Chapter 1 Porting to 64 bit

The portability of a program is its ability to function without major changes on computers and/or systems of different types.

Nat System development environments take account of this need and allow you to develop portable applications on 64-bit platforms easily.

In effect, NatStar 3.64 and NS-DK 3.64 enable you to integrate both new 64-bit applications and your current 32-bit applications. However, some corrections to your existing applications are necessary in order to ensure their interoperability.

Similarly, precautions when writing are now essential when using POINTER types.

You will find in this chapter

- The main differences between 32 and 64-bit systems.
- How to port a Nat System application to 64-bit.
- How to adapt and correct existing projects.

Contents

NatStar Architecture 1-4
Changes made 1-6
Changes to runtime sources 1-6
Optimizing transtyping 1-6
Changes to instructions 1-6
Help in defining types via [Shift]-[F1] 1-6
64-bit target platforms 1-8
Differences between 32 and 64-bit systems 1-9
Integer and pointer type sizes 1-9
Changes linked to the size of pointers 1-9
Arithmetic and assigning pointers 1-10
Decomposition/re-composition of integers and pointers 1-10
Arithmetic on pointers 1-10
Assignment of pointers 1-10
Assignment of addresses 1-11
Instructions SEND, POST, CALL*, OPEN* and PASS 1-11
The logic of their use 1-11
Former syntax 1-11
New syntax 1-11
Where the parameters are 1,2,3 or 4, 16-bit integers
Where the parameters are two pointers
Where the parameters are two integers and a pointer
Where the parameters are two pointers and one integer
Example of use 1-13
Sending an event
Receipt of pointers in the event code
Changing a parameter type into pointer 1-13
The process of porting an application 1-14
First step: Installation 1-14
Second step: Migration of NCL services 1-14
Third step: Changing the generation parameters 1-14
NatStar 1-14
NS-DK 1-15
Fourth step: Changing the source code 1-15
Fifth step: Generation 1-16
Warning messages 1-17
Assignment between pointer and integer 1-17
Memory management 1-17

Pointer integer comparison 1-18
NCL verbs 1-18
Address assignment in integers 1-19
Appendices....1-20
Choice of handle type for 64-bit resources 1-20
List of NCL verbs requiring pointers 1-21
Table of correspondence between NCL types, Nat System C types and native 32 and 64-bit C types 1-23

NatStar Architecture

Porting to 64-bit targets has made changes to some NatStar and NS-DK modules necessary:

- The NSGEN generator which converts NCL into C has been changed so as to detect all the problems linked to POINTER and INT(4) types.
- The NSLIB.H interface file.
- Some portable libraries.
- Some NCL interface file libraries.
- The NatStar runtime.
- The Oracle and Tuxedo drivers.

This new architecture has meant a change in type for some variables, in particular pointers.

The C generated by NatStar 3.64 and higher versions is different from that generated by NatStar 3.00, even for the Windows 32-bit platform.

