May 1998

In this issue

3 DB2 and the year 2000
8 Year 2000 test experience
9 Dynamic plan switching for development projects
25 Partitioned tablespaces page number calculator
31 REXX extensions for DB2 – part 4
48 DB2 news

© Xephon plc 1998
Contributions
Articles published in DB2 Update are paid for at the rate of £170 ($250) per 1000 words and £90 ($140) per 100 lines of code for original material. To find out more about contributing an article, without any obligation, please contact us at any of the addresses above and we will send you a copy of our Notes for Contributors.

DB2 Update on-line
Code from DB2 Update can be downloaded from our Web site at http://www.xephon.com; you will need the user-id shown on your address label.

Editor
Trevor Eddolls

Disclaimer
Readers are cautioned that, although the information in this journal is presented in good faith, neither Xephon nor the organizations or individuals that supplied information in this journal give any warranty or make any representations as to the accuracy of the material it contains. Neither Xephon nor the contributing organizations or individuals accept any liability of any kind howsoever arising out of the use of such material. Readers should satisfy themselves as to the correctness and relevance to their circumstances of all advice, information, code, JCL, and other contents of this journal before making any use of it.

Subscriptions and back-issues
A year’s subscription to DB2 Update, comprising twelve monthly issues, costs £245.00 in the UK; $365.00 in the USA and Canada; £251.00 in Europe; £257.00 in Australasia and Japan; and £255.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the January 1994 issue, are available separately to subscribers for £21.00 ($31.00) each including postage.
DB2 and the year 2000

Ensuring Year 2000 compliance is a pervasive issue throughout the IT industry. Yet many people assume that DB2 is immune from the Year 2000 problem because it provides date data types, numerous date formats, and date logic. As welcome as this support is, it can only be used once the offending date columns are identified. Many DB2 applications were implemented using character data types to store date values. This occurred for many reasons, foremost of which were ignorance (a lack of knowledge about date data types), fear (a lack of understanding about date data types), a desire to store a non-date and non-null default value, or the fact that application was implemented prior to DB2 Version 1.3 (the first version of DB2 to provide date data types).

In this article I address the Year 2000 crisis as it impacts DB2 from various perspectives – including administration and development, data administration, staffing, performance, and tools.

TWO APPROACHES TO THE YEAR 2000 PROBLEM

There are two approaches you can take to address the Year 2000 problem in DB2 tables. Firstly, you can convert the affected columns to DATE type and change the application code appropriately. This is known as the ‘field expansion approach’. This involves making changes to both the DB2 tables and the application logic that accesses the tables.

Alternatively, you can use the ‘application logic approach’. Under this approach you will update each piece of application logic that accesses a date so that it can handle the date without the century. This involves transformation logic to understand whether the date is pre- or post-year 2000. This type of code typically takes a form similar to the following pseudo-code:

```plaintext
IF YY > 50 THEN
   CC = 19
ELSE
   CC = 20;
```
This code assigns 20 to the century component of the year if the date is less than 50, 19 if it is greater than 50. Of course, the value 50 is somewhat arbitrary and will vary based on the type of processing. Additionally, the actual code can be much more complex because of processing issues such as whether valid dates can fall under both the current and future century (e.g., the years 1925 and 2025 are both valid for the application in question).

For either approach, the following steps need to be taken.

**Identify columns that need to be addressed**

You must compile a list of the columns that need to be examined. To compile such a list, a good starting point is to look for columns having the CHAR data type with the string ‘DATE’ (or something similar) in its name.

Consider implementing standards for database dates so that all dates in all tables follow the standard – a failure to do so can cause performance degradation. For example, consider the implications when date columns are joined or used in SQL predicates. Comparing a date data type to a character – even if each column contains the same date – will either not match or cause performance to slow as one data type is converted so the match can occur.

Additionally, CHECK constraints on date columns (whether actually of DATE data type or not) should be re-evaluated to ensure that the appropriate checks will be performed when the century changes.

**Identify code that references the column**

Once the columns have been identified, you must find all the application code that references the columns. Start by searching for occurrences of the column names in your source code libraries and the DB2 Catalog. However, this may not identify all of the application code using date columns. You could miss dynamic SQL and statements that access the column using a view. Ensure that you have a dependable method of locating all affected code. Automated tools exist that can help you to identify COBOL and SQL code that reference date columns.
Set up a test environment

Consider setting up a test environment exclusively for Year 2000 testing. This enables you to isolate Year 2000 testing from other development and testing efforts. You might create a duplicate of an existing test environment or create a scaled-down version of a production system. You also need some way to generate test data, with dates in the future, for your integrated system testing.

DBAs will need to populate test beds for Year 2000 testing. Products that automatically maintain referential integrity of test beds generated from production databases may become cost-justifiable within the scope of Year 2000 compliance testing.

Make the changes

For the application logic approach, programmers must modify the source code as needed. For the field expansion approach, you must alter the columns to DATE types. This requires dropping and recreating the tables, including unloading data, converting the dates, and reloading the tables with the converted data. Before dropping the tables, make sure to identify any dependencies on the tables, such as referential integrity, indexes, views, aliases, synonyms, and authorizations, because these also will be dropped. When you recreate the tables, you must also recreate the tables’ dependencies.

When changes are made to application programs that expand date columns in RDBMSs, DBAs need to be involved in the process of moving the changes to production because of possible referential integrity changes to primary key (PK)-foreign key (FK) relationships. If the PK is changed, the FK must be changed at the same time, and vice versa. Failure to do so will result in poor performance at best or data integrity violations at worst.

Test the applications

Planned changes to systems, programs, utilities, and databases must be tested to ensure that coded changes are working as designed, are producing the desired results, and production processing failures are
prevented. Therefore, all applications that are affected must be thoroughly tested using dates up to and beyond the year 2000. In addition, you should include some critical test dates as part of your test plan. For instance, including dates such as ‘28-Feb-2000’, ‘29-Feb-2000’, and ‘1-Mar-2000’ will help to verify correct leap year processing. The testing phase is crucial because testing will probably account for more than half the time spent on Year 2000 compliance. Although testing will consume the majority of your Year 2000 project life-cycle, if it is done correctly, it will keep risk exposure to a minimum.

Processing failures can be minimized by performing thorough testing of all system components to simulate real-world processing conditions wherever possible. Various tools are available to assist with Year 2000 testing including: date simulators that roll the system clock forward and terminal capture and replay tools.

**Implement the changes**

For the field expansion method, once your testing is done, you need to migrate the changed tables to your production systems. For all code changes, a change and configuration management plan is needed. The database and code changes should be synchronized to occur simultaneously. If one is rolled back from production, a process needs to be in place to roll the other back too.

These are the basic black and white issues of ensuring Year 2000 compliance. However, let’s examine some of the grey areas surrounding the Year 2000 and its impact on IT.

**FUTURE PERFORMANCE IMPLICATIONS**

Sometimes organizations take shortcuts to avoid Year 2000 application problems. Consider a credit card application that does not support four-digit dates. To ‘get around’ this problem the issuer sets the expiration date for all new and expiring credit cards to be no later than December 1999. The thinking goes: ‘when the applications are fixed we can re-assign all of the credit cards to future dates’. However, this is a ‘fix’ that can cause more problems than it actually fixes.
Assume that the application is not fixed until late 1998. At that point in time, the issuer can begin to issue credit cards with a post-2000 date. But what happens in 1999 when all of the credit cards that were issued over a multiple year period begin to expire? Are the systems ready for the increased transaction workload that would normally be spread across multiple years? If not, the organization must plan on issuing new cards much earlier – before they begin to expire. In this case, the company opens itself up to questions from its customers regarding why they must start using a new credit card before the old one expires. What a nightmare!

A DATA ADMINISTRATION OPPORTUNITY
As part of the Year 2000 process, automated tools may be used to scan entire production application portfolios. These scans are used to identify and correct date problems within the applications. Consider using this opportunity, when the entire program library is being scanned anyway, to document and catalogue the metadata for these applications in a repository or data dictionary tool. A project to capture such a massive amount of information may be impossible to cost-justify outside the scope of the Year 2000 project. As such, do not squander the opportunity to proactively catalogue and document your metadata.

STAFFING ISSUES
Many enterprises have the absolute minimum number of DBAs assigned to support DB2. This means that limited administration and technical support is available for Year 2000 projects because the DB2 DBAs are scrambling to support the new development that is occurring. This causes a staffing shortage that will be difficult to alleviate without additional headcount being allocated for Year 2000 support. This may be a good opportunity to get additional database administration support because the purse-strings are often easier to open for Year 2000 projects than for any other type of project. Let’s face it – there is a hard and fast deadline that can’t be missed!
SYNOPSIS

The Year 2000 problem is pervasive and will consume many development cycles and dollars. Be prepared by automating the process as much as possible using Year 2000 and database administration tools to minimize risk and increase the speed of conversion.

Craig S Mullins (USA) © Craig S Mullins 1998

Year 2000 test experience

We would be very interested to hear from our readers what their experience is with Year 2000 testing – especially any valuable lessons they learned, and any hints and tips that would benefit other DB2 users.

DB2 Update is also looking for REXX EXECs, macros, program code, etc, that experienced DB2 users have written to make their life easier. We will publish them (after vetting by our expert panel) and send you a cheque when the article is published. Articles can be of any length and can be sent to Trevor Eddolls, editor of DB2 Update, at any of the addresses shown on page 2, or e-mailed to our Compuserve address, also shown on page 2. We pay $250 (£170) per 1000 words and $140 (£90) per 100 lines of code published, when Xephon is given copyright.

Why not call now for a free copy of our Notes for contributors?
Dynamic plan switching for development projects

INTRODUCTION

Development projects, especially large ones that involve multiple teams, can often benefit from the ability to execute the same set of CICS transactions against multiple DB2 databases. I have been involved in several new development projects where multiple teams were assigned different functional areas of the application and each team required their own database so they could tailor the data to fit their specific needs. Often, each team did not need their own CICS region, therefore a means to allow these teams to share one set of CICS programs, yet access multiple different databases, is necessary.

Dynamic plan selection is the mechanism that will provide this functionality. In order to use the function shown here, an application would have to support dynamic plan allocation. This feature is specified in the CICS RCT entries with parameters PLNEXIT=YES and PLNPME=exit-program-name. See below for an example of RCT entries that specify dynamic plan selection. The DB2 Administration Guide covers this topic in detail. I will not discuss the details of dynamic plan selection here; I will assume that anyone interested in implementing a function like the one shown here will already be familiar with the details of this feature.

RCT EXAMPLE

**********************************************************************
* POOL
**********************************************************************
* DSNCRCT TYPE=POOL,
  AUTH=(USERID,*,*),
  DPMODE=LOW,
  ROLBE=YES,
  PLNEXIT=YES,
  PLNPME=RASXUEXT,
  THRDM=25,THRDA=20,THRDS=0,
  TWAIT=YES
**********************************************************************

THE SWPL FUNCTION

I have created a CICS transaction that developers can invoke to select which database they will access when executing the CICS application programs. The transaction name is SWPL (short for switch plans), and has come to be known by developers as the ‘swape’ function.

There are four components that make up this function. Firstly, the VSAM file is defined to the CICS region, and is used to store CICS login-ids and plan prefix characters. The example below is the IDCAMS definition of the RASSPLSW dataset. It must also be defined in the FCT for the CICS region that it is to be used in. Our implementation of dynamic plan selection uses the first character of the plan name to differentiate between databases. For example, if the program name is PRSXX123, plan BRSXX123 will be bound to the ‘B’ database, while plan CRSXX123 is bound to the ‘C’ database. Because we replace the first letter of the program to identify the plan names, we store this character in the VSAM file along with the user’s login-id.

