLOR			O	O	O	O	O			O	O		O
.BOLD		O											
.BORDER													
.CASE					O	O					O		
.CANRIGHTCLICK													
.CHARACTERS					O	O							
.COLOREDFRAME							O						
.COLUMN					O	O					O		
.COLUMNSTYPE													
.ColumnSwappnigEnabled													
.DISABLE_x_TAB		O											
.DISABLED	O		O	O	O	O	O	O		O	O		O
.DRAGDROP													
.EDITABLE													
.ERASERECT							O						
.FILLTEXT					O	O							
.FONTNAME			O	O	O	O	O			O	O		O
.FONTSELS			O	O	O	O	O			O	O		O
.FORCECHILD													
.FORECOLOR			O	O	O	O	O			O	O		O
.FORMAT					O	O							
.HALFTONE							O						
.HANDLE		O											
.HEIGHT	O			(g)	(g)		O			O	O		O
.HIDDEN	O		O	O	O	O	O	O	O	O	O		O
.HIDETEXT					O	O							
.HILITESELECTION													
.HORZSCROLLBAR										O			
.HOTTRACKED													
.INTERNALVALUE													
.ISTREEBOX													
.JUSTIFICATION					O	O	O						
.KIND	O		(c)	O	O	O	O	O	O	O	O		O

.LINE										O	O		
.LISTWIDTH				O	O								
.LOCKED	O		O	O	O	O		O		O	O		O
.MARGIN						O	O						
.MAXLEN						O	O						
.MNEMONIC							O						
.MOUSEPOINTER			O	O	O	O		O		O	O		O
.MULTIPLESEL										O			
.NB_COLUMNS													
.NBLINES										O	O		
.NOADJUSTPOS										O	O		
.NOARROW				O	O								
.NOBLANKS					O	O							
.NOCHARACTER													
.NOCHECKING													O
.NOPOINTERFOCUS(k)	O												O
.PARENTID												O	
.PICTURE												O	
.R*C*_FORECOLOR													
.R*C*_BACKCOLOR													
.REALTIMESCROLL				O	O					O			
.RELIEF	O		O	O	O	O	O	O		O	O		
.REQUIRED					O	O							
.SEARCHABLE													
.SEPARATORS				O						O			
.SHEETBITMAPSWIDTH													
.SHOWTITLE													
.ShowDottedLines													
.ShowLeavesDottedLines													
.SKIPBLANKS					O	O							
.SORTFUNC													
.SORTORDER													
.SORTUIENABLED													
.TAB_IDENT		O											

.TABULATIONS				O						O			
.TEXT	(j)		O	O	O	O	O			O	O		O
.TEXTSEL(1)						O					O		
.TITLE													
.TITLEBACKCOLOR													
.TITLEBACKCOLOR													
.TOOLTIP	O	O	O	O	O	O	O	O	O	O	O	O	O
.TOOLTIPSEPARATOR													
.TOPLINE										O	O		
.TRANSPARENCY													
.TreeBoxColumnWidth													
.TYPE					O	O							
.UPDATED				O	O					O	O		
.VALUE	O		(d)	O	O	O	O	O	O	O	O		O
.VERTSCROLLBAR										O			
.WIDTH	O		(f)	O	O	O	O	O		O	O		O
.WORDWRAP											O		
.X	O		O	O	O	O	O	O	O	O	O		O
.Y	O		O	O	O	O	O	O	O	O	O		O

Notes complémentaires :

1. Pour un Radio Button, .AUTOSIZE correspond au "Auto Width" du volet de propriétés.
2. Pour un Static Text, .HEIGHT ne fonctionne que si .WORDWRAP vaut TRUE%.
3. Pour un Check Box, .KIND vaut DI_CHECKBOX% ou DI_CHECKBOX3% selon l'état de "3 States" dans le volet de propriétés.
4. Pour un Check Box, .VALUE vaut CHECKED%, UNCHECKED% ou INDETERMINATE%.
5. Pour un Radio Button, .VALUE retourne la valeur du Radio Button sélectionné.
6. Pour un Radio Button, un Static Text, une Check Box, .WIDTH ne fonctionne que si AUTOSIZE=TRUE%.
7. Pour une Combo Box et une CBE, .HEIGHT correspond à la hauteur de la List Box en pixels. A partir de la version 5.0, on peut également indiquer la hauteur de la List Box en nombre de lignes visibles en utilisant le caractère #. Par exemple, CB.HEIGHT="10" indique que la List Box associée à la Combo Box CB est de 10 pixels. CB.HEIGHT="#10" indique que la List Box quand elle

est déroulée à 10 lignes. Par défaut, la hauteur d'une List Box associée à une Combo Box et à une CBE est de 6 lignes.

8. Pour un Static Text, .WORDWRAP=FALSE% ne provoque de redimensionnement que si AUTOSIZE=TRUE%.

9. .VALUE est équivalent à .TEXT pour les Combo Box, CBE, Entry Field, Text, List Box, MLE, Group Box et Push Button.

10. Pour un Bitmap, .TEXT correspond au handle du bitmap 'Released'.

11. Uniquement pour un contrôle bitmap de type Push button et un contrôle Push Button. Un clic souris ne donne alors pas le focus au bitmap.

12. Ce paramètre agit en lecture sur les Entry Field et les MLE. Il permet de récupérer la sélection courante effectuée dans ces contrôles. Maximum récupérable: 255 caractères.

13. Pour plus d'informations sur les paramètres dynamiques spécifiques au contrôle SheetBox, reportez-vous à la rubrique Paramétrage dynamique d'un contrôle SheetBox.

(*) Pour un Entry Field en 'exploding', les seuls paramétrages dynamiques sont : Margin, Required, FillText, Case, SkipBlanks, NoBlanks et Characters

LISTBOX : LINE : N° de la ligne sélectionnée, NBLINES : nombre de lignes affichables. TOPLINE N° d'ordre de la 1ere ligne affichée.

Opérateur ABS

Syntaxe	**ABS** *entier* | *réel*

Exemple :

```
MOVE ABS -14 TO I% ; I% vaut 14
```

Voir aussi ***ROUND%***, ***TRUNC%***

Opérateur AND

Syntaxe	*entier* **AND** *entier*

1. L'opérateur teste si les deux opérandes valent TRUE%, ou plus généralement si les deux opérandes ont des valeurs différentes de zéro.
2. L'opérande de droite n'est calculé que si nécessaire. En effet, lorsque l'opérande de gauche est égal à zéro, le résultat est toujours FALSE%.

Exemple :

```
MOVE (TRUE% AND TRUE%) TO I% ; I% vaut TRUE%
MOVE (TRUE% AND FALSE%) TO I% ; I% vaut FALSE%
MOVE (FALSE% AND TRUE%) TO I% ; I% vaut FALSE%
MOVE (FALSE% AND FALSE%) TO I% ; I% vaut FALSE%
```

***Voir aussi** NOT, OR, BAND, BOR, BXOR, BNOT*

Fonction ASC%

Syntaxe	**ASC%** *(chaîne)*

ASC% retourne 0 si la chaîne est vide.

Exemple :

```
MOVE ASC% ("A") TO I% ; met 65 dans I%
MOVE CHR$ (ASC% ("A")) TO S$ ; met "A" dans S$
```

Voir aussi ***CHR$***

Séparateur ASCENDING

Syntaxe	INSERT **ASCENDING** | DESCENDING *chaîne* TO *contrôle* LOAD [**ASCENDING** | DESCENDING] *nom-fichier* TO *contrôle*

Voir aussi Instructions INSERT, LOAD

Fonction ASK2%

Syntaxe	**ASK2%** *(chaîne-titre,chaîne-message)*

1. Le message à afficher doit figurer sous forme de question dont la réponse ne peut être que Oui ou Non.
2. ASK2% retourne YES% si l'utilisateur appuie sur le bouton Yes et NO% s'il appuie sur No.
3. L'intitulé des boutons Yes et No est affiché dans la langue définie pour le système d'exploitation.

Exemple :

```
IF ASK2% ('Attention', 'Voulez-vous terminer ?') = YES%
; appui sur le bouton Yes
CLOSE
ENDIF
```

Voir aussi :
ASK3%, MESSAGE
Constantes YES%, NO%
MESSAGE% de la librairie NSMisc

Fonction ASK3%

Syntaxe	**ASK3%** *(chaîne-titre, chaîne-message)*

1. Le message à afficher doit figurer sous forme de question. L'utilisateur peut répondre par Oui ou Non ou annuler l'action prête à être effectuée.
2. ASK2% retourne YES% si l'utilisateur appuie sur le bouton Yes, NO% s'il appuie sur No et CANCEL% s'il appuie sur Cancel.
3. L'intitulé des boutons Yes, No et Cancel est affiché dans la langue définie pour le système d'exploitation.

Exemple :

```
MOVE ASK3% ('Attention', 'Voulez-vous terminer ?') TO I%
IF I% = YES%
; Yes sélectionné
ELSEIF I% = NO%
; No sélectionné
ELSE
; Cancel sélectionné
ENDIF
```

Voir aussi

ASK2%, MESSAGE

Constantes YES% , NO%, CANCEL%

MESSAGE% de la librairie NSMisc

Séparateur AT

Syntaxe	DELETE [**AT** *position*] FROM *objet* INSERT **AT** *position* | END *chaîne* TO *objet*

Voir aussi Instructions DELETE, INSERT

Opérateur BAND

Syntaxe	*entier* **BAND** *entier*

Exemple :

```
MOVE (2#0011 BAND 2#0101) TO I% ; I% vaut 2#0001, soit 1 en décimal
```

Voir aussi ***AND*****,** ***OR*****,** ***NOT*****,** ***BOR*****,** ***BNOT*****,** ***BXOR***

Instruction BEEP

Syntaxe	**BEEP** *[nb_beep]*

Cette instruction peut être utilisée dans des buts de mise au point, pour avoir un signal sonore sur un passage précis dans le code. Elle peut aussi être associée à un MESSAGE pour, par exemple, insister sur un avertissement important.
Exemple :

```
BEEP ; Un BIP
BEEP I% * 3 ; I% * 3 BIPS
```

Voir aussi MESSAGE

Opérateur BNOT

Syntaxe	**BNOT** *entier*

Exemple :

```
LOCAL I%(1) ; I% stocké sur un octet
MOVE BNOT 2#00101110 TO I% ; I% vaut 2#11010001
```

***Voir aussi** AND, OR, NOT, BAND, BOR, BXOR*

Opérateur BOR

Syntaxe	*entier* **BOR** *entier*

Exemple :

```
MOVE (2#0011 BOR 2#0101) TO I% ; I% vaut 2#0111 soit 7 en décimal
```

Voir aussi ***AND*****,** ***OR*****,** ***NOT*****,** ***BAND*****,** ***BNOT*****,** ***BXOR***

Instruction BREAK

Arrêt immédiat d'une opération logique de type LOOP/ENDLOOP, REPEAT/UNTIL, WHILE/ENDWHILE et FOR/ENDFOR par branchement direct sur l'instruction qui suit le ENDLOOP, UNTIL, ENDWHILE ou ENDFOR.

Syntaxe	**BREAK**

Cette instruction n'a aucun effet en dehors des constructions LOOP/ENDLOOP, REPEAT/UNTIL et WHILE/ENDWHILE et FOR/ENDFOR.

Exemple 1 :

```
MOVE ... TO J%
MOVE 1 TO I%
REPEAT
MOVE I%+1 TO I% ; Répétée jusqu'à ce que I%=10
IF J% = FALSE% ; sauf si J% vaut FALSE%  :
BREAK  ; fin du REPEAT et I% vaudra 2
ENDIF
UNTIL I% = 10
```

Exemple 2 :

```
MOVE ... TO J%
MOVE 1 TO I%
WHILE COPY$ (TEST, I%, 1) <> "-"
MOVE I%+1 TO I% ; Répété jusqu'à ce que COPY$ <> "-"
IF J% = FALSE% ; sauf si J% vaut FALSE%  :
BREAK ; fin du WHILE et I% vaudra 2
ENDIF
ENDWHILE
```

***Voir aussi** LOOP, REPEAT, WHILE, FOR, CONTINUE*

Opérateur BXOR

Syntaxe	*entier* **BXOR** *entier*

Exemple :

```
MOVE (2#0011 BXOR 2#0101) TO I% ; I% vaut 2#0110, soit 6 en décimal
```

Voir aussi ***AND*****,** ***OR*****,** ***NOT*****,** ***BAND*****,** ***BOR*****,** ***BNOT***

Instructions CALL, CALLH

Ouvre une fenêtre en mode modal.

Syntaxes	**CALL,** **CALLH** *nom-fenêtre [, handle-fenêtre] [USING Param1 [, Param2 [, Param3 [, Param4 [, variable-retour]]]]]* **CALL,** **CALLH** *nom-fenêtre [, handle-fenêtre] [USING Param1 [, Param2 [, Param34 [, variable-retour]]]]* **CALL,** **CALLH** *nom-fenêtre [, handle-fenêtre] [USING Param12 [, Param3 [, Param4 [, variable-retour]]]]* **CALL,** **CALLH** *nom-fenêtre [, handle-fenêtre] [USING Param12 [, Param34 [, variable-retour]]]*

1. Une fenêtre modale est une fenêtre qui bloque les autres fenêtres de l'application : seuls les événements de la fenêtre modale ne sont pas interrompus tant que la fenêtre fille n'est pas appelée. En fait, le handle de la fenêtre est stocké dès le début du CALL, ce qui fait que la variable handle-fenêtre peut être utilisée par le code NCL de la fenêtre démarrée, et non bien sûr par la fenêtre qui a fait le CALL qui récupère "trop tard" ce handle. La seule condition est que cette variable soit globale.

2. Puisque CALL ne retourne que lorsque la fenêtre démarrée a été fermée, il peut paraître inutile de spécifier le paramètre handle-fenêtre. Celui-ci, retourné par CALL, est invalide dés l'instruction située sur la ligne suivante. En fait, le handle de la fenêtre utilisée par le code NCL de la fenêtre démarrée, et non bien sûr par la fenêtre qui a fait le CALL qui récupère "trop tard" ce handle. La seule condition est que cette variable soit globale.

3. Les paramètres Param1, Param2, Param3, Param4, Param12, Param34 seront reçus dans les fonctions PARAM1%, PARAM2%, PARAM3%, PARAM4%, PARAM12%, PARAM34% de l'événement INIT de la fenêtre ouverte.

4. Le handle de fenêtre retourné n'a pas d'intérêt car sa valeur n'a plus de sens une fois la fenêtre appelée fermée. il correspond au SELF% de la fenêtre appelée.

5. variable-retour, entier sur 2 octets, est la valeur retournée par l'instruction CLOSE de la fenêtre appelée.
6. L'instruction CALLH a un comportement un peu différent de CALL car elle laisse cachée la fenêtre ouverte. Un emploi ultérieur de l'instruction SHOW permet de la faire apparaître. Le SHOW ne peut être effectué que depuis la fenêtre démarrée et non depuis la fenêtre qui a réalisé le CALLH.
7. N'utilisez ni CALL ni CALLH pour ouvrir une fenêtre fille MDI (de classe quelconque, mais dont la mère est de classe MDI Window).
8. Respectez la description syntaxique de ces instructions. Ne pas faire figurer la variable-retour directement après USING. Lorsque tous les paramètres Param1, Param2, Param3, Param4, Param12, Param34 ne sont pas utilisés, il est nécessaire de remplacer chaque paramètre non employé par la valeur 0 pour prendre en compte la variable retour dans l'instruction.
9. Les fonctions PARAM12% et PARAM34% sont désormais des pointeurs.
10. Dans les emplacements prévus pour les paramètres PARAM1% et PARAM3%, il est possible de remplacer les opérateurs LOW expr_ent%, HIW expr_ent% par POINTER expr_ent%.
11. De même, si une expression de type ptr, ptr+expr_ent%, ptr-expr_ent% ou expr_ent%+ptr (où ptr est soit une variable de type POINTER, soit l'adresse d'une variable ou d'une chaîne littérale) apparaît à l'un des emplacements de PARAM1% et PARAM3% le pointeur est passé comme PARAM12% ou PARAM34% sans même avoir besoin de le préfixer par le type POINTER.
12. Si les expressions passées en paramètre sont de type POINTER, alors les instructions CALL, CALLH utilisent PARAM12% au lieu de PARAM1% et PARAM2% ou PARAM34% au lieu de PARAM3% et PARAM4%.
13. Si une expression de type POINTER est passé sans changement de type en INT aux emplacements prévus pour PARAM2% ou PARAM4%, vous obtiendrez un avertissement ou une erreur.

Exemple 1 :

```
CALL TEST ; ouvre la fenêtre TEST (fichier TEST.SCR)
BEEP ; le bip ne sera fait qu'après la fermeture de TEST
```

Exemple 2 :

```
LOCAL PAR1%, PAR2%, PAR3%, PAR4%
LOCAL RETOUR%(2)
; Initialisation des paramètres
...
CALL D_SCREEN, H_SCREEN% USING PAR1%, PAR2%, \
PAR3%, PAR4%, RETOUR%

; Récupération du code de retour après fermeture
```

```
; de la fenêtre D_SCREEN
MESSAGE "Retour% = " , RETOUR%
```

Voir aussi STRCALL , OPEN , STROPEN , SHOW, CLOSE, Fonctions PARAM*%, PARAM*$, WINDOWNAME$ (Librairie NSMisc)

Constante CANCEL%

Syntaxe	CANCEL%

Sa déclaration interne est :

```
CONST CANCEL% 2
```

Voir aussi Fonction ASK3%, Constantes YES%, NO%

Instruction CAPTURE

Syntaxe	**CAPTURE** *[handle-fenêtre]*

1. Si aucun paramètre n'est passé, la fenêtre courante capture tous les événements souris jusqu'à un prochain UNCAPTURE.Si un paramètre est passé, ce sera la fenêtre dont le handle est spécifié qui capturera les événements souris.
2. Les captures souris sont souvent utilisées lorsqu'un objet est déplacé dans une fenêtre : l'utilisateur maintient l'objet par la souris (bouton enfoncé) et on ne désire pas que l'utilisateur "perde" l'objet en débordant par inadvertance les limites physiques de la fenêtre. En effet, si l'on ne capture pas la souris, les événements souris sont toujours passés à la fenêtre physiquement située sous le pointeur souris.
3. Ne pas oublier de faire par la suite un UNCAPTURE. La logique est souvent de faire un CAPTURE sur l'événement BUTTONDOWN, et de faire un UNCAPTURE sur BUTTONUP.
4. A un instant donné, une seule fenêtre peut capturer la souris.

Exemple :

```
CAPTURE ;la fenêtre courante capture la souris
CAPTURE TEST% ;la fenêtre dont le handle est dans TEST%
;capture la souris
```

Voir aussi :
Instruction UNCAPTURE
Evénements BUTTONUP, BUTTONDOWN, MOUSEMOVE
Exemple dans le chapitre NSGraph du Manuel de Programmation Librairies

Instruction CHANGE

Syntaxe	**CHANGE** *[handle-fenêtre]* **TO** *pos-x, pos-y [,largeur, hauteur]*

1. Position et taille doivent être données en pixels.
2. Les coordonnées pos-x et pos-y sont celles du coin inférieur gauche de la fenêtre
3. Si handle-fenêtre n'est pas précisé, le changement s'applique à la fenêtre courante.
4. La position d'une fenêtre est relative à l'écran, à l'exception d'une fenêtre fille dont la position est relative à sa fenêtre mère.

Exemple 1 :

```
CHANGE TO 10,10 ; Place la fenêtre courante en 10,10
```

Type CHAR

1. L'intérêt de travailler avec des CHAR, plutôt qu'avec des STRING ou CSTRING, est de pouvoir traiter une chaîne comme un tableau.
2. L'inconvénient de travailler avec des CHAR est que toutes les fonctions ou instructions NCL n'autorisent en paramètre que des STRING ou CSTRING : une variable CHAR passée en paramètre est donc finalement elle aussi limitée dans la pratique à 255.
3. TOUTEFOIS, il est possible d'utiliser les sous-chaînes afin de convertir des parties de variables CHAR en variable STRING.

Exemple :

```
LOCAL CHAR C$(1000) ; possible jusqu'à 65000 !
LOCAL S1$, S2$, S3$, S4$ ; CSTRING de 255 caractères
...
; Stockage de C$ en quatre CSTRING
MOVE C$(0..249) TO S1$
MOVE C$(250..499) TO S2$
MOVE C$(500..749) TO S3$
MOVE C$(750..999) TO S4$
; Chaque CSTRING est maintenant employable par les fonctions et instructions NCL !
```

***Voir aussi** INT, NUM, STRING, CSTRING, CONTROL, LOCAL, GLOBAL, SEGMENT, FUNCTION, INSTRUCTION*

Constante CHECKED%

Syntaxe	CHECKED%

Sa déclaration interne est :

```
CONST CHECKED% 1
```

Exemple :

```
MOVE CHECKED% TO MENUITEM1 ; coche l'élément de menu
IF CHECKBOX1 = CHECKED%
MESSAGE "Vérification", "Le contrôle CHECKBOX1 est coché"
ENDIF
```

Voir aussi Constantes UNCHECKED%, INDETERMINATE%

Fonction CHR$

Syntaxe	**CHR$** *(codeASCII)*

Exemple :

```
MOVE CHR$ (65) TO S$ ; met "A" dans S$
MOVE CHR$ (ASC% ("A")) TO S$ ; met "A" dans S$
```

Voir aussi ASC%

Instruction CLOSE

Syntaxe	**CLOSE** *[handle-fenêtre [, variable-retour]]*

1. Si aucun handle fenêtre n'est spécifié, la fenêtre courante est fermée.
2. Cette instruction a pour effet de générer un événement CHECK vers la fenêtre. Si cet événement n'est pas traité, ou s'il finit par un RETURN 0, un événement TERMINATE est ensuite généré.
3. Il est dangereux de continuer le script derrière un CLOSE de la fenêtre courante si le traitement porte sur la fenêtre elle-même, sur l'un de ses composants ou affiche des fenêtres fille ou des boîtes de message. Comme la fenêtre est généralement immédiatement fermée (selon le traitement des événements CHECK et TERMINATE qui sont instantanément envoyés, sans attendre la fin de traitement du script qui effectue le CLOSE), ce script peut provoquer, de manière aléatoire, une erreur ou un blocage système.
4. Un CLOSE peut être suivi de code NCL dès que le traitement ne porte pas sur l'un des cas précisés auparavant. Il est ainsi possible de coder un RETURN.
5. Le handle de la fenêtre doit nécessairement être spécifié dans l'instruction CLOSE pour pouvoir utiliser la variable code-retour.
6. code retour, entier de 2 octets, est renvoyé à la fenêtre ayant effectué le CALL de la fenêtre. Si aucun code retour n'est spécifié, le code retour renvoyé vaut 0.

Exemple 1 :

```
CLOSE TEST% ;Ferme la fenêtre dont le handle est TEST%
```

Exemple 2 :

```
; Affectation du code retour
IF EF_NOM <> ""
MOVE TRUE% TO RETOUR%
ELSE
MOVE FALSE% TO RETOUR%
ENDIF
CLOSE SELF%, RETOUR%
```

Voir aussi Instructions CALL, OPEN, Evénements CHECK, TERMINATE

Constantes COL_*%

Syntaxe	**COL_ACTIVEBORDER%** **COL_ACTIVETITLE%** **COL_ACTIVETITLETEXT%** **COL_ACTIVETITLETEXTBGND%** **COL_APPWORKSPACE%** **COL_BACKGROUND%** **COL_BFALSE%** **COL_BLACK%** **COL_BLUE%** **COL_BROWN%** **COL_BTRUE%** **COL_BUTTONDARK%** **COL_BUTTONDEFAULT%** **COL_BUTTONLIGHT%** **COL_BUTTONMIDDLE%** **COL_BUTTONTEXT%** **COL_CYAN%** **COL_DARKGRAY%** **COL_DIALOGBACKGROUND%** **COL_ENTRYFIELDBACKGROUND%** **COL_GREEN%** **COL_HELPBACKGROUND%** **COL_HELPHILITE%** **COL_HELPTEXT%** **COL_HILITEBACKGROUND%** **COL_HILITEFOREGROUND%** **COL_ICONTEXT%** **COL_INACTIVEAREA%** **COL_INACTIVEBORDER%** **COL_INACTIVETITLE%** **COL_INACTIVETITLETEXT%** **COL_INACTIVETITLETEXTBGND%** **COL_LIGHTBLUE%** **COL_LIGHTCYAN%** **COL_LIGHTGRAY%** **COL_LIGHTGREEN%** **COL_LIGHTMAGENTA%**

	COL_LIGHTRED% **COL_MAGENTA%** **COL_MENU%** **COL_MENUTEXT%** **COL_NEUTRAL%** **COL_OUTPUTTEXT%** **COL_RED%** **COL_SHADOW%** **COL_SCROLLBAR%** **COL_TITLEBOTTOM%** **COL_TITLETEXT%** **COL_WHITE%** **COL_WINDOW%** **COL_WINDOWBACKGROUND%** **COL_WINDOWFRAME%** **COL_WINDOWSTATICTEXT%** **COL_WINDOWTEXT%** **COL_YELLOW%**

1. Les 16 premières COL_*% (de 0 à 15, soit donc de COL_BLACK% à COL_WHITE%) représentent la totalité des couleurs qu'il est possible d'obtenir. Chaque pixel d'un écran EGA ou VGA est composé de 4 bits : un bit pour l'intensité, un bit pour le rouge, un bit pour le vert et un bit pour le bleu.

2. La totalité des combinaisons possibles pour un pixel donne 16 couleurs (= 2^4).

Couleur	Intensité	Rouge	Vert	Bleu
COL_BLACK%	0	0	0	0
COL_BLUE%	0	0	0	1
COL_GREEN%	0	0	1	0
COL_CYAN%	0	0	1	1
COL_RED%	0	1	0	0
COL_MAGENTA%	0	1	0	1
COL_BROWN%	0	1	1	0
COL_LIGHTGRAY%	0	1	1	1
COL_DARKGRAY%	1	0	0	0
COL_LIGHTBLUE%	1	0	0	1

COL_LIGHTGREEN%	1	0	1	0
COL_LIGHTCYAN%	1	0	1	1
COL_LIGHTRED%	1	1	0	0
COL_LIGHTMAGENTA%	1	1	0	1
COL_YELLOW%	1	1	1	0
COL_WHITE%	1	1	1	1

Le tableau précédent prend son importance lors de l'utilisation de la librairie NSGraph (notamment instruction GPI_SETMIXMODE et constantes GPM_*%).

3. Les COL_*% au-delà de COL_WHITE% reprennent ces 16 couleurs, en leur rajoutant éventuellement un pattern (cf librairie NSGraph pour la définition des patterns) : elles sont généralement modifiables par l'utilisateur grâce au Panneau de Configuration.

a) COL_BACKGROUND% est la couleur utilisée pour peindre le fond des fenêtres. Elle est paramétrable dans le Control Panel, et vaut généralement COL_WHITE%. Voir instruction GPI_ERASE de la librairie NSGraph. Voir aussi événement PAINT.

b) COL_NEUTRAL% est la couleur utilisée pour dessiner (lignes, caractères, ...) dans les fenêtres. Elle est paramétrable dans le Control Panel, et vaut généralement COL_BLACK%. Voir instruction GPI_SETCOLOR de la librairie NSGraph.

c) Sur un écran vidéo, COL_BFALSE% a le même effet que COL_BLACK%, et COL_BTRUE% a le même effet que COL_WHITE%. C'est l'opposé sur une imprimante : les écrans vidéo sont des périphériques à fond noir alors que les imprimantes sont des périphériques à fond blanc.

4. COL_WINDOWBACKGROUND% désigne la couleur de fond de l'écran (et non celle de la fenêtre). Le fond des fenêtres standards est désigné indifféremment par COL_WINDOW% ou COL_BACKGROUND%.

5. COL_DIALOGBACKGROUND% désigne la couleur appelée "Arrière plan d'incrustation" (Pop-Up window background) dans le Panneau de configuration (Control Panel). La liste des applications actives (Task List) est un exemple de ce type de fenêtre.

6. COL_ENTRYFIELDBACKGROUND% est la couleur de fond des entry field et list box.
COL_BUTTONTEXT% est la couleur par défaut du texte des Push buttons.

7. COL_INACTIVEAREA% est la couleur du coin inférieur droit d'une fenêtre lorsqu'elle possède des barres de défilement.

8. Leur déclaration interne est :

```
CONST COL_BLACK% 0
CONST COL_BLUE% 1
CONST COL_GREEN% 2
CONST COL_CYAN% 3
CONST COL_RED% 4
CONST COL_MAGENTA% 5
CONST COL_BROWN% 6
CONST COL_LIGHTGRAY% 7
CONST COL_DARKGRAY% 8
CONST COL_LIGHTBLUE% 9
CONST COL_LIGHTGREEN% 10
CONST COL_LIGHTCYAN% 11
CONST COL_LIGHTRED% 12
CONST COL_LIGHTMAGENTA% 13
CONST COL_YELLOW% 14
CONST COL_WHITE% 15
CONST COL_BACKGROUND% 16
CONST COL_NEUTRAL% 17
CONST COL_BFALSE% 18
CONST COL_BTRUE% 19
CONST COL_BUTTONLIGHT% 20
CONST COL_BUTTONMIDDLE% 21
CONST COL_BUTTONDARK% 22
CONST COL_BUTTONDEFAULT% 23
CONST COL_TITLEBOTTOM% 24
CONST COL_SHADOW% 25
CONST COL_ICONTEXT% 26
CONST COL_DIALOGBACKGROUND% 27
CONST COL_HILITEFOREGROUND% 28
CONST COL_HILITEBACKGROUND% 29
CONST COL_INACTIVETITLETEXTBGND% 30
CONST COL_ACTIVETITLETEXTBGND% 31
CONST COL_INACTIVETITLETEXT% 32
CONST COL_ACTIVETITLETEXT% 33
CONST COL_OUTPUTTEXT% 34
CONST COL_WINDOWSTATICTEXT% 35
CONST COL_SCROLLBAR% 36
CONST COL_WINDOWBACKGROUND% 37
CONST COL_ACTIVETITLE% 38
CONST COL_INACTIVETITLE% 39
CONST COL_MENU% 40
CONST COL_WINDOW% 41
CONST COL_WINDOWFRAME% 42
CONST COL_MENUTEXT% 43
CONST COL_WINDOWTEXT% 44
CONST COL_TITLETEXT% 45
CONST COL_ACTIVEBORDER% 46
CONST COL_INACTIVEBORDER% 47
CONST COL_APPWORKSPACE% 48
CONST COL_HELPBACKGROUND% 49
CONST COL_HELPTEXT% 50
CONST COL_HELPHILITE% 51
CONST COL_ENTRYFIELDBACKGROUND% 52
CONST COL_INACTIVEAREA% 53
CONST COL_BUTTONTEXT% 54
```

Voir aussi :

GETBACKCOL%, GETFORECOL%, SETBACKCOL, SETFORECOL

Librairie NSGraph (GPM_*%, GPI_SETCOLOR, GPI_SETBACKCOLOR, DRAW_TEXT, DRAW_BORDER, DRAW_FILLRECT, et nombreuses instructions GRAPH_*)

Constantes COL_REDEF*%

Les constantes COL_REDEF*% sont des couleurs définies à l'aide de l'instruction CREATERGBPALETTE ou GETSETRGBPALETTE et qui peuvent être utilisées à la place des couleurs standards (COL_*%) dans les applications.

Syntaxe	**COL_REDEF1%** **COL_REDEF2%** **COL_REDEF3%** **COL_REDEF4%** **COL_REDEF5%** **COL_REDEF6%** **COL_REDEF7%** **COL_REDEF8%** **COL_REDEF9%** **COL_REDEF10%** **COL_REDEF11%** **COL_REDEF12%** **COL_REDEF13%** **COL_REDEF14%** **COL_REDEF15%** **COL_REDEF16%** **COL_REDEF17%** **COL_REDEF18%**

Leur déclaration interne est :

```
CONST COL_REDEF1%  55
CONST COL_REDEF2%  56
...
CONST COL_REDEF17%  71
CONST COL_REDEF18%  72
```

Voir aussi CREATERGBPALETTE, GETSETRGBPALETTE, RGBCOLOR

Instruction COMMENT

Syntaxe	**COMMENT** *commentaire*

Cette instruction est rigoureusement identique au caractère spécial ";".
Exemple :

```
COMMENT Ceci est un commentaire ; Ceci est un autre commentaire
```

Instruction CONST

Syntaxe	**CONST** *nom-constante expression-constante*

1. nom-constante est l'identifieur de la constante. Des définitions multiples de constantes peuvent être déclarées sur la même ligne en séparant chaque définition par une virgule (,).
2. L'identifieur de la constante doit avoir un suffixe indiquant son type.
3. Les définitions CONST doivent être placées soit dans une librairie .NCL, soit dans un événement INIT d'une fenêtre, soit dans une ressource de type Constante.
   Si une définition ne respecte pas cette règle (située dans un événement EXECUTED d'un Push-Button par exemple), le vérificateur 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.

Exemple 1 :

```
CONST PI£ 3.1415, E£ 2.718
CONST ONE% 1, ZERO% 0
CONST NAME$ 'Pierre DUPONT'
```

Exemple 2 :

```
; Exemple incorrect (sur événement INIT)
MOVE 3 TO P%
CONST HJ% 5 ; Invalid place !
```

Exemple 3 :

```
; Exemple incorrect (sur événement EXECUTED d'un Push-Button)
CONST HJ% 5 ; Definition not allowed !
MOVE 3 TO P%
```

Exemple 4 :

```
; Exemple correct (sur événement INIT de la fenêtre)
CONST HJ% 5
MOVE 3 TO P%
```

Instruction CONTINUE

Rebranche directement en début de boucle LOOP/ENDLOOP, REPEAT/UNTIL, WHILE/ENDWHILE et FOR/ENDFOR sans prendre en compte les instructions qui suivent. Une nouvelle boucle est donc immédiatement effectuée

Syntaxe	**CONTINUE**

Cette instruction n'a aucun effet en dehors des constructions LOOP/ENDLOOP, REPEAT/UNTIL, WHILE/ENDWHILE et FOR/ENDFOR

Exemple 1 :

```
MOVE 1 TO I%
REPEAT
 MOVE I%+1 TO I% ; Répété jusqu'à ce que I%=10
 IF I% <> 3
  CONTINUE
 ENDIF
 MESSAGE "Test CONTINUE", "I% vaut 3 !"
UNTIL I% = 10
```

Exemple 2 :

```
MOVE 1 TO I%
WHILE COPY$ (TEST, I%, 1) <> "-"
 MOVE I%+1 TO I% ; Répété jusqu'à ce que COPY$ <> "-"
 IF I% <> 3
  CONTINUE
 ENDIF
 MESSAGE "Test CONTINUE", "I% vaut 3 !"
ENDWHILE
```

Voir aussi LOOP, REPEAT, WHILE, FOR, BREAK

Type CONTROL

1. Ce type est utile pour pouvoir manipuler des contrôles passés en paramètre à une fonction ou instruction.
2. Le caractère spécial "@" permet d'affecter les variables de type CONTROL. Voir exemples.

Exemple 1 :

```
; ***** DECLARATION D'UNE INSTRUCTION
INSTRUCTION CHANGEENABLE CONTROL CTRL
; CHANGEENABLE change l'état du contrôle passé en paramètre (effectue un ENABLE si
inactif, DISABLE si actif)
IF ISDISABLED%(CTRL)
ENABLE CTRL
ELSE
DISABLE CTRL
ENDIF
ENDINSTRUCTION

; ***** UTILISATION DE CETTE INSTRUCTION
; BUTTON00001 étant un contrôle Push-Button de la fenêtre
ENABLE BUTTON00001
WAIT 500
CHANGEENABLE BUTTON00001 ; Effectue un DISABLE
WAIT 500
CHANGEENABLE BUTTON00001 ; Effectue un ENABLE
```

Exemple 2 :

```
; Pour affecter un contrôle existant à une variable de type CONTROL, il faut utiliser "@".
GLOBAL CONTROL CTRL
MOVE BUTTON00001 TO @CTRL
DISABLE CTRL
```

Exemple 3 :

```
; Explication de la nuance entre CTRL et @CTRL
GLOBAL CONTROL CTRL
MOVE ENTRY00001 TO @CTRL
; La "variable" CTRL vaut ENTRY00001 c'est-à-dire : agir sur CTRL est maintenant
équivalent à agir sur ENTRY00001
MOVE ENTRY00002 TO CTRL
; Le contrôle associé à la variable CTRL prend la même valeur que le contrôle ENTRY00002
; c'est-à-dire : le contenu de l'entry-field ENTRY00002 est copié dans l'entry-field
ENTRY00001
```

Voir aussi :

INT, NUM, CHAR, STRING, CSTRING, LOCAL, GLOBAL, SEGMENT, FUNCTION, INSTRUCTION

Paramétrage dynamique .KIND, Contrôle SELF

Librairie NSMisc (FIRSTCONTROL, NEXTCONTROL, CONTROLNAME$, WINDOWFROMCONTROL%)

Fonction COPY$

Syntaxe	**COPY$** *(chaîne-source, position, longueur)*

1. La partie à extraire est identifiée par sa longueur et sa position dans la chaîne source.
2. position = 1 correspond au premier caractère de la chaîne source, quel que soit le type de la chaîne (STRING ou CSTRING).

