Processing Natural Stored Procedures and UDFs

Natural for DB2 supports the writing and executing of Natural stored procedures and Natural user-defined functions (Natural UDFs).

Natural stored procedures are user-written programs that are invoked by the SQL statement `CALL` and executed by DB2 in the SPAS (Stored Procedure Address Space). SPAS is a separate address space reserved for stored procedures.

A function is an operation denoted by a function name followed by zero or more operands that are enclosed in parentheses. A function represents a relationship between a set of input values and a set of result values. If a function has been implemented by a user-written program, DB2 refers to it as a user-defined function (UDF).

The following topics are covered below:

- Types of Natural UDF
- PARAMETER STYLE
- Writing a Natural Stored Procedure
- Writing a Natural UDF
- Example Stored Procedure
- Example Natural User Defined Function

Types of Natural UDF

There are two types of Natural used defined functions (UDF):

- **Scalar UDF**

  The scalar UDF accepts several input arguments and returns one output value. It can be invoked by any SQL statement like a DB2 built-in-function.

- **Table UDF**

  The table UDF accepts several input arguments and returns a set of output values comprising one table row during each invocation.

  You invoke a table UDF with a Natural SQL `SELECT` statement by specifying the table-function name in the `FROM Clause`. A table UDF performs as a DB2 table and is invoked for each `FETCH` operation for the table-function specified in the `SELECT` statement.

PARAMETER STYLE

The PARAMETER STYLE identifies the linkage convention used to pass parameters to a DB2 stored procedure or a DB2 user defined functions (UDFs).

This section describes the PARAMETER STYLEs and the STCB Natural for DB2 uses for processing Natural for DB2 stored procedures or Natural UDFs.

Note:
PARAMETER STYLE GENERAL (or GENERAL WITH NULL) and STCB Layout only apply to Natural stored procedures.

- GENERAL and GENERAL WITH NULL
- STCB Layout
- DB2SQL

GENERAL and GENERAL WITH NULL

Note:
Only applies to Natural stored procedures.

A Natural stored procedure defined with PARAMETER STYLE GENERAL only receives the user parameters specified.

A Natural stored procedure defined with PARAMETER STYLE GENERAL WITH NULL receives the user parameters specified and, additionally, a NULL indicator array that contains one NULL indicator for each user parameter.

Natural stored procedures defined with PARAMETER STYLE GENERAL/PARAMETER STYLE GENERAL WITH NULL, require that the definition of the stored procedure within the DB2 catalog includes one additional parameter of the type VARCHAR in front of the user parameters of the stored procedure.

This parameter in front of the parameters is the Stored Procedure Control Block (STCB); see also *STCB Layout* below.

Below is information on:

- Stored Procedure Control Block
- Example of PARAMETER STYLE GENERAL
- Example of GENERAL WITH NULL

Stored Procedure Control Block

The Stored Procedure Control Block (STCB) contains information the Natural for DB2 server stub uses to execute Natural stored procedures, such as the library and the subprogram to be invoked. It also contains the format descriptions of the parameters passed to the stored procedure.

The STCB is invisible to the Natural stored procedure called. The STCB is evaluated by the Natural for DB2 server stub and stripped off the parameter list that is passed to the Natural stored procedure.

If the caller of a Natural stored procedure defined with PARAMETER STYLE GENERAL/PARAMETER STYLE GENERAL WITH NULL is a Natural program, the program must use a Natural SQL CALLDBPROC statement with the keyword CALLMODE=NATURAL.

If the caller of the Natural stored procedure is *not* a Natural program, the caller has to set up the STCB for the DB2 CALL statement and pass the STCB as the first parameter.

If an error occurs during the execution of a Natural stored procedure defined with PARAMETER STYLE GENERAL/PARAMETER STYLE GENERAL WITH NULL, the error message text is returned to the STCB.

If the caller is a Natural program that uses CALLDBPROC and CALLMODE=NATURAL, the Natural for DB2 runtime will wrap up the error text in the NAT3286 error message.