IDCAMS DEFINE FOR RASSPLSW

```assembly
IDCAMS DEFINE FOR RASSPLSW
//********************************************************************
//*  IDCAMS                                                          *
//********************************************************************
//STEPØ1Ø  EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*    
//SYSIN    DD *             
DEFINE CLUSTER                             -
TXID=(MENU,DMØØ,DMØ1,DMØ2,DMØ4,DMØ5,DM21)
```
The second component is the plan exit program. DB2 ships a sample exit program called DSNCUEXT. It uses the name of the DBRM containing the first SQL statement executed for the plan name. The example below shows the plan exit program used for the SWPL function. Once invoked, this program reads the RASSPLSW VSAM file, looking for a record with a key matching the current user’s login-id. If a record is found, the plan prefix character stored on that record replaces the first character in the program name and invokes the plan with this name. If a record is not found for the user, a default plan prefix character is used.

RASXUEXT PROGRAM

TITLE 'RASXUEXT DB2 CICS/VS ATTACH - RAS DYNAMIC PLAN EXIT'
***********************************************************************
*                                                                      *
* MODULE NAME=  RASXUEXT                                              *
*                                                                      *
* DESCRIPTIVE NAME=DYNAMIC PLAN EXIT                                  *
*                                                                      *
* THIS PROGRAM CHECKS VSAM FILE RASSPLSW FOR A RECORD WITH A KEY      *
* MATCHING THE USER'S LOGIN-ID. IF A RECORD IS FOUND, THE CHARACTER   *
* IN THAT RECORD IS USED AS THE FIRST CHARACTER OF THE PLAN NAME      *
* THAT WILL BE INVOKED. IF A RECORD IS NOT FOUND, THE LETTER 'P'      *
* WILL BE USED FOR THE FIRST CHARACTER OF THE PLAN NAME.              *
*                                                                      *
***********************************************************************
EJECT
***********************************************************************
*        REGISTER EQUATES                                             *
***********************************************************************
DFHREGS
EJECT
***********************************************************************
*        DYNAMIC STORAGE                                              *
***********************************************************************
CICS COBOL and BMS code for the program and map used by the SWPL transaction that allow the developers to change their default plan prefix are shown below.

RASSSWP BMS CODE

RASSWP  DFHMSD MODE=INOUT,LANG=COBOL,TIOAPFX=YES,TYPE=MAP,*
        CTRL=(FREEKB,FRSET),STORAGE=AUTO
RASSWP  DFHMDI SIZE=(24,Ø0),*
        COLUMN=Ø001,*
LINE=0001
L01C01 DFHMDF POS=(1,10),
          LENGTH=60,
          ATTRB=(ASKIP,BRT)
      DFHMDF INITIAL='RAS PLAN SWITCHING FUNCTION',
          POS=(2,23),
          LENGTH=30,
          ATTRB=(ASKIP,NORM)
      DFHMDF INITIAL='DEVELOP',
          POS=(5,35),
          LENGTH=7,
          ATTRB=(ASKIP,NORM)
      DFHMDF INITIAL='SYS-B',
          POS=(5,43),
          LENGTH=5,
          ATTRB=(ASKIP,NORM)
      DFHMDF INITIAL='SYS-C',
          POS=(5,52),
          LENGTH=5,
          ATTRB=(ASKIP,NORM)
      DFHMDF INITIAL='SYS-D',
          POS=(5,61),
          LENGTH=5,
          ATTRB=(ASKIP,NORM)
      DFHMDF INITIAL='TRAINING',
          POS=(5,70),
          LENGTH=8,
          ATTRB=(ASKIP,NORM)
      DFHMDF INITIAL='USERID',
          POS=(6,18),
          LENGTH=6,
          ATTRB=(ASKIP,NORM)
      DFHMDF INITIAL='DB',
          POS=(6,35),
          LENGTH=4,
          ATTRB=(ASKIP,NORM)
      DFHMDF INITIAL='DB',
          POS=(6,43),
          LENGTH=4,
          ATTRB=(ASKIP,NORM)
      DFHMDF INITIAL='DB',
          POS=(6,52),
          LENGTH=4,
          ATTRB=(ASKIP,NORM)
      DFHMDF INITIAL='DB',
          POS=(6,61),
          LENGTH=4,
          ATTRB=(ASKIP,NORM)
      DFHMDF INITIAL='DB',
          POS=(6,70),
          LENGTH=4,
DFHMDF POS=(08,18), LENGTH=8,
ATTRB=(PROT,ASKIP,NORM)

LØ8C18 DFHMDF POS=(8,27), LENGTH=1,
ATTRB=(ASKIP,NORM)

LØ8C36 DFHMDF POS=(08,36), LENGTH=1,
ATTRB=(UNPROT,NORM,FSET,IC)

LØ8C44 DFHMDF POS=(08,44), LENGTH=1,
ATTRB=(UNPROT,NORM,FSET)

LØ8C53 DFHMDF POS=(08,53), LENGTH=1,
ATTRB=(UNPROT,NORM,FSET)

LØ8C59 DFHMDF POS=(08,62), LENGTH=1,
ATTRB=(UNPROT,NORM,FSET)

LØ8C65 DFHMDF POS=(08,71), LENGTH=1,
ATTRB=(UNPROT,NORM,FSET)

DFHMDF INITIAL='TYPE AN ''X'' UNDER ONE OF THE TEAMS',
POS=(11,34), LENGTH=34,
ATTRB=(ASKIP,NORM)

DFHMDF INITIAL='AND PRESS ENTER TO UPDATE',
POS=(12,34), LENGTH=25,
ATTRB=(ASKIP,NORM)

DFHMDF INITIAL='PRESS CLEAR TO EXIT',
POS=(14,34), LENGTH=19,
ATTRB=(ASKIP,NORM)
DFHMSD TYPE=FINAL
END
RASSOSWP COBOL CODE

ID DIVISION.
  PROGRAM-ID. RASSOSWP.
  DATE-WRITTEN. MAR 1997.
  DATE-COMPILED.
  EJECT
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
COPY DFHAID.
  EJECT
COPY DFHBMSCA.
  EJECT

01 RASSWPD.
  05 FILLER PIC X(12).
  05 RASSWPD-ERROR-MSG-L PIC S9(4) COMP.
  05 RASSWPD-ERROR-MSG-A PIC X(01).
  05 RASSWPD-ERROR-MSG-IO PIC X(60).
  05 RASSWPD-USERID-L PIC S9(4) COMP.
  05 RASSWPD-USERID-A PIC X(01).
  05 RASSWPD-USERID-IO PIC X(08).
  05 RASSWPD-SW-1-L PIC S9(4) COMP.
  05 RASSWPD-SW-1-A PIC X(01).
  05 RASSWPD-SW-1-IO PIC X(01).
  05 RASSWPD-SW-2-L PIC S9(4) COMP.
  05 RASSWPD-SW-2-A PIC X(01).
  05 RASSWPD-SW-2-IO PIC X(01).
  05 RASSWPD-SW-3-L PIC S9(4) COMP.
  05 RASSWPD-SW-3-A PIC X(01).
  05 RASSWPD-SW-3-IO PIC X(01).
  05 RASSWPD-SW-4-L PIC S9(4) COMP.
  05 RASSWPD-SW-4-A PIC X(01).
  05 RASSWPD-SW-4-IO PIC X(01).
  05 RASSWPD-SW-5-L PIC S9(4) COMP.
  05 RASSWPD-SW-5-A PIC X(01).
  05 RASSWPD-SW-5-IO PIC X(01).
  EJECT

01 RASSPLSW-RECORD.
  05 RASSPLSW-USERID PIC X(08).
  05 RASSPLSW-PREFIX PIC X(01).
  05 FILLER PIC X(03).
  EJECT

01 MISC.
  05 W-EDIT-ERROR-SW PIC X(01) VALUE SPACE.
  05 W-SW-COUNT PIC 9(01) COMP.
  05 W-USERID PIC X(08).
  05 W-RESP PIC S9(04) COMP.
  05 W-RESP-ZONED PIC 9(02).
  EJECT

1  W-ABEND-LINE.
  05 FILLER PIC X(7) VALUE 'ERROR: '.
  05 FILLER PIC X(8) VALUE 'STATUS: '.
  05 W-ABEND-RESP PIC X(2).
  05 FILLER PIC X(2) VALUE SPACES.
  05 FILLER PIC X(6) VALUE 'FUNC: '.
  05 W-ABEND-FUNC PIC X(8).
  05 FILLER PIC X(2) VALUE SPACES.
  05 FILLER PIC X(6) VALUE 'PARA: '.
  05 W-ABEND-PARA PIC X(5).
  05 FILLER PIC X(14) VALUE SPACES.
EJECT
1  W-MESSAGES.
  05 W-MSG-1 PIC X(60) VALUE 'MUST SELECT A SYSTEM  '.
  05 W-MSG-2 PIC X(60) VALUE 'INVALID CHARACTER: NOT AN X'.
  05 W-MSG-3 PIC X(60) VALUE 'SELECT ONLY 1 SYSTEM  '.
  05 W-MSG-4 PIC X(60) VALUE 'UPDATE SUCCESSFUL  '.
EJECT
1  W-COMMAREA.
  05 W-COMMAREA-USERID PIC X(8).
LINKAGE SECTION.
1  DFHCOMMAREA.
  05 DFHCOMMAREA-USERID PIC X(8).
EJECT
PROCEDURE DIVISION.
0000-MAINLINE.
  *  PERFORM 1000-PROCESS-INPUT THRU 1000-EXIT.
  PERFORM 2000-PROCESS-OUTPUT THRU 2000-EXIT.
  *
0000-EXIT.
  EXIT.
  EJECT
1000-PROCESS-INPUT.
  *  IF EIBAID = DFHCLEAR
     PERFORM 9200-RETURN THRU 9200-EXIT.
  *
     IF EIBCALEN = ZERO
     NEXT SENTENCE
     ELSE
     GO TO 1000-EXIT.
     *
     PERFORM 8000-ASSIGN-USERID THRU 8000-EXIT.
     PERFORM 7000-READ-RASSPLSW THRU 7000-EXIT.
     PERFORM 1300-BUILD-MAP THRU 1300-EXIT.
PERFORM 6000-SET-ATTRIBUTES THRU 6000-EXIT.
PERFORM 8100-SEND-MAP THRU 8100-EXIT.
MOVE W-USERID TO W-COMMAREA-USERID.
PERFORM 9000-RETURN THRU 9000-EXIT.

* 1000-EXIT.
  EXIT.
  EJECT
1300-BUILD-MAP.
  *
  MOVE LOW-VALUES TO RASSSWPD.
  *
  MOVE W-USERID TO RASSSWPD-USERID-IO.
  *
  IF W-RESP = ZERO
    NEXT SENTENCE
  ELSE
    MOVE 'X' TO RASSSWPD-SW-1-IO
    GO TO 1300-EXIT.
  *
  IF RASSPLSW-PREFIX = 'P'
    MOVE 'X' TO RASSSWPD-SW-1-IO.
  *
  IF RASSPLSW-PREFIX = 'B'
    MOVE 'X' TO RASSSWPD-SW-2-IO.
  *
  IF RASSPLSW-PREFIX = 'C'
    MOVE 'X' TO RASSSWPD-SW-3-IO.
  *
  IF RASSPLSW-PREFIX = 'D'
    MOVE 'X' TO RASSSWPD-SW-4-IO.
  *
  IF RASSPLSW-PREFIX = 'X'
    MOVE 'X' TO RASSSWPD-SW-5-IO.
  *
1300-EXIT.
  EXIT.
  EJECT
2000-PROCESS-OUTPUT.
  *
  IF EIBCALEN > ZERO
    NEXT SENTENCE
  ELSE
    GO TO 2000-EXIT.
  *
  PERFORM 8300-RECEIVE-MAP THRU 8300-EXIT.
  PERFORM 8000-ASSIGN-USERID THRU 8000-EXIT.
  PERFORM 6000-SET-ATTRIBUTES THRU 6000-EXIT.
  PERFORM 2100-EDIT-MAP THRU 2100-EXIT.
  PERFORM 2200-UPDATE-RASSPLSW THRU 2200-EXIT.