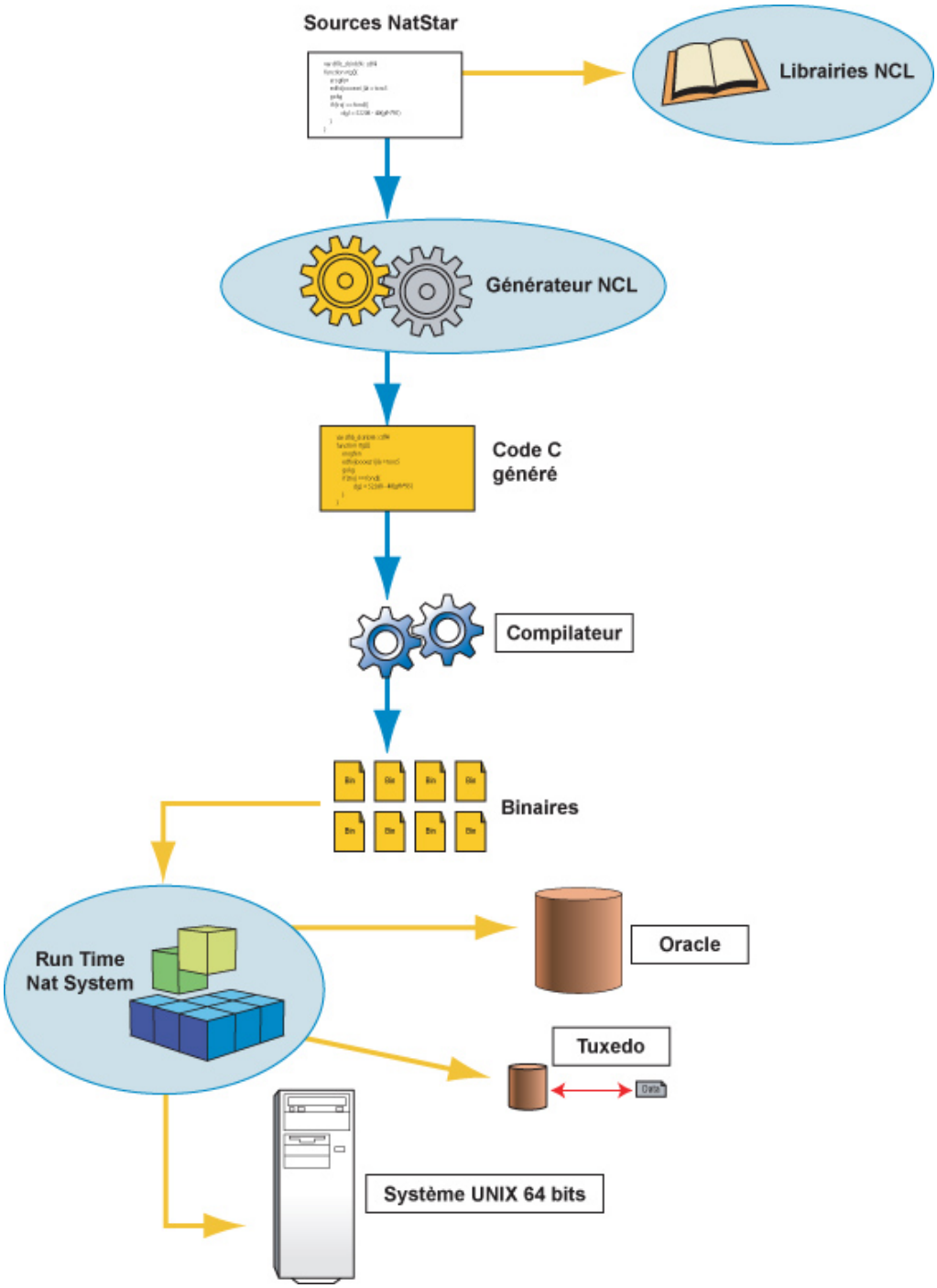

Figure - New NatStar architecture

Changes made

Changes to runtime sources

The runtime system sources of the NatStar and NS-DK tools have been changed, in particular in order to maintain the difference between pointers and integers and to integrate the new configuration of certain instructions.

Optimizing transtyping

Nat System allows the conversion of integers into pointers using the keyword POINTER in the CALL*, OPEN*, SEND, POST, STRCALL*, STROPEN* and PASS instructions. It should be positioned in front of the expression to be converted.

In the PARAM1% and PARAM3% locations, it is possible to replace the LOW *int_expr%*, HIW *int_expr%* operators by POINTER *int_expr%*.

Alike if an expression as ptr, ptr+*int_expr%*, ptr-*int_expr%* or *int_expr%*+ptr (where ptr is a POINTER type variable, or an address variable, or a literal STRING) appears at a PARAM1% and PARAM3% locations, the pointer is passed as PARAM12% or PARAM34% without prefixing by the POINTER type.

Example:

```
SEND USER0,
SEND USER1, POINTER $12345678, 12345, 23456
;sending USER0 to the current window with Param12%=$12345678, Param3%=12345 et
;Param4%=23456
```

Changes to instructions

Passing the parameters PARAM1%, PARAM2%,… to the instructions (SEND, OPEN, CALL, PASS …) handling behavior events has been improved in order to permit 64-bit portability.

Help in defining types via [Shift]-[F1]

In order to make changes to types easier, the NCL editor help displaying the prototypes of instructions and functions has been improved.

By pressing the [Shift]-[F1] keys, the exact prototype appears at the top of the page with the types of parameters shown explicitly.