Exemple :

```
MOVE "NAT SYSTEM" TO NAT$
MOVE COPY$ (NAT$, 5, 3) TO S$ ; met "SYS" dans S$ quel que
; soit le type de S$

; Equivalent à
MOVE NAT$(4..6) TO S$ ; si S$ de type CSTRING (type par
; défaut des chaînes de caractères)

; ou équivalent à
MOVE NAT$(5..7) TO S$ ; si S$ de type STRING
```

***Voir aussi** DELETE$, FILLER$, INSERT$, POS%, MOV*

Type CSTRING

Type chaîne C utilisé au sein des instructions LOCAL, GLOBAL, SEGMENT, FUNCTION et INSTRUCTION.

1. Une chaîne CSTRING est terminée par un caractère nul. La taille interne est donc supérieure de un octet à celle précisée par LOCAL ou GLOBAL.
2. Pour une chaîne, le type STRING ou CSTRING est indifférent en NCL. Par contre il est important pour un interfaçage avec une DLL, selon que cette dernière ait été écrite en Pascal ou en C.

Voir aussi : ***INT, NUM, CHAR, STRING, CONTROL, DYNSTR, LOCAL, GLOBAL, SEGMENT, FUNCTION, INSTRUCTION***

Constante DEFRET%

Constante utilisable essentiellement dans deux cas :

- En paramètre de RETURN, afin d'appeler le traitement par défaut de l'événement en cours de traitement.
- En valeur de PARAM1% pour les événements de comportement GETVALUE et SETVALUE lorsqu'aucun index n'est employé lors du MOVE qui a généré ces événements.

Syntaxe	**DEFRET%**

Sa déclaration interne est:

```
CONST DEFRET% 32767
```

Exemple :

```
; Evénement PAINT
;
; Un bip sonore est effectué à chaque redessin de la fenêtre.
; Le RETURN DEFRET% est ajouté afin que le traitement par
; défaut de cet événement soit toujours appelé (sans ce
; RETURN, la fenêtre n'est pas repeinte, lui donnant un
; aspect "transparent").
;
BEEP
RETURN DEFRET%
```

Voir aussi PASS, RETURN, et Evénements de Comportement

Instruction DELETE

Supprime une ou plusieurs lignes dans un objet liste.

Syntaxe	**DELETE** *[AT position]* **FROM** *objet*

1. La première ligne d'une liste a pour position 0 (zéro).
2. Si AT position est mentionné, DELETE efface la ligne figurant à cette position.
   Si AT position n'est pas mentionné, DELETE efface tout le contenu du contrôle. DELETE n'efface pas le contrôle lui-même : le contrôle reste bien sûr visible et fonctionnel.
3. L'objet doit être l'un des suivants :
   a) Window classe Edit ou List
   b) List-box
   c) Combo-box
   d) CBE-Field
   e) MLE-Field
4. Ne pas confondre avec la fonction DELETE$.
5. DELETE ne permet pas de détruire un élément d'un menu déroulant. Il est plutôt conseillé de créer tous les éléments de menu au préalable, puis d'utiliser HIDE et SHOW.
6. Les contrôles List Box, Combo Box et CBE ne peuvent pas gérer plus de 32766 lignes, ni un index de lignes supérieur à 32766.

Exemple 1 :

```
; Effacement de toute la list-box LISTE
DELETE FROM LISTE
```

Exemple 2 :

```
; Effacement du premier élément de la list-box LISTE
DELETE AT 0 FROM LISTE
```

***Voir aussi** INSERT, UPDATE, NOUPDATE, LINECOUNT%, SELECTION%, ISSELECTED%*

Fonction DELETE$

Supprime une partie d'une chaîne et retourne la chaîne résultante.

Syntaxe	**DELETE$** *(chaîne-source, position, longueur)*

1. La partie à supprimer est identifiée par sa longueur et sa position dans la chaîne source.
2. position = 1 correspond au premier caractère de la chaîne source, quel que soit le type de la chaîne (STRING ou CSTRING).
3. Ne pas confondre avec l'instruction DELETE.

Exemple :

```
MOVE DELETE$ ("NAT SYSTEM", 5, 3) TO S$ ; met "NAT TEM" ; dans S$
```

***Voir aussi** COPY$, FILLER$, INSERT$, POS%*

Séparateur DESCENDING

Séparateur utilisé par les instructions INSERT et LOAD.

Syntaxe	INSERT ASCENDING | **DESCENDING** *chaîne* TO *contrôle* LOAD [ASCENDING | **DESCENDING**] *nom-fichier* TO *contrôle*

Voir aussi Instructions INSERT, LOAD

Instruction DISABLE

Rend inactif une fenêtre, un contrôle, ou un élément de menu d'une fenêtre.

Syntaxe	**DISABLE** *contrôle/handle de fenêtre*

Si le nom d'une fenêtre ne figure pas dans le nom du contrôle, l'instruction s'applique à un contrôle de la fenêtre courante.

Exemple 1 :

```
DISABLE TEST ; rend inactif le contrôle TEST dans la fenêtre courante
```

Exemple 2 :

```
DISABLE WINDOW(I%).TEST ; rend inactif le contrôle TEST dans la fenêtre dont le nom est
WINDOW et de handle I%
```

*E*xemple 3 :

```
DISABLE self% ; rend inactif la fenêtre courante et tous les contrôles , ils ne reçoivent
plus les évènements souris et clavier.
```

Note :

- Un disable d'une fenêtre ne change pas son apparence.
- Si une fenêtre a reçu la commande disable, il n'est pas possible d'activer des éléments individuellement

Voir aussi ENABLE, ISDISABLED%, LOCK, Paramétrage dynamique .DISABLED

Instruction DISPOSE

Libération d'une zone mémoire précédemment allouée par NEW.

NatStar 3.00 et NS-DK 3.00 proposent une nouvelle syntaxe pour l'instruction DISPOSE qui s'ajoute à la syntaxe existante. Elle dispose le segment pointé par @PointeurTypé.

Syntaxes	**DISPOSE** *adresse-mem* **DISPOSE** *@PointeurTypé*

Exemple :

```
SEGMENT SAMPLE
INT I
INT J
ENDSEGMENT

NEW SAMPLE, HSEG%
MOVE 1 TO SAMPLE(HSEG%).I
MOVE 2 TO SAMPLE(HSEG%).J
...
DISPOSE HSEG%
```

Voir aussi NEW, SEGMENT, les exemples 4 et 5 de l'instruction NEW

Fonction DWORD%

Génère un entier de 4 octets à partir de deux entiers.

Syntaxe	**DWORD%** *(mot-fort, mot-faible)*

La valeur retournée vaut (mot-faible modulo 65536) + (mot-fort modulo 65536) * 65536.

Exemple 1 :

```
MOVE DWORD% (2, 1) TO I% ; I% vaut 131073
```

Exemple 2 :

```
MOVE DWORD% (HIW I%, LOW I%) TO I% ; I% est inchangé quelle que soit sa valeur
```

Voir aussi ***HIW***, ***LOW***, ***WORD%***, ***PARAM12%***, ***PARAM34%***

Séparateur DYNAMIC

Précise dans une déclaration de type FUNCTION ou INSTRUCTION, qu'une fonction, située dans une DLL doit être chargée dynamiquement.

Syntaxe	*FUNCTION nom_fonct [([type] [@]paramètre1 [(taille)] [, [type] [@]paramètre2 [(taille)] [, ...]])] [RETURN type]* **DYNAMIC** *INSTRUCTION nom_instr [[type] [@]paramètre1 [(taille)] [, [type] [@]paramètre2 [(taille)] [, ...]]]* **DYNAMIC**

1. Les fonctions ou instructions déclarées en DYNAMIC se trouvent, comme celle déclarées en EXTERNAL, dans une DLL (Dynamic Link Library ou librairie dynamique). L'utilisation de DLLs permet entre autre une meilleure modularité du développement et ces modules peuvent être utilisés par plusieurs applications.
2. Contrairement aux fonctions/instructions déclarées en EXTERNAL, qui sont chargées au lancement de l'exécutable, les fonctions/instructions déclarées en DYNAMIC sont chargées et déchargées à la demande de l'application (c'est-à-dire du développeur). Ceci permet une gestion plus précise des DLLs.
3. Les fonctions/instructions d'une application utilisées occasionnellement peuvent alors être chargées uniquement lorsque cela est nécessaire, à l'entrée du module les contenant, et déchargées dès qu'elles ne sont plus utiles.
4. Comme tout chargement dynamique, l'appel d'une fonction ou d'une instruction de la DLL après le déchargement de la DLL provoque une erreur fatale de violation mémoire détectée par le système.

Exemple :

```
; Déclaration des fonction et instruction en DYNAMIC
FUNCTION MYFUNCTION%(A$, B%) DYNAMIC
INSTRUCTION MYINSTRUCTION D%, E% DYNAMIC

; Chargement de la DLL
LOADDLL "MYDLL" , HMOD%
IF HMOD% = 0
MESSAGE "Erreur" , "Chargement de la DLL impossible"
EXIT
```

```
ENDIF

; Chargement des fonctions et instructions
GETPROC HMOD%, "MY_FUNCT" , MYFUNCTION%
GETPROC HMOD%, "MY_INST" , MYINSTRUCTION
```

***Voir aussi** FUNCTION, INSTRUCTION, LOADDLL, GETPROC, UNLOADDLL,**Séparateur** EXTERNAL*

Type DYNSTR

Type chaîne de caractères dynamiques utilisé au sein des instructions LOCAL, SEGMENT, FUNCTION et INSTRUCTION. La définition de la taille de la chaîne de caractères ainsi que l'allocation du type ne sont pas nécessaires.

Le type DYNSTR permet de communiquer plus simplement avec des modules COM/DCOM ou Services Web.

Le type DYNSTR se comporte globalement comme le type CSTRING. Il sera converti automatiquement en CSTRING en cas d'affectation vers une variable de ce type.

1. Une chaîne DYNSTR ne peut être utilisée avec les instructions FILL et MOV. En effet, ces instructions nécessitent de connaître au préalable la longueur de la chaîne.
2. Une chaîne DYNSTR peut se terminer ou non par un signe $. Cependant, la variable DS ne correspond pas à la variable DS$.
3. Dans l'ensemble, les chaînes dynamiques s'utilisent comme les CSTRING. Cependant, leur taille n'est pas déterminée lors de l'écriture du code et de la compilation du programme. Une expression peut mélanger arbitrairement les quatre types de chaînes, mais certains comportements sont différents (en particulier avec les expressions comme S$(3..5)).
4. Les variables dynamiques ne permettent pas d'échanger plus de 255 caractères avec des contrôles.
5. Certains opérateurs (& et &&) et certaines fonctions et instructions standards (COPY$, DELETE$, INSERT$, SKIP, LSKIP, RSKIP, UPCASE, LOWCASE) du NCL acceptent une ou plusieurs chaînes en paramètre et retournent une nouvelle chaîne résultant de leur traitement de ces paramètres. Lorsqu'une chaîne dynamique intervient dans un ou plusieurs des paramètres de type chaîne de ces opérateurs ou fonctions, la chaîne retournée est alors elle-même une chaîne dynamique, ceci de façon à pouvoir prendre en compte des chaînes de longueur arbitraire. Cependant, on peut toujours affecter le résultat d'une expression dont le type est une chaîne dynamique à une variable d'un autre type, le résultat sera alors tronqué à la taille de la variable de réception.
6. En ce qui concerne le passage de paramètre par adresse à une fonction ou une instruction, on peut non seulement passer des variables de type DYNSTR à ce genre de paramètre, mais aussi (uniquement pour les variables simples, pas pour les tableaux) des variables (locales, globales, champs de segments ou paramètres) déclarées comme CString var{(constante entière litérale)}.

Dans ce cas, la longueur maximale du contenu de param est bien sûr de constante-entière-litérale (255 par défaut si non spécifié) caractères puisque la variable passée n'est pas une DYNSTR. La troncature lors des modifications du contenu de param (et donc de var) est alors automatique.

7. Dans le cas où l'on souhaite allouer dynamiquement un segment contenant une ou des DynStr (même indirectement, si le segment alloué contient lui-même un ou plusieurs autres segments qui contiennent des des DynStr), il faut impérativement faire un NEW (et donc libérer par DISPOSE) en utilisant l'un des trois options suivantes (en supposant l'existence de variables MonSeg@ seg et Pointer pSeg) :

```
NEW @seg
NEW MonSeg, P
NEW sizeOf MonSeg, P
```

Pour permettre à l'allocateur d'initialiser proprement le segment. Faire NEW taille%, p entrainerait des plantages, on ne peut donc pas utiliser une fonction ou instruction chargée d'allouer de la mémoire sans connaître le type du segment lors du NEW, ni utiliser un autre allocateur.

Si le segment n'a pas besoin de persister en dehors de la portée de la fonction, instruction ou événement qui l'utilise, une variable locale suffit (local MonSeg seg), il est alors inutile de faire NEW et DISPOSE, l'initialisation et la libération des DynStr dans seg sont automatiquement placées au début et à la fin du code généré.

a) Pour allouer dynamiquement un tableau de DynStr (soit parce qu'on souhaite le conserver au-delà de la portée du code correspondant, soit parce que le nombre d'éléments est variable), il faut déclarer une variable DynStr@ chaines$[] et faire NEW [nb%]@chaines$ (et donc DISPOSE @chaines$), là aussi pour garantir une bonne initialisation.

8. Il est possible d'utiliser une variable de type DYNSTR comme variable hôte dans une requête SQL (SQL_EXEC, SQL_EXECSTR, SQL_EXECCSTR). Cependant, il est totalement exclu d'utiliser des variables de type segment contenant un ou plusieurs champs de type DYNSTR (même indirectement via un champ de type segment), puisque le segment est échangé comme un bloc binaire avec la base de données, ce qui provoquerait toutes sortes de bug en cas de présence de DYNSTR.

9. L'adresse d'une variable de type DYNSTR présente peu d'intérêt, puisqu'il est interdit de l'utiliser. Donc, @ds (si l'on a déclaré DYNSTR ds) retourne à la place l'adresse du texte de la DYNSTR qui peut être considéré comme un pointeur sur une CSTRING de taille arbitrairement grande et passé comme tel. Par exemple, comme paramètre d'un SEND (PARAM12% ou PARAM34%, la

lecture par PARAM1$ ou PARAM2$ tronque la chaîne à 255 caractères dans certains cas), ou bien à une fonction prenant un POINTER sur une CSTRING. Il est cependant impératif de respecter les conditions suivantes :

a) Le texte de la DYNSTR doit absolument être considéré comme constant. Aucune modification n'est permise, même si cela ne change pas la longueur de la chaîne. En effet, plusieurs variables DYNSTR peuvent partager un même texte et la modification serait alors visible dans toutes ces variables.

b) La validité du pointeur n'est garantie que tant que la variable source n'est pas modifiée (si une seconde variable DYNSTR partageant le même texte est modifiée, cela ne change pas le texte de la première).

Exemple :

```
; Déclaration d'une variable globale
Global DynStr ds, DynStr table_ds$[3][4]
; Déclaration d'une variable locale
Local DynStr ds$, DynStr table_ds[3][4]
; Déclaration d'un paramètre de fonction ou d'instruction
MaFonction%(DynStr ds, DynStr@ ref_table_ds[][10])
; Déclaration d'un type de retour d'une fonction
Function MaFonction2$(i%) Return DynStr
; Déclaration de champs de segment
Segment MonSeg
 …
 DynStr ds$
 …
 DynStr table_ds[3][4]
 …
 DynStr table_sans_taille_ds2[][10]
EndSegment ; MonSeg
```

Voir aussi ***INT***, ***NUM***, ***CHAR***, ***STRING***, ***CONTROL***, ***CSTRING***, ***LOCAL***, ***SEGMENT***, ***FUNCTION***, ***INSTRUCTION***

Instructions ELSE, ELSEIF

ELSE est utilisé au sein de constructions IF ou EVALUATE.
ELSEIF est utilisé au sein de constructions IF.

Syntaxe	**ELSE** **ELSEIF** *condition*

Voir aussi Instructions EVALUATE, IF

Instruction ENABLE

Rend actif une fenêtre, un contrôle, ou un élément de menu.

Syntaxe	**ENABLE** *contrôle / handle fenêtre*

Si aucune fenêtre n'est spécifiée dans le nom du contrôle, l'instruction s'applique à un contrôle de la fenêtre courante.

Exemple 1 :

```
ENABLE TEST ; rend le contrôle TEST actif dans la fenêtre courante
```

Exemple 2 :

```
ENABLE WINDOW(HW%).TEST ; rend le contrôle TEST actif dans la fenêtre de nom WINDOW et de handle HW%
```

Exemple 3 :

```
ENABLE HW% ; rend active la fenêtre de handle HW%
```

Voir aussi DISABLE, ISDISABLED%, UNLOCK, Paramétrage dynamique .DISABLED

Séparateur END

Séparateur utilisé par l'instruction INSERT.

Syntaxe	INSERT AT *position*|**END** *chaîne* TO *contrôle*

Voir aussi Instruction INSERT

Instructions END*

Indicateurs de fin de constructions logique. Ces instructions sont utilisées respectivement avec : EVALUATE, FUNCTION, IF, INSTRUCTION, LOOP, SEGMENT, EVALUATE/WHERE, WHILE.

Syntaxe	**ENDVEVALUATE**
	ENDFUNCTION
	ENDIF
	ENDINSTRUCTION
	ENDLOOP
	ENDSEGMENT
	ENDWHERE
	ENDWHILE

Ces instructions provoquent une erreur de syntaxe lorsque le début de la construction a été omis (par exemple, ENDFUNCTION sans FUNCTION préalable).

Voir aussi EVALUATE, FUNCTION, IF, INSTRUCTION, LOOP, WHILE

Instruction EVALUATE

Evalue une expression et cherche son appartenance à un des intervalles précisés à l'aide des WHERE suivants (ou une égalité lorsque TO n'est pas employé).
Si un des WHERE correspond, le code NCL situé entre ce WHERE et le ENDWHERE est exécuté.
Si aucun WHERE ne correspond, le code situé entre ELSE/ENDEVALUATE est exécuté (si ELSE n'est pas spécifié, aucun code NCL n'est exécuté).

Syntaxe	**EVALUATE** *expression* **WHERE** *expression [TO expression], expression [TO expression],...* *...* **ENDWHERE** *[WHERE expression [TO expression], expression [TO expression],...* *...* *ENDWHERE* *[...]* *[ELSE* *...]* **ENDEVALUATE**

1. Les WHERE évitent une série fastidieuse de IF.
2. Les intervalles listés par les WHERE doivent être disjoints entre eux.
3. Attention à un EVALUATE effectué directement sur un contrôle. Voir l'exemple 4.

Exemple 1 :

```
EVALUATE SEXE$
WHERE "MASCULIN"
MESSAGE "Bonjour", "Monsieur"
ENDWHERE
WHERE "FEMININ"
MESSAGE "Bonjour", "Madame ou Mademoiselle"
ENDWHERE
ELSE
MESSAGE "Danger !", "Sexe " & SEXE$ & " Inconnu !"
ENDEVALUATE
```

Exemple 2 :

```
EVALUATE I%
WHERE 0
MESSAGE "I%", "Nul"
ENDWHERE
WHERE 1
MESSAGE "I%", "Un"
ENDWHERE
```

```
WHERE 2 TO 10
MESSAGE "I%", "compris entre deux et dix"
ENDWHERE
ELSE
MESSAGE "I%", "non compris entre 0 et 10"
ENDEVALUATE
```

Exemple 3 :

```
MOVE "DUPONT" TO A$
MOVE "DURAND" TO B$
EVALUATE S$
WHERE A$ TO B$
MESSAGE "S$", \
"se classe alphabétiquement entre" && A$ \
&& "et" && B$
ENDWHERE
ENDEVALUATE
```

Exemple 4 :

```
; ATTENTION ! Un EVALUATE sur un contrôle peut être trompeur!
; Ici si l'Entry-Field contient par exemple 20, l'EVALUATE considèrera qu'il est compris
entre 1 et 9 !
; Quelques explications...
; Pour NCL, un contrôle ENTRY1 est de type chaîne de caractères et donc l'EVALUATE
comparera
; des chaînes. Grâce à la conversion automatique des types, le WHERE 1 TO 9 est considéré
comme un WHERE "1" TO "9" :
; "20" est considéré supérieur à "1" et inférieur à "9" lors d'une comparaison de chaînes
(ordre alphabétique).
; Il suffit de remplacer l'EVALUATE ENTRY1 par un EVALUATE NUM ENTRY1 pour avoir le
fonctionnement espéré.
EVALUATE ENTRY1
WHERE 1 TO 9
MESSAGE "Contrôle ENTRY1", "Compris entre 1 et 9"
ENDWHERE
ELSE
MESSAGE "Contrôle ENTRY1", "Non compris entre 1 et 9"
ENDEVALUATE
```

Voir aussi IF

Instruction EXIT

Sortie de traitement de l'instruction ou de l'événement en cours.

Syntaxe	EXIT

1. A la différence de HALT, EXIT n'arrête pas l'application.
2. Lorsque le traitement d'un événement arrive à la dernière instruction, un EXIT implicite est fait automatiquement.
3. Pour les instructions, EXIT est rigoureusement équivalent à RETURN sans paramètre. Pour les fonctions, EXIT retourne zéro si la fonction est de type entier, "0.0" si elle est de type réelle, une chaîne vide si la fonction est de type chaîne, et retourne une référence NULL si la fonction est de type référence. Enfin, pour les événements, EXIT retourne zéro.

Exemple 1 :

```
; La boîte de message 2 n'est jamais affichée
MESSAGE "Test EXIT", "1. Bonjour"
EXIT
MESSAGE "Test EXIT", "2. Jamais appelé !"
```

Exemple 2 :

```
; Strictement équivalent à EXEMPLE1
; (ce MESSAGE doit être la dernière instruction
; de l'événement -> EXIT implicite)
MESSAGE "Test EXIT", "1. Bonjour"
```

Voir aussi BREAK, HALT, RETURN

Séparateur EXTERNAL

Précise, dans une déclaration de type FUNCTION ou INSTRUCTION, que la fonction ou instruction est externe, située dans une DLL.

Syntaxe

```
FUNCTION nom_fonct [( [type] [@]paramètre1 [,
  [type] [@]paramètre2 [,
  ...]]
 )] [RETURN type]
 EXTERNAL 'nomDLL.nomfonctionDLL'

INSTRUCTION nom_inst [ [type] [@]paramètre1 [,
     [type] [@]paramètre2 [,
     ...]]]
 EXTERNAL 'nomDLL.nominstructDLL'
```

Voir aussi FUNCTION, INSTRUCTION, DYNAMIC, Exemples de FUNCTION EXTERNAL et INSTRUCTION EXTERNALDEB à la fin du chapitre "Utilisation du Langage" : Ecriture d'une DLL en Microsoft C

Constante FALSE%

Valeur logique retournée par les expressions conditionnelles lorsque le résultat est faux.

Syntaxe	FALSE%

Sa déclaration interne est:

```
CONST FALSE% 0
```

Voir aussi IF, TRUE%

Instruction FILL

Remplit une zone de données avec un entier compris entre 0 et 255.

Syntaxe	**FILL** *adresse-zone, longueur-zone, entier*

1. La zone à remplir est identifiée par son adresse et sa longueur.
2. Cette instruction peut s'avérer dangereuse lorsque l'adresse et la longueur ne désignent pas une zone correcte. Ainsi l'exemple suivant a de grandes chances de provoquer un General Protection Fault sous (erreur de violation mémoire détectée par le système d'exploitation) :

```
FILL 0, 10000, 0 ; TRES MAUVAIS EXEMPLE !!!
```

Exemple 1 :

```
GLOBAL CSTRING S$
MOVE "TTTTTT" TO S$
FILL @S$, 3, ASC%("A") ; S$ vaut "AAATTT"
```

Exemple 2 :

```
SEGMENT TOTO
...
ENDSEGMENT
NEW TOTO, A%
FILL A%, SIZEOF TOTO, 0 ; met à zéro tous les
; champs du segment
```

Exemple 3 :

```
SEGMENT TITI
INT I(1) ; octet 0
STRING S1(4) ; octets 1 à 5
CSTRING S2(4) ; octets 6 à 10
INT J(1) ; octet 11
ENDSEGMENT
NEW TITI, A%

FILL A%, 1, 3 ; équivalent à MOVE 3 TO TITI(A%).I

FILL A%+1, 1, 2 ; équivalent à
FILL A%+2, 2, ASC%("A") ; MOVE "AA" TO TITI(A%).S1


FILL A%+6, 3, ASC%("B") ; équivalent à
FILL A%+9, 1, 0 ; MOVE "BBB" TO TITI(A%).S2

FILL A%+11, 1, 255 ; équivalent à MOVE -1 TO TITI(A%).J
```

***Voir aussi** MOV, FILLER$, NEW, SIZEOF*

Fonction FILLER$

Retourne une chaîne remplie avec un caractère spécifié par son code ASCII, sur une longueur donnée.

Syntaxe	**FILLER$** *(codeASCII, longueur-chaîne)*

Exemple :

```
MOVE FILLER$(ASC%("A"), 3) TO S$ ; S$ vaut "AAA"
```

Voir aussi ***FILL*****,** ***COPY$*****,** ***DELETE$*****,** ***INSERT$*****,** ***POS%***

Instruction FOR

L'instruction FOR répète la séquence d'instructions entre le FOR et le ENDFOR un nombre spécifié de fois.

Syntaxe	**FOR** *compteur = expression1* **TO** *expression2 [STEP incr_expression]* ... **ENDFOR**

1. compteur est une variable entière utilisée comme compteur de boucle :
   a) expression1 est une expression dont le résultat donne la valeur de départ du compteur,
   b) expression2 est une expression dont le résultat donne la valeur de terminasion du compteur,
   c) incr_expression est une expression dont le résultat est utilisé pour incrémenter ou décrémenter le compteur. Par défaut, l'incrémentation est de 1.
2. expression1, expression2 et incr_expression sont considérées comme des expressions entières. Attention aux conversions implicites : toute expression non entière est alors égale à 0.
3. Les instructions entre FOR et ENDFOR peuvent n'être jamais exécutées : cela se produit lorsque le signe du pas d'incrément ne permet pas d'atteindre la valeur de expression2.
4. Pour éviter une boucle infinie, la valeur 0 est à proscrire pour incr_expression.

Exemple :

```
; Affichage de 1 puis 3 puis 5
FOR I%=1 TO 5 STEP 2
MESSAGE "Valeur de I%", I%
ENDFOR

; Insertion du code ASCII des lettres A à Z dans une list box
FOR I% = ASC%("A") TO ASC%("Z")
INSERT AT END "Code ASCII de" && CHR$(I%) & "=" && I% TO LB
ENDFOR
```

Voir aussi ***BREAK*****,** ***CONTINUE*****,** ***IF*****,** ***LOOP*****,** ***REPEAT*****,** ***WHILE***

Séparateur FROM

Séparateur utilisé par les instructions DELETE, SELECT et UNSELECT.

Syntaxe	DELETE [AT *position*] **FROM** *objet* SELECT *position* **FROM** *objet* UNSELECT *position* **FROM** *objet*

Voir aussi Instructions DELETE, SELECT, UNSELECT

Instruction FUNCTION

Déclaration de fonction

A partir de la version 3.00, le champ <type> peut contenir un nom de type suivi par @. Ceci a pour effet de définir une variable pointeur sur le type mentionné.

L'intérêt de cette syntaxe est de permettre à l'analyseur syntaxique de vérifier les affectations. On ne peut plus faire pointer un pointeur d'entier vers une chaîne de caractère, ou affecter un pointeur sur un segment A à un pointeur sur un segment B, le contrôleur est aussi capable de valider la saisie des variables de segment.

Comme c'est une définition de pointeurs, il est obligatoire de les initialiser pour éviter les erreurs générales de protection.

Syntaxe

```
FUNCTION nom_fonct [( [type] [@]paramètre1 [(taille)] [,
  [type] [@]paramètre2 [(taille)] [,
  ...]]
 )] [RETURN type]
...
[RETURN valeur]
...
ENDFUNCTION

FUNCTION nom_fonct [( [type] [@]paramètre1 [(taille)] [,
  [type] [@]paramètre2 [(taille)] [,
  ...]]
 )] [RETURN type]
 EXTERNAL 'nomDLL.nomfonctionDLL' | DYNAMIC
```

1. Une fonction permet d'appeler facilement du code NCL ou toute fonction exportée d'une DLL externe. Cela permet ainsi d'enrichir le langage NCL : une fois la fonction déclarée, elle s'emploie comme toute autre fonction NCL et retourne la valeur spécifiée par RETURN valeur.
2. Par défaut, les paramètres sont passés par valeur. Si un @ précède le paramètre, le paramètre (qui doit alors être une variable) est passé par adresse, ce qui signifie que la variable passée pourra être modifiée par la fonction.
3. Si "RETURN type" n'est pas précisé, le nom de la fonction doit être terminé par le caractère d'identification du type : %, £,# ou $.

4. Il est possible de retourner une variable référencée dans "RETURN type". Cette variable ne peut pas être une variable locale. Sauf si cette variable locale est elle-même une référence à une variable non-locale (ou est allouée par l'instruction NEW). Autrement, la référence retournée pointe une variable qui n'existe plus.
5. Voir le début de ce chapitre pour les différences entre FUNCTION et INSTRUCTION.
6. EXTERNAL dll, et/ou RETURN type, doivent se situer impérativement sur la même ligne que FUNCTION. Si la ligne est trop longue, il est possible d'employer le caractère spécial de continuation de ligne "\".
7. Une fonction ne peut pas être déclarée au sein d'un événement, il faut l'inclure dans un fichier .NCL déclaré en tant que librairie. La fonction peut alors être appelée dans toute l'application.
8. Si des variables doivent être utilisées localement dans la fonction, il est judicieux de faire directement suivre FUNCTION par des déclarations de variables faites avec LOCAL. Dès le ENDFUNCTION, ces variables n'auront plus d'existence.
9. Une variable segment peut être passée par adresse (avec @), jamais par valeur).
10. Une FUNCTION peut retourner tous les types prédéfinis, sauf un contrôle ou un segment. Pour un segment (exemples 3 et 4) la solution est de passer l'adresse du segment.
11. Les définitions FUNCTION et INSTRUCTION doivent être placées dans une librairie .NCL. 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".

Exemple 1 :

```
FUNCTION CUBE£(A£)
 RETURN A£ * A£ * A£
ENDFUNCTION
MOVE CUBE£(3.1) TO A£ ; A£ vaut 29.791
```

Exemple 2 :

```
; INVERSE% retourne TRUE% lorsque A£ a été modifié en 1/A£
; INVERSE% retourne FALSE% si A% n'a pu être inversé
; le '@' est précisé lors de la définition de la fonction
FUNCTION INVERSE%(@A£)
IF A£ = 0
  MESSAGE "Erreur !", "Division par zéro."
  RETURN FALSE%
ELSE
  MOVE 1/A£ TO A£
  RETURN TRUE%
```

```
ENDIF
ENDFUNCTION

MOVE ... TO A£
; le '@' n'est pas utilisé lors de l'appel de la fonction
; le passage par adresse est implicite car il a été précisé
; lors de la définition de la fonction
MOVE INVERSE%(A£) TO J%
IF J%
MESSAGE "OK", "Inverse = " & A£
ENDIF
```

Exemple 3 :

```
SEGMENT IDENTITE
STRING NOM
STRING PRENOM
ENDSEGMENT

FUNCTION MODIFIEPRENOM% (IDENTITE @A)
MOVE "Victor" TO IDENTITE(@A).PRENOM
RETURN TRUE%
ENDFUNCTION

GLOBAL IDENTITE CL1
MOVE "HUGO" TO CL1.NOM
IF MODIFIEPRENOM%(CL1)
MESSAGE "Le prénom est  :", CL1.PRENOM
ENDIF
```

Exemple 4 :

```
SEGMENT IDENTITE
STRING NOM
STRING PRENOM
ENDSEGMENT

FUNCTION MODIFIEPRENOM% (A%)
MOVE "Jules" TO IDENTITE(A%).PRENOM
RETURN TRUE%
ENDFUNCTION

GLOBAL IDENTITE CL1
MOVE "VERNE" TO CL1.NOM
IF MODIFIEPRENOM%(@CL1)
MESSAGE "Le prénom est  :", CL1.PRENOM
ENDIF

NEW IDENTITE, CL2%
MOVE "ZOLA" TO IDENTITE(CL2%).NOM
IF MODIFIEPRENOM%(CL2%)
MESSAGE "Le prénom est  :", IDENTITE(CL2%).PRENOM
ENDIF
```

Voir aussi :

INSTRUCTION, INCLUDE, EXTERNAL, DYNAMIC

Exemples de FUNCTION EXTERNAL à la fin du Chapitre "Utilisation du Langage", Ecriture d'une DLL en Microsoft C

Fonction GETBACKCOL%

Retourne la couleur d'arrière-plan d'un objet.

Syntaxe	**GETBACKCOL%** *[(objet)]*

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

Exemple :

```
IF GETBACKCOL%(EF1) = COL_RED%
MESSAGE "EF1", "Rouge !"
ENDIF
```

Voir aussi :

SETBACKCOL*, *SETFORECOL*, *GETFORECOL%

Paramétrage dynamique .BACKCOLOR

Fonction GETCLIENTHEIGHT%

Retourne la hauteur de l'aire client d'une fenêtre.

Syntaxe	**GETCLIENTHEIGHT%** *[(handle-fenêtre)]*

1. Si aucun paramètre n'est précisé, c'est la hauteur de la fenêtre courante qui est retournée.
2. Les fonctions GETCLIENTHEIGHT% et GETCLIENTWIDTH% sont importantes pour les applications utilisant la librairie NSGraph.

***Voir aussi** GETCLIENTWIDTH%, GETHEIGHT%, GETWIDTH%, SCREENHEIGHT%, SCREENWIDTH%*

Fonction GETCLIENTWIDTH%

Retourne la largeur de l'aire client d'une fenêtre.

Syntaxe	**GETCLIENTWIDTH%** *[(handle-fenêtre)]*

1. Si aucun paramètre n'est précisé, c'est la largeur de la fenêtre courante qui est retournée.
2. Les fonctions GETCLIENTHEIGHT% et GETCLIENTWIDTH% sont importantes pour les applications utilisant la librairie NSGraph.

Voir aussi ***GETCLIENTHEIGHT%, GETHEIGHT%, GETWIDTH%, SCREENHEIGHT%, SCREENWIDTH%***

Fonction GETDATA%

Retourne le pointeur associé à une fenêtre, préalablement stocké par SETDATA.

Syntaxe	**GETDATA%** *[(handle-fenêtre)]*

1. Si aucun handle de fenêtre n'est précisé, il s'agit, par défaut, de la fenêtre courante.
2. Cette fonction permet de manipuler des données instanciées, locales à une fenêtre. Cela est important par exemple pour des applications MDI, où la même fenêtre peut être démarrée plusieurs fois : le code est unique, mais chacune des fenêtres doit manipuler des données différentes.

Exemple :

```
; Evénement INIT
; Un segment INSTDATA différent est alloué à chaque
; ouverture de cette même fenêtre. Le pointeur associé est
; ici utilisé pour sauvegarder l'adresse de ce segment.
LOCAL POINTER PDATA
SEGMENT INSTDATA
CSTRING MACHAINE
...
ENDSEGMENT
NEW INSTDATA, PDATA
SETDATA PDATA

; Evénement quelconque
; écriture d'une donnée locale à la fenêtre courante
MOVE ENTRY00001 TO INSTDATA(GETDATA%).MACHAINE

; Autre événement quelconque
; lecture d'une donnée locale à la fenêtre courante
MESSAGE "Donnée locale", INSTDATA(GETDATA%).MACHAINE

; Evénement TERMINATE
DISPOSE GETDATA%
```

Voir aussi SETDATA, LOCAL, Stockage de données

Fonction GETFORECOL%

Syntaxe	**GETFORECOL%** *[(objet)]*

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

Exemple :

```
IF GETFORECOL%(EF1) = COL_BLUE%
MESSAGE "EF1", "Bleu !"
ENDIF
```

Voir aussi :

SETBACKCOL*, *SETFORECOL*, *GETBACKCOL%

Paramétrage dynamique .FORECOLOR

Fonction GETHEIGHT%

Retourne la hauteur d'un objet.

Syntaxe	**GETHEIGHT%** *[(objet)]*

Si aucun paramètre n'est précisé, c'est la hauteur de la fenêtre courante qui est retournée.

Voir aussi :

CHANGE*, *GETWIDTH%*, *GETCLIENTHEIGHT%*, *GETCLIENTWIDTH%*, *SCREENHEIGHT%*, *SCREENWIDTH%

Paramétrage dynamique .HEIGHT

Instruction GETPROC

Charge une fonction située dans une DLL et déclarée en DYNAMIC dans une librairie NCL.