* IF W-EDIT-ERROR-SW = 'Y'
  PERFORM 8150-SEND-MAP THRU 8150-EXIT
  PERFORM 9000-RETURN THRU 9000-EXIT
  GO TO 2000-EXIT.
*
  PERFORM 8200-SEND-MAP THRU 8200-EXIT.
  MOVE W-USERID TO W-COMMAREA-USERID.
  PERFORM 9000-RETURN THRU 9000-EXIT.
*
2000-EXIT.
  EXIT.
  EJECT
2100-EDIT-MAP.
*
  MOVE SPACES TO W-EDIT-ERROR-SW.
*
  IF ( RASSWPD-SW-1-IO = SPACES OR LOW-VALUES )
    AND ( RASSWPD-SW-2-IO = SPACES OR LOW-VALUES )
    AND ( RASSWPD-SW-3-IO = SPACES OR LOW-VALUES )
    AND ( RASSWPD-SW-4-IO = SPACES OR LOW-VALUES )
    AND ( RASSWPD-SW-5-IO = SPACES OR LOW-VALUES )
    MOVE W-MSG-1 TO RASSWPD-ERROR-MSG-IO
    MOVE 'Y' TO W-EDIT-ERROR-SW
    GO TO 2100-EXIT.
*
  IF RASSWPD-SW-1-IO > SPACES
    AND RASSWPD-SW-1-IO NOT = 'X'
    MOVE DFHBMBRY TO RASSWPD-SW-1-A
    MOVE -1 TO RASSWPD-SW-1-L
    MOVE W-MSG-2 TO RASSWPD-ERROR-MSG-IO
    MOVE 'Y' TO W-EDIT-ERROR-SW.
*
  IF RASSWPD-SW-2-IO > SPACES
    AND RASSWPD-SW-2-IO NOT = 'X'
    MOVE DFHBMBRY TO RASSWPD-SW-2-A
    MOVE -1 TO RASSWPD-SW-2-L
    MOVE W-MSG-2 TO RASSWPD-ERROR-MSG-IO
    MOVE 'Y' TO W-EDIT-ERROR-SW.
*
  IF RASSWPD-SW-3-IO > SPACES
    AND RASSWPD-SW-3-IO NOT = 'X'
    MOVE DFHBMBRY TO RASSWPD-SW-3-A
    MOVE -1 TO RASSWPD-SW-3-L
    MOVE W-MSG-2 TO RASSWPD-ERROR-MSG-IO
    MOVE 'Y' TO W-EDIT-ERROR-SW.
*
  IF RASSWPD-SW-4-IO > SPACES
    AND RASSWPD-SW-4-IO NOT = 'X'
    MOVE DFHBMBRY TO RASSWPD-SW-4-A
MOVE -1 TO RASSSWPD-SW-4-L
MOVE W-MSG-2 TO RASSSWPD-ERROR-MSG-IO
MOVE 'Y' TO W-EDIT-ERROR-SW.

* 
IF RASSSWPD-SW-5-IO > SPACES AND RASSSWPD-SW-5-IO NOT = 'X'
MOVE DFHBMBRY TO RASSSWPD-SW-5-A
MOVE -1 TO RASSSWPD-SW-5-L
MOVE W-MSG-2 TO RASSSWPD-ERROR-MSG-IO
MOVE 'Y' TO W-EDIT-ERROR-SW.

* 
IF W-EDIT-ERROR-SW = 'Y'
GO TO 21ØØ-EXIT.

* 
MOVE ZERO TO W-SW-COUNT.

* 
IF RASSSWPD-SW-1-IO = 'X'
ADD 1 TO W-SW-COUNT.

* 
IF RASSSWPD-SW-2-IO = 'X'
ADD 1 TO W-SW-COUNT.

* 
IF RASSSWPD-SW-3-IO = 'X'
ADD 1 TO W-SW-COUNT.

* 
IF RASSSWPD-SW-4-IO = 'X'
ADD 1 TO W-SW-COUNT.
IF RASSSWPD-SW-5-IO = 'X'
ADD 1 TO W-SW-COUNT.

* 
IF W-SW-COUNT > 1
MOVE W-MSG-3 TO RASSSWPD-ERROR-MSG-IO
MOVE 'Y' TO W-EDIT-ERROR-SW
GO TO 21ØØ-EXIT.

* 
21ØØ-EXIT.
EXIT.
EJECT
22ØØ-UPDATE-RASSPLSW.

* 
IF W-EDIT-ERROR-SW = 'Y'
GO TO 22ØØ-EXIT.

* 
PERFORM 7Ø5Ø-READ-RASSPLSW THRU 7Ø5Ø-EXIT.

* 
IF W-RESP = ZERO
PERFORM 221Ø-FND-PROCESS THRU 221Ø-EXIT
ELSE
PERFORM 222Ø-NOTFND-PROCESS THRU 222Ø-EXIT.
* MOVE W-MSG-4 TO RASSSWPD-ERROR-MSG-IO.
* 2200-EXIT.
  EXIT.
  EJECT
  2210-FND-PROCESS.
* IF RASSSWPD-SW-1-IO = 'X'
  PERFORM 7100-DELETE-RASSPLSW THRU 7100-EXIT
  GO TO 2210-EXIT.
* IF RASSSWPD-SW-2-IO = 'X'
  MOVE 'B' TO RASSPLSW-PREFIX
  PERFORM 7200-UPDATE-RASSPLSW THRU 7200-EXIT
  GO TO 2210-EXIT.
* IF RASSSWPD-SW-3-IO = 'X'
  MOVE 'C' TO RASSPLSW-PREFIX
  PERFORM 7200-UPDATE-RASSPLSW THRU 7200-EXIT
  GO TO 2210-EXIT.
* IF RASSSWPD-SW-4-IO = 'X'
  MOVE 'D' TO RASSPLSW-PREFIX
  PERFORM 7200-UPDATE-RASSPLSW THRU 7200-EXIT
  GO TO 2210-EXIT.
* IF RASSSWPD-SW-5-IO = 'X'
  MOVE 'X' TO RASSPLSW-PREFIX
  PERFORM 7200-UPDATE-RASSPLSW THRU 7200-EXIT
  GO TO 2210-EXIT.
* 2210-EXIT.
  EXIT.
  EJECT
  2220-NOTFND-PROCESS.
* IF RASSSWPD-SW-1-IO = 'X'
  GO TO 2220-EXIT.
* IF RASSSWPD-SW-2-IO = 'X'
  MOVE SPACES TO RASSPLSW-RECORD
  MOVE W-USERID TO RASSPLSW-USERID
  MOVE 'B' TO RASSPLSW-PREFIX
  PERFORM 7300-WRITE-RASSPLSW THRU 7300-EXIT
  GO TO 2220-EXIT.
* IF RASSSWPD-SW-3-IO = 'X'
  MOVE SPACES TO RASSPLSW-RECORD
  MOVE W-USERID TO RASSPLSW-USERID
  MOVE 'C' TO RASSPLSW-PREFIX
PERFORM 73ØØ-WRITE-RASSPLSW THRU 73ØØ-EXIT
GO TO 222Ø-EXIT.

* IF RASSWPD-SW-4-IO = 'X'
  MOVE SPACES TO RASSPLSW-RECORD
  MOVE W-USERID TO RASSPLSW-USERID
  MOVE 'D' TO RASSPLSW-PREFIX
  PERFORM 73ØØ-WRITE-RASSPLSW THRU 73ØØ-EXIT
  GO TO 222Ø-EXIT.

* IF RASSWPD-SW-5-IO = 'X'
  MOVE SPACES TO RASSPLSW-RECORD
  MOVE W-USERID TO RASSPLSW-USERID
  MOVE 'X' TO RASSPLSW-PREFIX
  PERFORM 73ØØ-WRITE-RASSPLSW THRU 73ØØ-EXIT
  GO TO 222Ø-EXIT.

222Ø-EXIT.
EXIT.
EJECT
6ØØØ-SET-ATTRIBUTES.

* MOVE DFHBMPRF TO RASSWPD-USERID-A.

* MOVE DFHBMFSE TO RASSWPD-SW-1-A
  RASSWPD-SW-2-A
  RASSWPD-SW-3-A
  RASSWPD-SW-4-A
  RASSWPD-SW-5-A.

6ØØØ-EXIT.
EXIT.
EJECT
7ØØØ-READ-RASSPLSW.

* EXEC CICS
  READ DATASET ('RASSPLSW')
    INTO (RASSPLSW-RECORD)
    RIDFLD (W-USERID)
    RESP (W-RESP)
END-EXEC.

* IF W-RESP = ØØ OR 13
  NEXT SENTENCE
ELSE
  MOVE W-RESP TO W-RESP-ZONED
  MOVE W-RESP-ZONED TO W-ABEND-RESP
  MOVE '7ØØØ' TO W-ABEND-PARA
  MOVE 'READ' TO W-ABEND-FUNC
  MOVE W-ABEND-LINE TO RASSWPD-ERROR-MSG-IO
  PERFORM 815Ø-SEND-MAP THRU 815Ø-EXIT
PERFORM 9000-RETURN  THRU 9000-EXIT.

* 7000-EXIT.
   EXIT.
   EJECT
7050-READ-RASSPLSW.

* EXEC CICS
   READ DATASET ('RASSPLSW')
       INTO    (RASSPLSW-RECORD)
       RIDFLD (W-USERID)
       UPDATE
       RESP   (W-RESP)
   END-EXEC.

* IF W-RESP = 00 OR 13
   NEXT SENTENCE
ELSE
   MOVE W-RESP               TO W-RESP-ZONED
   MOVE W-RESP-ZONED         TO W-ABEND-RESP
   MOVE '7050'               TO W-ABEND-para
   MOVE 'READU'              TO W-ABEND-FUNC
   MOVE W-ABEND-LINE         TO RASSSWPD-ERROR-MSG-IO
   PERFORM 8150-SEND-MAP     THRU 8150-EXIT
   PERFORM 9000-RETURN       THRU 9000-EXIT.

* 7050-EXIT.
   EXIT.
   EJECT
7100-DELETE-RASSPLSW.

* EXEC CICS
   DELETE DATASET ('RASSPLSW')
       RESP   (W-RESP)
   END-EXEC.

* IF W-RESP = ZERO
   NEXT SENTENCE
ELSE
   MOVE W-RESP               TO W-RESP-ZONED
   MOVE W-RESP-ZONED         TO W-ABEND-RESP
   MOVE '7100'               TO W-ABEND-para
   MOVE 'DELETE'             TO W-ABEND-FUNC
   MOVE W-ABEND-LINE         TO RASSSWPD-ERROR-MSG-IO
   PERFORM 8150-SEND-MAP     THRU 8150-EXIT
   PERFORM 9000-RETURN       THRU 9000-EXIT.