NCL type used	Prototype display
INT(1)	INT(1)
INT(2)	INT(2)
INT(4), INTEGER, %, INT	INT(4)
POINTER	POINTER

64-bit target platforms

Any application developed with Nat System development environments can be ported to one of the following UNIX targets, as long as certain rules are followed during its development:

- HPUX 11.0
  C compiler: c89, Version : HP92453-01 B.11.11.08 HP C Compiler
  Tuxedo : 8.1
  Oracle : 10.1

- IBM AIX 5L (version 5.2)
  C compiler: Visual Age C++ Professional / C for AIX Compiler, Version 6.
  Tuxedo : 8.1
  Oracle : 10.1

☞ Using Nat System 3.64 tools, a client computer on a 32-bit platform can communicate with a 64-bit server.

Differences between 32 and 64-bit systems

This section presents the main differences made to Nat System tools in order to facilitate 64-bit porting.

Integer and pointer type sizes

Integer and pointer type sizes in the NCL language are different according to the platform, 32 or 64-bit.

Type of identifier	Size in 32 bits		Size in 64 bits	
	Bytes	Bits	Bytes	Bits
POINTER	4	32	8	64
INT(4)	4	32	4	32
INT(2)	2	16	2	16

In a 32-bit system, the types INT(4) and POINTER are the same size. It is therefore possible to substitute one for the other.

In a 64-bit system, the type INT(4) **is no longer** compatible with the type POINTER so assignment can no longer be made from one to the other.

Changes linked to the size of pointers

This change affects:

- Comparisons between pointer and integer types.
- The breakdown of a pointer into two, two-byte integers. Thus the HIW and LOW operators and the DWORD% function can no longer be used to manipulate pointers.
- Sending POINTER type parameters to an event using the SEND, POST, CALL, … functions.
- In the 32-bit version, a pointer was declared as either NT, INTEGER or POINTER. The NCL libraries have been corrected so as to declare a type POINTER for a pointer explicitly. The type POINTER must therefore now be used for all NCL instructions and functions using a pointer.

Arithmetic and assigning pointers

Decomposition/re-composition of integers and pointers

It is possible to use the commands HIW, LOW and DWORD% on the INT(2) and INT(4) types.

An example of valid commands:

```
local int i2a (2)
local int i2b (2)
local int i4 (4)

i2a = 100
i2b = 200

i4 = DWORD% (i2a,i2b)       ;  *** i4 equals 6553800
i2a = HIW (i4)              ;  *** i2a equals 100
i2b = LOW (i4)              ;  *** i2b equals 200
```

It is NO LONGER possible to decompose a pointer with the operators HIW and LOW and to re-compose it with the function DWORD% in 64-bit mode.

Arithmetic on pointers

It is possible to calculate shifts on pointers.

An example of valid commands:

```
Local pointer pt, pointer pt2
Local p%

P% = 3
Pt = pt + 4
Pt = pt + p%
P% = pt2 – pt
Pt = pt + sizeof (MY_SEG)
```

Assignment of pointers

It is possible to assign the value of a pointer to a CSTRING and vice-versa.

Example:

The character string CSTRING will here contain the value of the pointer, then the pointer will be incremented by two units, and finally regain its original value.

```
Local pointer pt
Local s$

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

Assignment of addresses

In this example, the pointer will contain the address of the string z$.

