

# NatStar

---

Version 5.00 Edition 1

# NS-DK

---

Version 5.00 Edition 1

# NatWeb

---

Version 4.00 Edition 1

# Informix



*Information in this document is subject to change without notice as a result of changes in the product. The software described in this document is furnished under a license agreement. The software may be used or copied only in accordance with the terms of the agreement. It is illegal to reproduce the software on any medium unless specifically authorized within the terms of the agreement. No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose other than the purchaser's personal use without the written permission of Nat System.*

© 2006 Nat System, All rights reserved



# Contents

About this Manual.....	iii
Supported configurations .....	iii
Development environment .....	iii
Client environment.....	iii
Server environment .....	iii
Relationship to other manuals .....	iii
What's new in this edition .....	iv
Organization of the manual.....	iv
Conventions.....	v
<b>Chapter 1 Informix interface</b>	<b>1-1</b>
Introduction .....	1-3
Correspondance between drivers and Informix versions .....	1-3
Installation .....	1-4
Informix Interface: Limitations .....	1-4
Functional categories of NSnnINFx.....	1-5
Initializing and stopping application use of the DBMS .....	1-5
Opening and closing a database.....	1-5
Choose the DBMS and the base for a SQL statement .....	1-5
Executing a SQL command: SELECT, INSERT, UPDATE, CREATE TABLE ...	1-5
Managing the current database.....	1-5
Managing the cursor.....	1-5
Running a stored procedure .....	1-6
Handling errors.....	1-6
NSnnINFx library reference .....	1-7
NS_FUNCTION extensions.....	1-29



# About this Manual

This is the Informix manual for Nat System's development tools. This manual describes the Informix interface allowing the access to an Informix database.

## Supported configurations

### Development environment

- Windows 32 bits : 95, 98, NT 4.0, 2000

### Client environment

Operating system	DBMS drivers available
Windows 32 bits	Informix 7.2, 9.2

### Server environment

Operating system	DBMS drivers available
Windows NT Windows 2000 32 bits	Informix 7.2, 9.22
AIX 4.1 32bits	Informix 7.2, 7.2 XA
AIX 4.3 32bits	Informix 7.2, 7.2 XA Informix 9.22, 9.22 XA
HP-UX 10.x 32 bits	Informix 7.2, 7.2 XA
HP-UX 11.x (Risc 2) 32 bits	Informix 7.2, 7.2 XA Informix 9.22, 9.22 XA
Sun Solaris 2.5 32 bits	Informix 7.2, 7.2 XA
Sun Solaris 2.7 32 bits	Informix 7.2, 7.2 XA Informix 9.22, 9.22 XA

## Relationship to other manuals



Before reading this manual you are expected to have read the « Overview » and « Getting started » manuals. You should not need to use this manual unless you have been advised to do so or if you are already an experienced Nat System developer. If this is the case, you can use this manual to learn in detail about the components it describes.



Strictly speaking, in standard use of NatStar's Information Modeling tool, you don't have to program data accesses yourself. The Information Modeling engine takes care of that. In this case, you don't need to look at the libraries described in this manual. However this manual will prove usefeul if you want to program your applications' data accesses yourself.

## What's new in this edition

In this edition, the structure of the older manual entitled « Database Access Reference » has been modified to ease the using and to provide faster ways of finding the information you need. Thus, each library is described in a specifical manual.

## Organization of the manual

This manual contains one chapter, which describes the set of API components of the Informix interface.

### **Chapter 1**

#### **Informix interface**

Describes the functions of the NSnnINFx library associated with the Informix database.



# Conventions

## Typographic conventions

<b>Important term</b>	Important terms are printed in <b>bold</b> .
<i>Interface component</i>	The names of windows, dialog boxes, controls, buttons, menus and options are printed in <i>italics</i> .
[F9]	Function key names appear in square brackets.
FILENAME	Filenames are printed in UPPERCASE.
syntax example	Syntax examples are printed in a fixed-width font.

## Notational conventions

- A round bullet is used for lists
- ◆ A diamond is used for alternatives
- 1. Numbers are used to mark the steps in a procedure to be carried out in sequence

## Operating conventions

Choose XXX \ YYY	This means you need to open the XXX menu, then choose the YYY command (option) from this menu. You can perform this action using the mouse or mnemonic characters on the keyboard.
Click the XXX \ YYY button	This means you need to display the tool bar named XXX, then click the YYY button in this tool bar (the name of each button is shown by its help bubble). You can only perform this action with the mouse.
Choose the XXX button	This means you need to choose the XXX button in a dialog box. You can perform this action using the mouse or mnemonic characters on the keyboard.

## Icon codes



**Comment**, note, etc.



**Reference** to another part of the documentation



**Danger**: precaution to be taken, irreversible action, etc.



**Suggestion**: helpful hints, etc.



**To go a step further**: level of detail or expertise greater than the average level of the document



Indicates specific information on using the software under DOS-Windows (all versions)



Indicates specific information on using the software under DOS-Windows 32 bits



Indicates specific information on using the software under OS/2-PM 2.x or DOS-Windows 32 bits



Indicates specific information on using the software under Unix systems

## Chapter 1

# Informix interface

The NSnnODCx library allows your applications built with Nat System development tools to interface with client versions of Informix.



NatStar 5.00 supports in environment Windows 32 bits: Informix 7.2, 9.2 and 10.0.



To ease the writing in the documentation NSnnINF7 and NSnnINF9 will be gathered in NSnnINFx generic name.

### ***This chapter explains***

- How to install this library
- The components of this library, arranged in functional categories
- The reference of the components of this library

## Contents

Introduction.....	1-3
Correspondance between drivers and Informix versions	1-3
Installation .....	1-4
Informix Interface: Limitations	1-4
Functional categories of NSnnINfx .....	1-5
Initializing and stopping application use of the DBMS	1-5
Opening and closing a database	1-5
Choose the DBMS and the base for a SQL statement	1-5
Executing a SQL command: SELECT, INSERT, UPDATE, CREATE TABLE ...	1-5
Managing the current database	1-5
Managing the cursor	1-5
Running a stored procedure	1-6
Handling errors	1-6
NSnnINfx library reference .....	1-7
NS_FUNCTION extensions .....	1-29

## Introduction

This library enables applications to interact transparently with the services provided by the Informix DBMS. This library can be accessed through NCL language, using a number of instructions and functions.