Example of PARAMETER STYLE GENERAL

In the Natural stored procedure, define the parameters as shown in the example program below:

```
DEFINE DATA PARAMETER
01 P1 ...
01 P2 ...
...
...
01 Pn ...
LOCAL
...
...
END-DEFINE
```

Example of GENERAL WITH NULL

In the Natural stored procedure, define the parameters as shown in the example program below:

```
DEFINE DATA PARAMETER
01 P1 ...
01 P2 ...
...
...
01 Pn ...
01 NULL-INDICATOR-ARRAY  (I2/1:n)
LOCAL
...
...
END-DEFINE
```

STCB Layout

Note:
Only applies to Natural stored procedures.

The following table describes the first parameter passed between the caller and the Natural stored procedure if CALLMODE=NATURAL is specified in a Natural SQL CALLDBPROC statement.

Name	Format	Processing Mode Server
STCBL	I2	Input (size of following information)
Procedure Information		
STCBLENG	A4	Input (printable STCBL)
STCBID	A4	Input (STCB)
STCBVERS	A4	Input (version of STCB 310)
STCBUSER	A8	Input (user ID)
STCBLIB	A8	Input (library)
STCBPROG	A8	Input (calling program)
STCBPSW	A8	Unused (password)
STCBSTNR	A4	Input (CALLDBPROC statement number)
STCBSTPC	A8	Input (procedure called)
STCBPANR	A4	Input (number of parameters)
Error Information		
STCBERNR	A5	Output (Natural error number)
STCBSTAT	A1	Unused (Natural error status)
STCBLIB	A8	Unused (Natural error library)
STCBPRG	A8	Unused (Natural error program)
STCBLVL	A1	Unused (Natural error level)
STCBOTP	A1	Unused (error object type)
STCBEDYL	A2	Output (error text length)
STCBEDYT	A88	Output (error text)
	A100	Reserved for future use
Parameter Information		
STCBPADE	A variable	Input. See also *PARAMETER DESCRIPTION (STCBPADE)* below.

PARAMETER DESCRIPTION (STCBPADE)

PARAMETER DESCRIPTION contains a description for each parameter passed to the Natural stored procedure consisting of parameter type, format specification and length. Parameter type is the AD attribute of the Natural CALLNAT statement as described in the Natural *Statements* documentation.

Each parameter has the following format description element in the STCBPADE string

atl,*p*[,*d1*]....

where

- *a* is an attribute mark which specifies the parameter type:

Mark	Type	Equivalent AD Attribute	Equivalent DB2 Clause
M	modifiable	AD=M	INOUT
O	non-modifiable	AD=O	IN
A	input only	AD=A	OUT

- *t* is one of the following Natural format tokens:

t	Description	*l*	*p*	*dl*	Example
A	Alphanumeric	1-253	0	1-32767 or -	A30,0 or A30,0,10
N	Numeric unpacked	1-29	0-7	-	N10,3
P	Packed numeric	1-29	0-7	-	P13,4
I	Integer	2 or 4	0	-	I2,0
F	Floating point		0	-	I4,0
B	Binary		0	-	B23,0
D	Date	6	0	-	D6
T	Time	12	0	-	T12
L	Logical (unsupported)				

- *l* is an integer denoting the length/scale of the field. For numeric and packed numeric fields, *l* denotes the total number of digits of the field that is, the sum of the digits left and right of the decimal point. The Natural format N7.3 is, for example, represented by N10.3. See also the table above.

- *p* is an integer denoting the precision of the field. It is usually 0, except for numeric and packed fields where it denotes the number of digits right of the decimal point. See also the table above.

- *d1* is also an integer denoting the occurrences of the alphanumeric array (alphanumeric only). See also the table above.

This descriptive/control parameter is invisible to the calling Natural program and to the called Natural stored procedure, but it has to be defined in the parameter definition of the stored procedure row with the CREATE PROCEDURE statement and the DB2 PARAMETER STYLE GENERAL/PARAMETER STYLE GENERAL WITH NULL.

The following table shows the number of parameters which have to be defined with the CREATE PROCEDURE statement for a Natural stored procedure defined with PARAMETER STYLE GENERAL depending on the number of user parameters and whether the client (that is, the caller of a stored procedure for DB2) and the server (that is, the stored procedure for DB2) is written in Natural or in

another standard programming host language. *n* denotes the number of user parameters.

Client\Server	Natural	not Natural
Natural	*n* + 1	*n* (CALLMODE=NONE)
non-Natural	*n* + 1	*n*

DB2SQL

Note:
PARAMETER DB2SQL applies to Natural stored procedures and Natural UDFs.