Syntaxe	**GETPROC** *handle-DLL, nomfonctionDLL, nomfonctionNCL*

1. nomfonctionDLL et nomfonctionNCL identifient complètemen la fonction ou l'instruction à charger : ce sont respectivement son nom dans la DLL et son nom dans la librairie NCL.
   Une fois chargée, elle peut ensuite être appelée comme toute fonction ou instruction NCL.
2. Le handle de la DLL est obtenu par appel préalable à LOADDLL.
3. Il est également possible de charger les fonctions/instructions d'une DLL en utilisant le numéro d'ordre dans cette DLL à la place du nom (la première fonction dans la DLL à un numéro d'ordre 1, la deuxième 2, etc...).
4. Comme tout chargement dynamique, l'appel d'une fonction ou d'une instruction après le déchargement de la DLL provoque une erreur fatale de violation mémoire détectée par le système.
5. En cas d'erreur lors du chargement, le handle retourné vaut 0. Voir exemple 2.

Exemple 1 :

```
; Déclaration des fonctions et instructions en DYNAMIC
FUNCTION MYFUNCTION%(A$, B%) DYNAMIC
INSTRUCTION MYINSTRUCTION D%, E% DYNAMIC
FUNCTION THIRDFUNCTION% DYNAMIC

; Chargement de la DLL
LOADDLL "MYDLL" , HMOD%
IF HMOD% = 0
MESSAGE "Erreur" , "Chargement de la DLL impossible"
EXIT
ENDIF
; Chargement des fonctions et instructions
GETPROC HMOD%, "MY_FUNCT" , MYFUNCTION%
GETPROC HMOD%, "MY_INST" , MYINSTRUCTION
GETPROC HMOD%, "#3" , THIRDFUNCTION%
```

Exemple 2 :

```
GETPROC h%, "fonction1", FC%
if @FC% = 0
 message "erreur", "impossible de charger la fonction"
 exit
endif
```

Voir aussi LOADDLL, UNLOADDLL, Séparateur DYNAMIC

Fonction GETPTR%

Retourne le handle du pointeur souris courant.

Syntaxe	GETPTR%

Ce handle peut, par exemple, être mémorisé pour être ensuite affecté à nouveau à l'aide de SPTR%.

Exemple :

```
LOCAL SAVEPTR%

SAVEPTR% = GETPTR%
SETPTR GETSPTR%(SPTR_WAIT%)
... ; traitement après modification du pointeur souris
SETPTR SAVEPTR%
```

Voir aussi GETSPTR%, SETPTR, SPTR_*%, CREATEPTR%, DELETEPTR (Librairie NSGraph)

Fonction GETSPTR%

Retourne un handle de pointeur souris.

Syntaxe	**GETSPTR%** *(pointeur-système)*

pointeur-système doit être l'une des constantes SPTR_*%.

Exemple :

```
; Sur événement MOUSEMOVE
SETPTR GETSPTR%(SPTR_WAIT%) ; Le curseur est changé en
; sablier
```

Voir aussi GETPTR%, SETPTR, SPTR_*%, CREATEPTR%, DELETEPTR (Librairie NSGraph)

Fonction GETTEXT$

Retourne le titre d'une fenêtre, le texte d'un menu ou de certains contrôles.

Syntaxe	**GETTEXT$** *[(objet)]*

1. Si aucun paramètre n'est précisé, c'est le titre de la fenêtre courante qui est renvoyé.
2. GETTEXT$ retourne le texte des contrôles radio-button, check-box, push-button, entry-field, static-text et group-box.
3. Pour un contrôle bitmap, GETTEXT$ retourne le handle du bitmap affiché.
4. GETTEXT$ retourne les 255 premiers caractères contenus dans une MLE. Pour certains contrôles, la fonction GETTEXT$ est équivalente à un MOVE. Voir l'exemple.

Exemple 1 :

```
; Affiche le titre de la fenêtre
MESSAGE "Le titre est : ", GETTEXT$
```

Exemple 2 :

```
; Si ENTRY00001 est un Entry-Field, l'exemple suivant
; est équivalent à faire un MOVE ENTRY00001 TO A$
MOVE GETTEXT$(ENTRY00001) TO A$
```

Voir aussi SETTEXT, Paramétrage dynamique .TEXT

Fonction GETWIDTH%

Retourne la largeur d'un objet.

Syntaxe	**GETWIDTH%** *[(objet)]*

Si aucun paramètre n'est précisé, c'est la largeur de la fenêtre courante qui est retournée.

Voir aussi :

CHANGE, GETHEIGHT%, GETCLIENTHEIGHT%, GETCLIENTWIDTH%,SCREENHEIGHT%, SCREENWIDTH%

Paramétrage dynamique .WIDTH

Fonction GETXPOS%

Retourne la position horizontale, en pixels, d'un objet.

Syntaxe	**GETXPOS%** *[(objet)]*

1. La position est celle du coin inférieur gauche de l'objet.
2. Si le paramètre objet n'est pas précisé, il s'agira de la position de la fenêtre courante.
3. Si l'objet est un contrôle, la position est relative à la fenêtre à laquelle il appartient.
4. Si l'objet est une fenêtre fille, la position est relative à sa fenêtre mère.
5. Si la fenêtre est la fenêtre principale de l'application (MAINWINDOW%), les coordonnées sont absolues (liées à l'origine de l'écran physique).

Exemple :

```
; Déplace la fenêtre de 10 pixels vers le haut et vers
; la droite
SETPOS GETXPOS% + 10, GETYPOS% + 10
```

Voir aussi GETYPOS%, SETPOS, CHANGE, Paramétrages dynamiques .X et .Y

Fonction GETYPOS%

Retourne la position verticale, en pixels, d'un objet.

Syntaxe	**GETYPOS%** *[(objet)]*

1. La position est celle du coin inférieur gauche de l'objet
2. Si objet n'est pas précisé, il s'agira de la position verticale de la fenêtre courante.
3. Si l'objet est un contrôle, la position est relative à la fenêtre à laquelle il appartient.
   Si l'objet est une fenêtre fille, la position est relative à sa fenêtre mère.
4. Si la fenêtre est la fenêtre principale de l'application (MAINWINDOW%), les coordonnées sont absolues (liées à l'origine de l'écran physique).

Exemple :

```
; Déplace la fenêtre de 10 pixels vers le haut et vers
; la droite
SETPOS GETXPOS% + 10, GETYPOS% + 10
```

Voir aussi GETXPOS%, SETPOS, CHANGE, Paramétrages dynamiques .X et .Y

Instruction GLOBAL

Définit une variable globale ou un tableau de variables globales.

A partir de la version 3.00, le champ <type> peut contenir un nom de type suivi par @. Ceci à pour effet de définir une variable pointeur sur le type mentionné.

L'intérêt de cette syntaxe est de permettre à l'analyseur syntaxique de vérifier les affectations. On ne peut plus faire pointer un pointeur d'entier vers une chaîne de caractère, ou affecter un pointeur sur un segment A à un pointeur sur un segment B, le contrôleur est aussi capable de valider la saisie des variables de segment.

Comme c'est une définition de pointeurs il est obligatoire de les initialiser pour éviter les erreurs générales de protection.

Syntaxes	**GLOBAL** *<type> nom-variable <(taille)> <[dimension] <[dimension] ...> >* ou **GLOBAL** *<type[@]>nom-variable <(taille)> <[dimension] <[dimension] ...> >*

Ici, < et > signifient un champ optionnel, alors que [et] sont réellement les caractères crochets.

1. Si type est précisé, cela force le type de la variable :
   a) INT ou INTEGER pour entier,
   b) POINTER pour pointeur,
   c) NUM pour nombre réel,
   d) STRING pour chaîne de caractères (format Pascal),
   e) CSTRING pour chaîne de caractères (format C),
   f) CHAR pour caractère simple,
   g) CONTROL pour contrôle.
2. Si type n'est pas précisé, le nom de la variable doit être terminé par le caractère d'identification du type : %, £, # ou $. Cela n'est possible que pour des variables respectivement de type INT, NUM ou STRING.
3. Type peut aussi être le nom d'un segment préalablement défini par SEGMENT. A ce moment-là, nom-variable ne peut être terminé par un caractère spécial (%, £, #, $).
4. Si un ou plusieurs [Dimension] sont précisés, il s'agit d'une déclaration d'un tableau de variables. [Dimension] est une expression entière ou une constante de type entier qui représente le nombre d'éléments du tableau. Le premier élément a un index 0.

5. Pour accéder à un élément du tableau, il faut utiliser les crochets ([ ]) comme indiqué dans les exemples suivants.
6. (taille) permet de préciser la taille de la variable ou de chacun des éléments du tableau :
7. Pour les entiers, les tailles possibles sont 1, 2, ou 4 octets (4 étant la taille par défaut). Quelle que soit la taille, les entiers sont toujours signés.
8. Pour les réels, les tailles possibles sont 4, 8, et 10 octets (8 étant la taille par défaut).
9. Pour les caractères, les tailles possibles peuvent aller de 1 à 65000 octets (1 étant la taille par défaut).
10. Pour les chaînes au format Pascal (STRING), les tailles possibles peuvent aller de 1 à 255 octets (255 étant la taille par défaut).
11. Pour les chaînes au format C (CSTRING), les tailles possibles peuvent aller de 1 à n octets (la valeur maximale de n dépendant de l'environnement: n=65000 en 32 bits, 255 étant la taille par défaut).
12. Pour les segments comme pour les contrôles, aucune taille ne peut être indiquée.
13. Il est possible d'insérer des définitions multiples de variables dans la même ligne en insérant un séparateur virgule (,) entre chaque définition.
14. Contrairement à la notation utilisée dans le reste de la documentation, les crochets [ ] de la syntaxe ne signifient pas un champ optionnel mais sont réellement les caractères [ ].
15. Une case à cocher Reference dans la boîte de dialogue Modify Description for Global permet de déclarer les variables typées.

```
GLOBAL TYPE@ VAR ;La checkbox Reference correspond au '@'.
```

16. La différence entre STRING et CSTRING se situe au niveau du stockage mémoire. Le premier octet d'un STRING contient le nombre effectif de caractères de la chaîne, le deuxième octet contient le code du premier caractère, etc... Le premier octet d'un CSTRING contient le code du premier caractère, le deuxième octet contient celui du deuxième caractère, ... et cela est terminé par un octet supplémentaire contenant la valeur 0 pour indiquer la fin de la chaîne.
17. Lorsque le champ <type> est précisé, le nom de la variable peut être quelconque, sans obligation de terminer par un % pour les entiers, £ ou # pour les réels, et $ pour les chaînes. Pour résumer :

```
GLOBAL INT TOTO ; OK
GLOBAL INT TOTO% ; OK
```

```
GLOBAL INT TOTO€ ; Non !
GLOBAL TOTO ; Non !
GLOBAL TOTO% ; OK

SEGMENT IDENTITE
...
ENDSEGMENT
GLOBAL IDENTITE TOTO ; OK
GLOBAL IDENTITE TOTO% ; Non !
```

18. Une variable globale peut aussi être déclarée dans un fichier séparé, extension .VAR, grâce à l'éditeur de variables. Cette déclaration est alors valable pour toute fenêtre de la même application.

19. Les définitions GLOBAL doivent être placées soit dans une librairie .NCL, soit dans un événement INIT d'une fenêtre, soit dans une ressource de type Variable. Si une définition ne respecte pas cette règle (située dans un événement EXECUTED d'un Push-Button 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.

20. Chaque dimension maximum théorique est 65000. En pratique, la dimension maximum est inférieure, chaque variable ne pouvant occuper plus de 64K en mémoire. Ainsi :

```
GLOBAL I%[30000]; Incorrect car 4*30000=120000 > 65000
GLOBAL I%(1)[50000]; Correct !
GLOBAL S$[50][10]; Incorrect car 256*50*10 = 128000
GLOBAL S$(3)[100][100]; Correct ! car 4*100*100=40000
```

21. Comme la taille par défaut d'une chaîne est de 256 octets, il est recommandé de préciser la taille réellement utilisée de chaque variable chaîne afin d'optimiser les allocations mémoire.

22. Lors de la génération des fichiers C en vue d'obtenir un exécutable, les variables globales NCL deviennent des variables statiques C. Dans le code C généré, ces variables ne sont pas initialisées par défaut. Bien que de nombreux compilateurs assurent cette initialisation à zéro par défaut et que le module de test initialise également ces variables à 0, nous vous conseillons de toujours initialiser vos variables globales avant leur utilisation pour être toujours maître de votre code et de ne pas risquer un comportement différent entre le test et l'application générée finale.

Exemple 1 :

```
GLOBAL A%, INT B
GLOBAL NUM N#
GLOBAL CSTRING C$(32)
GLOBAL ARRAY1%[12]
GLOBAL ARRAY2€[3], ARRAY3$[5], ARRAY4$(10)[9]
```

```
GLOBAL ARRAY5%[2][3]
; A% et B sont des entiers, N# est un nombre réel, C$ est une
; chaîne de 32 caractères maximum.

; ARRAY1% est un tableau de 12 éléments entiers. ARRAY1%[0]
; accède au premier élément. ARRAY1%[11] accède au dernier
; élément.

; ARRAY2£ est un tableau de 3 éléments réels.

; ARRAY3$ est un tableau de 5 chaînes. Par défaut une chaîne
; peut contenir 255 caractères.

; ARRAY4$ est un tableau de 9 chaînes. La taille par défaut
; (255) peut être modifiée en définissant explicitement la
; taille des éléments (dans notre exemple, c'est 10).

; ARRAY5% est un tableau d'entiers à deux dimensions.
```

Exemple 2 :

```
Segment Seg_EtatCivil
Prenom$(31)
Nom$(31)
DateDeNaissance%
CodePostal$(7)
Ville$(63)
EndSegment ; {Seg_EtatCivil}

Segment Seg_Adresse
No$(7)
NomRue$(35)
CodePostal$(7)
Ville$(63)
Pays$(15) ; unused
EndSegment ; {Seg_Adresse}

Segment SEG_Personne
Seg_EtatCivil@ EtatCivil
Seg_Adresse@ Adresse
SEG_Personne@ Suivante ; liste chaînée, remplace POINTER Suivante
EndSegment ; {SEG_Personne}

Segment s_Ville
Ville$(31)[410] ; kNbVille%
EndSegment ; {s_Ville}

Global s_Ville@ pVillePPB1
```

***Voir aussi** LOCAL, SEGMENT, SIZEOF, CONTROL, SETDATA, GETDATA%, NEW*

Instruction HALT

Arrête complètement l'exécution de l'application : les instructions qui suivent HALT ne sont jamais exécutées, ni aucune autre instruction d'aucun autre événement de l'application.

Syntaxe	**HALT** *[code-retour]*

1. L'application peut retourner un code-retour, utilisable par exemple dans des fichiers .BAT (batchs DOS) sous Windows.
2. Si code-retour n'est pas spécifié, l'application retourne 0 (HALT est strictement équivalent à HALT 0).
3. L'utilisation de HALT doit demeurer une procédure d'exception car le système ne libère pas les ressources allouées par l'application et ne décharge pas les DLLs chargées dynamiquement depuis l'application.

Voir aussi ***EXIT***

Opérateur HIB

Retourne l'octet haut d'un entier.

Syntaxe	**HIB** *entier*

1. La valeur retournée est la partie entière de entier/256.
2. Lorsque l'entier est stocké sur 2 octets, il s'agit de l'octet le plus significatif. HIB = high byte.
3. Lorsque l'entier est stocké sur un octet, HIB retourne toujours 0.
4. Lorsque l'entier est stocké sur 4 octets (et >= 65536), les 2 octets les plus significatifs sont ignorés. L'entier retourné est toujours compris entre 0 et 255, quel que soit l'entier paramètre.

Exemple :

```
MOVE HIB 10000 TO I% ; I% vaut 39
MOVE HIB 100 TO I% ; I% vaut 0
```

***Voir aussi** LOB, WORD%, HIW*

Instruction HIDE

Cache un objet.

Syntaxe	**HIDE** *[objet]*

1. Si l'objet n'est pas spécifié, le traitement s'applique à la fenêtre courante
2. L'objet existe toujours, mais est invisible. Il peut être rendu à nouveau visible avec SHOW.
3. Un HIDE sur un menu déroulant (Pulldown menu) fait disparaître la totalité de ce menu avec tous ses éléments de menu. Attention, un HIDE sur un élément de menu associé à un accélérateur clavier n'inhibe pas cet accélérateur, il faut tout d'abord faire un DISABLE de l'élément de menu. Un HIDE sur un élément de menu n'enlève que cet élément de menu.

Exemple :

```
HIDE ;Cache la fenêtre courante
HIDE TEST% ;Cache la fenêtre dont le handle est TEST%
HIDE TEST ;Cache le contrôle TEST% dans la fenêtre courante
HIDE WINDOW(I%).TEST ;Cache le contrôle TEST dans la fenêtre WINDOW ayant le handle I%
```

Voir aussi ISHIDDEN%, SHOW, Paramétrage dynamique .HIDDEN

Opérateur HIW

Retourne le mot haut d'un entier.

Syntaxe	**HIW** *entier*

1. La valeur retournée est la partie entière de entier/65536.
2. Un entier standard étant mémorisé sur quatre octets, il s'agit des deux octets les plus significatifs. HIW = high word.
3. Lorsque l'entier est stocké sur un ou deux octets (cf. LOCAL et GLOBAL), HIW retourne toujours 0.

Exemple :

```
MOVE HIW 500000 TO I% ; I% vaut 7
; (50000 s'écrit 7A210 en hexa)
MOVE HIW 10000 TO I% ; I% vaut 0
```

Voir aussi LOW, DWORD%, HIB

Instruction IF

Une expression « if » démarre avec le mot clé IF suivi par une expression logique appelée la condition et finit avec le mot clé ENDIF. Les lignes entre IF et ENDIF contiennent des instructions que l'interpréteur exécute seulement si la condition est évaluée à TRUE%.
Les mots clés optionnels ELSE et ELSEIF traitent les instructions dans le cas où la condition est évaluée à FALSE%.

Syntaxe	**IF** *condition* *....* *[ELSEIF* *condition* *....]* *[ELSE* *....]* **ENDIF**

1. En fait, les instructions situées derrière le IF sont exécutées dès que la condition est différente de zéro, et non pas uniquement quand la condition vaut TRUE% (=1).
2. Attention aux IF effectués directement sur des contrôles. En effet, ces derniers ne sont pas considérés comme des nombres mais comme des chaînes de caractère. Voir l'exemple 4 de l'instruction EVALUATE.

Exemple 1 :

```
IF SKIP UPCASE TEST = "PETER"
 MOVE "Hello Peter" TO MSG ;S'exécute si la
;condition est TRUE%
ELSEIF SKIP UPCASE TEST = "PAUL"
 MOVE "Hello Paul" TO MSG
ELSEIF SKIP UPCASE TEST = "JEAN"
 MOVE "Hello Jean" TO MSG
ELSE
 MOVE "!!" TO MSG ;S'exécute si la
;condition est FALSE%
ENDIF
```

Exemple 2 :

```
IF 15 ; 15 étant différent de zéro
BEEP ; le bip sera toujours effectué
ENDIF
Exemple 3
; ATTENTION ! Un IF sur un contrôle peut être trompeur !
; Ici si l'ENTRY00001 contient par exemple 20, et si
; l'ENTRY00002 contient 9, le IF considérera que l'ENTRY00002
; est supérieur à l'ENTRY00001!
; Quelques explications... Pour NCL, un contrôle est
```

```
; de type chaîne de caractères et donc le IF comparera des
; chaînes.
; "20" est considéré supérieur à "1" et inférieur à "9" lors
; d'une comparaison de chaînes (ordre alphabétique).
; Il suffit de remplacer le IF ENTRY00002 > ENTRY00001 par un
; IF NUM ENTRY00002 > NUM ENTRY00001 pour avoir le
; fonctionnement espéré.
IF ENTRY00002 > ENTRY00001
MESSAGE "Contrôle ENTRY00002", "Supérieur à ENTRY00001"
ENDIF
```

Voir aussi EVALUATE, REPEAT, WHILE

Instruction INCLUDE

Permet d'inclure du code source NCL situé dans le fichier texte nom-fichier.

Syntaxe	**INCLUDE** *nom-fichier*

Pour inclure du code natif C, il suffit d'employer le caractère spécial "%". Ce code natif, contrairement au code NCL inclus par INCLUDE, ne pourra pas être testé avec NSTEST.EXE. Attention, le code C est inclus tel quel dans une fonction C correspondant au traitement d'un événement, par conséquent il ne doit pas comprendre de déclaration de fonction 'main'.

Exemple 1 :

```
; Inclusion de code NCL
INCLUDE "MYCODE.NCL"
```

Exemple 2 :

```
; Inclusion de source natif
%
{
/* Ceci est du source C */
#include <mycode.c> /* Inclusion de fichier source C */
/* Fin du source C */
}
%
```

Voir aussi Utilisation des librairies NCL FUNCTION, INSTRUCTION

Constante INDETERMINATE%

Valeur d'un contrôle check-box lorsqu'il est grisé (3 states uniquement).

Syntaxe	INDETERMINATE%

Sa déclaration interne est:

```
CONST INDETERMINATE% 2
```

Exemple :

```
; Grise le contrôle check-box CHECKBOX00001
MOVE INDETERMINATE% TO CHECKBOX00001
```

Voir aussi Constantes CHECKED%, UNCHECKED%

Instruction INSERT

Insère une chaîne dans un objet de type liste. L'insertion est effectuée soit par ordre alphabétique, soit à une position donnée.

Syntaxe	**INSERT ASCENDING** | **DESCENDING** *chaîne* **TO** *objet* **INSERT AT** *position* | **END** *chaîne* **TO** *objet*

1. Avec "ASCENDING", la chaîne est insérée de façon à obtenir la liste par ordre alphabétique normal (A->Z).
   a) Avec "DESCENDING", la chaîne est insérée de façon à obtenir la liste par ordre alphabétique inverse (Z->A).
   b) Avec "AT position", la chaîne est insérée à la position indiquée (position=0 signifiant que la chaîne est insérée en tête de la liste).
   c) Avec "AT END", la chaîne est insérée en fin de liste : elle se retrouve donc en dernier élément de la liste.
2. L'objet doit être l'un des suivants :
   a) Window classe Edit ou List
   b) List-box
   c) Combo-box
   d) CBE-Field
   e) MLE-Field
3. ASCENDING et DESCENDING ne sont pas valables pour les MLE-fields, ni pour les fenêtres de classe Edit.
4. Ne pas confondre avec la fonction INSERT$.
5. INSERT ne permet pas d'insérer un nouvel élément dans un menu déroulant. Il est plutôt conseillé de créer tous les éléments de menu au préalable, puis d'utiliser HIDE et SHOW.
6. INSERT permet d'insérer un descripteur de présentation dans une List Box ou une Combo Box. Pour plus d'information sur cette utilisation de INSERT, reportez-vous à l'annexe E.
7. Les contrôles List Box, Combo Box et CBE ne peuvent pas gérer plus de 32766 lignes, ni un index de lignes supérieur à 32766.

Exemple 1 :

```
; Insertion de "Jean" en première ligne
; dans la list-box LISTE
INSERT AT 0 "Jean" TO LISTE

; Ajout de "Jean" en dernière ligne
; dans la list-box LISTE
```

```
INSERT AT END "Jean" TO LISTE

; idem mais dans l'aire client d'une fenêtre de classe List
INSERT AT END "Jean" TO SCREEN(HANDLE%).CLIENT

; Insertion de "Jean" avant la ligne sélectionnée
; dans la list-box LISTE
INSERT AT SELECTION%(LISTE) "Jean" TO LISTE
```

Exemple 2 :

```
; Remplissage de 30 lignes affichant les 30 premiers
; entiers  : 1, 2, 3, 4, ..., 29, 30.
DELETE FROM LISTE
NOUPDATE LISTE
MOVE 0 TO I%
REPEAT
MOVE I%+1 TO I%
INSERT AT END I% TO LISTE
UNTIL I% = 30
UPDATE LISTE
```

***Voir aussi** DELETE, UPDATE, NOUPDATE, LINECOUNT%, SELECTION%, ISSELECTED%, Descripteurs de présentation*

Fonction INSERT$

Insère une chaîne dans une autre chaîne à une position donnée et retourne la chaîne résultante.

Syntaxe	**INSERT$** *(chaîne-source, chaîne-destination, position-insertion [, nb-car-chaîne-dest-suppr])*

1. position = 1 correspond au premier caractère de la chaîne destination, quel que soit le type de la chaîne (STRING ou CSTRING).
2. Le paramètre optionnel nb-car-chaîne-dest-suppr permet de définir le nombre de caractères à supprimer dans la chaîne destination lors de l'insertion. Cette option permet essentiellement de remplacer un certain nombre de caractères dans une chaîne.
3. Ne pas confondre avec l'instruction INSERT.

Exemple :

```
MOVE INSERT$ ("SYS", "NAT TEM", 5) TO S$
; S$ vaut "NAT SYSTEM" quel que
; soit son type

MOVE "at" TO S$(1..2) ; S$ vaut "Nat SYSTEM" si S$ est
; de type CSTRING (type par défaut)
; ou vaut "atT SYSTEM" si S$ est
; de type STRING.
```

***Voir aussi** COPY$, DELETE$, FILLER$, POS%*

Instruction INSTRUCTION

Déclaration d'une instruction.

A partir de la version 3.00, le champ <type> peut contenir un nom de type suivi par @. Ceci à pour effet de définir une variable pointeur sur le type mentionné.

L'intérêt de cette syntaxe est de permettre à l'analyseur syntaxique de vérifier les affectations. On ne peut plus faire pointer un pointeur d'entier vers une chaîne de caractère, ou affecter un pointeur sur un segment A à un pointeur sur un segment B, le contrôleur est aussi capable de valider la saisie des variables de segment .

Comme c'est une définition de pointeurs il est obligatoire de les initialiser pour éviter les erreurs générales de protection.

Syntaxes

INSTRUCTION *nom_inst [([type] [@]paramètre1 [(taille)] [, [type] [@]paramètre2 [(taille)] [, ...]]]*
...
[RETURN | EXIT]
...
ENDINSTRUCTION

INSTRUCTION *nominst [([type] [@]paramètre1 [(taille)] [, [type] [@]paramètre2 [(taille)] [, ...]]]*
EXTERNAL 'nomDLL.nominstDLL' | DYNAMIC

INSTRUCTION instrname [[type[@]] [@]parameter1 [(size)] [, [type[@]] [@]parameter2 [(size)] [, ...]]]
ENDINSTRUCTION

1. Une instruction permet d'appeler facilement du code NCL ou toute instruction exportée d'une DLL externe. Cela permet ainsi d'enrichir le langage NCL : une fois l'instruction déclarée, elle s'emploie comme toute autre instruction NCL.
2. Par défaut, les paramètres sont passés par valeur. Si un @ précède le paramètre, le paramètre (qui doit alors être une variable) est passé par adresse, ce qui signifie que la variable passée pourra être modifiée par l'instruction.

3. Voir le début de ce chapitre pour les différences entre FUNCTION et INSTRUCTION.

4. EXTERNAL dll doit se situer impérativement sur la même ligne que INSTRUCTION. Si la ligne est trop longue, il est possible d'employer le caractère spécial de continuation de ligne "\".

5. Une instruction ne peut pas être déclarée au sein d'un événement, il faut donc l'inclure dans un fichier .NCL déclaré en tant que librairie. Cette instruction peut être alors appelée dans toute l'application.

6. Si des variables doivent être utilisées localement dans l'instruction, il est judicieux de faire directement suivre INSTRUCTION par des déclarations de variables faites avec LOCAL. Dès le ENDINSTRUCTION, ces variables n'auront plus d'existence.

7. Une variable segment peut être passée par adresse (avec @), jamais par valeur.

8. RETURN ou EXIT permettent de sortir de l'instruction avant ENDINSTRUCTION.

9. Les définitions FUNCTION et INSTRUCTION doivent être placées dans une librairie .NCL. 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".

Exemple 1 :

```
INSTRUCTION MSGERREUR S$
BEEP 2
MESSAGE "Erreur fenêtre.SCR !", S$
ENDINSTRUCTION

MSGERREUR "ATTENTION, DANGER"
```

Exemple 2 :

```
; '@' utilisé lors de la définition de l'instruction
; indique que les variables sont passées par adresse
INSTRUCTION EXCHANGE @I%, @J%
LOCAL TEMP%
MOVE I% TO TEMP%
MOVE J% TO I%
MOVE TEMP% TO J%
ENDINSTRUCTION

MOVE 3 TO A% ; A% vaut 3
MOVE 4 TO B% ; B% vaut 4
; le passage des variables par adresse est implicite
; car indiqué lors de la définition de l'instruction
; '@' n'est donc pas utilisé lors de l'appel
EXCHANGE A%, B% ; maintenant A% vaut 4
; et B% vaut 3
```

Exemple 3 :

```
SEGMENT IDENTITE
STRING NOM
STRING PRENOM
ENDSEGMENT

INSTRUCTION AFFICHEPRENOM IDENTITE @A
MESSAGE "Le prénom est", IDENTITE(@A).PRENOM
ENDINSTRUCTION

GLOBAL IDENTITE CL1
MOVE "CITROEN" TO CL1.NOM
MOVE "André" TO CL1.PRENOM
AFFICHEPRENOM CL1
```

Exemple 4 :

```
SEGMENT IDENTITE
STRING NOM
STRING PRENOM
ENDSEGMENT

INSTRUCTION AFFICHEPRENOM A%
MESSAGE "Le prénom est", IDENTITE(A%).PRENOM
ENDINSTRUCTION

GLOBAL IDENTITE CL1
MOVE "MAGNE" TO CL1.NOM
MOVE "Charles" TO CL1.PRENOM

AFFICHEPRENOM @CL1

NEW IDENTITE, CL2%
MOVE "MARTIN" TO IDENTITE(CL2%).NOM
MOVE "Jean" TO IDENTITE(CL2%).PRENOM

AFFICHEPRENOM CL2%
```

Exemple 5 :

```
INSTRUCTION PASSETABLEAU @TAB%[20]
MOVE 12 TO TAB%[3]
ENDINSTRUCTION

LOCAL MONTAB%[20]
PASSETABLEAU MONTAB%
MESSAGE "Vérif", MONTAB%[3] ; affiche 12
```

Exemple 6 :

```
Segment New_SEG_Personne
Prenom$(31)
Nom$(31)
Age%
Adresse$
New_SEG_Personne@ Suivante ; liste chaînée, remplace POINTER Suivante
EndSegment ; {New_SEG_Personne}

Instruction NewEcritTout Control LB, New_SEG_Personne@ Personne
```

```
Local New_SEG_Personne@ TmpPers

@TmpPers = Personne
While @TmpPers <> 0
Insert At End \
TmpPers.Prenom$ & #9 & \
TmpPers.Nom$ & #9 & \
TmpPers.Age% & #9 & \
TmpPers.Adresse$ \
To LB
@TmpPers = TmpPers.Suivante ; type vérifié à la compilation NSGEN
EndWhile
EndInstruction ; {NewEcritTout}
```

Voir aussi :

FUNCTION, INCLUDE, Séparateur DYNAMIC, GETPROC

Exemple de INSTRUCTION EXTERNAL à la fin du Chapitre "Utilisation du Langage" : Ecriture d'une DLL en Microsoft C

Opérateur INT

Convertit en entier.

Syntaxe	**INT** *entier*|*réel*|*chaîne*

1. Equivalent à la fonction TRUNC% lorsque l'opérande est réel
2. INT peut aussi être utilisé en tant que "type" au sein des instructions LOCAL, GLOBAL, SEGMENT, FUNCTION et INSTRUCTION.

Exemple :

```
MOVE INT 1.3 TO I% ; I% vaut 1
MOVE INT "12" TO I% ; I% vaut 12
```

Voir aussi ***NUM, STRING, ISINT%, ROUND%, TRUNC%, DYNSTR, LOCAL, GLOBAL, SEGMENT, FUNCTION, INSTRUCTION***

Type INT

Type entier, utilisé au sein des instructions LOCAL, GLOBAL, SEGMENT, FUNCTION et INSTRUCTION.

1. Les tailles possibles pour une variable de type INT sont 1, 2 ou 4 octets (4 étant la taille par défaut). Quelle que soit la taille, les entiers sont toujours signés.
2. La taille d'un entier influe sur les nombres minimum et maximum acceptés :

Taille	Minimum et maximum acceptés
1	[-127, +127]
2	[-32767, + 32767]
4	[-2147483647, + 2147483647]

3. Préférez le type INTEGER pour les applications devant être générées sous différents systèmes.

***Voir aussi** INTEGER, LOCAL, GLOBAL, SEGMENT, FUNCTION, INSTRUCTION*

Type INTEGER

Type entier en taille native, utilisé au sein des instructions LOCAL, GLOBAL, SEGMENT, FUNCTION et INSTRUCTION.

1. A la différence du type INT, aucune taille ne doit être spécifiée après le type INTEGER : celle-ci est automatiquement adaptée au système. Ainsi, une variable de type INTEGER est générée sous forme de 4 octets pour un système 32 bits.
2. Utiliser le type INTEGER assure une meilleure portabilité pour les applications devant être générées sous différents systèmes.

***Voir aussi** INT, LOCAL, GLOBAL, SEGMENT, FUNCTION, INSTRUCTION*

Instruction INVALIDATE

Invalide une fenêtre : son contenu doit être redessiné.

Syntaxe	**INVALIDATE** *[handle-fenêtre]*

1. Si aucun paramètre n'est spécifié, invalide la fenêtre courante.
2. Cette instruction a pour unique effet de faire générer un événement PAINT vers la fenêtre. Toutefois, contrairement à un SEND PAINT, l'événement PAINT sera traité APRES le traitement de l'événement courant.

Exemple :

```
INVALIDATE ;invalide la fenêtre courante
INVALIDATE HW% ;invalide la fenêtre dont le handle
;vaut HW%
```

Voir aussi Evénement PAINT

Fonction ISDISABLED%

Teste si une fenêtre, un contrôle ou un élément de menu est inactif.

Syntaxe	**ISDISABLED%** *(contrôle/handle fenêtre)*

La fonction rend TRUE% si la fenêtre ou le contrôle est inactif, FALSE% s'il est actif.

Voir aussi ENABLE, DISABLE, Paramétrage dynamique .DISABLED

Fonction ISHIDDEN%

Teste si un objet caché.

Syntaxe	**ISHIDDEN%** *[(objet)]*

1. La fonction rend TRUE% si l'objet est caché, FALSE% s'il n'est pas caché.
2. Si aucun objet n'est précisé, c'est le statut de la fenêtre courante qui est retourné.

Voir aussi HIDE, SHOW, Paramétrage dynamique .HIDDEN

Fonction ISINT%

Teste si une chaîne est convertible en un entier.

Syntaxe	**ISINT%** *(chaîne)*

La fonction rend TRUE% si la chaîne est convertible en un entier, FALSE% si elle ne l'est pas.

Exemple 1 :

```
MOVE ISINT%("3R") TO I% ; I% vaut FALSE%
MOVE ISINT%("4521") TO I% ; I% vaut TRUE%
MOVE ISINT%("4521.3E+04") TO I% ; I% vaut FALSE%
```

Exemple 2 :

```
IF ISINT%(S$)
MOVE INT S$ TO I%
ELSE
MESSAGE "Attention", "Impossible de convertir S$ en entier"
ENDIF
```

Voir aussi INT, ISNUM%

Fonction ISLOCKED%

Teste si un objet est bloqué.

Syntaxe	**ISLOCKED%** *[(contrôle)]*

1. La fonction rend TRUE% si l'objet est bloqué, FALSE% s'il ne l'est pas.
2. Si aucun objet n'est précisé, c'est le statut de la zone CLIENT de la fenêtre courante (si elle est de classe Edit ou List) qui est retourné.

Exemple :

```
IF ISLOCKED%(EF_NOM)
MESSAGE "Statut de EF_NOM", "Bloqué !"
ENDIF
```

Voir aussi LOCK, UNLOCK, Paramétrage dynamique .LOCKED

Fonction ISMAXIMIZED%

Teste si une fenêtre est maximisée.

Syntaxe	**ISMAXIMIZED%** *[(handle-fenêtre)]*

1. La fonction rend TRUE% si la fenêtre est maximisée, FALSE% si elle ne l'est pas.
2. Si aucun handle n'est précisé, c'est le statut de la fenêtre courante qui est retourné.

***Voir aussi** MINIMIZE, MAXIMIZE, RESTORE, ISMINIMIZED%*

Fonction ISMINIMIZED%

Teste si une fenêtre est minimisée.

Syntaxe	**ISMINIMIZED%** *[(handle-fenêtre)]*

1. La fonction rend TRUE% si la fenêtre est minimisée, FALSE% si elle ne l'est pas.
2. Si aucun handle n'est précisé, c'est le statut de la fenêtre courante qui est retourné.

***Voir aussi** MINIMIZE, MAXIMIZE, RESTORE, ISMAXIMIZED%*

Fonction ISMOUSE%

Teste la présence d'une souris.

