NatStar

Version 5.00 Edition 1

NS-DK

Version 5.00 Edition 1

NatWeb

Version 4.00 Edition 1

Sybase

Ref. n° NS500SYBU00033

Contents

About this Manual

This is the Sybase manual for Nat System‘s development tools. This manual describes the Sybase interface allowing the access to a Sybase database.

Supported configurations

Development environment

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

Client environment

Operating system	DBMS drivers available
Windows 32 bits	Sybase 11, 12
Windows NT4, 95, 98 32 bits	Sybase 12.5
Windows 2000, XP 32 bits	Sybase 12.5, 12.5.3

Server environment

Operating system	DBMS drivers available
Windows NT 4, 2000, 2003 32 bits	Sybase 11, 11XA Sybase 12, 12XA Sybase 12.5, 12.5.3
AIX 4.1 32 bits	Sybase 11
AIX 4.3 32 bits	Sybase 11
AIX 4.3.3 32 bits	Sybase 12.5, 12.5.3
AIX 5.2 32 bits	Sybase 12.5, 12.5.3
HP-UX 10.x 32 bits	Sybase 11
HP-UX 11.x (Risc 2) 32 bits	Sybase 11 Sybase 12.5, 12.5.3

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.

Organization of the manual

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

Chapter 1	**Sybase System CTLIB interface** Describes the functions of the NSnnSxx library associated with the Sybase database.

Conventions

Typographic conventions

Important term	Important terms are printed in **bold**.
Interface component	The names of windows, dialog boxes, controls, buttons, menus and options are printed in *italics*.
[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)
32	Indicates specific information on using the software under DOS- Windows 32 bits
32 bits	Indicates specific information on using the software under OS/2- PM 2.x or DOS- Windows 32 bits
UNIX	Indicates specific information on using the software under Unix systems

Chapter 1 Sybase System CTLIB interface

The NSnnSxx library allows your applications built with Nat System development tools to interface with client versions of Sybase System CTLIB.

⚠ Nat System supports in a Windows 32 bit environment: Sybase 11, Sybase 12 and 12.53.

⚠ To simplify documentation NSnnS11, NSnnS12 and NSnnS125 will be grouped under the generic name of NSnnSxx.

This chapter explains

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

Contents

Introduction

The NSnnSxx libraries allow your applications built with Nat System development tools to interface with client versions of Sybase. You can access this library via NCL using a number of instructions and functions prefixed by SQL_

To simplify documentation NSnnS11 and NSnnS12 will be grouped under the generic name of NSnnSxx.

nn stands for the version number of the interface that you have installed.

The NSnnSxx library has the following new features, such as:

- remote procedure calls
- the COMPUTE function
- etc.

The use of different cursor modes lets you use the command language, cursor commands and dynamic commands.

They also support the RECORD and REEXECUTE commands which combine the adaptability of dynamic SQL with the speed of static SQL. In fact, a dynamic SQL order is used when the RECORD command is executed. When the REEXECUTE command is executed, as the analysis of query has already been done by the motor, only the values of the host variables are set.

NSnnSxx uses the CTLIB version of the Sybase System libraries. You may notice a number of functional differences between earlier versions (NSnnS42) and NSnnSxx.

Correspondence between drivers and the versions of Sybase

The name of the driver should be used in particular with the instruction THINGS_DB_INIT for NatStar and the instruction SQL_INIT for NatStar, NatWeb and NS-DK

The following table indicates the versions of Sybase and the corresponding drivers.

Version of Sybase	Driver
Sybase 11	NSnnS11.dll
Sybase 11 XA	NSnnS11.dll
Sybase 12	NSnnS12.dll
Sybase 12 XA	NSnnS12.dll
Sybase 12.5.3	NSnnS125.dll
Sybase 12.5.3 XA	NSnnS125.dll

nn stands for the number of the version of the interface you have installed:

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

Mode XA

The XA mode is a protocol which allows to synchronize updates on two different databases keeping coherence between them. The principle is to carry out a *pre-commit* on each of database and deffer the global *commit* in case the two *pre-commit* succedeed.

To simplify documentation NSnnS11, NSnnS12 and NSnnS125 will be grouped under the generic name of NSnnSxx.

Previous versions of the drivers no longer supported by Nat System in the Windows 32 bit environment are installed in the CONTRIB(S) folder: NSw2S42.dll and NSw2S10.dll.

Installation

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 Sybase vendor.

Implicit Output Data Conversions

The NSnnSxx libraries use conversion functions of Sybase.

Refer to the User Manual of Sybase to see the allowing conversions.

Nevertheless, for certain data, use the following conversions:

Sybase	NCL
DECIMAL, NUMERIC	CSTRING (*), CHAR, INT, NUM
BINARY,VARBINARY	BLOBS, SEGMENT
TEXT, IMAGE	BLOBS, SEGMENT

(*) it is strongly recommended to use this type of data.

blob	A **blob** (or Binary Large Object) corresponds to a field of a database able to contain images, video and sound for multimedia applications.

Environment variables

You can declare environment variables to change the functions of the driver.

In this version, only one environment variable is available.

Syntax

CRYPT = ON or **OFF**

Description

- CRYPT = ON is the default value,
- CRYPT = OFF cancels the encryption of the character string used during SQL_OPEN

For more information about environment variables and their syntax, see your operating system's documentation.

Example

```
; Example on Windows on a command line of a DOS box

set CRYPT=ON
```

Functional categories in the NSnnSxx library

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

Initializing and stopping application use of the DBMS

Opening and closing a database

Managing the current database

Choose the DBMS and the base for an SQL command

Executing a SQL command: SELECT, INSERT, UPDATE, CREATE TABLE …

Managing the cursor

Running a stored procedure

Stored procedures are the best means of accessing a Sybase database. They avoid the compilation phase of a query which takes a long time on this DBMS and using them gives good performance.

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

Recording and reexecuting an SQL command

Managing blobs

Managing buffers

Managing TCP/IP packets

Configuring DBMS behavior

SQL query advanced management

Handling errors

Sybase commands

NSnnSxx library reference

SQL_INIT instruction

Loads the driver needed to use Sybase System CTLIB.

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 Sybase System CTLIB 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 Sybase System CTLIB database: NSnnSxx.
   - **nn** indicates the target platform: w2 for Windows 32 bits, w4 for Windows 64 bits.
   - **xx** indicates the installed version of Sybase.

Example

```
SQL_INIT "NS02S11"  ; loads the DLL working with the client part of
                    ; Sybase 11 on a Windows 32 bits environment
SQL_STOP            ; unloads the DLL
```

See also SQL_STOP, SQL_INITMULTIPLE%, SQL_STOPMULTIPLE, SQL_STOPALL, SQL_ERROR%, SQL_ERRMSG$

SQL_STOP Instruction

Unloads the current DBMS and closes simultaneously all opened databases and cursors.

Syntax **SQL_STOP**

Note

1. It is compuilsory that the application ends with SQL_STOP or an equivalent instruction.

Example

Refer to the example of SQL_INIT instruction.

SQL_OPEN instruction

Opens a database.

Syntax **SQL_OPEN** *logical-DBname, password-string*

Parameters

logical-DBname	CSTRING	I	logical name of the database to open
password-string	CSTRING	I	password to connect to a server

Notes

1. Sybase System CTLIB could open several databases through the network.
2. The *logical-DBname* specified the logical database name. There are two possible cases :

   - The logical database name matches its physical name: the specified database will be the current database.

   - The logical database name is different from its physical name: the default database will be the current database.

   In addition, *logical-DBname* can also contain an option **nocursor**.

   Example :
   ```
   SQL_OPEN  "pubs2/nocursor", "sybuser/passwd@SYB10"
   ```

   The parameter **nocursor** leads to the opening of a physical connection to simulate a cursor.
3. The *password-string* parameter specifies the command string used to connect to a local or remote database, as follows :
   ```
   "USERID[/PASSWORD][@<CONNECTOR>]
   [#LOGINTIMEOUT]
   [!PACKETSIZE][;HOST=hostname][;APP=appname]
   [$OFF]"
   ```

   where :

`[USERID]`	name of the user account on the server
`[PASSWORD]`	password for the user account
`[@<CONNECTOR>]`	used to specify which open database will receive the query.

	If it is not specified, the local database will be accessed.
`[#LOGINTIMEOUT]`	specifies the connection TIMEOUT in seconds. If it is not specified, the TIMEOUT is set to 120 seconds.
`[!PACKETSIZE]`	specifies the size of the packets sent to the network. If it is not specified, the size of the packets is specified during the configuration of Sybase System.
`[;HOST=hostname]`	specifies the name of the machine. This name will be visible with SP_WHO. The name of the applicative does not contain the following characters: space, !, #, @ or /. Do not use spaces in hostname, use the underscore '_' to act like a space

```
SQL_OPEN ""db"", ""sybuser/passwd@dbhost;HOST=hostname""
```

`[;APP=appname]`	specifies the name of the application. This name will be visible with SP_WHO. The name of the applicative does not contain the following characters: space, !, #, @ or /. Do not use spaces in appname, use the underscore '_' to act like a space

```
SQL_OPEN ""db"", ""sybuser/passwd@dbhost;APP=appname""
```

`[$OFF]`	cancel the encoding of the password string. By default, each parameter of the SQL_OPEN is encoded.

4. If a password is not required to open the database, pass an empty string in password-string and enter only the name of the user in the `[USERID]` server.
5. If the database that you want to open, is not the user's default database, SQL_OPEN must be followed by SQL_EXEC USE, which should indicate the name of the database that you want to use (Cf. *SYBASE System Manual*).
6. If the connection parameter [@<CONNECTOR>] is not filled, the driver will use the environment variable DSQUERY (Cf. *OpenClient Installation Manual*).
7. The configuration of the connection is realized by Sybase System CTLIB with :
   - SQLEDIT, DSEDIT on PC,
   - SYBINIT on UNIX.

Example

```
SQL_OPEN "BASE1", "SYBUSER/PASSWD"
IF SQL_ERROR% <> 0
   MESSAGE "Error on Base1", SQL_ERRMSG$(SQL_ERROR%)
```

```
ENDIF
SQL_OPEN "BASE2", "SYBUSER/PASSWD@SERV2"
IF SQL_ERROR% <> 0
   MESSAGE " Error on Base 2", SQL_ERRMSG$(SQL_ERROR%)
ENDIF
...
SQL_CLOSE "BASE2"
SQL_CLOSE "BASE1"
```

See also SQL_CLOSE, NS_FUNCTION CHANGEDBCNTX, AT, SQL_ERROR%, SQL_ERRMSG$, NS_FUNCTION GETDBNAME,

SQL_CLOSE instruction

Closes a connection of a database.

Syntax	**SQL_CLOSE** *logical-DBname*			
Parameter	*logical-DBname*	CSTRING	I	logical name of the database to close

Notes

1. Although that you recommend that you close each of the databases opened by an application, an SQL_CLOSE instruction is automatically generated for each open database when an application is closed.

Example

Refer to the example of the SQL_OPEN instruction.

See also

SQL_OPEN, NS_FUNCTION CHANGEDBCNTX, AT, SQL_ERROR%, SQL_ERRMSG$, NS_FUNCTION GETDBNAME

AT command

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

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

Parameters

logical-DBname	CSTRING	I	logical database name
SQL-statement	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 go from one database to another, we suggest using the NS_FUNCTION CHANGEDBCNTX command because the AT command may no longer be supported in future releases.

Example

```
SQL_OPEN "BASE1", "SYBUSER/PASSWD@SERVER1"
SQL_OPEN "BASE2", "SYBUSER/PASSWD@SERVER2"
SQL_OPEN "BASE3", "SYBUSER/PASSWD@SERVER3"
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, NS_FUNCTION CHANGEDBCNTX, SQL_ERROR%, SQL_ERRMSG$, NS_FUNCTION GETDBNAME

SQL_EXEC instruction

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

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

Parameters

logical-DBname	CSTRING	I	logical name of database
SQL-command	CSTRING	I	SQL command to execute
cursor-handle	INT(4)	I	cursor value

Notes

1. The SQL command is passed directly without quotes. It can correspond to any Sybase SQL command , whether it's a data definition command (CREATE TABLE, CREATE INDEX, ...) or a data manipulation command (SELECT, INSERT, UPDATE, ...).
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. In *nocursor* mode, the optional parameter **USING** *cursor-handle* is not available.
5. The SQL command can return values in NCL variables. For this, just pass these variables in parameters.
6. It is possible to pass a segment's field as a data-receiving variable in an SQL query.
7.
8. SQL_EXEC, SQL_EXECSTR and SQL_EXEC_LONGSTR depend on SQL language accepted by Sybase (Cf. *Sybase manuals*).
9. For SQL commands that are too long, it is possible to use the special continuation character "\":

```
SQL_EXEC UPDATE SAMPLE SET COMPANY =:A$ \
                        WHERE CITY =:C$ \ AND \
                              COUNTRY =:D$
```

10. The types of variables recognized by the interface are:
    - INT(1), INT(2), and INT(4),
    - NUM(8), NUM(4),
    - STRING,
    - CSTRING,

- CHAR.

11. Refer to the chapter *Implicit output data conversion* for more information about the conversion of NCL types to authorized SQL types.

12. 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]
[, ... ] ]
```

13. 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 a table.

14. Always put a ":" before the name of a variable or flag.

15. 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.

16. 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 ("").

17. All hosts variables hosts for an insert must be initialized even if you want to insert a null in the associated column.