A Natural stored procedure or Natural user defined function (UDF) with PARAMETER STYLE DB2SQL first receives the user parameters specified and then the parameters listed below, under *Additional Parameters Passed*. For a Natural UDF, the input parameters are passed before the output parameters.

Additional Parameters Passed:

- A NULL indicator for each user parameter of the CALL statement,
- the SQLSTATE to be returned to DB2,
- the qualified name of the Natural stored procedure or UDF,
- the specific name of the Natural stored procedure or UDF,
- the SQL DIAGNOSE field with a diagnostic string to be returned to DB2.

The SQLSTATE, the qualified name, the specific name and the DIAGNOSE field are defined in the Natural parameter data area (PDA) DB2SQL_P which is supplied in the Natural system library SYSDB2.

If the optional feature SCRATCHPAD *nnn* is specified additionally in the CREATE FUNCTION statement for the Natural UDF, the SCRATCHPAD storage parameter is passed to the Natural UDF.

Use the following definitions:

```
01 SCRATCHPAD A(4+nnn)
01 REDEFINE SCRATCHPAD
02 SCRATCHPAD_LENGTH(I4)
02 ...
```

Redefine the SCRATCHPAD in the Natural UDF according to your requirements.

The first four bytes of the SCRATCHPAD area contain an integer length field. Before initially invoking the Natural UDF with an SQL statement, DB2 resets the SCRATCHPAD area to x'00' and sets the size *nnn* specified for the SCRATCHPAD into the first four bytes as an integer value.

Thereafter, DB2 does not reinitialize the SCRATCHPAD between the invocations of the Natural UDF for the invoking SQL statement. Instead, after returning from the Natural UDF, the contents of the SCRATCHPAD is preserved and restored at the next invocation of the Natural UDF.

Below is information on:

- Parameter CALL TYPE
- Parameter DBINFO
- Determining Library, Subprogram and Parameter Formats
- Invoking a Natural Stored Procedure
- Error Handling
- Lifetime of Natural Session
- Example of DB2SQL - Natural Stored Procedure
- Example of DB2SQL - Natural UDF

Parameter CALL TYPE

Note:
This parameter is optional and only applies to Natural UDFs.

The `CALL TYPE` parameter is passed if the `FINAL CALL` option is specified for a Natural scalar UDF, or if the Natural UDF is a table UDF. The `CALL TYPE` parameter is an integer indicating the type of call DB2 performs on the Natural UDF. See the *DB2 SQL GUIDE* for details on the parameter values provided in the `CALL_TYPE` parameter.

Parameter DBINFO

This parameter is optional.

If the option `DBINFO` is used, the `DBINFO` structure is passed to the Natural stored procedure or UDF. The `DBINFO` structure is described in the Natural PDA `DBINFO_P` supplied in the Natural system library `SYSDB2`.

Determining Library, Subprogram and Parameter Formats

The Natural for DB2 server stub determines the subprogram and the library from the qualified and specific name of the Natural stored procedure or UDF. The SCHEMA name is used as library name, and the procedure or function name is used as subprogram name.

The `ROUTINEN` subprogram is supplied in the Natural system library `SYSDB2`. This subprogram is used to access the DB2 catalog to determine the formats of the user parameters defined for the Natural stored procedure or UDF. After the formats have been determined, they are stored in the Natural buffer pool. During subsequent invocations of the Natural stored procedure, the formats are then retrieved from the Natural buffer pool. This requires that at least `READS SQL DATA` is specified for Natural stored procedures or UDFs with `PARAMETER STYLE DB2SQL`.

The `ROUTINEN` subprogram is generated statically. The DBRM of `ROUTINEN` is bound as package in the `COLLECTION SAGNDBROUTINENPACK`. Before starting to access the DB2 catalog, the subprogram will save the `CURRENT PACKAGESET` and set `SAGNDBROUTINENPACK` to `CURRENT PACKAGESET`. After processing, the `ROUTINEN` subprogram will restore the `CURRENT PACKAGESET` saved.

Invoking a Natural Stored Procedure

If the caller of the Natural stored procedure with PARAMETER STYLE DB2SQL is a Natural program, the caller must use the Natural SQL CALLDBPROC statement with the specification CALLMODE=NATURAL, which is the default.