Syntaxe	ISMOUSE%

La fonction rend TRUE% si une souris a été détectée, FALSE% si aucune souris n'est présente.

Voir aussi NBBUTTONS%

Fonction ISNUM%

Teste si une chaîne est convertible en réel.

Syntaxe	**ISNUM%** *(chaîne)*

La fonction rend TRUE% si la chaîne est convertible en un réel, FALSE% si elle ne l'est pas.

Exemple 1 :

```
MOVE ISNUM%("3R") TO I% ; I% vaut FALSE%
MOVE ISNUM%("4521") TO I% ; I% vaut TRUE%
MOVE ISNUM%("4521.3E+04") TO I% ; I% vaut TRUE%
```

Exemple 2 :

```
IF ISNUM%(S$)
MOVE NUM S$ TO N£
ELSE
MESSAGE "Attention", \
"Impossible de convertir" && S$ && "en réel"
ENDIF
```

Voir aussi NUM, ISINT%

Fonction ISSELECTED%

Teste si une ligne d'un objet de type liste est sélectionné.

Syntaxe	**ISSELECTED%** *(objet, position)*

1. La ligne est identifiée par sa position. position = 0 spécifie la première ligne de la liste.
2. La fonction rend TRUE% si la ligne indiquée est sélectionnée, FALSE% si elle ne l'est pas.
3. L'objet doit être un des suivants :
   a) Window classe List
   b) List-box
   c) Combo-box
   d) CBE-Field
4. Cette fonction est surtout utile dans le cas de list-box à sélection multiple.
5. Dans le cas d'une MLE ou d'une fenêtre de classe Edit, cette fonction indique si le curseur est à la ligne position.

Exemple :

```
IF ISSELECTED% (LISTBOX00001, 3)
MESSAGE "List-box", "La ligne numéro 3 est sélectionnée"
ENDIF
```

Voir aussi LINECOUNT%, SELECT, SELECTION%, UNSELECT, événement SELECTED

Constantes JUST_*%

Utilisées pour le paramétrage dynamique .JUSTIFICATION d'un contrôle.

Syntaxe	**JUST_LEFT%** **JUST_CENTER%** **JUST_RIGHT%**

Signification des constantes :

- JUST_LEFT% Le texte est calé à gauche
- JUST_CENTER% Le texte est centré
- JUST_RIGHT% Le texte est calé à droite

Leur déclaration interne est :

```
CONST JUST_LEFT% 0
CONST JUST_CENTER% 1
CONST JUST_RIGHT% 2
```

Exemple

```
; Centrer le texte de l'entry field ENTRY00001
MOVE JUST_CENTER% TO ENTRY00001.JUSTIFICATION
```

Voir aussi Paramètre dynamique .JUSTIFICATION

Constantes KF_*%

Masques correspondants à l'état des touches du clavier, passées dans le PARAM1% de l'événement CHARACTER.

Syntaxe	**KF_ALT%** **KF_CHAR%** **KF_CTRL%** **KF_KEYUP%** **KF_LONEKEY%** **KF_PREVDOWN%** **KF_SHIFT%** **KF_VIRTUALKEY%**

1. Reportez-vous à l'événement CHARACTER pour une description complète de chacune de ces constantes.
2. Leur déclaration interne est :

```
CONST KF_CHAR% 1 ; 2#00000001
CONST KF_VIRTUALKEY% 2 ; 2#00000010
CONST KF_KEYUP% 4 ; 2#00000100
CONST KF_PREVDOWN% 8 ; 2#00001000
CONST KF_LONEKEY% 16 ; 2#00010000
CONST KF_SHIFT% 32 ; 2#00100000
CONST KF_ALT% 64 ; 2#01000000
CONST KF_CTRL% 128 ; 2#10000000
```

3. Les constantes KF_*% sont toutes des puissances de 2. C'est dans le but de pouvoir stocker simultanément plusieurs de ces constantes dans le même paramétre de l'événement CHARACTER. Ainsi, si PARAM1% de CHARACTER vaut 161 (ce qui s'écrit en binaire 2#101000001), cela signifie que KF_CHAR% + KF_SHIFT% + KF_CTRL% sont positionnés : une touche a donc été enfoncée, tout en appuyant sur [Shift] et [Ctrl].

Voir aussi Evénement CHARACTER, Constantes VK_*%, Opérateurs binaires BAND, BOR, BNOT , BXOR

Opérateur LENGTH

Syntaxe	**LENGTH** *chaîne*

Ne pas confondre SIZEOF (taille mémoire occupée en octets par une variable) et LENGTH (nombre de caractères d'une chaîne).

Exemple 1 :

```
LOCAL S$
MOVE "Longueur" TO S$
MESSAGE "SizeOf =" && SIZEOF S$, "Length =" && LENGTH S$
; Affiche "SizeOf = 256" et "Length = 8"
```

Exemple 2 :

```
MOVE LENGTH "Combien ?" TO I% ; I% vaut 9
```

Voir aussi SIZEOF

Fonction LINECOUNT%

Retourne le nombre :

- de lignes de la fenêtre courante, si elle est de classe Edit ou List,
- de lignes, pour un contrôle de type liste,
- de fenêtres à onglet, pour un contrôle Classeur.

Syntaxe	**LINECOUNT%** *[(objet)]*

1. Pour une fenêtre courante de classe Edit ou List, le paramètre objet est facultatif. S'il n'est pas spécifié, alors la fonction retourne le nombre de lignes de la fenêtre courante.
2. Pour un contrôle de type liste (List-box, Combo-box, CBE-Field et MLE-Field), le paramètre objet est obligatoire.
3. Pour un contrôle Classeur, Le paramètre objet est obligatoire.
4. La fonction LINECOUNT% est limité à 32767 caractères. Une fois cette limite dépassée, elle retourne une valeur négative.

Exemple 1 :

```
; Pour un contrôle de type liste :
; Efface le dernier élément de la list-box LISTE
DELETE AT LINECOUNT%(LISTE)-1 FROM LISTE
```

Exemple 2 :

```
; Pour un contrôle Classeur  :
; Le contrôle Classeur CC_TAB contient n fenêtres
; à onglet,toutes rattachées à la même fenêtre DTAB1.
; Ce code affecte à l'entry field ENTRY de DTAB1
; un numéro correspondant au rang de la fenêtre
; à onglet dans le contrôle Classeur.

Local H% (4)
Local I% (1)

I%=0

FOR I% TO LINECOUNT%(CC_TAB)-1
 H%=CC_TAB[I]
 DTAB1(H%).ENTRY=I%
ENDFOR
```

Voir aussi pour un contrôle de type Liste : ***INSERT***, ***DELETE***, ***SELECT***, ***SELECTION%***, ***ISSELECTED%***
Voir aussi pour un contrôle Classeur : ***SELECT***, ***SELECTION%***

Instruction LOAD

Charge un fichier texte dans un objet de type liste.

Syntaxe	**LOAD** [**ASCENDING** | **DESCENDING**] *nom-fichier TO objet*

1. Le nom du fichier passé à l'instruction LOAD est limitée à 254 caractères.
2. Si ASCENDING ou DESCENDING est spécifié, les lignes du fichier sont classées par ordre alphabétique normal (A->Z) ou inverse (Z->A) dans l'objet.
3. L'objet doit être l'un des suivants :
   a) Window classe Edit ou List
   b) List-box
   c) Combo-box
   d) CBE-Field
   e) MLE-Field
4. ASCENDING et DESCENDING ne sont pas valables pour les MLE-fields, ni pour les fenêtres de classe Edit.
5. Le changement de ligne dans les listes a lieu sur chaque ensemble CR-LF du fichier texte.
6. LOAD écrase le contenu d'un contrôle de type MLE non vide par le fichier nouvellement chargé. Pour un contrôle d'un autre type non vide, LOAD rajoute le fichier chargé à la fin de la liste.
7. Des parenthèses utilisées au sein de la chaîne nom-fichier permettent d'utiliser les variables d'environnement. Voir exemple.

Exemple :

```
; Remplit la list-box LISTE avec le fichier SAMPLE.TXT
LOAD "SAMPLE.TXT" TO LISTE

; Charge le fichier SAMPLE.TXT situé dans le répertoire
; spécifié par la variable d'environnement SET NS-LST
; (ces variables peuvent être définies dans
; AUTOEXEC.BAT sous Windows)
LOAD "(NS-LST)SAMPLE.TXT" TO LISTE
```

Voir aussi SAVE et Librairie NSMisc

Instruction LOADDLL

Charge dynamiquement une DLL et fournit son handle.

Syntaxe	**LOADDLL** *nomDLL, handle-DLL*

1. Le handle, de type pointer, est retourné dans la variable handle-DLL passé en paramètre.
2. Les fonctions et instructions de cette DLL peuvent ensuite être chargées grâce à l'instruction GETPROC.
3. En cas d'erreur lors du chargement, le handle retourné vaut 0.

Exemple :

```
GLOBAL pointer HMOD
; Chargement de la DLL MYDLL
LOADDLL "MYDLL" , HMOD
IF HMOD = 0
  MESSAGE "Erreur" , "Chargement de la DLL impossible"
  EXIT
ENDIF
```

Voir aussi GETPROC, UNLOADDLL, Séparateur DYNAMIC

Opérateur LOB

Retourne l'octet bas d'un entier.

Syntaxe	**LOB** *entier*

1. La valeur retournée est le reste de la division entier/256
2. Lorsque l'entier est stocké sur 2 ou 4 octets, il s'agit de l'octet le moins significatif (LOB = low byte).
   Lorsque l'entier est stocké sur un octet, LOB retourne toujours intégralement l'entier.
3. L'entier retourné est toujours compris entre 0 et 255, quel que soit l'entier paramètre.

Exemple :

```
MOVE LOB 10000 TO I% ; I% vaut 16
MOVE LOB 100 TO I% ; I% vaut 100
```

Voir aussi HIB, WORD%, LOW

Instruction LOCAL

Définit une variable locale ou un tableau de variables locales.

A partir de la version 3.00, le champ <type> peut contenir un nom de type suivi par @. Ceci à pour effet de définir une variable pointeur sur le type mentionné.

L'intérêt de cette syntaxe est de permettre à l'analyseur syntaxique de vérifier les affectations. On ne peut plus faire pointer un pointeur d'entier vers une chaîne de caractère, ou affecter un pointeur sur un segment A à un pointeur sur un segment B, le contrôleur est aussi capable de valider la saisie des variables de segment .

Comme c'est une définition de pointeurs il est obligatoire de les initialiser pour éviter les erreurs générales de protection.

Syntaxes	**LOCAL** *<type> nom-variable <(taille)> <[dimension] <[dimension] ...> >* ou **LOCAL** *<type[@]>nom-variable <(taille)> <[dimension] <[dimension] ...>* >*

Ici, < et > signifient un champ optionnel, alors que [et] sont réellement les caractères crochets.

1. Une telle variable ne pourra être utilisée que localement, c'est-à-dire au sein de l'événement, de la fonction, ou de l'instruction sur lequel est fait la déclaration. Une autre variable locale de même nom située sur un autre événement, ou dans du code externe à la fonction ou à l'instruction, aura une existence complètement séparée.
2. Si type est précisé, cela force le type de la variable :
   a) INT ou INTEGER pour entier,
   b) POINTER pour pointeur,
   c) NUM pour nombre réel,
   d) STRING pour chaîne de caractères (format Pascal),
   e) CSTRING pour chaîne de caractères (format C),
   f) CHAR pour caractère simple,
   g) CONTROL pour contrôle.
3. Si type n'est pas précisé, le nom de la variable doit être terminé par le caractère d'identification du type : %, £, # ou $. Cela n'est possible que pour des variables respectivement de type INT, NUM ou STRING.

4. Type peut aussi être le nom d'un segment préalablement défini par SEGMENT. A ce moment-là, nom-variable ne peut être terminé par un caractère spécial (%, £, #, $).

5. Si un ou plusieurs [Dimension] sont précisés, il s'agit d'une déclaration d'un tableau de variables. [Dimension] est une expression entière ou une constante de type entier qui représente le nombre d'éléments du tableau. Le premier élément a un index 0.

6. Pour accéder à un élément du tableau, il faut utiliser les crochets ([]) comme indiqué dans les exemples suivants.

7. (taille) permet de préciser la taille de la variable ou de chacun des éléments du tableau :

8. Pour les entiers, les tailles possibles sont 1, 2, ou 4 octets (4 étant la taille par défaut). Quelle que soit la taille, les entiers sont toujours signés.

9. Pour les réels, les tailles possibles sont 4, 8, et 10 octets (8 étant la taille par défaut).

10. Pour les chaînes au format Pascal (STRING), les tailles possibles peuvent aller de 1 à 255 octets (255 étant la taille par défaut).

11. Pour les chaînes au format C (CSTRING), les tailles possibles peuvent aller de 1 à n octets (n étant la valeur maximale qui peut être allouée dans la pile de l'application comme indiqué dans le commentaire 9, 255 étant la taille par défaut).

12. Pour les caractères, les tailles possibles peuvent aller de 1 à 65000 octets (1 étant la taille par défaut).

13. Pour les segments comme pour les contrôles, aucune taille ne peut être indiquée.

14. Il est possible d'insérer des définitions multiples de variables dans la même ligne en insérant un séparateur virgule (,) entre chaque définition.

15. Contrairement à la notation utilisée dans le reste de la documentation, les crochets [ ] de la syntaxe ne signifient pas un champ optionnel mais sont réellement les caractères [ ].

16. La différence entre STRING et CSTRING se situe au niveau du stockage mémoire. Le premier octet d'un STRING contient le nombre effectif de caractères de la chaîne, le deuxième octet contient le code du premier caractère, etc... Le premier octet d'un CSTRING contient le code du premier caractère, le deuxième octet contient celui du deuxième caractère, ... et cela est terminé par un octet supplémentaire contenant la valeur 0 pour indiquer la fin de la chaîne.

17. Lorsque le champ <type>est précisé, le nom de la variable peut être quelconque, sans obligation de terminer par un % pour les entiers, £ ou # pour les réels, et $ pour les chaînes. Pour résumer :

```
LOCAL INT TOTO ; OK
LOCAL INT TOTO% ; OK
LOCAL INT TOTO£ ; Non !
LOCAL TOTO ; Non !
LOCAL TOTO% ; OK

SEGMENT IDENTITE
...
ENDSEGMENT
LOCAL IDENTITE TOTO ; OK
LOCAL IDENTITE TOTO% ; Non !
```

18. Dans le code NCL associé à chaque événement, les définitions LOCAL doivent être placées et regroupées IMPERATIVEMENT en tête de script. Si une définition ne respecte pas cette règle, le message d'erreur "Invalid place for definition" apparaît.

19. Utilisez SETDATA et GETDATA% pour manipuler des données locales à une fenêtre.

20. Chaque dimension maximum théorique est 65000. En pratique, la dimension maximum est inférieure, chaque variable ne pouvant occuper plus de 64K en mémoire. Ainsi :

```
LOCAL I%[30000]; Incorrect car 4*30000 =120000 > 65000
LOCAL I%(1)[50000]; Correct !
LOCAL S$[50][10] ; Incorrect car 256*50*10 = 128000
LOCAL S$(3)[100][100]; Correct! car 4*100*100 = 40000
```

21. Les variables locales sont stockées dans la pile. L'ensemble des variables locales d'un même événement ne peut donc dépasser le STACKSIZE spécifié dans le fichier de définition .DEF. Il est toutefois recommandé de ne pas dépasser 1 à 2 Ko pour la taille totale des variables locales d'un événement

22. Comme la taille par défaut d'une chaîne est de 255 octets, il est recommandé de préciser la taille réellement utilisée de chaque variable chaîne afin d'optimiser les allocations mémoire.

23. Bien que le module de test initialise les variables locales à 0, il est IMPERATIF d'initialiser ou d'affecter chaque variable locale avant son utilisation sous peine de ne pas avoir un comportement correct de l'application générée finale. En effet, dans le code C généré aucune variable locale n'est initialisée à une valeur par défaut, elles ont donc la valeur de la partie de la pile où elles ont été affectées !

Exemple 1

```
LOCAL A%, INT B
LOCAL NUM N#
LOCAL CSTRING C$(32)
LOCAL ARRAY1%[12]
LOCAL ARRAY2£[3], ARRAY3$[5], ARRAY4$(10)[9]
LOCAL ARRAY5%[2][3]
;
; A% et B sont des entiers, N# est un nombre réel, C$ est
; une chaîne de 32 caractères maximum.
;
; ARRAY1% est un tableau de 12 éléments entiers. ARRAY1%[0]
; accède au premier élément. ARRAY1%[11] accède au dernier
; élément.
;
; ARRAY2£ est un tableau de 3 éléments réels.
;
; ARRAY3$ est un tableau de 5 chaînes. Par défaut une chaîne
; peut contenir 255 caractères.
;
; ARRAY4$ est un tableau de 9 chaînes. La taille par défaut
; (255) peut être modifiée en définissant explicitement la
; taille des éléments (dans notre exemple, c'est 10).
;
; ARRAY5% est un tableau d'entiers à deux dimensions.
```

Exemple 2

```
LOCAL @I1%(1), @I2%(2), @I4%, @N4#(4), @N8#, @S80$(80), @S255$, I%(1)
I%=0
; initializing the pointer
@I1% =I%

message 'I1%', I1% ; displays 0

I1% = I1% + 1
message 'I1%', I1% ; displays 1

I% = I% + 1
message 'I1%', I1% ; displays 2
```

Voir aussi ***GLOBAL***, ***SEGMENT***, ***SIZEOF***, ***CONTROL***, ***FUNCTION***, ***INSTRUCTION***, ***SETDATA***, ***GETDATA%***, ***NEW***

Instruction LOCK

Bloque toute entrée utilisateur sur un contrôle.

Syntaxe	**LOCK** *contrôle*

1. Cette instruction est utilisable sur les contrôles Entry-Fields, CBE ou MLE afin de rendre impossible toute saisie utilisateur. Elle est équivalente à cocher "Lock text" dans leurs volets de propriétés. Après le LOCK, le focus peut toujours être dirigé sur ces contrôles et une sélection de caractères y est toujours possible, par exemple pour pouvoir copier la sélection dans le presse-papiers avec [Shift]+[Inser].
2. Cette instruction est aussi utilisable sur tous les autres contrôles (List-Box, Check-Box, Radio-Button, Scroll-Bar, ...) afin de rendre impossible toute action ou sélection utilisateur. Après le LOCK, Le focus ne peut plus être dirigé sur ces contrôles sauf sur la List-Box qui peut toujours être déroulée.
3. L'instruction LOCK ne s'applique pas aux contrôles Win32.
4. Il n'y a aucune différence visuelle entre un contrôle bloqué et non bloqué, contrairement à DISABLE qui grise le contrôle.

Exemple :

```
LOCK ENTRY00001
```

Voir aussi UNLOCK, ISLOCKED%, DISABLE, Paramétrage dynamique .LOCKED

Instruction LOOP

Répète indéfiniment les instructions situées entre LOOP et ENDLOOP.

Syntaxe	**LOOP** **...** **ENDLOOP**

1. Comme il est interdit de garder la main indéfiniment sur un événement Windows, sous peine de bloquer le multi-tâches du système d'exploitation, il est fortement conseillé d'avoir au moins un BREAK (ou EXIT, ou RETURN) au milieu de la boucle.
2. La construction LOOP/ENDLOOP est strictement équivalente à REPEAT/UNTIL FALSE%, WHILE TRUE%/ENDWHILE ou FOR/ENDFOR avec un pas d'incrémentation nul.

Exemple 1 :

```
; Comment bloquer le système d'exploitation
; (= très mauvais exemple !!)
LOOP
ENDLOOP
```

Exemple 2 :

```
; La construction suivante est équivalente à
; REPEAT
; MOVE I%+1 TO I%
; UNTIL I% = 100

MOVE 0 TO I%
LOOP
MOVE I%+1 TO I%
IF I% = 100
BREAK
ENDIF
ENDLOOP
```

***Voir aussi** REPEAT, WHILE, FOR, BREAK, CONTINUE*

Opérateur LOW

Retourne la partie basse d'un entier.

Syntaxe	**LOW** *entier*

1. La valeur retournée est le reste de la division entier/65536.
2. Un entier standard étant mémorisé sur quatre octets, il s'agit des deux octets les moins significatifs (LOW = low word).
3. Lorsque l'entier est stocké sur un ou deux octets, LOW retourne toujours intégralement l'entier.

Exemple :

```
MOVE LOW 500000 TO I% ; I% vaut 41248
; (50000 s'écrit 7A210 en hexa, et A210 vaut 41248 en décimal)
MOVE LOW 10000 TO I% ; I% vaut 10000
```

***Voir aussi** HIW, DWORD%, LOB*

Opérateur LOWCASE

Convertit une chaîne en minuscules.

Syntaxe	**LOWCASE** *chaîne*

Exemple :

```
MOVE LOWCASE "AbcD1" TO S$ ; S$ vaut "abcd1
```

Voir aussi ***UPCASE***

Opérateur LSKIP

Enlève les éventuels espaces et tabulations situés en tête d'une chaîne.

Syntaxe	**LSKIP** *chaîne*

Exemple :

```
MOVE LSKIP " un exemple " TO S$ : S$ vaut "un exemple "
```

Voir aussi RSKIP, SKIP

Fonction MAINWINDOW%

Retourne le handle de la première fenêtre ouverte de l'application, également appelée fenêtre principale.

Syntaxe	**MAINWINDOW%**

1. Dans l'événement INIT de la fenêtre principale, MAINWINDOW% vaut 0.
2. Lors de l'utilisation de NatStar PM, MAINWINDOW% vaut toujours 0.

Exemple 1 :

```
; Ouverture d'une fenêtre fille de la fenêtre principale
OPEN SCREEN, MAINWINDOW%
```

Exemple 2 :

```
; Autre exemple
CLOSE MAINWINDOW% ; Fermeture de l'application
```

Voir aussi SELF%, PARENTWINDOW%

Instruction MAXIMIZE

Maximise une fenêtre.

Syntaxe	MAXIMIZE [handle-fenêtre]

Maximise la fenêtre courante si aucun handle n'est précisé.

Exemple 1 :

```
MAXIMIZE ;maximise la fenêtre courante
```

Exemple 2 :

```
MAXIMIZE HW% ;maximise la fenêtre dont le handle ;vaut HW%
```

***Voir aussi** MINIMIZE, RESTORE, ISMINIMIZED%, ISMAXIMIZED%*

Instruction MESSAGE

Ouvre une boîte de message avec un titre et un message d'information. Une icône point d'exclamation et un bouton OK sont affichés dans la boîte.

Syntaxe	**MESSAGE** *chaîne-titre, chaîne-message*

1. MESSAGE ne doit être utilisé que pour afficher des informations à l'utilisateur. Celui-ci n'a d'autre action possible que de fermer la boîte par appui sur le bouton OK.
2. Cette instruction peut également être utilisée dans des buts de mise au point sans utilisation du débogueur, pour visualiser un passage dans le code, afficher des valeurs de variables, etc.

Exemple :

```
MESSAGE "Valeur de I%", I%
MESSAGE "Attention", "Valeur non valide !"
```

Voir aussi BEEP, ASK2%, ASK3%, MESSAGE% de la librairie NSMisc

Instruction MINIMIZE

Minimise une fenêtre.

Syntaxe	**MINIMIZE** *[handle-fenêtre]*

Minimise la fenêtre courante si aucun handle n'est spécifié.

Exemple 1 :

```
MINIMIZE ;minimise la fenêtre courante
```

Exemple 2 :

```
MINIMIZE TEST% ;minimise la fenêtre dont le handle ;est dans TEST%
```

Voir aussi ***MAXIMIZE***, ***RESTORE***, ***ISMINIMIZED%***, ***ISMAXIMIZED%***

Constantes MOU_*%

Identifiant du bouton souris utilisé.

Syntaxe	**MOU_PRIMARY%** **MOU_SECONDARY%** **MOU_TERTIARY%**

1. Signification des constantes :
   a) MOU_PRIMARY% Bouton primaire de la souris
   b) MOU_SECONDARY% Bouton secondaire de la souris
   c) MOU_TERTIARY% Bouton tertiaire (du milieu) de la souris
2. Leur déclaration interne est :

```
CONST MOU_PRIMARY%   1
CONST MOU_SECONDARY% 2
CONST MOU_TERTIARY%  3
```

3. Par défaut, MOU_PRIMARY% correspond au bouton gauche de la souris et MOU_SECONDARY% au bouton droit. C'est le contraire si les actions des boutons ont été inversées dans le Control Panel.

Exemple :

```
; Evénement BUTTONDOWN d'une fenêtre
IF PARAM3% = MOU_PRIMARY%
  MESSAGE "ButtonDown" , "Bouton primaire enfoncé"
ELSE
  MESSAGE "ButtonDown" , "Bouton secondaire enfoncé"
ENDIF
```

Voir aussi Evénements ***BUTTONDOWN***, ***BUTTONUP***, ***BUTTONDBLCLK*** ***NBBUTTONS%***

Instruction MOV

Copie une zone mémoire source vers une zone mémoire destination.

Syntaxe	**MOV** *adresse-source, adresse-destination, longueur*

1. La longueur de la zone mémoire source à recopier doit être inférieure ou égale à la longueur de la zone mémoire destination.
2. Ne pas confondre avec l'instruction MOVE (avec un E final).
3. Cette instruction peut s'avérer dangereuse lorsque adresse-destination et longueur ne désignent pas une zone correcte. Ainsi l'exemple suivant a de grandes chances de provoquer un General Protection Fault sous Windows (erreur de violation mémoire détectée par le système d'exploitation) :

```
MOV @I%, 0, 10000 ; TRES MAUVAIS EXEMPLE !!!
```

Exemple :

```
SEGMENT TEST
INT I(1) ; octet 0
STRING S1(4) ; octets 1 à 5
CSTRING S2(4) ; octets 6 à 10
INT J(1) ; octet 11
ENDSEGMENT
NEW TEST, A%
NEW TEST, B%
...
MOV A%, B%, 6 ; copie des octets 0 à 5 de A% vers B%
   ; ce qui équivalent à un MOVE TEST(A%).I TO TEST(B%).I
   ; plus un MOVE TEST(A%).S1 TO TEST(B%).S1
```

***Voir aussi** FILL, COPY$, NEW, SIZEOF*

Instruction MOVE

Assigne la valeur définie dans une expression à une variable ou à un contrôle.

Syntaxe	**MOVE** *expression* **TO** *variable*|*contrôle*

1. Cette instruction est très complète car elle permet de facilement modifier les champs d'une boîte de dialogue. Il suffit d'assigner les nouvelles valeurs des contrôles et les champs seront automatiquement modifiés.
2. Par sa puissance de traitement à la fois sur les nombres, les chaînes, et les contrôles, MOVE est l'instruction LA PLUS IMPORTANTE du langage NCL.
3. L'assignation est également possible à l'aide du "=". Dans ce cas là, la syntaxe est :

```
variable | contrôle = expression
```

4. L'utilisation du MOVE ou du "=" pour l'assignation est indifférente en NCL.
5. Ne pas confondre avec l'instruction MOV (sans le E final).
6. Un MOVE permet de modifier dynamiquement les images affichées dans un Bitmap de type Icon ou Push Button. Voir la partie "Bitmaps" du chapitre Eléments du langage DEBde ce manuelFIN pour plus de détails.
7. MOVE permet d'insérer un descripteur de présentation dans une List Box ou une Combo Box. Pour plus d'information sur cette utilisation de MOVE, reportez-vous à l'annexe E.

Exemple 1 :

```
MOVE 1+2 TO TEST% ;Met 3 dans la variable entière TEST%

MOVE "BONJOUR" TO TEST$ ;Met "BONJOUR" dans la chaîne TEST$
MOVE "BONJOUR" TO TEST ;Met "BONJOUR" dans le contrôle TEST
MOVE BONJOUR TO TEST$ ;Met la valeur du contrôle BONJOUR
; dans la variable TEST$
MOVE BONJOUR TO TEST ;Met la valeur du contrôle BONJOUR
; dans le contrôle TEST

MOVE CHECKED% TO CHECKBOX ; Coche le check-box CHECKBOX

MOVE SCREEN(H%).ENTRY TO S$ ; S$ contient le texte affiché
; dans l'entry-field ENTRY de la
; fenêtre SCREEN dont le
; handle est H%
```

Exemple 2 :

```
; Script d'un événement SELECTED d'une list-box LISTE
MOVE LISTE TO ENTRY ; à chaque sélection d'une ligne de
; la list-box, la ligne sélectionnée
; sera copiée dans l'entry-field ENTRY
```

```
; Le MOVE d'écriture sur un Entry-Field Spin-Button
; est spécifique.
MOVE "8 4 20 2" TO ENTRY00001 ; l'Entry-Field Spin-Button
; prend alors les caractéristiques suivantes  :
; 8 est la valeur initiale qu'il affiche
; 4 est la valeur minimale
; 20 est la valeur maximale
; 2 est l'incrément
```

Exemple 3 :

```
; Le MOVE de lecture sur un Entry-Field Spin-Button
; est normal (comme un Entry-Field standard).
MOVE ENTRY00001 TO S$ ; S$ contient la valeur affichée par
; l'Entry-Field
```

Voir aussi Evénements de comportement GETVALUE et SETVALUE, Paramétrage dynamique .VALUE

Fonction NBBUTTONS%

Retourne le nombre de boutons de la souris (par exemple, retourne 2 pour les souris Microsoft ou IBM).

Syntaxe	NBBUTTONS%

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

Voir aussi ISMOUSE%, Constantes MOU_*%

Instruction NEW

Alloue une zone de donnée en mémoire et fournit l'adresse de la zone mémoire allouée.

Syntaxes	**NEW** *nom-segment | taille, adresse-mem [, nom-public]* **NEW** *@pointeurTypé* **NEW** *[dim-tableau-sans-taille]@nom-tableau*

1. Le premier paramètre détermine le découpage interne de la zone de donnée.
   a) Si c'est un nom de segment, ce découpage est identique à celui du segment.
   b) Si c'est une valeur numérique, il n'y a aucun découpage, la zone est identifiée par sa taille exprimée en octets.
2. nom-public est optionnel. Ce paramètre permet de préciser le nom public de la variable : il est typiquement utilisé dans le cas d'échange de données entre applications.
3. adresse-mem est nulle lorsque NEW n'a pu allouer la mémoire demandée.
4. La libération mémoire s'obtient par DISPOSE.
5. La deuxième syntaxe permet d'allouer un segment de même type que le type de pointeur passé en paramètre et affecte son handle au pointeur @pointeurTypé.
6. La troisième syntaxe permet d'allouer un tableau dont la taille n'avait pas été définie préalablement.
7. L'adresse d'une variable préalablement déclarée s'obtient directement grâce au caractère "@" suivi du nom de la variable.
8. La déclaration d'une variable suit la syntaxe suivante :
9. {Type}{@} NomVar{(TailleType)}{[Dimension1]{[Dimension2]...}}
10. De même, les déclarations de retour de fonction suivent la syntaxe suivante :

```
Function NomFonction{(paramètres)} Return Type{(TailleType)}{@}
```

La première dimension du tableau [Dimension1] peut être nulle. Cela signifie que la taille n'est pas connue lors de la déclaration et permet d'avoir un tableau de taille variable. Il est alors nécessaire de spécifier la taille lors de l'allocation du segment.

```
LOCAL SEG@ rSeg[]
NEW [expr%] @rSeg
```

11. Par ailleurs, si vous spécifiez un tableau de taille non définie dans un segment, il est absolument nécessaire de le définir comme dernier champ du segment.

```
SEGMENT SEG
 …
 CSTRING TEXTS[] ; impérativement le dernier champ
ENDSEGMENT
```

12. Il est impératif d'utiliser l'instruction NEW pour allouer dynamiquement un segment contenant une ou plusieurs variables de type DYNSTR. En effet, il insère dans l'en-tête du bloc mémoire retourné un pointeur permettant à l'instruction DISPOSE de libérer la ou les variables de type DYNSTR du segment ou du tableau de segment.

13. Par contre, pour la même raison, il est vivement déconseillé de faire NEW expression-entière, ptr_var, si ptr_var pointe sur un segment contenant des DYNSTR. Il faut faire soit :

```
 Local Pointer ptr_var
 …
 New segAvecDS, ptr_var
 …
 Dispose ptr_var
```

soit (plus pratique) :

```
 Local segAvecDS@ segVar
 …
 New @segVar
 …
 Dispose @segVar
```

Exemples de déclarations :

```
;Un pointeur typé sur une chaîne Pascal et un pointeur typé sur une chaîne C d'au ;plus
$FFC0 (= 65472) caractères (la longueur maximale).
Global String@ Str, @Str2$($FFC0)
;Un pointeur typé sur un segment et un pointeur typé sur un tableau de 36 entiers
;(attention, ce n'est pas un tableau de 36 pointeurs typés sur des entiers).
Local typeSeg@ varSeg, @I%[36]
;Un pointeur typé sur un segment de même type (liste chaînée) et un pointeur typé sur ;un
autre segment.
Segment typeSeg2
typeSeg2@ next
…
typeSeg3@ moreInfo
…
EndSegment ; typeSeg2
;Un pointeur typé sur un segment contenant un pointeur type sur un autre segment, ;utile
quand on doit passer un pointeur sur un pointeur.
Segment typeSeg3
… ; définition des champs
EndSegment ; typeSeg3
```

```
Segment typePSeg3
 typeSeg3@ pSeg3
EndSegment ; typePSeg3
…
Local typePSeg3@ pPSeg3
Exemple 1
SEGMENT SAMPLE
INT I
INT J
ENDSEGMENT
NEW SAMPLE, HSEG%
MOVE 1 TO SAMPLE(HSEG%).I
MOVE 2 TO SAMPLE(HSEG%).J
...
DISPOSE HSEG%

; Equivalent à :
GLOBAL SAMPLE S1
MOVE 1 TO S1.I
MOVE 2 TO S1.J
```

Exemple 2 :

```
; Alloue une zone de 1024 octets en mémoire
; l'adresse de cette zone est rangée par l'instruction dans H%
GLOBAL H%
NEW 1024, H%
```

Exemple 3 :

```
; Code NCL de l'application A
SEGMENT SAMPLE2
INT I
ENDSEGMENT
NEW SAMPLE2, HSEGA%, "SAMP"

; Code NCL de l'application B
; Si le code de l'appli A avait été exécuté
; préalablement, le NEW de l'application B
; ne fait aucune allocation réelle mais permet
; simplement de pointer sur la même variable.
SEGMENT SAMPLE2
INT I
ENDSEGMENT
NEW SAMPLE2, HSEGB%, "SAMP"

; Les deux applications ont maintenant en commun
; la même variable.

; Code NCL de l'application A
MOVE 2 TO SAMPLE2(HSEGA%).I

; Code NCL de l'application B
MOVE SAMPLE2(HSEGB%).I TO I% ; I% vaut 2

; Code NCL de l'application A
DISPOSE HSEGA%

; Code NCL de l'application B
; Même si l'application A a fait préalablement un
; DISPOSE, l'application B peut continuer à utiliser
; la variable. Elle sera réellement désallouée lors
```

```
; du dernier DISPOSE.
DISPOSE HSEGB%
```

Exemple 4 :

```
Utilisation d'un segment typé
Segment sPersonne
  Prenom$(31)
  Nom$(31)
  DateDeNaissance%
  No$(7)
  NomRue$(35)
  CodePostal$(7)
  Ville$(63)
EndSegment ; {sPersonne}

Local sPersonne @personne

New @personne ; Allocation du segment
personne.Prenom$ = "François"
personne.Nom$ = "PIGNON"
...
//
Dispose @personne ; desallocation du Segment
```

Exemple 5 :

```
Utilisation d'un tableau de taille définie
Const kDimArray% 100
Segment sPersonne
  Prenom$(31)
  Nom$(31)
  DateDeNaissance%
  No$(7)
  NomRue$(35)
  CodePostal$(7)
  Ville$(63)
EndSegment ; {sPersonne}

Global sPersonne@ TabPers[kDimArray] ; taille du tableau imposée de 100

Local I%

New @TabPers ; Allocation du tableau
For I% = 0 To kDimArray% - 1
    Tabpers[I%].Prenom$ = "François"
    Tabpers[I%].Nom$ = "PIGNON"
    ...
EndFor

//
Dispose @TabPers ; desallocation du Tableau
```

Exemple 6 :

```
Utilisation d'un tableau de taille non-prédéfinie
Segment sPersonne
  Prenom$(31)
  Nom$(31)
  DateDeNaissance%
  No$(7)
```

```
  NomRue$(35)
  CodePostal$(7)
  Ville$(63)
EndSegment ; {sPersonne}

Global sPersonne@ TabPers[] ; taille du tableau non prédéfinie

Local I%
Local kDimArray%
; Ajustement de la taille du tableau
kDimArray%  = 50
New [kDimArray%]@TabPers ; Allocation du tableau une fois la taille ;déterminée
For I% = 0 To kDimArray% - 1
    Tabpers[I%].Prenom$ = "François"
    Tabpers[I%].Nom$ = "PIGNON"
    ...

EndFor
//
Dispose @TabPers ; desallocation du Tableau
```

***Voir aussi** DISPOSE, SEGMENT, FILL, MOV, SIZEOF, GLOBAL, LOCAL*

Constante NO%

Valeur retournée par les fonctions ASK2% et ASK3% lorsque le bouton No a été appuyé.