Example

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

CITY$ = "PARIS"

; ========
;  1st CASE
; ========
; ---- Selection of the subset
SQL_EXEC SELECT CODE,COUNTRY \
         FROM WORLD \
         WHERE CITY = :CITY$
; ---- Reading from the 1st to the last row of the selection
WHILE SQL_ERROR% = 0
  SQL_EXEC FETCH INTO :CODE%,:COUNTRY$
  IF SQL_ERROR% = 0
     INSERT AT END CODE% && COUNTRY$ TO LBOX1
  ENDIF
ENDWHILE

; =============================
;  2nd CASE (the most performant)
; =============================
; ---- Selection of the subset
;      and reading of the 1st row of the selection
SQL_EXEC SELECT CODE,COUNTRY \
         FROM WORLD \
         INTO :CODE%,:COUNTRY$ \
         WHERE CITY = :CITY$
; ---- reading of the 2nd to the lat row of the selection
WHILE SQL_ERROR% = 0
  INSERT AT END CODE% && COUNTRY$ TO LBOX1
  SQL_EXEC FETCH
ENDWHILE

; =========
;  3rd CASE
; =========
; ---- Selection of the subset
SQL_EXEC SELECT CODE,COUNTRY \
         FROM WORLD \
         WHERE CITY = :CITY$
; ---- reading from the 1st to the last row of the selection by filling the two
; tables TCODE% and TCOUNTRY$
I% = 0
WHILE (SQL_ERROR% = 0) AND (I% < 10)
  SQL_EXEC FETCH INTO :TCODE%[I%],:TCOUNTRY$[I%]
  I% = I% + 1
ENDWHILE

; ============================
;  Utilisation of indicators
; ============================
; Creation
SQL_EXEC drop table FAMILY
if sql_error% <> 0
  message 'error drop' , sql_errmsg$(sql_error%)
endif
SQL_EXEC create table FAMILY ( ID INT, LONGSTR  VARCHAR(200) NULL, COLIMAGE
IMAGE NULL, REEL1 REAL NULL, \
 ASTEXT TEXT NULL, SON CHAR(10) NULL, FATHER CHAR(10) NULL, NUM Numeric NULL)
IF sql_error% <> 0
  message 'error create' , sql_errmsg$(sql_error%)
ENDIF

;Insertion
LOCAL FATHER$, AGE% ,SON$, IND1%, IND2%
;All the variables have to be initialized even if their value will be null
SON$= ""
FATHER$ = "PIERRE"
AGE% = 1
IND1% = 0
IND2% = NULL_VALUE_INDICATOR

SQL_EXEC INSERT INTO FAMILLE (ID, FATHER, SON) VALUES (:AGE%,:FATHER$:IND1%, \
:SON$:IND2%)
IF SQL_ERROR% <> 0
    MESSAGE "Error ",SQL_ERRMSG$(SQL_ERROR%)
ENDIF
;Select
LOCAL FATHER$, AGE% ,SON$, IND1%, IND2%

SQL_EXEC SELECT ID, FATHER, SON INTO :AGE%, :FATHER$:IND1%,:SON$:IND2% \
FROM FAMILY
IF SQL_ERROR% <> 0
    MESSAGE "Error ",SQL_ERRMSG$(SQL_ERROR%)
 ENDIF
```

```
WHILE SQL_ERROR% = 0
 IF IND2% = NULL_VALUE_INDICATOR
  INSERT AT END FATHER$ & " has no sons" TO LISTBOX1
 ELSE
  INSERT AT END "The son of " & FATHER$ &"is named" & SON$ TO LISTBOX1
 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, CREATE TABLE …

Syntax **SQL_EXECSTR** *SQL-command* [, *variable* [, *variable* [, ...]]] [**USING** *handle-name*]

Parameters

SQL-command	CSTRING	I	SQL order to execute
variable		I	NCL variable list
handle-name	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%

; ---- Selection of persons older than 20 years
;      in the FAMILY table
SQL_EXECSTR REQ$, FATHER$, IND1%, AGE%, SON$, IND2%, AGE% USING CURS1%

WHILE SQL_ERROR% = 0
  IF IND2% = -1
     INSERT AT END FATHER$ & " has non son" TO LBOX
  ELSE
     INSERT AT END "The son of " & FATHER$ & "is named" & SON$ TO LBOX
  ENDIF
  SQL_EXEC FETCH USING CURS1%
ENDWHILE
```

```
; ---- Closing 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 very long SQL statement : SELECT, INSERT, UPDATE, CREATE TABLE …

Syntax **SQL_EXEC_LONGSTR** *sql-string-address*, *var-array-address*, *cursor-num*

Parameters

sql-string-address	INT(4)	I	address of the character string containing the SQL statement to execute
var-array-address	INT(4)	I	address of the array containing the host variables (or indicators)
cursor-num	INT(2)	I	cursor value

Notes

1. The executed statement can contain any SQL command in the host language (DML or DDL). The string size depends on Sybase.
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
CONST TYPE_SQL_SEGMENT%     10
CONST TYPE_SQL_IMAGE%       11
CONST TYPE_SQL_SELECT_BLOB% 12
CONST TYPE_SQL_INSERT_BLOB% 13
```

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 functions 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$          ; receiving variables
LOCAL CONDITION%            ; in variable
; ---- reset the table
FILL @VARLIST, SIZEOF VARLIST, 0
SQL_STR$ = "SELECT FATHER, ID " & "FROM BASE " & "WHERE ID >= :"
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$, @VARLIST, DEFAULT_CURSOR
 IF SQL_ERROR% = 0
  INSERT AT END  "SELECT"&& VAR1% && VAR2$ TO LISTBOX1
 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%**
Return value	INT(4)

Notes

1. After opening the cursor, it can be used with the following instructions:
```
    SQL_EXEC SELECT ... USING cursor-handle
    SQL_EXEC FETCH  ... USING cursor-handle
```
2. A cursor is an internal resource managed by the NSnnSxx 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 NSnnSxx 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%.
11. If the database has been opened with nocursor option, each SQl_OPENCURSOR% create a physical connection with the parameters of the SQl_OPEN attached.

Example

Refer to the example of the SQL_CLOSETHECURSOR instruction.

See also SQL_CLOSECURSOR, SQL_OPENTHECURSOR%, SQL_CLOSETHECURSOR, 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, situated at the top of the LIFO (Last In First Out) cursor stack.
2. SQL_CLOSECURSOR must only close cursors opened with SQL_OPENCURSOR%.
3. The error codes returned by SQL_ERROR% when this instruction is executed could be -32003 ou -32005.
4. The SQL_CLOSECURSOR instruction must not be used with the IM module of NatStar.
5. Nat System recommends you to use SQL_CLOSETHECURSOR instead of SQL_CLOSECURSOR.

Example

Refer to the example of the SQL_CLOSETHECURSOR instruction.

See also SQL_OPENCURSOR%, SQL_OPENTHECURSOR%, SQL_CLOSETHECURSOR, SQL_ERROR%, SQL_ERRMSG$

SQL_OPENTHECURSOR% function

Opens a cursor and returns its handle.

☞ Prefer this function to SQL_OPENCURSOR%, because it offers a better resources management.

Syntax **SQL_OPENTHECURSOR%**

Return value INT(2)

Notes

1. After opening the cursor, it can be used with the following instructions:
   ```
   SQL_EXEC SELECT ... USING cursor-handle
   SQL_EXEC FETCH ... USING cursor-handle
   ```
2. A cursor is an internal resource managed by the NSnnSxx 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 an 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 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.
8. When opening several databases at the same time, the cursor opened by SQL_OPENTHECURSOR% is immediately associated with the current database.
9. 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

Refer to the example of the SQL_CLOSETHECURSOR instruction.

See also SQL_OPENCURSOR%, SQL_CLOSECURSOR, SQL_CLOSETHECURSOR, SQL_ERROR%, SQL_ERRMSG$

SQL_CLOSETHECURSOR instruction

Closes the cursor associated with the given handle.

Syntax **SQL_CLOSETHECURSOR** *cursor-handle*

Parameter	*cursor-handle*	INT(4)	I	handle of the cursor to close

Note

1. SQL_CLOSETHECURSOR can only close cursors opened with SQL_OPENTHECURSOR%.

Example

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

See also SQL_OPENCURSOR%, SQL_CLOSECURSOR, SQL_OPENTHECURSOR%, SQL_ERROR%, SQL_ERRMSG$

SQL_ERROR% function

Returns the error code of the last SQL_ instruction executed.

Syntax **SQL_ERROR%**

Return 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. This function can be used with all DBMS drivers
3. There are two types of errors returned::
   - Proprietary Sybase SQL error codes which are described in the Sybase's manuals.
   - Internal Nat System error codes. They correspond to errors not handles by Sybase. These error messages are numbered and have the format "32XXX".

     Example :
     ```
     -32004 "NSSQLE004 ** NO MORE CURSORS AVAILABLE"
     ```
4. Sybase System has no direct primtive that supplies the message corresponding to an error number. As a result, these messages are handled directly by the library.
5. List of codes and messages of internal errors:

   0 "NSSQLE000 ** UNUSED NATSYS ERROR CODE"

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

   +200 "NSSQLW200 ** COMPUTE RESULT IN PROGRESS"

   Cause : The SELECT command containing the COMPUTE command is in execution.

   -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 OR TRYING TO DEALLOCATE ONLY CURSOR"

Cause : Attempt to free a cursor whereas there is no cursor anymore.

-32006 "NSSQLE006 ** INVALID INTO CLAUSE in FETCH/SELECT"

Cause : Syntax error in the INTO clause of a SELECT or a 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 the 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.

-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 the 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"

+32022 "NSSQLW022 ** FUNCTION NOT SUPPORTED IN SYBASE DATABASE"

Cause : the executed instruction is not available.

-32023 "NSSQLE023 ** TOO MANY OPENED DATABASES"

Cause : Too many 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 happened.

-32026 "NSSQLE026 ** INVALID DATABASE NAME REF"

Cause : Unknown database name used in the AT clause of the SQL_EXEC statement (database not opened)

-32028 "NSSQLE028 ** UNABLE TO GET SYBASE LOGIN"

Cause : Failed to connect to Sybase (e.g. server name error).

-32029 "NSSQLE029 ** SYBASE VARIABLE INPUT BIND FAILED"

Cause : Type mismatch between a variable and a database field.

-32030 "NSSQLE030 ** SYBASE VARIABLE OUTPUT BIND FAILED"

Cause : Type mismatch between a variable and a database field.

-32031 "NSSQLE031 ** SYBASE BUFFER FILL ERROR"

Cause : Buffer overflow (due to data conversion,etc.)

-32032 "NSSQLE032 ** RPC PARAMETER NAME EXPECTED"

Cause : Remote procedure call : parameters missing

-32033 "NSSQLE033 ** TOO MANY RPC PARAMETERS"

Cause : Remote procedure call : too many parameters specified

-32034 "NSSQLE034 ** RPC PROCEDURE NAME EXPECTED"

Cause : Remote procedure call : procedure name missing

-32035 "NSSQLE035 ** NOT ENOUGH PARAMETERS FOR RPC CALLS"

Cause : Remote procedure call : parameters missing

-32036 "NSSQLE036 ** INVALID RPC PARAMETERS SUPPLIED"

Cause : Remote procedure call : invalid parameters specified

-32037 "NSSQLE037 ** INVALID RPC PROCEDURE INITIALIZATION"

-32038 "NSSQLE038 ** RPC PROCEDURE EXECUTION FAILED"

-32039 "NSSQLE039 ** MEMORY CONSISTENT ERROR"

-32040 "NSSQLE040 ** INVALID TYPE FOR INDICATOR"

-32041 "NSSQLE041 ** CONTEXT NOT CREATED"

-32042 "NSSQLE042 ** CONTEXT NOT FOUND"

-32044 "NSSQLE044 ** NO SET LOGIN TIME"

-32045 "NSSQLE045 ** NO SET TIME"

-32046 "NSSQLE046 ** SET MAXPROCS FAILED"

-32047 "NSSQLE047 ** DB OPEN FAILED"

-32048 "NSSQLE048 ** DB NOT OPENED"

-32049 "NSSQLE049 ** LOGIN RECORD NOT ALLOCATED"

-32050 "NSSQLE050 ** MEMORY DEALLOCATION ERROR"

-32051 "NSSQLE051 ** CURSOR NOT FOUND"

-32052 "NSSQLE052 ** MUST EXECUTE SELECT BEFORE THE FETCH COMMAND"

-32053 "NSSQLE053 ** ERROR IN CLOSING DATABASE"

-32054 "NSSQLE054 ** ERROR IN EXECUTING SQL STATEMENT"

-32055 "NSSQLE055 ** ERROR IN EXECUTING FETCH COMMAND"

-32056 "NSSQLE056 ** INDICATOR'SIZE TOO SMALL TO HOLD VALUE"

-32057 "NSSQLE057 ** UNKNOWN NCL VARIABLE TYPE PASSED"

-32058 "NSSQLE058 ** RPC : INIT (Sybase) ERROR"

-32059 "NSSQLE059 ** RPC : PARAMETERS FOUND BUT NO VALUE CLAUSE"

-32060 "NSSQLE060 ** RPC : PARAMETER TYPE MISMATCH"

-32061 "NSSQLE061 ** RPC : PROCEDURE NAME MISSING"

-32062 "NSSQLE062 ** RPC : INDICATORS MAY ONLY BE ON OUT VARIABLES"

-32063 "NSSQLE063 ** RPC : SQL SERVER ERROR DURING RPC PREPARATION"

-32064 "NSSQLE064 ** RPC : SQL SERVER ERROR DURING RPC EXECUTION"

-32065 "NSSQLE065 ** RPC : SQL SERVER ERROR DURING RPC EXEC CHECK"

-32066 "NSSQLE066 ** RPC : PROCEDURE NOT PREPARED"

-32067 "NSSQLE067 ** LOGGER : CAN'T OPEN FILE"

-32068 "NSSQLE068 ** PARSER : TOKEN TABLE FULL"

-32069 "NSSQLE069 ** SYBEXEC : INCOMPATIBLE CURSOR MODE"

-32070 "NSSQLE070 ** SYBEXEC : SQL SERVER ERROR DURING SIZE BUFFERING EXECUTION"

-32071 "NSSQLE071 ** SYBEXEC : SQL SERVER ERROR DURING SIZE BUFFERING DELETION"

-32072 "NSSQLE072 ** SYBEXEC : INVALID CURSOR MODE"

-32073 "NSSQLE073 ** SYBEXEC : THAT ROW IS NOT IN BUFFER"

-32074 "NSSQLE074 ** SYBEXEC : INCORRECT SYNTAX FOR THIS CURSOR MODE"

-32075 "NSSQLE075 ** SYBEXEC : MISSING INTO CLAUSE FOR THIS CURSOR MODE"

-32076 "NSSQLE076 ** SYBEXEC : INVALID SIZE FOR ROW BUFFERING"

-32077 "NSSQLE077 ** SYBEXEC : INVALID ROW NUMBER"

-32078 "NSSQLE078 ** SYBEXEC : MEMORY DEALLOCATION ERROR FOR SCROLL STATUS"

-32079 "NSSQLE079 ** SYBEXEC : SQL SERVER : ROW IS MISSING IN SCROLL BUFFER"

-32080 "NSSQLE080 ** NO STATEMENT IN PROGRESS"

-32081 "NSSQLE081 ** DATA NOT READY TO RESULT PROCESSING"

-32082 "NSSQLE082 ** INVALID WINDOW HANDLE"

-32083 "NSSQLE083 ** USER MESSAGE MUST BE RANGE IN 0 AND 15"

-32084 "NSSQLE084 ** INVALID STATEMENT SEND TO DLL"

-32085 "NSSQLE085 ** NO MORE RESULT TO FETCH"

-32086 "NSSQLE086 ** INVALID PARAMETER TO CHANGE OPTION"

-32087 "NSSQLE087 ** INVALID PARAMETER TO CHANGE OPTION VALUE"

-32088 "NSSQLE088 ** LOGIN TIME CHANGE FAILED"

-32089 "NSSQLE089 ** TIMEOUT CHANGE FAILED"

-32090 "NSSQLE090 ** INVALID NS_FUNCTION STATEMENT"

-32091 " NSSQLE091 ** INVALID DATABASE NAME"

-32092 "NSSQLE092 ** INVALID INTO CLAUSE WHEN ASYNCHRONOUS MODE"

-32093 "NSSQLE093 ** INVALID LENGTH FOR DATABASE NAME "

-32095 "NSSQLE095 ** INVALID LENGTH FOR USER NAME "

-32096 "NSSQLE096 ** INVALID LENGTH FOR PASSWORD "

-32097 "NSSQLE097 ** INVALID LENGTH FOR SERVER NAME "

-32098 "NSSQLE098 ** INVALID LENGTH FOR SERVER NAME "

-32099 "NSSQLE099 ** INVALID STATEMENT IN XA MODE "

-32101 "NSSQLE0101 ** RECORD STATEMENT IS INCOMPATIBLE WITH RPC "

-32102 "NSSQLE0102 ** REEXECUTE STATEMENT IS INCOMPATIBLE WITH RPC "

-32103 "NSSQLE0103 ** INVALID LENGTH FOR LOGIN TIME "

-32104 "NSSQLE0104 ** LOGIN TIME MUST BE DIGIT VALUE "

-32105 "NSSQLE0105 ** INVALID DATA TYPE FOR SECOND PARAMETER OF CHANGEOPTION."

-32106 "NSSQLE0106 ** INVALID LENGTH FOR TDS PACKET SIZE"

-32107 "NSSQLE0107 ** TDS PACKET SIZE TIME MUST BE DIGIT VALUE"

-32108 "NSSQLE0108 ** UNABLE TO OPEN FILE "

-32109 "NSSQLE0109 ** NO MEMORY AVAILABLE "

-32110 "NSSQLE0110 ** NO CONNECTION AVAILABLE TO UPDATE IMAGE/TEXT "

-32111 "NSSQLE0111 ** INVALID LENGTH FOR APPLICATION NAME "

-32112 "NSSQLE0112 ** INVALID LENGTH FOR CRYPT. FLAG "

-32113 "NSSQLE0113 ** INVALID LENGTH FOR HOSTNAME FLAG "

-32114 "NSSQLE0114 ** INVALID LENGTH FOR APPLICATION NAME FLAG "

-32115 "NSSQLE0115 ** UNABLE TO GET HOSTNAME OR APPLICATION NAME STRING"

-32116 "NSSQLE0116 ** FUNCTION NOT SUPPORTED IN THIS VERSION "

Example

```
SQL_EXEC SELECT ENO,ENAME INTO :NO%,:NAME$ FROM EMPLOYEE
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 **Error messages in the chapter *NSnn_SQL library*.**

SQL_ERRMSG$, NS_FUNCTION ERRORCOUNT, NS_FUNCTION GETERROR, NS_FUNCTION CALLBACK

SQL_ERRMSG$ function

Returns the error message (character string) for the last SQL_ instruction executed.

