

Cible Windows 64 bits pour les outils Nat System

NS-DK / NatStar / NatXtend

Avantages,
mise en œuvre.

Eric Plichon

Table des matières

Nouvelle cible Windows 64 bits	3
Pourquoi passer au 64 bits ?	3
Avantages du système 64 bits	3
Mémoire adressable	3
Un peu d'assembleur	3
Historique des versions 64 bits chez Microsoft (x86)	3
Conclusion	3
Création d'une cible de génération 64 bits	4
Utilisation des TOOLSET 64 bits	4
Contexte du portage vers les cibles 64 bits	5
Principales différences entre le 32 bits et le 64 bits.	5
Changements liés à la taille des pointeurs.	5
Fonction SEND/POST, CALL* OPEN* et PASS	5
Logique d'utilisation	5
Ancienne syntaxe	5
Nouvelle syntaxe	6
Exemple d'utilisation	6
Arithmétique et affectation de pointeurs	7
Arithmétique sur entiers 16/32 bits	7
Arithmétique sur pointeurs	7
Affectation de pointeurs	7
Affectation d'adresses	8
Adaptation et correction des projets existants.	8
Modification d'un projet	8
Lecture et interprétation des Warning, Exemples	8
Verbes NCL	9
Affectation d'adresse dans des entiers	9
Annexes	9
Liste des verbes NCL qui nécessitent des pointeurs	9
Tableau de correspondance entre les types	11

Avantages

Nouvelle cible Windows 64 bits

Avec la version 11, NSDK et NatStar peuvent être livrés en 32 ou en 64 bits

Pour ce faire, l'intégralité des composants des produits Nat System a été portée en C++ 64 bits et les outils (NS-DK, NatStar, NatExtend) peuvent maintenant tourner en mode 64 bits sous Windows.

Ces version 64 bits de NS-DK / NatStar peuvent maintenant générer des applications ou des DLL Windows au format 32 bits ou au format 64 bits.

La cible 64 bits est fournie avec un ensemble de règle de portage des applications.

Pourquoi passer au 64 bits ?

- Tous les nouveaux PC sont livrés avec un processeur 64 bits.
- Le 64 bits offre de bien meilleures performances du processeur et de gestion de la mémoire.
- De plus en plus d'applications et librairies externes sont livrées dans cette architecture : Driver de Base de données (PostgreSQL n'est fourni qu'en 64 bits depuis la version 11).

A terme, il n'y aura plus d'application 32 bits qui tourneront sur les machines Windows.

Avantages du système 64 bits

Mémoire adressable

- Les applications 32 bits peuvent adresse 4 gigas de mémoire maximum.
- Les applications 64 bits peuvent gérer 16 exa octets (16 milliards de giga octets)

Meilleure performance, intégration à l'architecture 64 bits qui se généralise.

Pérennité des processeurs et applications 64 bits.

Un peu d'assembleur

Les processeurs passent de 8 registres généraux 32 bits à 16 registres généraux 64 bit, bien meilleurs pour l'optimisation.

Les opérations sont plus rapides car basée sur des entiers à 8 octets.

Historique des versions 64 bits chez Microsoft (x86)

Les systèmes suivants existent en version 64 bits :

- Windows XP, (2005)
- Windows Server 2003, Windows Server 2008,
- Windows Vista,
- Windows 7
- Windows 8.
- Windows 10

Ces systèmes 64 bits ne supportent plus les sous-systèmes 16 bits

Note : Il s'est écoulé environ 18 ans entre la sortie de Windows 95 (premier 32 bits) et la fin du support 16 bits.

Conclusion

Avec cette cible, Nat System assure la viabilité des applicatifs client pour de nombreuses années sur les futures versions de Windows.

La cible 64 bits est l'assurance pour le futur

Mise en œuvre.


Création d'une cible de génération 64 bits

Utilisation des TOOLSET 64 bits

Avec NS-DK

Lancer NS-DK 64 bits. Ouvrir un projet puis ouvrir le menu « Options/build configuration »

Dans le champ Configuration, créer une nouvelle configuration win64, Dans la Combo box toolset , choisir MSVC-Win64

Cliquer sur l'image bouton « create New Configuration » , une nouvelle configuration de build, prévue pour Windows 64 bits est créée.

Remarque

Il est conseillé de changer les répertoires de génération dans la configuration 64 bits, afin que les fichiers 32 et 64 bits ne se mélangent pas

Avec Natstar

Lancer NatStar 64 bits. Charger un workspace puis ouvrir le menu « Build / Set configuration »

Copie d'un configuration 32 bits existante en nouvelle configuration 64 bits

Sélectionner la configuration 32 bits à copier.