Syntaxe	NO%

Sa déclaration interne est :

```
CONST NO% 1
```

Voir aussi Fonctions ASK2%, ASK3%, Constantes YES%, CANCEL%

Constante NOSELECTION%

Valeur retournée par la fonction SELECTION% si aucune ligne n'est sélectionnée dans le contrôle de type liste.

Syntaxe	**NOSELECTION%**

Sa déclaration interne est :

```
CONST NOSELECTION% -1
```

Exemple :

```
IF SELECTION%(LB_SOCIETE) = NOSELECTION%
  MESSAGE "Erreur" , "Aucune ligne n'a été sélectionnée"
ELSE
...
ENDIF
```

Voir aussi SELECTION%

Opérateur NOT

Effectue un NON logique sur un entier.

Syntaxe	**NOT** *entier*

L'opérateur teste si l'entier vaut FALSE%: il retourne 0 (FALSE%) si l'entier est différent de zéro, et retourne 1 (TRUE%) si l'entier est égal à zéro.

Exemple :

```
MOVE NOT TRUE% TO I% ; I% vaut FALSE%
MOVE NOT FALSE% TO I% ; I% vaut TRUE%
MOVE NOT 12 TO I% ; I% vaut 0 (FALSE%)
```

Voir aussi AND, OR, BAND, BOR, BXOR, BNOT

Instruction NOUPDATE

Bloque la mise à jour automatique d'un objet de type liste.

Syntaxe	**NOUPDATE** *objet*

1. L'objet doit être l'un des suivants :
   a) Window classe Edit ou List
   b) List-box
   c) Combo-box
   d) CBE-Field
   e) MLE-Field
2. Une fois le NOUPDATE effectué, les instructions telles que INSERT ou DELETE sur l'objet n'auront plus d'effet visible à l'affichage. La mise à jour de l'affichage ne sera faite que lors du UPDATE correspondant suivant. Cela évite le désagréable "clignotement" lors de modifications successives de la liste par INSERT ou DELETE.
3. NOUPDATE incrémente un compteur "mise à jour" sur le contrôle, alors que UPDATE décrémente ce compteur. La mise à jour (donc le réaffichage de la liste) n'aura lieu que lorsque ce compteur revient à zéro.

Exemple :

```
NOUPDATE LISTE
MOVE 0 TO I%
REPEAT
MOVE I%+1 TO I%
INSERT AT END I% TO LISTE
UNTIL I% = 50
UPDATE LISTE
```

Voir aussi DELETE, INSERT, UPDATE, LINECOUNT%, Paramétrage dynamique .UPDATED

Opérateur NUM

Convertit en nombre réel.

Syntaxe	**NUM** *entier*|*réel*|*chaîne*

NUM peut aussi être utilisé en tant que "type" au sein des instructions LOCAL, GLOBAL, SEGMENT, FUNCTION et INSTRUCTION

Exemple :

```
MOVE NUM 1 TO I£ ; I£ vaut 1.
MOVE NUM "1.2" TO I£ ; I£ vaut 1.2
```

***Voir aussi** INT, STRING, ISNUM%, LOCAL, GLOBAL, SEGMENT, FUNCTION, INSTRUCTION*

Type NUM

Type réel, utilisé au sein des instructions LOCAL, GLOBAL, SEGMENT, FUNCTION et INSTRUCTION.

1. Les tailles possibles pour une variable de type NUM sont 4, 8 ou 10 octets (8 étant la taille par défaut).
2. La taille d'un réel influe sur les nombres minimum et maximum acceptés :

Taille	Minimum et maximum acceptés
4	signe sur 1 bit, mantisse sur 24 bits, exposant sur 7 bits soit environ [+/- 8.43 E-37, +/- 3.37 E+38]
8	signe sur 1 bit, mantisse sur 52 bits, exposant sur 11 bits soit environ [+/- 4.19 E-307, +/- 1.67 E+308]
10	signe sur 1 bit, mantisse sur 64 bits, exposant sur 15 bits soit environ [+/- 3.4 E-4932, +/- 1.2 E+4932]

Le type NUM (10) n'étant plus supporté par les run-time Microsoft, il a été mappé sur les NUM (8)

Voir aussi ***INT***, ***INTEGER***, ***DYNSTR***, ***LOCAL***, ***GLOBAL***, ***SEGMENT***, ***FUNCTION***, ***INSTRUCTION***

Instructions OPEN*

Ouvre une fenêtre non-modale.

Syntaxes	**OPEN, OPENH, OPENS, OPENSH** *nom-fenêtre, 0 | handle-fenêtre-mère [,handle-fenêtre]* *[USING Param1 [, Param2 [, Param3 [, Param4]]]]* **OPEN, OPENH, OPENS, OPENSH** *nom-fenêtre, 0 | handle-fenêtre-mère [,handle-fenêtre]* *[USING Param1 [, Param2 [,Param34[, Param4]]]* **OPEN, OPENH, OPENS, OPENSH** *nom-fenêtre, 0 | handle-fenêtre-mère [,handle-fenêtre]* *[USING Param12 [, Param3 [, Param4]]]* **OPEN, OPENH, OPENS, OPENSH** *nom-fenêtre, 0 | handle-fenêtre-mère [,handle-fenêtre]* *[USING Param12 [, Param34]]*

1. Une fenêtre non modale est une fenêtre qui existe parallèlement aux autres fenêtres de l'application, et qui ne bloque donc pas les événements des autres fenêtres. L'instruction après l'OPEN sera directement exécutée après l'ouverture de la fenêtre.

2. Le second paramètre est soit 0 (dans ce cas, la fenêtre créée n'est pas fille), soit le handle-fenêtre de la fenêtre mère (la fenêtre créée est fille). Le troisième paramètre optionnel est une variable entière dans laquelle est retourné le handle de la fenêtre.
Les paramètres Param1, Param2, Param3, Param4, Param12, Param34 seront reçus dans les paramètres PARAM1%, PARAM2%, PARAM3%, PARAM4%, PARAM12%, PARAM34% de l'événement INIT de la fenêtre ouverte.

3. Même si le paramètre handle-fenêtre est optionnel, il est souvent nécessaire de le spécifier. En effet, les fonctions ou instructions agissant sur les fenêtres nécessitent en paramètre le handle cible (sauf toutefois lorsque la fenêtre veut agir sur elle-même). Ainsi, dans le cas d'un OPENH, si la fenêtre qui a demandé l'OPENH veut faire un SHOW, il lui faudra repasser en paramètre ce handle.

4. Le handle fenêtre retourné ne correspond pas au handle interne du système d'exploitation. Pour connaître le handle réel du système, utilisez la

fonction GETCLIENTHWND% (ou GETFRAMEHWND%) de la librairie NSWin sous Windows.

5. OPENH a un comportement un peu différent de OPEN car il laisse caché la fenêtre ouverte. Un emploi ultérieur de l'instruction SHOW permet de la faire apparaître.

6. OPENS crée une fenêtre non-modale secondaire. Contrairement à OPEN et OPENH qui créent des fenêtres filles limitées à l'aire client de la fenêtre mère, OPENS crée une fenêtre qui peut apparaître en dehors de l'aire client de la fenêtre qui a provoqué l'ouverture. Attention, une ouverture de fenêtre secondaire n'implique pas une relation mère-fille.

7. Comme OPENS, OPENSH ouvre une fenêtre non-modale secondaire. Cette dernière est cachée au moment de son ouverture comme une fenêtre ouverte à l'aide de OPENH.

8. Toutes les fenêtres (deuxième paramètre non nul) ouvertes par OPEN, OPENH et OPENS sont liées aux déplacements de la fenêtre qui a provoqué l'ouverture, et ne peuvent jamais être cachée par cette dernière.

9. OPEN et OPENS sont équivalents lorsque la fenêtre créée est une fenêtre principale (deuxième paramètre valant 0).

10. De même, OPENH et OPENSH sont équivalents lorsque la fenêtre créée est une fenêtre principale (deuxième paramètre valant 0)

11. Une fenêtre fille MDI (de classe quelconque, mais dont la mère est de classe MDI Window) ne doit pas être démarrée avec OPENS, OPENSH, CALL ou CALLH. Employez uniquement OPEN ou OPENH.

12. Les fonctions PARAM12% et PARAM34% sont désormais des pointeurs.

13. Dans les emplacements prévus pour les paramètres PARAM1% et PARAM3%, il est possible de remplacer les opérateurs LOW expr_ent%, HIW expr_ent% par POINTER expr_ent%.

14. De même, si une expression de type ptr, ptr+expr_ent%, ptr-expr_ent% ou expr_ent%+ptr (où ptr est soit une variable de type POINTER, soit l'adresse d'une variable ou d'une chaîne littérale) apparaît à l'un des emplacements de PARAM1% et PARAM3% le pointeur est passé comme PARAM12% ou PARAM34% sans même avoir besoin de le préfixer par le type POINTER.

15. Si les expressions passées en paramètre sont de type POINTER, alors les instructions OPEN* utilisent PARAM12% au lieu de PARAM1% et PARAM2%, ou PARAM34% au lieu de PARAM3% et PARAM4%.

16. Si une expression de type POINTER est passé sans changement de type en INT aux emplacements prévus pour PARAM2% ou PARAM4%, vous obtiendrez un avertissement ou une erreur.

Exemple 1 :

```
OPEN PARENT, 0, PARENT% ; ouvre une fenêtre principale nommée PARENT (fichier PARENT.SCR)
BEEP ; le bip est fait dès l'ouverture de la fenêtre PARENT
```

Exemple 2 :

```
OPENH ENFANT, PARENT%, ENFANT% ; ouvre une fenêtre affiliée à PARENT pour l'instant
invisible
MESSAGE "Attente", "Fenêtre créée, mais non encore visible"
SHOW ENFANT% ; ENFANT devient visible
```

Exemple 3 :

```
OPEN SCREEN, SELF% ; Ouvre une fenêtre fille de la fenêtre courante
```

Exemple 4 :

```
LOCAL PAR1%, PAR2%, PAR3%, PAR4%
; Initialisation des paramètres
...
OPEN D_SCREEN, SELF%, H_SCREEN% USING PAR1%, PAR2%, \
PAR3%, PAR4%
```

Voir aussi :
STROPEN, STROPENH, STROPENS (cf. STROPEN*)
CALL, CALLH, STRCALL, STRCALLH (cf. STRCALL*)
Fonctions PARAM*%, PARAM*$
SHOW, CLOSE
MAINWINDOW%, SELF%
WINDOWNAME$ (Librairie NSMisc)

Opérateur OR

Effectue un « OU » inclusif entre deux opérandes.

Syntaxe	*entier* **OR** *entier*

1. L'opérateur teste si au moins un des deux opérandes vaut TRUE%, ou plus généralement si au moins un des deux opérandes a une valeur différente de zéro.
2. L'opérande de droite n'est calculé que si nécessaire. En effet, lorsque l'opérande de gauche est différent de zéro, le résultat est toujours TRUE%.

Exemple :

```
MOVE (TRUE% OR TRUE%) TO I% ; I% vaut TRUE%
MOVE (TRUE% OR FALSE%) TO I% ; I% vaut TRUE%
MOVE (FALSE% OR TRUE%) TO I% ; I% vaut TRUE%
MOVE (FALSE% OR FALSE%) TO I% ; I% vaut FALSE%
```

***Voir aussi** AND, NOT, BAND, BOR, BNOT, BXOR*

Fonctions PARAM*%, PARAM*$

Retournent des informations additionnelles passées avec les événements, dont la signification dépend de chaque événement.

Les fonctions PARAM12% et PARAM34% ont été modifiées dans les versions 5.00 afin d'améliorer la portabilité 64 bits. Les fonctions PARAM12% et PARAM34% sont désormais des pointeurs. Les fonctions PARAM1%, PARAM2%, PARAM3% et PARAM4% restent des entiers 16 bits.

Syntaxe	**PARAM1%** **PARAM2%** **PARAM3%** **PARAM4%** **PARAM12%** **PARAM34%** **PARAM1$** **PARAM2$**

1. Quatre entiers (de taille 2) sont passés avec chaque événement. Ils peuvent être obtenus en appelant les fonctions PARAM1%, PARAM2%, PARAM3% et PARAM4%.
2. PARAM12% et PARAM34% correspondent désormais à des pointeurs. PARAM1%, PARAM2%, PARAM3% et PARAM4% restent des entiers 16 bits. Par conséquent, sur une plateforme 64 bits, PARAM12% n'est plus équivalent à DWORD%(PARAM2%, PARAM1%), et PARAM34% n'est plus équivalent à DWORD%(PARAM4%, PARAM3%). Sur une plateforme 32 bits, PARAM12% et PARAM34% peuvent encore permettre d'obtenir deux entiers (de taille 4) combinés à partir des quatre entiers précédents. PARAM12% est équivalent à DWORD%(PARAM2%, PARAM1%), et PARAM34% est équivalent à DWORD%(PARAM4%, PARAM3%).
3. PARAM1$ et PARAM2$ sont aussi des facilités de programmation permettant de travailler avec des chaînes dont l'adresse serait stockée dans PARAM12% et PARAM34%. PARAM12% est équivalent à @PARAM1$, et PARAM34% est équivalent à @PARAM2$. PARAM1$ et PARAM2$ sont notamment utiles pour le traitement des événements de comportement.
4. Ces fonctions sont comme des variables "read only". Il est donc impossible de les modifier en faisant par exemple :

```
MOVE 0 TO PARAM1% ; Interdit !
```

5. sauf, dans certains cas, PARAM1$ et PARAM2$ comme indiqué ci-après.

6. Pour pouvoir utiliser PARAM1$ (en lecture uniquement) et PARAM2$ (en lecture/écriture), il est nécessaire que PARAM12% et PARAM34% pointent sur une zone allouée. En dehors des événements de comportement GETVALUE et SETVALUE pour lesquels PARAM2$ est utilisable, et en dehors d'éventuels événements utilisateurs correctement programmés, PARAM1$ et PARAM2$ ne doivent pas être employés.

```
; Evénement EXECUTED d'un Push-Button
MOVE "Coucou" TO PARAM2$
; TRES MAUVAIS EXEMPLE!
; GP Fault sous Windows
```

7. Par ailleurs, il est désormais possible de passer un entier 32 bits en une seule fois (sans utiliser les opérateurs HIW et LOW) dans PARAM12% et PARAM34%. Pour cela, il faut indiquer le mot-clé POINTER avant l'expression prévue pour PARAM1%/PARAM12% ou PARAM3%/PARAM34%.

```
LOCAL POINTER P, I%, retVar%(2)

SEND USER0,
SEND USER1, POINTER $12345678, 12345, 23456
;envoie USER0 à la fenêtre courante avec ;Param12%=$12345678, Param3%=12345 et
Param4%=23456
CALL MYDIALOG USING P + 1, @I%, retVar%
;Ancienne syntaxe : CALL MYDIALOG USING\
;LOW(P + 1),HIW(P+1),LOW@I%,HIW@I%,retVar%
;ouvre la fenêtre modale MYDIALOG et passe P + 1 ;dans Param12% et @I% dans Param34% à
;l'événement INIT
```

8. Les quinze instructions suivantes sont impactées par ces nouveautés : PASS, SEND, POST, CALL, CALLH, STRCALL, STRCALLH, OPEN, OPENH, OPENS, OPENSH, STROPEN, STROPENH, STROPENS, STROPENSH.

Exemple :

```
; Passage d'une chaîne en paramètre d'un événement
GLOBAL S$
MOVE "Bonjour !" TO S$
SEND USER1, LOW @S$, HIW @S$ TO H%

; Evénement USER1 de fenêtre H%
; Affiche la chaîne qui vaut ici "Bonjour !"
MESSAGE "Démo PARAM1$", PARAM1$
```

Voir aussi :

Chapitre "Référence des Evénements", POINTER

PASS, SEND, POST, CALL*, STRCALL*, OPEN*, STROPEN*

Fonction PARENTWINDOW%

Retourne le handle de la fenêtre mère d'une fenêtre.

Syntaxe	**PARENTWINDOW%** *[(handle-fenêtre)]*

1. Si aucun handle n'est spécifié, PARENTWINDOW% renvoie le handle de la fenêtre mère de la fenêtre courante. PARENTWINDOW% est donc équivalent à PARENTWINDOW%(SELF%)
2. La fonction retourne 0 si la fenêtre n'a pas de fenêtre mère.
3. Cette fonction retourne uniquement le handle de la fenêtre mère. Dans le cas de fenêtres secondaires, la fenêtre qui a effectué le OPENS n'est pas considérée comme une fenêtre mère, PARENTWINDOW% retourne donc 0.

Exemple :

```
; Evt INIT de la fenêtre D_SCREEN de handle H_SCREEN%

OPEN D_CHILD1, MAINWINDOW%, H_CHILD1%

MESSAGE "Fenêtre mère de D_CHILD1" , \
PARENTWINDOW%(H_CHILD1%)
; Affichera la valeur de MAINWINDOW%

OPENS D_CHILD2, H_SCREEN%, H_CHILD2%

MESSAGE "Fenêtre mère de D_CHILD2" , \
PARENTWINDOW%(H_CHILD2%)
; Affichera 0
```

Voir aussi OPEN, OPENH (cf. OPEN*), STROPEN, STROPENH (cf. STROPEN*), MAINWINDOW%, SELF%

Instruction PASS

Appelle le traitement par défaut de l'événement courant.

Syntaxes	**PASS** *[param1 [, param2 [, param3 [, param4] [, variable-retour]]]]]* **PASS** *[param1 [, param2 [, param34 [, variable-retour]]]]* **PASS** *[param12 [, param3 [, param4 [, variable-retour]]]]* **PASS** *[param12 [, param34 [, variable-retour]]]*

1. Il s'agit d'un appel DIRECT de la routine par défaut associée à l'événement : après cet appel, l'exécution revient à l'instruction qui suit le PASS. Il est ainsi possible de surclasser le traitement par défaut d'un événement (PAINT, par exemple) tout en faisant appel à ce traitement par défaut, avant, pendant ou après le traitement particulier.
2. L'utilisation du surclassement d'un événement peut également être effectué dans un Control Template. Dans ce cas, PASS effectue l'appel au traitement par défaut codé dans le Template.
3. Les paramètres Param1, Param2, Param3, Param4, Param12, Param34 seront reçus dans les fonctions PARAM1%, PARAM2%, PARAM3%, PARAM4%, PARAM12%, PARAM34% de l'événement INIT de la fenêtre ouverte.
4. variable-retour reçoit la valeur retournée par le traitement par défaut de l'événement. Pour être spécifiée, elle doit être précédée des paramètres Param* adéquats.
5. RETURN DEFRET% effectue un PASS de façon implicite. Voir les événements de comportement.
6. Les fonctions PARAM12% et PARAM34% sont désormais des pointeurs.
7. Si les expressions passées en paramètre sont de type POINTER, alors l'instruction PASS utilise PARAM12% au lieu de PARAM1% et PARAM2% ou PARAM34% au lieu de PARAM3% et PARAM4%.
8. Si une expression de type POINTER est passée sans changement de type en INT aux emplacements prévus pour PARAM2% ou PARAM4%, vous obtiendrez un avertissement ou une erreur.

Exemple :

```
; Sur événement DISABLE d'un Push-Button PUSH00001
; (voir "Evénements de Comportement" dans le chapitre
; "Référence des Evénements")
IF PARAM1% = TRUE%
MESSAGE "Disable", "Début de traitement"
ELSE
MESSAGE "Enable", "Début de traitement"
ENDIF
PASS PARAM1%, PARAM2%, PARAM3%, PARAM4%
```

```
MESSAGE "Disable / Enable", "Fin de traitement"
RETURN

; Sur un événement d'un autre objet
DISABLE PUSH00001 ; Affiche la 1ère boîte de message
; (traitement spécifique)
; puis grise et désactive le bouton
; (PASS  : traitement par défaut)
; puis affiche la 2ème boîte de message
; (traitement spécifique)
```

Voir aussi ***SEND, RETURN, DEFRET%, Evénements de Comportement ,Fonctions PARAM*%, PARAM*$***

Type POINTER

Type entier de même taille qu'une adresse ou qu'un pointeur (handle), utilisé au sein des instructions LOCAL, GLOBAL, SEGMENT, FUNCTION, INSTRUCTION.

1. La taille d'un entier de type POINTER est automatiquement adaptée au système cible. Pour la plupart de ces systèmes, elle est de 4 octets donc équivalente à INT(4).
2. Il est conseillé d'utiliser ce type pour tout pointeur (NEW, SELF%, etc) afin de faciliter la portabilité d'une application sur plusieurs systèmes.
3. En NCL natif, en dehors des besoins particuliers de développement, il existe deux types de pointeurs :
4. les handles de segment, obtenus par l'instruction NEW,
5. les handles de fenêtres, obtenus par les instructions OPEN* ou CALL*, SELF% ou MAINWINDOW%.
6. L'intérêt de déclarer ces pointeurs avec le type POINTER est de ne pas spécifier leur taille, elle est déterminée par le système cible lors de la génération.

***Voir aussi** INT, GLOBAL, LOCAL, SEGMENT, FUNCTION, INSTRUCTION*

Fonction POS%

La fonction POS% évolue pour permettre de retourner la énième position d'une occurrence de chaîne de caractères dans une autre chaîne de caractères à partir d'une position précise.

Syntaxe	**POS%** *(chaîne-cherchée, chaîne-source, ordre, position-de-départ)*

1. Position = 1 correspond à l'index du premier caractère de la chaîne source, quel que soit le type de la chaîne (STRING ou CSTRING).
2. La fonction retourne 0 si la chaîne n'a pas été trouvée.
3. Les deux derniers paramètres permettent la recherche de la nième occurrence à partir d'une position précise.
4. Les deux derniers paramètres ordre et position-de-départ sont facultatifs. Le paramètre ordre peut être utilisé sans position-de-départ. Le paramètre position-de-départ ne peut pas être utilisé sans le paramètre ordre.
5. L'ordre indique le rang de l'occurrence recherchée.
6. L'ordre et la position de départ commencent à 1.
7. Les deux derniers paramètres peuvent être négatifs.
   * Une valeur négative dans le paramètre ordre indique que le comptage du rang de l'occurrence recherchée commence à partir de la fin de la chaîne. Par exemple, -1 signale le dernière occurrence de la chaîne, -2 l'avant-dernière.
   * Par ailleurs, une valeur négative dans le paramètre position-de-départ indique que le comptage de la position commence également à partir de la fin de la chaîne de caractères. Par exemple, -5 indique que la recherche s'effectue seulement parmi les cinq derniers caractères de la chaîne.

Exemple 1 :

```
MOVE POS% ("SYS", "NAT SYSTEMES") TO I% ; met 5 dans I%
```

Exemple 2 :

```
local Dynstr lib_doc$
lib_doc$ = "Déclaration Déclaration"
message "POS ",POS%("Décla",lib_doc$, 1 ) ; renvoie 1
message "POS ",POS%("Décla",lib_doc$, 1, 1 ) ; renvoie 1
message "POS ",POS%("Décla",lib_doc$, 2 ) ; renvoie 13
message "POS ",POS%("Décla",lib_doc$, 2, 1 ) ; renvoie 13
message "POS ",POS%("Décla",lib_doc$, 1, 5 ) ; renvoie 13
```

***Voir aussi** COPY$, DELETE$, FILLER$, INSERT$*

Constantes REL_*%

Constantes utilisées pour le paramétrage dynamique .RELIEF d'un contrôle.

Syntaxe	**REL_NONE%** **REL_LIGHT%** **REL_DARK%** **REL_LIGHT_OPT%** **REL_DARK_OPT%**

1. Signification des constantes :
   a) REL_NONE% : Pas de relief
   b) REL_LIGHT% : Relief Light (bombé)
   c) REL_DARK% : Relief Dark (en creux)
   d) REL_LIGHT_OPT% : Relief Light'opt (uniquement pour les group box)
   e) REL_DARK_OPT% : Relief Dark'opt (uniquement pour les group box)
2. Leur déclaration interne est :

```
CONST REL_NONE% 0
CONST REL_LIGHT% 1
CONST REL_DARK% 2
CONST REL_LIGHT_OPT% 3
CONST REL_DARK_OPT% 4
```

Exemple :

```
; Supprime le relief associé à l'entry field ENTRY00001
MOVE REL_NONE% TO ENTRY00001.RELIEF
```

Voir aussi Paramètre dynamique .RELIEF, Paramétrage dynamique .*

Paramètre .RELIEF

En lecture/écriture.

Le paramètre .RELIEF permet de définir le type du relief souhaité pour le contrôle.

Syntaxe	NomContrôle**.RELIEF** = REL_*

1. Ce paramètre dynamique correspond à la propriété Shadow du volet de propriétés des contrôles.
2. Le contrôle SheetBox est un cas particulier. Le paramètre .RELIEF permet ici de paramétrer ou non l'aspect bombé du contrôle. La syntaxe est donc différente : NomContrôleSheetBox=TRUE% | FALSE%.

Exemple :

```
; positionne le relief en creux
ENTRY00001.RELIEF=REL_DARK%
```

***Voir aussi** Constantes REL_* , Paramétrage dynamique .**

Instruction POST

Envoie de manière asynchrone un événement à un objet.

Syntaxes	**POST** *événement [, param1 [, param2 [, param3 [, param4]]]]* [**TO** *objet]* **POST** *événement [, param1 [, param2 [, param34]]]* [**TO** *objet]* **POST** *événement [, param12 [, param3 [, param4]]]* [**TO** *objet]* **POST** *événement [, param12 [, param34]]* [**TO** *objet]*

1. Un événement envoyé par POST est placé dans la queue de messages de l'objet et n'est pris en compte que lorsque tous les événements y figurant ont été traités.

2. Les paramètres Param1, Param2, Param3, Param4, Param12, Param34 seront reçus dans les paramètres PARAM1%, PARAM2%, PARAM3%, PARAM4%, PARAM12%, PARAM34% de l'événement envoyé de l'objet destination.

3. La signification des paramètres 1 à 4 dépend du traitement éventuel de PARAM1% à PARAM4% fait par l'événement appelé.

4. Si aucun objet n'est spécifié, l'événement est envoyé à la fenêtre courante.

5. Lorsque le nom de la fenêtre à laquelle appartient l'objet n'est pas précisé (par POST... TO H%), l'événement est envoyé à la fenêtre de même nom que la fenêtre courante.

6. L'instruction POST ne fonctionne pas dans l'événement TERMINATE d'une fenêtre.

7. Les fonctions PARAM12% et PARAM34% sont désormais des pointeurs.

8. Dans les emplacements prévus pour les paramètres PARAM1% et PARAM3%, il est possible de remplacer les opérateurs LOW expr_ent%, HIW expr_ent% par POINTER expr_ent%.

9. De même, si une expression de type ptr, ptr+expr_ent%, ptr-expr_ent% ou expr_ent%+ptr (où ptr est soit une variable de type POINTER, soit l'adresse d'une variable ou d'une chaîne littérale) apparaît à l'un des emplacements de PARAM1% et PARAM3% le pointeur est passé comme PARAM12% ou PARAM34% sans même avoir besoin de le préfixer par le type POINTER.

10. Si les expressions passées en paramètre sont de type POINTER, alors l'instruction POST utilise PARAM12% au lieu de PARAM1% et PARAM2% ou PARAM34% au lieu de PARAM3% et PARAM4%.

11. Si une expression de type POINTER est passée sans changement de type en INT aux emplacements prévus pour PARAM2% ou PARAM4%, vous obtiendrez un avertissement ou une erreur.

Exemple :

```
; Envoie l'événement utilisateur MYEVENT à la fenêtre
; WINDOW avec les paramètres qui lui sont spécifiques
; Cet événement est placé dans la pile des messages de
; la fenêtre et ne sera traité que lorsque les messages
; déjà présents auront été traités
POST MYEVENT, X%, Y% % TO WINDOW(H%)
```

Voir aussi :

SEND, PASS, Fonctions PARAM*%, PARAM*$

Tous les événements avec leurs différents PARAMx%

Fonction SENDMESSAGE% et Instruction POSTMESSAGE (Librairie NSWin sous Windows).

Instruction REPEAT

Débute une boucle logique REPEAT/UNTIL.

Syntaxe	**REPEAT** ... **UNTIL** *condition*

1. La séquence d'instructions comprise entre le REPEAT et le UNTIL est répétée jusqu'à ce que l'expression condition devienne TRUE%.
2. Dans tous les cas, les instructions entre REPEAT et UNTIL sont exécutées au moins une fois, la condition n'étant testée qu'après la première boucle. C'est ce qui diffère d'une construction WHILE/ENDWHILE.

Exemple :

```
MOVE 0 TO I%
REPEAT
MOVE I%+1 TO I% ; Sera répété jusqu'à ce que I%=10
UNTIL I% = 10
```

Voir aussi IF, LOOP, WHILE, FOR, BREAK, CONTINUE

Instruction RESTORE

Restaure une fenêtre avec la taille et la position qu'elle occupait avant d'avoir été maximisée ou minimisée.

Syntaxe	**RESTORE** *[handle-fenêtre]*

1. Si aucun handle n'est précisé, la restauration s'applique à la fenêtre courante.
2. La fenêtre spécifiée doit être maximisée ou minimisée, et RESTORE a l'effet d'une bascule. Si elle n'est ni maximisée, ni minimisée, RESTORE n'a aucun effet.

Exemple :

```
RESTORE ; Restaure la fenêtre courante
RESTORE HW% ; Restaure la fenêtre dont le handle vaut HW%
```

***Voir aussi** MINIMIZE, MAXIMIZE, ISMINIMIZED%, ISMAXIMIZED%*

Instruction RETURN

Fin d'exécution de la fonction, de l'instruction, ou de l'événement en cours.

Syntaxe	**RETURN** *[expression]*

1. Lorsque RETURN est employé au sein d'une instruction, aucune expression ne doit être passée en paramètre.
2. Lorsque RETURN est employé au sein d'une fonction ou d'un événement, il est obligatoire de passer une expression en paramètre : ce sera le code retourné par la fonction (voir FUNCTION) ou par l'événement (voir SEND).
3. Au sein d'une instruction, RETURN sans paramètre est rigoureusement équivalent à EXIT.
4. Ne pas confondre avec le séparateur RETURN utilisé avec l'instruction FUNCTION (impérativement sur la même ligne) qui précise le type retourné.
5. S'il est situé dans un événement, RETURN DEFRET% est rigoureusement équivalent à :

```
PASS PARAM1%, PARAM2%, PARAM3%, PARAM4%
```

***Voir aussi** BREAK, EXIT, HALT, FUNCTION, INSTRUCTION, PASS, SEND, DEFRET%, **Evénement** CHECK, Evénements de Comportement*

Fonction ROUND%

Convertit un nombre réel dans l'entier arrondi la plus proche.

Syntaxe	**ROUND%** *(réel)*

Exemple :

```
MOVE ROUND% (1.7) TO I% ; met 2 dans I%
MOVE ROUND% (1.3) TO I% ; met 1 dans I%
```

***Voir aussi** INT, TRUNC%*

Opérateur RSKIP

Enlève les éventuels espaces et tabulations situés en fin d'une chaîne.

Syntaxe	**RSKIP** *chaîne*

Exemple :

```
MOVE RSKIP " un exemple " TO S$ ; S$ vaut " un exemple"
```

Voir aussi LSKIP, SKIP

Instruction SAVE

Sauvegarde le contenu d'un objet liste dans un fichier texte.

Syntaxe	**SAVE** *objet* **TO** *nom-fichier*

1. L'objet doit être l'un des suivants :
   a) Window classe Edit ou List
   b) List-box
   c) Combo-box
   d) CBE-Field
   e) MLE-Field
2. Chaque ligne des listes sera séparée dans le fichier texte par un ensemble CR-LF
3. Si le fichier existe déjà, il est écrasé et remplacé par le nouveau.
4. Dans le cas d'une CBE, c'est le contenu de sa liste qui est sauvé, sans tenir compte du contenu de son champ d'édition.
5. Des parenthèses utilisées au sein de la chaîne nom-fichier permettent d'utiliser les variables d'environnement. Voir exemple.

Exemple 1 :

```
; Sauve la list-box LISTE dans le fichier SAMPLE.TXT
SAVE LISTE TO "SAMPLE.TXT"
```

Exemple 2 :

```
; Sauve dans le fichier SAMPLE.TXT dans le répertoire
; spécifié par la variable d'environnement SET NS-LST
; (ces variables peuvent être définies dans AUTOEXEC.BAT
; sous Windows)
SAVE LISTE TO "(NS-LST)SAMPLE.TXT"
```

Voir aussi LOAD et librairie NSMisc

Constantes SB_*%

Type de défilement effectué dans un objet, fenêtre ou contrôle, comportant une barre de défilement.

Syntaxe	**SB_ENDSCROLL%** **SB_LINEDOWN%** **SB_LINELEFT%** **SB_LINERIGHT%** **SB_LINEUP%** **SB_PAGEDOWN%** **SB_PAGELEFT%** **SB_PAGERIGHT%** **SB_PAGEUP%** **SB_SLIDERPOSITION%** **SB_SLIDERTRACK%**

1. Ces valeurs sont passées dans PARAM1% des événements HSCROLL et VSCROLL. Reportez-vous à ces événements pour une description de chacune de ces constantes.
2. Leur déclaration interne est :

```
CONST SB_LINELEFT% 1
CONST SB_LINEUP% 2
CONST SB_LINEDOWN% 3
CONST SB_PAGELEFT% 4
CONST SB_PAGERIGHT% 5
CONST SB_PAGEUP% 6
CONST SB_PAGEDOWN% 7
CONST SB_SLIDERTRACK% 8
CONST SB_SLIDERPOSITION% 9
CONST SB_ENDSCROLL% 10
```

Voir aussi Evénements HSCROLL et VSCROLL

Fonction SCREENHEIGHT%

Retourne la hauteur, en pixels, de l'écran physique.

Syntaxe	SCREENHEIGHT%

Exemple :

```
; Sur l'écran VGA, affiche Hauteur 480 et Largeur 640
MESSAGE "Hauteur écran" && SCREENHEIGHT%,
"Largeur écran" && SCREENWIDTH%
```

***Voir aussi** SCREENWIDTH%, GETCLIENTHEIGHT%, GETCLIENTWIDTH%, GETHEIGHT%, GETWIDTH%*

Fonction SCREENWIDTH%

Retourne la largeur, en pixels, de l'écran physique.

Syntaxe	**SCREENWIDTH%**

Exemple :

```
; Sur l'écran VGA, affiche Hauteur 480 et Largeur 640
MESSAGE "Hauteur écran" && SCREENHEIGHT%,
"Largeur écran" && SCREENWIDTH%
```

Voir aussi *SCREENHEIGHT%, GETCLIENTHEIGHT%, GETCLIENTWIDTH%, GETHEIGHT%, GETWIDTH%*

Instruction SEGMENT

Définit un segment de données. Entre SEGMENT et ENDSEGMENT, seules des déclarations de champs sont acceptées, avec une déclaration par ligne.
A partir de la version 3.00, le champ <type> peut contenir un nom de type suivi par @. Ceci à pour effet de définir une variable pointeur sur le type mentionné.
L'intérêt de cette syntaxe est de permettre à l'analyseur syntaxique de vérifier les affectations. On ne peut plus faire pointer un pointeur d'entier vers une chaîne de caractère, ou affecter un pointeur sur un segment A à un pointeur sur un segment B, le contrôleur est aussi capable de valider la saisie des variables de segment.

> Comme c'est une définition de pointeurs il est obligatoire de les initialiser pour éviter les erreurs générales de protection.

Syntaxes

SEGMENT *nomsegment*
<type> nom-champ1 <(taille)> <[dimension] <[dimension] ...> >
<type> nom-champ2 <(taille)> <[dimension] <[dimension] ...> >
...
ENDSEGMENT

ou

SEGMENT *nomsegment*
<type@> nom-champ1 <(taille)> <[dimension] <[dimension] ...> >
<type@> nom-champ2 <(taille)> <[dimension] <[dimension] ...> >
...
ENDSEGMENT

Ici, < et > signifient un champ optionnel, alors que [et] sont réellement les caractères crochets.

1. Attention, cette définition est virtuelle : nomsegment n'est pas une variable, mais un nouveau type de variable. Pour utiliser une variable qui sera de type nomsegment, il suffit de faire ensuite :

```
LOCAL nomsegment nomvariable
```

ou :

```
GLOBAL nomsegment nomvariable
```

Si vous préférez utiliser l'adresse pour travailler avec la variable, il faut faire :

```
NEW nomsegment, adressesegment
```

2. Les types acceptés sont INT (entier) ou INTEGER, NUM (réel), POINTER, CHAR (caractère), STRING (chaîne), CSTRING (chaîne C), CONTROL (contrôle) ou un segment préalablement défini.