Syntax	**SQL_ERRMSG$** *(error-code)*			
Parameters	*error-code*	INT(4)	I	error code
Return value	CSTRING			

Notes

1. SQL_ERRMSG$ returns the last message stored in a work area in the NSnnSxx DLL when the error occurred.
2. This function can be used with all Sybase drivers.
3. Warning : in some cases, NSnnSxx handles errors and warnings differently from earlier systems. Many warnings may now be treated as errors.
4. Refer to a SQL_ERROR% function for a detailed list of error messages and codes.

Example

See the example of the SQL_ERROR command.

See also Error messages of the chapter *NSnn_SQL library*

SQL_PROC command

Executes a stored procedure.

Syntax

SQL_PROC *procedure-name*

[(@Parameter-name1 [OUT] [,@Parameter-name2 [OUT]...])

VALUES (:Host-Variable1 [,:Host-Variable2...])]

Parameter	*procedure-name*	CSTRING I	name of the procedure to call

Notes

1. The SQL_PROC keyword informs the parser that a procedure is being called. The word that follows SQL_PROC specifies the name of the SQL_PROC.
2. If the called procedure has parameters, their names must be specified exactly as they are defined in the procedure.

Parameters	**Syntax**
Input parameter	*Parameter-name* in the parameter list. *Parameter-name* can be followed by IN, which is the default in any case.
Output or input/output parameters	*Parameter-name* must be followed by OUT.
Values of the procedure's input parameters	After the VALUES keywords, supply a listof host variables containing the values of the parameters.

3. **Reminder** : host variables are preceded by a colon ':'.
4. **Important** :

   - The maximum size allowed for ALPHA host variables (CHAR, VARCHAR, etc...) is 255 characters.

   - The size of the host variable must match the size of the parameter passed to the procedure. Therefore, when you declare a string host variable, you must also specify its size : `LOCAL A$(15)`.

- If the size of the character string retrieved in the host variable is less than the reserved size, any space that is not occupied by the data item will be padded with blanks (ASCII code 32). The NCL operator , SKIP, can be used to remove any unwanted spaces (cf. *NCL Programming Manual*).

- When a procedure is called whit output parameters (OUT), you must use SQL_RETURN to retrieve any errors.

5. The syntax checker does not check anything typed after ‘SQL_EXEC’. So, if SQL_PROC is mispelt, a syntax error will not be generated but SQL errors will occur.

Example

```
; Creation of the procedure

SQL_EXEC CREATE PROCEDURE RPCTEST(@PARAM1 INT OUT, \
                          @PARAM2 INT OUT, \
                          @PARAM3 INT OUT, \
                          @PARAM4 INT OUT) \
 AS \
 BEGIN \
 SELECT @PARAM1 = @PARAM1 + @PARAM4 \
 SELECT @PARAM2 = @PARAM2 + @PARAM4 \
 SELECT @PARAM3 = @PARAM3 + @PARAM4  \
 SELECT AU_LNAME FROM AUTHORS  \
 RETURN 200 \
 END


IF SQL_ERROR% <> 0
    MESSAGE "Error create proc with FETCH",SQL_ERRMSG$(SQL_ERROR%)
      sql_exec ns_function statement into :s$
      message "",s$
ENDIF

; Call of the RPCTEST procedure
LOCAL A%, B%, C%, D%, R%
LOCAL  B$

A% = 2
B% = 3
C% = 4
D% = 1

SQL_EXEC SQL_PROC RPCTEST (@PARAM1 OUT, @PARAM2 OUT, \
                           @PARAM3 OUT, @PARAM4 OUT )     \
         VALUES ( :A% , :B% , :C% , :D% )

if SQL_ERROR% <> 0
   MESSAGE "Error SQL_PROC RPCTEST",SQL_ERRMSG$(SQL_ERROR%)
ENDIF

; Retrieve the result of the procedure in NCL variables
; Here we retrieve the "Message retrieved by a FETCH" in B$
; Retrieve the result of a SELECT executed on AUTHORS
WHILE SQL_ERROR% = 0
  SQL_EXEC FETCH INTO :B$
  IF SQL_ERROR% <> 0
      if SQL_ERROR% < 0
         MESSAGE "Error AUTHORS:B$",SQL_ERRMSG$(SQL_ERROR%)
      ENDIF
  ELSE
```

```
         INSERT AT END B$ TO LB1
  ENDIF
ENDWHILE

; return code of the procedure
SQL_EXEC SQL_RETURN:R%
if SQL_ERROR% <> 0
   MESSAGE "Error SQL_RETURN:R%",SQL_ERRMSG$(SQL_ERROR%)
ENDIF

; Display the OUT parameters of the procedure
; A%, B%, C% have been incremented by 1 (by d%)
INSERT AT END "Result"&&A%&&B%&&C% TO LB1

; Display R% (= 200)
INSERT AT END  "Return code from RPCTEST"&&R% TO LB1
```

See also SQL_RETURN, SQL_DB2PROC, SQL_ERROR%, SQL_ERRMSG$

SQL_DB2PROC command

Executes a stored procedure (calls a DB2 package) on AS/400.

⚠ The SQL_DB2PROC is not support anymore from Sybase 10.

Syntax **SQL_DB2PROC** *procedure-name*
[(@Parameter-name1 [OUT] [,@Parameter-name2 [OUT]...])
VALUES *(:Host-Variable1 [,:Host-Variable2...])]*

Parameter	*procedure-name*	CSTRING I	name of the procedure to call

Notes

1. In NSnnSxx, calls to DB2 packages have been improved. To take advantage of this feature, you should always use SQL_EXEC followed by the new keyword SQL_DB2PROC.
2. The SQL_DB2PROC keyword informs the parser that a package is being called. This keyword must be followed by the name of the package called.

   A package is similar to a stored procedure. It contains precompiled SQL queries that are executed when the package is called.
3. If the package called has parameters, their names must be specified exactly as they are defined in the procedure.

Parameters	**Syntax**
Input parameter	*parameter-name* in the parameter list. *parameter-name* can be followed by IN which is the default in any case.
Output or Input/Output	*parameter-name* must be followed by OUT
Values of the procedure's Input parameters	after the VALUES keyword, supply a list of host variables containing the values of the parameters.

Example

```
; Execution of the following RPCTEST procedure created on a une
; Sybase server database via DB2
; This procedure is doing only the calculation part of the SQL_PROC example.

LOCAL A%, B%, C%,D%, R%

A% = 2
B% = 3
```

```
C% = 4
D% = 1

SQL_EXEC SQL_DB2PROC RPCTEST (@param1 OUT, @param2 OUT,  \
                              @param3 OUT, @param4)      \
         VALUES (:A%,:B%,:C%, :D%)
IF SQL_ERROR% <> 0
      ; Error processing
ENDIF

; It ends the procedure and retrieves the return code of the procedure

SQL_EXEC SQL_RETURN :R%
IF SQL_ERROR% <> 0
      ; Error processing
ENDIF

MESSAGE "Procedure result", a% && b% && c%

; Displays of R%
MESSAGE "return code RPCTEST",R%
```

See also SQL_RETURN, SQL_PROC, SQL_ERROR%, SQL_ERRMSG$

SQL_RETURN command

Retrieves the error code from the last procedure executed on a Sybase System server.

Syntax	**SQL_RETURN** : *return-code*			
Parameter	*return-code*	INT(4)	O	return code

Notes

1. If the procedure was executed unsuccessfully, a negative error code (between -1 and -n) will be returned. This error code depends on the client libraries used.
2. This command MUST be used when you call procedures with OUT parameters (cf. SQL_PROC command).
3. The syntax checker does not check anything typed after 'SQL_EXEC'. So, if SQL_RETURN is misspelled, a syntax error will not be generated but SQL errors will occur.
4. When processing a stored procedure, all of the SQL_EXEC commands must be verified by SQL_ERROR% to ensure the correct execution of the procedure.

Example

```
; Creation of the procedure

SQL_EXEC CREATE PROCEDURE RPCTEST(@PARAM1 INT OUT, \
                          @PARAM2 INT OUT, \
                          @PARAM3 INT OUT, \
                          @PARAM4 INT OUT) \
 AS \
 BEGIN \
 SELECT @PARAM1 = @PARAM1 + @PARAM4 \
 SELECT @PARAM2 = @PARAM2 + @PARAM4 \
 SELECT @PARAM3 = @PARAM3 + @PARAM4  \
 SELECT AU_LNAME FROM AUTHORS  \
 RETURN 200 \
 END


IF SQL_ERROR% <> 0
    MESSAGE "Error create proc with FETCH",SQL_ERRMSG$(SQL_ERROR%)
      sql_exec ns_function statement into :s$
      message "",s$
ENDIF

; Call of the RPCTEST procedure
LOCAL A%, B%, C%, D%, R%
LOCAL  B$

A% = 2
B% = 3
C% = 4
```

```
D% = 1

SQL_EXEC SQL_PROC RPCTEST (@PARAM1 OUT, @PARAM2 OUT, \
                           @PARAM3 OUT, @PARAM4 OUT )     \
         VALUES ( :A% , :B% , :C% , :D% )

if SQL_ERROR% <> 0
   MESSAGE "Error SQL_PROC RPCTEST",SQL_ERRMSG$(SQL_ERROR%)
ENDIF

; Retrieve the result of the procedure in NCL variables
; Here we retrieve the "Message retrieved by a FETCH" in B$
; Retrieve the result of a SELECT executed on AUTHORS
WHILE SQL_ERROR% = 0
  SQL_EXEC FETCH INTO :B$
  IF SQL_ERROR% <> 0
      if SQL_ERROR% < 0
         MESSAGE "Error AUTHORS:B$",SQL_ERRMSG$(SQL_ERROR%)
      ENDIF
  ELSE
         INSERT AT END B$ TO LB1
  ENDIF
ENDWHILE

; return code of the procedure
SQL_EXEC SQL_RETURN:R%
if SQL_ERROR% <> 0
   MESSAGE "Error SQL_RETURN:R%",SQL_ERRMSG$(SQL_ERROR%)
ENDIF

; Display the OUT parameters of the procedure
; A%, B%, C% have been incremented by 1 (by d%)
INSERT AT END "Result"&&A%&&B%&&C% TO LB1

; Display R% (= 200)
INSERT AT END  "Return code from RPCTEST"&&R% TO LB1
```

See also SQL_PROC, SQL_DB2PROC, SQL_ERROR%, SQL_ERRMSG$

RECORD, REEXECUTE commands

The RECORD command records an SQL command so that it can be re-executed with REEXECUTE command. When the REEXECUTE command is called, you pass only the new values as parameters.

Syntax **RECORD** *SQL-statement*

and

REEXECUTE

Parameter	*SQL-statement*	CSTRING	I	SQL sequence to record

Notes

1. ⚠ The parameters in the sequence SQL must still be valid when SQL_EXEC REEXECUTE is called.
2. After RECORD, any other SQL statement other than REEXECUTE cancels the current RECORD operation.
3. They also support the RECORD REEXECUTE commands which combines the adaptability of dynamic SQL with the speed of static SQL. In fact, a dynamic SQL order is used when the RECORD command is executed. When the REEXECUTE command is executed, as the analysis of query has already been done by the motor, only the values of the host variables are set.

Example

```
LOCAL CODE%
LOCAL NAME$(25)

SQL_EXEC CREATE TABLE EMP(EMPNO INTEGER,ENAME CHAR(25))

CODE = 1
NAME$ = "NAME1"

SQL_EXEC RECORD INSERT INTO EMP VALUES (:CODE%,:NAME$)
FOR I= 2 TO 100
    CODE = I
    NAME$ = "NAME" & I
    SQL_EXEC REEXECUTE
ENDFOR

; ---- The EMP table contains now
; (  1, "NAME1")
; (  2, "NAME2")
; (  3, "NAME3")
; ...
; ( 99, "NAME99")
; (100, "NAME100")
```

TYPE_SQL_INSERT_BLOB%, TYPE_SQL_SELECT_BLOB% types for blobs

Enables management of binary large objects, larger than 32K but whose size remains limited by Sybase.

Notes

1. Two new NCL data types have been added to NSDBMS.NCL and are to be declared in the *Type_Var* field of the NCLVAR structure:
   - **TYPE_SQL_INSERT_BLOB%**
   - **TYPE_SQL_SELECT_BLOB%**

   They are used for :
   - inserting a binary file into the database
   - retrieving a binary file from the database
2. The images inserted with TYPE_SQL_INSERT_BLOB% are not limited in size.
3. The selection of images with TYPE_SQL_SELECT_BLOB% is limited by default to 1 Mo. However, this size should be modified by NS_FUNCTION CHANGEOPTION TEXTSIZE instruction.
4. Some using rules :
   - the table must contain just one column of IMAGE or TEXT type.
   - The SELECT clause must have just one column : IMAGE.
   - We must force the IMAGE column to NULL by a UPDATE clause before the insertion.
   - The insertion is done by a SELECT clause (and not by an INSERT clause) with *Type_Var* TYPE_SQL_INSERT_BLOB%.
   - The cursor mode must be fixed to 3.