Changer le nom dans la champ « Configuration Name » (par exemple en rajoutant 64 à la fin)

Cliquer sur le bouton « Copy », la nouvelle configuration est créée.

Cliquer sur le bouton « Gen », Choisir le Toolset MSVC-Win64.

Remarque,

Il est possible de mémoriser la liste des librairies depuis la configuration 32 bits, (group box .DLL, champ Libraries) pour le recopier dans celui de la configuration 64 bits.

Cliquer sur « OK » puis sur « Select » la nouvelle configuration 64 bits est créée et peut être générée et testée.

Options de génération 32 et 64 bits pour Windows

Options	En 32 bits	En 64 bits
Toolkit	MSVC-WIN32	MSVC-Win64
Options de génération	/TOOLKIND:MSVC32	/TOOLKIND:MSVC64
Options de compilation	/c /W3 /Gs /Zp /DWIN32	/c /W3 /Gs /DWIN64 /MD
Options de links	/SUBSYSTEM:WINDOWS	/SUBSYSTEM:WINDOWS /MACHINE:X64
Options de librairies d'import (Implib)	/MACHINE:IX86	/MACHINE:X64
Librairies Standards	NSLIB.LIB	NSLIB64.LIB
Répertoires de génération conseillé	(PROJDIR)\GEN	(PROJDIR)\GEN64

Contexte du portage vers les cibles 64 bits

La cible serveur 64 bits est déjà disponible sur les machines Unix suivantes

- IBM AIX
- Linux

Cette nouvelle architecture impose un changement de types pour certaines variables, notamment les pointeurs.

Nat-System a modifié les générateurs NCL → C, le fichier d'interface NSLIB.H, les bibliothèques portables, les bibliothèques d'interface NCL et le moteur NATSTAR afin de s'adapter à ce nouveau mode de fonctionnement.

Principales différences entre le 32 bits et le 64 bits.

Rappel des tailles des principaux types d'entier et pointeurs utilisés en NCL

Type de l'identificateur	Taille en 32 bits		Taille en 64 bits	
	Octets	Bits	Octets	Bits
POINTER	4	32	8	64
INT (2)	2	16	2	16
INT (4)	4	32	4	32
INT (8)	8	64	8	64

En 32 bits, les INT (4) et les pointeurs ont la même taille, Il est donc possible de les substituer l'un l'autre.

En 64 bits, le type INT (4) n'est **plus compatible** avec le type pointeur, on ne peut donc plus faire d'affectations de l'un à l'autre.

Changements liés à la taille des pointeurs

En 64 bits, il n'est pas possible de décomposer et recomposer un pointeur à l'aide des macros HIW() / LOW () et DWORD()

Ces macros servent à faire des conversions 16 <- -> 32. En mode 64 bits on ne peut plus les utiliser pour manipuler des pointeurs.

Les bibliothèques NCL ont été corrigées, là où un type pointeur est réellement requis, il est maintenant expressément déclaré alors qu'en version 32bits, la déclaration pouvait aussi bien être INT que INTEGER ou POINTEUR.

Le type pointeur doit absolument être utilisé pour toutes les APIs et fonctions NCL qui le requièrent.

Fonction SEND/POST, CALL* OPEN* et PASS

Logique d'utilisation

Dans les commandes SEND et POST CALL* et PASS, il fallait envoyer 1, 2, 3 ou 4 entiers 16 bits en paramètres. Maintenant on peut envoyer 1 ou 2 pointeurs ou bien 1, 2, 3 ou 4 entiers 16 bits. Le générateur reconnaît automatiquement le type des paramètres envoyés.

Il est également possible de mixer les 2 types, et d'envoyer un pointeur et 2 entiers 16 bits.

Ancienne syntaxe

SEND événement [, param1 [, param2 [, param3 [, param4]]]] [TO objet [, variable-retour]]

Cette syntaxe permet d'envoyer 1, 2, 3 ou 4 entiers 16 bits.

Pour les utiliser en passant des pointeurs (4 octets) en paramètre, il fallait les décomposer en mots de 2 octets avec les macros LOW et HIW et les recomposer à l'arrivée avec les macros DWORD% ou bien, en réception utiliser le paramètre PARAM12% ou PARAM34% qui est directement un pointeur.

Nouvelle syntaxe

La syntaxe permet de choisir entre l'envoi de 4 entiers de 16 bits ou de 2 pointeurs de (32/64 bits selon le système d'exploitation).

