In this issue

3  Bimodal – what’s it all about?
6  Standard module structure and design
10 Handling ad hoc reports querying large volumes of data – a case study
13 JES subsystem information
25 Extended ISPF configuration utility
48 Managing uncatalogued datasets on SMS volumes
57 What LPAR?
58 Bar code printing from PL/I programs
74 MVS news
Bimodal – what’s it all about?

On 13 August 2002, IBM announced z/OS Version 1.4 in announcement letter 202-190. This announcement letter also introduced the concept of the z/OS Bimodal Migration Accommodation. At the time there was quite a flurry of interest in this new feature on the major IBM mainframe Internet newsgroup, IBM-MAIN, and subsequently it has become a regular topic of IBM-MAIN for subscribers to ask whether the z/OS Bimodal Migration Accommodation feature is appropriate to their z/OS implementation strategy.

This article discusses the Bimodal feature in the hope that it will clear up some of the misconceptions and uncertainties surrounding it.

So just what is the z/OS Bimodal Migration Accommodation feature? In essence, it’s IBM’s answer to a problem that has arisen because of the upgrade path necessitated by its hardware and software announcements over the last couple of years.

IBM ushered its mainframe clients into the 64-bit architecture world with the announcement of the zSeries processors and the z/OS operating system. At the time of that announcement, IBM apparently made a marketing, as opposed to a technical, decision that z/OS had to run in 64-bit mode on a zSeries processor. But z/OS could also run on an IBM S/390 Parallel Enterprise Server G5 or G6 model (or comparable server) in 31-bit mode. At the same time, OS/390 2.10 can run on a zSeries processor in either 31-bit or 64-bit mode.

Customers immediately questioned whether z/OS could not actually run on a zSeries processor in 31-bit mode, but the answer was that this would violate the terms of the z/OS licence.

For customers who have already begun migration to zSeries hardware, there is really no issue. They can run OS/390 2.10 on the zSeries and freely switch it between 31-bit and 64-bit modes.
using the ARCHLVL parameter. The typical migration route might be:

OS/390 2.10 (31-bit on S/390) --> OS/390 2.10 (31-bit on zSeries) --> OS/390 2.10 (64-bit on zSeries) --> z/OS 1.n (64-bit on zSeries)

But there exists another, apparently rather large, group of customers who find that they need to migrate to z/OS before they are in a position to upgrade their hardware to a zSeries. The problem then crystallizes in the situation where a customer already migrated to z/OS implements a hardware upgrade from an S/390 to a zSeries processor. Because of the aforementioned IBM requirement that z/OS can run only in 64-bit mode on a zSeries, the customer finds that the hardware upgrade automatically switches them into 64-bit mode. If the customer uncovers no problems with their IBM, ISV, and locally-developed software on switching to 64-bit mode, then all is well. But if they do find problems, which is, of course, highly unlikely with IBM software products, highly unlikely with ISV products as long as all ISVs have been thoroughly polled on their products’ 64-bit compliance, and anybody’s guess where locally-developed software is concerned, then the only backout route available (to a single footprint site anyway) is to turn the S/390 back on again.

The typical migration route for such a customer is supposed to be:

OS/390 2.10 (31-bit on S/390) --> z/OS 1.n (31-bit on S/390) --> z/OS 1.n (64-bit on zSeries)

Apparently a sufficient number of customers were unhappy with this migration route to make IBM to reconsider, and to announce the z/OS Bimodal Migration Accommodation feature. This allows customers to run z/OS Release 1.2 and upwards in 31-bit mode on a zSeries processor for a period of 6 months, so the migration route becomes:

OS/390 2.10 (31-bit on S/390) --> z/OS 1.n (31-bit on S/390) --> z/OS 1.n (64-bit on zSeries, optional fallback to 31-bit, Bimodal feature)
Note that the Bimodal feature in no way allows z/OS (or OS/390 for that matter) to be run in some kind of ‘quasi 64-bit’ mode on non-64-bit hardware. There is no such accommodation – 64-bit mode is available solely on 64-bit hardware.

Interestingly, although IBM initially emphasized the ‘fallback’ nature of the Bimodal feature, meaning that it was only intended to be used if problems in 64-bit mode were detected, many customers seem to feel more comfortable with the idea of deliberately invoking the Bimodal feature as a migration step thus:

OS/390 2.10 (31-bit on S/390) --> z/OS 1.n (31-bit on S/390) --> z/OS 1.n (31-bit on zSeries, Bimodal feature) --> z/OS 1.n (64-bit on zSeries)

Further confusion surrounds the 6-month period associated with the Bimodal feature, specifically with exactly when the period begins. The official line on this is “when z/OS is licensed to the server”, ‘the server’ being a 64-bit capable zSeries or equivalent. Note that it is when z/OS itself is licensed, not when the Bimodal feature is downloaded from IBM’s Web site, nor when it is installed on z/OS running on a 31-bit server. At the same time, it is important to note that z/OS should not be “licensed to the server” until it is ready to run there, or else the clock starts ticking. Fine tuning the actual dates involved should definitely be carefully discussed with IBM when contemplating activating the Bimodal feature.

Finally, it should be reiterated that the Bimodal feature is not available for z/OS 1.1. Since z/OS 1.1 is very close to being OS/390 2.10, which does have the ability to run in either 31-bit or 64-bit mode on zSeries hardware, one might guess that z/OS 1.1 can do likewise without requiring the Bimodal feature. Whether z/OS 1.1 can do this or not remains moot, because IBM has clearly stated that it is not a supported option, and running it like that would violate the terms of the licence.

In summary, the z/OS Bimodal Migration Accommodation feature is targeted at increasing the comfort level for customers who
wish to upgrade to z/OS before they upgrade to 64-bit capable hardware.

Patrick Mullen
Consultant (Canada) © Xephon 2003

Standard module structure and design

Previous articles (‘Structured design approach to program messages’, *MVS Update*, issue 199, April 2003, and ‘Structured design and program messages’, issue 200, May 2003) have discussed the concept of utilizing a standard message format, and how we could use a standard table structure to house the messages. We have also looked at how we could use a simple macro to facilitate looking up these messages in the table. This article will examine the overall module structure, and propose a simple model that can be followed to produce a standard load module. We feel that definite productivity gains can be realized by using this standardized approach, and with a programming team size of one or many.

![Figure 1: Example load module](image)

Executable code

Message table

© 2003. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.
In our previous discussion of a message table, we opted to make the table structure a CSECT with no executable code. This gave us a load module that looked like Figure 1.

We would like to propose extending this module with the addition of what we will refer to as the ‘eye catcher’ section of the program. The eye catcher will also be defined as a CSECT with no executable code. This eye catcher section can contain whatever information is meaningful to your site or installation. As a minimum, we would suggest that the date and time of the program assembly be included in this section. We have included a copy of a macro that we use to construct this eye catcher section. The macro name is $EDTEYEC.

$EDTEYEC

MACRO
&LABEL $EDTEYEC
**********************************************************************
.*
.* THIS MACRO IS USED TO HELP DEFINE A NON-EXECUTABLE SECTION
.* AT THE BEGINNING OF THE MODULE. WE WILL USE THIS MODULE
.* TO STORE DOCUMENTATION ABOUT THE CSECT AND THE ENVIRONMENT
.* IT WAS ASSEMBLED IN.
.*
.* EXAMPLES:
.*
.* SECTNAME $EDTEYEC
.*
**********************************************************************
.* SET UP SOME LOCAL VARIABLES.
**********************************************************************
LCLA &L_SYSASM
LCLA &L_SYSIN_DSN
LCLA &L_SYSIN_MEMBER
LCLA &L_SYSJOB
LCLA &L_SYSTEM_ID
LCLC &EYECACH
**********************************************************************
.* PRIME THE LOCAL VARIABLES
**********************************************************************
&EYECACH SETC 'EYEC'.'&SYSTIME'(1,2)
&EYECACH SETC '&EYECACH'.'&SYSTIME'(4,2)
&L_SYSASM SETA K'&SYSASM
&L_SYSIN_DSN SETA K'&SYSIN_DSN
&L_SYSIN_MEMBER SETA K'&SYSIN_MEMBER

As you can see from looking at the macro, we have chosen to include several items of information. Bear in mind we are not proposing that these are the only pieces of information that should be included – these are the data elements that we have chosen to include. You can take the $EDTEYEC macro and tailor it for your specific needs. We feel the real advantage to be gained here is the repeated use of a standardized structure. So what would a sample program look like using the $EDTEYEC macro? Here is a very simple example:

```assembly
$EDTEYEC
    My executable program code
```

© 2003. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.
Pictorially this would look like Figure 2.

If you should choose to utilize the proposed layout, you will need to specify an entry point for the executable code when you link edit the module. This can be accomplished by using the entry directive as shown below in the sample JCL:

```plaintext
//L1       EXEC PGM=IEWL,COND=(5,LT,C1),
        PARM='LIST,LET,XREF'
//SYSLMOD  DD  DISP=SHR,DSN=MY.LOAD.LIBRARY
//COPYMOD  DD  DISP=SHR,DSN=SYS1.LNKLIB
//SYSUT1   DD  UNIT=DISK,SPACE=(1024,(200,20))
//SYSPRINT DD  SYSOUT=*  
//SYSIN    DD  DSN=&&OBJ,DISP=(OLD, PASS)
//        DD  DDNAME=SYSIN
//        ENTRY OURMODUL
//        NAME OURMODUL(R)
/*
```
Handling *ad hoc* reports querying large volumes of data – a case study

**REQUIREMENTS**

Often during the day, *ad hoc* reports are requested by lots of users, on large volumes of history data.

Along with other details, a user keys in the history dates, which identifies the portion of the archival data that needs to be used for report generation.

Other technical requirements and standards include:

- A common database should be used to capture and maintain all user requests. CICS is our front end.
- Users’ requests should wait for their turn, in a queue, if the system is busy processing requests already entered.
- Batch jobs should be submitted only through job scheduler software.
- Procs and JCL will be kept in standard libraries, which should not be updated dynamically.

**SOLUTION**

Daily data is appended to a GDG flat file with the current date as a file key. A month’s data is pooled into one GDG version. No data updates/inserts are anticipated. So no DB2 tables/VSAM files
were used to maintain the history data. This also eliminated the maintenance overhead of large database/VSAM files.

Users’ input is captured using CICS screens and stored in a table with a timestamp indicating the entry time of the request. A user enters the date range of the history needed for the query. Next, CICS triggers a batch job to read the requests in the order they were entered. But this job can’t run a report right away! There is too great a volume of history data to be searched and we can’t afford to have JCL that will concatenate the complete set of history versions as an input file for every request run. The ad hoc report jobs would slow down the system considerably – if not halt it!

So the right portion of the history data, ie the correct subset of the input files, is identified (the GDG relative version numbers are derived) using the date range keyed in through CICS. Now the report JCL/Proc needs to be overwritten with this information. Bottom line, a set of JCL DD statements needs to be prepared by a program. This set needs to be attached to the report JCL before every run. This set is written to a PDS member (name it as an INCLUDE member). This is done with the batch job (we call it the INCLUDE job) that is triggered by the CICS screen. The INCLUDE job in turn triggers the report job. The INCLUDE verb was used in the report JCL to overwrite input files with history files. To enable the system to search, a PDS containing the INCLUDE member should be concatenated to the JCLLIB datasets of the report JCL.

```
JOB1
//job1 JOB ()
//step1 EXEC PGM=INCLD
//outfile DD DSN=incldpds(mem1), DISP=NEW
//incldpds(mem1)
//histfile DD DSN=history(-nn)

JOB2
//job2 JOB ()
```
RUNNING MULTIPLE REQUESTS

Each request is a logical unit of work with two stages:

1. Preparing an INCLUDE member (Job 1).
2. Submitting the respective report JCL (Job 2).

Each request requires this job pair to be run in the same order. Job 1 prepares a PDS member, which is used by Job 2 JCLLIB.

In any case, if there are system delays in expanding the report JCL or in using its INCLUDE member, and a user enters another request meanwhile, the INCLUDE member data may get updated and become inconsistent. Job schedulers will normally allow variables to be defined and set as switches, which will hold the INCLUDE JCL in the pipeline until their reporting JCL (Job 2) completes (using the INCLUDE member). These switches can be set/reset in the last step of the job. The logical unit described above would be:

1. Preparing an INCLUDE member (Job 1/step 1).
2. Setting SWITCH1 (Job 1/step 2).
3. Submitting respective report JCL (Job 2/step 1).
4. Resetting SWITCH1 (Job 2/step 2).

Thus multiple requests can be processed one after another in the order they were entered with the help of job schedulers. Note that a report program cannot be added as a step of an INCLUDE job.

Interestingly, CLIST-running ISPF panels can be used as a front end. A CLIST itself can write INCLUDE members and trigger report JCL. The front end can display a message if the INCLUDE member dataset was in use by another user/report JCL. Users
can try entering requests later. This eliminates the need for a separate INCLUDE job and scheduler switches, but doesn’t allow users to feed in multiple requests at once.

Common/separate INCLUDE members can be set for different reports. Appropriate switches in the job scheduler can make reports run simultaneously/sequentially and control system load.

Sreenivas Peddiraju
IT Analyst (India) © Xephon 2003

JES subsystem information

Obtaining specific information about the primary subsystem under which a given job is running can be cumbersome. You generally need to know something about the internal control block structure to get at the information you are trying to locate. What makes this an even more difficult process is that the control block structures for JES2 and JES3 are different and you cannot leverage data location techniques across the two environments.

IBM has tried to provide a little assistance in this area. There exists a directed subsystem interface (SSI) call that can be used to extract information about a specific subsystem. The upside of this subsystem call is that it can be made in a consistent fashion independently of the subsystem your program is running under (either JES2 or JES3). The downside is that the subsystem call currently returns only a small subset of information about the primary subsystem in control. If your information requirements can be satisfied by this subsystem call, it does provide a consistent method and allows your program to run successfully without concern for which JES environment may be in control.

The subsystem call I am referring to is the directed subsystem function call to SSI function code 54 – Subsystem Version Information (SSVI). Although you could build in support for function code 54 in your own subsystems, this article will focus
on SSI function code 54 because it specifically relates to JES2 or JES3.

The function code 54 subsystem function call returns information in the supplied SSOB (SubSystem Options Block). The SSVI function specific area of the SSOB consists of a fixed-length header component and a variable-length data area component.