The NSnnINFx library allows your applications built with Nat System's development tools to interface with client versions of Informix databases.

### Correspondance between drivers and Informix versions

The name of the driver is used with THINGS\_DB\_INIT instruction for NatStar and SQL\_INIT instruction for NatStar, NatWeb and NS-DK.

The following table presents Informix versions and the corresponding drivers.

Informix's versions	Driver
<b>Informix 7.2</b>	NSnnINF7.dll
<b>Informix 9.2</b>	NSnnINF9.dll
<b>Informix 10.0</b>	NSnnINF10.dll



nn indicates the target platform:

- w2 for Windows 32 bits
- w4 for Windows 64 bits



To ease the writing on this documentation NSnnINF5, NSnnINF7 and NSnnINF9 will be gathered in NSnnINFx generic name.



For now, Informix DLLs don't support DynStr type.



The older drivers's versions which are not supported by Nat System in Windows 32 bits environment are installed in the CONTRIB(S) folder: NSw2INF.dll.

## Installation

Copy the file NSnnINFx.DLL into the directory that contains the DLLs for your Nat System environment (e.g. C:\NATSTAR\BIN, C:\NATWEB\BIN...).



The SQL libraries supplied with your Nat System development tools interface with the DLLs supplied by the DBMS manufacturer. In some cases, a utility also needs to be run. Check your configuration using the manuals supplied by your DBMS vendor.

### Informix Interface: Limitations

The TEXT and BYTE data types, which are specific to **online**, have not been implemented in this library.

## Functional categories of NSnnINFx

Here is a list, arranged by functional category, of the instructions, functions and constants in the NSnnINFx library.

### Initializing and stopping application use of the DBMS

SQL_INIT .....	1-8
SQL_STOP .....	1-9

### Opening and closing a database

SQL_OPEN.....	1-10
SQL_CLOSE.....	1-11

### Choose the DBMS and the base for a SQL statement

AT command.....	1-12
-----------------	------

### Executing a SQL command: SELECT, INSERT, UPDATE, CREATE TABLE ...

SQL_EXEC.....	1-13
SQL_EXECSTR .....	1-16
SQL_EXEC_LONGSTR .....	1-18

### Managing the current database

NS_FUNCTION CHANGEDBCNTX.....	1-30
-------------------------------	------

### Managing the cursor

SQL_OPENCURSOR%.....	1-20
SQL_CLOSECURSOR	
<b>Erreur ! Signet non défini.</b>	
NS_FUNCTION DEFAULTCURSOR_TOP .....	1-31
NS_FUNCTION DEFAULTCURSOR_DEFAULT .....	1-33

## Running a stored procedure

### *Stored procedure*

- A **stored procedure** is a bit of code, written in a owner database language, stored in base.

SQL\_PROC .....1-27

## Handling errors

SQL\_ERROR% .....1-23

SQL\_ERRMSG\$ .....1-26



## **NSnnINfx library reference**

## SQL\_INIT instruction

Loads the driver needed to use a given version of Informix for a given target.

**Syntax**                **SQL\_INIT** *DLL\_name*


**Parameter**            *DLL\_name*                CSTRING    I            name of the driver to load

### Notes

1. This must be the first SQL\_ instruction called by any application that wants to use a DBMS with NCL: it is responsible for loading the library.
2. The *DLL\_name* parameter should contain the name of the DLL used to access the Informix database: NSnnINFx.



nn indicates the target platform: w2 for Windows 32 bits, p2 for OS/2 32 bits, 02 for OS/2 16 bits and Windows 16 bits.

3.  An error or warning for a SQL\_INIT is often linked to the fact that the Informix interface's dynamic libraries can't be loaded.

### Example

```
SQL_INIT "NS02INF" ;
; The error code must be tested
; A non-zero error code means that the DLL NS02INF
; could not be loaded
IF SQL_ERROR% <> 0
    MESSAGE "Error loading DLL",\
    "Make sure the DLL is in a directory specified" && \
    "by the PATH variable (Windows) or LIBPATH (OS/2)."&& "Application halted."
    HALT
ENDIF
SQL_OPEN "SAMPLE", ""
...
SQL_CLOSE "SAMPLE"
SQL_STOP
```

**See also**                SQL\_STOP, SQL\_INITMULTIPLE%, SQL\_STOPMULTIPLE, SQL\_STOPALL,  
SQL\_ERROR%, SQL\_ERRMSG\$

## SQL\_STOP instruction

Unloads the Informix driver and closes all open databases and cursors.

**Syntax**                **SQL\_STOP**

**Example**

See the example of SQL\_INIT instruction.

**See also**                SQL\_INIT, SQL\_INITMULTIPLE%, SQL\_STOPMULTIPLE, SQL\_STOPALL,  
SQL\_ERROR%, SQL\_ERRMSG\$.

---

## SQL\_OPEN instruction

Opens a database.

**Syntax**                **SQL\_OPEN** *logical-DBname, connection-string*

<b>Parameter</b>	<i>logical-DBname</i>	CSTRING	I	logical name of the database to open
	<i>connection-string</i>	CSTRING	I	connection string for a database

**Notes**

1. Informix allows you to open several databases on a network as described below.
2. **INFORMIX SE**: the default database pathname is specified by the DBPATH environment parameter. This parameter is automatically updated in the machine's configuration file (AUTOEXEC.BAT for Windows and CONFIG.SYS for OS/2) when Informix is installed (see Informix SE documentation).
3. The *connection-string* parameter specifies the command string used to connect to a local or remote database, as follows:  
" [USERID] [ /PASSWORD] [ @HOSTNAME] [ #SERVICENAME]  
where  
[ USERID]                name of the user account on the server  
[ /PASSWORD]           password for the user account  
[ @HOSTNAME]           name of the server on which Informix is installed.  
[ #SERVICENAME]        server node  
If you do not want to use any of these parameters, an empty string is accepted. In this case, the system will use the variables specified in the [ INET CONNECTION] section of the Informix.INI file.
4. **INFORMIX SE** : to use Informix in transactional mode, you can specify the name of the Informix SE log file by adding "#LogFileName" after the logical database name.  
SQL\_OPEN "MYBASE#/USER/MONLOG.LOG" , " "

**Example**

```
SQL_OPEN "BASE1", "SCOTT@SERVER#SQLEXEC"  
IF SQL_ERROR% <> 0  
    MESSAGE "Error BASE1", SQL_ERRMSG$(SQL_ERROR%)  
ENDIF  
...  
...  
SQL_CLOSE "BASE1"
```

**See also**                SQL\_CLOSE, AT, SQL\_ERROR%, SQL\_ERRMSG\$

## SQL\_CLOSE instruction

Closes a database.