Error Handling

If a Natural runtime error occurs during the execution of a Natural stored procedure or UDF with PARAMETER STYLE DB2SQL, SQLSTATE is set to 38N99 and the diagnostic string contains the text of the Natural error message.

If an error occurs in the Natural for DB2 server stub during the execution of the Natural stored procedure or UDF with PARAMETER STYLE DB2SQL, the SQLSTATE is set to 38S99 and the diagnostic string contains the text of the error message.

If the application wants to raise an error condition during the execution of a Natural stored procedure or UDF, the SQLSTATE parameter must be set to a value other than '00000'. See the *DB2 SQL Guide* for specifications of user errors in the SQLSTATE parameter.

Additionally, a text describing the errors can be placed in the DIAGNOSE parameter.

If a Natural table UDF wants to signal to DB2 that it has found no row to return, '02000' must be returned in the SQLSTATE parameter.

Lifetime of Natural Session

For a Natural UDF that contains the attributes DISALLOW PARALLEL and FINAL CALL, the Natural for DB2 server stub retains the Natural session allocated earlier. This Natural session will then be reused by all subsequent UDF invocations until Natural encounters the final call.

Example of DB2SQL - Natural Stored Procedure

In a Natural stored procedure, define the parameters as shown in the example program below:

```
DEFINE DATA PARAMETER
01 P1 ...
01 P2 ...
...
...
01 PN ...
01 N1 (I2)
01 N2 (I2)
...
...
01 N
n (I2)
PARAMETER USING DB2SQL_P
[ PARAMETER USING DBINFO_P ]   /*  only if DBINFO is defined
LOCAL
...
...
END-DEFINE
```

Example of DB2SQL - Natural UDF

In a Natural UDF, define the parameters as shown in the example program below:

```
DEFINE DATA PARAMETER
01 PI1 ...  /* first input parameter
01 PI2 ...
...
...
01 PIn ...  /*  last input parameter
01 RS1...  /* first result parameter
...
...
01 RSn ...  /* last result parameter
01 N_PI1 (I2)  /* first NULL indicator
01 N_PI2 (I2)
...
...
01 N_Pin (I2)
01 N_RS1 (I2)
...
...
01 N_RSn (I2) /* last NULL indicator
PARAMETER USING DB2SQL_P /* function, specific, sqlstate, diagnose
PARAMETER
01 SCRATCHPAD A(4+nnn)  /* only if SCRATCHPAD nnn is specified
  01 REDEFINES SCRATCHPAD
02 SCRATCHPAD_LENGTH (I4)
02 ...
01 CALL_TYPE (I4) /* --- only if FINAL CALL is specified or table UDF


PARAMETER USING DBINFO_P   /* ---- only if DBINFO is specified
LOCAL
...
...
END-DEFINE
```

Writing a Natural Stored Procedure

This section provides a general guideline of how to write a Natural Stored Procedure and what to consider when writing it.

To write a Natural stored procedure

1. Determine the format and attributes of the parameters that are passed between the caller and the stored procedure. Consider creating a Natural parameter data area (PDA). Stored procedures do not support data groups and redefinition within their parameters.
2. Determine the `PARAMETER STYLE` of the stored procedure: `GENERAL`, `GENERAL WITH NULL` or `DB2SQL`.
   - If you use `GENERAL WITH NULL`, append the parameters to the Natural stored procedure by defining a `NULL` indicator array that contains a `NULL` indicator (I2) for each other parameter.

- If you use DB2SQL, append the parameters of the Natural stored procedure by defining NULL indicators (one for each parameter), include the PDA DB2SQL_P and the PDA DBINFO_P (only with DBINFO specified), if desired. See also the relevant DB2 literature by IBM.

3. Decide which and how many result sets the stored procedure will return to the caller.

4. Code your stored procedure as a Natural subprogram.

   - **Returning result sets**
     To return result sets, code a Natural SQL SELECT statement with the WITH RETURN option.

     To return the whole result set, code an ESCAPE BOTTOM statement immediately after the SELECT statement.

     To return part of the result set code, an IF *COUNTER = 1 ESCAPE TOP END-IF immediately following the SELECT statement. This ensures that you do not process the first empty row that is returned by the SELECT WITH RETURN statement. To stop row processing, execute an ESCAPE BOTTOM statement.

     If you do not leave the processing loop initiated by the SELECT WITH RETURN via ESCAPE BOTTOM, the result set created is closed and nothing is returned to the caller.