The fixed-length header contains one particularly useful bit of information, which is referred to as the subsystem common name. The value returned for the subsystem common name is either JES2 or JES3, depending on the JES environment under which the program is running. This can be important, especially if your program is running under a secondary JES that has a subsystem name other than the traditional JES2 or JES3 we may expect to see.

The variable-length data includes (but is not limited to) information about the JES node name, the JES member name, whether or not four-digit device numbers are supported, and how output classes are currently set up. For a complete list of information available in the variable length data see section ‘3.1.5.16 – Format of the Variable Output Sections’ in the OS/390 MVS Using the Subsystem Interface manual.

A practical use for the information available from this subsystem function call is to determine viable external writer output classes. A traditional MVS external writer using the process SYSOUT subsystem function call (SSI function code 1) in a JES2 environment requires that you direct SYSOUT output to output classes with certain specific characteristics if the desired output is to be properly processed by the external writer. Specifically, a HOLD or PURGE output class should not be used as a target for SYSOUT output if the external writer is to operate as expected. A directed SSI function code 54 call can be made to extract subsystem information independently of the JES environment, and the information about output class characteristics and the JES environment can be obtained without regard for JES specifics.
The program provided with this article, SSINFO, uses the function code 54 subsystem call to obtain subsystem information. It displays information, via WTO, to the operator console, about the JES environment that SSINFO is running under as well as information about the output classes and their current characteristics.

Sample JCL to linkedit SSINFO:

```
//IEWL     EXEC  PGM=HEWLHØ96,PARM='XREF,LIST,MAP'
//SYSPRINT DD    SYSOUT=*  
//SYSUT1   DD    UNIT=SYSDA,SPACE=(CYL,(2,1))  
//OBJECT   DD    DSN=object.library,DISP=SHR
//SYSLMOD  DD    DSN=load.library,DISP=SHR
//SYSLIN   DD    *
    INCLUDE OBJECT(SSINFO)
ENTRY   SSINFO
NAME    SSINFO(R)
```

Sample JCL for running the SSINFO program:

```
//SSINFO   EXEC  PGM=SSINFO
//STEPLIB  DD    DSN=load.library,DISP=SHR
```

Any output produced by SSINFO is sent to the operator console. Sample console output generated from SSINFO:

```
SSINFO - ACTIVE SUBSYSTEM NAME IS JESA
SSINFO - COMMON SUBSYSTEM IS JES2
SSINFO - ADDITIONAL JESA SUBSYSTEM INFORMATION:
    JES_NODE='ESS     '
    JES_MEMBERNAME='JESA'
    DYNAMIC_OUTPUT='YES'
    INITIATOR_RESTART='YES'
    MULTIPLE_STCTSO='YES'
    FOUR_DIGIT_DEVNUMS='YES'
    AUTO_RESTART_MANAGER='YES'
    SAPI='YES'
    SAPI_CHARS='NO'
    CLIENT_PRINT='YES'
    TSO_SYSOUT_CLASS='H,K,O,X'
    WTR_SYSOUT_CLASS='A,B,C,D,E,F,G,I,J,L,M,N,P,Q,R,S,T,U,V,W,Y,Ø,1,2,3,4
                     6,7,8'
SSINFO - OUTPUT CLASS INFORMATION
    CLASS: A NON HELD
    CLASS: B NON HELD
    CLASS: C NON HELD
    CLASS: D NON HELD
    CLASS: E NON HELD
```
Notice that in this example the subsystem name is JESA, but the common subsystem name is JES2. In this case, the SSINFO utility was run under a secondary JES2 subsystem running as JESA.

The SSINFO program provides an example of the environment set-up necessary to make a directed subsystem call using the IEFSSREQ macro. It is also useful in providing information about the JES subsystem under which it is running. Try it out in your environment and check your results.

SSINFO.ASM

SSINFO CSECT
SSINFO AMODE 31
**THE SSINFO UTILITY IS USEFUL FOR DISPLAYING SUBSYSTEM INFORMATION FOR EITHER JES2 OR JES3 SUBSYSTEMS.**

**IT PROVIDES SAMPLE CODE FOR PERFORMING A DIRECTED SUBSYSTEM FUNCTION CALL USING THE IEFSSREQ MACRO.**

---

**USING SSINFO, R15**

**SET TEMPORARY ADDRESSABILITY**

**BEGIN**

**BRANCH TO PROGRAM LOGIC**

**BEGIN EQU ***

**DROP R15**

**STM R14, R12, 12(R13)**

**SAVE REGISTERS**

**LR R10, R15**

**COPY MODULE ADDRESS**

**LA R11, 4095(, R10)**

**SET UP SECOND ...**

**LA R11, 1(, R11)**

**BASE REGISTER**

**USING SSINFO, R15, R11**

**SET ADDRESSABILITY**

**LR R9, R13**

**COPY SAVEAREA ADDRESS**

**LR R2, R1**

**COPY PARAMETER ADDRESS**

**STORAGE OBTAIN, LENGTH=WORKLEN, LOC=ANY**

**LR R0, R1**

**SAVE THE STORAGE ADDRESS**

**LR R14, R15**

**COPY IT AGAIN**

**LR R3, R1**

**ONCE MORE FOR GOOD MEASURE**

**L R1, =A(WORKLEN)**

**GET THE LENGTH**

**XR R15, R15**

**CLEAR THE FILL BYTE**

**MVCL R0, R14**

**CLEAR WORKING STORAGE**

**ST R9, 4(, R3)**

**SAVE PREVIOUS SAVEAREA**

**LR R13, R3**

**COPY WORKING STORAGE ADDRESS**

**USING WORKAREA, R13**

**SET ADDRESSABILITY**

**L R15, 16**

**GET CVT ADDRESS**

**L R15, Ø(, R15)**

**GET TCB/ASCB AREA ADDRESS**

**L R15, 4(, R15)**

**GET CURRENT TCB**

**L R15, TCBJSCB-TCB(, R15)**

**GET JSCB ADDRESS?**

**L R15, JSCBSSIB-I EIZJSCB(, R15)**

**GET SSIB ADDRESS?**

**MVC SSNMSAVE(8), =8C**

**INITIALIZE THE AREA**

**MVC SSNMSAVE(4), SSIBSSNM-SSIB(R15)**

**MV SSNAME**

**LA R1, SSNMSAVE**

**GET STARTING ADDRESS**

**XR R15, R15**

**CLEAR COUNTER**

**LENLP1 EQU ***

**C R15, =F' 8'**

**MAX LENGTH?**

**BNL LENEND1**

**YES - WE'RE DONE WITH LENGTH**

**CLI Ø(R1), C' '**

**A BLANK?**

**BE LENEND1**

**YES - WE'RE DONE WITH LENGTH**

**CLI Ø(R1), X'00'**

**A NULL?**
BE  LENEND1  YES - WE'RE DONE WITH LENGTH
LA  R1, 1,(R1)   POINT TO NEXT BYTE
LA  R15, 1,(R15)  ADD ONE TO COUNT
B  LENLP1  CHECK NEXT BYTE

LENEND1  EQU  *
ST  R15, SSNMLEN  SAVE THE LENGTH
MVC  WTOWK(WTO1LN), WTO1LS  MOVE IN THE MESSAGE MODEL
MVC  WTOWK+4(120), MSG1  MOVE IN MESSAGE CONTENT
MVC  WTOWK+4+34(8), SSNSAVE  MOVE IN SUBSYSTEM NAME
WTO  MF=(E, WTOWK)  ISSUE THE MESSAGE

***********************************************************************
LA  R2, SSOBHSIZ  GET SSOB HEADER LENGTH
LA  R14, SSVIFSIZ  GET SSVI FIXED LENGTH SIZE
ST  R14, VISIZE  SAVE SIZE
AR  R2, R14  CALCULATE A STARTING LENGTH
ST  R2, SSOBLN  SAVE LENGTH VALUE

SSCALL  EQU  *
L  R2, SSOBADDR  GET SSOB ADDRESS
STORAGE OBTAIN, LENGTH=(R2), LOC=ANY
ST  R1, SSOBADDR  SAVE STORAGE ADDRESS
LR  R0, R1  COPY STORAGE ADDRESS
LR  R14, R1  COPY STORAGE ADDRESS
L  R1, SSOBLN  GET LENGTH
XR  R15, R15  SET FILL BYTE
MVCL  R0, R14  CLEAR THE STORAGE

***********************************************************************
L  R2, SSOBADDR  GET SSOB ADDRESS
USING SSOBEGIN, R2  SET ADDRESSABILITY
XC  SSOB(SSOBHSIZ), SSOB  CLEAR THE SSOB
MVC  SSOBID(4), =C'SSOB'  SET SSOB ID
MVC  SSOBFUNC(2), =AL2(SSOBSSVI)  SET FUNCTION ID
MVC  SSOBLEN(2), =AL2(SSOBHSIZ)  SET SSOB HEADER SIZE
LR  R3, R2  GET SSOB ADDRESS
AH  R3, SSOBLN  CALCULATE SSVI ADDRESS
USING SSVI, R3  SET ADDRESSABILITY
ST  R3, SSOBDNDV  SAVE SSVI ADDRESS

***********************************************************************
XC  SSVIHEAD(SSVI MSIZ), SSVIHEAD  CLEAR THE SSVI
L  R15, VISIZE  GET SSVI AREA LENGTH
STCM  R15, 0011', SSVI LEN  SAVE THE LENGTH
MVI  SSVIVER, SSVICVER  MOVE CURRENT VERSION NUMBER IN
MVC  SSVIV ID, =A(SSVIV ID)  SAVE THE IDENTIFIER
LR  R1, R2  GET SSOB ADDRESS
O  R1, ' 80000000'  TURN ON X'80' BIT
ST  R1, SSOBPTR  SAVE SSOB PTR
LA  R1, SSOBPTR  POINT TO SSOB PTR
IEFSSREQ,  MAKE SUBSYSTEM REQUEST
CLC  SSOBRETN(4), =F' 0'  SUBSYSTEM DATA RETURNED?
BE  SSDDATAOK  YES - THINGS ARE GOOD
CLC  SSOBRETN(4), =F' 8'  DATA BUFFER TOO SMALL?
BNE SSVIERR NO - ISSUE AN ERROR

L R1,SSOBADDR GET STORAGE ADDRESS
L R0,SSOBLN GET STORAGE LENGTH
CLC VISIZE+2(2),SSVI RLEN CURRENT SZ & REQUIRED SZ EQUAL?
BE SSVIERR YES - ISSUE AN ERROR
XC VISIZE(4),VISIZE CLEAR CURRENT SSVI LENGTH
MVC VISIZE+2(2),SSVI RLEN COPY THE REQUIRED SIZE
STORAGE RELEASE,LENGTH=(R0),ADDR=(R1)
LA R2,SSOBSIZE GET SSOB HEADER LENGTH
L R14,VISIZE SSVI REQUIRED SIZE
AR R2,R14 CALCULATE A STARTING LENGTH
ST R2,SSOBLN SAVE LENGTH VALUE
B SSCALL TRY AGAIN

SSDATAOK EQU *
MVC WTOWK(WTO1LN),WTO1LS MOVE IN THE MESSAGE MODEL
MVC WTOWK+4(12Ø),MSG2 MOVE IN MESSAGE CONTENT
MVC WTOWK+4+29(8),SSVICNAM MOVE IN SUBSYSTEM COMMON NAME
WTO MF=(E,WTOWK) ISSUE THE MESSAGE
MVC CNAMSAVE(8),SSVICNAM SAVE SUBSYSTEM COMMON NAME

L R7,SSVISDOF GET OFFSET OF SYSTEM DEFINED INFO
LTR R7,R7 ANY SYSTEM DEFINED INFO?
BZ RETURN NO - WE'RE DONE
LA R7,Ô(R7,R3) POINT TO SYSTEM DEFINED INFO
CLC Ô(2,R7),=H'Ô' ANY DATA?
BE RETURN NO - WE'RE DONE
XR R8,R8 CLEAR R8
ICM R8,'ÔØ11',Ô(R7) GET DATA LENGTH
LA R7,2{(R7) POINT TO FIRST KEYWORD
LA R8,Ô(R8,R7) POINT TO END OF DATA

MVC WTOWK(WTO1LN),WTO1LS MOVE IN THE MESSAGE MODEL
MVC WTOWK+4(12Ø),MSG6 MOVE IN MESSAGE CONTENT
MVC WTOWK+4+2Ò(4),SSNMSAVE MOVE IN SUBSYSTEM NAME
WTO MF=(E,WTOWK) ISSUE THE MESSAGE