**Syntax**                **SQL\_CLOSE** *logical-DB-name*

**Parameter**            *logical-DB-name*                CSTRING    I    logical name of the database to close

**Notes**

1. Although we recommend that you close the databases opened by an application, an SQL\_CLOSE instruction is automatically generated for these databases when an application is closed.

**Example**

See the example of the SQL\_OPEN instruction

**See also**                SQL\_CLOSE, AT, SQL\_ERROR%, SQL\_ERRMSG\$

---


## AT command

Specifies the name of the logical database affected by the SQL statement that follows.

**Syntax**                    *AT logical-DBname, SQL-statement*

<b>Parameters</b>	<i>logical-DBname</i>	CSTRING	I	logical database name
	<i>SQL-statement</i>	CSTRING	I	SQL statement to execute

**Notes**

1. *logical-DBname* was passed as the first parameter to the SQL\_OPEN statement used to open the database.
2. If several databases have been opened simultaneously, the last database opened is taken as the default.
3.  To pass from a database to another, we recommend to use NS\_FUNCTION CHANGEDBCNTX, because AT command could be not supported in the future versions.

**Example**

```
SQL_OPEN "BASE1" , "INFORMIX/INFORMIX96@INF1"
SQL_OPEN "BASE2" , "INFORMIX/INFORMIX96@INF2"
SQL_OPEN "BASE3" , "INFORMIX/INFORMIX96@INF3"
SQL_EXEC SELECT... ; SELECT on BASE3

SQL_EXEC AT BASE2 SELECT... ; SELECT on BASE2
SQL_EXEC AT BASE2 FETCH... ; FETCH on BASE2
SQL_EXEC FETCH... ; FETCH on BASE3
```

**See also**                    SQL\_OPEN, SQL\_CLOSE, SQL\_ERROR%, SQL\_ERRMSG\$

## SQL\_EXEC instruction

Executes an SQL command: SELECT, INSERT, UPDATE, CREATE TABLE ...

**Syntax**                    **SQL\_EXEC** [**AT** *logical-DBname*] *SQL-command* [**USING** *cursor-handle*]

<b>Parameters</b>	<i>logical-DBname</i>	CSTRING	I	logical name of database
	<i>SQL-command</i>	CSTRING	I	SQL command to execute
	<i>cursor-handle</i>	INT(4)	I	value of the cursor

### Notes

1. The SQL command is passed directly without quotes. It can correspond to any Informix SQL command, whether it's a data definition command (CREATE TABLE, CREATE INDEX, ....) or a data manipulation command (SELECT, INSERT, UPDATE, etc.)
2. The AT command can only be used with databases which allow several simultaneous connections. The query is sent to the database specified after the AT command (without quotes and *case-sensitive*). If the AT command isn't specified, the SQL\_EXEC executes on the current database.
3. If USING *cursor-handle* is specified, it indicates which cursor previously opened by SQL\_OPENCURSOR% must be used to execute the SQL command. If no cursor has been opened, the cursor's value is that of DEFAULT\_CURSOR: -1.
4. The SQL command can return values in NCL variables. For this, just pass these variables in parameters.
5. It is possible to pass a segment's field as a data-receiving variable in an SQL query
6. The commands SQL\_EXEC, SQL\_EXECSTR and SQL\_EXEC\_LONGSTR depend on the SQL language accepted by the DBMS in use (*Refer to the DBMS documentation*).
7. For SQL commands that are too long, it is possible to use the special continuation character "\":  

```
SQL_EXEC UPDATE SAMPLE SET COMPAGNY =:A$ \
                                WHERE TOWN =:C$ \
AND \
                                COUNTRY =:D$
```
8. The types of variables recognized by the interface are:
  - INT(1), INT(2), et INT(4),
  - NUM(8), NUM(4),
  - STRING,
  - CSTRING,
  - CHAR.

9. Each database has its own implementation of SQL. In your DBMS documentation, refer to the topics concerning your database for more information about the conversion of NCL types to authorized SQL types.
10. The INTO clause is used by the SELECT and FETCH commands. It defines a list of host variables. Its syntax is:  
 INTO :var1 [:indic1] [, :var2 [:indic2]\  
 [, ... ] ]
11. We suggest using INTO in a SELECT to improve performance because during a FETCH, in each loop, the driver has to analyze the variables of the INTO clause. Using the INTO clause in a FETCH should be restricted to doing things like be entering elements into an array.
12. Always put a ":" before the name of a variable or flag.
13. A flag is an NCL integer variable which can have the following values:
  - ♦ NULL\_VALUE\_INDICATOR (i.e. -1) indicates that the associated NCL variable which precedes it has a NULL value.
  - ♦ Any other value indicates that the associated NCL variable which precedes it has a NOT NULL value, and can therefore be used.
14. In SQL, NULL does not mean 0 or an empty string (""). However, to make it possible to assign a value in all cases, when a column contains a NULL value, a numeric target NCL variable will be assigned a 0 and a string target NCL variable will be assigned an empty string ("").