* 7100-EXIT.
   EXIT.
   EJECT
7200-UPDATE-RASSPLSW.
* EXEC CICS
  REWRITE DATASET ('RASSPLSW')
    FROM    (RASSPLSW-RECORD)
    RESP    (W-RESP)
END-EXEC.
*
IF W-RESP = ZERO
  NEXT SENTENCE
ELSE
  MOVE W-RESP                  TO W-RESP-ZONED
  MOVE W-RESP-ZONED            TO W-ABEND-RESP
  MOVE '72ØØ'                  TO W-ABEND-para
  MOVE 'UPDATE'                TO W-ABEND-FUNC
  MOVE W-ABEND-LINE            TO RASSSWPD-ERROR-MSG-IO
  PERFORM 815Ø-SEND-MAP        THRU 815Ø-EXIT
  PERFORM 9ØØØ-RETURN          THRU 9ØØØ-EXIT.
*
72ØØ-EXIT.
  EXIT.
  EJECT
73ØØ-WRITE-RASSPLSW.
*
EXEC CICS
  WRITE DATASET ('RASSPLSW')
    FROM    (RASSPLSW-RECORD)
    RIDFLD  (W-USERID)
    RESP    (W-RESP)
END-EXEC.
*
IF W-RESP = ZERO
  NEXT SENTENCE
ELSE
  MOVE W-RESP                  TO W-RESP-ZONED
  MOVE W-RESP-ZONED            TO W-ABEND-RESP
  MOVE '73ØØ'                  TO W-ABEND-para
  MOVE 'WRITE'                 TO W-ABEND-FUNC
  MOVE W-ABEND-LINE            TO RASSSWPD-ERROR-MSG-IO
  PERFORM 815Ø-SEND-MAP        THRU 815Ø-EXIT
  PERFORM 9ØØØ-RETURN          THRU 9ØØØ-EXIT.
*
73ØØ-EXIT.
  EXIT.
  EJECT
8ØØØ-ASSIGN-USERID.
*
EXEC CICS
  ASSIGN USERID (W-USERID)
END-EXEC.
*
8ØØØ-EXIT.
EXIT.
EJECT
8100-SEND-MAP.
*
EXEC CICS
  SEND MAP ('RASSWP')
    FROM (RASSSWPD)
    ERASE
END-EXEC.
*
8100-EXIT.
EXIT.
EJECT
8150-SEND-MAP.
*
EXEC CICS
  SEND MAP ('RASSWP')
    FROM (RASSSWPD)
    FREEKB
    ALARM
END-EXEC.
*
8150-EXIT.
EXIT.
EJECT
8200-SEND-MAP.
*
EXEC CICS
  SEND MAP ('RASSWP')
    FROM (RASSSWPD)
    ERASE
END-EXEC.
*
8200-EXIT.
EXIT.
EJECT
8300-RECEIVE-MAP.
*
EXEC CICS
  RECEIVE MAP ('RASSWP')
    INTO (RASSSWPD)
END-EXEC.
*
8300-EXIT.
EXIT.
EJECT
9000-RETURN.
*
EXEC CICS
  RETURN TRANSID ('SWPL')
    COMMAREA (W-COMMAREA)
Partitioned tablespaces page number calculator

Some of the stand-alone DB2 utilities, like DSN1PRNT and DSN1COPY, and other DB2 utilities, like REPAIR, require a DB2 page number as the input parameter. If the tablespace is partitioned, the byte string in PAGE X‘byte-string’ designates the partition number in certain high order bits and the page number in low order bits. The coding of partition and page number within the 24-bit string and...
conversion to a hex value has a logic which is based on page size (4K/32K), total number of partitions (up to 16, between 17 and 32, more than 32), and partition number to which the page belongs. Often, it becomes an additional job for a DBA to calculate a page number in hex before he runs these utilities.

This REXX EXEC takes page size, total number of partitions, partition number, and page serial number as input through an ISPF screen, calculates the bit values, converts to hex, and displays page number in hex on the screen.

/*REXX*/
DUMMY=MSG("OFF")
MSG=''
EOF='NO'
nparts=''
partnum=''
pgsize=''
pagesrl=''
parts_cat=''
val_nparts=Ø
val_partnum=Ø
string_1=''
string_2=''
pgnumhex=''
ADDRESS "ISPEXEC"
"LIBDEF ISPPLIB DATASET ID ( '<ISPPLIB>' )"
DO WHILE EOF='NO'
"DISPLAY PANEL (PPGNUMC)"
MSG=''
IF nparts=' ' THEN EXIT
if datatype(nparts,'w') = Ø then
    do
        msg='Number of partitions entered is not numeric'
        iterate
    end
    val_nparts=value(nparts)
select
    when val_nparts = 0 then
        do
            msg='Number of partitions entered is INVALID'
            iterate
        end
    when val_nparts < 17 then parts_cat='1'
    when val_nparts < 33 then parts_cat='2'
    when val_nparts < 65 then parts_cat='3'
    otherwise
        do

msg='Number of partitions entered is INVALID'
iterate
end

if ( pgsiz = '4K' ) | ( pgsiz = '32K' ) then
  nop
else
do
  msg='Page size supplied is INVALID'
iterate
end

if ( partnum='' ) | ( partnum=' ' ) then
do
  msg='Partition number given is INVALID'
iterate
end

if datatype(partnum.'w') = Ø then
do
  msg='Partition number entered is not numeric'
iterate
end
val_partnum=value(partnum)
if ( val_partnum = Ø ) | ( val_partnum > 64 ) then
do
  msg='Partition number given is INVALID'
iterate
end

if val_partnum > val_nparts then
do
  msg='Partition number given is INVALID'
iterate
end

if datatype(pagesrl.'w') = Ø then
do
  msg='Page serial number entered is not numeric'
iterate
end
val_pagesrl=value(pagesrl)
if val_pagesrl > 999999 then
do
  msg='Page serial number entered is too long'
iterate
end

CALL PROCESS_RTN
END /* DO WHILE */
ADDRESS "ISPEXEC"
"LIBDEF ISPPLIB"
EXIT

process_rtn:
val_partnum = val_partnum - 1

t = d2x(val_partnum)

t = x2b(t)

num_bits = length(t)

select
  when parts_cat = '1' then
    if pgsize = '4K' then
      select
        when num_bits = 4 then
          string_1 = t || copies('Ø', 2Ø)
        when num_bits > 4 then
          string_1 = substr(t, num_bits - 3, 4) || copies('Ø', 2Ø)
        otherwise nop
      end
    else
      select
        when num_bits = 4 then
          string_1 = 'ØØØ' || t || copies('Ø', 17)
        when num_bits = 5 then
          string_1 = 'ØØ' || t || copies('Ø', 17)
        when num_bits = 6 then
          string_1 = 'Ø' || t || copies('Ø', 17)
        when num_bits = 7 then
          string_1 = t || copies('Ø', 17)
        when num_bits > 7 then
          string_1 = substr(t, num_bits - 6, 7) || copies('Ø', 17)
        otherwise nop
      end
    end
  when parts_cat = '2' then
    if pgsize = '4K' then
      select
        when num_bits = 4 then
          string_1 = 'Ø' || t || copies('Ø', 19)
        when num_bits = 5 then
          string_1 = t || copies('Ø', 19)
        when num_bits > 5 then
          string_1 = substr(t, num_bits - 4, 5) || copies('Ø', 19)
        otherwise nop
      end
    else
      select
        when num_bits = 4 then
          string_1 = 'ØØØØ' || t || copies('Ø', 16)
        when num_bits = 5 then
          string_1 = 'ØØØ' || t || copies('Ø', 16)
        when num_bits = 6 then
          string_1 = 'ØØ' || t || copies('Ø', 16)
        when num_bits = 7 then
          string_1 = 'Ø' || t || copies('Ø', 16)
        when num_bits = 8 then
          string_1 = 'Ø' || t || copies('Ø', 16)
          otherwise nop
        end
      end
    end
  otherwise nop
end
string_1=t||copies('Ø',16)
when num_bits > 8 then
  string_1=substr(t,num_bits-7,8)||copies('Ø',16)
otherwise nop
end
when parts_cat='3' then
  if pgsize='4K' then
    select
      when num_bits = 4 then
        string_1='ØØ'||t||copies('Ø',18)
      when num_bits = 5 then
        string_1='Ø'||t||copies('Ø',18)
      when num_bits = 6 then
        string_1=t||copies('Ø',18)
      when num_bits > 6 then
        string_1=substr(t,num_bits-5,6)||copies('Ø',18)
      otherwise nop
    end
  else
    select
      when num_bits = 4 then
        string_1='ØØØØØ'||t||copies('Ø',15)
      when num_bits = 5 then
        string_1='ØØØØ'||t||copies('Ø',15)
      when num_bits = 6 then
        string_1='ØØØ'||t||copies('Ø',15)
      when num_bits = 7 then
        string_1='ØØ'||t||copies('Ø',15)
      when num_bits = 8 then
        string_1='Ø'||t||copies('Ø',15)
      when num_bits = 9 then
        string_1=t||copies('Ø',15)
      when num_bits > 9 then
        string_1=substr(t,num_bits-8,9)||copies('Ø',15)
      otherwise nop
    end
  end
t=d2x(val_pagesrl)
t=x2b(t)
um_bits=length(t)
select
  when parts_cat='1' then
    if pgsize='4K' then
      string_out=substr(string_1,1,4)||copies('Ø',20-num_bits)||t
    else
      string_out=substr(string_1,1,7)||copies('Ø',17-num_bits)||t
    end
  when parts_cat='2' then
    if pgsize='4K' then
      string_out=substr(string_1,1,5)||copies('Ø',19-num_bits)||t
    else
      string_out=substr(string_1,1,8)||copies('Ø',18-num_bits)||t
    end
  otherwise nop
end
string_out = substr(string_1,1,8)||copies('Ø',16-num_bits)||t
when parts_cat='3' then
    if pgsz='4K' then
        string_out = substr(string_1,1,6)||copies('Ø',18-num_bits)||t
    else
        string_out = substr(string_1,1,9)||copies('Ø',15-num_bits)||t
    end
end
t=b2x(string_out)
pgnumhex=t
return
/**************************************************************/
)ATTR
/**************************************************************/
/*                                                                    */
/* PPGNUMCR - Page Number HEX Calculation for Partitioned tablespace  */
/**************************************************************/
+ TYPE(TEXT)   INTENS(HIGH)  COLOR(BLUE)         SKIP(ON)
^ TYPE(TEXT)   INTENS(HIGH)  COLOR(GREEN)        SKIP(ON)
% TYPE(TEXT)   INTENS(HIGH)  COLOR(WHITE)        SKIP(ON)
$ TYPE(TEXT)   INTENS(HIGH)  COLOR(RED)          SKIP(ON)
# TYPE(OUTPUT) INTENS(HIGH)  COLOR(BLUE)         CAPS(ON)
)BODY CMD(C)
%---------------------+ DB2 PAGE NUMBER (Hex) CALCULATOR  --------------- +
%OPTION ===_C                                                +
+    $Enter Blanks in Total Number of Partitions,   to  Quit      +
+    $------------------------------------------------------      +
+    %Total Number of Partitions ( 1 - 64, 1 for Non-partioned):_z +
+    %Page Size ( 4K / 32K ):_z                              +
+    %Partition Number ( 1 - 64, 1 for Non-partitioned):_z      +
+    %Page Serial Number ( 1 - 999999 ):_z                     +
+    ^--------------------------------------------------------------- +
+    ^DB2 Page Number ( in hex ):#pgnumhex                      +
+    +
+    +
+    +
+    +
+    +
+    +
+    ^msg                                                           +
+    +
+    +
+    +
+    +
)INIT
.ZVARS = '(nparts,pgsz,partnum,pagesrl)'
)END

Sharad Kumar Pande
DB2 DBA (USA)  © Xephon 1998
This month we continue the set of functions and subroutines that extend IBM REXX. These functions interface with DB2. Requests to DB2 are made under TSO using standard SQL through the ADDRESS DB2 statement.