* *
* R7 POINTS TO THE START OF THE DATA. *
* R8 POINTS TO THE END OF THE DATA. *
* *
* KEYWORDS CAN BE RELIABLY LOCATED USING THE FOLLOWING RULE: *
* *
* ,KEYWORD= *
* *
* WE ARE AT THE START OF THE DATA. THE FIRST KEYWORD WILL END *
* AT THE FIRST '=' CHARACTER. POSITION PAST THE '=' AND SEARCH *
* FOR THE NEXT KEYWORD LOCATION. ONCE WE DETECT THE NEXT *
* KEYWORD, WE KNOW WHERE THE DATA FOR THE PREVIOUS KEYWORD ENDS. *
NEXTKWD EQU *
    LR R9, R7                  COPY R7
KWDENDLP EQU *
    CR R9, R8                  END OF DATA?
    BNL SYSINFOE               YES - NOT NORMAL
    CLI Ø(R9), C='='            KEYWORD END?
    BE KWDNEXT                YES - FIND NEXT KEYWORD
    LA R9, 1(R9)               POINT TO NEXT DATA BYTE
    B KWDENDLP                 CHECK IT OUT
KWDNEXT EQU *
    LA R9, 1(R9)               POINT TO KEYWORD DATA
KWDNXTLP EQU *
    CR R9, R8                  END OF DATA?
    BNL KWDLAST                YES - WE HAVE THE LAST KEYWORD
    CLI Ø(R9), C='='            KEYWORD END INDICATOR?
    BNE KWDNXTØ5               NO - CONTINUE SEARCH
    LA R9, 1(R9)               POINT TO NEXT DATA BYTE
    CLI Ø(R9), C三是"        START OF KEYWORD DATA?
    BNE KWDNXTLP               NO - CHECK AGAIN
KWDNXTØ5 EQU *
    BCTR R9, Ø                  BACK UP ONE BYTE
    CLI Ø(R9), C='            COMMA?
    BE KWDNEXTE                YES - WE'VE FOUND NEXT KEYWORD
    B KWDNXTØ5                 TRY AGAIN
KWDNXT1Ø EQU *
    LA R9, 1(R9)               POINT TO NEXT DATA BYTE
    B KWDNXTLP                 TRY AGAIN
KWDLAST EQU *
    OI PARSEFLG, LAST          SET LAST KEYWORD FLAG
KWDLASTL EQU *
    BCTR R9, Ø                  POINT TO PREVIOUS BYTE
    CLI Ø(R9), C='        A BLANK?
    BE KWDLASTL                YES - CHECK PREVIOUS
    CLI Ø(R9), X'00'        NULL?
    BE KWDLASTL                YES - CHECK PREVIOUS
    LA R9, 1(R9)               POINT PAST END OF DATA
KWDNEXTE EQU *
    LA R7, 1(R, R7)            SKIP PAST COMMA
    LR R15, R9                 GET END ADDRESS
    SR R15, R7                 GET LENGTH
    C R15, =F'118'              MAX LENGTH OK?
    BNH DATALNOK                YES - LENGTH IS OK
    L R15, =F'118'              SET MAX LENGTH
DATALNOK EQU *
    CLC Ø(7, R7), =C'EXW_SYS'  JES3 EXTERNAL WRITER CLASS?
    BNE CHKSYSO2                NO - CHECK NEXT SYSOUT TYPE
    ST R7, EYSOUTA             SAVE KEYWORD START ADDRESS
    B KWDOUT                  GO ISSUE MESSAGE
CHKSYSO2 EQU *
CLC Ø(7,R7),=C' TSO_SYS' TSO HELD OUTPUT CLASS?
BNE CHKSYSO3 NO - CHECK NEXT SYSOUT TYPE
ST R7,TYSYSOUTA SAVE KEYWORD START ADDRESS
B KWDOUT GO ISSUE MESSAGE

CHKSYSO3 EQU *
CLC Ø(7,R7),=C' WTR_SYS' NON HELD OUTPUT CLASS?
BNE KWDOUT NO - NOT ONE WE'RE INTERESTED IN
ST R7,WSYSOUTA SAVE KEYWORD START ADDRESS
B KWDOUT GO ISSUE MESSAGE

KWDOUT EQU *
***********************************************************************
BCTR R15,Ø REDUCE BY ONE
MVC WTOPK(WTO1LN),WTO1LS MOVE IN THE MESSAGE MODEL
MVC WTOPK+4(120),MSG7 MOVE IN MESSAGE CONTENT
EX R15,DATAMVC MOVE THE DATA
WTO MF=(E,WTOPK) ISSUE THE WTO
***********************************************************************
TM PARSEFLG,LAST ARE WE DONE?
BO LISTSYSO YES - LIST OUTPUT CLASSES
LR R7,R9 POINT TO NEXT KEYWORD
B NEXTKWD GO CHECK IT OUT

***********************************************************************
LI STSYSO EQU *
CLC ESYSOUTA(4),=F' Ø' EXTERNAL WRITER CLASS DATA?
BNE CLASDATA YES - WE HAVE CLASS DATA
CLC TYSYSOUTA(4),=F' Ø' TSO HELD CLASS DATA?
BNE CLASDATA YES - WE HAVE CLASS DATA
CLC WSYSOUTA(4),=F' Ø' NON HELD CLASS DATA?
BNE CLASDATA YES - WE HAVE CLASS DATA
B RETURN NO CLASS DATA

CLASDATA EQU *
MVC WTOPK(WTO1LN),WTO1LS MOVE IN THE MESSAGE MODEL
MVC WTOPK+4(120),MSG3 MOVE IN MESSAGE CONTENT
WTO MF=(E,WTOPK) ISSUE THE MESSAGE

***********************************************************************
LA R5,CLASSLST GET CLASS LIST ADDRESS
CLASSLP EQU *
CLI Ø(R5),X' FF' END OF LIST?
BE RETURN YES - WE'RE DONE

***********************************************************************
L R9,ESYSOUTA GET EXTERNAL WRITER AREA ADDRESS
LTR R9,R9 ANY DATA?
BZ NXTTYPE2 NO - CHECK NEXT TYPE
LA R9,18(,R9) POINT TO CLASS DATA

TYPelp1 EQU *
CLI Ø(R9),C''' END QUOTE?
BE NXTTYPE2 YES - CHECK NEXT TYPE
CLC Ø(1,R9),Ø(R5) A CLASS MATCH?
BE CLASMCH1 YES - SET MATCH TYPE 1 VALUES
CLI 1(R9),C''''    END QUOTE?
BE NXTTYPE2    YES - CHECK NEXT TYPE
LA R9,2(R9)    POINT TO NEXT CLASS VALUE
B TYPELP1    GO CHECK IT OUT

CLASMCH1 EQU *
  MVC CLASSID(20),=CL20'JES3 EXTERNAL WRITER'
  B CLASSOUT    GO ISSUE THE MESSAGE
***********************************************************************

NXTTYPE2 EQU *
  L R9,TSYSOUTA    GET TSO AREA ADDRESS
  LTR R9,R9    ANY DATA?
  BZ NXTTYPE3    NO - CHECK NEXT TYPE
  LA R9,18(R9)    POINT TO CLASS DATA

TYPELP2 EQU *
  CLI Ø(R9),C''''    END QUOTE?
  BE NXTTYPE3    YES - CHECK NEXT TYPE
  CLC Ø(1,R9),Ø(R5)    A CLASS MATCH?
  BE CLASMCH2    YES - SET MATCH TYPE 2 VALUES
  CLI 1(R9),C''''    END QUOTE?
  BE NXTTYPE3    YES - CHECK NEXT TYPE
  LA R9,2(R9)    POINT TO NEXT CLASS VALUE
  B TYPELP2    GO CHECK IT OUT

CLASMCH2 EQU *
  MVC CLASSID(20),=CL20'TSO HELD'
  B CLASSOUT    GO ISSUE THE MESSAGE
***********************************************************************

NXTTYPE3 EQU *
  L R9,WSYSOUTA    GET NON HELD AREA ADDRESS
  LTR R9,R9    ANY DATA?
  BZ PRGCLASS    NO - MUST BE A PURGE CLASS
  LA R9,18(R9)    POINT TO CLASS DATA

TYPELP3 EQU *
  CLI Ø(R9),C''''    END QUOTE?
  BE PRGCLASS    YES - MUST BE A PURGE CLASS
  CLC Ø(1,R9),Ø(R5)    A CLASS MATCH?
  BE CLASMCH3    YES - SET MATCH TYPE 3 VALUES
  CLI 1(R9),C''''    END QUOTE?
  BE PRGCLASS    YES - MUST BE A PURGE CLASS
  LA R9,2(R9)    POINT TO NEXT CLASS VALUE
  B TYPELP3    GO CHECK IT OUT

CLASMCH3 EQU *
  MVC CLASSID(20),=CL20'NON HELD'
  B CLASSOUT    GO ISSUE THE MESSAGE
***********************************************************************

PRGCLASS EQU *
  MVC CLASSID(20),=CL20'PURGE'
  B CLASSOUT    GO ISSUE THE MESSAGE
***********************************************************************

CLASSOUT EQU *
  MVC WTOR(WTO1LN),WTO1LS    MOVE IN THE MESSAGE MODEL
MVC WTOWK+4(120), MSG4 MOVE IN MESSAGE CONTENT
MVC WTOWK+4+16(1), Ø(R5) MOVE IN THE CLASS
MVC WTOWK+4+19(20), CLASSID MOVE IN THE CLASS INFORMATION

NOEXTIND EQU *
WTO MF=(E, WTOWK) ISSUE THE WTO

NXTCLASS EQU *
LA R5, 1(, R5) POINT TO NEXT CLASS
B CLASSLP GO CHECK IT OUT

*******************************************************************************
RETURN EQU *
ICM R1, B'1111', SSOBADDR GET SSOB ADDRESS
BZ NOSSOB IF NO SSOB DON'T RELEASE
L R0, SSOBLEN GET STORAGE LENGTH
STORAGE RELEASE, LENGTH=(R0), ADDR=(R1)

NOSSOB EQU *
L R3, SAVEAREA+4 SAVE OLD SAVEAREA ADDRESS
LR R1, R13 GET WORKING STORAGE ADDRESS
STORAGE RELEASE, LENGTH=WORKLEN, ADDR=(R1)
LR R13, R3 COPY OLD SAVEAREA ADDRESS
LM R14, R12, 12(R13) RESTORE THE REGISTERS
XR R15, R15 CLEAR R15
BR R14

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

SSVIERR EQU *
WTO 'SSINFO - UNABLE TO OBTAIN SUBSYSTEM INFORMATION.', X
ROUTCODE=(1), DESC=(6)
B RETURN GO HOME

*******************************************************************************
SYSINFOE EQU *
WTO 'SSINFO - INVALID END OF SUBSYSTEM INFORMATION DATA DETECTED', X
ROUTCODE=(1), DESC=(6)
B RETURN GO HOME

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

NOSSC VT EQU *
MVC WTOWK(WTO1LN), WTO1LS MOVE IN THE MESSAGE MODEL
MVC WTOWK+4(120), MSG5 MOVE IN MESSAGE CONTENT
MVC WTOWK+4+40(4), SSNMSAVE MOVE IN SUBSYSTEM NAME
WTO MF=(E, WTOWK) ISSUE THE MESSAGE
B RETURN GO HOME

*******************************************************************************
DATAMVC MVC WTOWK+4+2(*-*), Ø(R7) MOVE SUBSYSTEM INFO DATA

*******************************************************************************
CLASSLST DC C'ABCDEFGHIJKLMNOPQRSTUVWXYZØ123456789', X'FF'

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

WT01LS WTO ' 1 2 3 4 5 X
6 7 8 9 0 1 X
2', ROUTCODE=(1), DESC=(6), MF=L

WT01LN EQU '.WT01LS
MSG1 DC CL120' SSINFO - ACTIVE SUBSYSTEM NAME IS XXXXXXXX'
MSG2     DC    CL1200'SSSINFO - COMMON SUBSYSTEM IS XXXX   '
MSG3     DC    CL1200'SSSINFO - OUTPUT CLASS INFORMATION   '
MSG4     DC    CL1200'                 CLASS: X                  '
MSG5     DC    CL1200'SSSINFO - SSCVT NOT LOCATED FOR SUBSYSTEM XXXX   '
MSG6     DC    CL1200'SSSINFO - ADDITIONAL XXXX SUBSYSTEM INFORMATION:'
MSG7     DC    CL1200'                  LTORG   '

WORKAREA DSECT
SAVEAREA DS    18F
SSOBPTR DS    F
CLASS DS    CL1
PARSEFLG DS    XL1
COMMA EQU   X'80'
QUOTE EQU   X'40'
BLANK EQU   X'20'
EQUAL EQU   X'10'
LAST EQU   X'08'

DBL1 DS    2D
DBL2 DS    2D
ESYSOUTA DS    F
TSYSOUTA DS    F
WSYSOUTA DS    F
CLASSID DS    CL20
CNAMESAVE DS    CL8
SSNMSAVE DS    CL8
SSNMLLEN DS    F
SSOBADDR DS    F
SSOBLN DS    F
VISIZE DS    F

WTOWK DS    DD, CL(WTOLN)

WORKLEN EQU   '* WORKAREA
CVT DSECT=YES

IEFJESCT
IEFJSCVT
IEESMCA
IKJTCB
IEZJSCB
IEFJSSIB
IEFSSO8H
IEFSSVI DSECT=YES

R0 EQU   0
R1 EQU   1
R2 EQU   2
R3 EQU   3
R4 EQU   4
R5 EQU   5
R6 EQU   6
R7 EQU   7
R8 EQU   8
R9 EQU   9
Extended ISPF configuration utility

Would you like to be able to configure your ISPF to your own standards? Here’s the code for an ISPF dialog which will let you do it.

INTRODUCTION

There can (optionally) be an ISPF configuration module, called ISPCFIGU, allocated in your TSO session to control your ISPF set-up. Such a module is usually created by the systems programmers and shared by all users.

Since ISPF Version 4.8 (which was part of OS/390 2.8), IBM has supplied an ISPF Configuration Utility dialog for that. The dialog enables you to create a file with keywords for the desired configuration, then generate a new module to implement that configuration.

This article contains code to extend the functionality of IBM’s dialog. It adds an option 0 to the standard panel so that it looks like this:

Extended ISPF Configuration Utility           BROWNR1 on SYS1
Option ===>  
0  Create/View Keyword File from ACTIVE Configuration
1  Create/Modify Settings and Regenerate Keyword File
2  Edit Keyword File Configuration Table
3  Verify Keyword Table Contents
4  Build Configuration Table Load Module
5  Convert Assembler Configuration Table to Keyword File
6  Build SMP/E USERMOD
Keyword File Data Set
  Data Set . . . BROWN1.ISPCFG
  Member . . . KEY01
Configuration Table Assembler Source Data Set
  Data Set . . . BROWN1.ISPOBJ
  Member . . . ASM01          Debug . . DEBUG

Output File Content for Keyword File
  1. Include only non-default values
  2. Include defaults as comments
  3. Include all values

The problem with IBM’s dialog is that it cannot show you the current (active) configuration as the BASE for your desired changes; it can start only from the ISPF defaults, or else you must get a copy of the configuration file used by the systems programmers.

My new option ‘0’ can generate a keyword file directly from the ACTIVE ISPF configuration module. It reads the module in your TSO session’s storage, generates a matching Assembler source file, then converts that to an equivalent keyword file and shows it to you in VIEW mode – which can be a simple way to check exactly what your current configuration is.

The IBM dialog can then use the resultant keyword file to show your actual configuration in a series of standard ISPF panels.

The IBM dialog also allows you to modify it and use it as input to build a new ISPF configuration module. That can be very useful if you want to have some special configuration parameters of your own, but otherwise remain consistent with your site’s existing standards, and you don’t have a copy of the last-used keyword file. For example you might like to have Edit RECOVERY ON, or DISPLAY_SEQUENCE_NUMBERS OFF or SHOW_PFK OFF as defaults, or you could specify your own User Command Table name. There are many possibilities.