**Example**

```

LOCAL CODE%,I%,AGE%,IND1%,IND2%
LOCAL COUNTRY$,CITY$,A$,B$
LOCAL TCODE%[10]
LOCAL TCOUNTRY$[10]

CITY$ = "NEW YORK"
; =====
; 1st example
; =====
; ---- Select a subset
SQL_EXEC SELECT CODE,COUNTRY \
          FROM WORLD \
          WHERE TOWN =:CITY$
; ---- Read the first to last entry
WHILE SQL_ERROR% = 0
  SQL_EXEC FETCH INTO :CODE%,:COUNTRY$
  IF SQL_ERROR% = 0
    INSERT AT END CODE% && COUNTRY$ TO LBOX1
  ENDIF
ENDWHILE
; =====
; 2nd example (most efficient)
; =====
; ---- Select a subset
; and read the first entry
SQL_EXEC SELECT CODE,COUNTRY \
          FROM WORLD \
          INTO:CODE%,:COUNTRY$ \
          WHERE TOWN =:CITY$
; ---- Read the second to the last entry

```



```

WHILE SQL_ERROR% = 0
  INSERT AT END CODE% && COUNTRY$ TO LBOX1
  SQL_EXEC FETCH
ENDWHILE

; =====
; 3rd example
; =====
; ---- Select a subset
SQL_EXEC SELECT CODE,COUNTRY \
          FROM WORLD \
          WHERE TOWN =:CITY$
; ---- Read 1st entry to last entry
;       by filling TCODE% and TCOUNTRY$ tables
I% = 0
WHILE (SQL_ERROR% = 0) AND (I% < 10)
  SQL_EXEC FETCH INTO :TCODE%[I%],:TCOUNTRY$[I%]
  I% = I% + 1
ENDWHILE

; =====
; Using flags
; =====
SQL_EXEC CREATE TABLE FAMILY( NAME          VARCHAR(10),\
                               AGE           NUMBER,\
                               CHILDNAME     VARCHAR(10))

FATHER$ = "STEVE"
AGE%    = 35
SON$    = "PETER"
IND1%   = 0
IND2%   = 0
; --- Insert "STEVE",35,"PETER" into table
SQL_EXEC INSERT INTO FAMILY\
          VALUES (:FATHER$:IND1%, :AGE%, :SON$:IND2%)

FATHER$ = "PETER"
AGE%    = 10
IND1%   = 0
IND2%   = NULL_VALUE_INDICATOR
; --- Insert "PETER",10,NULL into table
SQL_EXEC INSERT INTO FAMILY VALUES (:FATHER$:IND1%, :AGE%, :SON$:IND2%)

; ---- The SELECT loop places the listbox LBOX
;       'STEVE's son is PETER'
;       'PETER does not have a son.'
SQL_EXEC SELECT NAME, AGE, CHILDNAME\
          INTO:PERE$:IND1%,:AGE%,:SON$:IND2% \
          FROM FAMILY
WHILE SQL_ERROR% = 0
  ; ---- IND1% is always 0 here
  IF IND2% = -1
    INSERT AT END FATHER$ & "does not have a son." TO LBOX
  ELSE
    INSERT AT END FATHER$ & "'s son " &\
          "is" & SON$ TO LBOX
  ENDIF
  SQL_EXEC FETCH
ENDWHILE

```

See also            SQL\_EXECSTR, SQL\_EXEC\_LONGSTR, SQL\_ERROR%, SQL\_ERRMSG\$

## SQL\_EXECSTR instruction

Executes an SQL command: SELECT, INSERT, UPDATE, DELETE ...

<b>Syntax</b>	<b>SQL_EXECSTR</b> <i>SQL-command</i> [, <i>variable</i> [, <i>variable</i> [, ....]]] [ <b>USING</b> <i>handle-cursor</i> ]			
<b>Parameters</b>	<i>SQL-command</i>	CSTRING	I	SQL order to execute
	<i>variable</i>		I	NCL variable list
	<i>handle_cursor</i>	INT(4)	I	cursor value

### Notes

1. *SQL-command* is either a string *host* variable or a character string containing the SQL command to execute in quotation marks.
2. When you use the SQL\_EXEC instruction, you write the names of the *host* variables directly in the text of the SQL query. When you use the SQL\_EXECSTR instruction, the *host* variables are parameters of the instruction.
3. When you use the SQL\_EXECSTR instruction, each *host* variable is represented in the text of the query by a ":" character. The first ":" corresponds to the first *host* variable passed as a parameter, and so on.
4. The other functions of the SQL\_EXECSTR command are the same as SQL\_EXEC.

### Example

```

LOCAL REQ$, TABLE$, FATHER$, SON$
LOCAL AGE%, IND1%, IND2%, CURS1%

TABLE$ = "FAMILY"
AGE% = 20
REQ$ = "SELECT NAME, AGE, NAMECHILD INTO : : , : : : FROM '" & \
      TABLE$ & "' WHERE AGE > :"
```

```

; ---- Opens a cursor
CURS1%=SQL_OPENCURSOR%

; ---- Select persons older than 20 from
; the FAMILY table
SQL_EXECSTR REQ$, FATHER$, IND1%, AGE%, SON$, IND2%, AGE% USING CURS1%

WHILE SQL_ERROR% = 0
  IF IND2% = NULL_VALUE_INDICATOR
    INSERT AT END FATHER$ & "has no son" TO LBOX
  ELSE
    INSERT AT END "The son of " & FATHER$ & "is" & SON$ TO LBOX
  ENDIF
  SQL_EXEC FETCH USING CURS1%
ENDWHILE
; ---- Closes the cursor
SQL_CLOSECURSOR
```

**See also**        SQL\_EXEC, SQL\_EXEC\_LONGSTR, SQL\_OPENCURSOR%,  
                  SQL\_CLOSECURSOR, SQL\_ERROR%, SQL\_ERRMSG\$

## SQL\_EXEC\_LONGSTR instruction

Executes an SQL statement: SELECT, INSERT, UPDATE, DELETE ...