Règle : si les paramètres d'appel sont

- Des entiers, ils sont passés sur 2 octets,
- Des pointeurs, ils sont passés sont 4 ou 8 octets, selon que l'on est en 32 ou 64 bits.

Les paramètres sont 1, 2, 3 ou 4 entiers de 16 bits

SEND événement [, param1 [, param2 [, param3 [, param4]]]] [TO objet [, variable-retour]]

Exemple : SEND USER0, 1, 2, 3, 4 to SELF%

Dans ce mode rien n'est changé, l'événement reçoit 4 (au maximum) entier de 16 bits. Dans PARAM1% PARAM2%, PARAM3% et PARAM4%

Les paramètres sont 2 pointeurs :

Remarque : dans ce mode, on peut passer directement l'adresse d'une variable/

SEND événement [param12 [, param34]] TO objet [, variable-retour]]

```
Local pointer pt
Pt = @s$
SEND USER0, @s$, pt to SELF%
```

Dans ce cas, dans l'événement reçoit les 2 pointeurs dans PARAM12% et PARAM34%

Les paramètres sont des entiers et 1 pointeur (ou bien l'inverse):

```
SEND événement, param1, param2, param34 [TO objet [, variable-retour] ]
```

Ou bien

```
SEND événement, param12, param3, param4 [TO objet [, variable-retour] ]
```

Dans ce cas, 2 INT(2) et un pointeur sont envoyés.

Dans ce mode l'événement reçoit

```
2 entiers de 16 bits dans PARAM1% PARAM2% et un pointeur dans PARAM34%
```

Ou bien

Un pointeur dans PARAM12% et 2 entiers de 16 bits dans PARAM3% PARAM4%

Exemple d'utilisation

Envoi d'un évènement

Ancienne syntaxe :

```
Local pointer p12, pointer p34
send USER0, hiw (p12),low (p12),hiw (p34),low (p34)
```

Nouvelle syntaxe

```
send USER0, p12, p34 to self%
```

Réception des pointeurs dans le code de l'évènement

Ancienne syntaxe

```
local pointer p12, pointer p34
P12 = DWORD% (param1%, param2%)
P34 = DWORD% (param3%, param4%)
```

Nouvelle syntaxe

```
local pointer p12, pointer p34  
  
p12 = param12%  
p34 = param34%
```

Forçage d'un type de paramètre en pointeur

On peut être amené à envoyer comme paramètre un entier et désirer qu'il soit considéré comme un pointeur reçu dans param12% ou param34%

Il faut pour cela faire un transtypage :

```
SEND événement, POINTER intvalue, POINTER intvalue [TO objet [, variable-retour] ]
```

Par exemple : ce mode peut être utilisé pour passer un pointeur NULL

```
SEND USER0, POINTER 0, @text$
```

Arithmétique et affectation de pointeurs

Arithmétique sur entiers 16/32 bits

Il est possible d'utiliser les commandes HIW, LOW et DWORD% sur les INT(2) et les INT(4)

Exemple de commandes valides.

```
local int i2a (2)  
local int i2b (2)  
local int i4 (4)  
  
i2a = 100  
i2b = 200  
  
i4 = DWORD% (i2a,i2b)      ; *** i4 vaut 6553800  
.  
i2a = HIW (i4)              ; *** i2a vaut 100  
i2b = LOW (i4)              ; *** i2b vaut 200
```

Arithmétique sur pointeurs

Il n'est PLUS POSSIBLE de décomposer un pointeur avec HIW et LOW et de le recomposer avec DWORD%

Il est cependant, possible de calculer des déplacements sur les pointeurs

Exemples de commandes valides

```
Local pointer pt, pointer pt2  
Local p%  
  
P% = 3  
Pt = pt + 4  
Pt = pt + p%  
P% = pt2 - pt  
Pt = pt + sizeof (MY_SEG)
```

Affectation de pointeurs

Il est possible d'affecter la valeur d'un pointeur à une string

```
Local pointer pt  
Local s$  
  
S$ = pt  
Pt = pt + 2  
Pt = s$
```

Dans cet exemple, la string contiendra la valeur du pointeur, puis le pointeur sera incrémenté de 2 unité, et enfin retrouvera sa valeur d'origine.

Affectation d'adresses

```
Z$ = « TOTO »
Pt% = @Z$
```

Dans cet exemple, le pointeur contiendra l'adresse de la chaîne z\$.

Adaptation et correction des projets existants.