THE CODE
The code is one panel, three REXX EXECs, and one Assembler module:
• Panel JSPPCONF – a modified version of IBM’s ISPPCONF, which includes the extra Option 0 – ‘Create/View Keyword File from ACTIVE Configuration’.

• REXX JSPCCONF – a modified version of IBM’s ISPCCONF, which displays the new panel and adds an extra selection option invoking REXX JSPCATAB.

• REXX JSPCATAB – creates a keyword file by:
  – calling Assembler module LOADPNT to find the storage address of the ISPCFIGU configuration module
  – using the standard IBM skeleton ISPCSkel to map ISPCFIGU and generate ASM statements
  – calling REXX JSPCCONV, a modified version of IBM’s ISPCCONV, to convert the ASM statements to a keyword file, and then view it.

To save space I have not included any HELP panels in this article, but it would be a good idea to create a modified ISPPC000 panel with an extra option ‘0’ pointing to a new help panel describing what it does.

PANEL JSPPCONF

Note that this panel includes many non-printable hex bytes, which are used exactly as in the original panel (but they are often not correct after being copied from MVS to a PC and back again). Therefore, implement this by comparing it with a standard ISPPCONF panel and making the changes manually, as detailed in the comment box at the start.

```cpp
)PANEL KEYLIST(ISRSNAB, ISR)
/* ------------------ CHANGES from IBM panel ISPPCONF ------------------ */
/* Window width increased from 76 to 80 */
/* Heading changed to "Extended" ISPF Configuration Utility */
/* &ZUSER on &ZSYSID added, so this is obviously not a standard panel */
/* option 0 added: Create/View Keyword file from ACTIVE Configuration */
/* line above 'Keyword File Data Set' deleted */
/* )PROC change verification for ZCMD to allow '0' */
/* )PNTS add extra field for option 0, all VAL(x) values adjusted */
/* --------------------------- */
```

Extended ISPF Configuration Utility

	Option ===> Z
SAREA38
)
AREA SAREA38

0
Create/View Keyword File from ACTIVE Configuration

1
Create/Modify Settings and Regenerate Keyword File

2
Edit Keyword File Configuration Table

3
Verify Keyword Table Contents

4
Build Configuration Table Load Module

5
Convert Assembler Configuration Table to Keyword File

6
Build SMP/E USERMOD

--Keyword File Data Set--

ZZZZData Set . . . Z

ZZZZMember . . . .Z

--Configuration Table Assembler Source Data Set--

ZZZZData Set . . . Z

ZZZZMember . . . .Z

ZZZ 1. Include only non-default values
ZZZ 2. Include defaults as comments
ZZZ 3. Include all values

)INIT
.ZVARS = ' (ZCMD ZCNVKWD ZCNVKWDM ZCNVSRCE ZCNVSRCM SHOWCTP)'
.HELP = ISPPOO
SHOWCTP = '
.ATTR(SHOWCTP) = 'CSRGRP(99) RADIO(ON)'
IF (&ZCNVOPT = 'NEW') &SHOWCTP = '1'
IF (&ZCNVOPT = 'CHG') &SHOWCTP = '2'
IF (&ZCNVOPT = 'ALL') &SHOWCTP = '3'
IF (&SHOWCTP = &Z)
SHOWCTP = 2
)PROC
SHOWCTP = TRANS(DTRUNC(&ZCMD,'.'))
X,EXIT

VER(&ZCNVKWD DSNAMEPQ)
VER(&ZCNVKWD, NONBLANK)
VER(&ZCNVKWDM NAME)
VER(&ZCNVKWDM, NONBLANK)
VER(&ZCNVSRCE DSNAMEPQ)
VER(&ZCNVSRCE, NAME)
&ZCHAR = TRANS( &DEBUG, 1)
IF (&ZCHAR = D) &DEBUG = DEBUG
ELSE            &DEBUG = &Z
VER(&SHOWCTP RANGE, 1, 3)
IF (&SHOWCTP = '1') &ZCNVOPT = 'NEW'
IF (&SHOWCTP = '2') &ZCNVOPT = 'CHG'
IF (&SHOWCTP = '3') &ZCNVOPT = 'ALL'
IF (&ZCMD = Ø | &ZCMD = 1 | &ZCMD = 5) /* Ron, added option Ø */
  VER( &SHOWCTP, NB)
)HELP
FIELD(ZCNVKWD) PANEL(ISPOO1)
FIELD(ZCNVKWDM) PANEL(ISPOO2)
FIELD(ZCNVSRCE) PANEL(ISPOO3)
FIELD(ZCNVSRCM) PANEL(ISPOO4)
FIELD(SHOWCTP) PANEL(ISPOO4)
)PNTS
/* Ron, added option Ø */
FIELD(ZPS01001) VAR(ZCMD) VAL(Ø)
FIELD(ZPS01002) VAR(ZCMD) VAL(1)
FIELD(ZPS01003) VAR(ZCMD) VAL(2)
FIELD(ZPS01004) VAR(ZCMD) VAL(3)
FIELD(ZPS01005) VAR(ZCMD) VAL(4)
FIELD(ZPS01006) VAR(ZCMD) VAL(5)
FIELD(ZPS01007) VAR(ZCMD) VAL(6)
)END

REXX JSPCCONF
Prepare this by taking the code here, then add the last part of the
standard ISPCCONF EXEC, as detailed in the comment at the end.

```rexx
/*REXX****************************************************************/
/* EXEC NAME := JSPCCONF   (based on IBM EXEC: ISPCCONF) */
/*                                                                   */
/* DESCRIPTIVE_NAME := Configuration table main driver             */
/*                                                                   */
/* FUNCTION    = Main driver EXEC for the configuration table dialog */
/*                                                                   */
/* MODIFICATIONS:                                                    */
/* 1) use panel JSPPCONF   (instead of IBM panel: ISPPCONF)          */
/* 2) add extra option Ø, to invoke %JSPCATAB                       */
/*********************************************************************/
Trace 0
Parse Upper Arg debug
Address ispexec
'CONTROL ERRORS RETURN'
'VGET (ZCNVSRCZ ZCNVSRM ZCNVKWD ZCNVKWD2 ZCNVOPT) PROFILE'
'VGET (ZCNVLOAD ZCNVOBJ) PROFILE'
m = '

c = 'ZCMD'
local_zcmd = ''
dsrc = Ø
display_rc = Ø
display_rc2 = Ø
Do While display_rc = Ø
  If m ¬= ' ' Then
    msg = 'MSG('m')'
  Else
    msg = ''
  If dsr > Ø Then
    zcmd = local_zcmd
    'DISPLAY PANEL(JSPPCONF)' msg 'CURSOR('c')'        /** Ron **/
    display_rc = rc
  m = ''
  c = 'ZCMD'
  local_zcmd = zcmd
  zcmd = ''
  dsr = Ø
  If display_rc = Ø Then
    Do
      'VPUT (ZCNVSRCZ ZCNVSRM ZCNVKWD ZCNVKWD2 ZCNVOPT) PROFILE'
        Select
          When local_zcmd = ' ' Then
            m = 'ISR292'
          /* ---- start of lines added by Ron ------------------- */
          When local_zcmd = Ø Then
            Do
              m = 'JSPC255'
```
Address TSO 'ISPCATAB' debug
If rc > Ø Then
  m = 'ISPC256'
  'VGET (ZCNVCSR) PROFILE'
c = zcnvcsr
End
/* ---- end of lines added by Ron ------------------------ */
When local_zcmd = 1 Then
  m = 'ISPC250'
  Call verify_keyword_file
***** ... the rest is the same as the original ISPCCONF EXEC *****

REXX JSPCATAB

/***********************************************>
REXX <*********************************************/
/* JSPCATAB: Build keyword file for the active ISPF config table */
/* */
/* 1 Program LOADPNT used to find address of ISPCFIGU module. */
/* 2 Find standard IBM skeleton ISPCSKEL and use it to map the ISPF */
/*   configuration module ISPCFIGU and generate ASM statements. */
/* 3 Call EXEC JSPCCONV to convert ASM statements to keyword file. */
/* */
/* Written: 29 May 2002    Last Updated: 17 Feb 2003    by: Ron Brown */
/***********************************************<
Trace 0
Parse Upper Arg debug
numeric digits 20
Address ISPEXEC
skel = 'ISPCSKEL'     /* standard IBM skeleton in SISPSLIB library */
/* Find location of ISPF configuration module in storage */
pgm_addr = LOADPNT('ISPCFIGU')     /* get pgm load address */
dec_addr = X2D(C2X(pgm_addr))
If dec_addr > Ø Then /* if program is loaded .. */
  offset = Ø
Else Do
  Call ISPF_MSG('No ISPCFIGU module found, ISPF is running with',
               'the default configuration.
               'You can use option 1 to generate a default',
               'Keyword File.' )
  Exit 4
End
/* Load skeleton into 'inskel.' array, and allocate output file */
skel_rc = GET_SKELETON()     /* load skeleton into 'inskel.' */
If skel_rc > Ø Then Exit 4

asma_rc = ALLOC_OUTPUT()  /* allocate file for asm output */
If asma_rc > Ø Then Exit 4
/* Process the skeleton, creating output in 'outasm.' array */
/*----------------------------------------------------------*/
j = Ø
Do i = 1 To inskel.Ø
Select
  When Left(inskel.i,3) = ')'CM' Then Iterate  /* ignore comment */
  When Word(inskel.i,1) <> 'DC',
    & Word(inskel.i,2) <> 'DC' Then Do  
j = j + 1
  outasm.j = inskel.i
  End
  Otherwise Call PROCESS_DC
End
/*------------------------------------------------------*/
/* write output to disk, and (optionally) VIEW the file */
/*------------------------------------------------------*/
Address TSO "EXECIO * DISKW ICONFASM (FINIS STEM outasm."
write_rc = rc
If write_rc <> Ø Then Do
  Call ISPF_MSG('Unable to write assembler output file' asmout, 
    'return code =' write_rc)
End
Else If debug = 'DEBUG' Then  /* if DEBUG - view assembler file */
  Call VIEW_ASM_FILE
Address TSO "FREE FILE(ICONFASM)"
/* Call EXEC to convert the Assembler source to a Keyword File */
/*-------------------------------------------------------------*/
If write_rc = Ø Then
  Address TSO "%JSPCCONV"
Exit  /* That's all folks */
/* set up an ISPF (long) message */
/*-------------------------------------------------------------*/
ISPF_MSG:
Parse Arg ZERRLM
ZERRSM = ''
ZERRALRM = 'YES'
ZERRHM = '*'
"SETMSG MSG(ISRZØØ2)"
Return
/* load the skeleton into array inskel. */
/*-------------------------------------------------------------*/
GET_SKELETON:
"FTOPEN TEMP"  /* output -> DDNAME in ZTEMPN variable */
"FTINCL" skel "NOFT" /* no variable substituition or interpretation */
incl_rc = rc
"FTCLOSE"
If incl_rc > Ø Then Do
   Call ISPF_MSG('Unable to read' skel 'skeleton. FTINCL',
                  'return code =" incl_rc )
   Return incl_rc
End
"VGGET ZTEMPN"
Address TSO "EXECIO * DISKR" ZTEMPN "(FINIS STEM inskel."
read_rc = rc
If read_rc <> Ø Then
   Call ISPF_MSG('Unable to read' skel 'skeleton. EXECIO rc='read_rc)
Else If inskel.Ø = Ø Then Do
   Call ISPF_MSG('The' skel 'skeleton is empty; unable to continue')
   read_rc = 4
End
Return read_rc
/*=================================================================*
/* view the generated Assembler file (matching ISPFIGU module)    */
/*----------------------------------------------------------------*/
VIEW_ASM_FILE:
"CONTROL REFLIST NOUPDATE"
"LMINIT DATAID(data1) DATASET("zcnvsrce")"
IF ZERRMSG <> 'ZERRMSG' Then
   "SETMSG MSG(ZERRMSG)"
Else Do
   "VIEW DATAID("data1") MEMBER("zcnvsrccm")"
   "LMFREE DATAID("data1")"
End
"CONTROL REFLIST UPDATE"
Return
/*=================================================================*
/* allocate output file (Assembler mapping of the active table)    */
/*-----------------------------------------------------------------*/
ALLOC_OUTPUT:
"VGGET (ZCNVSRC E ZCNVSRCE) PROFILE"
If zcnvsrcm <> ' ' Then
   If Left(zcnvsrce,1) = '"' Then
      asmout = Strip(zcnvsrce,'T','"')("zcnvsrccm")"
   Else
      asmout = zcnvsrce("zcnvsrcm")"
Else
   asmout = zcnvsrce
Address TSO "ALLOC FILE(ICONFASM) DSN("asmout") SHR REU"
alloc_rc = rc
If alloc_rc > Ø Then
   Call ISPF_MSG('Unable to allocate assembler output file',
                  'asmout ' ,
   "ALLOC FILE("skel") DSN("lib"("skel")') SHR REU"
   )
Return alloc_rc
/*====================================================================*/
/* process an Assembler DC statement                                */
/*====================================================================*/