<b>Syntax</b>	<b>SQL_EXEC_LONGSTR</b> <i>sql-string-address</i> , <i>var-array-address</i> , <i>cursor-num</i>			
<b>Parameters</b>	<i>sql-string-address</i>	INT(4)	I	address of the character string containing the SQL statement to execute
	<i>var-array-address</i>	INT(4)	I	address of the array containing the host variables (or indicators)
	<i>cursor-num</i>	INT(2)	I	cursor value

### Notes

1. The executed statement can contain any SQL command in the host language (DML or DDL). The size of the string depends on the RDBMS used; it is unlimited for certain database engines and limited to 4096 characters for others.
2. *sql-string-address* is the address of the string which contains the SQL command to execute.
3. *var-array-address* is an array of NCLVAR segments which describe the NCL host variables. If your SQL statement does not use any variables, pass 0 in *var-array-address*.
4. When you use the SQL\_EXEC\_LONGSTR instruction, each *host* variable is represented in the text of the query by a ":" character. The first ":" corresponds to the first *host* variable in the array of *host* variables, and so on.
5. The NCLVAR segment and any constants used are declared in the NSDBMS library as follows:

```

SEGMENT NCLVAR
    INT      PTR_VAR ( 4 )
    INT      TYPE_VAR ( 2 )
    INTEGER  SIZE_VAR
    INT      RESERVED ( 4 )
ENDSEGMENT

CONST TYPE_SQL_INT%           0
CONST TYPE_SQL_STRING%       1
CONST TYPE_SQL_CSTRING%      2
CONST TYPE_SQL_NUM%          3

```
6. This array of segments should have an **index that is greater than** the number of variables used (the last element contains 0). You are therefore advised to fill this array initially (using the NCL FILL verb) to ensure that element 0 actually exists, since the end of the scan is determined by this element.

7. If no cursors have been opened, the cursor value must be set to that of the DEFAULT CURSOR: -1.
8. SQL\_EXEC\_LONGSTR replaces SQL\_EXECLONGSTR%. To use this instruction, you will still find the code you need in the notes of NSDBMS.NCL.
9. The other function of SQL\_EXEC\_LONGSTR instruction are the same as SQL\_EXEC.

### Example

```

LOCAL NCLVAR VARLIST[3]      ; for 2 variables
LOCAL SQL_STR$              ; string to pass
LOCAL VAR1%, VAR2$          ; host variables
LOCAL CONDITION%            ; input variable

; ---- Set the array to 0
FILL @VARLIST, SIZEOF VARLIST, 0

SQL_STR$ = "SELECT VCHAR, VINT " & "FROM TAB1 " "WHERE VINT >= :"

VARLIST[0].PTR_VAR = @CONDITION%
VARLIST[0].TYPE_VAR = TYPE_SQL_INT%
VARLIST[0].SIZE_VAR = SIZEOF @CONDITION%

SQL_EXEC_LONGSTR @SQL_STR$, @VARLIST, DEFAULT_CURSOR

FILL @VARLIST, SIZEOF VARLIST, 0
SQL_STR$ = "FETCH INTO :, :"

VARLIST[0].PTR_VAR = @VAR2$
VARLIST[0].TYPE_VAR = TYPE_SQL_CSTRING%
VARLIST[0].SIZE_VAR = SIZEOF VAR2$
VARLIST[1].PTR_VAR = @VAR1%
VARLIST[1].TYPE_VAR = TYPE_SQL_INT%
VARLIST[1].SIZE_VAR = SIZEOF VAR1%

WHILE SQL_ERROR% = 0

    SQL_EXEC_LONGSTR @SQL_STR$, @VARLIST, DEFAULT_CURSOR
    IF SQL_ERROR% = 0
        MESSAGE "SELECT", VAR1% && VAR2$
    ENDIF
ENDWHILE

```

### See also

FILL (NCL), NSDBMS.NCL, SQL\_EXEC, SQL\_EXECSTR, SQL\_ERROR%, SQL\_ERRMSG\$

## SQL\_OPENCURSOR% function

Opens a cursor and returns its handle.

**Syntax**                **SQL\_OPENCURSOR%**

**Returned value**     INT(4)

### Notes

1. After opening the cursor, it can be used with the following instructions:  

```
SQL_EXEC SELECT ... USING handle-cursor
SQL_EXEC FETCH ... USING handle-cursor
```
2. A cursor is an internal resource managed by the Nat System DLL and is used, for example, to store the current table row position for the next SQL call.
3. When the system is opened, only one cursor is defined, known as the DEFAULT\_CURSOR.
4. If no cursors have been opened, this DEFAULT\_CURSOR will be used to execute all SQL statements that maintain current positions within the database, including SELECT and FETCH statements.
5. A problem occurs if an SQL statement other than FETCH (for example UPDATE or INSERT) is embedded in a scanning sequence; the current position is lost and the FETCH statement that follows the embedded statement will terminate with the error.  
SQL\_OPENCURSOR% solves this problem by executing all SELECT and FETCH commands with the new cursor.
6. Generally speaking, a new cursor should be opened each time you wish to perform a SELECT FETCH scan while another similar scan is still in progress with the last cursor opened.
7. The Nat System DLL specifically designed for the DBMS stores cursors in a LIFO (Last In First Out) stack: SQL\_OPENCURSOR% stacks and SQL\_CLOSECURSOR unstacks.
8. The following rules apply when executing a statement with a cursor:  
Statements are always executed with the specified cursor.  
If with SQL\_EXEC, the USING clause isn't specified, the commands are executed with the DEFAULT\_CURSOR.
9. When several databases are opened simultaneously, the cursor opened by SQL\_OPENCURSOR% is immediately associated with the current database.
10. If you want to open a cursor in a database other than the current one, you must execute the SQL\_EXEC CHANGEDBCNTX:otherbase\$ command to change databases before you execute SQL\_OPENCURSOR%.

**Example**

See the example of the SQL\_CLOSECURSOR instruction.

**See also**

SQL\_CLOSECURSOR, SQL\_ERROR%, SQL\_ERRMSG\$

## SQL\_CLOSECURSOR instruction

Closes the last cursor opened and the last occupied by SQL\_OPENCURSOR%.

**Syntax**                    **SQL\_CLOSECURSOR**

**Notes**

1. SQL\_CLOSECURSOR closes the last cursor opened, located at the top of the LIFO (Last In First Out) cursor stack.
2. The error codes returned by SQL\_ERROR% for this instruction are: -32003 or -32005.
3. The SQL\_CLOSECURSOR instruction must not be used with the IM module of NatStar.
4. Nat System recommends you to use SQL\_CLOSETHECURSOR instead of SQL\_CLOSECURSOR.