Example

```
LOCAL NCLVAR HL[4]
LOCAL INT IMAGNO
LOCAL DESCRIP$
LOCAL FIMAGE$
LOCAL INT J
LOCAL SQL$
LOCAL BMP%
LOCAL CURSORMODE%
LOCAL Opt$
LOCAL Val%, i%
LOCAL CSTRING Req$(2000)

CURSORMODE% = DB_SYBASE_CURSORNONE ;3
SQL_EXEC NS_FUNCTION SETCURSORMODE :CURSORMODE%
SQL_EXEC DROP TABLE BIGIMAGE
IF SQL_ERROR% <> 0
    MESSAGE "Error ",SQL_ERRMSG$(SQL_ERROR%)
 ENDIF
SQL_EXEC NS_FUNCTION STATEMENT INTO :Req$
IF SQL_ERROR% <> 0
    MESSAGE "Error ",SQL_ERRMSG$(SQL_ERROR%)
 ENDIF
Insert AT END Req$ TO LISTBOX1
; Creation of the table
SQL_EXEC CREATE TABLE BIGIMAGE(NUMERO INTEGER,\
DESCRIPTION VARCHAR(255), \
COLIMAGE IMAGE NULL)
IF SQL_ERROR% <> 0
    MESSAGE "Error ",SQL_ERRMSG$(SQL_ERROR%)
 ENDIF
SQL_EXEC NS_FUNCTION STATEMENT INTO :Req$
Insert AT END Req$ TO LISTBOX1

; ---- Insert of TINTIN.BMP in table BIGIMAGE
SQL_EXEC INSERT INTO BIGIMAGE \
VALUES(1,'This is a big picture > 32000 bytes', NULL)
IF SQL_ERROR% <> 0
    MESSAGE "Error INSERT",SQL_ERRMSG$(SQL_ERROR%)
 ENDIF
SQL_EXEC NS_FUNCTION STATEMENT INTO :Req$
Insert AT END Req$ TO LISTBOX1
i% = 1
SQL_EXEC UPDATE BIGIMAGE SET COLIMAGE = NULL WHERE NUMERO = :i%
IF SQL_ERROR% <> 0
    MESSAGE "Error UPDATE",SQL_ERRMSG$(SQL_ERROR%)
 ENDIF
SQL_EXEC NS_FUNCTION STATEMENT INTO :Req$
Insert AT END Req$ TO LISTBOX1

FILL @HL, SIZEOF HL, 0
FIMAGE$ = F_IMAGE
HL[0].PTR_VAR = @FIMAGE$
HL[0].TYPE_VAR = TYPE_SQL_INSERT_BLOB%
HL[0].SIZE_VAR = SIZEOF FIMAGE$

SQL$="SELECT COLIMAGE INTO :FROM BIGIMAGE WHERE NUMERO = 1"
SQL_EXEC_LONGSTR @SQL$, @HL, DEFAULT_CURSOR
IF SQL_ERROR% <> 0
    MESSAGE "Error ",SQL$&&SQL_ERRMSG$(SQL_ERROR%)
ENDIF
SQL_EXEC NS_FUNCTION STATEMENT INTO :Req$
Insert AT END Req$ TO LISTBOX1
; ---- SELECT with automatic writing  in file EXTRACT.BMP
; ---- One & One column only in the SELECT clause , the IMAGE !!!!!
FILL @HL, SIZEOF HL, 0

FIMAGE$ = "C:\TEMP\EXTRACT.BMP"
FERASE FIMAGE$
HL[0].PTR_VAR = @FIMAGE$
HL[0].TYPE_VAR = TYPE_SQL_SELECT_BLOB%
HL[0].SIZE_VAR = SIZEOF FIMAGE$

SQL$="SELECT COLIMAGE INTO : FROM BIGIMAGE"
SQL_EXEC_LONGSTR @SQL$, @HL, DEFAULT_CURSOR
```

```
IF SQL_ERROR% <> 0
    MESSAGE "Error ",SQL$&&SQL_ERRMSG$(SQL_ERROR%)
 ENDIF
SQL_EXEC NS_FUNCTION STATEMENT INTO :Req$
Insert AT END Req$ TO LISTBOX1

; ---- Display of the picture
BMP% = CREATEBMP%(FIMAGE$)
MOVE BMP% TO BMPF

CURSORMODE% = DB_SYBASE_CURSORDEFAULT ;0
SQL_EXEC NS_FUNCTION SETCURSORMODE :CURSORMODE%
```

See also NSDBMS.NCL, SQL_ERROR%, SQL_ERRMSG$, NS_FUNCTION IMAGEON, NS_FUNCTION IMAGEOFF

COMPUTE command

The COMPUTE command in Sybase System must be preceded by a SQL_EXEC instruction and respect the following syntax.

Syntax **COMPUTE** *Aggregate-Function* **(Column-Name)**

Notes

1. See the "*Sybase SQL Command Reference Manual.*".
2. The SELECT command must be made of a incremental function (COUNT, MIN, MAX, AVG, SUM).
3. The execution of a search command containing COMPUTE will cause an error: NSSQLW200 ** COMPUTE RESULT IN PROGRESS.

Example

```
local ROWS_AFFECTED%, A$, B$,cursormode%

SQL_EXEC SELECT EMPNO ,ENAME INTO :A$, :B$,:ROWS_AFFECTED% FROM EMP COMPUTE \
COUNT(EMPNO)
if sql_error% <> 0
  message 'error' , sql_errmsg$(sql_error%)&&sql_error%
endif

WHILE SQL_ERROR%=0 OR SQL_ERROR%=200
 IF SQL_ERROR% =200
  MESSAGE "COMPUTE Number of lines : ", ROWS_AFFECTED%
 ELSE
  INSERT AT END  "ID et NAME"&&A$ && B$ TO LISTBOX1
 ENDIF
 SQL_EXEC FETCH
ENDWHILE

SQL_EXEC NS_FUNCTION ROWCOUNT INTO :ROWS_AFFECTED%

if sql_error% <> 0
  message 'error' , sql_errmsg$(sql_error%)&&sql_error%
endif

MESSAGE "ROWCOUNT Number of lines : ", ROWS_AFFECTED%
```

EXECUTESELECT command

Enables the execution of a stored procedure in cursor mode according to the Sybase System method. Results are only FETCH, no RETURN or OUT variables.

Syntax **EXECUTESELECT** *Procedure-name* **[:Host-Variable1 [,:Host-Variable2...]] :**

Notes

1. For more information about working with stores procedures, see the "*Sybase SQL Command Reference Manual*."
2. The stored procedure can only have:
   - SELECT commands,
   - IN parameters.
3. The parameters must be *Host-Variables*.

Example

```
LOCAL SEARCH$, NAME$, CRDATE$
SQL_EXEC DROP PROCEDURE RPCSELECT
SQL_EXEC CREATE PROCEDURE RPCSELECT(@SEARCH VARCHAR(255) ) \
  AS\
  BEGIN\
  SELECT name, crdate FROM sysobjects \
  WHERE name LIKE (@SEARCH) \
     END
MESSAGE sql_error%, sql_errmsg$ (sql_error%)

SEARCH$ = 'sys%'

; RPCSELECT is executed in cursor mode
SQL_EXEC EXECUTESELECT RPCSELECT :SEARCH$

; Recuperation of data without A$ and B$
SQL_EXEC FETCH INTO :Name$, :CrDate$

WHILE SQL_ERROR%=0
 INSERT AT END "NAME & Create DATE"&& Name$ && CrDate$ to LISTBOX1
 SQL_EXEC FETCH INTO :Name$, :CrDate$
ENDWHILE
```

UPDATE, CURRENT clauses

Principle

1. **FOR UPDATE OF** is used to stop other users from modifying one or more columns for the duration of a transaction when they have been selected by a given user.

   For example, in SQL pseudo code:

```
; ---- Select and lock the columns
SELECT ... WHERE ... FOR UPDATE OF List-of-columns
 ...
 ... SQL orders of modification
 ... using columns of List-of-columns
 ...
; ---- End of transaction and unlock columns
COMMIT ou ROLLBACK
```

2. **WHERE CURRENT OF** is used to update a record which has been preceded by a SELECT command. This way, for this update command, no calculation of the cursor is required (evaluation of a Where-Clause) for the table, as the cursor is already on the record to update.

   For example, in SQL pseudo code shows the use of the FOR UPDATE OF and WHERE CURRENT OF combination:

```
; ---- Select and lock the columns
SELECT COL1 FROM TABLE1 WHERE COL2 > 10 FOR UPDATE OF COL3
WHILE NOTFOUND% = FALSE
  ; ---- Already on the record to modify
  UPDATE TABLE1 SET COL3 = COL3 + 2 WHERE CURRENT OF
  ; ---- go to next record
  FETCH ...
ENDWHILE

; ---- End of transaction and unlock columns
COMMIT ou ROLLBACK
```

Use

1. When using NSnnSxx libraries, the commands are executed dynamically and are based on OCIs. Some SQL commands are only available in *Embedded SQL*, which is the case of the WHERE CURRENT OF clause.
2. When using the NSnnSxx libraries, be sure to respect the following rules:
   - You must be in a transaction to use **FOR UPDATE** or **CURRENT OF**.
   - The order in which the selection of the record must be in the correct format, for example: SELECT ... WHERE ... **FOR UPDATE OF** *Col-list* in order to prevent other users from modifying the current record.
   - When the modification is finished, the transaction must finish with a COMMIT or ROLLBACK, to remove the locks on the *Col-list* columns.
   - FOR UPDATE applies only on a SELECT command.
   - CURRENT OF applies only on a UPDATE or a DELETE commands.

- The SELECT clause must have a INTO parameter or be followed by a FETCH thus the CURRENT OF could be effective.

Example

```
local Name$
SQL_EXEC BEGIN TRANSACTION
; ---- Selection with a lock in the SELECT column
SQL_EXEC SELECT ENAME INTO :Name$ FROM EMP WHERE EMPNO = 10  FOR UPDATE
if sql_error% <> 0
  message 'error' , sql_errmsg$(sql_error%)
endif

WHILE SQL_ERROR% = 0
 SQL_EXEC UPDATE EMP SET ENAME = 'Lara' CURRENT OF
 if sql_error% <> 0
  message 'error' , sql_errmsg$(sql_error%)
 endif

 SQL_EXEC FETCH INTO :Name$
 if sql_error% <> 0 and sql_ERROR% <> 100
  message 'error' , sql_errmsg$(sql_error%)
 endif

ENDWHILE
; ---- To free the lock and take account of the updates
SQL_EXEC COMMIT TRANSACTION
```

See also Sybase System manuals for more informations on transactions.

NS_FUNCTION extensions

The NS_FUNCTIONs correspond to new functionality developed by Nat System to extend database interface options.

The new functionalities can be accessed by the NCL language.

Some commands must be preceded compulsory by the keyword NS_FUNCTION :

SQL_EXEC **NS_FUNCTION** *command*

See also SQL_ERROR%, SQL_ERRMSG$

This section provides an alphabetically arranged description of these functions.

NS_FUNCTION ANSIOFF, ANSION

☞ The two NS_FUNCTION ANSIOFF and ANSION functions have been designed to cope with the following problem. The command SQL_EXEC UPDATE … WHERE … sets SQL_ERROR% to 0, even if there is no record.

In the ANSIOFF mode, if an UPDATE or DELETE statement does not affect any records, no errors are returned.

In the ANSION mode, if an UPDATE or DELETE statement does not affect any records, an error (warning) is returned with the code "100".

Syntax **NS_FUNCTION ANSIOFF**

and

NS_FUNCTION ANSION

Notes

1. ANSIOFF is the default mode.
2. SQL_ERROR% enables you to retrieve the warning returned.

Example

```
; ---- ANSIOFF mode by default
SQL_EXEC DELETE ... WHERE ...
; ---- even if there is no deleted row
;      SQL_ERROR% equals zero.

; ---- ANSION mode
SQL_EXEC NS_FUNCTION ANSION
SQL_EXEC UPDATE ... WHERE ...
IF SQL_ERROR% = 100
   MESSAGE "No rows update",
           SQL_ERROR% && SQL_ERRMSG$(SQL_ERROR%)
ENDIF

; ---- Return to the default mode
SQL_EXEC NS_FUNCTION ANSIOFF
```

See also NSDBMS.NCL, SQL_ERROR%, SQL_ERRMSG$,NS_FUNCTION ERRORCOUNT, GETERROR

NS_FUNCTION CALLBACK

This function enables the redirection of Sybase System CTLIB error messages to an application window. Each time an error message appears, Sybase sends an event to the window.

Lets you set up centralized management of errors for your application. You no longer need to call SQL_ERROR% and SQL_ERRMSG$ after every command.

Syntax **NS_FUNCTION CALLBACK** *:window-handle, :user-event*

Variables

window-handle	INT(4)	I	handle of the window
user-event	INT(4)	I	user event (USER0 - USER15)

Notes

1. ⚠ In UNIX, *window-handle* must use the Nat System handle of the window that will receive a notification each time an error occurs.

   For all other targets, *window-handle* must be assigned using the NCL GETCLIENTHWND%(...) function which receives as input the Nat System handle of the window that will receive a notification each time an error occurs.
2. To determine the processing carried out, you must program the user event. To obtain the notification of the event in *user-event* must contain 0 for USER0, 1 for USER1,.... or 15 for USER15.
3. To cancel this function, set the window handle to zero.
4. Errors and warnings from the DBMS database being used are returned in their native, proprietary format (see the NSDBMS.NCL file for more information about these formats) :

```
SEGMENT DB_ORACLE_CLIENT_STRUCT
  INT   errorType(2)    ; =1 => Warning , =2 => Error
  INT   dbError(2)      ; Error code in abs()
  CHAR  dbErrorStr(512) ; Error message (with a '\0')
ENDSEGMENT
```

5. The type of the error’s message (clientmsg) is sent to PARAM12%. The handle of the structure is sent to PARAM34%.
6. **If the call NS_FUNCTION CALLBACK is carried out without USING**, the window manages error messages with the cursor. In this situation, it is the default cursor error messages that are redirected. It is useful, therefore, to use a cursor window.
7. ☞ The error 100 « NO ROW WAS FOUND » is not traced by the NS_FUNCTION CALLBACK.