PROCESS_DC:
j = j + 1
Parse Var inskel.i defn '&' .
Parse Var inskel.i ' DC' type .
Select
  /*----------*/
  /* ADDRESS */
  /*----------*/
  When Left(type,1) = 'A' Then Do
    Parse Var type ' AL' leng '{' rest
    chars = GET_STOR()
    dec = X2D(C2X(chars))
    newdef = ' AL'\"\"dec\"\"
    dpos = Pos(' AL',inskel.i) + 2
  End
  /*-----------*/
  /* CHARACTER */
  /*-----------*/
  When Left(type,1) = 'C' Then Do
    Parse Var type ' CL' leng '& rest
    chars = GET_STOR(
    chars = Strip(chars,'T') /* remove trailing blanks */
    If chars = '' Then chars = '' /* for long blank strings */
    qpos = Pos('"',chars) /* double any quotes: ' */
    If qpos > Ø Then Do until qpos = Ø
      chars = Left(chars,qpos)||Substr(chars,qpos)
      qpos = Pos('"",chars,qpos+2)
    End
    apos = Pos('&',chars) /* double any ands: & */
    If apos > Ø Then Do until apos = Ø
      chars = Left(chars,apos)||Substr(chars,apos)
      apos = Pos('"",chars,apos+2)
    End
    newdef = ' CL'\""\"chars\""
    dpos = Pos(' CL',inskel.i) + 2
  End
  /*----------*/
  /* FULLWORD */
  /*----------*/
  When Left(type,1) = 'F' Then Do
    If Substr(type,2,1) = 'L'
      Then Parse Var type ' FL' leng '& rest
    Else leng = 4
    remain = offset // leng
    If remain > Ø Then offset = offset + leng * remain
chars = GET_STOR()
dec = X2D(C2X(chars))
If leng = 4
   Then newdef = "F"dec"
   Else newdef = "FL"leng"dec"
   dpos = Pos('F',inskel.i) + 2
End
/*----------*/
/* HALFWORD */
/*----------*/
When Left(type,1) = 'H' Then Do
   leng = 2
   chars = GET_STOR()
   dec = X2D(C2X(chars))
   newdef = "H"dec"
   dpos = Pos('H',inskel.i) + 2
End
/*--------------*/
/* HEXIDECIMAL */
/*--------------*/
When Left(type,1) = 'X' Then Do
   Parse Var type 'XL' leng '&' rest
   chars = GET_STOR()
   hex = C2X(chars)
   newdef = 'XL'leng"hex"
   dpos = Pos('XL',inskel.i) + 2
End
/*----------------*/
/* All the rest remain unchanged */
/*----------------*/
Otherwise
   outasm.j = defn
   Return
End
/*-----------------------------*/
/* Update the line with the value from storage */
/*-----------------------------*/
outasm.j = Overlay(newdef,inskel.i,dpos)
offset = offset + leng /* point offset to next field in storage */
Return
/*=================================================================* /
/* get the required bytes from storage */
/*=================================================================* /
GET_STOR:
   addr = D2X(dec_addr + offset)
   chars = Storage(addr,leng)
Return chars
PROGRAM LOADPNT

TITLE 'GET ADDRESS OF PROGRAM LOAD POINT'