**Example**

```
; ---- Example showing the two different types of
;         cursors (for clarity, we have not
;         included error test code)

SQL_EXEC ....                ; uses the default cursor
C1% = SQL_OPENCURSOR%        ; opens the C1% cursor
SQL_EXEC UPDATE ...          ; uses the default cursor
SQL_EXEC SELECT ...          ; uses the default cursor
SQL_CLOSETHECURSOR C1%       ; => error
C2% = SQL_OPENTHECURSOR%     ; opens the C2% cursor
SQL_EXEC UPDATE ...          ; uses the default cursor
SQL_EXEC UPDATE ... USING C1% ; uses the C1% cursor
SQL_EXEC SELECT ... USING C2% ; uses the C2% cursor
SQL_EXEC SELECT ... USING C1% ; uses the C1% cursor
SQL_CLOSECURSOR              ; closes the C1% cursor
SQL_EXEC UPDATE ....         ; uses the default cursor
SQL_EXEC SELECT .... USING C2% ; uses the C2% cursor
SQL_CLOSECURSOR%             ; => error
SQL_CLOSETHECURSOR C2%       ; closes the C2% cursor
SQL_EXEC ....                ; uses the default cursor
```

**See also**                    SQL\_OPENCURSOR%, SQL\_ERROR%, SQL\_ERRMSG\$



## SQL\_ERROR% function

Returns the error code of the last SQL\_ instruction executed.

**Syntax**                    **SQL\_ERROR%**

**Returned value**        INT(4)

### Notes

1. SQL\_ERROR% complies with SQL conventions. The function returns:
  - ◆ 0 if no errors occurred.
  - ◆ A positive number for non-fatal errors (the instruction was executed but issued a warning).
  - ◆ A negative number for fatal errors (the instruction could not be executed).
2. A common example of a warning is +100 for:
  - "No row was found or last row reached",
  - "The result of query is an empty table".
3. Fatal errors can be caused by:
  - **Syntax** (parsing error for an SQL statement):  
`-103 : "The numeric literal <Literal> is not valid"`
  - **System** (error encountered on the database while executing the statement):  
`-968 : "The file system is full".`
4. There are two types of errors returned:
  - Proprietary DBMS SQL error codes which are described in the editor's manuals.
  - Internal Nat System error codes. They correspond to errors not handles by the host DBMS. These error messages are numbered and have the format "32XXX".

#### Example :

```
-32004 "NSSQLE004 ** NO MORE CURSORS AVAILABLE"
```

5. List of internal Nat System errors:
  - 32001 NSSQLE001 \*\* HEAP ALLOCATION ERROR**  
**Cause:** Internal memory allocation/deallocation error.
  - 32002 NSSQLE002 \*\* DYNAMIC ALLOCATION ERROR**  
**Cause:** Internal memory allocation/deallocation error.
  - 32003 NSSQLE003 \*\* DYNAMIC FREE STORAGE ERROR**  
**Cause:** Internal memory allocation/deallocation error.
  - 32004 NSSQLE004 \*\* NO MORE CURSORS AVAILABLE**  
**Cause:** Too many cursors opened simultaneously.
  - 32005 NSSQLE005 \*\* NO MORE CURSORS TO FREE**  
**Cause:** Attempt to free a cursor while there are no more open cursors.
  - 32006 NSSQLE006 \*\* INVALID INTO CLAUSE IN FETCH/SELECT**  
**Cause:** Syntax error in the INTO clause of a SELECT or FETCH statement.
  - 32007 NSSQLE007 \*\* TOO MANY VARIABLES IN INTO CLAUSE**  
**Cause:** Too many variables in the INTO clause.

**-32008 NSSQLE008 \*\* MISSING HOST VARIABLE AFTER ','**  
**Cause:** Syntax error in an INTO clause. Variable missing after a continuation comma.

**+32009 NSSQLW009 \*\* INTO CLAUSE: NOT ENOUGH VARIABLES**  
**Cause:** A SELECT statement contains an INTO clause with fewer variables than the number of variables returned by the query.  
**Warning:** The system will still fill the host variables supplied to it.

**+32010 NSSQLW010 \*\* AN OPENED CURSOR WAS CLOSED BY SYSTEM**  
**Cause:** Following the arrival of a new SQL command for a cursor, the system forced the closure of a cursor containing an active query.

**-32011 NSSQLE011 \*\* WHERE/VALUE CLAUSE: NOT ENOUGH VARIABLES**  
**Cause:** Not enough host variables received in the table to be able to substitute the variables specified in the WHERE clause.

**-32012 NSSQLE012 \*\* INVALID INPUT VARIABLE DATA TYPE**  
**Cause:** Invalid data type in a WHERE clause.

**-32013 NSSQLE013 \*\* MISSING 'OF' AFTER 'WHERE CURRENT'**  
**Cause:** Syntax error in UPDATE WHERE CURRENT OF statement.

**-32014 NSSQLE014 \*\* NO OUTPUT VARIABLES DEFINED FOR FETCH**  
**Cause:** The FETCH and the prior SELECT have not defined any output variables (INTO clause).

**-32015 NSSQLE015 \*\* CURSOR NOT READY (MISSING SELECT)**  
**Cause:** FETCH attempted without a prior SELECT or cursor closed by the system between the SELECT and FETCH statements.

**-32016 NSSQLE016 \*\* INVALID SQL DATA TYPE**  
**Cause:** Data type invalid for output.

**-32017 NSSQLE017 \*\* INVALID DATA CONVERSION REQUESTED**  
**Cause:** Type conversion invalid for output.  
 STRING -> NUM  
 NUM -> STRING  
 REAL -> INTEGER  
 INTEGER -> REAL

**-32018 NSSQLE018 \*\* NUMERIC DATA TYPE: INVALID LENGTH**  
**Cause:** Invalid length for the data type (for example, real number with a length of 48).