```
Z$ = « TOTO »
Pt% = @Z$
```

Instructions SEND, POST, CALL*, OPEN* and PASS

The logic of their use

The instructions SEND, POST, CALL*, STRCALL*, OPEN*, STROPEN* and PASS enable you to send 1, 2, 3 or 4, 16-bit integers as parameters.

In a 32-bit system, it was possible to send a pointer by breaking it down into two 16-bit integers using the HIW and LOW instructions.

With NatStar and NS-DK 3.64, you can now send 1 or 2 pointers, or even 1, 2, 3 or 4, 16-bit integers. The generator automatically recognizes the type of parameter sent.

It is also possible to link the two types, and send a pointer and 2, 16-bit integers.

☞ In the rest of the section, we will use the syntax of the SEND event as an example. However, the demonstration is also valid for the POST, CALL*, OPEN*, STROPEN*, STRCALL* and PASS instructions.

Former syntax

SEND event [, param1 [, param2 [, param3 [, param4]]]] [**TO** object [, variable-return]]

This syntax lets you send from 1 to 4, 16-bit integers.

In order to use them by passing pointers (4 bytes) as parameters, you need to break them down into 2-byte words using the LOW and HIW operators and re-compose them on arrival with the DWORD% function. On receipt, it was necessary to use the parameter PARAM12% or PARAM34% which is directly a pointer.

New syntax

The new syntax lets you choose between sending 4, 16-bit integers or 2, 32/64 bit pointers according to the operating system.

PARAM12% and PARAM34% now correspond to pointers. PARAM1%, PARAM2%, PARAM3% and PARAM4% remain 16-bit integers.

In consequence, on a **64-bit platform**, PARAM12% is no longer equivalent to DWORD%(PARAM2%, PARAM1%), and PARAM34% no longer equivalent to DWORD%(PARAM4%, PARAM3%).

If the expressions passed as parameters are of the POINTER type, then the instructions use PARAM12% instead of PARAM1% and PARAM2%, or PARAM34% instead of PARAM3% and PARAM4%.

If an expression of the type POINTER is passed without a change of type as INT to the locations expected for PARAM2% or PARAM4%, you will get a warning or an error message.

Where the parameters are 1,2,3 or 4, 16-bit integers

SEND event [, param1 [, param2 [, param3 [, param4]]]] [**TO** object [, variable-return]]

Example:

```
SEND USER0, 1, 2, 3, 4 to SELF%
```

In this mode nothing is changed, the event receives a maximum of four 16-bit integers in PARAM1%, PARAM2%, PARAM3% and PARAM4%.

Where the parameters are two pointers

In this mode, you can pass the address of a variable directly.

SEND event [param12 [, param34]] **TO** object [, variable-return]]

Example:

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

In this case, the event receives the two pointers in PARAM12% and PARAM34%.

Where the parameters are two integers and a pointer

SEND event, param1, param2, param34 [**TO** object [, variable-return]]

In this case, two integers INT(2) and a pointer are sent.

In this mode, the event receives two 16-bit integers in PARAM1%, PARAM2% and a pointer in PARAM34%.

Where the parameters are two pointers and one integer

SEND event, param12, param3, param4 [**TO** object [, variable-return]]

In this case, the event receives one pointer in PARAM12%, and two 16-bit integers in PARAM3% and PARAM4%.

Example of use

Sending an event

Former syntax:

```
Local pointer p12, pointer p34

send user0, hiw (p12),low (p12),hiw (p34),low (p34)
```

New syntax:

Receipt of pointers in the event code

Former syntax:

```
local pointer p12, pointer p34

P12 = DWORD% (param1%,param2%)
P34 = DWORD% (param3%,param4%)
```

New syntax:

```
local pointer p12, pointer p34

p12 = param12%
p34 = param34%
```

Changing a parameter type into pointer

You may need to send an integer as a parameter and want it to be considered as being a pointer received in PARAM12% or PARAM34%.

To do this you must carry out transtyping by giving the keyword POINTER before the value to be converted:

SEND event, POINTER intvalue, POINTER intvalue [**TO** object [, variable-return]]

For example, this mode can be used to pass a NULL pointer:

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

The process of porting an application

In order for an NS-DK or NatStar project to be changed correctly from 32 to 64 bits, there should be NO confusion between POINTER and INT(4). This section shows you how to adapt your existing applications to 64-bit mode.

First step: Installation

1. In order to be able to work in 64-bit mode, UNIX 64-bit servers are needed.
2. Identify the Oracle and Tuxedo modules used by the application and make sure they are configured correctly on the development computer and the server.
3. Finally, install the NatStar or NS-DK 3.64 development environments on the Windows machine and the Nat System runtime on the UNIX servers.

Second step: Migration of NCL services

Nat System has changed the NCL services in order to be able to integrate the 64-bit system. It is necessary, therefore, to import the new versions of these services.

In NatStar 3.64, enable the menu *Options/Services*, and the *Modify Services* box appears. In the *Options/Services* dialogue box, replace the installed services by those from NatStar 3.64.

Third step: Changing the generation parameters

In order to place the level of the warning message at the highest level during generation of the C code, execute the following instructions depending on the tool used.

NatStar

Enable the *Options/Configuration* menu, the window *Select Default Configuration* is displayed. Select a configuration and click on the button *Gen …*, the *Generator dialogue box* appears.

In the field *Generator Options*, after /TOOLKIND:<*name*>, add /W:1. The new generation option becomes therefore /TOOLKIND:<*name*> /W:1.

☞ /TOOLKIND indicates the family of tools used.

Example:

```
/TOOLKIND:MSVC32 /W:1
```

NS-DK

Enable the *Project/Generation options setup* menu, the *Project generation* window is displayed. Click on the button *Generator*, the *Setup Generator* dialogue box appears.

The Setup Generator box enables you to configure, for the current configuration file, the set of source files generated by NS-Gen: presentation, size, particular code, …

In the field *Options*, after /TOOLKIND:<*name*>, add /W:1. The new generation option becomes therefore /TOOLKIND:<*name*> /W:1.

☞ /TOOLKIND indicates the family of tools used.

Example:

```
/TOOLKIND:MSVC32 /W:1
```

Fourth step: Changing the source code

In order for projects to function correctly in 64 bits, you need to change the existing code and replace all the INT(4)s and INTEGERs used in the place of pointers. The following list shows the main points that need to be checked:

- Look in the code for the declarations HIW, LOW, DWORD% in order to analyze whether they represent pointers that have been decomposed. **All** conversions between pointer and integers by the HIW, LOW, DWORD% commands must disappear. In general, these functions are linked to the instructions OPEN, CALL, SEND and POST. The code must be adapted to the new syntax.

- In the declarations, within the segments, as global or local variables, references to objects MUST be declared in the form of pointers: window handle, DLL, procedures. Verification is necessary.

- For sequences and all THINGS verbs, replace the INT(4) types of arguments and return variables by POINTERS.

- Make sure that the functions F_OPEN%, F_CREATE%, T_OPEN% et T_CREATE% do actually manipulate integers and not pointers.

- Variables declared as INT(4) must be converted into POINTERS in all cases where they are used as an address, except in the case of file handles (h%=T_CREATE%). This concerns the event code of functions/instructions, arguments of functions/instructions, segments and global variables.

Fifth step: Generation

In order to display any 64-bit porting errors, generate the application code by unticking the *No Warnings* box. Errors appear in the Log window.

Correction is made easier in the NCL editor by pressing the [Shift]-[F1] keys which display the parameter types precisely.

☞ For further information, please refer to the section “Help in defining types via [Shift]-[F1]” in this manual.

Warning messages

This section presents the warning messages listed. For each of them, the possible causes are indicated. The general action to be taken to remedy them is to no longer use integers in the place of pointers. Compatibility between these two types no longer being guaranteed in the 3.64 versions.

☞ Some warnings concerning code internal to the NatStar tool have been considered as being of no consequence.

Example of warnings generated:

```
E. MAIN.PB_TES.EXECUTED(35): Warning: Non-portable POINTER comparison.
E. MAIN.PB_TES.EXECUTED(39): Warning: Non-portable INT/POINTER conversion.
```

Assignment between pointer and integer

Code:

```
Local p%
Local pointer pt

Pt = p%
P% = pt
```

Warning messages on generation:

```
E. MAIN.INIT(5): Warning: Non-portable INT/POINTER conversion.
E. MAIN.INIT(6): Warning: Non-portable INT/POINTER conversion.
```

Cause:

It is not possible to put an integer in a pointer (half the pointer is invalid, because it is 0).

It is not possible to put a pointer in an integer (half the information is lost).

Memory management

Code:

```
local p%
local pointer pt

new MYSEG, p%
fill p%, sizeof MYSEG, 0
dispose p%d
```

Warning messages on generation:

```
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.
```

Cause:

You cannot use memory management commands with integers.

Pointer integer comparison

Code:

```
Local pointer pt

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

Warning message on generation:

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

Cause:

To compare a pointer and an integer, you need to convert the integer into a pointer (half the new pointer is invalid as it is 0).

☞ This syntax may perhaps be accepted for testing pointers that are not zero.