*DATA******************************************************************
VDSNALI DC V(DSNALI)       IBM DB2
LTORG
TABHEXE DC CL16'Ø123456789ABCDEF'
LMAXSQL DC F'33Ø16'        LNG OF A BIG SQLDA
NBSQVAR EQU 75Ø             NUMBER OF SQLVAR FOR SUCH A SQLDA
DS ØF
FØ DC X'ØØ0000000F'     TO TRANSLATE A 1-BYTE EXTENDED NB TO BIN
COLSANOM DC C'**'        INDICATE A W/O NAME COLUMN IN DESC
WRITEERR DC CL8'WRITEERR' IBM PROGRAM TO DISPLAY ERROR MESSAGES
NSQLCODE DC C'SQLCODE'   NAME OF A REXX VARIABLE
NSQSTAT DC C'SQLSTATE'   NAME OF A REXX VARIABLE
NREASON DC C'REASON'     NAME OF A REXX VARIABLE
BLANCS DC CL8'
SYNC DC CL4'SYNC'
PLAN DC CL8'$IRXDB3H'   NAME OF DB2 PLAN
OPEN DC CL12'OPEN'
LCLOSE DC CL12'CLOSE'
IDSQLDA DC CL8'SQLDA'
NOTNULL DC C' NOT NULL'
LNULL DC C'NULL_'        PREFIX FOR VAR RECEIVING NULL ATTRIBUTE
DECUR1 DC C'DECLARE C'
DECUR2 DC C'
DECUR3 DC C' CURSOR FOR '
DECURL EQU -*DECUR1 MIN LNG FOR A "DECLARE CURSOR..."
DECURH1 DC C'DECLARE H'
DECURH2 DC C'
DECURH3 DC C' CURSOR WITH HOLD FOR '
DECURHL EQU -*DECURH1 MIN LNG FOR A "DECLARE CURSOR WITH HOLD"
LT384 DC C' DATE ' THESE NUMBERS ARE FIXED BY IBM...
LT388 DC C' TIME '
LT392 DC C' TIMESTAMP '
LT448 DC C' VARCHAR('
LT452 DC C' CHAR('
LT456 DC C' LONG VARCHAR '
LT464 DC C' VARGRAPHIC(''
LT468 DC C' GRAPHIC(''
LT472 DC C' LONG VARGRAPHIC '
LT48Ø DC C' FLOAT(''
LT484 DC C' DECIMAL(''

LT496 DC C' INTEGER '
LT500 DC C' SMALLINT '
LTINC DC C' ? '

TYPES DS @F             SEE DSECT "TYPED"
T384 DC H'384',AL2(LT384-1),A(LT384) DATE
T388 DC H'388',AL2(LT388-1),A(LT388) TIME
T392 DC H'392',AL2(LT392-1),A(LT392) TIMESTAMP
T448 DC H'448',AL2(LT448-1),A(LT448) VARCHAR
T452 DC H'452',AL2(LT452-1),A(LT452) CHAR
T456 DC H'456',AL2(LT456-1),A(LT456) LONG VARCHAR
T464 DC H'464',AL2(LT464-1),A(LT464) VARYING GRAPHIC
T468 DC H'468',AL2(LT468-1),A(LT468) GRAPHIC
T472 DC H'472',AL2(LT472-1),A(LT472) LONG VARYING GRAPHIC
T480 DC H'480',AL2(LT480-1),A(LT480) FLOAT
T496 DC H'496',AL2(LT496-1),A(LT496) INTEGER
T500 DC H'500',AL2(LT500-1),A(LT500) SMALLINT
DC X'FFFF',AL2(LTINC),A(LTINC) UNKNOWN

TABFCT DS @F             POSSIBLE FUNCTIONS. SEE DSECT "FONCTD"
DC CL8'FETCH ',A(FETCH),AL2(6),AL2(Ø)
DC CL8'DELETE ',A(DELETE),AL2(7),AL2(Ø)
DC CL8'UPDATE ',A(UPDATE),AL2(7),AL2(Ø)
DC CL8'INSERT ',A(INSERT),AL2(7),AL2(Ø)
DC CL8'ROLLBACK',A(ROLLBACK),AL2(8),AL2(Ø)
DC CL8'GRANT ',A(GRANT),AL2(6),AL2(Ø)
DC CL8'REVOKE ',A(REVOKE),AL2(7),AL2(Ø)
DC CL8'ALTER ',A(ALTER),AL2(6),AL2(Ø)
DC CL8'CREATE ',A(CREATE),AL2(7),AL2(Ø)
DC CL8'COMMIT ',A(COMMIT),AL2(6),AL2(Ø)
DC CL8'DROP ',A(DROP),AL2(5),AL2(Ø)
DC CL8'DECLARE ',A(DECLARE),AL2(8),AL2(Ø)
DC CL8'OPEN ',A(OPEN),AL2(5),AL2(Ø)
DC CL8'CLOSE ',A(CLOSE),AL2(6),AL2(Ø)
DC CL8'SET ',A(SET),AL2(4),AL2(Ø)
DC CL8'LOCK ',A(LOCK),AL2(5),AL2(Ø)
LCONNEC DC CL8'CONN ',A(CONNECT),AL2(5),AL2(Ø)
LDISCON DC CL8'DISC ',A(DISCONN),AL2(4),AL2(Ø)
DC X'FF'                 END OF THIS TABLE
CODECF4E DC H'0'         CODE FOR CONV FLOAT->EXTENDED
CODECF8E DC H'8'         CODE FOR CONV DOUBLE FLOAT->EXTENDED
ANORM DC A(NORM)
AP2D DC A(P2D)

*BELOW ARE ROUTINES WITH THEIR OWN BASE REGISTERS    **********************
* CONVERT A PACKED DECIMAL (MAX LNG 16) TO EXTENDED. UNPK DOES NOT WORK
* FOR SUCH A LONG NUMBER.
* INPUT: R2->NUMBER, R1=LNG=Ø. MUST NOT BE $IRPACK1 NOR $IRPACK2
* OUTPUT, R2->EXTENDED NUMBER, R1=LENGTH
P2D EQU *
STM R14,R12,12(R13) SAVE REGISTERS
BASR R11,0       INIT BASE REGISTER
USING *,R11
XC $IRPAC16,$IRPAC16 RESET WORK AREA
LA R15,$IRPAC16+16 R15->...
SR R15,R1         WORK AREA
BCTR R1,0        LNG -1 FOR EX
EX R1,P2DMVC     STORE DECIMAL LENGTH
* HERE, $IRPAC16 CONTAINS DECIMAL VALUE (16 BYTES)
UNPK $IRET312,$IRPACK2 CONV HIGHER PART
MVO $IRDE9,$IRPACK1 STORE SHIFTED HIGHER PART
OI $IRDE9+8,X'0F' WHERE WE STUFF A SIGN
UNPK $IRZ16,$IRDE92 AND WHICH WE CONVERT TO EXTENDED
MVC $IRET311,$IRZ16 RE-STORE HIGHER PART
MVC $IRET311(1),$IRDE9 STUFF 1ST PACKED DIGIT AGAIN
OI $IRET311,X'F0' WHICH WE EXTEND
* HERE, $IRET31 CONTAINS EXTENDED NUMBER (31 BYTES), WITH SIGN ON 1ST
* HALF BYTE OF LAST BYTE.
LA R2,$IRET31     R2->RESULT
LA R1,L'$IRET31   R1=LNG OF RESULT
L R14,12(R13)    RESTORE REGS. BUT R1 AND R2...
XR R15,R15       ...AND ZERO R15...
L R0,20(R13)
LM R3,R12,32(R13)
BR R14
P2DMVC MVC Ø(*-*,R15),Ø(R2)
******************************************************************************
* NORMALIZE AN EXTENDED NUMBER, SIGNED ON LAST CHARACTER, BUT WITHOUT
* "E+99". MAY CONTAIN A DECIMAL POINT.
* INPUT: R2->NUMBER, R1=LENGTH=Ø
* OUTPUT:R2->NORMALIZED NUMBER,R1=LENGTH
NORM EQU *
STM R14,R12,12(R13) SAVE REGISTERS
BASR R11,0       INIT BASE REGISTER
USING *,R11
LA R14,Ø(R1,R2)  R14->...
BCTR R14,0       LAST BYTE OF THE NUMBER
XC $IRXØ,$IRXØ RESET WORK AREA FOR SIGN
MVZ $IRXØ,Ø(R14) STORE SIGN INTO WORK AREA
NI $IRDраГ1,X'FF'-$IRDраГØ INDICATE SIGN IS "+
CLI $IRXØ,X'D0' IS THE NUMBER <Ø ?
BE NORM1Ø YES, B
CLI $IRXØ,X'B0' IS THE NUMBER <Ø ?
BNE NORM2Ø NO, B
NORM1Ø EQU *
OI $IRDраГ1,$IRDраГØ INDICATE SIGN IS "-
NORM2Ø EQU *
OI Ø(R14),X'F0' FORCE LAST CHAR DISPLAYABLE
NORM3Ø EQU *
CLI Ø(R2),C'0' IS IT A ZERO?
BNE NORM5Ø NO, FOUND.
LA R2,1(,R2)   YES. NEXT BYTE
BCT R1,NORM3Ø LOOP
BCTR R2,Ø ONLY ZEROS. COME BACK ON LAST ZERO
LA R1,1 AND FORCE LENGTH=1

NORM5Ø EQU * HERE R2->1ST ¬Ø, R1=REMAINDING L.W/O SIGN
LR R15,R1 R15=LNG OF RESULT
LA R14,$IRMFE R14->AREA TO BE FILLED UP
TM $IRDRAP1,$IRDRØ THE NUMBER WAS< Ø ?
BNO NORM6Ø NO, B
MVI Ø(R14),C'-' YES, STORE MINUS SIGN
LA R14,1(,R14) THE NB WILL BE STUFFED 1 CHAR FARTHER
LA R1,1(,R1) AND WILL HAVE 1 MORE BYTE

NORM6Ø EQU *
BCTR R15,0 LNG-1 FOR EX
EX R15,NORMMVC2 STORE THE NORMALIZED NUMBER
LA R2,$IRMFE R2->RESULT, R1=LNG OF RESULT
L R14,12(,R13) RESTORE REGISTERS BUT R1 AND R2...
XR R15,R15 ZERO R15
L R0,20(,R13)
LM R3,R12,32(R13)
BR R14

NORMMVC2 MVC Ø(R15-R15,R14),Ø(R2)