*******************************************
* LOADPNT                                            *
* ~~~~~~~~~                                          *
* External REXX function to return the address of a loaded module. *
*                                                     *
* Written by : Ron Brown - January 2000              *
*******************************************

LOADPNT RMODE ANY
LOADPNT AMODE 31
LOADPNT CSECT
USING LOADPNT, R15 SET UP ADDRESSABILITY
SAVE (14, 12)
LR R12, R15
DROP R15
USING LOADPNT, R12
B START GO AROUND EYECATCHER

* PROGNAME DC CL8'LOADPNT' EYECATCHER
DC CL8' VERSION'
VERSION  DC CL6'01.00'
DATE_ASM DC CL11'&SYSDATE' DATE ASSEMBLED
TIME_ASM DC CL8'&SYSTIME' TIME ASSEMBLED

* START DS ØH REXX STUFF:
USING EFPL, R1 MAP EVAL PARM LIST
L R2, EFPLEVAL LOAD EVAL POINTER
L R11, Ø(R2) ADDRESS OF EVAL BLOCK
USING EVALBLOCK, R11 MAP EVAL BLOCK
L R5, EFPLARG LOAD ARG LST PNTR
USING ARGTABLE_ENTRY, R5 MAP THE ENTRY

*----------------------------------------------------------------------
*        GET PROGRAM NAME FROM PARM
*----------------------------------------------------------------------

LM R2, R3, ARGTABLE_ARGSTRING_PTR LOAD ARG ADDR/LNGTH
C R3, =F'8' > 8?
BH BADPARM YES, GOTO BADPARM
* MVC PGNAME, Ø(R2) (only valid if length=8)
LA R8, PGNAME
LR R9, R3
MVCL R8, R2 SET PGNAME

* LA R5, ARGTABLE_NEXT-ARGTABLE_ENTRY(R5) GET NEXT PARM
* ADDR/LNGTH
CLC ARGTABLE_ARGSTRING_LENGTH, =8X'FF' MORE PARMS?
BE CSVQUERY NO, INVOKE QUERY
* ADDR/LNGTH
REXX JSPCCONV

This code is presented showing all the modifications, but many of the original lines of code have been omitted to save space. To implement this take a copy of the original IBM code and apply these changes to it. Every change is accompanied by a comment (with the word ‘Ron’).

/*REXX*************************************************************************/
/*                                                                       */
/* EXEC NAME := JSPCCONV (based on IBM EXEC: ISPCCONV)                    */
/*                                                                       */
/* Modified to accept current Assembler input (ie ISPF Version 5.2)       */
/*               plus APAR OW56583 from z/OS 1.4                          */
/*                                                                       */
** DESCRIPTIVE_NAME := Configuration table conversion routine **

FUNCTION = Converts configuration table Assembler file into keyword format used in ISPF for OS/390 R8 and later. **

***** ... about 105 lines of original ISPCCONV unchanged here *****

Do qq = 1 to config.Ø while substr(config.qq,1,1) = '*'

End

If pos('ISPCFIGU CSECT', config.qq) = Ø Then /* Ron */

Do

***** ... about 70 lines of original ISPCCONV unchanged here *****

get_values_from_asm_source: procedure expose configname. config.

continue

configname. = '?'

do a = 1 to config.Ø

parse upper var config.a cname dc cvalue comment

if dc = 'DC' then

do

type=substr(cvalue,1,1)

select

when(type='C') then parse var config.a ""cvalue"" . /*Ron*/

when(type='F') then parse var cvalue . ""cvalue"" .

when(type='A') then parse var cvalue . "("cvalue")" .

when(type='H') then parse var cvalue . ""cvalue"" . /*Ron*/

when(type='X') then parse var cvalue . ""cvalue"" . /*Ron*/

otherwise nop
end

cvalue=remove_double_ampersands(cvalue)

line=cname cvalue

configname. cname=line
end

if substr(config.a,72,1)¬=' ' then /* If continuation in col 72*/

continue = 1

/* Mark for a later message */
end

return

/* Rectify_Differences: Compare isrconfg with defaults and create ***/

rectify_differences:

dlen=Ø;

do pass = 1 to 3

lastgroup=Ø

do a = 1 to defaultlines

parse upper value defaults.a with oldname newname'='default

oldlineno=outlines

if configname.oldname ¬= '?' then

do

if lastgroup¬=group.a then

do
lastgroup = group.a
if (zcnvopt = 'ALL') | pass=3 then
    do
        call addline '/*' copies('-',68)
call addline ('/*'center(grouptitle.lastgroup,68)'*/')
call addline '/*' copies('-',68)
end
end

parse var configname.oldname . ncvalue

/* start of new lines added by Ron -----------------------------*/
if newname = 'LOG_DISPLAY_REQUIRED' & ncvalue = 'Y' Then
    ncvalue = 'YES'
if newname = 'LOG_DISPLAY_REQUIRED' & ncvalue = 'N' Then
    ncvalue = 'NO'
if newname = 'LOG_KPT' & ncvalue = 'Y' Then
    ncvalue = 'YES'
if newname = 'LOG_KPT' & ncvalue = 'N' Then
    ncvalue = 'NO'
if newname = 'LOG_MESSAGE_ID' & ncvalue = 'Y' Then
    ncvalue = 'YES'
if newname = 'LOG_MESSAGE_ID' & ncvalue = 'N' Then
    ncvalue = 'NO'
if newname = 'LIST_DISPLAY_REQUIRED' & ncvalue = 'Y' Then
    ncvalue = 'YES'
if newname = 'LIST/display_required' & ncvalue = 'N' Then
    ncvalue = 'NO'
if newname = 'LIST_KPT' & ncvalue = 'Y' Then
    ncvalue = 'YES'
if newname = 'LIST_KPT' & ncvalue = 'N' Then
    ncvalue = 'NO'
if newname = 'SCREEN_FORMAT' & ncvalue = 'S' Then
    ncvalue = 'STD'
if newname = 'HOST_COLORS' & ncvalue = '' Then
    ncvalue = 'OFF'
if newname = 'PC_COLORS' & ncvalue = '' Then
    ncvalue = 'OFF'
if newname = 'DEFAULT_MESSAGE_ID' & ncvalue = '1' Then
    ncvalue = 'OFF'
if newname = 'DEFAULT_PANEL_ID' & ncvalue = '1' Then
    ncvalue = 'OFF'
if newname = 'DEFAULT_SCREEN_NAME' & ncvalue = '1' Then
    ncvalue = 'OFF'
if newname = 'ENABLE_EURO_SYMBOL' & ncvalue = 'Y' Then
    ncvalue = 'YES'
if newname = 'ENABLE_EURO_SYMBOL' & ncvalue = 'N' Then
    ncvalue = 'NO'
if newname = 'GUI_DISPLAY_ENTER_KEY' & ncvalue = '/' Then
    ncvalue = 'YES'
if newname = 'GUI_DISPLAY_ENTER_KEY' & ncvalue = '' Then
    ncvalue = 'NO'
if newname = 'SAVE_GUI_VALUES' & ncvalue = 'Y' Then
    ncvalue = 'YES'
if newname = 'SAVE_GUI_VALUES' & ncvalue = '' Then
    ncvalue = 'NO'
if newname = 'GUI_ACCELERATOR_SUPPORT' & ncvalue = '/' Then
    ncvalue = 'YES'
if newname = 'GUI_ACCELERATOR_SUPPORT' & ncvalue = '' Then
    ncvalue = 'NO'
if newname = 'GUI_DOWNLOAD_IMAGES' & ncvalue = '/' Then
    ncvalue = 'YES'
if newname = 'GUI_DOWNLOAD_IMAGES' & ncvalue = '' Then
    ncvalue = 'NO'
if newname = 'GUI_MAKEPATH_FOR_IMAGES' & ncvalue = '/' Then
    ncvalue = 'YES'
if newname = 'GUI_MAKEPATH_FOR_IMAGES' & ncvalue = '' Then
    ncvalue = 'NO'
if newname = 'CONTINUE_3270_AFTER_LOSS_OF_WS_CONNECTION' &,
    ncvalue = '/' Then
    ncvalue = 'YES'
if newname = 'CONTINUE_3270_AFTER_LOSS_OF_WS_CONNECTION' &,
    ncvalue = '' Then
    ncvalue = 'NO'
if newname = 'BROWSE_FTP_ERRORS' & ncvalue = '/' Then
    ncvalue = 'YES'
if newname = 'BROWSE_FTP_ERRORS' & ncvalue = '' Then
    ncvalue = 'NO'
if newname = 'CREATE_DIRECTORY_ON_WSA_DOWNLOAD' &,
    ncvalue = '/' Then
    ncvalue = 'YES'
if newname = 'CREATE_DIRECTORY_ON_WSA_DOWNLOAD' &,
    ncvalue = '' Then
    ncvalue = 'NO'
if newname = 'WSA_DOWNLOAD_DATA_SET' Then Do
    If left(ncvalue,1) = '\' Then
        ncvalue = substr(ncvalue,2,Length(ncvalue)-2)
    end
if newname = 'FRAME_COLOR' Then
    ncvalue = left(ncvalue,1)
if newname = 'FRAME_INTENSITY' Then
    ncvalue = right(ncvalue,1) / 4
if newname = 'ENABLE_ISPF_EXITS' & ncvalue = '60' Then
    ncvalue = 'NO'
if newname = 'ENABLE_ISPF_EXITS' & ncvalue = 'E0' Then
    ncvalue = 'YES'
if newname = 'USE_MVS_OPEN_EDITION_SOCKETS' &,
    ncvalue = '0' Then
    ncvalue = 'NO'
if newname = 'USE_MVS_OPEN_EDITION_SOCKETS' &,
    ncvalue = '1' Then
    ncvalue = 'YES'
if ncvalue='' then ncvalue='NONE'
if newname = 'COMMAND_LINE_PLACEMENT' & ncvalue = 'NONE' Then
       ncvalue = 'BOTTOM'
if newname = 'SITE_COMMAND_TABLE_SEARCH_ORDER' &
       ncvalue = 'A' Then
       ncvalue = 'AFTER'
if newname = 'SITE_COMMAND_TABLE_SEARCH_ORDER' &,
       ncvalue = 'B' Then
       ncvalue = 'BEFORE'
if newname = 'ALLOWED_ALLOCATION_UNITS' & ncvalue = 'A' Then
       ncvalue = 'ANY'
if newname = 'USE_KEYLISTS' & ncvalue = 'Y' Then
       ncvalue = 'YES'
if newname = 'USE_KEYLISTS' & ncvalue = 'N' Then
       ncvalue = 'NO'
if newname = 'GLOBAL_COLORS' & ncvalue = 'NONE' Then
       ncvalue = '1234567'
if newname = 'USE_ALTERNATE_DIALOG_TEST_PANEL' &,
       ncvalue = 'NO' Then
       ncvalue = 1
if newname = 'SUPERC_LIST_DATA_SET_BLOCK_SIZE' &,
       ncvalue = '13566' Then
       ncvalue = 'Ø'
if newname = 'SUPERC_UPDATE_DATA_SET_BLOCK_SIZE' &,
       ncvalue = '13680' Then
       ncvalue = 'Ø'
if newname = 'SUPERC_PROFILE_DATA_SET_BLOCK_SIZE' &,
       ncvalue = '13680' Then
       ncvalue = 'Ø'
if newname = 'SUPERC_STATMENTS_DATA_SET_BLOCK_SIZE' &,
       ncvalue = '13680' Then
       ncvalue = 'Ø'
if newname = 'PRINTDS_DEST_OR_WRITER_OPTION' Then
       if oldname = 'PDSOPR1' & ncvalue ≠ 'DEST' Then
       Iterate
       else
       if oldname = 'PDSOPR2' & ncvalue ≠ 'WRITER' Then
       Iterate
indent = 0 /* Lines start in col 1 */
/* This loop was used to check each word of the value, and finally */
/* write only the first word of the value into the keyword file, */
/* which doesn't make much sense to me. Therefore this loop is no */
/* longer used (now whole multi-word values go into the file) Ron */
/* do until ncvalue='' ** Process multi-Word values .... Ron */
/* len=maxlen.lastgroup */
len=44
if ((pass=2 & zcnvopt = 'CHG') ) then
len=45
if pass=2 & zcnvopt¬='ALL' then len=dlen
len=len-indent
/*          parse var ncvalue cvalue ncvalue    ????? .... Ron */
/*-----------------------------------------------------------------*/
/* instead of the previous line we'll write the whole value into */
/* the keyword file (required for multi-word values).       Ron */
/*-----------------------------------------------------------------*/
ncvalue = Strip(ncvalue,'L')   /* no leading blanks   Ron */
if ncvalue ¬= default then     /* cvalue -> ncvalue   Ron */
if zcnvopt='CHG' & pass = 3 then nop
else
  if pass=1 then
    dlen=max(dlen,length(space(newname)))
  else
    if pass=2 then
      call addline left(newname,len)'= 'ncvalue  /* Ron */
    else nop
  endif
else
  if zcnvopt='CHG' & pass <3 then nop
  else
    if zcnvopt='CHG' & pass = 3 then
      call addline '/*'left(newname,len-2)||,
      '= 'ncvalue                    /* Ron */
    else
      if zcnvopt = 'ALL' then
        call addline left(newname,len)'= 'ncvalue  /* Ron */
      /*         if pass>1 &,
      oldlineno<outlines & Ø<wordpos(oldname,color) then
      call add_color_comments */
      indent = 2    /* Continuations start in col 3 */
      indent = Ø     /* Lines start in col 1 */
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
end
If Left(zcnvsrc,1) <> "" Then Do
  dspref = Sysvar(SYSPREF)
  If dspref <> ''
    Then asmsrce = dspref'.zcnvsrc
  Else asmsrce = zcnvsrc /* TSO NOPREFIX */
End
Else asmsrce = Strip(zcnvsrc,"B",""") /* remove the quotes */
If zcnvsrcm ¬= '' Then
call addline '/* into library member' asmsrce'(zcnvsrcm)'
Else
call addline '/* into file 'asmsrce
call addline '/* Then JSPCCONV converted that assembler',
  'to this Keyword File.'
call addline '/*'
/* -------- end of lines added by Ron --------------------------*/
/* -------- lines removed by Ron --------------------------------*/
/* If zcnvsrcm ¬= '' Then                                           */
/*   call addline '** Converted from 'zcnvsrc' member 'zcnvsrcm  */
/* Else                                                             */
/*   call addline '** Converted from 'zcnvsrc               */
/*   call addline '** by user 'userid()'.'                         */
/*----------------------------------------------------------------------*/
if zcnvopt = 'CHG' then
call addline '/* Defaults were included as comments.'
**** ... about 80 lines of original ISPCCONV unchanged here ****
Else
  Address ISPEXEC 'SETMSG MSG(ISPC289)'
return
/* LMF keywords removed migrating from ISPF 5.0 -> 5.2 .... Ron */
*GROUP LMF Control Status data set specifications
Cblksize     Control_Status_Block_Size=13600
Clrecl       Control_Status_Record_Length=160
Csize        Control_Status_Lines_Per_Page=50
Cprimqty     Control_Status_Primary_Quantity=200
Csecqty      Control_Status_Secondary_Quantity=100
Cdblks       Control_Status_Directory_Blocks=10
*GROUP LMF Member Status data set specifications
Mblksize     Member_Status_Block_Size=13600
Mlrecl       Member_Status_Record_Length=160
Msieze       Member_Status_Lines_Per_Page=50
Mprimqty     Member_Status_Primary_Quantity=200
Msecqty      Member_Status_Secondary_Quantity=100
Mdblks       Member_Status_Directory_Blocks=10
/* new keywords added migrating from ISPF 5.0 -> 5.2 .... Ron */
*GROUP LMF
lfmlock      fail_on_lmf_lock=Yes
editcutdef   Edit_Cut_Default=Replace
editpasdef   Edit_Paste_Default=Keep
zcnvdsc      Scroll_Default=Page
zcnvdst      Status_Area_Default=SES
/* keywords required but missing for ISPF 5.2 .... Ron */
PDFFVBRL "FORCE_PRESERVE_VB_RECORD_LENGTH=NO"
PDFFVBRL "PRESERVE_VB_RECORD_LENGTH=NO"
/* corrected the following keywords for ISPF 5.2 .... Ron */
<table>
<thead>
<tr>
<th>Wrong</th>
<th>Right</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supcslpr</td>
<td>SUPCLSPR</td>
<td>Supercl_Listing_Primary_Quantity=50</td>
</tr>
<tr>
<td>Supcslsc</td>
<td>SUPCLSCL</td>
<td>Supercl_Listing_Secondary_Quantity=100</td>
</tr>
<tr>
<td>Supcsupr</td>
<td>SUPCUPPR</td>