   - **Attention when accessing other databases**
     You can access other databases (for instance Adabas) within a Natural stored procedure. However, keep in mind that your access to other databases is synchronized neither with the updates done by the caller of the stored procedure, nor with the updates done against DB2 within the stored procedure.

   - **Natural for DB2 handling of COMMIT and ROLLBACK statements**
     DB2 does not allow a stored procedure to issue Natural SQL COMMIT or ROLLBACK statements (the execution of those statements puts the caller into a must-rollback state). Therefore, the Natural for DB2 runtime handles those statements as follows when they are issued from a stored procedure:

     COMMIT against DB2 will be skipped. This allows the stored procedure to commit Adabas updates without getting a must-rollback state from DB2.

     ROLLBACK against DB2 will be skipped if it is created by Natural itself.

     ROLLBACK against DB2 will be executed if it is user-programmed. Thus, after a Natural error, the caller receives the Natural error information and not the unqualified must-rollback state. Additionally, this function ensures that, if the user program backs out the transaction, every database transaction of the stored procedure is backed out.

5. **For DB2 UDB:** Issue a CREATE PROCEDURE statement that defines your stored procedure, for example:

```
CREATE PROCEDURE <PROCEDURE>
  (INOUT  STCB       VARCHAR(274+13*N),
   INOUT  <PARM1>    <FORMAT>,
   INOUT  <PARM2>    <FORMAT>,
   INOUT  <PARM3>    <FORMAT>
   .
  )
  DYNAMIC RESULT SET <RESULT_SETS>
  EXTERNAL NAME <LOADMOD>
  LANGUAGE ASSEMBLE
  PROGRAM TYPE <PGM_TYPE>
  PARAMETER STYLE GENERAL <WITH NULLS depending on LINKAGE>;
```

The data specified in angle brackets (< >) correspond to the data listed in the table above, `PARM1` - `PARM3` and `FORMAT` depend on the call parameter list of the stored procedure. See also *Example Stored Procedure NDBPURGN*, Member `CR6PURGN`.

6. Code your Natural program invoking the stored procedure via the Natural SQL `CALLDBPROC` statement.

   Code the parameters in the `CALLDBPRO`C statement in the same sequence as they are specified in the stored procedure. Define the parameters in the calling program in a format that is compatible with the format defined in the stored procedure.

   If you use result sets, specify a `RESULT SETS` clause in the `CALLDBPROC` statement followed by a number of result set locator variables of `FORMAT (I4)`. The number of result set locator variables should be the same as the number or result sets created by the stored procedure. If you specify fewer than are created, some result sets are lost. If you specify more than are created, the remaining result set locator variables are lost. The sequence of locator variables corresponds to the sequence in which the result sets are created by the stored procedure.

   Keep in mind that the fields into which the result set rows are read have to correspond to the fields used in the `SELECT WITH RETURN` statement that created the result set.

Writing a Natural UDF

This section provides a general guideline of how to write a Natural user defined function (UDF) and what to consider when writing it.

See also the section *Writing a Natural Stored Procedure*.

To write a Natural UDF

1. Determine the format and attributes of the parameters, which are passed between the caller and the stored procedure.

2. Create a Natural parameter data area (PDA).

3. Append the parameter definitions of the Natural UDF by defining `NULL` indicators (one for each parameter) and include the PDA `DB2SQL_P`.

4. If required, code a `SCRATCHPAD` area in the parameter list.

5. If required, code a call-type parameter. If you have specified DBINFO, include the PDA DBINFO_P. See also the relevant DB2 literature by IBM.

6. Code your UDF as a Natural subprogram and consider the following:

   - **Attention when accessing other databases**
     You can access other databases (for example, Adabas) within a Natural UDF. However, keep in mind that your access to other databases is synchronized neither with the updates done by the caller of the stored procedure, nor with the updates done against DB2 within the stored procedure.

   - **Natural for DB2 handling of COMMIT and ROLLBACK statements**
     DB2 does not allow a stored procedure to issue COMMIT or ROLLBACK statements; the execution of these statements results in a must-rollback state. If a Natural stored procedure issues a COMMIT or ROLLBACK, the Natural for DB2 runtime processes these statements as follows:

     COMMIT against DB2 is skipped. This allows the stored procedure to commit Adabas updates without entering a must-rollback state by DB2.