NCL verbs

The instructions FILL, MOV, LOADDLL, UNLOADDLL, GETPROC, CAPTURE, … take pointers as an argument. In a situation where an integer is used, you receive the following message:

```
Warning: Pointer to integer conversion.
```

or perhaps:

```
Warning: Non-portable INT/POINTER conversion.
```

Cause:

As compatibility between integer and pointer types is no longer guaranteed, it is necessary to use pointers as arguments to these instructions.

Address assignment in integers

Code:

```
Local p%
Local s$

p% = @s$
```

Warning message on generation:

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

Cause:

It is now IMPOSSIBLE to put an address in an integer.

Appendices

Choice of handle type for 64-bit resources

Type of resource	Example of use	Type of handle used
Window (SCR)	SELF%, MAINWINDOW%	POINTER
Binary and text file	T_OPEN%, F_OPEN%	INT(4)
XML document	NWX_DOCUMENT_LOAD%, NWX_DOCUMENT_OUTPUT%	POINTER
XML node	NWX_NODE_OUTPUT%, NWX_NODE_OUTPUTFILE%	POINTER
HTTP	NS_HTTP_CONNECT, NS_HTTP_EOF	POINTER
Soap	NS_SOAPPARAM_SET_TYPE	POINTER
Memory and segment	NEW, DISPOSE	POINTER
DLL, dynamic library	LOADDLL, GETPROC	POINTER
Function address	Pt=@myFunc%	POINTER
Thing, handle of an instance	THINGS_*	POINTER ou THING
Printing session	REP_OPEN%	POINTER
Report	REPORT_OPEN%	POINTER
SQL cursor	SQL_OPENCURSOR%, SQL_EXEC_LONGSTR, SQL_OPENTHECURSOR%	INT(2)
Handle of the database context	THINGS_GET_DB%, THINGS_DB_*%	POINTER
Handle of an instance	THINGS_CHECK%, THINGS_SET_VAR%	POINTER
Handle of a memory version	THINGS_BEGIN, THINGS_ANY_CHANGE%	POINTER
Handle of a class	THINGS_CLMG_*, THINGS_REGISTER_CLASS	POINTER
Handle of the search sequence	THINGS_DB_SELECT_*	POINTER
Handle of the sequence	THINGS_SEQUENCE%	POINTER

Type of graphical resource	Example of use	Type of handle used
BMP	DELETEBMP, GETBMPSIZE	INT(4)
Graphe	DRAW_BITMAP, DRAW_TEXT	INT(4)
TreeBox	TREE_ATTACH, TREE_INSERT%	INT(4)
DDE	ADVISEDDE%, CLOSEQUEUE	INT(4)
Classifier control	TAB_ADD_TAB%, TAB_LAST_ID%	POINTER
Sheet control	STV_APPEND%, STV_CREATE%	POINTER
Sheet node	STV_NODE_FROM_LINE%, STV_NODE_FROM_POS%	POINTER
ActiveX		POINTER

List of NCL verbs requiring pointers

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 (like 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 (like STRCALL)

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

OPENH (like OPEN)

OPENS (like OPEN)

OPENSH (like OPEN)

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

STROPENH (like STROPEN)

STROPENS (like STROPEN)

STROPENSH (like STROPEN)

CAPTURE [window-handle]

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

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

FILL destination-address, length, integer

GETCLIENTHEIGHT% [(window-handle)]

GETCLIENTWIDTH% [(window-handle)]

GETDATA% [(window-handle)]

GETPROC DLL-handle, DLLfunctionname, NCLfunctionname

INVALIDATE [window-handle]

ISMAXIMIZED% [(window-handle)]

ISMINIMIZED% [(window-handle)]

LOADDLL DLL-name, DLL-handle

MAINWINDOW%

MAXIMIZE [window-handle]

MINIMIZE [window]

MOV source-address, destination-address, length

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

PARAM12%

PARAM34%

PARENTWINDOW% [(window-handle)]

SELF%

STARTTIMER number, time [**TO** window-handle]

STOPTIMER number [**TO** window-handle]

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

UNLOADDLL DLL-handle

Table of correspondence between NCL types, Nat System C types and native 32 and 64-bit C types

NCL Type	C Type in nslib.h	32 bits		64 bits	
		C Type	Size in bytes	C Type	Size in bytes
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
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	Long double	8	Long double	8