**-32019 NSSQLE019 \*\* INVALID DECIMAL PACKED FORMAT**  
**Cause:** Unable to convert data to packed decimal format.

**+32020 NSSQLW020 \*\* STRING DATA TRUNCATED**  
**Cause:** The string passed as a variable is shorter than the field received from the DBMS. The received field has been truncated.

**-32021 NSSQLE021 \*\* RESET STORAGE ERROR**  
**Cause:** Error deallocating internal heap.

**+32022 NSSQLW022 \*\* FUNCTION NOT SUPPORTED IN INFORMIX DATABASE**  
**Cause:** The executed instruction is not available.

**-32023 NSSQLE023 \*\* TOO MANY OPENED DATABASES**  
**Cause:** More than 5 databases opened simultaneously.

**+32024 NSSQLW024 \*\* DB ALREADY OPENED**

**Cause:** The database used with SQL\_OPEN has already been opened.

**-32025 NSSQLE025 \*\* DB NOT PREVIOUSLY OPENED**

**Cause:** Attempt to close a database that has not been opened.

**-32026 NSSQLE026 \*\* INVALID DATABASE NAME REF**

**Cause:** An unknown database name has been used.

**-32027 NSSQLE027 \*\* INFORMIX ROUTINE DID NOT RETURN.**

**-32028 NSSQLE028 \*\* INFORMIX OSD-ERROR**

**Cause :** OS dependent error.

**+32030 NSSQLW030 \*\* MEMORY CONSISTENT ERROR**

**Cause:** Error allocating or deallocating memory.

**+100 NSSQLW100 \*\* NO ROW WAS FOUND OR LAST ROW REACHED**

**Cause:** No (more) rows found after a FETCH or SELECT statement.

### Example

```
...  
  
SQL_EXEC SELECT ENO,ENAME INTO :NO%,:NAME$ FROM EMPLOYE  
WHILE SQL_ERROR% = 0  
    INSERT AT END "NO=" & NO% & " NAME=" & NAME$ TO LBOX1  
    SQL_EXEC FETCH INTO :NO%,:NAME$  
ENDWHILE  
IF SQL_ERROR% < 0  
    MESSAGE "fatal error", SQL_ERROR% && SQL_ERRMSG$(SQL_ERROR%)  
ELSE  
    IF SQL_ERROR% > 0  
        MESSAGE "Warning", SQL_ERROR% && SQL_ERRMSG$(SQL_ERROR%)  
    ELSE  
        MESSAGE "OK no error", SQL_ERROR% && SQL_ERRMSG$(SQL_ERROR%)  
    ENDIF  
ENDIF  
...  
...
```

### See also

*NSnn\_SQL Library Error messages*

SQL\_ERRMSG\$

---

## SQL\_ERRMSG\$ function

Returns the error message (character string) for the last SQL\_ instruction executed.

**Syntax**                 **SQL\_ERRMSG\$** (*error-code*)

**Parameter**            *error-code*                     INT(4)     I            error code

**Returned value**     CSTRING

**Notes**

1.    SQL\_ERRMSG\$ returns the last message stored in a work area in the DLL when the error occurred.
2.    This function can be used with all DBMS drivers.

**Example**

Refer to the example of the SQL\_ERROR% fonction.

**See also**                *NSnn\_SQL Library Error messages*  
SQL\_ERROR%

## SQL\_PROC command

Executes a stored procedure.

**Syntax**                **SQL\_PROC** [*SCHEMA*].*procedure\_name* (... , *Param\_J* *Type*, ...) **VALUES** (... ,:*Host\_Var\_J*, ...)

By default, *SCHEMA* stands for the user name of the connection who has created a stored procedure/function.

<b>Parameter</b>	<i>procedure_name</i>	name of the procedure
	<i>Param_J</i>	name of the parameter
	<i>Type</i>	IN or OUT
	<i>Host_Var_J</i>	NCL variable

### Notes

1. SQL\_EXEC followed by SQL\_PROC lets you execute stored procedures.
2. The *Param\_J* parameters are not used by Informix, but it is still possible to specify the input and output parameters. This means you can specify parameter names which are different than the ones declared in the procedure.
3. If no type is specified, the type IN is default.
4. The list after the key word VALUES contains a list of variables preceded by a ":".
5. The variables which correspond to IN parameters let you pass values to the procedure. The variables which correspond to OUT parameters; lets you recuperate values returned from the procedure.
6. To use a procedure which returns several columns(CURSORY PROCEDURE,) you have to specify a cursor in order to execute a FETCH.
7. To pass input values to a procedure, they must be placed in NCL variables then pass them in the VALUES section when calling the procedure. You cannot place values directly in the list of VALUES.

### Example

```
; creation of read_some procedure

SQLEXEC CREATE PROCEDURE READ_SOME (NOM CHAR(15), AGE INT) \
SQLEXEC RETURNING CHAR(15), INT;\
DEFINE F_NAME CHAR(15);\
DEFINE ID INT;\
FOREACH\
  SELECT FIRSTNAME, CUSTOMER_ID INTO F_NAME, ID \
  FROM CUSTOMER WHERE NAME = NOM AND AGE = AGE \
  RETURN F_NAME, ID WITH RESUME;\
END FOREACH;\
END PROCEDURE

; using the procedure
```

```
MYNAME$ = "DAVID"
MYAGE% = 18
CURS% = SQL_OPENCURSOR%
SQL_STR$ = "SQL_PROC READ_SOME (@P1,@P2,@P4 OUT,@P5 OUT)" && \
          "VALUES (:, :, :, :)"
SQL_EXECSTR SQL_STR$, MYNAME$, MYAGE%, FIRSTNAME$, CUST_ID% USING CURS%

; retrieving values in FIRSTNAME$ and CUST_ID%

SQL_EXEC FETCH USING CURS%

; retrieving values in FIRSTNAME$ and CUST_ID%

SQL_EXEC FETCH INTO :A$, B% USING CURS%

; retrieving values in A$ in B%

SQL_CLOSECURSOR
```

**See also**                SQL\_ERROR%, SQL\_ERRMSG%

## **NS\_FUNCTION extensions**

The NS\_FUNCTION extensions have been developed to increase the functionalities of database's interface.