3. Il est possible d'imbriquer des segments, puisque type peut être un nom de segment (cf. Exemple 3). Il est par contre interdit d'employer des imbrications directes :

```
SEGMENT seg1
...
SEGMENT seg2 ; Interdit !
...
ENDSEGMENT
...
ENDSEGMENT
```

4. Les déclarations situées entre SEGMENT et ENDSEGMENT ont la même syntaxe que LOCAL et GLOBAL (en ôtant bien sûr le mot-clé LOCAL et GLOBAL). La seule différence est que nomvariable de LOCAL et GLOBAL ne correspondent pas ici à des variables mais à des champs de la structure : il n'est pas obligatoire de terminer les noms de ces champs par un caractère spécial correspondant à son type (%, £, # ou $).

5. Les définitions SEGMENT doivent être placées soit dans une librairie .NCL, soit dans un événement INIT d'une fenêtre, soit dans une ressource de type Segment. Si une définition ne respecte pas cette règle (située dans un événement EXECUTED d'un Push-Button 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.

6. Il est possible de ne pas spécifier la taille d'un tableau inclus dans un segment. Dans ce cas, le tableau doit impérativement être le dernier champ du segment.

```
Segment typeSeg3
…
Int words(2)[] ; pas d'autre champ possible en dessous !
EndSegment ; typeSeg3
```

7. Mais, comme la taille du segment devient variable (en fonction de la taille réelle du tableau), ce type de segment ne peut plus être utilisé que pour un cast (typeSeg3(ptr)) ou bien là où les tableaux sans taille prédéfinie peuvent être déclarés, c'est-à-dire comme pointeur typé ou comme dernier champ d'un autre segment (pour lequel ces restrictions s'imposent alors également).

8. Si vous soumettez un SEGMENT terminé par un tableau sans taille, ou un SEGMENT contenant des pointeurs typés à une requête SQL, le message d'erreur suivant apparaît : "Forbidden POINTER found (define environment var. NSSQLRELAXED=PTR or ALL to disable this check)". La variable d'environnement NSSQLRELAXED peut être positionné à ALL, ou à un ou plusieurs des trois éléments suivants séparés par le caractère "+" : PTR, REF et DIM0 (pour autoriser les POINTER, les pointeurs typés et/ou les tableaux sans taille). Si vous soumettez à une requête SQL un SEGMENT contenant des Dynstr, l'application plante.

Exemple 1 :

```
SEGMENT IDENTITE ; Déclaration du segment IDENTITE
STRING NOM(32) ; (32 caractères maxi dans les
STRING PRENOM(32) ; chaînes NOM et PRENOM)
INT ANNEE_MARIAGE
ENDSEGMENT

GLOBAL IDENTITE MOI_MEME ; Déclaration de deux variables
GLOBAL IDENTITE MON_EPOUSE ; de type IDENTITE

MOVE "DESPIERRES" TO MOI_MEME.NOM ; Affectation de MOI_MEME
MOVE "Roger" TO MOI_MEME.PRENOM
MOVE 1973 TO MOI_MEME.ANNEE_MARIAGE

MOVE MOI_MEME TO MON_EPOUSE ; MOI_MEME et MON_EPOUSE avons
; le même nom, et la même année
; de mariage

MOVE "Françoise" TO MON_EPOUSE.PRENOM ; mais un prénom
; différent !
```

Exemple 2 :

```
SEGMENT IDENTITE ; Déclaration du segment IDENTITE
STRING NOM(32)
STRING PRENOM(32)
INT ANNEE_MARIAGE
ENDSEGMENT

NEW IDENTITE, MOI_MEME% ; Allocation mémoire
; Rigoureusement identique à
; GLOBAL IDENTITE MOI_MEME

MOVE "DESPIERRES" TO IDENTITE(MOI_MEME%).NOM
; Affectation de MOI_MEME
MOVE "Roger" TO IDENTITE(MOI_MEME%).PRENOM
MOVE 1973 TO IDENTITE(MOI_MEME%).ANNEE_MARIAGE
...
DISPOSE MOI_MEME% ; Libération mémoire
```

Exemple 3 :

```
SEGMENT POINT
INT X
INT Y
```

```
ENDSEGMENT

SEGMENT RECTANGLE
POINT DOWNLEFT
POINT UPRIGHT
ENDSEGMENT

GLOBAL RECTANGLE RECT1
MOVE WIDTH% TO 100
MOVE HEIGHT% TO 200

MOVE 10 TO RECT1.DOWNLEFT.X
MOVE 20 TO RECT1.DOWNLEFT.Y
MOVE RECT1.DOWNLEFT TO RECT1.UPRIGHT
MOVE RECT1.UPRIGHT.X + HEIGHT% TO RECT1.UPRIGHT.X
MOVE RECT1.UPRIGHT.Y + WIDTH% TO RECT1.UPRIGHT.Y
```

Exemple 4 :

```
Segment Seg_EtatCivil
Prenom$(31)
Nom$(31)
DateDeNaissance%
CodePostal$(7)
Ville$(63)
EndSegment ; {Seg_EtatCivil}

Segment Seg_Adresse
No$(7)
NomRue$(35)
CodePostal$(7)
Ville$(63)
Pays$(15) ; unused
EndSegment ; {Seg_Adresse}

Segment SEG_Personne
Seg_EtatCivil@ EtatCivil
Seg_Adresse@ Adresse
SEG_Personne@ Suivante ; liste chaînée, remplace POINTER Suivante
EndSegment ; {SEG_Personne}

Segment s_Ville
Ville$(31)[410] ; kNbVille%
EndSegment ; {s_Ville}

Global s_Ville@ pVillePPB1
```

Voir aussi LOCAL, GLOBAL, SIZEOF, NEW, DISPOSE, INDEXES (Librairie NSDB)

Instruction SELECT

Sélectionne :

- une ligne dans un contrôle de type liste,
- une fenêtre à onglet dans un contrôle Classeur.

Syntaxe	**SELECT** *position* **FROM** *objet*

Pour un contrôle de type liste :

1. Un contrôle de type liste doit être l'un des suivants :
   a) Window classe List
   b) List-box
   c) Combo-box
   d) CBE-Field
2. La première ligne d'un contrôle de type liste a pour index 0 (zéro).

Dans le cas d'une MLE ou d'une fenêtre de classe Edit, SELECT positionne le curseur à la ligne index.

3. Cette instruction génère un événement SELECTED reçu par le contrôle de type liste.

Les contrôles List Box, Combo Box et CBE ne peuvent pas gérer plus de 32766 lignes, ni un index de lignes supérieur à 32766.

Pour un contrôle Classeur :

1. La première fenêtre à onglet d'un contrôle Classeur a pour index 0 (zéro).
2. Cette instruction génère un événement SELECTED pour une fenêtre à onglet dans un contrôle Classeur.

Exemple 1 :

```
; Pour un contrôle de type liste :
SELECT 0 FROM TEST ; Sélectionne le premier
; élément de la liste TEST
```

Exemple 2 :

```
; Pour un contrôle Classeur :
SELECT 0 FROM CC_TAB ; Sélectionne la première
; fenêtre à onglet de CC_TAB
```

Voir aussi pour un contrôle de type liste : ISSELECTED%, LINECOUNT%, SELECTION%, UNSELECT, Evénement SELECTED

Voir aussi pour un contrôle Classeur : LINECOUNT%, SELECTION%, Evénement SELECTED

Fonction SELECTION%

Retourne l'index :

- de la ligne sélectionnée dans un contrôle de type liste,
- de la fenêtre à onglet sélectionnée dans un contrôle Classeur.

Syntaxe	**SELECTION%** *[(objet)]*

1. Pour un contrôle de type liste :
2. Un contrôle de type liste doit être l'un des suivants :
   a) Window classe List
   b) List-box
   c) Combo-box
   d) CBE-Field
3. La première ligne d'un contrôle de type liste a pour index 0 (zéro).
4. La fonction retourne NOSELECTION% si aucune ligne n'est sélectionnée.
5. Dans le cas d'une MLE ou d'une fenêtre de classe Edit, SELECTION% indique la ligne où est situé le curseur.
6. Si le paramètre objet n'est pas spécifié, la fonction retourne le numéro de ligne sélectionnée de la fenêtre courante.
7. Pour un contrôle Classeur : la première fenêtre à onglet d'un contrôle Classeur a pour index (zéro).

Exemple 1 :

```
; Pour un contrôle de type liste :
Local I%(1)

; Insère une ligne "Test" devant la ligne
; sélectionnée de la list-box LISTE

MOVE SELECTION%(LISTE) TO I%
IF I% <> NOSELECTION%
INSERT AT I% "Test" TO LISTE
ENDIF
```

Exemple 2 :

```
; Pour un contrôle Classeur :
Local I%(1)

; Retourne le numéro de la fenêtre
; à onglet courante (CC_TAB)

I%=SELECTION%(CC_TAB)
```

Voir aussi pour un contrôle de type liste : ISSELECTED%, LINECOUNT%, SELECT, UNSELECT, NOSELECTION%, Evénement SELECTED (PARAM1% est équivalent à SELECTION%)

Voir aussi pour un contrôle Classeur : LINECOUNT%, SELECT

Contrôle SELF

Contrôle système utilisable au sein des évènements d'un contrôle template (et des contrôles qu'il contient).

Syntaxe	SELF

1. Ce contrôle permet de passer un template à une fonction ou instruction qui attend un paramètre de type CONTROL.
La fonction ou instruction agit sur le premier contrôle du template (défini comme First Control dans les propriétés du template), sauf surcharge des événements de comportement du template, qui permettent de définir un comportement sur-mesure

2. Il permet également de désigner le control-template à partir d'un contrôle défini dans le template.

3. Ne pas confondre avec la fonction SELF% (avec suffixe %). Dans les événements d'un template, SELF% désigne la fenêtre qui contient le template, même indirectement
(soit un template B utilisé dans un template A qui est lui-même utilisé dans une fenêtre W, alors aussi bien dans les événements de A que de B, self% correspond à la fenêtre W).

Exemple 1 :

```
; Déclaration d'une fonction
FUNCTION FCHANGEENABLE% (CONTROL CTRL)
; FCHANGEENABLE change l'état du contrôle passé
; en paramètre
;(effectue un ENABLE si inactif, DISABLE si actif)
IF ISDISABLED%(CTRL)
ENABLE CTRL
RETURN TRUE%
ELSE
DISABLE CTRL
RETURN FALSE%
ENDIF
ENDFUNCTION

; Le template comprend deux contrôles PUSH-BUTTON
; intitulés PB01 et PB02.
; PB01 est le premier contrôle du template.
; Evénement EXECUTED du PUSH-BUTTON PB02 du template  :
IF FCHANGEENABLE%(SELF)
MOVE "DESACTIVER" TO PB02
ELSE
MOVE "ACTIVER" TO PB02
ENDIF

; l'appui de PB01 entraîne l'appel de la fonction
; FCHANGEENABLE% qui désactive ou réactive PB01,
; ainsi qu'un changement du texte du PUSH_BUTTON PB02.
```

Exemple 2 :

```
; Evénement LOSEFOCUS d'un contrôle
; défini dans un template
SEND CHECK TO SELF
; La perte du focus par le contrôle entraine
; l'événement CHECK du control-template
```

Voir aussi CONTROL, FUNCTION, SELF%

Fonction SELF%

Retourne le handle de la fenêtre à laquelle appartient l'objet qui effectue le SELF%.

Syntaxe	SELF%

1. Cette fonction ne peut être employée dans une librairie NCL, en dehors d'un script événement. Pour agir sur la fenêtre courante depuis une librairie NCL, il est nécessaire de passer le handle en paramètre à la fonction ou instruction : voir exemple 3.
2. Ne pas confondre avec SELF (sans suffixe %).

Exemple 1 :

```
; Ouverture d'une fenêtre fille de la fenêtre courante
OPEN SCREEN, SELF%
```

Exemple 2 :

```
; Autre exemple
CLOSE ; Rigoureusement équivalent à CLOSE SELF%
```

Exemple 3 :

```
; Pour agir sur la fenêtre courante depuis une librairie
; Exemple incorrect
INSTRUCTION MYOPEN
 OPEN SCREEN, SELF% ; Erreur "No meaning outside a script"
ENDINSTRUCTION

MYOPEN ; Appel de l'instruction précédente

; Exemple correct
INSTRUCTION MYOPEN H%
 OPEN SCREEN, H%
ENDINSTRUCTION

MYOPEN SELF% ; Appel de l'instruction précédente
```

***Voir aussi** MAINWINDOW%, PARENTWINDOW%, SELF*

Instruction SEND

Envoie un événement à un objet.

Syntaxes	**SEND** *événement [, param1 [, param2 [, param3 [, param4]]]] [TO objet [, variable-retour]]* **SEND** *événement [, param1 [, param2 [, param34]]] [TO objet [, variable-retour]]* **SEND** *événement [, param12 [, param3 [, param4]]] [TO objet [, variable-retour]]* **SEND** *événement [, param12 [, param34]] [TO objet [, variable-retour]]*

1. Les paramètres Param1, Param2, Param3, Param4, Param12, Param34 seront reçus dans les paramètres PARAM1%, PARAM2%, PARAM3%, PARAM4%, PARAM12%, PARAM34% de l'événement envoyé de l'objet destination.

2. La signification des paramètres 1 à 4 dépend du traitement éventuel de PARAM1% à PARAM4% fait par l'événement appelé.

3. Si aucun objet n'est spécifié, l'événement est envoyé à la fenêtre courante.

4. Le paramètre variable-retour est optionnel. Il retourne un entier dans la variable spécifiée, à la condition que l'événement appelé ait fait un RETURN d'une valeur.

5. Lorsque le nom de la fenêtre à laquelle appartient l'objet n'est pas précisé (par SEND ... TO H%), l'événement est envoyé à la fenêtre de même nom que la fenêtre courante.

6. Il s'agit d'un appel DIRECT de la routine associée à l'événement : après cet appel, l'exécution revient à l'instruction qui suit le SEND.

7. Pour des raisons de récursivité infinie, il faut éviter d'envoyer un SEND de l'événement qui est en train d'être traité !

8. Les fonctions PARAM12% et PARAM34% sont désormais des pointeurs.

9. Dans les emplacements prévus pour les paramètres PARAM1% et PARAM3%, il est possible de remplacer les opérateurs LOW expr_ent%, HIW expr_ent% par POINTER expr_ent%.

10. De même, si une expression de type ptr, ptr+expr_ent%, ptr-expr_ent% ou expr_ent%+ptr (où ptr est soit une variable de type POINTER, soit l'adresse d'une variable ou d'une chaîne littérale) apparaît à l'un des emplacements de PARAM1% et PARAM3% le pointeur est passé comme PARAM12% ou PARAM34% sans même avoir besoin de le préfixer par le type POINTER.

11. Si les expressions passées en paramètre sont de type POINTER, alors l'instruction SEND utilise PARAM12% au lieu de PARAM1% et PARAM2%, ou PARAM34% au lieu de PARAM3% et PARAM4%.

12. Si une expression de type POINTER est passée sans changement de type en INT aux emplacements prévus pour PARAM2% ou PARAM4%, vous obtiendrez un avertissement ou une erreur.

Exemple 1 :

```
; Envoie l'événement EXECUTED à l'objet PUSHBUTTON.
SEND EXECUTED TO PUSHBUTTON
```

Exemple 2 :

```
; Envoie l'événement PAINT à la fenêtre WINDOW
; dont le handle est H%
SEND PAINT POINTER PS% TO H%
```

Exemple 3 :

```
; Exemples avec code retour
; Si l'événement EXECUTED d'un Push-Button PUSH contient  :
RETURN 12

; Alors tout événement de tout autre contrôle peut faire  :
SEND EXECUTED TO PUSH, I%
; et I% vaut 12
```

Voir aussi :

POST, PASS, Fonctions PARAM*%, PARAM*$

Tous les événements avec leurs différents PARAMx%

Fonction SENDMESSAGE% et Instruction POSTMESSAGE (Librairie NSWin sous Windows)

Instruction SETBACKCOL

Permet de définir la couleur d'arrière-plan d'un objet ou la couleur de remplacement de la couleur « transparente » des bitmaps.

Syntaxe	**SETBACKCOL** *couleur [TO objet]*

1. La couleur doit être une des constantes COL_*% ou TRANSP_*%.
2. Si l'objet n'est pas précisé, la modification de couleur s'applique à la fenêtre courante.

Exemple :

```
SETBACKCOL COL_RED% TO ENTRY00001 ; L'arrière-plan de l'objet est maintenant rouge
```

Voir aussi : ***SETFORECOL*****,** ***GETBACKCOL%*****,** ***GETFORECOL%*****, Paramétrage dynamique** ***.BACKCOLOR***

Instruction SETDATA

Associe un pointeur à une fenêtre.

Syntaxe	**SETDATA** *pointeur [TO handle-fenêtre]*

1. Cette instruction permet de manipuler des données instanciées, locales à une fenêtre. Cela est important par exemple pour des applications MDI, où la même fenêtre peut être démarrée plusieurs fois : le code est unique, mais chacune des fenêtres doit manipuler des données différentes.
2. Le pointeur associé peut ensuite être obtenu à l'aide de GETDATA%.
3. Il y a UN SEUL pointeur associé par fenêtre. Un second SETDATA sur une même
fenêtre écrase donc le pointeur préalablement associé.
4. Plusieurs données peuvent néanmoins être associées à une fenêtre. Dans ce cas, elles doivent appartenir à une même zone mémoire, définie à l'aide d'un segment alloué par NEW. Le pointeur précisé après SETDATA est alors l'adresse de cette zone mémoire.

Exemple :

```
; Evénement INIT
; Un segment INSTDATA différent est alloué à chaque
; ouverture de cette même fenêtre. Le pointeur associé est
; ici utilisé pour sauvegarder l'adresse de ce segment.

LOCAL POINTER PDATA

SEGMENT INSTDATA
 CSTRING MACHAINE
ENDSEGMENT

NEW INSTDATA, PDATA

SETDATA PDATA

; Evénement quelconque
; écriture d'une donnée locale à la fenêtre courante
MOVE ENTRY00001 TO INSTDATA(GETDATA%).MACHAINE

; Autre événement quelconque
; lecture d'une donnée locale à la fenêtre courante
MESSAGE "Donnée locale", INSTDATA(GETDATA%).MACHAINE

; Evénement TERMINATE
DISPOSE PDATA
```

***Voir aussi** GETDATA%, LOCAL, NEW , Stockage de données*

Instruction SETFOCUS

Donne le focus à un objet.

Syntaxe	**SETFOCUS** *[objet]*

1. Cette instruction a pour effet de générer un événement LOSEFOCUS sur l'objet qui possédait préalablement le focus, puis un événement GETFOCUS sur le nouvel objet.
2. Ne pas employer cette instruction depuis un événement GETFOCUS ou LOSEFOCUS.
3. Si l'objet n'est pas spécifié, le focus est donné à la fenêtre courante.
4. Lorsqu'un objet obtient le focus, toutes les actions de l'utilisateur ont pour conséquence l'envoi d'événements vers cet objet.

Exemple 1 :

```
SETFOCUS ;Donne le focus à la fenêtre courante
```

Exemple 2 :

```
SETFOCUS TEST% ;Donne le focus à la fenêtre dont le handle est TEST%
```

Exemple 3 :

```
SETFOCUS TEST ;Donne le focus au contrôle TEST dans la fenêtre courante
```

Exemple 4 :

```
SETFOCUS WINDOW(I%).TEST ;Donne le focus au contrôle TEST dans la fenêtre WINDOW ayant le
handle I%
```

Voir aussi Evénements GETFOCUS, LOSEFOCUS, FOCUSCONTROL, SETCHECKMODE, GETCHECKMODE% de la librairie NSMisc

Instruction SETFORECOL

Modifie la couleur d'avant-plan d'un objet ou la couleur de transparence des bitmaps.

Syntaxe	**SETFORECOL** *couleur [TO objet]*

1. La couleur doit être une des constantes COL_*% ou TRANSP_*%.
2. La couleur d'avant-plan s'applique au texte affiché dans l'objet
3. Si l'objet n'est pas précisé, la modification de couleur s'applique à la fenêtre courante si celle-ci est de classe Edit ou List.

Exemple :

```
SETFORECOL COL_BLUE% TO ENTRY00001 ; Les caractères affichés dans l'entry field sont
maintenant en bleu
```

***Voir aussi** SETBACKCOL, GETBACKCOL%, GETFORECOL%, **Paramétrage dynamique** .FORECOLOR*

Instruction SETPOS

Positionne un objet.

Syntaxe	**SETPOS** *posx, posy [TO objet]*

1. Les coordonnées à spécifier sont celles du coin inférieur gauche de l'objet. Elles doivent être données en pixels.
2. Si l'objet est un contrôle, ses coordonnées sont relatives à la fenêtre à laquelle il appartient.
3. Si l'objet est une fenêtre fille, ses coordonnées sont relatives à sa fenêtre mère, sinon elles sont relatives à l'écran.
4. Si l'objet n'est pas spécifié, le positionnement s'applique à la fenêtre courante.

Exemple :

```
; Déplace la fenêtre courante de 10 pixels vers le haut et vers la droite
SETPOS GETXPOS% + 10, GETYPOS% + 10
```

***Voir aussi** GETXPOS%, GETYPOS%, CHANGE, Paramétrages dynamiques .X et .Y*

Instruction SETPTR

Définit un nouveau pointeur souris.

Syntaxe	**SETPTR** *handle-pointeur*

handle-pointeur est le handle d'un pointeur retourné par les fonctions GETSPTR% (NCL) ou CREATEPTR% (Librairie NSGraph).

Exemple :

```
; Sur événement MOUSEMOVE
SETPTR GETSPTR%(SPTR_WAIT%) ; Le pointeur souris est changé en sablier
```

Voir aussi GETSPTR%, SPTR_*%, CREATEPTR%, DELETEPTR (Librairie NSGraph)

Instruction SETRANGE

Définit la nouvelle position et les positions minimum et maximum d'un ascenseur dans une barre de défilement.

Syntaxe	**SETRANGE** *position, mini, maxi* **TO** *contrôle*

1. Le contrôle peut être un contrôle Horizontal Scroll Bar ou Vertical Scroll-Bar, ainsi qu'une barre de défilement associée à une fenêtre (style "Horz Scroll-Bar" ou "Vert Scroll-Bar" coché dans le volet de propriétés de la fenêtre).
2. position doit avoir une valeur comprise entre mini et maxi
3. Tant que SETRANGE n'a pas été appelé, les minimum et maximum valent, par défaut : mini = 0, maxi = 100 sauf si, dans le cas des contrôles scroll-bars, ils ont été redéfinis dans le volet de propriétés.
4. Le paramètre mini correspond à la valeur obtenue lorsque l'ascenseur est situé en son point le plus haut (pour une barre verticale) ou le plus à gauche (pour une barre horizontale).
5. Le paramètre maxi est la valeur obtenue lorsque l'ascenseur est situé en son point le plus bas (pour une barre verticale) ou le plus à droite (pour une barre horizontale).
6. Ces positions sont retournées dans PARAM1% de l'événement SELECTED ou dans PARAM2% des événements HSCROLL et VSCROLL.
7. Pour spécifier la barre de défilement associée à une fenêtre (style "Horz Scroll-Bar" ou "Vert Scroll-Bar" coché dans le volet de propriétés de la fenêtre), il faut nommer le contrôle :

```
[nom-fenêtre (handle-fenêtre).]HSCROLL|VSCROLL
```

8. Cette instruction fonctionne aussi sur un Entry-Field ayant "Spin-Button" coché dans son volet de propriétés. Elle est alors équivalente à :

```
MOVE "position mini maxi incrément" TO contrôle
```

sauf que l'incrément n'est pas modifié par SETRANGE.

Exemple 1 :

```
SETRANGE 25, 0, 50 TO VSCROLL ; Redéfinit le mini et maxi de
; l'ascenseur vertical associé à la fenêtre courante (style "Vert Scroll-Bar" coché)
; et de plus l'ascenseur sera positionné en son milieu

; Rappel. Pour modifier uniquement la position, il suffit d'employer MOVE  :
MOVE 25 TO VSCROLL
```

Exemple 2 :

```
SETRANGE 25, 0, 50 TO TEST ; Idem, mais pour le contrôle TEST
```

Exemple 3 :

```
SETRANGE 0, 0, 1000 TO WINDOW(I%).TEST
```

Exemple 4 :

```
SETRANGE 0, 0, 1000 TO WINDOW(I%).HSCROLL
```

Exemple 5 :

```
SETRANGE TEST, 0, 80 TO TEST ; Modifie le mini et le maxi mais ne modifie pas la position
```

Voir aussi Evénements HSCROLL, VSCROLL, SELECTED

Instruction SETTEXT

Fixe le titre d'une fenêtre, le texte d'un élément de menu ou le texte associé à certains contrôles.

Syntaxe	**SETTEXT** *chaîne [TO objet]*

1. Si aucun objet n'est modifié, c'est le titre de la fenêtre courante qui est modifié.
2. SETTEXT modifie le texte des contrôles radio-button, check-box, push-button, entry-field, static-text, group-box et MLE.
3. Pour certains contrôles, l'instruction SETTEXT est équivalente à un MOVE. Voir l'exemple.
4. Un SETTEXT permet aussi de modifier dynamiquement les images affichées dans un Bitmap de type Check Box. Voir la partie "Bitmap" du chapitre Eléments du langage de ce manuel pour plus de détails.

Exemple 1 :

```
; Modifie le titre de la fenêtre
SETTEXT "Nouveau titre"
```

Exemple 2 :

```
; Si ENTRY00001 est un Entry-Field, l'exemple suivant
; est équivalent à faire un MOVE "Bonjour" TO ENTRY0000
SETTEXT "Bonjour" TO ENTRY00001
```

Exemple 3 :

```
; Modifie un menu
SETTEXT "~Open..." TO MENUITEM00001
```

Voir aussi GETTEXT$, Paramétrage dynamique .TEXT

Instruction SHOW

Rend visible un objet caché.

Syntaxe	**SHOW** *[objet]*

1. Cette instruction doit également être utilisée après un CALLH ou un OPENH pour rendre une fenêtre visible.
2. Si aucun objet n'est spécifié, l'instruction s'applique à la fenêtre courante.

Exemple :

```
SHOW ; Rend visible la fenêtre courante
SHOW TEST% ; Rend visible la fenêtre dont le handle est TEST%
SHOW TEST ; Rend visible le contrôle TEST% dans la
;fenêtre courante
SHOW WINDOW(I%).TEST ; Rend visible le contrôle TEST dans
; la fenêtre WINDOW ayant le handle I%
```

Voir aussi :

HIDE, ISHIDDEN%

CALLH (cf. CALL), STRCALLH (cf. STRCALL*), OPENH (cf. OPEN*), STROPENH (cf. STROPEN*), Paramétrage dynamique .HIDDEN

Opérateur SIZEOF

Retourne la taille en octets d'une variable.

Syntaxe	**SIZEOF** *variable* | *nomsegment*

Ne pas confondre SIZEOF (taille mémoire occupée en octets par une variable) et LENGTH (nombre de caractères d'une chaîne).

```
LOCAL S$
MOVE "Longueur" TO S$
MESSAGE "SizeOf ="&& SIZEOF S$,"Length ="&& LENGTH S$
; Affiche "SizeOf = 256" et "Length = 8"
```

Exemple 1 :

```
MOVE SIZEOF I% TO A% ; A% vaut 1, 2 ou 4
MOVE SIZEOF N£ TO B% ; B% vaut 4, 8 ou 10
MOVE SIZEOF S$ TO C% ; C% vaut 256 ou toute autre valeur
; si S$ a été déclarée par STRING.
```

Exemple 2 :

```
SEGMENT POINT
INT X
INT Y
ENDSEGMENT

LOCAL POINT PT
NEW POINT, HPT%

MOVE SIZEOF POINT TO D% ; D% vaut 8
MOVE SIZEOF PT TO E% ; E% vaut 8
MOVE SIZEOF POINT(HPT%) TO F% ; F% vaut 8
```

***Voir aussi** LOCAL, GLOBAL, SEGMENT, FILL, MOV, LENGTH*

Opérateur SKIP

Enlève les éventuels espaces et tabulations situés en tête et en fin de chaîne.

Syntaxe	**SKIP** *chaîne*

Exemple :

```
MOVE SKIP " un exemple " TO S$ ; S$ vaut "un exemple"
```

Voir aussi LSKIP, RSKIP

Constantes SPTR_*%

Syntaxe	**SPTR_APPICON%** **SPTR_ARROW%** **SPTR_BANGICON%** **SPTR_CPTR%** **SPTR_FILE%** **SPTR_FOLDER%** **SPTR_HANDICON%** **SPTR_ICONERROR%** **SPTR_ICONINFORMATION%** **SPTR_ICONQUESTION%** **SPTR_ICONWARNING%** **SPTR_ILLEGAL%** **SPTR_MOVE%** **SPTR_MULTFILE%** **SPTR_NOTEICON%** **SPTR_PROGRAM%** **SPTR_QUESICON%** **SPTR_SIZENESW%** **SPTR_SIZENS%** **SPTR_SIZENWSE%** **SPTR_SIZEWE%** **SPTR_TEXT%** **SPTR_WAIT%**

Leur déclaration interne est :

```
CONST SPTR_ARROW% 1
CONST SPTR_TEXT% 2
CONST SPTR_WAIT% 3
CONST SPTR_MOVE% 5
CONST SPTR_SIZENWSE% 6
CONST SPTR_SIZENESW% 7
CONST SPTR_SIZEWE% 8
CONST SPTR_SIZENS% 9
CONST SPTR_APPICON% 10
CONST SPTR_ICONINFORMATION% 11
CONST SPTR_ICONQUESTION% 12
CONST SPTR_ICONERROR% 13
CONST SPTR_ICONWARNING% 14
CONST SPTR_CPTR% 14
; Identique à SPTR_ICONWARNING%
CONST SPTR_ILLEGAL% 18
CONST SPTR_FILE% 19
CONST SPTR_FOLDER% 20
CONST SPTR_MULTFILE% 21
```

```
CONST SPTR_PROGRAM% 22
CONST SPTR_HANDICON% 11
; Identique à SPTR_ICONINFORMATION%
CONST SPTR_QUESICON% 12
; Identique à SPTR_ICONQUESTION%
CONST SPTR_BANGICON% 13
; Identique à SPTR_ICONERROR%
CONST SPTR_NOTEICON% 14
; Identique à SPTR_ICONWARNING%
```

Exemple 1

```
; Affiche le pointeur SPTR_FILE% dans un contrôle icône
MOVE SPTR_FILE% TO ICON00001
```

Exemple 2

```
; Affiche le pointeur SPTR_FILE% dans un contrôle bitmap
MOVE "#" & SPTR_FILE% TO BITMAP00001
```

Voir aussi

GETSPTR%, SETPTR

CREATEPTR%, DELETEPTR (Librairie NSGraph)

Constantes MB_*% (Librairie NSMisc)

Instruction STARTTIMER

Démarre un timer avec une temporisation donnée, précise en milli-secondes.

Syntaxe	**STARTTIMER** *numéro, tempo [TO handle-fenêtre]*

1. L'événement TIMER est ensuite envoyé régulièrement toutes les tempo milli-secondes à la fenêtre spécifiée ou, par défaut, à la fenêtre courante.
2. numéro identifie le timer. Cela permet à une fenêtre de démarrer plusieurs timers. Ils peuvent ensuite être distingués entre eux grâce au PARAM1% de l'événement TIMER : numéro est intégralement repassé dans ce paramètre.
3. Lors de l'utilisation d'un événement DDE, ne pas utiliser le TIMER 1, car ce dernier est utilisé par le protocole DDE.

Exemple :

```
; **** Evénément INIT

STARTTIMER 1, 1000

; **** Evénement TIMER

; Bip sonore émis à chaque seconde
BEEP

; **** Evénement TERMINATE

STOPTIMER 1
```

Voir aussi STOPTIMER, Instruction WAIT, Evénement TIMER, Librairie NSdate

Instruction STOPTIMER

Arrête un timer démarré au préalable avec STARTTIMER.

Syntaxe	**STOPTIMER** *numéro [TO handle-fenêtre]*

1. Lorsque le handle de la fenêtre n'est pas spécifié, STOPTIMER agit sur la fenêtre courante.
2. STOPTIMER n'arrête que le timer précisé par numéro : l'événement TIMER peut continuer à arriver si la fenêtre avait démarré d'autres timers.

Exemple :

```
; **** Evénément INIT

STARTTIMER 1, 1000

; **** Evénement TIMER

; Bip sonore émis à chaque seconde
BEEP

; **** Evénement TERMINATE

STOPTIMER 1
; L'événement TIMER n'est plus reçu
```

Voir aussi STARTTIMER, Instruction WAIT, Evénement TIMER, Librairie Nsdate

Instructions STRCALL*

Ouvre une fenêtre, dont le nom est passé sous forme de chaîne de caractères, en mode modal.

Syntaxe	**STRCALL,** **STRCALLH** *chaîne-nom-fenêtre* *[,handle-fenêtre]* *[USING* *[Param1* *[,* *Param2* *[,Param3* *[,Param4[,variable-retour]]]]]]*
	STRCALL, **STRCALLH** *chaîne-nom-fenêtre* *[,handle-fenêtre]* *[USING* *[Param1* *[,* *Param2* *[,Param34* *[,variable-retour]]]]]*
	STRCALL, **STRCALLH** *chaîne-nom-fenêtre* *[,handle-fenêtre]* *[USING* *[Param12* *[,* *Param3* *[,Param4* *[,variable-retour]]]]]*
	STRCALL, **STRCALLH** *chaîne-nom-fenêtre* *[,handle-fenêtre]* *[USING* *[Param12* *[,* *Param34* *[,variable-retour]]]]*

1. Le paramètre variable-retour, entier sur 2 octets, sera la valeur retournée par l'instruction CLOSE.

2. Les paramètres Param1, Param2, Param3, Param4, Param12, Param34 seront reçus dans les fonctions PARAM1%, PARAM2%, PARAM3%, PARAM4%, PARAM12%, PARAM34% de l'événement INIT de la fenêtre ouverte.

3. L'instruction STRCALLH a un comportement un peu différent de STRCALL car elle laisse cachée la fenêtre que l'on ouvre. Un emploi ultérieur de l'instruction SHOW permet de la faire apparaître.

4. Mis à part le passage du nom de la fenêtre sous forme de chaîne de caractères, ces instructions ont le même comportement que les instructions CALL et CALLH. Veuillez vous reporter aux notes accompagnant la description de ces instructions pour plus de détails sur leur utilisation.

5. Les fonctions PARAM12% et PARAM34% sont désormais des pointeurs.

6. Dans les emplacements prévus pour les paramètres PARAM1% et PARAM3%, il est possible de remplacer les opérateurs LOW expr_ent%, HIW expr_ent% par POINTER expr_ent%.

7. De même, si une expression de type ptr, ptr+expr_ent%, ptr-expr_ent% ou expr_ent%+ptr (où ptr est soit une variable de type POINTER, soit l'adresse

d'une variable ou d'une chaîne littérale) apparaît à l'un des emplacements de PARAM1% et PARAM3% le pointeur est passé comme PARAM12% ou PARAM34% sans même avoir besoin de le préfixer par le type POINTER.

8. Si les expressions passées en paramètre sont de type POINTER, alors les instructions STRCALL* utilisent PARAM12%au lieu de PARAM1% et PARAM2% ou PARAM34% au lieu de PARAM3% et PARAM4%.

9. Si une expression de type POINTER est passé sans changement de type en INT aux emplacements prévus pour PARAM2% ou PARAM4%, vous obtiendrez un avertissement ou une erreur.

Exemple 1 :

```
STRCALL "test1" ; ouvre la fenêtre TEST1 (fichier TEST.SCR)
```

Exemple 2 :

```
LOCAL NomFen$(40)
MOVE "test2" TO NomFen$
STRCALL NomFen$ ; ouvre la fenêtre TEST2
```

Exemple 3 :

```
; Supposons une List Box LB_FEN contenant des noms de fenêtres
STRCALL LB_NOMFEN[4] ; Ouvre la fenêtre dont le nom figure en 5ème ligne
```

***Voir aussi** CALL, CALLH , Fonctions PARAM*%, PARAM*$*

Opérateur STRING

Convertit en chaîne (format Pascal).

Syntaxe	**STRING** *entier* | *réel* | *chaîne* | *(entier, base)*

1. STRING peut aussi être utilisé en tant que "type" au sein des instructions LOCAL, GLOBAL, SEGMENT, FUNCTION et INSTRUCTION.
2. En NCL, le type STRING ou CSTRING est indifférent, c'est pour cela qu'il n'existe qu'un opérateur de conversion en chaîne de caractères.
3. Les paramètres entier et base permettent de convertir un nombre décimal en une autre base (2, 8 ou 16).

Exemple :

```
s$ = STRING 1.3       ; S$ vaut "1.3"
s$ = STRING 12        ; S$ vaut "12"
s$ = STRING (7676,16) ; S$ vaut "1DFC"
```

Voir aussi** CSTRING, NUM, INT, LOCAL, GLOBAL, SEGMENT, FUNCTION, INSTRUCTION, **STRING$ de la Librairie NSMisc

Type STRING

Type chaîne au format Pascal, utilisé au sein des instructions LOCAL, GLOBAL, SEGMENT, FUNCTION et INSTRUCTION.