Pour qu'un projet NSDK puisse correctement passer de 32 à 64 bits, il faut qu'il n'y ait AUCUNE confusion entre pointeur et INT (4)

Le générateur NSGEN qui convertit le NCL en C a été modifié afin de détecter tous les problèmes liés aux pointeurs et INT(4)

Modification d'un projet

Pour que les projets fonctionnent correctement en 64 bits, il faut trouver tous les endroits où les INT(4) et les INTEGER sont utilisés à la place de pointeurs

Modification des paramètres de génération dans NS-DK.

Ouvrir la boîte de dialogue « Option / Build configuration, onglet Générateur

Les paramètres de générations s'affichent. Le niveau de Warning s'affiche dans un spin bouton. Changer sa valeur pour le mettre à « 1 »

Cette indication place le niveau de message d'avertissement à 1 lors de la génération du code C.

Modification des paramètres de génération dans Natstar.

Ouvrir la boîte de dialogue « Générateur » : menu « Build / Set Configuration » bouton « Gen »

Les paramètres de générations s'affichent. Après l'option « /TOOLKIND:<name> », rajouter « /W:1 », la nouvelle option de génération devient donc « /TOOLKIND:<name> /W:1 ».

Cette indication place le niveau de message d'avertissement à 1 lors de la génération du code C.

Liste des principaux problèmes rencontrés lors du portage en 64 bits

- Dans les déclarations, à l'intérieur des segments, en variable globale ou locale les références sur les objets doivent ABSOLUMENT être déclarées sous forme de pointeurs : handle de fenêtre, de DLL, de procédures, paramètres de CALL. Une vérification est nécessaire.
- Les appels aux fonctions SEND et POST peuvent être retouchés (voir chapitre SEND/POST)
- TOUTES les conversions de pointeur \rightarrow entiers par les ordres HIW, LOW, DWORD doivent disparaître. En général, ces fonctions sont liées aux OPEN et CALL, SEND et POST. Il faut adapter le code à la nouvelle syntaxe.
- Certains paramètres des bibliothèques NCL ont été modifiés afin de bien différencier les pointeurs et INT(4), il faut donc modifier les sources du projet pour s'y adapter.

Lecture et interprétation des Warning, Exemples

Exemple de Warning générés.

E. MAIN.PB_TES.EXECUTED(35): Warning: Non-portable POINTER comparison.

E. MAIN.PB_TES.EXECUTED(39): Warning: Non-portable INT/POINTER conversion.

Affectation pointeur \leftrightarrow Entier

```
Local p%
Local pointer pt
Pt = p%
P% = pt
```

Warnings à la génération :

E. MAIN.INIT(5): Warning: Non-portable INT/POINTER conversion.

E. MAIN.INIT(6): Warning: Non-portable INT/POINTER conversion.

Il n'est pas possible de mettre un entier dans un pointeur (la moitié du pointeur est invalide, car à 0)

Il n'est pas possible de mettre un pointeur dans un entier (perte de la moitié de l'information)

Gestion mémoire

```
local p%  
local pointer pt  
  
new MYSEG, p%  
fill p%, sizeof MYSEG, 0  
dispose p%
```

Warnings à la génération :

I. LB64BITS.TESTFILL(61): Warning: Pointer to integer conversion.

I. LB64BITS.TESTFILL(62): Warning: FILL, Conversion to POINTER.

I. LB64BITS.TESTFILL(63): Warning: Non-portable INT/POINTER conversion.

En effet, on ne peut utiliser les commandes de gestions mémoire avec des entiers.

Comparaison pointeur entiers

```
Local pointer pt  
  
if pt <> 1  
...  
Endif
```

Warnings à la génération :

E. MAIN.PB_TES.EXECUTED(100): Warning: Non-portable INT/POINTER conversion.

En effet pout compare un pointeur et un entier, il faut convertir l'entier en pointeur (la moitié du nouveau pointeur est invalide, car à 0)

Remarque : Cette syntaxe peut tout de même est acceptée pour les tests de pointeurs <> 0

Verbes NCL

Les fonctions FILL, MOV, LOADDLL, UNLOADDLL, GETPROC, CAPTURE et d'autres, prennent des pointeurs comme argument. Dans le cas où un entier est utilisé, on obtient le message

Warning: Pointer to integer conversion.

Ou bien

Warning: Non-portable INT/POINTER conversion.

Affectation d'adresse dans des entiers

```
Local p%  
Local s$  
p% = @s$
```

Warnings à la génération :

E. MAIN.PB_TES.EXECUTED(142): Warning: Non-portable INT/POINTER conversion.

En effet, il est IMPOSSIBLE de mettre une adresse dans un entier.