Example

```
; the redirection typically at the beginning of the application
LOCAL HDLE_CATCHERR%
LOCAL USER_EVENT%
LOCAL WINDOW_HANDLE%

OPENS CATCHERR,Self%,HDLE_CATCHERR%
MOVE GETCLIENTHWND%(HDLE_CATCHERR%) TO WINDOW_HANDLE%
MOVE 1 TO USER_EVENT%
SQL_EXEC NS_FUNCTION CALLBACK :WINDOW_HANDLE% , :USER_EVENT%
if sql_error% <> 0
  message 'error BODY' , sql_errmsg$(sql_error%)
endif

; the cancel of the redirection

LOCAL USER_EVENT%
LOCAL WINDOW_HANDLE%
;Stop the callback
MOVE 0 TO WINDOW_HANDLE%
MOVE 1 TO USER_EVENT%
SQL_EXEC NS_FUNCTION CALLBACK :WINDOW_HANDLE% , :USER_EVENT%

; ---------------------------------------------
; In the USER1 event of the CATCHERR window
; ---------------------------------------------
```

```
LOCAL MESSAGETYPE%(4)
LOCAL PTR%(4)
MOVE PARAM12% TO MESSAGETYPE%
MOVE PARAM34% TO PTR%

 EVALUATE MESSAGETYPE%
  WHERE CLIENTMSG
   INSERT AT END "CLIENT" TO self%
   INSERT AT END "SEVERITY " & DB_SYBASE_CLIENT_STRUCT(PTR%).SEVERITY TO self%
   INSERT AT END "MSGNUMBER " & DB_SYBASE_CLIENT_STRUCT(PTR%).MSGNUMBER TO self%
   IF (DB_SYBASE_CLIENT_STRUCT(PTR%).MSGSTRINGLEN > 0)
    INSERT AT END "MSGSTRING " & DB_SYBASE_CLIENT_STRUCT(PTR%).MSGSTRING TO \
self%
   ENDIF
   INSERT AT END "OSNUMBER " & DB_SYBASE_CLIENT_STRUCT(PTR%).OSNUMBER TO self%
   IF (DB_SYBASE_CLIENT_STRUCT(PTR%).OSSTRINGLEN > 0)
    INSERT AT END "OSSTRING " & DB_SYBASE_CLIENT_STRUCT(PTR%).OSSTRING TO self%
   ENDIF
   INSERT AT END "OSSTRINGLEN " & DB_SYBASE_CLIENT_STRUCT(PTR%).OSSTRINGLEN TO \
self%
  ENDWHERE
  WHERE SERVERMSG
   INSERT AT END "SERVER" TO self%
   INSERT AT END "MSGNUMBER " & DB_SYBASE_SERVER_STRUCT(PTR%).MSGNUMBER TO self%
   INSERT AT END "STATE " & DB_SYBASE_SERVER_STRUCT(PTR%).STATE TO self%
   INSERT AT END "SEVERITY " & DB_SYBASE_SERVER_STRUCT(PTR%).SEVERITY TO self%
   IF (DB_SYBASE_SERVER_STRUCT(PTR%).TEXTLEN > 0)
    INSERT AT END "TEXT " & DB_SYBASE_SERVER_STRUCT(PTR%).TEXT TO self%
   ENDIF
   INSERT AT END "TEXTLEN " & DB_SYBASE_SERVER_STRUCT(PTR%).TEXTLEN TO self%
   IF (DB_SYBASE_SERVER_STRUCT(PTR%).SVRNLEN > 0)
    INSERT AT END "SVRNAME " &DB_SYBASE_SERVER_STRUCT(PTR%).SVRNAME TO self%
   ENDIF
   INSERT AT END "SVRNLEN " & DB_SYBASE_SERVER_STRUCT(PTR%).SVRNLEN TO self%
   IF (DB_SYBASE_SERVER_STRUCT(PTR%).PROCLEN > 0)
    INSERT AT END "PROC " & DB_SYBASE_SERVER_STRUCT(PTR%).PROC TO self%
   ENDIF
   INSERT AT END "PROCLEN " &DB_SYBASE_SERVER_STRUCT(PTR%).PROCLEN TO self%
   INSERT AT END "LINE " & DB_SYBASE_SERVER_STRUCT(PTR%).LINE TO self%
   INSERT AT END "STATUS " & DB_SYBASE_SERVER_STRUCT(PTR%).STATUS TO self%
   IF (DB_SYBASE_SERVER_STRUCT(PTR%).SQLSTATELEN > 0)
    INSERT AT END "SQLSTATE " & DB_SYBASE_SERVER_STRUCT(PTR%).SQLSTATE TO self%
   ENDIF
   INSERT AT END "SQLSTATELEN " & DB_SYBASE_SERVER_STRUCT(PTR%).SQLSTATELEN TO \
self%
  ENDWHERE
 ENDEVALUATE
```

See also NSDBMS.NCL, SQL_ERROR%, SQL_ERRMSG$,NS_FUNCTION ERRORCOUNT, GETERROR

NS_FUNCTION CHANGEDBCNTX

Enables switching from one database to another among the application's open databases.

☞ This function was developed to carry out SQL queries on different databases.

⚠ To move from one database to another, we recommend using the command NS_FUNCTION CHANGEDBCNTX. It is likely that the AT command will not be supported in future versions.

Syntax **NS_FUNCTION CHANGEDBCNTX** : *logical-DBname*

Variable	*logical-DBname*	CSTRING	I	logical name of the database which becomes the current one

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.

Exemple

```
LOCAL DATABASENAME$

MOVE "PUBS" TO DATABASENAME$

SQL_OPEN "PUBS" , "SYBUSER/PASSWD@SERVER1"

; current base master
SQL_OPEN "MASTER" , "SYBUSER/PASSWD@SERVER2"

; the current base becomes pubs
SQL_EXEC NS_FUNCTION CHANGEDBCNTX :DATABASENAME$
```

See also SQL_OPEN, SQL_CLOSE, AT, SQL_ERROR%, SQL_ERRMSG$, SQL_EXEC

NS_FUNCTION CHANGEOPTION

Changes the connection parameters between the server and the client.

Syntax **NS_FUNCTION CHANGEOPTION :***parameter,:option*

Parameters

parameter	CSTRING	I	parameter to change
option	*	I	value assigned to the parameter

*: depending on the value of *parameter*, the second variable will either be an INTEGER or a STRING (cf. table below).

Notes

For more details on these parameters, see the section on the SET command in the *Sybase System* manual.

Parameter	NCL Type	Minimum	Option	Default
ANSINULL	INTEGER		TRUE/FALSE	FALSE
ANSIPERM	INTEGER		TRUE/FALSE	FALSE
ARITHABORT	INTEGER		TRUE/FALSE	FALSE
ARITHIGNORE	INTEGER		TRUE/FALSE	FALSE
AUTHOFF	STRING		"sa" "sso" "oper"	
AUTHON	STRING		"sa" "sso" "oper"	
CHAINXACTS	INTEGER		TRUE/FALSE	FALSE
CURCLOSEON	INTEGER		TRUE/FALSE	FALSE
CURREAD	STRING		STRING VALUE	NULL

Parameter	NCL Type	Minimum	Option	Default
CURWRITE	STRING		STRING VALUE	NULL
DATEFIRST	INTEGER		1=SUNDAY 2=MONDAY .... 7=SATURDAY	SUNDAY
DATEFORMAT	INTEGER		1=MDY 2=DMY 3=YMD 4=YDM 5=MYD 6=DYM	MDY
FIPSFLAG	INTEGER		TRUE/FALSE	FALSE
FORCEPLAN	INTEGER		TRUE/FALSE	FALSE
FORMATONLY	INTEGER		TRUE/FALSE	FALSE
GETDATA	INTEGER		TRUE/FALSE	FALSE
IDENTITYOFF	STRING		table name	NULL
IDENTITYON	STRING		table name	NULL
ISOLATION	INTEGER		1=LEVEL1 3=LEVEL3	LEVEL1
NOCOUNT	INTEGER		TRUE/FALSE	FALSE
NOEXEC	INTEGER		TRUE/FALSE	FALSE
PARSEONLY	INTEGER		TRUE/FALSE	FALSE
QUOTED_IDENT	INTEGER		TRUE/FALSE	FALSE
RESTREES	INTEGER		TRUE/FALSE	FALSE
ROWCOUNT	STRING	0	INTEGER VALUE	0
SHOWPLAN	INTEGER		TRUE/FALSE	FALSE
STATS_IO	INTEGER		TRUE/FALSE	FALSE

Parameter	NCL Type	Minimum	Option	Default
STATS_TIME	INTEGER		TRUE/FALSE	FALSE
STR_RTRUNC	INTEGER		TRUE/FALSE	FALSE
TEXTSIZE	INTEGER		INTEGER VALUE	32768
TIMEOUT	INTEGER	0	See the Sybase System manual	120 seconds
TRUNCIGNORE	INTEGER		TRUE/FALSE	FALSE

N.B. TRUE = 1 and FALSE = 0

Example

```
LOCAL PARAMETER$
LOCAL TMPS%

MOVE 30 TO TMPS%
MOVE "TIMEOUT" TO PARAMETER$

SQL_EXEC NS_FUNCTION CHANGEOPTION :PARAMETER$, :TMPS%
```

NS_FUNCTION CHARTOHEXAOFF, CHARTOHEXAON

CHARTOHEXAONmode automatically converts binary values in STRING or CSTRING form into BINARY or VARBINARY types.
CHARTOHEXAOFF disables automatic conversions.

Syntax **NS_FUNCTION CHARTOHEXAOFF**

and

NS_FUNCTION CHARTOHEXAON

Notes

1. CHARTOHEXAOFF is the default mode.
2. In the CHARTOHEXAON mode, BINARY or VARBINARY types are recognized by analyzing the contents of STRING or CSTRING parameters. If the character string begins with 0x, then it is considered to contain a binary value.
3. STRING or CSTRING content constraints :
   - First two characters are: "0x" or "0X",
   - The following characters have the values: [0-9] or [A-F]
   - The number of characters is even
4. ☞ In CHARTOHEXAON mode, the insertion in a column of a the following string ‘0x11FF’ is truncated : ‘0x11’.

Example

```
SQL_EXEC NS_FUNCTION CHARTOHEXAON

...

MOVE '0x11FF' TO A$
SELECT * FROM TABLE WHERE col1 = :A$

; is considered as a SELECT * FROM TABLE WHERE col1 = 0x11FF

SQL_EXEC NS_FUNCTION Statement INTO :Req$
Insert AT END Req$ TO LISTBOX1
; result of NS_FUNCTION Statement SELECT * FROM TABLE WHERE col1 = 0x11FF

; return to the active mode by default
SQL_EXEC NS_FUNCTION CHARTOHEXAOFF

SELECT * FROM TABLE WHERE col1 = :A$

SQL_EXEC NS_FUNCTION Statement INTO :Req$
Insert AT END Req$ TO LISTBOX1
; result of NS_FUNCTION Statement SELECT * FROM TABLE WHERE col1 = '0x11FF'
;'0x11FF' is considered as a string not as an Hexadecimal value
```

See also SQL_ERROR%, SQL_ERRMSG$

NS_FUNCTION ERRORCOUNT

Retrieves the number of errors or error messages encountered while executing a query. The numerotation of messages begins with 0.

Syntax **NS_FUNCTION ERRORCOUNT INTO** : *nbr-errors*

Parameter	*nbr-errors*	INT(4)	I/O	number of errors encountered while executing a query

Example

```
LOCAL I%, ROW_COUNT%, ERROR%
local var1%
local  TOTO$
; the error stack is initialized after every query

var1% = 25
TOTO$ = 3.5

SQL_EXEC  drop table TOTO
SQL_EXEC  INSERT INTO TOTO (NUM, COL1) VALUES (:var1%, :TOTO$)

MOVE 0 TO ROW_COUNT%
SQL_EXEC NS_FUNCTION ERRORCOUNT INTO :ROW_COUNT%
INSERT AT END "ROW_COUNT%" && ROW_COUNT% TO LISTBOX1
; Retrieves the number of errors in ROW_COUNT%
IF ROW_COUNT% <> 0
MOVE 0 TO I%
WHILE i% < ROW_COUNT%
 SQL_EXEC NS_FUNCTION GETERROR :i% INTO :ERROR%
 ;Retrieve for each error its number in ERROR%
 INSERT AT END "ERROR" && I% && SQL_ERRMSG$(ERROR%) TO LISTBOX1
 I% = I% + 1
ENDWHILE
ENDIF
; here the insert generate only one error
```

See also NS_FUNCTION GETERROR, SQL_ERROR%, SQL_ERRMSG$, NS_FUNCTION CALLBACK

NS_FUNCTION GETCURRENTDBCNTX

Gives the logical name of the current database.

Syntax **NS_FUNCTION GETCURRENTDBCNTX INTO**:*database-name*

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

Note

1. *database-name* was specified by the first parameter at the opening of the database: SQL_OPEN.

Example

```
SQL_OPEN "TEST1" , " "
SQL_EXEC NS_FUNCTION GETCURRENTDBCNTX INTO :DATABASENAME$
;DATABASENAME$="TEST1"

SQL_OPEN "TEST2" , " "
SQL_EXEC NS_FUNCTION GETCURRENTDBCNTX INTO :DATABASENAME$
;DATABASENAME$="TEST2"
```

See also SQL_OPEN, SQL_CLOSE, NS_FUNCTION GETDBNAME

NS_FUNCTION GETDBNAME

Retrieves the logical name of the current database.

Syntax **NS_FUNCTION GETDBNAME INTO** :*logical-DBname*

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

Example

```
 LOCAL DBNAME$,DBNAME2$

SQL_OPEN "BASE1",'SYBUSER/PASSWD@models'
SQL_OPEN "BASE2",'SYBUSER/PASSWD@pubs'

SQL_EXEC NS_FUNCTION GETDBNAME INTO :DBNAME$

MESSAGE "THE CURRENT DATABASE IS :",DBNAME2$
;DBNAME2$ will get the physical name of Base2 for ex. Models

SQL_EXEC AT BASE1 NS_FUNCTION GETDBNAME INTO :DBNAME$

MESSAGE "the current database is :",DBNAME$
;DBNAME$ will get the physical name of BASE1 for example : Pubs
```

See also SQL_OPEN, SQL_CLOSE, NS_FUNCTION CHANGEDBCNTX

NS_FUNCTION GETERROR

Retrieves an error code based on its occurrence in the error list. Error numbers lie between 0 and the value returned by NS_FUNCTION ERRORCOUNT minus one.

Syntax **NS_FUNCTION GETERROR** *:error-index%* **INTO** *:error-nbr%*

Parameters	*error-index%*	INT(4)	I	index of the error number
	error-nbr%	INT(4)	O	error number

Example

```
LOCAL I%, ROW_COUNT%, ERROR%
local var1%
local TOTO$
; the error stack is initialized after every query

var1% = 25
TOTO$ = 3.5

SQL_EXEC  drop table TOTO
SQL_EXEC  INSERT INTO TOTO (NUM, COL1) VALUES (:var1%, :TOTO$)

MOVE 0 TO ROW_COUNT%
SQL_EXEC NS_FUNCTION ERRORCOUNT INTO :ROW_COUNT%
INSERT AT END "ROW_COUNT%" && ROW_COUNT% TO LISTBOX1
;retrieves the number of errors in ROW_COUNT%
IF ROW_COUNT% <> 0
MOVE 0 TO I%
WHILE i% < ROW_COUNT%
 SQL_EXEC NS_FUNCTION GETERROR :i% INTO :ERROR%
 ;retrieves for ach error its number in ERROR%
 INSERT AT END "ERROR" && I% && SQL_ERRMSG$(ERROR%) TO LISTBOX1
 I% = I% + 1
ENDWHILE
ENDIF
; here the insertion generate only one error
```

See also NS_FUNCTION ERRORCOUNT,SQL_ERROR%, SQL_ERRMSG$, NS_FUNCTION CALLBACK

NS_FUNCTION GIVECOM

Retrieves in the segment COM_AREA the characteristics of a table whose components are not known at selection.

This function is especially useful when processing dynamic queries and removes the need to define host and FETCH command variables.

Syntax **NS_FUNCTION GIVECOM INTO** *: table-characteristics*

Parameter	*table-characteristics*	INT(4)	I/O	pointer to the segment COM_AREA used to retrieve the table's characteristics

Notes

The segment COM_AREA (defined in the file SQL_COM.NCL) is composed of different fields, two of which are pointers (HOST_PTR and SQL_PTR). These two pointers may be retrieved to browse the tables containing the NCL variables (the HOST_PTR pointer) and the SQL variables (the SQL_PTR pointer) concerned by the order being executed.

```
; Definition of the communication structure (GIVECOM INTO:)
SEGMENT COM_AREA
    int reserved(4)     ;reserved
    int transaction(2)  ;reserved
    int statement(2)    ;reserved
    int host_ptr(4)     ;handle towards a segment of NCLELEMENT
                       ;type (defining the NCL host variables)
    int sql_ptr(4)      ;handle towards a segment of SQLELEMENT
                        ;(defining the columns of the query tables)
    int com_ptr(4)      ;reserved
    int num_stat(2)     ;type of queries
                        ; 1 -> SELECT
                        ; 2 -> UPDATE
                        ; 3 -> DELETE
                        ; 4 -> INSERT
                        ; 5 -> others
    int num_col(2)      ; number of columns
    int num_col_compute(2) ;number of COMPUTE columns(not
                           ;applicable for Oracle)
    int len_buf_stat(2) ; size of the buf_stat below
    int buf_stat(4)     ; handle on a buffer containing the
; FETCH INTO instruction [ :,] and as much « :, » as variables to go
; through in a SELECT case
int inited(2)   ;TRUE if it’s OK, FALSE otherwise. To always test
;if it’s TRUE
ENDSEGMENT
```

The SQL_COM.NCL library provides a set of functions required to make use of the NS_FUNCTION GIVECOM INTO function:
Communication structure.
Functions that return the type of command to be executed.