1. Une chaîne STRING est précédée d'un octet mémorisant le nombre de caractères la constituant. Sa taille interne est donc supérieure de un octet à celle précisée par LOCAL ou GLOBAL.
2. La taille maximale d'une chaîne de type STRING est de 255 caractères.
3. Pour une chaîne, le type STRING ou CSTRING est indifférent en NCL. Par contre il est important pour un interfaçage avec une DLL, selon que cette dernière ait été écrite en Pascal ou en C.

Voir aussi ***INT, NUM, CHAR, CSTRING, CONTROL, DYNSTR, LOCAL, GLOBAL, SEGMENT, FUNCTION, INSTRUCTION***

Instructions STROPEN*

Ouvre une fenêtre, dont le nom est passé sous forme de chaîne de caractères, en mode non modal.

Syntaxes	**STROPEN, STROPENH, STROPENS, STROPENSH** *chaîne-nom-fenêtre, 0 | handle-fenêtre-parent, [, handle-fenêtre] [USING Param1 [, Param2 [, Param3 [, Param4]]]]* **STROPEN, STROPENH, STROPENS, STROPENSH** *chaîne-nom-fenêtre, 0 | handle-fenêtre-parent, [, handle-fenêtre] [USING Param1 [, Param2 [, Param34]]]* **STROPEN, STROPENH, STROPENS, STROPENSH** *chaîne-nom-fenêtre, 0 | handle-fenêtre-parent, [, handle-fenêtre] [USING Param12 [, Param3 [, Param4]]]* **STROPEN, STROPENH, STROPENS, STROPENSH** *chaîne-nom-fenêtre, 0 | handle-fenêtre-parent, [, handle-fenêtre] [USING Param12 [, Param34]]*

1. Le second paramètre est soit 0 (dans ce cas, la fenêtre créée n'est pas fille), soit le handle-fenêtre de la fenêtre parent (la fenêtre créée est fille). Le troisième paramètre optionnel est une variable entière dans laquelle est retourné le handle de la fenêtre. Les paramètres Param1, Param2, Param3, Param4, Param12, Param34 seront reçus dans les fonctions PARAM1%, PARAM2%, PARAM3%, PARAM4%, PARAM12%, PARAM34% de l'événement INIT de la fenêtre ouverte.

2. STROPENH a un comportement un peu différent de STROPEN car il laisse caché la fenêtre que l'on ouvre. Un emploi ultérieur de l'instruction SHOW permet de la faire apparaître.

3. STROPENS crée une fenêtre non-modale secondaire. STROPENSH crée une fenêtre non-modale secondaire cachée.

4. Mis à part le passage du nom de la fenêtre sous forme de chaîne de caractères, ces instructions ont le même comportement que les instructions OPEN, OPENH, OPENS et OPENSH. Veuillez vous reporter aux commentaires

accompagnant la description de ces instructions pour plus de détails sur leur utilisation.

5. Les fonctions PARAM12% et PARAM34% sont désormais des pointeurs.
6. Dans les emplacements prévus pour les paramètres PARAM1% et PARAM3%, il est possible de remplacer les opérateurs LOW expr_ent%, HIW expr_ent% par POINTER expr_ent%.
7. De même, si une expression de type ptr, ptr+expr_ent%, ptr-expr_ent% ou expr_ent%+ptr (où ptr est soit une variable de type POINTER, soit l'adresse d'une variable ou d'une chaîne littérale) apparaît à l'un des emplacements de PARAM1% et PARAM3% le pointeur est passé comme PARAM12% ou PARAM34% sans même avoir besoin de le préfixer par le type POINTER.
8. Si les expressions passées en paramètre sont de type POINTER, alors les instructions STROPEN* utilisent PARAM12% au lieu de PARAM1% et PARAM2%, ou PARAM34% au lieu de PARAM3% et PARAM4%.
9. Si une expression de type POINTER est passé sans changement de type en INT aux emplacements prévus pour PARAM2% ou PARAM4%, vous obtiendrez un avertissement ou une erreur.

Exemple 1 :

```
STROPEN "PARENT", 0, PARENT% ; ouvre fenêtre principale PARENT (fichier PARENT.SCR)
```

Exemple 2 :

```
STROPENH EF_ENFANT, PARENT%, ENFANT% ; ouvre une fenêtre affiliée à PARENT.
; Le nom de la fenêtre est lu dans l'Entry Field EF_ENFANT
SHOW ENFANT% ; ENFANT devient visible
```

Exemple 3 :

```
STROPEN "SCREEN", SELF% ; Ouvre une fenêtre fille de la fenêtre courante
```

Voir aussi OPEN, OPENH, OPENS (cf. OPEN*), Fonctions PARAM*%, PARAM*$

Séparateur TO

Séparateur utilisé par certaines instructions.

Syntaxe	CHANGE [fenêtre] TO pos-x, pos-y [,largeur, hauteur] INSERT ASCENDING | DESCENDING chaîne TO objet INSERT AT position | END chaîne TO objet LOAD [ASCENDING | DESCENDING] nom-fichier TO objet MOVE expression TO variable|contrôle POST événement[, param1[, param2[, param3[, param4]]]] [TO objet] SAVE objet TO nom-fichier SEND événement[, param1[, param2[, param3[, param4]]]] [TO objet] [, variable-retour] SETBACKCOL couleur [TO objet] SETDATA entier [TO fenêtre] SETFORECOL couleur [TO objet] SETPOS posx, posy [TO objet] SETRANGE position, mini, maxi TO contrôle SETTEXT chaîne [TO objet] STARTTIMER numero, tempo [TO fenêtre] STOPTIMER numero [TO fenêtre] WHERE expression [TO expression], expression [TO expression],...

Voir aussi :

CHANGE, INSERT, LOAD, MOVE, POST, SAVE, SEND, SETBACKCOL, SETDATA, SETFORECOL, SETPOS, SETRANGE, SETTEXT, STARTTIMER, STOPTIMER, WHERE

Constante TRUE%

Valeur logique retournée par les expressions conditionnelles lorsque le résultat est vrai.

Syntaxe	TRUE%

Sa déclaration interne est :

```
CONST TRUE% 1
```

***Voir aussi** IF, FALSE%*

Fonction TRUNC%

Convertit un réel en entier.

Syntaxe	**TRUNC%** *(réel)*

Equivalent à l'opérateur INT appliqué à un opérande réel.

Exemple :

```
MOVE TRUNC% (1.7) TO I% ; met 1 dans I%
MOVE TRUNC% (1.3) TO I% ; met 1 dans I%
```

Voir aussi INT, ROUND%

Constantes TYP_*%

Utilisé pour le paramétrage dynamique .TYPE (format de saisie) d'un contrôle Entry Field ou Combo Box Entry Field.

Syntaxe	**TYP_DATE%** **TYP_DEFAULT%** **TYP_INTEGER%** **TYP_NUMBER%** **TYP_TIME%**

1. Signification des constantes :
   a) TYP_DATE% : Le champ est de type date
   b) TYP_DEFAULT% : Le champ n'a pas de type de saisie
   c) TYP_INTEGER% : Le champ est de type entier
   d) TYP_NUMBER% : Le champ est de type réel
   e) TYP_TIME% : Le champ est de type heure
2. Leur déclaration interne est :

```
CONST TYP_DEFAULT% 0
CONST TYP_INTEGER% 1
CONST TYP_NUMBER% 2
CONST TYP_DATE% 3
CONST TYP_TIME% 4
```

Exemple :

```
; Modifier le format de saisie de l'entry field ENTRY00001
; en format Date
MOVE TYP_DATE% TO ENTRY00001.TYPE
```

Voir aussi Paramètre dynamique .TYPE

Instruction UNCAPTURE

Les événements souris ne sont plus capturés par la fenêtre qui avait été spécifiée lors de l'instruction CAPTURE. Ils sont maintenant dirigés vers la fenêtre physiquement située sous le pointeur souris.

Syntaxe	**UNCAPTURE**

1. Si aucune fenêtre n'avait capturé la souris, cette instruction n'a aucun effet.
2. A un instant donné, une seule fenêtre peut capturer la souris. Ce qui explique que UNCAPTURE n'ait aucun paramètre.

Exemple :

```
UNCAPTURE ; aucune fenêtre ne capture la souris
```

Voir aussi Instruction CAPTURE, Evénements BUTTONUP, BUTTONDOWN, MOUSEMOVE

Constante UNCHECKED%

Valeur d'un contrôle check-box ou d'un menu lorsqu'il n'est pas coché.

Syntaxe	UNCHECKED%

Sa déclaration interne est :

```
CONST UNCHECKED% 0
```

Exemple :

```
MOVE UNCHECKED% TO MENUITEM00001 ; décoche l'élément de menu

IF CHECKBOX00001 = UNCHECKED%
 MESSAGE "Vérification", \
  "Le contrôle CHECKBOX00001 n'est pas coché"
ENDIF
```

Voir aussi Constantes CHECKED%, INDETERMINATE%

Instruction UNLOADDLL

Décharge une DLL précédemment chargée par l'instruction LOADDLL.

Syntaxe	**UNLOADDLL** *handle-DLL*

Exemple :

```
; Chargement de la DLL
LOADDLL "MYDLL" , HMOD%
IF HMOD% = 0
MESSAGE "Erreur" , "Chargement de la DLL impossible"
EXIT
ENDIF
...
; Déchargement de la DLL
UNLOADDLL HMOD%
```

Voir aussi LOADDLL, GETPROC, Séparateur DYNAMIC

Instruction UNLOCK

Rend possible toute entrée utilisateur sur un contrôle précédemment bloqué par un LOCK.

Syntaxe	**UNLOCK** contrôle

Exemple :

```
UNLOCK ENTRY00001
```

Voir aussi LOCK, ISLOCKED%, ENABLE, Paramétrage dynamique .LOCKED

Instruction UNSELECT

Désélectionne une ligne d'un objet liste.

Syntaxe	**UNSELECT** *position* **FROM** *objet*

1. La ligne est identifiée par sa position dans la liste. Une position égale à 0 spécifie la première ligne de la liste.
   a) L'objet doit être l'un des suivants :
   b) Window classe List
   c) List-box
   d) Combo-box
   e) CBE-Field
2. Cette instruction n'a aucun effet sur une MLE, comme sur une fenêtre de classe Edit.

Exemple :

```
UNSELECT I% FROM WINDOW(W%).TEST
; Désélectionne l'élément I% de la liste TEST de la fenêtre WINDOW dont le handle est W%
```

Voir aussi ***SELECT*****,** ***SELECTION%*****,** ***ISSELECTED%***

Instruction UNTIL

Utilisé avec l'instruction REPEAT, pour préciser la condition de sortie de la boucle.

Syntaxe	**REPEAT** ... **UNTIL** *condition*

Voir aussi Instruction REPEAT

Opérateur UPCASE

Convertit une chaîne en majuscules.

Syntaxe	**UPCASE** *chaîne*

Exemple :

```
MOVE UPCASE "AbcD1" TO S$ ; S$ vaut "ABCD1"
```

Voir aussi LOWCASE

Instruction UPDATE

Autorise la mise à jour d'un objet liste bloquée par NOUPDATE.

Syntaxe	**UPDATE** objet

1. L'utilisation de cette instruction avec NOUPDATE évite le désagréable "clignotement" lors de modifications successives de la liste par INSERT ou DELETE.
2. L'objet doit être l'un des suivants :
   a) Window classe List ou Edit
   b) List-box
   c) Combo-box
   d) CBE-Field
   e) MLE-Field

Exemple :

```
NOUPDATE LISTE
MOVE 0 TO I%
REPEAT
MOVE I%+1 TO I%
INSERT AT END I% TO LISTE
UNTIL I% = 50
UPDATE LISTE
```

Voir aussi DELETE, INSERT, NOUPDATE, LINECOUNT%, Paramétrage dynamique .UPDATED

Séparateur USING

Utilisé avec CALL*, OPEN*, STRCALL* et STROPEN* pour passer des paramètres à la fenêtre appelée.

Syntaxe	*CALL nom-fenêtre [, handle-fenêtre]* *[**USING** Param1 [, Param2* *[, Param3 [, Param4* *[, variable-retour]]]]]* *OPEN nom-fenêtre, 0 | handle-fenêtre-parent [, handle-fenêtre]* *[**USING** Param1 [, Param2* *[, Param3 [, Param4]]]]*

Param1, Param2, Param3 et Param4 seront respectivement reçus dans PARAM1%, PARAM2%, PARAM3% et PARAM4% de l'événement INIT de la fenêtre appelée.

Voir aussi CALL, OPEN (cf. OPEN*), STRCALL (cf. STRCALL*), STROPEN (cf. STROPEN*), CLOSE

Constantes VK_*%

Touche virtuelle appuyée, passée dans PARAM4% de l'événement CHARACTER. Une touche est dite « virtuelle » lorsqu'elle n'a pas de code caractère correspondant.

Syntaxe	**VK_ALT%** **VK_APPS%** **VK_BACKSPACE%** **VK_BACKTAB%** **VK_BREAK%** **VK_CAPSLOCK%** **VK_CTRL%** **VK_DELETE%** **VK_DOWN%** **VK_END%** **VK_ENTER%** **VK_ESCAPE%** **VK_F1%** **VK_F2%** **...** **VK_F23%** **VK_F24%** **VK_HOME%** **VK_INSERT%** **VK_LEFT%** **VK_NEWLINE%** **VK_NUMLOCK%** **VK_PAGEDOWN%** **VK_PAGEUP%** **VK_PAUSE%** **VK_PRINTSCRN%** **VK_RIGHT%** **VK_SCROLLLOCK%** **VK_SHIFT%** **VK_SPACE%** **VK_SYSRQ%** **VK_TAB%** **VK_UP%**

1. La constante VK_APPS% est utilisée au sein de l'événement CHARACTER pour le support de la touche Applications des claviers comportant les touches

Windows. Cette touche est en général utilisée pour ouvrir un menu contextuel.

2. Dans cet événement, quand le masque KF_VIRTUALKEY% est différent de 0 dans PARAM1%, alors PARAM4% est valide et prend une des valeurs définies par VK_*%. Si l'utilisateur a enfoncé la touche Applications alors PARAM4% prendra la valeur VK_APPS%
3. Leur déclaration interne est :

```
CONST VK_LEFT% 1
CONST VK_RIGHT% 2
CONST VK_UP% 3
CONST VK_DOWN% 4
CONST VK_PAGEUP% 5
CONST VK_PAGEDOWN% 6
CONST VK_HOME% 7
CONST VK_END% 8
CONST VK_INSERT% 9
CONST VK_DELETE% 10
CONST VK_SPACE% 11
CONST VK_TAB% 12
CONST VK_BACKTAB% 13
CONST VK_BACKSPACE% 14
CONST VK_NEWLINE% 15
CONST VK_ENTER% 16
CONST VK_ESCAPE% 17
CONST VK_BREAK% 18
CONST VK_PAUSE% 19
CONST VK_PRINTSCRN% 20
CONST VK_SYSRQ% 21
CONST VK_CAPSLOCK% 22
CONST VK_NUMLOCK% 23
CONST VK_SCROLLLOCK% 24
CONST VK_SHIFT% 25
CONST VK_CTRL% 26
CONST VK_ALT% 27
CONST VK_F1% 28
CONST VK_F2% 29
CONST VK_F3% 30
CONST VK_F4% 31
CONST VK_F5% 32
CONST VK_F6% 33
CONST VK_F7% 34
CONST VK_F8% 35
CONST VK_F9% 36
CONST VK_F10% 37
CONST VK_F11% 38
CONST VK_F12% 39
CONST VK_F13% 40
CONST VK_F14% 41
CONST VK_F15% 42
CONST VK_F16% 43
CONST VK_F17% 44
CONST VK_F18% 45
CONST VK_F19% 46
CONST VK_F20% 47
CONST VK_F21% 48
CONST VK_F22% 49
CONST VK_F23% 50
```

```
CONST VK_F24% 51
CONST VK_APPS 71
```

Exemple :

```
; évenement CHARACTER d'un contrôle ou d'une fenêtre
if ((PARAM1% band KF_VIRTUALKEY%) <> 0) and (PARAM4% = VK_APPS%)
 ; ouverture menu contextuel
endif
```

Voir aussi Evénement CHARACTER, Constantes KF_*%

Instruction WAIT

Stoppe provisoirement l'application pendant un délai exprimé en milli-secondes. Seule l'application courante est stoppée : les autres applications continuent à s'éxécuter.

Syntaxe	**WAIT** *délai*

1. L'utilisateur est averti visuellement, grâce au pointeur souris qui se transforme en sablier (SPTR_WAIT%).
2. Il est déconseillé d'employer un délai trop important : plus aucun événement ne peut circuler au sein de l'application pendant le WAIT. Seule la réception d'un événement USER0 fait sortir immédiatement d'un WAIT, sans attendre la fin du délai.
3. La valeur maximum de délai est de FFFF en hexadécimal soit 4294967295 ms (environ 50 jours).

Exemple :

```
WAIT 1000 ; Stoppe l'exécution pendant 1 seconde
```

***Voir aussi** STARTTIMER, Evénements utilisateur*

Instruction WHERE

Utilisé au sein d'une construction EVALUATE/ENDEVALUATE.

Syntaxe	**WHERE** *expression [TO expression] , expression [TO expression],...* *...* **ENDWHERE**

Voir aussi EVALUATE

Instruction WHILE

L'instruction WHILE répète la séquence d'instructions entre le WHILE et le ENDWHILE tant que la condition est TRUE%.

Syntaxe	**WHILE** *condition ...* **ENDWHILE**

Les instructions entre WHILE et ENDWHILE peuvent n'être jamais exécutées : cela se produit lorsque la condition est fausse dès le début. C'est ce qui diffère d'une construction REPEAT/UNTIL.

Exemple :

```
MOVE 1 TO I%
WHILE COPY$ (TEST, I%, 1) <> "-"
MOVE I%+1 TO I% ; Répété tant que COPY$ <> "-"
ENDWHILE
```

Voir aussi IF, REPEAT, FOR, BREAK, CONTINUE

Fonction WORD%

Retourne la valeur entière (octet-faible modulo 256) + (octet-fort modulo 256) * 256.

Syntaxe	**WORD%** *(octet-fort, octet-faible)*

Exemple 1 :

```
LOCAL I%(2)
MOVE WORD% (HIB I%, LOB I%) TO I%
; I% est inchangé quel que soit sa valeur
```

Exemple 2 :

```
LOCAL J%(4)
MOVE DWORD% (WORD% (HIB HIW J%, LOB HIW J%), \
WORD% (HIB LOW J%, LOB LOW J%)) TO J%
; J% est inchangé quel que soit sa valeur
```

Exemple 3 :

```
MOVE WORD% (2, 1) TO K% ; K% vaut 513
```

Voir aussi ***HIB***, ***LOB***, ***DWORD%***

Constante YES%

Valeur retournée par les fonctions ASK2% et ASK3% lorsque le bouton Yes a été appuyé.

Syntaxe	YES%

Sa déclaration interne est :

```
CONST YES% 0
```

Voir aussi Fonctions ASK2%, ASK3% , Constantes NO%, CANCEL%

Instruction YIELD

Utilisé au sein d'une boucle ou d'un traitement, rend la main au système pour gérer les messages utilisateur ou les messages système.

Syntaxe	YIELD

1. Sans YIELD, le système ne peut traiter aucun message tant que la boucle ou le traitement n'est pas achevé.
2. Un exemple d'utilisation serait l'affichage d'une boîte donnant la possibilité d'arrêter une impression. L'utilisation de YIELD permet de recevoir, s'il y a lieu, la validation du bouton d'annulation et d'effectuer alors le traitement correspondant.

Exemple :

```
I% = 0
WHILE I% < 100000
...
I%= I% + 1
YIELD
ENDWHILE
```

Voir aussi REPEAT, WHILE, FOR

ANNEXE A : LISTE DES EVENEMENTS PAR OBJET

Window sauf Classes List et Edit

ACTIVATE
INIT
TERMINATE
HELP
CHARACTER
MOUSEMOVE
BUTTONDOWN
BUTTONUP
BUTTONDBLCLK
TIMER
PAINT
CHANGED
VSCROLL
HSCROLL
GETFOCUS
LOSEFOCUS
CHECK
DDE *

Window Classe List

ACTIVATE
INIT
TERMINATE
HELP
CHANGED
GETFOCUS
LOSEFOCUS
SELECTED
EXECUTED
CHECK
DDE *

Window Classe Edit

ACTIVATE
INIT
TERMINATE
HELP
CHANGED
GETFOCUS
LOSEFOCUS
SELECTED
CHECK
DDE *

Window Classe Report

Aucun événement

Menu-Item

INIT
ENDMENU
HELP
SELECTED
EXECUTED

Entry-Field

INIT
HELP
CHANGED
GETFOCUS
LOSEFOCUS
CHECK

Push-Button

INIT
HELP
GETFOCUS
LOSEFOCUS
EXECUTED

Static-Text

Aucun événement

Group-Box

Aucun événement

Scroll-Bar Horiz

INIT
HELP
GETFOCUS
LOSEFOCUS
SELECTED
CHECK

Scroll-Bar Vert

INIT
HELP
GETFOCUS
LOSEFOCUS
SELECTED
CHECK

Combo-Box

INIT
HELP
BUTTONDOWN
GETFOCUS
LOSEFOCUS
SELECTED
EXECUTED
CHECK

CBE (Combo-Box Entry-Field)

INIT
HELP
BUTTONDOWN
CHANGED
GETFOCUS
LOSEFOCUS
SELECTED
EXECUTED
CHECK

Sheet Box

INIT
TERMINATE
HELP
SELECTED
EXECUTED
CHECK

MLE (Multi-Line Entry-Field)

INIT
HELP
CHANGED
GETFOCUS
LOSEFOCUS
SELECTED
CHECK

Bitmap

INIT
HELP
GETFOCUS
LOSEFOCUS
SELECTED
EXECUTED
CHECK

Contrôle template

INIT
CHECK

Template .TPL

INIT
CHECK
et événements de comportement

Evénements de Comportement

DELETE
DELETEALL
DISABLE
GETBACKCOL
GETFORECOL
GETHEIGHT
GETTEXT
GETVALUE
GETWIDTH
GETXPOS
GETYPOS
HIDE
INITCONTEXT
INSERT
ISDISABLED
ISHIDDEN
ISLOCKED
ISSELECTED
LINECOUNT
LOAD
LOCK
SAVE
SELECT
SELECTION
SETBACKCOL
SETFOCUS
SETFORECOL
SETPOS
SETRANGE
SETTEXT
SETVALUE
TERMINATECONTEXT
UPDATE

ANNEXE B : FONCTIONS ET INSTRUCTIONS PAR OBJET

Window

Ouverture, Fermeture, Visibilité

CALL, CALLH
CLOSE
HIDE
ISHIDDEN%
OPEN, OPENS, OPENH, OPENSH
SHOW

Tailles et Positions

CHANGE
GETCLIENTHEIGHT%
GETCLIENTWIDTH%
GETHEIGHT%
GETWIDTH%
GETXPOS%
GETYPOS%
ISMAXIMIZED%
ISMINIMIZED%
MAXIMIZE
MINIMIZE
RESTORE
SETPOS

Couleurs

GETBACKCOL%
GETFORECOL%
SETBACKCOL
SETFORECOL

Souris

CAPTURE
UNCAPTURE

Timer

STARTTIMER
STOPTIMER

Titre

GETTEXT$
SETTEXT

Handles

MAINWINDOW%
PARENTWINDOW%
SELF%

Divers

INVALIDATE
SEND
SETFOCUS

Contrôles sauf éléments de menu

Activation

ENABLE
DISABLE
ISDISABLED%

Blocage

ISLOCKED%
LOCK
UNLOCK

Tailles et Positions

GETHEIGHT%
GETWIDTH%
GETXPOS%
GETYPOS%
SETPOS

Visibilité

HIDE
ISHIDDEN%
SHOW

Couleurs

GETBACKCOL%
GETFORECOL%
SETBACKCOL
SETFORECOL

Texte et Contenu

GETTEXT$
SETTEXT
MOVE

Divers

SEND
SETFOCUS

Liste (*)

DELETE
INSERT
ISSELECTED%
LINECOUNT%
LOAD
NOUPDATE
SAVE
SELECT
SELECTION%
UNSELECT
UPDATE

Classeur

LINECOUNT%
SELECT
SELECTION%

Barre de défilement (**)

SETRANGE

(*) Uniquement pour Contrôles List-Box, Combo-Box,CBE, MLE
mais aussi pour aires client de fenêtres de classe Edit ou List.
(**) Uniquement pour Contrôles H. et V. Scroll Bar, EF. Spin-Button mais aussi pour fenêtre avec barres de défilement.

Eléments de menu

Activation

ENABLE
DISABLE
ISDISABLED%

Visibilité

HIDE
ISHIDDEN%
SHOW

Texte et Contenu

GETTEXT$
SETTEXT
MOVE

ANNEXE C : VALEURS DES OBJETS

Bitmap

INIT
HELP
GETFOCUS
LOSEFOCUS
SELECTED
EXECUTED
CHECK

Check-Box

INIT
HELP
GETFOCUS
LOSEFOCUS
SELECTED
EXECUTED
CHECK

Classeur

INIT
TERMINATE
HELP
SELECTED
CHECK

Combo-Box / CBE

Texte affiché dans la ligne toujours visible.
"" si aucun texte.
Le nom suivi d'un index entre crochets [ ] permet de manipuler le texte d'une ligne quelconque.

Contrôle Template

INIT
CHECK

Entry-Field / Push-Button / Static-Text / Group-Box

Texte affiché.
"" si aucun texte.
Un MOVE sur un Entry-Field / Spin-Button est particulier. Voir section "Contrôles de base".

H. Scroll-Bar / V. Scroll-Bar

Position de l'ascenseur, comprise entre le mini et le maxi définis dans la boîte Info, éventuellement modifiés par l'instruction SETRANGE

Icon

Aucun événement

List-Box

INIT
HELP
GETFOCUS
LOSEFOCUS
SELECTED
EXECUTED
CHECK

Menu

CHECKED%
ou UNCHECKED% (**)

MLE

Texte de la ligne affichée à l'endroit du curseur clignotant.
"" si aucun texte.
Le nom suivi d'un index entre crochets [ ] permet de manipuler le texte d'une ligne quelconque.

Radio-Button

INIT
HELP
GETFOCUS
LOSEFOCUS
SELECTED
EXECUTED
CHECK

Template

Dépend du traitement des deux événements GETVALUE SETVALUE.

Comme pour un Template, les valeurs de TOUS CES OBJETS peuvent être gérés différemment en traitant les événements GETVALUE et SETVALUE (menu Add Event).

(*)
Un MOVE sur un Entry-Field Spin-Button est particulier. Voir section "Contrôles de base".
(**)
Le texte associé au contrôle peut être manipulé avec la fonction GETTEXT$ et l'instruction SETTEXT.
(***)
Le nom suivi d'un index entre crochets [ ] permet de manipuler le texte d'une ligne quelconque.
(****)
Le nom suivi d'un index entre crochets ([ ]) retourne le handle de la fenêtre rattachée à la fiche de la fenêtre à onglet associée à cet index.

Window

Aucune valeur associée

ANNEXE D : QUICK REFERENCE NCL

Caractères spéciaux du langage

Type	Description
\	continuation du code sur la ligne suivante
;	commentaire
@	adresse ou affectation de variable de type CONTROL
%	source natif
% $ £ #	type
" ' ^ #	ASCII
[ ]	tableaux
,	séparateur
.	identifieur de contrôle, champ de segment, qualificateur dynamique
..	sous-chaîne
$	hexa
#	base
. E +/-	nombre réel
( )	paramètres fonctions, priorité opérations, taille-variable, handle-fenêtre, ou handle-segment
+ - * / %	opérateurs numériques
& &&	opérateurs de concaténation de chaînes
= <> < >	tests
<= >=	tests

Dans les fonctions ou instructions :

contrôle	[nom-fenêtre(handle-fenêtre).]nom-contrôle ou [nom-fenêtre(handle-fenêtre).]CLIENT ou [nom-fenêtre(handle-fenêtre).]HSCROLL

	ou [nom-fenêtre(handle-fenêtre).]VSCROLL ou [nom-template.]nom-contrôle
objet	contrôle | nom-fenêtre | nom-fenêtre(handle-fenêtre)

A

Type	Description
O	ABS entier | réel
O	entier AND entier
F	ASC% (chaîne)
S	ASCENDING ***(voir INSERT, LOAD)***
F	ASK2% (chaîne-titre, chaîne-message)
F	ASK3% (chaîne-titre, chaîne-message)
S	AT ***(voir DELETE, INSERT)***
P	.AUTOSCROLL
P	.AUTOSIZE
P	.AUTOTAB

Légende :

C	constante prédéfinie
F	fonction
I	instruction
O	opérateur
P	paramétrage dynamique
S	séparateur
T	type
Y	contrôle système

B

Type	Description
P	.BACKCOLOR
O	entier BAND entier
I	BEEP [nb-beep]
O	BNOT entier
O	entier BOR entier
I	BREAK
O	entier BXOR entier

Légende :

C	constante prédéfinie
F	fonction
I	instruction
O	opérateur
P	paramétrage dynamique
S	séparateur
T	type
Y	contrôle système

C

Type	Description
I	CALL nom-fenêtre [, handle-fenêtre] [USING Param1 [, Param2 [, Param3 [, Param4 [, variable-retour]]]]]
I	CALLH nom-fenêtre [, handle-fenêtre] [USING Param1 [, Param2 [, Param3 [, Param4 [, variable-retour]]]]]
C	CANCEL%
I	CAPTURE [handle-fenêtre]
P	.CASE
I	CHANGE [handle-fenêtre] TO pos-x, pos-y [,largeur, hauteur]
T	CHAR
P	.CHARACTERS
C	CHECKED%
F	CHR$ (codeASCII)
Y	CLIENT
I	CLOSE [handle-fenêtre [, variable-retour]]
C	COL *%
P	.COLUMN
I	COMMENT commentaire
I	CONST nom-constante expression-constante
I	CONTINUE
T	CONTROL
F	COPY$ (chaîne-source, position, longueur)
T	CSTRING

Légende :

C	constante prédéfinie

F	fonction
I	instruction
O	opérateur
P	paramétrage dynamique
S	séparateur
T	type
Y	contrôle système

D

Type	Description
C	DEFRET%
I	DELETE [AT position] FROM objet
F	DELETE$ (chaîne-source, position, longueur)
S	DESCENDING ***(voir INSERT, LOAD)***
I	DISABLE contrôle
P	.DISABLED
I	DISPOSE adresse-mem
F	DWORD% (mot-fort, mot-faible)
S	DYNAMIC ***(voir FUNCTION, INSTRUCTION)***

Légende :

C	constante prédéfinie
F	fonction
I	instruction
O	opérateur
P	paramétrage dynamique
S	séparateur
T	type
Y	contrôle système

E

Type	Description
I	ELSE, ELSEIF ***(voir IF, EVALUATE)***
I	ENABLE contrôle
S	END ***(voir INSERT)***
I	ENDEVALUATE ***(voir EVALUATE)***
I	ENDFUNCTION ***(voir FUNCTION)***
I	ENDIF ***(voir IF)***
I	ENDINSTRUCTION ***(voir INSTRUCTION)***
I	ENDLOOP ***(voir LOOP)***
I	ENDSEGMENT ***(voir SEGMENT)***
I	ENDWHERE ***(voir EVALUATE)***
I	ENDWHILE ***(voir WHILE)***
P	.ERASERECT
I	EVALUATE expression WHERE expression [TO expression] ... ENDWHERE [WHERE expression [TO expression] ... ENDWHERE] [...] [ELSE ...] ENDEVALUATE
I	EXIT
S	EXTERNAL ***(voir FUNCTION, INSTRUCTION)***

Légende :

C	constante prédéfinie
F	fonction
I	instruction

O	opérateur
P	paramétrage dynamique
S	séparateur
T	type
Y	contrôle système

F

Type	Description
C	FALSE%
I	FILL adresse-zone, longueur-zone, entier
F	FILLER$ (codeASCII, longueur-chaîne)
P	.FILLTEXT
P	.FONTNAME
P	.FONTSELS
P	.FORECOLOR
P	.FORMAT
I	FOR compteur = expression1 TO expression2 [STEP incr_expression]
S	FROM ***(voir DELETE, SELECT, UNSELECT)***
I	FUNCTION nomfonct [([type] [@]paramètre1 [(taille)] [, [type] [@]paramètre2 [(taille)] [, ...]])] [RETURN type] ... [RETURN valeur] ... ENDFUNCTION
I	FUNCTION nomfonct [([type] [@]paramètre1 [(taille)] [, [type] [@]paramètre2 [(taille)] [, ...]])] [RETURN type] EXTERNAL 'nomDLL.nomfonctionDLL'|DYNAMIC

Légende :

C	constante prédéfinie
F	fonction
I	instruction
O	opérateur
P	paramétrage dynamique

S	séparateur
T	type
Y	contrôle système

G

Type	Description
F	GETBACKCOL% [(objet)]
F	GETCLIENTHEIGHT% [(handle-fenêtre)]
F	GETCLIENTWIDTH% [(handle-fenêtre)]
F	GETDATA% [(handle-fenêtre)]
F	GETFORECOL% [(objet)]
F	GETHEIGHT% [(objet)]
I	GETPROC handle-DLL, nomfonctionDLL, nomfonctNCL
F	GETPTR%
F	GETSPTR% (pointeur-système)
F	GETTEXT$ [(objet)]
F	GETWIDTH% [(objet)]
F	GETXPOS% [(objet)]
F	GETYPOS% [(objet)]
I	GLOBAL <type> nom-variable <(taille)> <[dim] <[dim] ...> >

Légende :

C	constante prédéfinie
F	fonction
I	instruction
O	opérateur
P	paramétrage dynamique
S	séparateur
T	type
Y	contrôle système

H

Type	Description
P	.HALFTONE
I	HALT [code-retour]
P	.HEIGHT
O	HIB entier
P	.HIDDEN
I	HIDE [objet]
P	.HIDETEXT
O	HIW entier
P	.HORZSCROLLBAR
Y	HSCROLL

Légende :

C	constante prédéfinie
F	fonction
I	instruction
O	opérateur
P	paramétrage dynamique
S	séparateur
T	type
Y	contrôle système

I

Type	Description
I	IF condition .... [ELSEIF condition ....] [ELSE ....] ENDIF
I	INCLUDE nom-fichier
C	INDETERMINATE%
I	INSERT ASCENDING | DESCENDING chaîne TO objet
I	INSERT AT position | END chaîne TO objet
F	INSERT$ (chaîne-source, chaîne-destination, position-insertion)
I	INSTRUCTION nominstr [[type] [@]paramètre1 [(taille)] [, [type] [@]paramètre2 [(taille)] [, ...]]] ... [RETURN | EXIT] ... ENDINSTRUCTION
I	INSTRUCTION nominstr [[type] [@]paramètre1 [(taille)] [, [type] [@]paramètre2 [(taille)] [, ...]]] EXTERNAL 'nomDLL.nomfonctionDLL' | DYNAMIC
O	INT entier | réel | chaîne
T	INT
T	INTEGER
P	.INTERNALVALUE
I	INVALIDATE [handle-fenêtre]
F	ISDISABLED% (contrôle)
F	ISHIDDEN% [(objet)]
F	ISINT% (chaîne)
F	ISLOCKED% [(contrôle)]

F	ISMAXIMIZED% [(handle-fenêtre)]
F	ISMINIMIZED% [(handle-fenêtre)]
F	ISMOUSE%
F	ISNUM% (chaîne)
F	ISSELECTED% (objet, position)

Légende :

C	constante prédéfinie
F	fonction
I	instruction
O	opérateur
P	paramétrage dynamique
S	séparateur
T	type
Y	contrôle système

J-K

Type	Description
P	.JUSTIFICATION
C	JUST *%
C	KF *%
P	.KIND

Légende :

C	constante prédéfinie
F	fonction
I	instruction
O	opérateur
P	paramétrage dynamique
S	séparateur
T	type
Y	contrôle système

L

Type	Description
O	LENGTH chaîne
P	.LINE
F	LINECOUNT% [(objet)]
P	.LISTWIDTH
I	LOAD [ASCENDING | DESCENDING] nom-fichier TO objet
I	LOADDLL nomDLL, handle-DLL
O	LOB entier
I	LOCAL <type> nom-variable <(taille)> <[dim] <[dim] ...> >
I	LOCK contrôle
P	.LOCKED
I	LOOP ... ENDLOOP
O	LOW entier
O	LOWCASE chaîne
O	LSKIP chaîne

Légende :

C	constante prédéfinie
F	fonction
I	instruction
O	opérateur
P	paramétrage dynamique
S	séparateur
T	type
Y	contrôle système

M

Type	Description
F	MAINWINDOW%
P	.MARGIN
I	MAXIMIZE [handle-fenêtre]
P	.MAXLEN
I	MESSAGE chaîne-titre, chaîne-message
I	MINIMIZE [handle-fenêtre]
P	.MNEMONIC
C	MOU *%
I	MOV adresse-source, adresse-destination, longueur
I	MOVE expression TO variable | contrôle
P	.MOUSEPOINTER
P	.MULTIPLESEL