     ROLLBACK against DB2 is skipped if it is implicitly issued by the Natural runtime.

     ROLLBACK against DB2 is executed if it is user-programmed. Thus, after a Natural error, the caller receives a corresponding Natural error message text, but does not enter an unqualified must-rollback state. Additionally, this reaction ensures that every database transaction the stored procedure performs is backed out if the user program backs out the transaction.

7. Issue a CREATE FUNCTION statement that defines your UDF, for example:

```
CREATE FUNCTION <FUNCTION>
  ([PARM1]    <FORMAT>,
   [PARM2]    <FORMAT>,
   [PARM3]    <FORMAT>

  )
  RETURNS <FORMAT>

  EXTERNAL NAME <LOADMOD>
  LANGUAGE ASSEMBLE
  PROGRAM TYPE <PGM TYPE>
  PARAMETER STYLE DB2SQL
.
.
.;
```

   In the example above, the variable data are enclosed in angle brackets (< >) and refer to the keywords preceding the brackets. Specify a valid value, for example:

   LOADMOD denotes the Natural for DB2 server stub module, for example, NDB*vr*SRV, where *vr* stands for the Natural version number. PARM1 - PARM3 and FORMAT relate to the call parameter list of the UDF. See also the *Example Natural User Defined Function*.

8. Code a Natural program containing SQL statements that invoke the UDF.

Specify the parameters of the Natural UDF invocation in the same sequence as specified in the Natural UDF definition. The format of the parameters in the calling program must be compatible with the format defined in the Natural UDF.

Example Stored Procedure

This section describes the example stored procedure NDBPURGN, a Natural subprogram which purges Natural objects from the buffer pool used by the Natural stored procedures server.

The following topics are covered below:

- Members of NDBPURGN
- Defining the Stored Procedure NDBPURGN

Members of NDBPURGN

The example stored procedure NDBPURGN comprises the following text members which are stored in the Natural system library SYSDB2:

Member	Explanation
CR6PURGN	Input member for SYSDB2 ISQL. Contains SQL statements used to declare NDBPURGN in DB2.
NDBPURGP	The client (Natural) program which ● Requests the name of the program to be purged and the library where it resides, ● Invokes the stored procedure NDBPURGN and ● Reports the outcome of the request.
NDBPURGN	The stored procedure which purges objects from the buffer pool. NDBPURGN invokes the application programming interface USR0340N supplied in the Natural system library SYSEXT. Therefore, USR0340N must be available in the library defined as the steplib for the execution environment.

Defining the Stored Procedure NDBPURGN

To define the example stored procedure NDBPURGN

1. Define the stored procedure in the DB2 catalog by using the SQL statements provided as text members CR5PURGN (for DB2 Version 5) and CR6PURGN (for DB2 Version 6).
2. Specify the name of the Natural stored procedure stub (here: NDB*vr*SRV, where *vr* stands for the Natural version number) as LOADMOD (V5) or EXTERNAL NAME (V6). The Natural stored procedure stub is generated during the installation by assembling the NDBSTUB macro.

3. As the first parameter, pass the internal Natural parameter STCB to the stored procedure. The STCB parameter is a VARCHAR field which contains information required to invoke the stored procedure in Natural:

   - The program name of the stored procedure and the library where it resides,
   - The description of the parameters passed to the stored procedure and
   - The error message created by Natural if the stored procedure fails during the execution.

   The STCB parameter is generated automatically by the CALLMODE=NATURAL clause of the Natural SQL CALLDBPROC statement and is removed from the parameters passed to the Natural stored procedure by the server stub. Thus, STCB is invisible to the caller and the stored procedure. However, if a non-Natural client intends to call a Natural stored procedure, the client has to pass the STCB parameter explicitly. See also *Stored Procedure Control Block* below.