All the functions used to retrieve pointers.
Types, sizes and names of the columns affected by the selection.

Once the type of the command has been identified as a SELECT statement (after using the SQL_GET_STATEMENT% and SQL_GET_STATEMENT$ functions), the SQL_EXEC_LONGSTR command can execute the query that will fill the receiving field. The results can be extracted from this field using the functions in the NCL library.

The following is a list of functions in the NCL library:

- **FUNCTION SQL_GET_HOSTPTR%**

  Returns a pointer to an array of variables named COM_NCLELEMENT (definition of NCL host variables).

Variable	*COM_BUFFER%*	INT(4)	Handle on COM_AREA

  Return value INT(4)

```
; Definition of the NCL receiving variables structure
SEGMENT COM_NCLELEMENT
    int buffer_ptr(4)
    int ncltype(2)
    integer ncllength
    int reserved1(2)
    int reserved2(2)
ENDSEGMENT
```

- **FUNCTION SQL_GET_SQLPTR%**

  Returns a pointer to an array of variables named COM_SQLELEMENT.

Variable	*COM_BUFFER%*	INT(4)	Handle on COM_AREA

  Return value INT(4)

```
; Definition of the SQL columns structure
SEGMENT COM_SQLELEMENT
    CSTRING colname(64)             ; Name of the column
    int collength(4)                ; Size of the column
    int coltype(2)                  ; Type of the column
    int colservice(2)               ; Service offered for this column
    int colcomputeref(2)            ; Reference of the column having the compute
ENDSEGMENT
```

- **FUNCTION SQL_GET_STATEMENT%**

  Returns the type of statement executed (integer value) from the num_stat buffer of the COM_AREA segment..

Variable	*COM_BUFFER%*	INT(4)	Handle on COM_AREA

Return value INT(2)

- **FUNCTION SQL_GET_STATEMENT$**

  Returns the type of statement executed (alphanuymeric value) from num_stat buffer of the COM_AREA segment and convert it to a CSTRING value.

  The values of the num_stat are the following :
  1 for SELECT
  2 for UPDATE
  3 for DELETE
  4 for INSERT
  0 for other type of queries

Variable	*STATEMENT%*	INT(4)	SQL_GET_STATEMENT%

Return value CSTRING

- **FUNCTION SQL_GET_NBCOL%**

  Returns the number of columns retrieved by the statement.

Variable	*COM_BUFFER%*	INT(4)	Handle on COM_AREA

Return value INT(2)

- **FUNCTION SQL_GET_LENGTHFETCH%**

  Returns the size of the fetch buffer.

Variable	*COM_BUFFER%*	INT(4)	Handle on COM_AREA

Return value INT(4)

- **FUNCTION SQL_GET_FETCHPTR%**

  Returns the pointer to the fetch buffer.

Variable	*COM_BUFFER%*	INT(4)	Handle on COM_AREA

  Return value INT(4)

- **FUNCTION SQL_GET_HOSTCOLUMNPTR%**

  Returns the pointer to the data in an element in the array of NCL variables.

Variable	*COM_BUFFER%*	INT(4)	Handle on COM_AREA
	COLUMN%	INT(2)	Order of the NCL variable

  Return value INT(4)

- **FUNCTION SQL_GET_HOSTCOLUMNTYPE%**

  Returns the data type for an element in the array of NCL variables (integer value).

Variable	*COM_BUFFER%*	INT(4)	Handle on COM_AREA
	COLUMN%	INT(2)	Order of the NCL variable

  Return value INT(2)

- **FUNCTION SQL_GET_HOSTCOLUMNTYPE$**

  Returns the data type for an element in the array of NCL variables (alphanumeric value).

Variable	*TYPE%*	INT(4)	SQL_GET_HOSTCOLUMNLENGTH%

  Return value CSTRING(80)

- **FUNCTION SQL_GET_HOSTCOLUMNLENGTH%**

  Returns the data size for an element in the array of NCL variables.

Variable	*COM_BUFFER%*	INT(4)	Handle on COM_AREA
	COLUMN%	INT(2)	Order of the NCL variable

  Return value INT(2)

- **FUNCTION SQL_GET_SQLCOLUMNNAME$**

Returns the column name in the array of SQL columns.

Variable	*COM_BUFFER%*	INT(4)	Handle on COM_AREA
	COLUMN%	INT(2)	Order of the NCL variable

Return value CSTRING(64)

Nat System informs you that the five next functions are not very useful with NS_FUNCTION GIVECOM. However, we let them in this documentation for compatibility with older documentations.

- **FUNCTION SQL_GET_SQLCOLUMNTYPE%**

  Returns the DBMS column type in the array of SQL columns.
  ```
  FUNCTION SQL_GET_SQLCOLUMNTYPE% \
      (INT COM_BUFFER%(4),INT COLUMN%(2))\
           RETURN INT(2)
  ```

- **FUNCTION SQL_GET_SQLCOLUMNLENGTH%**

  Returns the DBMS column size in the array of SQL columns.
  ```
  FUNCTION SQL_GET_SQLCOLUMNLENGTH% \
        (INT COM_BUFFER%(4),INT COLUMN%(2))\
           RETURN INT(4)
  ```

- **FUNCTION SQL_GET_SQLCOLUMNSERVICE%**

  Retrieves the DBMS column service in the array of SQL columns (integer value).
  ```
  FUNCTION SQL_GET_SQLCOLUMNSERVICE%  \
     (INT COM_BUFFER%(4),INT COLUMN%(2)) \
           RETURN INT(2)
  ```

- **FUNCTION SQL_GET_SQLCOLUMNREF%**

  Retrieves the column number referenced by COMPUTE.
  ```
  FUNCTION SQL_GET_SQLCOLUMNREF% \
    (INT COM_BUFFER%(4),INT COLUMN%(2))  \
           RETURN INT(2)
  ```

- **FUNCTION SQL_GET_SQLCOLUMNSERVICE$**

  Retrieves the DBMS service (alphanumeric value).
  ```
  FUNCTION SQL_GET_SQLCOLUMNSERVICE$ \
     (INT service%(2)) \
           RETURN CSTRING(80)
  ```

Example

```
LOCAL COM_AREA_RET%, TOTAL_COL%, I%, NCL_PTR%, BUFFER_PTR%
LOCAL COMPUTE% , HEADER$, A$
MOVE "SELECT * FROM EMP" TO A$
SQL_EXECSTR A$
MOVE 0 TO COM_AREA_RET%
```

```
WHILE SQL_ERROR% = 0
      SQL_EXEC NS_FUNCTION GIVECOM INTO :COM_AREA_RET%
      IF COM_AREA_RET% = 0
             BREAK
      ENDIF
      INSERT AT END SQL_GET_STATEMENT$ (SQL_GET_STATEMENT%(COM_AREA_RET%) ) TO \
LISTBOX1
      ; recuperation of the command string
      UPDATE LISTBOX1
      IF SQL_GET_STATEMENT%(COM_AREA_RET%) <> 1
             ; the value of the command is different from the SELECT
             RETURN 1
      ENDIF

      MOVE SQL_GET_HOSTPTR%(COM_AREA_RET%) TO NCL_PTR%
      ; recuperation of the pointer on the NCL variables array
      MOVE SQL_GET_NBCOL%(COM_AREA_RET%) + SQL_GET_NBCOMPUTE%(COM_AREA_RET%)\
      TO TOTAL_COL%
      ; retrieve the number of columns + the number of COMPUTE type column
      IF SQL_GET_LENGTHFETCH%(COM_AREA_RET%) <> 0
             ; if the buffer Fetch size is <> 0
             i% = SQL_GET_FETCHPTR%(COM_AREA_RET%)
             MOV i% , @A$ , 255
      INSERT AT END A$ TO LISTBOX1

             SQL_EXEC_LONGSTR SQL_GET_FETCHPTR%(COM_AREA_RET%) , NCL_PTR%, -1
             ; retrieve the pointer to the Fetch buffer + execute
      ELSE
             BREAK
      ENDIF
      WHILE SQL_ERROR% = 0
             MOVE 0 TO I%
             WHILE I% < TOTAL_COL%
             MOVE SQL_GET_HOSTCOLUMNPTR%(COM_AREA_RET% ,i% ) TO BUFFER_PTR%
                    ; retrieve a pointer containing a NCL variable
                    IF BUFFER_PTR% = 0
                           MOVE I% + 1 TO I%
                           CONTINUE
                    ENDIF
                    MOVE "    " to HEADER$
                    IF SQL_GET_SQLCOLUMNSERVICE% (COM_AREA_RET%,i%) <> 1
                    ; if the service is not a column
                           MOVE I% + 1 TO I%
                           CONTINUE
                    ENDIF
                    EVALUATE SQL_GET_HOSTCOLUMNTYPE%(COM_AREA_RET% ,i% )
                    ; evaluation of the column type
                    ;CONST TYPE_SQL_INT%         0
                    ;CONST TYPE_SQL_STRING%      1
                    ;CONST TYPE_SQL_CSTRING%     2
                    ;CONST TYPE_SQL_NUM%         3
                    ;CONST TYPE_SQL_INSERT_BLOB% 13
                    WHERE 0 ; integer
                           ; retrieve the size of the data
                    EVALUATE SQL_GET_HOSTCOLUMNLENGTH%(COM_AREA_RET% ,i%)
                           WHERE 1 ; 1-byte integer
                           ; retrieve the name of the column
                                  INSERT AT END HEADER$ && \

SQL_GET_SQLCOLUMNNAME$(COM_AREA_RET%, I% ) && ":" &&\
                                                   ASC%
(COM_INT1(BUFFER_PTR%).i1) TO LISTBOX1
                           ENDWHERE
                           WHERE 2 ; it’s an integer of 2
```

```
                            ; retrieve the name of the column
                            INSERT AT END HEADER$ && \

      SQL_GET_SQLCOLUMNNAME$(COM_AREA_RET% ,i% )&&\
                                                      ":" &&
COM_INT2(BUFFER_PTR%).i2 TO LISTBOX1
                            ENDWHERE
                            WHERE 4 ; it's an integer of 4
                            INSERT AT END HEADER$ && \

      SQL_GET_SQLCOLUMNNAME$(COM_AREA_RET% ,i% )&&\
                                                             ":" &&
COM_INT4(BUFFER_PTR%).i4 TO LISTBOX1
                            ENDWHERE
                            ENDEVALUATE
                            ENDWHERE
                            WHERE 2 ; it's a C string
                            INSERT AT END HEADER$ && \
                            SQL_GET_SQLCOLUMNNAME$(COM_AREA_RET%, I% ) &&\
                            ":" && COM_STRING(BUFFER_PTR%).CS TO LISTBOX1
                            ENDWHERE
                            WHERE 3 ; it's a real
                            EVALUATE SQL_GET_HOSTCOLUMNLENGTH%(COM_AREA_RET%,\
I%)
                            ; retrieve the size of the WHERE 4 ;
                            ; retrieve the name of the column and the value of
                            ; real of size 4
                            INSERT AT END HEADER$ && \

      SQL_GET_SQLCOLUMNNAME$(COM_AREA_RET%, I% ) &&\
      ":" && COM_FLOAT4(BUFFER_PTR%).f4 TO LISTBOX1
                            ENDWHERE
                            WHERE 8 ;
                            ; retrieve the name of the column and the value of
                            ; the real of 8
                            INSERT AT END HEADER$ && \

      SQL_GET_SQLCOLUMNNAME$(COM_AREA_RET%, I% ) &&\
                                                             ":" &&
COM_FLOAT8(BUFFER_PTR%).f8 TO LISTBOX1
                                           ENDWHERE
                                    ENDEVALUATE
                            ENDWHERE
                            ELSE
                            ; retrieve the type of the NCL column
                            INSERT AT END "NCLType" && "INVALID" && \
                            SQL_GET_HOSTCOLUMNTYPE%(COM_AREA_RET%, i%) TO \
LISTBOX1
                            ; retrieve the type of the NCL column
                     ENDEVALUATE
                     MOVE I% + 1 to I%
              ENDWHILE
              SQL_EXEC_LONGSTR SQL_GET_FETCHPTR%(COM_AREA_RET%),NCL_PTR%,-1
              ; execution of the fetch from the pointer on the FETCH buffer
      ENDWHILE
      UPDATE LISTBOX1
      IF SQL_ERROR% = 100
              INSERT AT END "END OF FETCH" TO LISTBOX1
              INSERT AT END "" TO LISTBOX1
      ENDIF
      IF SQL_ERROR% <> 100
              IF SQL_ERROR% > 0
                     MESSAGE "WARNING" && SQL_ERROR% , SQL_ERRMSG$(SQL_ERROR%)
              ENDIF
              IF SQL_ERROR% < 0
```

```
                     IF SQL_ERROR% = -32085;No more results to fetch
                        INSERT AT END "END OF RESULT" TO LISTBOX1
                     ELSE
                            MESSAGE "ERROR" &&SQL_ERROR% ,
SQL_ERRMSG$(SQL_ERROR%)
                     ENDIF
              ENDIF
      ENDIF
ENDWHILE
```

See also SQL_EXEC_LONGSTR.

NS_FUNCTION IMAGEOFF, IMAGEON

IMAGEON mode enables binary object management (for example bitmaps) of size limited to 32 000 bytes. This handling is carried out in the NCL SEGMENT SQL_IMAGE defined in NSDMS.NCL.

Syntax **NS_FUNCTION IMAGEOFF**

and

NS_FUNCTION IMAGEON

Notes

1. IMAGEOFF is the default mode.
2. Binary objects are manipulated using an SQL_IMAGE segment:
```
SEGMENT SQL_IMAGE
 INT REALSIZE(4) ; allocation size of the buffer
 INT LENGTH%(4)  ; size really read (when select)
 INT PTR%(4)     ; Address of the buffer
ENDSEGMENT
```
3. To do a SELECT in a binary object, you must be in **DB_SYBASE_CURSORNONE**(3) mode : compatible (cf. NS_FUNCTION SETCURSORMODE).
4. The maximum authorized size is 32K. If you want to handle BLOBs (large images) see SQL_EXEC_LONGSTR (TYPE_SQL_INSERT_BLOB% and TYPE_SQL_SELECT_BLOB%).
5. Images are not the only type of binary objects. Any type of binary file can be stored.
6. ⚠ Binary storage is not cross-platform.