A number of new functions can be accessed via the NCL language :


These commands must be preceded by the keyword NS\_FUNCTION, as follows :

**SQL\_EXEC NS\_FUNCTION** *command*

---

## NS\_FUNCTION CHANGEDBCNTX

Sets the current database.

 This function has been developed to manage several databases simultaneously.

**Syntax**                **NS\_FUNCTION CHANGEDBCNTX** :*logical-DBname*

**Parameter**            *logical-DBname*                CSTRING     I            logical name of the current database

**Notes**

1. The database specified in *logical-DBname* will become the current database.
2. If the specified database is invalid, the current database will not change.
3. If the SQL\_OPENCURSOR command is called after NS\_FUNCTION CHANGEDBCNTX, the new cursor will be associated with the database passed as an argument to this function.

**Example**

```
LOCAL DATABASENAME$  
  
MOVE "PUBS" TO DATABASENAME$  
  
SQL_OPEN "PUBS", "SCOTT/TIGER@SERVER1"  
  
; master current base  
SQL_OPEN "MASTER", "SCOTT/TIGER@SERVER2"  
  
; the current base becomes pubs  
SQL_EXEC NS_FUNCTION CHANGEDBCNTX :DATABASENAME$
```

**See also**                SQL\_OPEN, SQL\_CLOSE



## NS\_FUNCTION DEFAULTCURSOR\_TOP

Allows to differentiate at which cursor is treated a SQL command that not using USING CURSOR% syntax.

**Syntax**                    **NS\_FUNCTION DEFAULTCURSOR\_TOP**

**Note**

1. If USING CURSOR% is not used, the SQL command is treated on the last opened cursor.

**Example**

```
Local STRING ENT_RAIS (35), STRING EMP_ID (4), STRING EMP_LIB (35)
Local STRING ENT_ID (10)
Local curl%, ret%
Local Lib$
;With informix if you do not specify a cursor the statement is executed with
;the last opened cursor
;to override this behaviour and make informix use the default cursor use the
;following NS-FUNCTION DEFAULTCURSOR_DEFAULT
sql_exec NS_FUNCTION DEFAULTCURSOR_DEFAULT
if sql_error%
    message "Error SQL_Exec NS_FUNCTION \
DEFAULTCURSOR_DEFAULT"&&sql_error%,sql_errmsg$(sql_error%)
endif

; Opening the curl% cursor
curl% = SQL_OpenCursor%
if sql_error%
    message "Error Opencursor%"&&sql_error%,sql_errmsg$(sql_error%)
endif

ret% = 0
SQL_exec select ENT_ID, ENT_RAIS from soc using curl%
ret% = SQL_Error%
if ret%
    message "Error select"&&ret%,sql_errmsg$(ret%)
endif

While ret% = 0
    SQL_EXEC FETCH INTO ( :ENT_ID, :ENT_RAIS ) using curl%
    ret% = SQL_Error%
    if ret% = 0
        ; the following statement would produce an end of table if NS_FUNCTION
        ; DEFAULTCURSOR_DEFAULT was not specified
        SQL_Exec SELECT EMP_LIB INTO :Lib$ FROM EMPLOI Where ENT_ID=:ENT_ID
        if sql_error%
            message "Error Update \
emploid"&&sql_error%,sql_errmsg$(sql_error%)
        endif
    endif
EndWhile
```

```
; return to informix default mode the last cursor opened
;(the one on top of the stack) is used when no specific cursor is specified
sql_exec NS_FUNCTION DEFAULTCURSOR_TOP
if sql_error%
    message "Error SQL_Exec NS_FUNCTION \
DEFAULTCURSOR_DEFAULT"&&sql_error%,sql_errmsg$(sql_error%)
endif
```

## NS\_FUNCTION DEFAULTCURSOR\_DEFAULT

Allows to differentiate on which cursor is treated a SQL command that not using USING CURSOR% syntax. Specify the cursor as the one located on the top of the stack.

**Syntax**                    **NS\_FUNCTION DEFAULTCURSOR\_DEFAULT**

### Note

1. If USING CURSOR% is not used, the SQL command is treated by the default cursor.

### Example

```
Local STRING ENT_RAIS (35), STRING EMP_ID (4), STRING EMP_LIB (35)
Local STRING ENT_ID (10)
Local curl%, ret%
Local Lib$
;With informix if you do not specify a cursor the statement is executed with
;the last opened cursor
;to override this behaviour and make informix use the default cursor use the
;following NS-FUNCTION
;DEFAULTCURSOR_DEFAULT
sql_exec NS_FUNCTION DEFAULTCURSOR_DEFAULT
if sql_error%
    message "Error SQL_Exec NS_FUNCTION \
DEFAULTCURSOR_DEFAULT"&&sql_error%,sql_errmsg$(sql_error%)
endif

; opening the cursor curl%
curl% = SQL_OpenCursor%
if sql_error%
    message "Error Opencursor"&&sql_error%,sql_errmsg$(sql_error%)
endif

ret% = 0
SQL_exec select ENT_ID, ENT_RAIS from soc using curl%
ret% = SQL_Error%
if ret%
    message "Error select"&&ret%,sql_errmsg$(ret%)
endif

While ret% = 0
    SQL_EXEC FETCH INTO ( :ENT_ID, :ENT_RAIS ) using curl%
    ret% = SQL_Error%
    if ret% = 0
        ;the following statement would produce an end of table if NS_FUNCTION
        ;DEFAULTCURSOR_DEFAULT was not specified
        SQL_Exec SELECT EMP_LIB INTO :Lib$ FROM EMPLOI Where ENT_ID=:ENT_ID
        if sql_error%
            message "Error Update
emploie"&&sql_error%,sql_errmsg$(sql_error%)
        endif
    endif
endif
```

```
EndWhile
; return to informix default mode the last cursor opened
;(the one on top of the stack) is used when no specific cursor is specified
sql_exec NS_FUNCTION DEFAULTCURSOR_TOP
if sql_error%
    message "Error SQL_Exec NS_FUNCTION
DEFAULTCURSOR_DEFAULT"&&sql_error%,sql_errmsg$(sql_error%)
endif
```