Légende :

C	constante prédéfinie
F	fonction
I	instruction
O	opérateur
P	paramétrage dynamique
S	séparateur
T	type
Y	contrôle système

N

Type	Description
F	NBBUTTONS%
P	.NBLINES
I	NEW nom-segment | taille, adresse-mem [, nom-public]
C	NO%
P	.NOADJUSTPOS
P	.NOBLANKS
P	.NOCHECKING
P	.NOPOINTERFOCUS
C	NOSELECTION%
O	NOT entier
I	NOUPDATE objet
O	NUM entier | réel | chaîne
T	NUM

Légende :

C	constante prédéfinie
F	fonction
I	instruction
O	opérateur
P	paramétrage dynamique
S	séparateur
T	type
Y	contrôle système

O

Type	Description
I	OPEN nom-fenêtre, 0 | handle-fenêtre-parent, [, handle-fenêtre] [USING Param1 [, Param2 [, Param3 [, Param4]]]]
I	OPENH nom-fenêtre, 0 | handle-fenêtre-parent, [, handle-fenêtre] [USING Param1 [, Param2 [, Param3 [, Param4]]]]
I	OPENS nom-fenêtre, 0 | handle-fenêtre-parent, [, handle-fenêtre] [USING Param1 [, Param2 [, Param3 [, Param4]]]]
I	OPENSH nom-fenêtre, 0 | handle-fenêtre-parent, [, handle-fenêtre] [USING Param1 [, Param2 [, Param3 [, Param4]]]]
O	entier OR entier

Légende :

C	constante prédéfinie
F	fonction
I	instruction
O	opérateur
P	paramétrage dynamique
S	séparateur
T	type
Y	contrôle système

P-Q

Type	Description
F	PARAM1%
F	PARAM2%
F	PARAM3%
F	PARAM4%
F	PARAM12%
F	PARAM34%
F	PARAM1$
F	PARAM2$
F	PARENTWINDOW% [(handle-fenêtre)]
I	PASS [param1[, param2[, param3[, param4[, variable-retour]]]]]
T	POINTER
F	POS% (chaîne-cherchée, chaîne-source)
I	POST événement [, param1 [, param2 [, param3 [, param4]]]] [TO objet]

Légende :

C	constante prédéfinie
F	fonction
I	instruction
O	opérateur
P	paramétrage dynamique
S	séparateur
T	type
Y	contrôle système

R

Type	Description
P	.REALTIMESCROLL
C	REL *%
P	.RELIEF
I	REPEAT .... UNTIL condition
P	.REQUIRED
I	RESTORE [handle-fenêtre]
I	RETURN [expression]
S	RETURN ***(voir FUNCTION, INSTRUCTION)***
F	ROUND% (réel)
O	RSKIP chaîne

Légende :

C	constante prédéfinie
F	fonction
I	instruction
O	opérateur
P	paramétrage dynamique
S	séparateur
T	type
Y	contrôle système

S

Type	Description
I	SAVE objet TO nom-fichier
C	SB *%
F	SCREENHEIGHT%
F	SCREENWIDTH%
I	SEGMENT nom-segment <type> nom-champ1 <(taille)> <[dim] <[dim] ...> > <type> nom-champ2 <(taille)> <[dim] <[dim] ...> > ... ENDSEGMENT
I	SELECT position FROM objet
F	SELECTION% [(objet)]
Y	SELF
F	SELF%
I	SEND événement [, param1 [, param2 [, param3 [, param4]]]] [TO objet] [, variable-retour]
P	.SEPARATORS
I	SETBACKCOL couleur [TO objet]
I	SETDATA entier [TO handle-fenêtre]
I	SETFOCUS [objet]
I	SETFORECOL couleur [TO objet]
I	SETPOS posx, posy [TO objet]
I	SETPTR handle-pointeur
I	SETRANGE position, mini, maxi TO contrôle
I	SETTEXT chaîne [TO objet]
I	SHOW [objet]
O	SIZEOF variable | nomsegment
O	SKIP chaîne
P	.SKIPBLANKS

C	SPTR *%
I	STARTTIMER numéro, tempo [TO handle-fenêtre]
I	STOPTIMER numéro [TO handle-fenêtre]
I	STRCALL chaîne-nom-fenêtre [handle-fenêtre] [USING Param1 [, Param2 [, Param3 [, Param4 [, variable-retour]]]]]
I	STRCALLH chaîne-nom-fenêtre [, handle-fenêtre] [USING Param1 [, Param2 [, Param3 [, Param4 [, variable-retour]]]]]
O	STRING entier | réel | chaîne
T	STRING
I	STROPEN chaîne-nom-fenêtre, 0 | handle-fenêtre-parent, [, handle-fenêtre] [USING Param1 [, Param2 [, Param3 [, Param4]]]]
I	STROPENH chaîne-nom-fenêtre, 0 | handle-fenêtre-parent, [, handle-fenêtre] [USING Param1 [, Param2 [, Param3 [, Param4]]]]
I	STROPENS chaîne-nom-fenêtre, 0 | handle-fenêtre-parent, [, handle-fenêtre] [USING Param1 [, Param2 [, Param3 [, Param4]]]]
I	STROPENSH chaîne-nom-fenêtre, 0 | handle-fenêtre-parent, [, handle-fenêtre] [USING Param1 [, Param2 [, Param3 [, Param4]]]]

Légende :

C	constante prédéfinie
F	fonction
I	instruction

O	opérateur
P	paramétrage dynamique
S	séparateur
T	type
Y	contrôle système

T

Type	Description
P	.TABULATIONS
P	.TEXT
P	.TEXTSEL
S	TO (voir CHANGE, INSERT, LOAD, MOVE, SAVE SEND, SETBACKCOL, SETDATA, SETFORECOL, SETPOS, SETRANGE, SETTEXT, STARTTIMER, STOPTIMER, WHERE)
P	.TOPLINE
C	TRUE%
F	TRUNC% (réel)
C	TYP *%
P	.TYPE

Légende :

C	constante prédéfinie
F	fonction
I	instruction
O	opérateur
P	paramétrage dynamique
S	séparateur
T	type
Y	contrôle système

U

Type	Description
I	UNCAPTURE
C	UNCHECKED%
I	UNLOADDLL handle-DLL
I	UNLOCK contrôle
I	UNSELECT position FROM objet
I	UNTIL ***(voir REPEAT)***
O	UPCASE chaîne
I	UPDATE objet
P	.UPDATED
S	USING ***(voir CALL, OPEN, STRCALL, STROPEN, CLOSE)***

Légende :

C	constante prédéfinie
F	fonction
I	instruction
O	opérateur
P	paramétrage dynamique
S	séparateur
T	type
Y	contrôle système

V-W

Type	Description
P	.VALUE
C	VK_*%
Y	VSCROLL
I	WAIT délai
I	WHERE expression TO expression ***(voir EVALUATE)*** ... ENDWHERE
I	WHILE condition .... ENDWHILE
P	.WIDTH
F	WORD% (octet-fort, octet-faible)
P	.WORDWRAP

Légende :

C	constante prédéfinie
F	fonction
I	instruction
O	opérateur
P	paramétrage dynamique
S	séparateur
T	type
Y	contrôle système

X-Y-Z

Type	Description
P	.X
P	.Y
C	YES%
I	YIELD

Légende :

C	constante prédéfinie
F	fonction
I	instruction
O	opérateur
P	paramétrage dynamique
S	séparateur
T	type
Y	contrôle système

ANNEXE E : PRESENTATION DES LISTES

Les fonctionnalités standards des List Box peuvent être complétées à l'aide de descripteurs de présentation.

Ces descripteurs permettent de :

- Colorier, justifier et encadrer le contenu des listes,
- Modifier la hauteur des lignes,
- Bloquer une ou plusieurs lignes en haut de la liste
- Insérer des icônes ou bitmaps
- Modifier dynamiquement la largeur des colonnes lors de l'utilisation des listes

Un descripteur est spécifié dans un ordre INSERT ou dans un ordre MOVE.

Il est possible d'appliquer un descripteur de présentation à l'intersection d'une ligne et d'une colonne (cellule) de la List Box. Il est placé dans ce cas avant chaque texte à modifier lors de l'insertion de la ligne dans la liste. Un tel descripteur est appelé descripteur local :

```
INSERT AT xxx « <Descripteur><Texte><Séparateur><Descripteur><Texte><Séparateur> …… » TO LISTBOX
```

Un descripteur peut aussi modifier globalement les caractéristiques de toutes les cellules d'une ou plusieurs colonnes de la List Box. Dans ce cas, il est appelé descripteur global.

Cela signifie qu'il est possible de définir des caractéristiques qui vont être appliquées à toute nouvelle ligne de texte insérée par la suite. Cette modification globale est spécifiée par l'insertion d'une ligne commençant par le caractère « ~ » et composée uniquement de descripteurs :

```
INSERT AT xxx « ~<Descripteur><Séparateur><Descripteur><Séparateur> …… » TO LISTBOX
```

Les lignes de texte sont ensuite insérées « classiquement » :

```
INSERT AT xxx « <Texte><Séparateur><Texte><Séparateur> …. » TO LISTBOX
```

Le contrôle List Box ne peut pas gérer plus de 32766 lignes, ni un index de lignes supérieur à 32766.

Syntaxe d'un descripteur

La syntaxe d'un descripteur de présentation est la suivante :

```
{Type[Qualificateur], Type[Qualificateur],….}
```

Un descripteur est encadré par des accolades.
Il s'applique à la cellule correspondant à son emplacement dans l'ordre INSERT.

Ses composants sont séparés par une virgule.

Un composant est formé d'un identificateur de type, suivi des qualificateurs appropriés encadrés par des crochets.

Le type, représenté par une lettre majuscule, indique quel élément de présentation doit être affecté (couleur, justification, ...). Les qualificateurs de type précisent la ou les options choisies. Selon le type auquel il appartient, un qualificateur peut être une lettre, un mot prédéfini, une valeur numérique. Certains qualificateurs peuvent se combiner.

Il ne doit pas y avoir de caractère blanc entre le séparateur et l'accolade du descripteur, ni entre la virgule et un indicateur de type.

Exemple de descripteur :
{B[BLUE], F[YELLOW], G[LRBT]}
Dans lequel les types sont : B, F et G
Et les qualificateurs : BLUE, YELLOW, L, R, B et T.

Certains qualificateurs ne s'appliquent que globalement.

Le tableau suivant donne pour chaque identificateur de type : sa signification, son domaine d'application (local, global), ses qualificateurs et leurs applications.

Type	Domaine	Qualificateur	Signification/Application

F	local & global	BLACK BLUE BROWN CYAN DARKGRAY GREEN MAGENTA RED WHITE YELLOW LIGHTBLUE LIGHTCYAN LIGHTGRAY LIGHTGREEN LIGHTMAGENTA LIGHTRED	Foreground color;: couleur des caractères Noir Bleu Marron Cyan Gris Vert Magenta Rouge Blanc Jaune Bleu clair Cyan clair Gris clair Vert clair Magenta clair Rouge clair Les qualificateurs doivent être spécifiés en lettres majuscules.

B	local & global	BLACK BLUE BROWN CYAN DARKGRAY GREEN MAGENTA RED WHITE YELLOW LIGHTBLUE LIGHTCYAN LIGHTGRAY LIGHTGREEN LIGHTMAGENTA LIGHTRED	Background color : couleur d’arrière-plan Noir Bleu Marron Cyan Gris Vert Magenta Rouge Blanc Jaune Bleu clair Cyan clair Gris clair Vert clair Magenta clair Rouge clair Les qualificateurs doivent être spécifiés en lettres majuscules.

G	local & global	L R B T I K N l r b t A	Grid : tracé d'un cadre ou d'une partie de cadre autour d'une cellule Left : Tracé d'une verticale su la gauche. Right : Tracé d'une verticale sur la droite. Bottom : Tracé d'une horizontale en dessous. Top : Tracé d'une horizontale au dessus. Light : dessin en relief Dark : dessin en creux. None : annule i et k. Left : idem L mais avec un pixel de décalage Right : idem R mais avec un pixel de décalage Bottom : idem B mais avec un pixel de décalage Top : idem T mais avec un pixel de décalage Alternate : Tracé en pointillé. L'emploi des lettres minuscules évite le chevauchement des encadrements entre colonnes. L'emploi du type G sans qualificateur (G[]) annule l'encadrement d'une cellule.
J	local & global	C L R	Justification : positionnement du texte sur la ligne par rapport aux bords de la colonne. Center : Centrage du texte Left : Texte justifié à gauche Right : Texte justifié à droite Les qualificateurs doivent être spécifiés en lettres majuscules.

M	local & global	Y N	BitMap : insertion d'une icône ou d'une bitmap. Yes : Activation de la bitmap No : Désactivation locale de la bitmap Les qualificateurs doivent être spécifiés en lettres majuscules.
R	global		Resizable : colonne retaillable dynamiquement avec la souris. Aucun qualificateur n'est associé à ce type (ne pas mettre les []).
T	global	n	Top Lock : blocage d'une ou plusieurs lignes en haut de liste. Valeur numérique indiquant le nombre de lignes à bloquer en haut de la liste. Une fois bloquées, les lignes spécifiées ne défilent plus verticalement.
H	global	n	Height : hauteur (en pixels) des lignes de la liste. Valeur numérique indiquant la hauteur désirée. Ce type s'applique toujours à l'ensemble des lignes de la liste.

S	global	B F A Y N	Show Selection BackGround : Seule la couleur d'arrière-plan de la sélection est appliquée. La couleur des caractères reste celle d'une ligne non sélectionnée. Foreground : Seule la couleur des caractères de la sélection est appliquée. La couleur d'arrière-plan de la sélection reste celle d'une ligne non sélectionnée. All : La couleur de fond et du texte prennent les couleurs de sélection (mode normal). Yes : La couleur de fond et du texte prennent les couleurs de sélection (mode normal). No : La ligne sélectionnée de la colonne n'est pas mise en évidence. Les qualificateurs doivent être spécifiés en lettres majuscules.
C	local & global		Recopie les attributs de la cellule (local), ou de la colonne (global) précédente.
P	Local & global	Y N	Insertion d'un pointeur sur du texte à la place du texte lui-même. Yes : Activation du pointeur No : Désactivation du pointeur Les qualificateurs doivent être spécifiés en lettres majuscules.

Règles générales

Caractéristiques de la liste

Les descripteurs de présentation concernent :

- Les List Box composées de plusieurs colonnes pour lesquelles les champs Tabulations et Separators ont été définis dans la boîte info, ou pour lesquelles les paramètres dynamiques Tabulations et Separators ont été définis
- Les Combo Box composées de plusieurs colonnes pour lesquelles les paramètres dynamiques Tabulations et Separators ont été définis.

Les valeurs spécifiées dans le champ Tabulations désignent la largeur, en pixel, de chaque colonne. Ainsi, 0,60,100,100 définit 4 colonnes de largeurs respectives 0,60,100 et 100.

Le premier chiffre est la largeur de la colonne n°0. Cette colonne est particulière : elle ne peut pas recevoir de texte.

Cette colonne sert principalement à indenter les données affichées dans la liste d'un nombre de pixels équivalent à sa largeur. La première colonne qui contient du texte est donc la colonne numérotée 1.

Il n'est donc pas possible d'appliquer de descripteur local à la colonne n°0, un descripteur local étant toujours accolé au texte qu'il met en forme.

De ce fait, la colonne n°0 ne peut être modifiée que globalement. Les types valides pour cette colonne sont : B, F, G, R (F ne s'appliquant qu'à la couleur des cadres).

Application globale de descripteurs

```
INSERT AT xxx « ~<Descripteur><Séparateur><Descripteur><Séparateur> …… » TO LISTBOX
```

Le premier descripteur s'applique à la colonne n°0.

Application locale de descripteurs

```
INSERT AT xxx « ~<Descripteur><texte><Séparateur><Descripteur><texte><Séparateur> …… » TO
LISTBOX
```

Le premier descripteur s'applique à la première colonne qui contient du texte, c'est-à-dire à la colonne n°1.

Ecrasement d'un format

Pour une ligne et une colonne données, s'il existe un descripteur de format local, les options de ce dernier remplacent ou complètent celles qui ont pu être définies globalement.

De même, les options d'un format global remplacent ou complètent celles qui ont pu être définies dans un format global précédent.

Le format est considéré comme un texte si la syntaxe utilisée est incorrecte.

Utilisation des types

Background (B) et Foreground (F)

A la place d'un qualificateur de couleur, il est possible de spécifier une constante COL_*% ou la valeur qui lui est associée.
La syntaxe à respecter est la suivante :
[# « & COL_*% & »] : emploi d'une constante COL_*%
[valeur] : emploi de la valeur associée
Pour spécifier une constante, il est nécessaire de la transformer en chaîne de caractères par une concaténation.
Des couleurs définies au niveau du Control Panel, telles que COL_BACKGROUND% peuvent ainsi être utilisées.
A partir de NatStar 2.61, les descripteurs F (Foreground color : couleur des caractères) et B (Background color : couleur d'arrière-plan) permettent d'indiquer une couleur RGB. La nouvelle syntaxe esr {B[%n]} et {F[%n]}, où n est un entier (décimal ou hexadécimal).

```
{B[%$C0D0FF]}
```

donne une couleur d'arrière-plan bleu ciel (rouge=C0, vert=D0 et bleu = FF).

```
;Exemple
INSERT AT 0 "~|{M[Y],H[30],T[1]}|{B[%$C0D0FF]}" to LB2
```

Les qualificateurs

Les qualificateurs des descripteurs B et F peuvent désormais être abrégés. Seules les premières lettres permettant de les distinguer les unes des autres deviennent indispensables.

> Les préfixes DARK et LIGHT peuvent être abrégés en L et D. Les orthographes GREY (anglais) et GRAY (anglo-américain) sont permises.

BLA = BLACK
BLU = BLUE
G = GREEN
C = CYAN
R = RED
M = MAGENTA
BR = BROWN
LGREY = LIGHTGREY
LGRA = LIGHTGRAY
DG = DARKGREY ou DARKGRAY
LB = LIGHTBLUE
LGREE = LIGHTGREEN
LC = LIGHTCYAN
LR = LIGHTRED
LM = LIGHTMAGENTA
Y = YELLOW
W = WHITE

Encadrement (G)

Un encadrement complet d'une cellule s'obtient en spécifiant les quatre qualificateurs LRTB ou lrtb. L'ordre de leur spécification n'a pas d'importance.

Les mêmes qualificateurs d'encadrement pour deux cellules contiguës provoquent l'affichage d'un trait double de séparation. Un moyen d'y remédier est de ne spécifier qu'un demi-encadrement (bord droit et bord bas).

L'encadrement est, par défaut, dessiné avec une ligne continue. L'utilisation du qualificateur A modifie le tracé en ligne pointillée.

Justification (J)

La chaîne contenue dans une cellule déborde sur la cellule suivante lorsque la chaîne est plus large que la cellule ET que toutes les cellules suivantes sont vides. En d'autres termes, l'affichage d'une chaîne n'est pas circonscrit à une cellule si celle-ci est la dernière de la ligne dans laquelle figure un texte.

Cela signifie aussi que la largeur effective de la dernière colonne d'une liste est la largeur de la plus longue chaîne des cellules de la colonne.

Insertion d'icônes ou de bitmaps (M)

Il est possible d'insérer des bitmaps dans une List Box, sauf dans la colonne n°0.
Une bitmap est désignée par son handle :

- Les handles des pointeurs ou des icônes système sont exprimés en ASCII. Ils sont de la forme ^## où ## est la valeur d'une constante SPTR *%.
- Les handles des fichiers bitmaps sont obtenus avec la fonction CREATEBMP% du langage NCL. Cette fonction associe un handle de type INT à un fichier bitmap.

```
;Exemple
MOVE CREATEBMP% (« c:\bmp\mybmp.bmp ») to hbmp%
```

Une cellule ne peut pas recevoir à la fois du texte et une bitmap. Une cellule devant contenir une bitmap doit donc être déclarée comme réceptrice de bitmap (par défaut, une cellule reçoit du texte).
Cette déclaration peut être faite :

- Soit au moment de l'insertion de la bitmap, en accolant à son handle un descripteur local avec le type M et le qualificateur Y.
- Soit au moment de l'application d'un format global, en indiquant le type M avec le qualificateur Y pour chaque cellule devant contenir une bitmap.

Un simple ordre INSERT avec comme texte uniquement le handle de la bitmap suffira pour l'insérer ensuite.
Remarques :

1. Il est quand même possible de mettre du texte dans une cellule déclarée comme réceptrice de bitmap. Il faut pour cela désactiver localement la bitmap, c'est-à-dire faire précéder le texte d'un descripteur local avec le type M et le qualificateur N lors de son insertion.
2. Si la hauteur de la ligne est plus petite que la bitmap, celle-ci est tronquée. Dans tous les cas, la bitmap est affichée centrée verticalement.
3. Horizontalement, une bitmap peut être repositionnée à l'aide du descripteur de justification J.
4. Lorsqu'un handle est erroné, un rectangle grisé s'affiche dans la cellule.

Colonne retaillable (R)

Lors de l'exécution de l'application, il suffit pour retailler une colonne de positionner le pointeur souris sur la verticale située entre on bord droit et le bord gauche de la colonne suivante.

Le pointeur souris se transforme alors en « <-||-> ».

En appuyant sur le bouton gauche de la souris puis en déplaçant celle-ci tout en laissant le bouton enfoncé, la colonne s'élargit ou se rétrécit à volonté.

> Il est conseillé de donner des couleurs différentes aux colonnes ou de définir celles-ci avec le type G[R] pour que la séparation entre deux colonnes puisse être visible et que l'utilisateur sache où positionner le pointeur souris pour élargir une colonne.

Il n'est pas possible, dans un nouveau format de présentation global, de rendre non retaillable une colonne qui a été spécifiée comme retaillable.

Blocage de lignes du haut de la liste (T)

Une fois bloquées, les n lignes du haut de la liste ne défilent plus verticalement et restent donc toujours affichées. Elles ne sont donc pas sélectionnables.
Le blocage de lignes sert principalement à placer des titres de colonnes ou un titre général en haut de liste. Les titres restent ainsi affichés en permanence, même lorsqu'un défilement vertical est provoqué. Par contre, ils défilent horizontalement avec les colonnes.

> Attention, l'indexation des lignes de la liste n'est pas modifiée par un blocage, c qui signifie que les lignes bloquées continuent à être comptabilisées.

Copie des attributs de la cellule ou de la colonne précédente (C)

Le nouveau descripteur de présentation C recopie les attributs de la cellule ou de la colonne précédente. Le descripteur C permet donc d'éviter de répéter les mêmes attributs sur plusieurs colonnes consécutives.

De plus, il est possible d'ajouter des modifications aux attributs copiés. Par exemple, "{C,B[W]}" indique que l'on veut les même attributs que la cellule ou colonne précédente sauf la couleur de fond qui sera blanche (W = WHITE).

Les modifications sont ignorées si elles précèdent le descripteur de présentation C.

Insertion d'un pointeur sur du texte (P)

Le nouveau descripteur de présentation P permet d'insérer un pointeur sur du texte à la place du texte lui-même. Comme pour les bitmap ou icon, le pointeur doit être exprimé comme un entier (décimal ou hexadécimal).

Le pointeur doit toujours être ou nul (0), ou valide (pointer sur une CSTRING(n) où n <= 255). La List box ne gère pas la suppression du texte pointé lorsque l'élément est retiré de la List box ou que celle-ci disparaît, car il n'est pas obligatoirement alloué par un NEW (ça peut être une CONST ou une variable GLOBAL, ou le texte peut faire partie d'un SEGMENT).

Le pointeur doit impérativement rester valable pendant son existence dans la List box. Et, il ne faut pas utiliser l'adresse d'une variable locale ou d'un paramètre de fonction (sauf si c'est une fonction qui fait un CALL de la boite de dialogue contenant la List box). "{P[N]}" permet de désactiver localement l'effet d'un "{P[Y]}" global. L'intérêt du descripteur P est de pouvoir insérer une longueur totale de chaîne supérieure à 255 caractères comme ligne dans une List Box.

```
;Exemple
;Déclaration dans l'INIT de la fenêtre
GLOBAL MACHAINE$
machaine$='By Nat System e-mail pdupont@natsys.fr'
;notre List Box dans ce cas a comme séparateur de colonne le '|'
;Puis sous un événement quelconque :
Local int i(2)
local pointer h%
NOUPDATE LB3
; assigns the handle of a bitmap control (B1) to h%
move B1.text to h%
; the global format
INSERT AT 0 "~|{M[Y],H[30],T[1]}|{}|{P[Y]}" to LB3
for i = 1 to 10
 INSERT AT END h% & "|Test"&"|"&@machaine$ to LB3
endfor
UPDATE LB3
```

Extensions pour l'affichage d'une bitmap ayant une couleur transparente

Une bitmap a une couleur transparente lorsque la couleur de la bitmap est remplacée par la couleur de fond de la cellule de la List Box.

Syntaxe	handle-bitmap	[^[CouleurTransparente][^IndiceTranche^LargeurTranc
Paramètres	CouleurTransparente	Valeur de l'une des constantes TRANSP_*% ou COL_*%
	IndiceTranche	Index de la tranche de la bitmap, 0 pour la plus à gauch
	LargeurTranche	Largeur (en pixels) des tranches de la bitmap.

1. Les paramètres entre crochets [] sont optionnels.
2. Le paramètre CouleurTransparente permet de définir la couleur transparente. Tous les pixels de cette couleur sont remplacés par la couleur de fond de la colonne ou de la sélection. Si ce paramètre n'est pas spécifié, l'affichage n'est pas transparent.
3. Les extensions suivantes, permettent d'afficher les bitmaps ayant une couleur transparente et/ou une tranche horizontale d'une bitmap (si une bitmap contient plusieurs petites images de même largeur mises côte à côte).

Valeur du paramètre CouleurTransparente	Signification
TRANSP_DEFAULT% (ou -5) ou TRANSP_NEVER% (ou -6)	Couleur opaque, valeur par défaut si elle n'est pas spécifiée.
Valeur supérieure ou égale à 0	Indique une couleur COL_*%
TRANSP_TOPLEFT% (ou -1)	Indique que la couleur du pixel dans le coin (ou tranche) supérieur gauche de la bitmap sert de couleur transparente.
TRANSP_BOTLEFT% (ou -2)	Indique que la couleur du pixel dans le coin (ou tranche) inférieur gauche de la bitmap sert de couleur transparente.
TRANSP_TOPRIGHT% (ou -3)	Indique que la couleur du pixel dans le coin (ou tranche) supérieur droit de la bitmap sert de couleur transparente.
TRANSP_BOTRIGHT% (ou -4)	Indique que la couleur du pixel dans le coin (ou tranche) inférieur droit de la bitmap sert de couleur transparente.

4. Les paramètres IndiceTranche et LargeurTranche permettent de n'afficher qu'une partie de la bitmap en spécifiant un découpage en tranches horizontales de l'ensemble de la bitmap de LargeurTranche pixels de larges, IndiceTranche indiquant alors quelle tranche doit être dessinée (indice de 0 à (LargeurTotaleBitmap/LargeurTranche) - 1).
5. L'utilisation des tranches de bitmap présente quelques avantages :
6. Elles permettent de réduire le nombre de contrôles bitmap cachés contenant les bitmaps affichées dans une (ou plusieurs) List box et du même coup, de réduire la consommation de ressources GDI.
7. Avec la possibilité de charger une image GIF ou JPEG sous Windows elles permettent aussi de réduire la taille de l'exécutable ou de la DLL (s'il y a beaucoup de bitmaps). En effet, une grande bitmap est généralement mieux compressée qu'une série de petites.

```
;Exemple
12345678^TRANSP_TOPLEFT%
;indique une bitmap dont le handle est 12345678 et dont on veut ;utiliser la couleur du
pixel supérieur ;gauche comme couleur transparente.
hbmp%^^3^16
;indique une bitmap opaque dont le handle est hbmp% et dont on veut afficher la 4ème
tranche (à partir de ;la gauche) de 16 ;pixels de large.
```

Exemples

Supposons une List Box ayant pour nom LB_TEST, et ayant dans sa boîte Info les informations suivantes :
Dans le champ Tabulation : 0,70,70,70,70
Dans le champ Separators : '-'

> L'opérateur & est utilisé pour concaténer deux chaînes de caractères. Pour passer à la ligne au milieu d'une chaîne de caractères dans un code NCL, une autre solution consiste à ne pas indenter la deuxième partie de la chaîne de caractères.

Insertion avec descripteurs locaux de présentation

L'instruction suivante a pour effet de centrer aaa (en gris foncé) dans la colonne n°1, centrer bbb (en rouge) dans la colonne n°2 et aligner ccc (en vert) sur la droite dans la colonne n°3 :

```
INSERT AT END « {F[DARKGRAY],J[C]} aaa-{F[RED],J[C]}bbb » & « -{F[GREEN],J[R]ccc » TO
LB_TEST
```

Sans utiliser l'opérateur & cette instruction s'écrit :

```
INSERT AT END « {F[DARKGRAY],J[C]} aaa-{F[RED],J[C]}bbb » « -{F[GREEN],J[R]ccc » TO
LB_TEST
```

L'instruction suivante a pour effet d'insérer une colonne de caractères bleus sur fond jaune complètement encadrée :

```
INSERT AT END « {F[BLUE],B[YELLOW],G[LRTB]}cadre » TO LB_TEST
```

L'instruction suivante a pour effet d'insérer le texte « hello » en caractères bleus, fond blanc, encadré dans la colonne n°1, puis le texte « world » en caractères rouges, fond vert clair, souligné :

```
INSERT AT END «{F[BLUE],B[WHITE],G[TBLR]hello» & «-{F[RED],B[GREEN],G[B]}world» TO LB_TEST
```

Utilisation des constantes COL_*%

Les deux instructions suivantes ont le même effet, écrire aaa en caractères jaunes dans la colonne n°1 :

```
INSERT AT END « {F[#»&COL_YELLOW%& »]}aaa » TO LB_TEST
INSERT AT END « {F[#14]}aaa » TO LB_TEST
```

***Voir aussi** Constantes COL_*%*

Exemple de formatage global

Effaçons ce qui a été inséré précédemment dans la liste puis définissons un format global pour celle-ci : la colonne n°1 avec un fond vert et des caractères jaunes centrés et encadrés, la colonne n°2 avec un fond blanc et des caractères noirs encadrés, la colonne n°3 avec un fond identique à celui de la liste et des caractères rouges encadrés. Une astuce est utilisée pour encadrer les colonnes sans qu'il y ait chevauchement, en ne se servant que du bord droit et du bord bas (demi-encadrement).
Ce format se définit ainsi :

```
INSERT AT 0 « ~ - {F[YELLOW],B[GREEN],J[C], G[RB]} » & \
« - {F[BLACK],B[WHITE],G[RB]} – {F[RED],G[RB]} »\
TO LB_TEST
```

Par la suite, l'écriture de caractères dans les différentes colonnes de la liste respecte ce format :

```
INSERT AT END « aaa-bbb-ccc » TO LB_TEST
INSERT AT END « ddd-eee-fff » TO LB_TEST
```

Ainsi, la colonne contenant aaa et ddd est verte, les caractères sont jaunes et centrés dans la colonne.

Sans la présence du séparateur '-' juste derrière le caractère '~', le format {F[YELLOW],B[GREEN],J[C]} s'appliquerait à la colonne n°0, de largeur nulle dans notre exemple.

Application d'un nouveau format global

Définissons maintenant le nouveau format global suivant :

```
INSERT AT 0 « ~-----{B[YELLOW]} » TO LB_TEST
```

Ce format ne modifie pas la présentation des colonnes n°0,1,2,3,4. Ces colonnes conservent le format global précédemment défini. Par contre, la zone de la List Box qui se situe à droite de la colonne n°4 (dernière colonne) sera coloriée en jaune.

Descripteurs locaux de présentation dans une commande MOVE

Supposons que la liste contienne maintenant 8 lignes (indexées de 0 à 7) et formatées comme défini précédemment. Modifions la 4ème ligne (index 3) pour que la colonne n°1 ait un fond bleu et des caractères blancs, la colonne n°2 un fond blanc et des caractères noirs, la colonne n°3 un fond rouge et des caractères blancs. Tous les caractères sont centrés dans leur colonne respective.
La ligne est sélectionnée et le texte modifié :

```
SELECT 3 FROM LB_TEST
MOVE « {F[WHITE],B[BLUE],J[C]}Nat »&\
    « - {F[BLACK],B[WHITE],J[C]}Systèmes »&\
    « -{F[WHITE],B[RED],G[],J[C]}Paris » TO LB_TEST
SELECT 0 FROM LB_TEST
```

Les caractères de la colonne n°2 sont encadrés car aucun format G n'est spécifié, par contre les caractères de la colonne n°3 ne sont plus soulignés car G[] a été spécifié.
Sélectionnons ensuite la ligne suivante pour modifier son contenu :

```
SELECT 4 FROM LB_TEST
MOVE « 15-rue-Bleue 9ème arrondissement » TO LB_TEST
SELECT 0 FROM LB_TEST
```

Les caractères ont alors les attributs définis par le format global : colonne n°1 en fond vert avec caractères jaune centrés, etc …

> Le dernier texte spécifié dans un MOVE ou un INSERT n'est jamais tronqué, même si sa taille dépasse la largeur de la colonne précisée dans le champ tabulation. Par exemple, le texte « Fayette 10ème arrondissement » est inscrit en totalité dans la List box. Cette particularité est fort utile pour insérer un titre général dans une List Box.

Insertion de bitmaps

L'instruction suivante indique que la colonne n°1 est une colonne réceptrice de bitmaps :

```
INSERT AT 0 «~-{M[Y]}» TO LB_TEST
```

Par la suite, la bitmap système  peut être insérée de la façon suivante :

```
INSERT AT END «^20-bbb-ccc» TO LB_TEST
```

L'instruction suivante insère la bitmap de handle hbmp% :

```
INSERT AT END hbmp% & «-bbb-ccc» TO LB_TEST
```

Le handle de bitmap, qui est de type INT, doit être concaténé à la chaîne à insérer.

L'instruction suivante permet d'écrire du texte dans la colonne n°1, qui est pourtant déclarée comme colonne réceptrice de bitmap :

```
INSERT AT END «{M[N]}aaa-bbb-ccc» TO LB_TEST
```

L'instruction suivante insère la bitmap  dans la colonne n°3 qui n'a pas été déclarée dans le format global comme réceptrice de bitmap :

```
INSERT AT END « ^20-bbb-{M[Y]}20^ » TO LB_TEST
```

Spécification d'une colonne retaillable

L'instruction suivante indique que la colonne n°1 est une colonne retaillable :

```
INSERT AT 0 «~-{R}» TO LB_TEST
```

Blocage des deux premières lignes

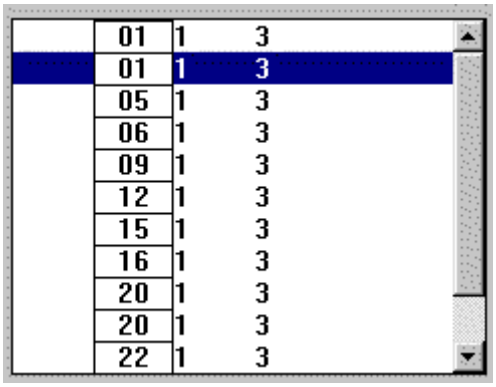
L'instruction suivante bloque les deux premières lignes de la List Box :

```
INSERT AT 0 «~-{T[2]}» TO LB_TEST
```

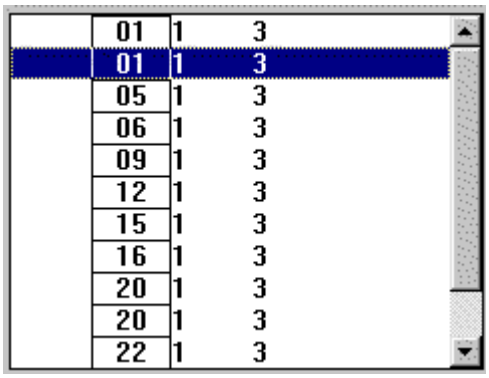
Affichage de la sélection dans une colonne

Soit une fenêtre contenant deux contrôles ListBox (L1 et L2). Le séparateur de colonne est désigné par le caractère '@'.

```
DELETE FROM L1
NOUPDATE L1
INSERT AT 0 "~@{R}@{R,G[LRT],J[C],S[N]}" TO L1
; l'élément sélectionné n'est pas affiché en
; surbrillance
insert ASCENDING " @20@1@3" To L1
insert ASCENDING " @30@1@3" To L1
insert ASCENDING " @12@1@3" To L1
UPDATE L1
```




```
DELETE FROM L2
NOUPDATE L2
INSERT AT 0 "~@{R}@{R,G[LRT],J[C]}" TO L2
; l'élément sélectionné est affiché en surbrillance
insert ASCENDING " @20@1@3" To L2
insert ASCENDING " @30@1@3" To L2
insert ASCENDING " @12@1@3" To L2
UPDATE L2
```



INDEX