Example
```
sql_exec drop table aelta
if sql_error% <> 0
  message 'error drop' , sql_errmsg$(sql_error%)
endif
;sql_exec create table aelta ( ID INT, LONGSTR  VARCHAR(200) NULL, COLIMAGE
;IMAGE NULL)
sql_exec create table aelta ( ID INT, LONGSTR  VARCHAR(200) NULL, COLIMAGE
BINARY NULL)
if sql_error% <> 0
  message 'error create' , sql_errmsg$(sql_error%)
endif

;Insert
LOCAL DEST$(80),DATA%,SIZE%(4),NBREAD%(2),FILE%,NIL%,FNAME$, hbmp%
LOCAL SQL_IMAGE LOCALIMAGE
LOCAL Opt$
LOCAL Val%
LOCAL CURSORMODE%

CURSORMODE% = 3
SQL_EXEC NS_FUNCTION SETCURSORMODE :CURSORMODE%
```

```
;;;;; To select binaries >  1 Mega, increase text size !!!
Opt$ = 'TEXTSIZE'
Val% = 30001
SQL_EXEC NS_FUNCTION CHANGEOPTION :Opt$, :Val%
if (sql_error% < 0)
      message 'textsize', sql_errmsg$ (sql_error%)
endif

; ---- Changing mode
SQL_EXEC NS_FUNCTION IMAGEON
IF SQL_ERROR% <> 0
      MESSAGE "IMAGEON", SQL_ERROR% && SQL_ERRMSG$(SQL_ERROR%)
      RETURN 1
ENDIF
; ---- Reading of the file and transferring in DATA%
FNAME$ = "(NS-BMP)\TINTIN.BMP"
HBMP%=CREATEBMP%(FNAME$)
BMPT = HBMP%
;twisting the FGETSIZE% bug. Does not interpret the environment variables
FNAME$ = "D:\TESTS\BMP\TINTIN.BMP"

SIZE%=FGETSIZE%(FNAME$) ; = 25000 in this example
INSERT AT END "SIZE "& SIZE% TO LISTBOX1
NEW SIZE%,DATA%
FILE%=F_OPEN%(1,FNAME$)
F_BLOCKREAD FILE%, DATA%, SIZE%, NBREAD%
IF F_ERROR%
      MESSAGE"ERROR", "Failed to load " & FNAME$ &"!"
      F_CLOSE FILE%
      DISPOSE DATA%
      RETURN 1
ENDIF
; ---- Insertion in the t_image table
LOCALIMAGE.REALSIZE = SIZE%
LOCALIMAGE.LENGTH% = SIZE%
LOCALIMAGE.PTR% = DATA%
SQL_EXEC INSERT INTO AELTA (ID, LONGSTR, COLIMAGE) VALUES (10,'An island \
Between the sky and the water', :LOCALIMAGE)
IF SQL_ERROR% <> 0
      MESSAGE "INSERT IMAGE", SQL_ERROR% && SQL_ERRMSG$(SQL_ERROR%)
      F_CLOSE FILE%
      DISPOSE DATA%
      RETURN 1
ENDIF
F_CLOSE FILE%
DISPOSE DATA%
SQL_EXEC COMMIT

;Select

LOCAL DEST$(80),DATA%,SIZE%(4),NBREAD%(2),FILE%,NIL%,FNAME$, hbmp%
LOCAL SQL_IMAGE LOCALIMAGE
LOCAL Opt$
LOCAL Val%
LOCAL CURSORMODE%

;CURSORMODE% = 3
;SQL_EXEC NS_FUNCTION SETCURSORMODE :CURSORMODE%

; ---- Changing mode
SQL_EXEC NS_FUNCTION IMAGEON
IF SQL_ERROR% <> 0
      MESSAGE "IMAGEON", SQL_ERROR% && SQL_ERRMSG$(SQL_ERROR%)
      RETURN 1
```

```
ENDIF

LOCALIMAGE.realsize = 3000
NEW LOCALIMAGE.realsize,LOCALIMAGE.PTR%
;SQL_EXEC SELECT COLIMAGE INTO :LOCALIMAGE FROM AELTA WHERE ID = 10
SQL_EXEC SELECT COLIMAGE INTO :LOCALIMAGE FROM AELTA
IF SQL_ERROR% <> 0
      MESSAGE "SELECT IMAGE",SQL_ERROR% && SQL_ERRMSG$(SQL_ERROR%)
ELSE
      ; ---- Displays the image in the CTRLBMP bitmap control
      ; (here LOCALIMAGE.length% values 25000)
      FNAME$="d:\tests\bmp\SOUVENIR.BMP"
      FILE%=F_CREATE%(1,FNAME$)
      INSERT AT END "FILE% "& FILE% TO LISTBOX1
      F_BLOCKWRITE FILE%, LOCALIMAGE.PTR%, LOCALIMAGE.REALSIZE, NBREAD%
      INSERT AT END "F_BLOCKWRITE REALSIZE "& LOCALIMAGE.REALSIZE TO LISTBOX1
      IF F_ERROR%
             MESSAGE"ERROR", "Failed to write " & FNAME$ &"!"
             F_CLOSE FILE%
             DISPOSE LOCALIMAGE.PTR%
             RETURN 1
      ENDIF
      HBMP%=CREATEBMP%(FNAME$)
      BMPOUT = HBMP%
      F_CLOSE FILE%
      DISPOSE LOCALIMAGE.PTR%
ENDIF
; ---- Return to the default mode
SQL_EXEC NS_FUNCTION IMAGEOFF
```

See also NSDBMS.NCL, SQL_ERROR%, SQL_ERRMSG$, TYPE_SQL_INSERT_BLOB%, TYPE_SQL_SELECT_BLOB%, NS_FUNCTION SETCURSORMODE

NS_FUNCTION KILLQUERY

Kills the query sent to the server.

☞ This function has been created to avoid the jamming of an application when a FETCH command take too many times to end.

Syntax **NS_FUNCTION KILLQUERY**

Note

1. This function can be used to abort a query currently being processed by the server.

Example

```
LOCAL I%, PRICEINCLUSIVEOFTAX%, TOTAL%

I%     = 0
TOTAL% = 0
SQL_EXEC SELECT PIOTAX FROM LFACTURE WHERE NOFACT = 10
WHILE SQL_ERROR% = 0
  IF I% >= 4
     SQL_EXEC NS_FUNCTION KILLQUERY
     BREAK
  ELSE
    SQL_EXEC FETCH INTO :PRICEINCLUSIVEOFTAX%
    TOTAL% = TOTAL% + PRICEINCLUSIVEOFTAX%
    I% = I% + 1
  ENDIF
ENDWHILE
MESSAGE " The sum of " & I% & \
        " first lines of the bill n° 10 is", TOTAL%
```

NS_FUNCTION PROCEMPTYSTRINGNULL

Enables inserting a NULL in the database instead of the empty string transmitted as an argument to a stored procedure.

Syntax **NS_FUNCTION PROCEMPTYSTRINGNULL**

Note

1. This NS_FUNCTION makes possible compensating the modification in the last version of Sybase server. Sybase does not allow any more insertion of empty strings as NULLs. Sybase recommends using null indicators. NS_FUNCTION PROCEMPTYSTRINGNULL bypasses the need to revising the application code by inserting NULLs for empty strings for any call of stored procedure without correcting it.

NS_FUNCTION QUOTEOFF, QUOTEON

The QUOTEON mode automatically manages quotation marks in input parameters for string-type values. This function is used for replacing strings of characters at application runtime.
The QUOTEOFF mode disables this function.

Syntax

NS_FUNCTION QUOTEOFF

and

NS_FUNCTION QUOTEON

Notes

1. QUOTEON is the default mode.
2. Quotation marks must be entered by the user in QUOTEOFF mode and be handled automatically in QUOTEON mode.
3. This functionality is useful only with **DB_SYBASE_CURSORNONE** (compatibility with NSnnS42)

```
MOVE 3 TO mode_cursor%
SQL_EXEC NS_FUNCTION SETCURSORMODE \
:mode_cursor%
```

Example

```
LOCAL FATHER$, ID% ,SON$, IND1%, IND2%, ID%, A$
LOCAL CSTRING Req$(2000)
LOCAL CURSORMODE%

CURSORMODE% = DB_SYBASE_CURSORNONE ;3
SQL_EXEC NS_FUNCTION SETCURSORMODE :CURSORMODE%

SQL_EXEC NS_FUNCTION QUOTEOFF

A$ ='"'&'PIERRE'&'"'
SQL_EXEC SELECT ID, FATHER, SON INTO :ID%, :FATHER$:IND1%, :SON$:IND2% FROM TOTO
WHERE FATHER =:A$
IF SQL_ERROR% <> 0
    MESSAGE "Error ",SQL_ERRMSG$(SQL_ERROR%)
ENDIF

SQL_EXEC NS_FUNCTION STATEMENT INTO :Req$
INSERT AT END Req$ TO LISTBOX1
; this way we can trace the value of A$ ("PIERRE" here)
; SELECT ID, FATHER, SON FROM TOTO WHERE FATHER ="PIERRE"
```

```
WHILE SQL_ERROR% = 0
 INSERT AT END FATHER$  TO LISTBOX1
 SQL_EXEC FETCH
ENDWHILE
SQL_EXEC NS_FUNCTION QUOTEON
CURSORMODE% = DB_SYBASE_CURSORDEFAULT ;0
SQL_EXEC NS_FUNCTION SETCURSORMODE :CURSORMODE%
A$ ='PIERRE'
SQL_EXEC SELECT ID, FATHER, SON INTO :ID%, :FATHER$:IND1%, :SON$:IND2% FROM TOTO
WHERE FATHER =:A$
IF SQL_ERROR% <> 0
    MESSAGE "Error ",SQL_ERRMSG$(SQL_ERROR%)
 ENDIF

; All the variables have to be initialized even if their value will be null
SQL_EXEC NS_FUNCTION STATEMENT INTO :Req$
INSERT AT END Req$ TO LISTBOX1
; SELECT ID, FATHER, SON FROM TOTO WHERE FATHER =@Param5
WHILE SQL_ERROR% = 0
 INSERT AT END FATHER$ TO LISTBOX1
 SQL_EXEC FETCH
ENDWHILE
```

See also NSDBMS.NCL, SQL_ERROR%, SQL_ERRMSG$

NS_FUNCTION ROWCOUNT

Returns the number of rows affected by a query DELETE or UPDATE or the number of FETCH realized after a SELECT.

Syntax **NS_FUNCTION ROWCOUNT INTO** *:nb-rows*

Parameter	*nb-rows*	INT(4)	O	number of rows affected by a query

Note

1. The number of rows retrieved by COMPUTE and ROWCOUNT may differ from one row. Indeed, ROWCOUNT can retrieve the number of FETCH realized, whereas COMPUTE retrieve the number of occurences.

Example 1

```
LOCAL ROWCOUNT%

SQL_EXEC DELETE FROM TABPRODUCT\
         WHERE NOPROD >= 30 AND NOPROD < 40

SQL_EXEC NS_FUNCTION ROWCOUNT INTO :ROWCOUNT%
; If 10 recordings correspond to this filter and then 10 recordingss have been
; deleted, thus ROWCOUNT% will contain 10
; If no recording correspond to this filter, thus ROWCOUNT% will contain 0
```

Example 2

```
LOCAL var1%
LOCAL test$
LOCAL ROWCOUNT%
SQL_EXEC SELECT NUM, COL1 FROM BASE
IF SQL_ERROR% <> 0
    MESSAGE "Error ",SQL_ERRMSG$(SQL_ERROR%)
ENDIF


WHILE SQL_ERROR% = 0
 SQL_EXEC FETCH INTO:var1%,:test$
 IF SQL_ERROR% <> 0
  BREAK
 ENDIF
 INSERT AT END "Var1"&&var1%&& "test"&&test$ TO LISTBOX1
ENDWHILE
SQL_EXEC NS_FUNCTION ROWCOUNT INTO :ROWCOUNT%
Message "Number of occurences = ", ROWCOUNT%
```

Example 3

```
LOCAL ROWS_AFFECTED%, A$, B$,cursormode%

SQL_EXEC SELECT EMPNO ,ENAME INTO :A$, :B$,:ROWS_AFFECTED% FROM EMP COMPUTE \
COUNT(EMPNO)
IF sql_error% <> 0
  MESSAGE 'error' , sql_errmsg$(sql_error%)&&sql_error%
ENDIF
```

```
WHILE SQL_ERROR%=0 OR SQL_ERROR%=200
 IF SQL_ERROR% =200
  MESSAGE "COMPUTE Number of lines : ", ROWS_AFFECTED%
 ELSE
  INSERT AT END  "ID et NAME"&&A$ && B$ TO LISTBOX1
 ENDIF
 SQL_EXEC FETCH
ENDWHILE

SQL_EXEC NS_FUNCTION ROWCOUNT INTO :ROWS_AFFECTED%

IF SQL_ERROR% <> 0
  MESSAGE 'error' , sql_errmsg$(sql_error%)&&sql_error%
ENDIF

MESSAGE "ROWCOUNT Number of lines : ", ROWS_AFFECTED%
```

See also COMPUTE, NS_FUNCTION ANSIOFF, NS_FUNCTION ANSION, SQL_ERROR%, SQL_ERRMSG$

NS_FUNCTION SETBUFFERSIZE

Specifies the buffer size for the using mode DB_SYBASE_CURSOR_DEFAULT(0).

Syntax **NS_FUNCTION SETBUFFERSIZE** *:buffer-size*

Parameter	*buffer-size*	INT(4)	I	size of the buffer

Notes

1. DB_SYBASE_CURSOR_DEFAULT(0) is the mode by default.
2. When A SELECT is executed in DB_SYBASE_CURSOR_DEFAULT(0) mode, we get this parameter to carry out the selection of the recordings. Thus, in that way, several recordings are read by the Sybase System client part.
3. Characteristics :

• **Mode**	Default
• **Maximum size**	Depends on the memory size
• **Minimum size**	1
• **Size by default**	100

Example 1

```
;segment S_COL1
;     int V_COL1
;endsegment

;segment S_COL2
;     cstring V_COL2(10)
;endsegment

;segment S_COL1_S
;     S_COL1 SEG[10]
;endsegment

;segment S_COL2_S
;     S_COL2 SEG[10]
;endsegment

LOCAL COL1%
LOCAL COL2$(10)
LOCAL HCOL1%, HCOL2%
Local int     I%(4)
```

```
sql_exec drop table JYM
SQL_EXEC CREATE TABLE JYM ( COL1 INTEGER , COL2 VARCHAR2(10) )
if sql_error% <> 0
  message 'error' , sql_errmsg$(sql_error%)
endif

; ---- Create segments
NEW S_COL1_S  , HCOL1%
NEW S_COL2_S  , HCOL2%


  S_COL1_S(HCOL1%).SEG[0].V_COL1 = 1
  S_COL2_S(HCOL2%).SEG[0].V_COL2 = "Albert%%%"
  S_COL1_S(HCOL1%).SEG[1].V_COL1 = 2
  S_COL2_S(HCOL2%).SEG[1].V_COL2 = "Bernard%%"
  S_COL1_S(HCOL1%).SEG[2].V_COL1 = 3
  S_COL2_S(HCOL2%).SEG[2].V_COL2 = "DANIEL%%%"
  S_COL1_S(HCOL1%).SEG[3].V_COL1 = 4
  S_COL2_S(HCOL2%).SEG[3].V_COL2 = "ROGER%%%%"


     Move 4 to I%
     SQL_EXEC NS_FUNCTION SETBUFFERSIZE :I% ;
 IF sql_error% <> 0
   MESSAGE 'error' , sql_errmsg$(sql_error%)
 ENDIF


     SQL_EXEC INSERT INTO JYM (COL1, COL2) \
                DESCRIBE (:COL1%, :COL2$) \
                VALUES (:HCOL1%, :HCOL2%)
    IF sql_error% <> 0
      MESSAGE 'error' , sql_errmsg$(sql_error%)
    ENDIF


dispose HCOL1%
dispose HCOL2%
```