Stored Procedure Control Block (STCB)

Below is the Stored Procedure Control Block (STBC) generated by the CALLMODE=NATURAL clause as generated by the stored procedure NDBPURGN *before* and *after* execution. Changed values are emphasized in boldface:

STCB before Execution:

```
004C82   0132F0F3  F0F6E2E3  C3C2F3F1  F040C8C7  *..0306STCB310 HG*    11097D42
004C92   D2404040  4040C8C7  D2404040  4040D5C4  *K     SAG     ND*    11097D52
004CA2   C2D7E4D9  C7D74040  40404040  4040F0F5  *BPURGP        05*    11097D62
004CB2   F7F0D5C4  C2D7E4D9  C7D5F0F0  F0F6F0F9  *70NDBPURGN000609*    11097D72
004CC2   F9F9F940  40404040  40404040  40404040  *999             *    11097D82
004CD2   40404040  40404040  40404040  40404040  *                *    11097D92
004CE2   40404040  40404040  40404040  40404040  *                *    11097DA2
004CF2   40404040  40404040  40404040  40404040  *                *    11097DB2
004D02   40404040  40404040  40404040  40404040  *                *    11097DC2
004D12   40404040  40404040  40404040  40404040  *                *    11097DD2
004D22   40404040  40404040  40404040  40404040  *                *    11097DE2
004D32   40404040  40404040  40404040  40404040  *                *    11097DF2
004D42   40404040  40404040  40404040  40404040  *                *    11097E02
004D52   40404040  40404040  40404040  40404040  *                *    11097E12
004D62   40404040  40404040  40404040  40404040  *                *    11097E22
004D72   40404040  40404040  40404040  40404040  *                *    11097E32
004D82   40404040  40404040  40404040  40404040  *                *    11097E42
004D92   40404040  D4C1F86B  F0D4C1F4  F06BF0D4  *    MA8,0MA40,0M*    11097E52
004DA2   C2F26BF0  D4C2F26B  F0D4C9F2  6BF0D4C9  *I2,0MI2,0MI2,0MI*    11097E62
004DB2   F26BF04B                               *2,0.            *    11097E72
```

STCB after Execution:

```
004C82   0132F0F3  F0F6E2E3  C3C2F3F1  F040C8C7  *..0306STCB310 HG*    11097D42
004C92   D2404040  4040C8C7  D2404040  4040D5C4  *K     SAG     ND*    11097D52
004CA2   C2D7E4D9  C7D74040  40404040  4040F0F5  *BPURGP        05*    11097D62
004CB2   F7F0D5C4  C2D7E4D9  C7D5F0F0  F0F6F0F0  *70NDBPURGN000600*    11097D72
004CC2   F0F0F040  40404040  40404040  40404040  *000             *    11097D82
004CD2   40404040  40404040  40404040  40404040  *                *    11097D92
004CE2   40404040  40404040  40404040  40404040  *                *    11097DA2
004CF2   40404040  40404040  40404040  40404040  *                *    11097DB2
004D02   40404040  40404040  40404040  40404040  *                *    11097DC2
004D12   40404040  40404040  40404040  40404040  *                *    11097DD2
004D22   40404040  40404040  40404040  40404040  *                *    11097DE2
```

```
004D32   40404040  40404040  40404040  40404040  *               *    11097DF2
004D42   40404040  40404040  40404040  40404040  *               *    11097E02
004D52   40404040  40404040  40404040  40404040  *               *    11097E12
004D62   40404040  40404040  40404040  40404040  *               *    11097E22
004D72   40404040  40404040  40404040  40404040  *               *    11097E32
004D82   40404040  40404040  40404040  40404040  *               *    11097E42
004D92   40404040  D4C1F86B  F0D4C1F4  F06BF0D4  *    MA8,0MA40,0M*    11097E52
004DA2   C2F26BF0  D4C2F26B  F0D4C9F2  6BF0D4C9  *I2,0MI2,0MI2,0MI*    11097E62
004DB2   F26BF04B                               *2,0.            *    11097E72
```

Example Natural User Defined Function

This section describes the example user defined function (UDF) NAT.DEM2UDFN, a Natural subprogram used to calculate the product of two numbers.

The example UDF NAT.DEM2UDF comprises the following members that are supplied in the Natural system library SYSDB2:

Member	Explanation
DEM2CUDF	Contains SQL statements used to create DEM2UDFN (see below).
DEM2UDFP	The client (Natural) program that • Fetches rows from the UDF NAT.DEMO table, • invokes the NAT.DEM2UDFN (see below) in the WHERE clause, and • Displays the rows fetched.
DEM2UDFN	The UDF that builds the product of two numbers. DEM2UDFN has to be copied to the Natural library NAT on the Natural sytem file FUSER in the executing environment.