*DSECTS****************************************************************
EXEC SQL INCLUDE SQLDA

EXEC SQL INCLUDE SQLDA

typed dsect table of types
typet ds h data type
typel ds h lng of string
typea ds f string
typedl equ *-typed length of 1 entry
fonct dsect
foncta ds cl8 function
foncta ds a address of module for this function
fonctl ds al2 length of function
dsa al2 unused
fonctl dol equ *-fonctd length of 1 entry
copy $irxdsec

sauv ds 18f,18f,18f registers save area, level 0, 1, 2
sauvl equ *-sauv
irxefpl
irxevalb
irxenvb
irxworkb

db2d dsect
exec sql include sqlca

db2sql ds h,cl2ø48 sql request
org db2sql
db2sqlld ds h
db2sqls ds cl2ø48
org
lng equ sqldlen
$IRX\ DSECT\ MAIN\ TABLE\ OF\ REXX/PLUS$
$IRXID\ DS\ CL4\ EYECATCHER\ "$IRX"
* HEREUNDER\ ARE\ SEVERAL\ DOUBLE\ WORDS:\ ADDRESS/LENGTH.\ THE\ POINTED
* AREAS\ WILL\ BE\ AUTOMATICALLY\ FREED\ BY\ CLEAN\ UP\ ROUTINE\ $IRXTERM$
$IRXTER\ DS\ ØF
$IRZFIC\ DS\ F  \ ->WORK\ AREA\ FOR\ FILES
$IRZFICL DS F \ LENGTH
$IRXMODU DS F \ ->TABLE\ OF\ ADDRESSES\ OF\ MODULES
$IRXMODL DS F \ LENGTH
$IRXDB2A DS F \ WORK\ AREA\ FOR\ DB2
$IRXDB2L DS F \ LENGTH
$IRSQLP1 DS ØF \ AREAS\ FOR\ DB2\ CURSORS\ (DSECT\ $IRSQLAD)$
   DS F \ CURSOR\ 1:\SQLDA
   DS F \ LENGTH\ OF\ THIS\ SQLDA
   DS F \ AREA\ RECEIVING\ THE\ COLUMNS
   DS F \ LNG
   DS F \ CURSOR\ 2:\SQLDA
   DS F \ LENGTH\ OF\ THIS\ SQLDA
   DS F \ AREA\ RECEIVING\ THE\ COLUMNS
   DS F \ LNG
   DS F \ CURSOR\ 3:\SQLDA\ FOR\ CURSOR\ 3
   DS F \ LENGTH\ OF\ THIS\ SQLDA
   DS F \ AREA\ RECEIVING\ THE\ COLUMNS
   DS F \ LNG
   DS F \ CURSOR\ 4:\SQLDA
   DS F \ LENGTH\ OF\ THIS\ SQLDA
   DS F \ AREA\ RECEIVING\ THE\ COLUMNS
   DS F \ LNG
   DS F \ UNUSED\ ADDRESS
   DS F \ UNUSED\ LENGTH.\ ADD\ AS\ MANY\ ADDR/LNG\ AS\ YOU\ WANT
* TABLE $IRX\ ITSELF\ MUST\ OF\ COURSE\ BE\ THE\ LAST\ AREA\ TO\ BE\ FREED:
$IRX$IRX DS F \ ADDRESS\ OF\ $IRX\ (ITSELF)$
$IRXLNG DS F \ LNG\ OF\ $IRX\ (ITSELF)$
* \------------END\ OF\ ADDRESSES/LENGTHS----------
$IRXNPAL DS F \ NUMBER\ OF\ ENTRIES\ TO\ FREE
$IRXENVB DS F \ ADDR\ OF\ ENV\ BLOCK\ GIVEN\ BY\ REXX
$IRXWBE DS F \ ADDR\ OF\ WORK\ BLOCK\ EXT\ UNDER\ WHICH\ $IRX\ WAS\ CREATED
$IRDDND DS F \ ->1ST\ DDND\ (A\ DDND\ DESCRIBES\ A\ FILE)
*----------------------------------------------------------
$IRNBARG DS X \ NB\ OF\ ARG\ (CALCULATED\ BY\ IRXFLOC)
$IRFONC DS X \ CODE\ FOR\ REQUESTED\ FUNCTION
$IRZ1 DS D \ WORK\ AREA\ (PACKED\ ARG\ 1)
$IRZ2 DS D \ WORK\ AREA\ (PACKED\ ARG\ 2)
$IRZ3 DS D WORK AREA (PACKED ARG 3)
$IRZ4 DS D WORK AREA (PACKED ARG 4)
$IRZ5 DS D WORK AREA (PACKED ARG 5)
$IRPAC16 DS CL16
  ORG $IRPAC16
$IRPACK1 DS D WORK AREA ... THESE 3 AREAS
$IRPACK2 DS D WORK AREA ... MUST BE
$IRPACK3 DS D WORK AREA ... CONTIGUOUS
$IRMFE DS 2OF WORK AREA
$IRFØ DS F WORK AREA
$IRF1 DS F WORK AREA
$IRF2 DS F WORK AREA
$IRF3 DS F WORK AREA
$IRHØ DS H WORK AREA
$IRH1 DS H WORK AREA
$IRH2 DS H WORK AREA
$IRH3 DS H WORK AREA
$IRXØ DS X WORK AREA
$IRX1 DS X WORK AREA
$IRX2 DS X WORK AREA
$IRX3 DS X WORK AREA
$IRDRAP1 DS X WORK AREA
$IRDRØØ EQU X'8Ø'
$IRDRØ4 EQU X'4Ø'
$IRDRØ2 EQU X'2Ø'
$IRDRØ1 EQU X'1Ø'
$IRDRØ8 EQU X'Ø8'
$IRDRØ4 EQU X'Ø4'
$IRDRØ2 EQU X'Ø2'
$IRDRØ1 EQU X'Ø1'
**********WORK AREA FOR IRXEXCOM (REXX VARIABLES ASSIGNMENT)
$IRXEXCA DS F ADDRESS OF IRXEXCOM
$IREXCOM DS ØF
$IRCOMNA DS F PTR TO "IRXEXCOM"
  DS F Ø
  DS F Ø
$IRADSHV DS F PTR TO SHVBLOCK ($IRNEXT)+X'8Ø...'
$IRCOMMN DS CL8 "IRXEXCOM"
$IRNEXT DS A STRUCTURE IDENTICAL TO REXX SHVBLOCK
$IRUSER DS F
$IRCODE DS CL1 "S": SET VARIABLE
  DS H'Ø'
$IRBUFL DS F LNG OF 'FETCH' VALUE BUFFER
$IRNAMA DS F ADDR NAME OF VAR
$IRNAML DS F LNG NAME OF VAR
$IRVALA DS A ADDR FOR VALUE
$IRVALL DS F LNG OF VALUE
$IRBLEN EQU *-$IREXCOM LNG BLOCK FOR IRXEXCOM
$IRZONE DS ØD WORK AREA FOR OBTAIN AND LOCATE
  DS XL265
$IRET31 DS CL31 WORK AREA
ORG $IRET31
$IRET311 DS CL16 FOR DECIMAL->EXTENDED CONVERSION
$IRET312 DS CL15 "
ORG $IRET31
$IREU32Ø DS CL1
$IREU321 DS CL15
$IREU322 DS CL15
$IRDE9 DS CL9
ORG $IRDE9
$IRDE91 DS CL1
$IRDE92 DS CL8
$IRZ16 DS CL16
$IRZ32 DS CL32
ORG $IRZ32
$IRZ22 DS CL22 FOR FLOAT.Conv +.9999999999999999E+99
ORG $IRZ32
$IRZ22S DS C SIGN
DS C DECIMAL POINT
$IRZ22N DS CL16 9999999999999999
ORG $IRZ22N
DS CL9 999999999 FOR SINGLE FLOAT.
$IRZ15E1 DS CL4 E+99 FOR SINGLE FLOAT.
$IRZ15L EQU *-$IRZ22S LNG OF A SINGLE PRECISION FLOATING POINT
DS CL3 NOTHING FOR SINGLE FLOAT.
$IRZ22E DS C E
$IRZ22ES DS C SIGN OF EXPONENT
$IRZ22E9 DS CL2 EXPONENT 99
$IRZ22L EQU *-$IRZ22S LNG OF AREA

*******************************WORK AREA FOR IRXRLT
$IRXRLT DS ØF (REQUEST FOR A NEW EVALBLOCK)
$IRLTGA DS A $IRLTG) PARAM1->'GETBLOCK'
$IRLTAA DS A $IRLTBA) PARAM2->WILL RECEIVE A(EVALBLOCK)
$IRLTLA DS F PARAM3->LNG OF DESIRED RESULT
$IRLTG DS CL8 'GETBLOCK'
$IRLTBA DS F ADDRESS OF RECEIVED EVALBLOCK

**********************************************************************
$IRXDB2 (SET UP HOST CMD)
$IRDB2EN DS CL32 1 ENTRY OF 'HOST CMD ENV TABLE'

**********************************************************************
$IRXDB2H DB2 PROCESSING
$IRDB2 DS CL8 DB2ID
$IRDB2RT DS F RETURN CODE
$IRDB2RE DS F REASON CODE
$IRCOLNT DS H NAME OF VAR RECEIVING COL DESCRIPTION
$IRCOLNV DS CL32 "
DS CL6 SAFETY MARGIN FOR COLUMN NUMBER
$IRCOL DS H NAME OF VAR RECEIVING NAMELESS COLUMNS
$IRCOLV DS CL32 "
DS CL6 SAFETY MARGIN FOR COLUMN NUMBER
$IRCOLVL EQU *-$IRCOLV
$IRDB2DR DS X       FLAGS:
$IRDB2FO EQU X'8Ø'  MUST FORCE NAME OF COLUMNS
$IRDB2NC EQU X'4Ø'  DON'T TRANSLATE DATA GOT FROM DB2
$IRAMCF DS A        ADDR OF MODULE FOR FLOAT. CONVERSION
$IRXLU EQU *-$IRX   USED LENGTH
                 DS XL(4095-$IRXLU) LNG 4K-1: ADDRESSABLE BY 1 REG
$IRXL EQU *-$IRX    LENGTH OF TABLE $IRX

***********************
$IRSQLAD DSECT      $IRXDB2x: MAP FOR 1 ENTRY ADDR/LNG
$IRSQLAØ DS F       ADDRESS OF SQLDA
$IRSQLLØ DS F       LNG OF THIS SQLDA
$IRCOLAØ DS F       ADDR OF AREA RECEIVING DB2 COLUMNS
$IRCOLLØ DS F       LNG OF THIS AREA
$IRSQLDL EQU *-$IRSQLAD LNG OF 1 ENTRY
ERØØØ EQU 0        ERØØ1 EQU 1
ERØØ2 EQU 2        ERØØ3 EQU 3
ERØØ4 EQU 4        ERØØ5 EQU 5
ERØØ6 EQU 6        ERØØ7 EQU 7
ERØØ8 EQU 8

ARGUM DSECT        PARAMETERS FOR IRXFLOC
ARGUM1P DS F       ->ARG 1
ARGUM1L DS F       LNG OF ARG 1
ARGUML EQU *-ARGUM
ARGUM2P DS F       ->ARG 2
ARGUM2L DS F       LNG
ARGUM3P DS F       ->ARG 3
ARGUM3L DS F       LNG
ARGUM4P DS F       ->ARG 4
ARGUM4L DS F       LNG
ARGUM5P DS F       ->ARG 5
ARGUM5L DS F       LNG
ARGUM6P DS F       ->ARG 6
ARGUM6L DS F       LNG
FRØ EQU 0          FLOATING POINT REGISTER Ø
FR2 EQU 2          COPY REGS
FR4 EQU 4

IRXMSG

$IRXMSG CSECT
$IRXMSG AMODE ANY
$IRXMSG RMODE ANY

******************************************************************
* DISPLAY THE ERROR MESSAGE NUMBER (R2)
******************************************************************
STM R14,R12,12(R13)
BALR R1Ø,0
USING *,R1Ø
LR R12,R13
LA R13,72(,R13)  NEXT SAVE AREA
ST R12,4(,R13)
ST R13,8(,R12)
USING $IRX,R8
SLL R2,3  MESSAGE NUMBER * 8
LA R3,MSGI(R2)  R3->MESSAGE
LA R2,MSGI+4(R2)  R2->LNG OF MESSAGE
LA R15,WRITEERR  R1->"WRITEERR"
ST R15,$IRMFE  STORE ADDR OF "WRITEERR"
ST R3,$IRMFE+4  STORE ADDR OF ADDR OF MESSAGE
ST R2,$IRMFE+8  STORE ADDR 3RD PARAM (LNG MSG)
OI $IRMFE+8,X'8Ø'  INDICATE "LAST PARAMETER"
LA R1,$IRMFE  R1->LIST OF ADDR OF PARAM
L R0,$IRXENVB  R0->ENV BLOCK
LINK EP=IRXSAY  DISPLAY THE MESSAGE
L R13,4(,R13)
LM R14,R12,12(R13)
XR R15,R15
BR R14