Example 2

```
;SEGMENT SNAME
;  CSTRING NAME(11)
;ENDSEGMENT


;SEGMENT BUF_NAME
  ;  SNAME EMP[10]
;ENDSEGMENT
SQL_EXECSTR "CREATE OR REPLACE PACKAGE PACK_TEST AS\
    TYPE namec_arr is table of emp.ename%type index by binary_integer;\
 PROCEDURE IN_OUT(NAMEC_ARRAY OUT NAMEC_ARR, FETCH_SIZE IN OUT INTEGER);\
 cursor c_in_out is select ename from emp;\
END PACK_TEST;"
IF sql_error% <> 0
  message 'error PACKAGE' , sql_errmsg$(sql_error%)
ENDIF
```

```
SQL_EXECSTR 'CREATE OR REPLACE PACKAGE BODY PACK_TEST AS \
PROCEDURE IN_OUT(NAMEC_ARRAY OUT NAMEC_ARR, FETCH_SIZE IN OUT INTEGER) IS \
counter binary_integer;\
begin\
  if not c_in_out%isopen then\
    open c_in_out;\
  end if;\
  counter := 1;\
  loop\
    fetch c_in_out into namec_array(counter);\
    exit when c_in_out%notfound or counter = fetch_size;\
    counter := counter + 1;\
  end loop;\
  if c_in_out%notfound then \
    fetch_size := counter - 1;\
    close c_in_out;\
  end if;\
end;\
end pack_test;'

if sql_error% <> 0
  message 'error BODY' , sql_errmsg$(sql_error%)
endif
;then
LOCAL CSTRING Ename(11)
LOCAL Nbr_Rows%, i%
LOCAL h_nos%,h_names%,h_names2%,h_dats%,curs%
MOVE 10 TO Nbr_Rows%
NEW BUF_NAME, h_names%
MOVE SQL_OPENCURSOR% TO CURS%
SQL_EXEC NS_FUNCTION SETBUFFERSIZE :Nbr_Rows% USING CURS%
; With Oracle 8 you have to describe your parameters if you use a
; custom type in your procedure
; Bug 6765 By aelta
SQL_EXEC SQL_PROC pack_test.in_out (:Ename OUT [VARCHAR2, 1],:Nbr_Rows% IN
[NUMBER, 0]) VALUES (:h_names%) USING CURS%
IF SQL_ERROR% <> 0
    Message "error", SQL_ERROR%&": "&SQL_ERRMSG$(SQL_ERROR%)
ENDIF
MOVE 0 TO i%
WHILE i% < Nbr_Rows%
 INSERT AT END "Longueur"&&length BUF_NAME(h_names%).EMP[i%].NAME&& \
  "Valeur="&BUF_NAME(h_names%).EMP[i%].NAME TO LISTBOX1
 MOVE i%+1 TO i%
ENDWHILE
SQL_CLOSECURSOR
DISPOSE h_names%
```

Example 3

```
MOVE 150 TO BUFSIZE%
SQL_EXEC NS_FUNCTION SETBUFFERSIZE :BUFSIZE%
; 150 recordings are stored in the client part of Sybase
SQL_EXEC SELECT * FROM AUTHORS
```

See also NS_FUNCTION SETCURSORMODE

NS_FUNCTION SETCURSORMODE

Sets the cursor mode used by SQL_OPENCURSOR% or SQL_OPENTHECURSOR% functions.

Syntax	**NS_FUNCTION SETCURSORMODE** *:mode*			
Parameter	*mode*	INT(4)	I	cursor mode used

Notes

1. There are two cursor modes:
   - the default mode,
   - the compatible mode.
2. Each mode has an associated value and default size:
   - `DB_SYBASE_CURSORDEFAULT` value
     - Default mode
     - Default size = 100 rows
   - `DB_SYBASE_CURSORNONE` value
     - Compatible mode
     - Default size = 1 row
3. A comparison of the significance of these modes:
   - **DB_SYBASE_CURSORNONE** (0) mode allows all queries to be carried out without authorizing relative positioning.
   - **DB_SYBASE_CURSORDEFAULT** (3) mode enables all the records contained in the specified buffer to be read with a single Fetch. This is particularly suited to a small selection (<1000 records).
4. The size can be modified by using the command NS_FUNCTION SETBUFFERSIZE.
5. **Default** is the standard mode. When you scan a table, each Fetch statement will be executed individually. This mode is based on Sybase System conventions for using cursors.
6. **Compatible** mode allows you to change the cursor mode for a given cursor (and switch to language mode). This mode ensures compatibility with prior version of Sybase Version.
7. ⚠ This function should only be used in very specific cases.

We suggest switching to this cursor mode only for specific queries then immediately switching to the default mode after.

Example

```
LOCAL MODECURS%

MOVE DB_SYBASE_CURSORBINDING TO MODECURS%
SQL_EXEC NS_FUNCTION SETCURSORMODE :MODECURS%

; the searches will be executed on compatible mode
; command language
```

See also NS_FUNCTION SETBUFFERSIZE

NS_FUNCTION SETLONGASTEXT

The NS_FUNCTION SETLONGASTEXT handles the buffers to be inserted in TEXT or IMAGE type columns, as text.

☞ This function has been developed to improve performances.

Syntax **NS_FUNCTION SETLONGASTEXT**

Note

1. Instead of send the maximum size of TEXT (30000), the driver send the real size of the text with two additional bytes at the Sybase database. This optimizes the send of data packages through the network.

Example

```
LOCAL CSTRING MyString$ (30000)
LOCAL var1%
LOCAL CSTRING Req$(2000)

MyString$="SetLongAsText"
var1%=20
;To insert long char values as a Text, you must call this ns_function :
;it'll send many bytes less over the network thus enhances the response time
SQL_EXEC NS_FUNCTION SETLONGASTEXT
IF SQL_ERROR% <> 0
      MESSAGE "Warning", SQL_ERROR% &&SQL_ERRMSG$(SQL_ERROR%)
ENDIF
; it must be a LONG type to store a variable
SQL_EXEC INSERT INTO TOTO (ID, ASTEXT) VALUES (:var1%, :MyString$)
IF SQL_ERROR% <> 0
      MESSAGE "Error", SQL_ERROR% &&SQL_ERRMSG$(SQL_ERROR%)
ENDIF

SQL_EXEC NS_FUNCTION Statement INTO :Req$
INSERT AT END Req$&"End" TO LISTBOX1

SQL_EXEC COMMIT
IF SQL_ERROR% <> 0
      MESSAGE "Warning", SQL_ERROR% &&SQL_ERRMSG$(SQL_ERROR%)
ENDIF
SQL_EXEC SELECT ASTEXT INTO :MyString$ FROM TOTO
IF SQL_ERROR% <> 0
    MESSAGE "Error ",SQL_ERRMSG$(SQL_ERROR%)
ENDIF
SQL_EXEC NS_FUNCTION STATEMENT INTO :Req$

WHILE SQL_ERROR% = 0
 INSERT AT END Req$&&"resultat"&& MyString$ TO LISTBOX1
 SQL_EXEC FETCH
ENDWHILE
```

```
SQL_EXEC NS_FUNCTION SetLongAsBin
IF SQL_ERROR% <> 0
      MESSAGE "Warning", SQL_ERROR% &&SQL_ERRMSG$(SQL_ERROR%)
ENDIF
```

NS_FUNCTION SETLONGASBIN

This function allows to return to the standard mode of handling the buffers. This mode consists in inserting buffers in the columns of LONG type, like binary code. In this case, the maximum size of the image (30000) is sent to the base.

Syntax **NS_FUNCTION SETLONGASBIN**

Example

```
LOCAL CSTRING MyString$ (30000)
LOCAL var1%
LOCAL CSTRING Req$(2000)

MyString$="SetLongAsText"
var1%=20
;To insert long char values as a Text, you must call this ns_function :
;it'll send many bytes less over the network thus enhances the response time
SQL_EXEC NS_FUNCTION SETLONGASTEXT
IF SQL_ERROR% <> 0
      MESSAGE "Warning", SQL_ERROR% &&SQL_ERRMSG$(SQL_ERROR%)
ENDIF
; it must be a LONG type to store a variable
SQL_EXEC INSERT INTO TOTO (ID, ASTEXT) VALUES (:var1%, :MyString$)
IF SQL_ERROR% <> 0
      MESSAGE "Error", SQL_ERROR% &&SQL_ERRMSG$(SQL_ERROR%)
ENDIF

SQL_EXEC NS_FUNCTION Statement INTO :Req$
INSERT AT END Req$&"End" TO LISTBOX1

SQL_EXEC COMMIT
IF SQL_ERROR% <> 0
      MESSAGE "Warning", SQL_ERROR% &&SQL_ERRMSG$(SQL_ERROR%)
ENDIF
SQL_EXEC SELECT ASTEXT INTO :MyString$ FROM TOTO
IF SQL_ERROR% <> 0
    MESSAGE "Error ",SQL_ERRMSG$(SQL_ERROR%)
ENDIF
SQL_EXEC NS_FUNCTION STATEMENT INTO :Req$

WHILE SQL_ERROR% = 0
 INSERT AT END Req$&&"result"&&MyString$ TO LISTBOX1
 SQL_EXEC FETCH
ENDWHILE

SQL_EXEC NS_FUNCTION SETLONGASBIN
IF SQL_ERROR% <> 0
      MESSAGE "Warning", SQL_ERROR% &&SQL_ERRMSG$(SQL_ERROR%)
ENDIF
```

NS_FUNCTION STATEMENT

Retrieves the full statement used in the query sent to the SQL engine.

Syntax **NS_FUNCTION STATEMENT INTO** *:query-string*

Parameter	*query-string*	CSTRING	O	statement used in the query sent to the SQL engine

Notes

1. ⚠ To track the host variables correctly, you **must** use the **DB_SYBASE_CURSORNONE 3** mode

2. The INTO clause (even if it is precised) is never tracked, except for using the **DB_SYBASE_CURSORNONE 3 mode**.

Example 1

```
LOCAL VALUES$, SENTENCE$

MOVE "HELLO" TO VALUES$
SQL_EXEC SELECT COL1 FROM TABLE WHERE COL2=:VALUES$

SQL_EXEC NS_FUNCTION STATEMENT INTO :SENTENCE$
MESSAGE "The exact command is :", SENTENCE$

; displays the following line
SELECT COL1 FROM TABLE WHERE COL2=@PARAM1
```

Example 2

```
LOCAL FATHER$, ID%, SON$, IND1%, IND2%, ID%, A$
LOCAL CSTRING Req$(2000)
LOCAL CURSORMODE%

CURSORMODE% = DB_SYBASE_CURSORNONE ;3
SQL_EXEC NS_FUNCTION SETCURSORMODE :CURSORMODE%

SQL_EXEC NS_FUNCTION QUOTEOFF

A$ ='"'&'PIERRE'&'"'
SQL_EXEC SELECT ID, FATHER, SON INTO :ID%, :FATHER$:IND1%, :SON$:IND2% FROM TOTO
\ WHERE FATHER =:A$
IF SQL_ERROR% <> 0
    MESSAGE "Error ",SQL_ERRMSG$(SQL_ERROR%)
ENDIF

SQL_EXEC NS_FUNCTION STATEMENT INTO :Req$
INSERT AT END Req$ TO LISTBOX1
; this way we can trace the value of A$ ("PIERRE" here)
; SELECT ID, FATHER, SON FROM TOTO WHERE FATHER ="PIERRE"
```

```
WHILE SQL_ERROR% = 0
 INSERT AT END FATHER$ TO LISTBOX1
 SQL_EXEC FETCH
ENDWHILE

SQL_EXEC NS_FUNCTION QUOTEON
CURSORMODE% = DB_SYBASE_CURSORDEFAULT ;0
SQL_EXEC NS_FUNCTION SETCURSORMODE :CURSORMODE%
A$ ='PIERRE'
SQL_EXEC SELECT ID, FATHER, SON INTO :ID%, :FATHER$:IND1%, :SON$:IND2% FROM TOTO
\ WHERE FATHER =:A$
IF SQL_ERROR% <> 0
    MESSAGE "Error ",SQL_ERRMSG$(SQL_ERROR%)
ENDIF
; All the variables have to be initialized even if their value will be null
SQL_EXEC NS_FUNCTION STATEMENT INTO :Req$
Insert AT END Req$ TO LISTBOX1
; SELECT ID, FATHER, SON FROM TOTO WHERE FATHER =@Param5
WHILE SQL_ERROR% = 0
 INSERT AT END FATHER$ TO LISTBOX1
 SQL_EXEC FETCH
ENDWHILE
```

NS_FUNCTION SETTCPNODELAY

By default, the TCP/IP protocol checks if packets are full each time before sending them. So, when a packet is half full, TCP/IP waits for new data before sending the packet. The SETTCPNODELAY function enables packets to be sent immediately without the size of the content being checked.

☞ The NS_FUNCTION SETTCPNODELAY enables setting of the Sybase TCP_NODELAY option.

Syntax **NS_FUNCTION SETTCPNODELAY**

Example

```
;To invoke TCP_NODELAY,
;code the following SQL_EXEC NS_FUNCTION SETTCPNODELAY

SQL_INIT "NS02S11"
SQL_OPEN "Pubs", "sybuser/passwd@syb11"
SQL_EXEC NS_FUNCTION SETTCPNODELAY
```

NS_FUNCTION TRIMCHAROFF, TRIMCHARON

In TRIMCHARON mode, when a SELECT is executed, the blank spaces at the end of strings are removed. This is very useful when the array table is CHAR or VARCHAR2.

The TRIMCHARON mode allows to limit the size of the buffer in the network.

In TRIMCHAROFF mode, the blank spaces are kept.

Syntax **NS_FUNCTION TRIMCHAROFF**

and

NS_FUNCTION TRIMCHARON

Notes

1. TRIMCHAROFF is the mode by default.
2. TRIMCHARON is of limited importance with Sybase. Sybase does not insert blanks at the end of a string. But TRIMCHARON may be useful if blanks have been inserted by a means external to Nat System development products.

Example

```
SQL_EXEC NS_FUNCTION TRIMCHARON
 SQL_EXEC SELECT 'But string with blanks at the end           ' INTO :C$
 INSERT AT END "{B[LIGHTBLUE]}CHAR=<" & C$ & ">"&"" TO LISTBOX1
 SQL_EXEC SELECT 'But string with blanks at the end           ' INTO :Str$
 INSERT AT END "{B[GREEN]}CHAR=<" & Str$ & ">"&"" TO LISTBOX1
;. The blanks are eliminated
SQL_EXEC NS_FUNCTION TRIMCHAROFF
 SQL_EXEC SELECT 'But string with blanks at the end           ' INTO :C$
 INSERT AT END "{B[LIGHTBLUE]}CHAR=<" & C$ & ">"&"" TO LISTBOX1
 SQL_EXEC SELECT 'But string with blanks at the end           ' INTO :Str$
 INSERT AT END "{B[GREEN]}CHAR=<" & Str$ & ">"&"" TO LISTBOX1
;. The blanks are present
```

See also SQL_ERROR%, SQL_ERRMSG$