Annexes

Liste des verbes NCL qui nécessitent des pointeurs.

PASS [Param1% or Param12%[, Param2% or Param3% or Param34%[,retVar% or Param3% or Param4% or Param34%[, retVar% or Param4%[, retVar]]]]]

SEND Event [, Param1% or Param12%[, Param2% or Param3% or Param34%[, Param3% or Param4% or Param34%[, Param4%]]]] [TO Object[, returnVariable]]

POST Event [, Param1% or Param12%[, Param2% or Param3% or Param34%[, Param3% or Param4% or Param34%[, Param4%]]]] [TO Object]

CALL WindowName [, retWindowHandle] [USING [Param1% or Param12%[, Param2% or Param3% or Param34%[, retVar% or Param3% or Param4% or Param34%[, retVar% or Param4%[, retVar]]]]]]

CALLH (comme CALL)

STRCALL WindowNameString [, retWindowHandle] [USING [Param1% or Param12%[, Param2% or Param3% or Param34%[, retVar% or Param3% or Param4% or Param34%[, retVar% or Param4%[, retVar]]]]]]

STRCALLH (comme STRCALL)

OPEN WindowName, 0 | ParentWindowHandle [, retWindowHandle] [USING Param1% or Param12%[, Param2% or Param3% or Param34%[, Param3% or Param4% or Param34%[, Param4%]]]]

OPENH (comme OPEN)

OPENS (comme OPEN)

OPENSH (comme OPEN)

STROPEN WindowNameString, 0 | ParentWindowHandle [,retWindowHandle] [USING Param1% or Param12%[, Param2% or Param3% or Param34%[, Param3% or Param4% or Param34%[, Param4%]]]]

STROPENH (comme STROPEN)

STROPENS (comme STROPEN)

STROPENSH (comme STROPEN)

INSTRUCTION CAPTURE [window-handle]

INSTRUCTION CHANGE [window-handle] TO x-expression, y-expression [, width-expression, height-expression]

INSTRUCTION CLOSE [window-handle [, return-variable]]

INSTRUCTION FILL destination-address, length, integer

FUNCTION GETCLIENTHEIGHT% [(window-handle)]

FUNCTION GETCLIENTWIDTH% [(window-handle)]

FUNCTION GETDATA% [(window-handle)]

INSTRUCTION GETPROC DLL-handle, DLLfunctionname, NCLfunctionname

INSTRUCTION INVALIDATE [window-handle]

FUNCTION ISMAXIMIZED% [(window-handle)]

FUNCTION ISMINIMIZED% [(window-handle)]

INSTRUCTION LOADDLL DLL-name, DLL-handle

FUNCTION MAINWINDOW%

INSTRUCTION MAXIMIZE [window-handle]

INSTRUCTION MINIMIZE [window]

INSTRUCTION MOV source-address, destination-address, length

INSTRUCTION NEW segment-name | size, segment-address [, public-name]

FUNCTION PARAM12%

FUNCTION PARAM34%

FUNCTION PARENTWINDOW% [(window-handle)]

FUNCTION SELF%

INSTRUCTION STARTTIMER number, time [TO window-handle]

INSTRUCTION STOPTIMER number [TO window-handle]

INSTRUCTION STRCALL window-name-string [, window-handle] [USING param1[, param2[, param3[, param4]]]]

INSTRUCTION UNLOADDLL DLL-handle

Tableau de correspondance entre les types

Type NCL, types C Nat-System et les types C natifs en 32 et 64 bits

Type NCL	Type C dans nslib.h	32 bits		64 bits	
		Type C	Taille en octets	Type C	Taille en octets
INT(1)	NS_SHORTINT	Signed char	1	Signed char	1
INT(2)	NS_INT	Short	2	Short	2
INT(4)	NS_LONG	long	4	int	4
INT(8)	NS_INT64	Long long	8	Long long	8
INTEGER	NS_INTEGER	int	4	int	4
POINTER	NS_PTR	void *	4	void *	8
NUM(4)	NS_FLOAT	float	4	float	4
NUM(8)	NS_REAL	double	8	double	8
NUM(10)	NS_EXTENDED	double	8	double	8
STRING	NS_PCHAR	Char *	4	char *	8
CSTRING	NS_PCHAR	Char *	4	char *	8
CHAR	NS_CHAR	Char	1	char	1
SEGMENT	NS_PTR	Void *	4	void *	8