*DATA****************************************************************
WRITEERR DC CLB'WRITEERR'
MSGI DS 0F
  DC A(MØ),A(L'MØ)
  DC A(M1),A(L'M1)
  DC A(M2),A(L'M2)
  DC A(M3),A(L'M3)
  DC A(M4),A(L'M4)
  DC A(M5),A(L'M5)
  DC A(M6),A(L'M6)
  DC A(M7),A(L'M7)
  DC A(M8),A(L'M8)
MØ DC C'$IRXØØØE THIS FUNCTION IS NOT OPERATIONAL YET'
M1 DC C'$IRXØ15E NUMBER OF ARGUMENTS INVALID'
M2 DC C'$IRXØ45E DB2: SYNTAX ERROR'
M3 DC C'$IRXØ46E STRING IS TOO LONG'
M4 DC C'$IRXØ47E FETCH WITHOUT PREVIOUS DECLARE'
M5 DC C'$IRXØ48E ERROR WHEN STORING IN REXX VARIABLE'
M6 DC C'$IRXØ49E RETURN CODE OF EXEC SQL ¬=Ø'
M7 DC C'$IRXØ5ØE $DB2INST: AN ARGUMENT IS INVALID'
M8 DC C'$IRXØ54E $DB2INST FUNCTION NOT PREVIOUSLY USED'
COPY $IRXDSEC
END
IRXTERM

$IRXTERM START Ø
$IRXTERM AMODE ANY
$IRXTERM RMODE ANY

*****************************************************************

* $IRXTERM, CLEAN UP ROUTINE. AUTHOR: PATRICK LELOUP.
*****************************************************************

STM   R14,R12,12(R13) SAVE REGISTERS
BALR  R10,Ø INIT BASE REGISTER
USING *,R10 ADDRESSABILITY
LR    R6,R0 R6->ENVBLOCK
GETMAIN R,LP=SUVL
ST    R1,8(R13)
ST    R13,4(R1)
LR    R13,R1
LTR   R6,R6 ENVBLOCK IS PRESENT?
BZ    FIN NO, NOTHING TO DO
USING ENVBLOCK,R6
L    R6,ENVBLOCK_WORKBLOK_EXT R6->WORK BLOCK EXTENSION
DROP  R6
LTR   R6,R6 WORK BLOCK EXTENSION PRESENT?
BZ    FIN NO, NOTHING TO DO
USING WORKBLOK_EXT,R6
L    R8,WORKEXT_USERFIELD R8->USER FIELD (OUR $IRX)
LTR   R8,R8 OUR TABLE PRESENT?
BZ    FIN NO, NOTHING TO DO
USING $IRX,R8
CLC $IRXID,ID IS IT REALLY OURS?
BNE   FIN NO, DON'T TOUCH ANYTHING!
C    R6,$IRXWBE THIS $IRX CREATED UNDER CURRENT WORKBLOK_EXT?
BNE   FIN NO, DON'T DO ANYTHING

* STARTING AT $IRXTER ARE DOUBLE WORDS: ADDR & LNG OF AREAS TO FREE
L    R5,$IRXNPAL R5:=NUMBER OF ENTRIES
LA    R2,$IRXTER R2->1ST ADDRESS
USING TABLGET,R2

L1Ø  EQU *
L    R3,TABLADR R3:=ADDRESS OF AREA TO BE FREED
LTR   R3,R3 SOMETHING TO DO WITH THIS ADDRESS?
BZ   L2Ø NO, LOOP
L    R4,TABLLNG YES, R4:=LENGTH OF AREA
FREEMAIN RU,LP=(R4),A=(R3) FREEMAIN THE AREA

L2Ø  EQU *
LA    R2,TABLGETL(R2) NEXT ENTRY
DROP  R2
BCT  R5,L1Ø LOOP

* CAUTION, LAST AREA IS $IRX ITSELF: DON'T USE IT ANYMORE:
DROP  R8

L3Ø  EQU *
FIN  EQU *

© 1998. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (940) 455 7050, fax (940) 455 2492.
L     R2,4(R13)
FREEMAIN R,LV=SAUVL,A=(R13)
LR    R13,R2
LM    R14,R12,12(R13) RESTORE REGISTER
XR    R15,R15         ZERO RETURN CODE
BR    R14             RETURN

*DATA************************************************************************
ID       DC    CL4'$IRX'       EYECATCHER FOR OU $IRX TABLE
BLANCS   DC    CL8' '          
DS    ØF
DELIM    DC    X'FFFFFFFF'     DELIMITER FOR AREAS TO FREE

*DSECTS************************************************************************
SAUV     DS    18F
SAUVL    EQU   *-SAUV
TABLGET  DSECT   AREAS TO FREE
TABLADR  DS    F         ADDRESS
TABLLNG  DS    F         LENGTH
TABLGETL EQU   *-TABLGET       LENGTH OF 1 ENTRY
COPY  $IRXDSEC
IRXEFPL
IRXEVALB
IRXENVB
IRXWORKB
END

A24
MACRO
A24   &REG
&R      SETC  '&REG'
AIF   ('&REG' NE '').L1
&R      SETC  '14'
.L1     ANOP
LA    &R,*+6          ADDRESS WITH BIT Ø = Ø
BSM   Ø,&R            SET 24 BITS ADDRESSING MODE
MEXIT
MEND

A31
MACRO
A31   &REG
&R      SETC  '&REG'
AIF   ('&REG' NE '').L1
&R      SETC  '14'
.L1     ANOP
&ETIQ   SETC  '&SYSNDX'
L     &R,A&ETIQ         ADDRESS WITH BIT Ø = 1
IRXFLOC

IRXFLOC CSECT
IRXFLOC AMODE ANY
IRXFLOC RMODE ANY
*********************************************************************
* IRXFLOC. AUTHOR: PATRICK LELOUP.
*********************************************************************
DC   CL8'IRXFPACK'   FIXED BY IBM
DC   FL4'24'         LNG OF HEADER
DC   AL4(NBENT2)     NB OF ENTRIES
DC   FL4'Ø'
DC   FL4'32'         LNG OF 1 ENTRY
NBENT1 EQU   *
$DB2INST MACFD
NBENT2 EQU   (*-NBENT1)/32   NB OF ENTRIES
**********
* EACH FUNCTION MUST LOAD R3 WITH THE MODULE NUMBER TO LOAD: 1, 2, 3...
* (SEE IN TABLE TABMODU: INSERT THE NAME OF THE MODULE TO LOAD, IN THE
* BEST PLACE). 2 OR MORE DIFFERENT FUNCTIONS MAY HAVE THE SAME MODULE.
* EACH FUNCTION MUST LOAD R7 WITH A FUNCTION NUMBER WHICH MUST BE
* UNIQUE FOR A GIVEN MODULE.
$DB2INST MACF MODULE=1,FONCTION=Ø    MODULE $IRXDB2 (DB2)
************************************************************
COMMUN   DS    ØH
BASR  R1Ø,Ø           INIT BASE REGISTER
USING *,R1Ø
LR    R9,R0           R9->ENVBLOCK
GETMAIN R,LV=SAUVL
ST    R1,8(.R13)
ST    R13,4(.R1)
LR    R13,R1
LR    R6,R9           R6->ENVBLOCK
USING ENVBLOCK,R6
L    R6,ENVBLOCK_WORKBLOK_EXT R6->WORK BLOCK EXTENSION
DROP  R6
USING WORKBLOK_EXT,R6
L    R8,WORKEXT_USERFIELD     R8->USER FIELD
USING $IRX,R8
LTR   R8,R8           OUR MAIN TABLE EXISTS ?
BNZ   L2Ø             YES, B
GETMAIN R,LV=$IRXL,SP=Ø,LOC=BLOW NO, GETMAIN IT
*   BELOW BECAUSE OF IBM MACROS IBM GET, PUT, ETC
ST    R1,WORKEXT_USERFIELD STORE ADDRESS OF OUR AREA
LR     R8,R1          R8->OUR AREA
LR     R14,R8         R14->AREA TO BE ZEROED

* INITIALIZE OUR MAIN TABLE $IRX
LA     R15,$IRXL      R15:=LNG OF THE TABLE TO ZERO
XR     R4,R4          ADDRESS OF EMISSION:=Ø
XR     R5,R5          LNG OF EMISSION:=Ø
MVCL   R14,R4         CLEAR THE TABLE
ST     R8,$IRX$IRX    STORE ADDRESS OF THE TABLE
LA     R15,$IRXL      R15:=LNG OF TABLE
ST     R15,$IRXLNG    STORE INTO ITSELF
MVC    $IRXID,ID       STORE ID
ST     R9,$IRXENVB     STORE ADDR ENV BLOCK GIVEN BY REXX
ST     R6,$IRXWBE      STORE ADR WORK BLOCK EXT UNDER WHICH
                      THE TABLE $IRX WAS CREATED

* DROP R6
* STORE NUMBER OF ENTRIES TO BE FREED BY $IRXTERM:
LA     R15,$IRXNPAL   R15:=ADDR OF LAST ENTRY + 1
LA     RØ,$IRXTER     RØ:=ADDR 1ST ENTRY
SR     R15,RØ         R15:=NB OF BYTES IN 1 ENTRY
SRL    R15,3          /8=>R15:=NB OF ENTRIES TO FREE
ST     R15,$IRXNPAL   STORE

* INITS FOR IRXEXCOM (REXX VARIABLES ASSIGNMENT)
MVC    $IRCOMNM,=CL8'IRXEXCOM' REXX NAME OF VAR MANIP MODULE
MVI    $IRCODE,C'S'    CODE FOR "SET VARIABLE"
LA     R15,$IRCOMNM    R15->1ST PARM FOR IRXEXCOM
ST     R15,$IRCOMNA    STORE
LA     R15,$IRNEXT     R15->4TH PARM FOR IRXEXCOM
ST     R15,$IRADSHV    STORE
OI     $IRADSHV,X'8Ø'  SET LEFT BIT TO 1 (LAST PARM)

* INITS FOR IRXRLT (REQUEST FOR A LARGER EVALBLOCK)
MVC    $IRLTG,=CL8'GETBLOCK' STORE FUNCTION NAME
LA     R15,$IRLTG      R15->1ST PARM (->'GETBLOCK')
ST     R15,$IRLTGA     STORE E
LA     R15,$IRLTBA     R15->2ND PARM (->Ø, RETURNS BLOCK ADDR)
ST     R15,$IRLTA     STORE
LA     R15,$IRVALL     R15->3RD PARM (LNG OF DESIRED RESULT)
ST     R15,$IRTLA     STORE
OI     $IRTLA,X'8Ø'    SET LEFT BIT TO 1 (LAST PARM)

* CREATE THE TABLE OF MODULES ADDRESSES
GETMAIN R, LV=MODUL, SP=Ø GETMAIN IT
ST     R1,$IRXMODU    STORE
LA     R15,MODUL      R15:=LNG OF THE MODULES TABLE
ST     R15,$IRXMODL   STORE IN TABLE $IRX
LR     R14,R1        R14->TABLE TO BE ZEROED
LA     R15,MODUL      R15:=LNG OF THE TABLE
XR     R4,R4          ADDR OF EMISSION:=Ø
XR     R5,R5          LNG OF EMISSION:=Ø
MVCL   R14,R4         CLEAR THE TABLE

L2Ø    EQU    *        HERE, R8->MAIN TABLE ($IRX)
CLC    $IRXID,ID      IS THIS TABLE OUR TABLE ?
BNE ABEND1 NO, B ABEND

L21 EQU * COUNT THE NUMBER OF ARGUMENTS
L R1,4(.R13) R1:...
L R1,24(.R1) R1 AT THE ENTRY OF IRXFLOC
USING EFPL,R1
L R1,EFPLARG R1->LIST OF ARGUMENTS
USING ARGUM,R1
XR R15,R15 R15 WILL BE THE NUMBER OF ARG

ARGC EQU *
CLC ARGUM1P,FINARG END OF ARGUMENTS ?
BE ARGC5 YES, B
LA R15,R15 ENTRY FOR THIS MODULE
LA R1,4(,R13) R1->PREVIOUS SAVE AREA
LA R0,20(,R1) R0 AT ENTRY OF IRXFLOC
LR R1,24(,R1) R1 AT ENTRY OF IRXFLOC
LR R2,R7 R2:=NUM OF FCT IN MODULE
STC R15,$IRNBARG STORE NUMBER OF ARGUMENTS

* LOAD EVENTUALY, AND CALL CONCERNED MODULE
L R4,$IRXMODU R4->TABLE OF MODULES ADDRESSES
BCTR R3,0 -1, SO RELATIVE TO Ø
SLL R3,2 NUM FCT * 4
L R15,Ø(R3,R4) R15->ENTRY FOR THIS MODULE
LTR R15,R15 ADDR=Ø ?
BNE L5Ø NO, B CALL THIS MODULE
LA R15,TABMODU YES, MUST LOAD. R15->MODULES NAMES
LR R14,R3 R14:=R3...
SLL R14,1 R3*2 BECAUSE 1 ENTRY IS 8 BYTES LONG
LA R2,Ø(R15,R14) R2->NAME OF THE MODULE TO LOAD
LOAD EPLOC=(R2) LOAD THE MODULE
ST R0,Ø(R3,R4) STORE MODULE ADDRESS
LR R15,RØ R15->MODULE

ARGC5 EQU *
STC R15,$IRNBARG STORE NUMBER OF ARGUMENTS

* WE CALL MODULE WITH RØ = RØ AT ENTRY OF IRXFLOC.
* R1 = R1 AT ENTRY OF IRXFLOC.
* R2 = NUM OF FUNCTION IN MODULE.
* (ALSO STORED IN $IRFONC DS C)
* R8 -> TABLE $IRX

BASR R14,R15 CALL THE MODULE

* AT EXIT OF MODULE, R15=Ø OR R15=NUM OF ERROR MESSAGE
* R1=Ø IF ALPHA,
* 4 IF NUMERIC, TO BE NORMALIZED
LTR R2,R15 RETURN CODE Ø ?
BNZ L6Ø NO, B
B *+4(R1) B ACCORDING TO NORMALIZ. TO BE DONE
B FIN B IF NOTHING TO DO
B     NORM     B IF NORMALIZATION REQUIRED
* NORMALIZATION OF A NUMERIC VALUE.
* NUMERIC VALUES MUST BE PROVIDED SIGNLESS EXTENDED, OR WITH
* OVERPUNCHED SIGN ON LAST BYTE, BUT W/O A LEADING SIGN
NORM EQU *            NORMALIZATION OF A NUMERIC VALUE
  L    R1,4(,R13)     R1:=...
  L    R1,24(,R1)     R1 AT ENTRY OF IRXFLOC
USING EFPL,R1
L    R4,EFPLEVAL
L    R4,0(,R4)       R4->EVALBLOCK
USING EVALBLOCK,R4
L    R1,EVALBLOCK_EVLEN R1:=LNG
LTR   R1,R1           LNG<=Ø?
BNP   NORM99          YES, NOTHING TO DO
CH    R1,=H'32'       LNG > 32 ?
BH    NORM99          YES, NOTHING TO DO
BCTR  R1,Ø            LNG -1 FOR EX
EX    R1,NORMMVC1     STORE NUMBER IN WORK AREA
LA    R14,$IRMFE(R1)  R14->LAST BYTE OF THE NUMBER
XC    $IRX0,$IRX0     CLEAR WORK AREA FOR SIGN
MVZ   $IRX0,Ø(R14)    STORE SIGN IN WORK AREA
NI    $IRDRAP1,X'FF'-$IRDR8Ø SET "WE HAVE A +"
CLI   $IRX0,X'D0'     IS NUMBER < Ø ?
BE    NORM1Ø          YES, B
CLI   $IRX0,X'B0'     IS NUMBER < Ø ?
BNE   NORM2Ø          NO, B
NORM1Ø EQU *          NUMBER IS < Ø
O1    $IRDRAP1,$IRDR8Ø SET "WE HAVE A -"
NORM2Ø EQU *
  O1    Ø(R14),X'F0'   FORCE LAST BYTE READABLE
  * HERE, $IRDRAP1 CONTAINS X'80' IF < Ø
  * $IRMFE CONTAINS THE EXTENDED NUMBER TO BE NORMALIZED
LA    R2,$IRMFE       R2->1ST BYTE TO BE PROCESSED
L    R1,EVALBLOCK_EVLEN R1:=LNG TO BE PROCESSED (>Ø)
NORM3Ø EQU *          LOOP FOR SEARCHING OF 1ST NON ZERO
CLI   Ø(R2),C'Ø'     IS IT A Ø ?
BNE   NORM5Ø          NO, 1ST ZERO FOUND
LA    R2,1(,R2)      YES, NEXT BYTE
BCT   R1,NORM3Ø       LOOP
BCTR  R2,Ø            WE HAD ONLY C'Ø'. POINT TO LAST C'Ø'
LA    R1,1           AND FORCE LNG:=1
NORM5Ø EQU *
  * HERE, R2->1ST NON Ø IN $IRMFE, R1=REMAINING LNG TO BE PROCESSED (>Ø)
LR    R15,R1     R15:=LNG OF RESULT
LA    R14,EVALBLOCK_EVDATA R14->AREA TO BE FILLED
TM    $IRDRAP1,$IRDR8Ø IS NUMBER < Ø ?
BNO   NORM6Ø          NO, -B
MVI   EVALBLOCK_EVDATA,C'-' YES, STORE "-" SIGN
LA    R14,1(,R14)    WE MUST FILL 1 BYTE LATER
LA    R15,1(,R15)    THE RESULT WILL HAVE 1 MORE BYTE (-SIGN)
NORM6Ø EQU *
ST R15,EVALBLOCK_EVLEN STORE LNG OF RESULT
BCTR R1,Ø LNG - 1 FOR EX
EX R1,NORMMVC2 STORE RESULT

NORM99 EQU *
XR R15,R15 CLEAR RETURN CODE FOR REXX
B FIN

NORMMVC2 MVC Ø(R1-R1,R14),Ø(R2) STORE $IRMFE IN EVDATA
NORMMVC1 MVC $IRMFE(R1-R1),EVALBLOCK_EVDATA
DROP R4

L6Ø EQU * SEND THE ERROR MESSAGE (R2)
LINK EP=$IRXMSG SEND THE MESSAGE (R2)
LA R15,4 WRONG RETURN CODE FOR REXX

FIN EQU *
LR R3,R15 SAVE R15
L R2,4(.R13)
FREEMAIN R,LV=SAUVL,A=(R13)
LR R13,R2
LR R15,R3 RESTORE R15
L R14,12(.R13) RESTORE R14
LM R0,R12,20(R13) RESTORE REGS
BR R14 RETURN

ABEND1 ABEND 4ØØ2 ABEND: MAIN TABLE IS NOT $IRX
LTORG

*DATA****************************************************************
TABMODU DS ØF TABLE OF MODULES NAMES
DC CL8'$IRXDB2 ' MODULE 1: INIT DB2
ID DC CL4'$IRX' ID OF TABLE $IRX
FINARG DC X'FFFFFFFF' FOR THE END OF ARGUMENTS
*DSECTS************************************************************
COPY $IRXDSEC
SAUV DS 18F,18F,18F,18F SAVE AREA LEVEL Ø, 1, 2, 3
SAUVL EQU *-SAUV
MODU DSECT MODULES ADDRESSES. PTD TO BY $IRXMODU
DS F MODULE 1: $IRXDB2
MODUL EQU *-MODU LNG OF TABLE
IRXEFPL
IRXENVB
IRXWORKB
END

MACF

MACRO
&NOM MACF &MODULE=,&FONCTION=
.* MACRO FOR EACH FUNCTION IN IRXFLOC
&NOM DS ØH
STM R14,R12,12(R13)
MACFD

MACRO
&NOM MACFD &FCT=
.* MACRO FOR EACH FUNCTION IN IRXFLOC: HEADER
.* EXAMPLES:
.* $FCT1 MACFD
.* IN THIS CASE, NAME OF REXX FUNCTION IS $FCT1
.* $FCT1 MACFD FCT=FONCT1
.* IN THIS CASE, NAME OF REXX FUNCTION IS FONCT1
&A SETC '&FCT'
AIF ('&A' NE '').L1
&A SETC '&NOM'
.L1 ANOP
DC CL8'&A'
DC AL4(&NOM)
DC FL4'Ø'
DC CL8' '
DC CL8' '
MEXIT
MEND

Patrick Leloup
System Engineer
Credit Agricole de Loire Atlantique (France)  © Xephon 1998

Code published in DB2 Update is available from our Web site, www.xephon.com. Once you have registered, you can select an article containing code that you want e-mailed to you. Remember to have your copy of the issue containing the article with you when you access our Web site.
Programart has announced Version 2.0 of its Strobe MVS for Sysplex, as well as Version 4.0 of its APMpower.

The new releases include Y2K compliance, and support for international date and time formats. Additional enhancements for the two products include support for the most recent versions of language compilers and subsystems including OS/390 2.4, DB2 Version 5, CICS Transaction Server for OS/390 Release 1.2, COOL:Gen (Composer/IEF) Release 4.1a, CA-IDMS Release 14, ADABAS 6.1.3, IBM COBOL 2.1, PL/I 1.8, IBM C/C++ 3.5, and LE 1.8.

There’s also support for IBM’s BatchPipes for MVS (part of SmartBatch for OS/390), aimed at improving the performance of applications that use BatchPipes.

Also new are enhancements for users of DB2, C, CA-IDMS, ADABAS/NATURAL, or COOL:Gen. The Strobe DB2 feature has been enhanced to help customers pinpoint resource-consuming SQL statements or those exceeding desired service levels.

For further information contact:
Programart, 124 Mt Auburn St, University Place, Cambridge, MA 02138, USA.
Tel: (617) 661 3020.

Iona Technologies is shipping Orbix on MVS and OS/390, with support for DB2, CICS, and IMS along with support for COBOL. The product is aimed at building applications that make use of mainframe resources, transactions, applications, and data from anywhere in the network, thereby integrating mainframes with Unix, Windows, OS/2, OpenVMS, and Java kit.

It comes in two versions: the native version runs in MVS Version 5.2.2 and OS/390 Release 3, allowing MVS batch servers and clients in C++ and COBOL to be run as started tasks or batch jobs. There’s also a version for MVS OpenEdition, which takes advantage of the Posix compliance built into the operating system. Both support DB2 and COBOL, as well as integration with the IMS 5.1 and CICS 4.1.

For further information contact:
Iona Technologies, 201 Broadway, 3rd Floor, Cambridge, MA 02139-1955, USA.
Tel: (617) 679 0900.

IBM has announced Version 1.2 of its Maintenance 2000 tool for analysing MVS programs. The software provides an impact analysis that focuses on data flow, generates a cross-reference list for programs, jobs, copybook (%INCLUDE) files, and datasets, supports both batch and on-line applications, works on DB2, CICS, and DL/I applications, and searches for two-digit date items for system-wide impact analysis.

For further information contact your local IBM representative.

Xephon is holding its DB2 Update ‘98 conference in London on 11-12 June; subscribers may attend at preferential rates.

For further information about DB2 Update ‘98 contact Xephon on +44 1635 33823.