<td>Supercl_Update_Primary_Quantity=15</td>
</tr>
<tr>
<td>Supcsusc</td>
<td>SUPCUPSC</td>
<td>Supercl_Update_Secondary_Quantity=30</td>
</tr>
<tr>
<td>Edtvbwarn</td>
<td>TRUNCWRN</td>
<td>Warn_On_Truncation_Of_Trailing_Blanks=Yes</td>
</tr>
<tr>
<td>editctudf</td>
<td>EDITCTUT</td>
<td>Edit_Cut_Default=Replace</td>
</tr>
<tr>
<td>editpaudf</td>
<td>EDITPAST</td>
<td>Edit_Paste_Default=Keep</td>
</tr>
<tr>
<td>vsamedite</td>
<td>VSAMEE</td>
<td>Vsam_Edit_Enabled=No</td>
</tr>
<tr>
<td>vsameditc</td>
<td>VSAMEC</td>
<td>Vsam_Edit_Command=Ditto ve</td>
</tr>
<tr>
<td>vsambrowse</td>
<td>VSAMBME</td>
<td>Vsam_Browse_Enabled=No</td>
</tr>
<tr>
<td>vsambrowsec</td>
<td>VSAMBC</td>
<td>Vsam_Browse_Command=Ditto vb</td>
</tr>
<tr>
<td>vsamviewe</td>
<td>VSAMVE</td>
<td>Vsam_View_Enabled=No</td>
</tr>
<tr>
<td>vsamviewc</td>
<td>VSAMVC</td>
<td>Vsam_View_Command=Ditto vb</td>
</tr>
<tr>
<td>zcnvdscl</td>
<td>PDFDSCRL</td>
<td>Scroll_Default=Page</td>
</tr>
<tr>
<td>zcnvdst</td>
<td>PDFDSTAT</td>
<td>Status_Area_Default=SES</td>
</tr>
<tr>
<td>Pdsopr1</td>
<td>PDSOPR</td>
<td>Printds_Dest_Or_Writer_Option=Dest</td>
</tr>
<tr>
<td>Pdsopr2</td>
<td>PDSOPR</td>
<td>Printds_Dest_Or_Writer_Option=Writer</td>
</tr>
<tr>
<td>Localp1rt</td>
<td>LOCALPR1</td>
<td>Local_Prints_Options=Nonum</td>
</tr>
<tr>
<td>Pdfhqlq</td>
<td>OPT34HLQ</td>
<td>Disallow_Wildcards_In_HLQ=No</td>
</tr>
<tr>
<td>tbadd#</td>
<td>TBAADDROW</td>
<td>Number_Of_Rows_For_Tbadd=1</td>
</tr>
<tr>
<td>retrcmdsz</td>
<td>RETCMDSZ</td>
<td>Retrieve_Command_Stack_Size=512</td>
</tr>
<tr>
<td>ispfexits</td>
<td>ISPFEIXIT</td>
<td>Enable_Ispf_Exits=No</td>
</tr>
<tr>
<td>tcpipdata</td>
<td>TCPDATA</td>
<td>Sas/c_Tcpip_Data_Value=Default</td>
</tr>
<tr>
<td>tcpippref</td>
<td>TCPPREF</td>
<td>Sas/c_Tcpip_Prefix_Value=Default</td>
</tr>
<tr>
<td>mvsoe</td>
<td>USEOE</td>
<td>Use_Mvs_Open_Edition_Sockets=NO</td>
</tr>
<tr>
<td>zfrmacl</td>
<td>ZFRAMIC</td>
<td>Frame_Color=1</td>
</tr>
<tr>
<td>zframai</td>
<td>ZFRAMIC</td>
<td>Frame_Intensity=2</td>
</tr>
<tr>
<td>zcnvtpp</td>
<td>ZDEFPPPAN</td>
<td>Default_Primary_Panel=ISP@MSTR</td>
</tr>
<tr>
<td>ztimesep</td>
<td>ZTSEP</td>
<td>Default_Time_Separator=D</td>
</tr>
</tbody>
</table>

/* removed the following 4 keywords */
Loglrecl    Log_Data_Set_Record_Length=125
Tcntlrcl    Record_Length_For_Temporary_Cntl_Data_Sets=80
Tlstlrcl    Record_Length_For_Temporary_List_Data_Sets=121
Twrkrlc    Record_Length_For_Temporary_Work_Data_Sets=256

/* All keywords in the following section were in Mixed Case, and all */
/* changes made by Ron are in CAPITALS, to make them easy to see. */
/*removed the above 4 keywords */

/*Start -- Do not change this comment line!!!*/
*GROUP PDF Exits
Allocpgm    Data_Set_Allocation_Program Exit=None
Prtpgm      Print_Utility_Program Exit=None
Prtclist    Print_Utility_Command Exit=None
Cmppgm  Compress_Utility_Program_Exit=None
Cmpclist  Compress_Utility_Clist Exit=None
Dlspgms  Data_Set_List_Program_Exit=None
Mlfpgm  Member_List_Filter_Program_Exit=None
Nmchpgm  Data_Set_Name_Change_Program_Exit=None
/* spelling correct on next line                           @ow56583*/
Dsllcpgm  Data_Set_List_Line_Command_Program_Exit=None
Instacct  Activity_Monitoring_Program_Exit=None
Memcpext  Member_List_Line_Command_Program_Exit=None
Memccext  Member_List_Line_Command_Command_Exit=None
*GROUP Data Set Allocation Settings
Pdfunit  Pdf_Default_Unit=Sysallda
Unitauth  Allowed_Allocation_Units=Any
Pcfalloc  Allocate_Before_Uncatalog=No
Checkexp  Verify_Expiration_Date=Yes
Delvol  Volume_Of_Migrated_Data_Sets=Migrat
Delcmd  Command_To_Delete_Migrated_Data_Sets=Hdelete
*GROUP Outlist data set specifications
Olrecl  Outlist_Record_Length=133
Oblksize  Outlist_Block_Size=13566
Oprimqty  Outlist_Primary_Quantity=200
Osecqty  Outlist_Secondary_Quantity=100
*GROUP SuperC data set specifications
Supcblk  SuperC_List_Data_Set_Block_Size=0
Supcublk  SuperC_Update_Data_Set_Block_Size=0
Supcpblk  SuperC_Profile_Data_Set_Block_Size=0
Supcsblk  SuperC_Statements_Data_Set_Block_Size=0
Supcpgm  Use_SuperC_Program_Interface=Yes
SUPCLSPR  SuperC_Listing_Primary_Quantity=50
SUPCLSSC  SuperC_Listing_Secondary_Quantity=100
SUPCUPPR  SuperC_Update_Primary_Quantity=15
SUPCUPSC  SuperC_Update_Secondary_Quantity=30
*GROUP LMF
lmflock  fail_on_lmf_lock=Yes
*GROUP Edit recovery data set specifications
Eblksize  Edit_Recovery_Block_Size=13680
Eprimqty  Edit_Recovery_Primary_Quantity=40
Esecqty  Edit_Recovery_Secondary_Quantity=200
*GROUP Move/Copy Settings
Copyrc  Maximum_Good_Iebcopy_Return_Code=0
Copyopt  Use_Iebcopy_Copy_Or_Copymod_Option=2
Iebcopt  When_To_Use_Iebcopy=0
Umcalloc  Allow_Data_Set_Creation_For_Move_Copy=Yes
*GROUP Edit related settings
Edtproft  Maximum_Edit_Profiles=25
Sclmchk  Sclm_Warning_Level=Warn
Undosize  Undo_Storage_Size=0
Anycolor  Allow_Edit_Highlighting=Yes
Dfltcolr  Default_Edit_Display=3
Editstor  Maximum_Storage_Allowed_For_Edit=0

Asmvideo  Enable_Assembler_Continuation_Errors=Yes
TRUNCWRN   Warn_On_Truncation_Of_Trailing_Blanks=Yes
Pdfceimacro Site_Wide_Initial_Macro=None
Tflow      Text_Flow_Terminators=.:& <--
EDITCUT    Edit_Cut_Default=Replace
EDITPAST   Edit_Paste_Default=Keep
Ecralloc   Allow_Data_Set_Creation_For_Create.Replace=Yes
PDFVBRL    FORCE_PRESERVE_VB_RECORD_LENGTH=NO
PDFVBRL    PRESERVE_VB_RECORD_LENGTH=NO
clipnum    Maximum_Number_Of_Edit_Clipboards=11
clipsize   Maximum_Edit_Clipboard_Size=Ø
VSAMEE     Vsam_Edit_Enabled=No
VSAMEC     Vsam>Edit_Command=Ditto ve /
VSAMBE     Vsam>Browse_Enabled=No
VSAMBC     Vsam>Browse_Command=Ditto vb /
VSAMVE     Vsam>View_Enabled=No
VSAMVC     Vsam>View_Command=Ditt vo /
*group edit site wide profile customizations
***** ... 59 lines of original ISPCCONV unchanged here *****
Zpfshow    Show_Pfkeys=On
Zpff      Reset_Show_Pfkeys=No
PDFDSCRPL  Scroll_Default=Page
PDFDSSTA7  Status_Area_Default=SES
Lstblks    List_Data_Set_Records_Per_Block=26
Logblksz   Log_Data_Set_Block_Size=129
Tpbblksz   Block_Size_For_Temporary_Cntl_Data_Sets=800
Tplblksz   Block_Size_For_Temporary_List_Data_Sets=3146
Twrbblks   Block_Size_For_Temporary_Work_Data_Sets=2560
Zctlpqty   ISPCTL_Primary_Quantity=10
Zctlsqty   ISPCTL_Secondary_Quantity=100
Zwrkpqty   ISPWKR_Primary_Quantity=10
Zwrgsqrty   ISPWKR_Secondary_Quantity=100
Ztmpunit   Use_PDFCUNIT_for_Temp_ISPF_Data_sets=No
Ztmpqqual   ISP_F Temporary_Data_set_Qualifier=No
PDSOPR     Printds_Dest_Or_Writer_Option=Dest
Localprt   Local_Printds_Options=Nonum
Pdsyfpalt  Use_Altimate_Dialog_Test_Panel=1
*GROUP default cua color settings
***** ... 64 lines of original ISPCCONV unchanged here *****
*GROUP Miscellaneous settings
Acexedma   Monitor_Edit_Macro_Commands=No
Brsubmit    Allow_Submit_From_Browse=Yes
Vwsubmit    Allow_Submit_From_View=Yes
Renamgdg    Warn_On_Rename_To_Gdg_Name=Yes
Dslmeml     Default_Edit/Browse/View_Member_List=Yes
Suppview    ls_View_Supported=Yes
Tsopanel    Use_Altimate_Panel_Isrtsoa=No
Icfprt      Print_U sing_Icf=No
OPT34HLQ    Disallow_Wildcards_In_HLQ=No
Scrmax      Maximum_Number_Of_Split_Screens=8
<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usercmds</td>
<td>Applid_For_User_Command_Table=None</td>
</tr>
<tr>
<td>Sitecmds</td>
<td>Applid_For_Site_Command_Table=None</td>
</tr>
<tr>
<td>Sctsrch</td>
<td>Site_Command_Table_Search_Order=Before</td>
</tr>
<tr>
<td>Year2ØØØ</td>
<td>Year_2ØØØ_Sliding_Rule=65</td>
</tr>
<tr>
<td>zshowenq</td>
<td>Show_Enq_Displays=YES</td>
</tr>
<tr>
<td>zdeflang</td>
<td>Default_session_language=ENGLISH</td>
</tr>
<tr>
<td>TBADDROW</td>
<td>Number_Of_Rows_For_Tbadd=1</td>
</tr>
<tr>
<td>RETCMDSZ</td>
<td>Retrieve_Command_Stack_Size=512</td>
</tr>
<tr>
<td>ISPFEXIT</td>
<td>Enable_Ispf_Exits=No</td>
</tr>
<tr>
<td>TCPDATA</td>
<td>Sas/c_Tcpip_Data_Value=Default</td>
</tr>
<tr>
<td>TCPPREF</td>
<td>Sas/c_Tcpip_Prefix_Value=Default</td>
</tr>
<tr>
<td>USEOE</td>
<td>Use_Mvs_Open_Edition_Sockets=NO</td>
</tr>
</tbody>
</table>

*GROUP Values formerly in ISPDFLTS*

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>zsetpds</td>
<td>Default_Panel_Id=Off</td>
</tr>
<tr>
<td>zsetsds</td>
<td>Default_Screen_Name=Off</td>
</tr>
<tr>
<td>ZDEFPPLAN</td>
<td>Default_Primary_Panel=ISP@MSTR</td>
</tr>
<tr>
<td>zchareur</td>
<td>Enable_Euro_Symbol=No</td>
</tr>
<tr>
<td>chareurf</td>
<td>Reset_Enable_Euro_Symbol=No</td>
</tr>
<tr>
<td>zdatefd</td>
<td>Date_Format_Zdatefd=DEFAULT</td>
</tr>
<tr>
<td>zdatef</td>
<td>Date_Format_Zdatef=DEFAULT</td>
</tr>
<tr>
<td>ZTSEP</td>
<td>Default_Time_Separator=D</td>
</tr>
</tbody>
</table>

*GROUP ISPSPROF Workstation Defaults*

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>***** ... 64 lines of original ISPCCONV unchanged here *****</td>
<td></td>
</tr>
<tr>
<td>***** ... last 39 lines of original ISPCCONV unchanged *****</td>
<td></td>
</tr>
</tbody>
</table>

### ISPF VERSION DEPENDENCE

It is based on the standard IBM dialog to avoid ‘re-inventing the wheel’ and to integrate it into the features supplied by IBM. Mapping the configuration module was based on the standard IBM skeleton ISPCSKEL, which is supplied with each version of ISPF, to ensure that it always produces a valid assembly file.

However, the REXX code JSPCCONV to convert the ASM file to keywords is version-dependent. The original code from IBM was first supplied with ISPF Version 4.8 (OS/390 2.8) and has had a few changes made with each version since. The version of JSPCCONV in this article is based on the IBM code from ISPF version 5.2 (z/OS 1.2), which had a few errors to be fixed, and it now has keywords added (and corrected) to allow the input of an assembly file for a version of ISPF after 4.5. As presented here, the code will produce keyword files for ISPF Version 5.2, but for later versions you may need to add more keywords.
If you use this for earlier versions of ISPF (eg Version 5.0 from OS/390 2.10) it may generate a keyword file with some keywords that are not valid for your version. If that occurs, just select option 1, which will use your freshly created file as input, to regenerate a new keyword file with only the keywords that are valid for your ISPF version.

IMPLEMENTATION
You’ll need you own libraries allocated to:

- DDNAME=ISPPLIB panel(s) – JSPPCONF, (help panels).
- DDNAME=SYSEXEC EXECs – JSPCCONF, JSPCATAB, JSPCCONV.
- DDNAME=ISPLLIB modules – LOADPNT, ISPCFIGU.

Put your generated ISPF configuration module into your library allocated to DDNAME=ISPLLIB, then stop ISPF, and your options will be used when ISPF is restarted.

If you have any problems implementing this, send me an e-mail at Ron_Brown@hotmail.com and I will help.

CONCLUSION
With this extension of the IBM dialog you will be able to check your own active ISPF configuration, customize it however you like, and not affect anyone else’s configuration.

Ron Brown
Consultant (Germany)  © Xephon 2003

Managing uncatalogued datasets on SMS volumes

One of the most common problems that OS/390-MVS systems programmers are called upon to fix involves datasets that have
become uncatalogued. These objects appear occasionally as a result of system errors, abends, or deliberate uncataloguing.

Non-VSAM, non-SMS objects can be recatalogued under ISPF 3.4 using the option ‘C’.

When the related ICF catalog entry is deleted (eg by a DELETE NOSCRATCH), the dataset can’t be found without explicit reference to the volume on which it still resides; but a VSAM or an SMS-managed dataset must be catalogued. It can be viewed in 3.4 ISPF DLIST by supplying the original VOLSER, but it can’t be used at all!

Using IDCAMS DIAGNOSE and EXAMINE we can detect inconsistencies between catalogs (BCS and VVDS) and a VTOC. For VSAM and SMS-managed datasets, there are some entries in VVDS called, respectively, VVR and NVR; sometimes we need to recatalog them or to scratch the damaged entries.

Note: VRRDS (Variable-length Relative Record Data Set) was first introduced with DFSMS V1.5, and is defined as ‘NUMBERED’ VSAM with RECORDSIZE(x y). This organization, like KSDS, has a DATA and INDEX component.

I have four simple solutions for the following situations:

• To recatalog uncatalogued VSAM objects – the RV CLIST.
• To recatalog uncatalogued NONVSAM SMS objects – the RNV CLIST.
• To scratch uncatalogued VSAM objects – the DVVR CLIST.
• To scratch uncatalogued NONVSAM SMS objects – the DNVR CLIST.

Use them in ISPF 3.4 by typing their names against the dataset names.

In order to be used, the CLISTs should be placed in a partitioned dataset concatenated to the //SYSPROC card of your logon procedure.

Look at the example below.
In ISPF 3.4, enter DSNAME LEVEL and VOLUME SERIAL (an SMS volume) of the uncatalogued entries:

Option ==>

<table>
<thead>
<tr>
<th>blank</th>
<th>Display data set list</th>
<th>P Print data set list</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>Display VTOC information</td>
<td>PV Print VTOC information</td>
</tr>
</tbody>
</table>

Enter one or both of the parameters below:

- **Dsname Level**: EE.CICCIO
- **Volume serial**: SMPRØ1

Then, use my CLISTs by entering their name before the object name:

```
------------------------------- DDLIST - Data Sets on volume SMPRØ1
Row 1 of 7
Command ==> Scroll ==> PAGE
```

```
Command - Enter "/" to select action
- Message
- Volume

- ------------------------ DVVR
  - EE.CICCIO.ESDS.DATA
  - EE.CICCIO.KSDS.DATA
  - SMPRØ1

- RV
  - EE.CICCIO.KSDS.INDEX
  - SMPRØ1

- RV / TYPE(LDS)
  - EE.CICCIO.RRDS.DATA
  - EE.CICCIO.VRRDS.DATA
  - SMPRØ1

- EE.CICCIO.VRRDS.INDEX
  - SMPRØ1

************************** End of Data Set list **************************
```

Whether you try to recatalog or delete a non-VSAM uncatalogued SMS-managed dataset directly from ISPF 3.4 volume list, the result is:

```
------------------------------- DDLIST - Data Sets on volume SMPRØ1
Row 1 of 7
Command ==> Scroll ==> PAGE
```

```
Command - Enter "/" to select action
- Message
- Volume

- C
  - EE.CICCIO.NONVSAM
  - SMPRØ1

************************** End of Data Set list **************************
```

So, you can type %DNVR to delete it or %RNV to recatalog it.

**RV CLIST**

**RV = Recatalog VSAM.**
Use this CLIST to recatalog VSAM clusters without a BCS entry.

You may recatalog all kinds of VSAM organization datasets (KSDS, ESDS, RRDS, VRRDS, and linear) starting from a DATA or INDEX component. You have to provide TYPE(type). If not specified, it defaults to TYPE(KSDS). It is sufficient to provide the first two characters (eg KS).

If the .data or .index components are not in standard format (.DATA and .INDEX), you may specify your own suffixes by typing DATA(data-component-suffix) and INDEX(index-component-suffix).

DB2 objects follow a different standard (DSNDDB/DSNDBD), so in this case the RV CLIST tries ‘automatically’ to recatalog the DSNDDBC cluster entry and the DSNDBD data entry without other instructions for use.

I suggest an undocumented trick to recatalog VRRDS entries in OS/390 2.10: as it doesn’t need to remember the RECORDSIZE(avg max) values, just use RECORDSIZE(n n+1); it works fine (see line 48 of RV CLIST).

PROC 1 DSN DATA(DATA) INDEX(INDEX) TYPE(KS) DEBUG
/*- SETUP FOR DEBUG IF REQUESTED -----------------------------------------------*/
CONTROL NOMSG NOFLUSH ENDO NOCONLIST NOPROMPT
IF &DEBUG = DEBUG THEN +
CONTROL MSG LIST NOFLUSH ENDO PROMPT SYMLIST CONLIST
/*- END OF SETUP -----------------------------------------------------------------*/
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/*                                                                   */
/*  RV:      DEFINE RECATALOG OF A VSAM CLUSTER ENTRY.                   */
/*           TO BE USED IN ISPF 3.4 ONLY (VOLUME LIST)                    */
/*           YOU MUST KNOW THE VOLUME WHERE ENTRY RESIDES BEFORE!          */
/*           THEN, LOCATE THE 'DATA' AND 'INDEX' VTOC ENTRIES:           */
/*           JUST TYPE 'RV' IN FRONT OF ONE OF THEM TO RECATALOG          */
/*           ALL THE VSAM COMPONENTS.                                     */
/*           DEFAULT: IF YOU DO NOT SPECIFY THE QUALIFIER FOR DATA OR INDEX*/
/*           COMPONENTS, CLIST TRIES TO RECATALOG A DB2 OBJECT (IF        */
/*           IT IS PRESENT A QUALIFIER '.DSNDBD'); ELSE, CLIST            */
/*           SEARCH FOR DATA/I INDEX COMPONENT '.DATA' AND '.INDEX';       */
/*           IF YOU DO NOT SPECIFY THE VSAM ORGANIZATION, CLIST           */
/*           DEFAULTS IT TO 'KSDS'.                                       */
/*                                                                   */
/*  VALID VSAM ORGANIZATIONS ARE:                                        */
/*  ES OR ESDS: ENTRY-SEQUENCED DATA SET                                 */

/* KS OR KSDS: KEY-SEQUENCED DATA SET */
/* RR OR RRDS: RELATIVE RECORD DATA SET */
/* VR OR VRRDS: VARIABLE-LENGTH RELATIVE RECORD DATA SET */
/* LS OR LDS OR LINEAR: LINEAR DATA SET */

/* EG: %RV MYVSAM.ESDS.DATA TYPE(ESDS) */
RESULTS IN RECATALOGING:
CLUSTER MYVSAM.ESDS
DATA MYVSAM.ESDS.DATA

%RV MYVSAM.KSDS.IND DATA(DAT) INDEX(IND)
RESULTS IN RECATALOGING:
CLUSTER MYVSAM.KSDS
DATA MYVSAM.KSDS.DAT
INDEX MYVSAM.KSDS.IND

%RV MYDB2.DSNDBD.DBEI CA01.TSEI CA01.I0001.A001
RESULTS IN RECATALOGING:
CLUSTER MYDB2.DSNDBC.DBEI CA01.TSEI CA01.I0001.A001
DATA MYDB2.DSNDBD.DBEI CA01.TSEI CA01.I0001.A001

YOU MAY USE THE 'DEBUG' OPTION TO SEE ALL CLIST MSGS.

/* RETURN CODES: 0, 4, 8, 12, 16 AS IDCAMS RETURNS FROM DEFINE RECATALOG */
/* 9: VSAM TYPE NOT RECOGNIZED */
/* 10: VSAM TYPE NOT RECOGNIZED, PARAMETER TOO SHORT */
/* 11: NOT .DATA OR .INDEX COMPONENT */

ISPEXEC VGET ZDLVOL

/* SEARCHING FOR DB2 NAMING STANDARDS (.DSNDBD) */

SET &L = &SYSINDEX(DSNDBD,&DSN)
IF &L = Ø THEN GOTO NODB2
ELSE DO
SET &ORG = LINEAR
SET &W = &LENGTH(&DSN)
SET &NOME=&STR(&SUBSTR(1:&L+4,&DSN))
SET &NOME=&NOME.C&STR(&SUBSTR(&L+6:&W-1,&DSN))
CONTROL MSG LIST
DEF CL(NAME(&NOME') RECATALOG +
 &ORG VOL(&ZDLVOL)) +
DATA(NAME(&DSN) VOL(&ZDLVOL))
EXIT CODE(&LASTCC)
ENDO
NODB2: +
SET &DATA=&STR(. &DATA)
SET &INDEX=&STR(.&INDEX)
IF &LENGTH(&TYPE) > 1 THEN +
SET &TYPE=&SUBSTR(1:2,&TYPE)
ELSE DO
WRITE ***********************************************
WRITE * &TYPE: UNKNOWN VSAM ORGANIZATION, PARAMETER TOO SHORT!  *
WRITE * VALID TYPE ARE: KSDS, ESDS, RRDS, VRRDS, LINEAR OR LDS. *
WRITE ***********************************************
EXIT CODE (10)
ENDO
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* SEARCHING FOR VSAM ORGANIZATION */
/* */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
SELECT &TYPE
  WHEN (KS)   SET &ORG=INDEXED
  WHEN (VR)   SET &ORG=&STR(NUMBERED RECSZ(1 2))
  WHEN (RR)   SET &ORG=NUMBERED
  WHEN (LD)   SET &ORG=LINEAR
  WHEN (LS)   SET &ORG=LINEAR
  WHEN (LI)   SET &ORG=LINEAR
  WHEN (ES)   SET &ORG=NONINDEXED
  OTHERWISE   DO
WRITE ***********************************************
WRITE * &TYPE: UNKNOWN VSAM ORGANIZATION!  *
WRITE * VALID TYPE ARE: KSDS, ESDS, RRDS, VRRDS, LINEAR OR LDS. *
WRITE ***********************************************
EXIT CODE (9)
ENDO
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* SEARCHING FOR .DATA OR .INDEX COMPONENT */
/* */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
SET &L  = &SYSINDEX(&DATA,&DSN)
IF &L = Ø THEN DO
  SET &L  = &SYSINDEX(&INDEX,&DSN)
  IF &L = Ø THEN GOTO FUOR
  ELSE GOTO BUON
ENDO
BUON: +
SET &NOME = &STR(&SUBSTR(1:&L-1,&DSN)
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
IF &TYPE=LS OR &TYPE=LI OR &TYPE=RR OR &TYPE=ES OR &TYPE=LS THEN DO
  CONTROL MSG LIST
  DEF CL(NAME(&NOME') RECATALOG +
    &ORG VOL(&ZDLVOL)) +
    DATA(NAME(&NOME&DATA') VOL(&ZDLVOL))
  ENDO
ELSE IF &TYPE=KS OR &TYPE=VR THEN DO
  CONTROL MSG LIST
  DEF CL(NAME(&NOME') RECATALOG +
    &ORG VOL(&ZDLVOL)) +
    DATA(NAME(&NOME&DATA') VOL(&ZDLVOL)) +
    INDEX(NAME(&NOME&INDEX') VOL(&ZDLVOL))
  ENDO
EXIT CODE(&LASTCC)
FUOR: +
  WRITE ***********************************************************
  WRITE * ENTER RV COMMAND BEFORE A .DATA OR .INDEX COMPONENT! *
  WRITE ***********************************************************
  EXIT CODE(11)
PROC 1 DSN DEBUG
/*- SETUP FOR DEBUG IF REQUESTED ------------------------------------*/
CONTROL NOMSG NOLIST NOFLUSH END(ENO) NOCONLIST NOPROMPT
  IF &DEBUG = DEBUG THEN +
    CONTROL MSG LIST NOFLUSH END(ENO) PROMPT SYMLIST CONLIST
/*- END OF SETUP ----------------------------------------------------*/
RNV CLIST
RNV = Recatalog Non-VSAM.
Use this CLIST to recatalog non-VSAM SMS-managed datasets.
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* RNV: DEFINE RECATALOG OF A NONVSAM SMS-MANAGED ENTRY. */
/* TO BE USED IN ISPF 3.4 ONLY (VOLUME LIST) */
/* YOU MUST KNOW THE VOLSER WHERE ENTRY RESIDES BEFORE! */
/* JUST TYPE 'RNV' IN FRONT OF DATASETNAME TO RECATALOG */
/* */
/* EG: %RNV MY.NONVSAM.DATASET */
/* */
/* YOU MAY USE THE 'DEBUG' OPTION TO SEE ALL CLIST MSGS. */
/* */
/* RETURN CODES: SEE IDCAMS DEFINE RECATALOG RETURN CODES */
/* */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
ISPEEXEC VGET (ZDLDEV ZDLVOL)
CONTROL MSG LIST
DEF NONVSAM(NAME(&DSN) RECATALOG DEVT(&ZDLDEV) VOL(&ZDLVOL))
PROC 1 DSN DEBUG
/* SETUP FOR DEBUG IF REQUESTED ------------------------------------*/
CONTROL NOMSG NOLIST NOFLUSH END(ENO) NOCONLIST NOPROMPT
IF &DEBUG = DEBUG THEN +
CONTROL MSG LIST NOFLUSH END(ENO) PROMPT SYMLIST CONLIST
/* END OF SETUP ---------------------------------------------------*/

DVVR CLIST

DVVR = Delete VSAM Volume Record.

Use this CLIST to delete a VVR (VSAM Volume Record) from VVDS.

ISPEXEC VGET ZDLVOL
/* TO DELETE A VVR, SYS1.VVDS OF THE VOLUME MUST BE ALLOCATED */
ALLOC F(SC) OLD VOL(&ZDLVOL) UNIT(SYSDA) DA('SYS1.VVDS.V&ZDLVOL') REU
CONTROL MSG LIST
DEL &DSN FILE(SC) VVR
SET &RC=&LASTCC
IF &RC = Ø THEN DO
WRITE ***********************************************************
WRITE * VVDS VVR ENTRY SUCCESSFULLY REMOVED FOR &DSN
WRITE ***********************************************************
EXIT CODE(Ø)
ENDO
ELSE DO
WRITE * ***********************************************
WRITE * VVDS VVR ENTRY NOT REMOVED FOR &DSN, RETURN CODE=&RC
WRITE * ***********************************************
EXIT CODE(&RC)
ENDO
FREE F(SC)
END

DNVR CLIST

DNVR = Delete Non-VSAM Volume Record.

Use this CLIST to delete an NVR (Non-VSAM Volume Record) from a VVDS.

PROC 1 DSN DEBUG
/*- SETUP FOR DEBUG IF REQUESTED --------------------------*/
CONTROL NOMSG NOLIST NOFLUSH ENDO NOCONLIST NOPROMPT
IF &DEBUG = DEBUG THEN +
CONTROL MSG LIST NOFLUSH ENDO PROMPT SYMLIST CONLIST
/*- END OF SETUP ------------------------------------------*/
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/*                                                                 */
/*  DNVR: TO DELETE UNCATALOGED NONVSAM SMS ENTRIES          */
/*        USING DELETE NVR.                                   */
/*        TO BE USED IN ISPF 3.4 ONLY (VOLUME LIST)           */
/*        YOU MUST KNOW THE VOLSER WHERE ENTRY RESIDES BEFORE!*/
/*                                                                 */
/*  EG:     %DNVR   MYNVSAM.UNCATLG.SMS.DATASET              */
/*                                                                 */
/*  RETURN CODES: SEE IDCAMS DELETE NVR RETURN CODES         */
/*                                                                 */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
ISPEXEC VGET ZDLVOL
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/*                                                                 */
/*  TO DELETE A NVR, SYS1.VVDS OF THE VOLUME MUST BE ALLOCATED*/
/*                                                                 */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
ALLOC F(SC) OLD VOL(&ZDLVOL) UNIT(SYSDA) DA('SYS1.VVDS.V&ZDLVOL') REU
CONTROL MSG LIST
DEL &DSN FILE(SC) NVR
SET &RC=&LASTCC
IF &RC = 0 THEN DO
WRITE * ***********************************************
WRITE * VVDS NVR ENTRY SUCCESSFULLY REMOVED FOR &DSN
/*                                   */
What LPAR?

This REXX can be used to check what LPAR and Boxid (from HCD) an MVS system is running on. As we give non-specific LPAR names (eg LPARA1) it can be worth double-checking before IPLs. This currently runs on an OS/390 2.10 system.

```rexx
/* +--------------------------- REXX ------------------------------+ */
/* + REXX EXEC to locate LPAR and BOX names...                    + */
/* +---------------------------------------------------------------+ */
Trace o
SMFID:
  smca = d2x(c2d(storage(10,4))+197) /* get SMCA from CVT */
  smfid = d2x(c2d(storage(smca,3))+16)
  smfid = storage(smfid,4)
IPA:
  ecvt = d2x(c2d(storage(10,4))+140) /* -> ECVT @CVT+X'8C' */
  ipa = d2x(c2d(storage(ecvt,4))+392) /* -> IPA @ECVT+X'188' */
  ipa = d2x(c2d(storage(ipa,4)))
  ipadata = storage(ipa,40) /* IPA data */
  boxid = Substr(ipadata,25,8)
  lpar = Substr(ipadata,33,8)
  say " "
  say " =====MVS"smfid"=====
  say " BOX ID: "boxid
  say " LPARNAME: "lpar
  say " ================"
  say " "
Exit
```

Grant Carson
Systems Programmer (UK) © Xephon 2003
Bar code printing from PL/I programs

With bar codes, every single product has its own unique identification so identifying each product using a bar code scanner is very easily and quickly done. This article will explain how to create bar codes on an MVS page printer using the PL/I programming language.

Documents that can be printed on page printers using IBM Print Services Facility (PSF) belong to the categories line-mode documents, fully-composed page documents, and mixed-mode documents.

Traditional line data is formatted using Page Definition containing information such as dimensions of the page, position of each record on it, and fonts to be used. On the other hand, fully-composed page documents (or MO:DCA documents) are based on Mixed Object Document Content Architecture and can include different kinds of data objects – text, graphics, image, and bar code. The syntax and semantics of each type of data object are defined by its corresponding Object Content Architecture (for example, Bar Code Object Content Architecture (BCOCA) is used to describe and generate bar code symbols). So, a MO:DCA document represents a mixture of specific data objects and control data structures, called structured fields, used to specify the formatting requirements. Such a document, described by an ordered stream of structured fields based on MO:DCA rules, can be interchanged with other applications or, in order to be printed, transformed by PSF directly into the corresponding IPDS (Intelligent Printer Data Stream) commands. Fully-composed documents can be produced by some general-purpose formatting software packages like DCF, or by customized applications developed using the AFP Application Toolbox.

In some situations an application that produces just line data needs to be changed by having some kind of data object (image, graphics, bar code) added to the final output. This can be accomplished by intermixing line data with the appropriate set of
MO:DCA structured fields that define the required data object. Page Definition is also needed in order to format only the line data part of the print file. The output produced this way is called a mixed-mode document.

A Structured Field (SF), interpreted as an MO:DCA command, occupies one record and consists of an introducer followed by the corresponding data. The introducer identifies the specific command, and includes its total length and some additional control information. The structure of the data part varies according to the type of the structured field and can include fixed parameters, keywords, triplets, etc. In order to be recognized by PSF, the structured fields must be prefixed with an X'5A' character, while line data, if intermixed with structured fields, must start with the appropriate carriage control character. Since structured fields vary in length, if a decision is made to use fixed-length records (as in our case), padding with blanks is needed to the required length.

The introducer, beside some optional fields, includes the following mandatory fields:

- **SFLENGTH (2 bytes)** – the total length of the SF including this field and optional padding field (the starting byte, X'5A', is not taken into account).
- **SFID (3 bytes)** – the type of the SF.
- **SFFLAG (8 bits)** – attributes of SF indicating the existence of a padding, extension, segmentation (bit ‘1’ in the fifth position means padding is in use).
- **SFRESERV (2 bytes)** – reserved field (X'0000').

The structured field’s data (max 32759 bytes) depends on the SF type. The padding field, if present, contains blanks except in the last byte, where its length (including that last byte) is stored.

In our case the existing traditional line data application had to be changed in order to produce a unique bar code on each page. So the block of structured fields that describes the specific bar code symbol is included with other related line data on the page. They
have a special structure and sequence. In our application, only mandatory structured fields are used and their description is shown below.

PAGE PRESENTATION PAGE

A physical printer page is mapped into a logical Page Presentation Space. It could include different data objects and one of them could be a Bar Code Object. A Page Presentation Page is illustrated in Figure 1.

The Bar Code Presentation Space represents a two-dimensional conceptual space where one or more bar code symbols are generated. It is mapped into a rectangular area in the Page Presentation Space, called Bar Code Object Area.

The lengths of the corresponding coordinate systems can be measured using a unit base of 10 inches or 10 centimeters. Logical unit (L-unit) represents a unit of linear measurement expressed with a unit base and units per unit-base. In our application, since we use the unit base of 10 inches and the number of units per 10 inches is 2,400, the L-unit equals 1/240 inch.

BAR CODE OBJECT

Bar Code Object structure

Begin Bar Code Object (BBC,D3A8EB)
The Begin Bar Code Object record begins and names the bar code data object.

Begin Object Environment Group (BOG,D3A8C7)
The Begin Object Environment Group record begins and names an Object Environment Group and establishes the environment parameters for the object. The scope of an Object Environment Group is its containing object. The object containing the Begin Object Environment Group record must also contain a subsequent
matching End Object Environment Group record.

**Object Area Descriptor (OBD,D3A66B)**

The Object Area Descriptor record specifies the size and attributes of an object area. The structured field's data part includes the following mandatory triplets in any order: X'43', X'4B', and X'4C'.

Description Position Triplet X'43' associates OBD with OBP SF:
- Byte(1) = X'03'X – the length of the triplet.
- Byte(2) = X'43'X – the identification of the triplet.
- Byte(3) identifies the corresponding OBP record (X'01'-X'7F') (the same as in byte(1) of the data part of the OBP record).

Measurement Units Triplet X'4B' defines the measurement units of the object area:
Object Area Size Triplet X'4C' defines the X and Y dimensions of the object area:

- Byte(1) = X'09'X – the length of the triplet.
- Byte(2) = X'4C'X – the identification of the triplet.
- Byte(3) = X'02' – the actual object area size type.
- Byte(4-6) = X dimension of the object area in L-units.
- Byte(7-9) = Y dimension of the object area in L-units.

Object Area Position (OBP,D3AC6B)

An Object Area Position record contains one Object Area Position group. An Object Area Position group specifies the origin (the top-left corner) and orientation of the object area within the Page Presentation Space and the origin and orientation of the Data Presentation Space within the object area.

The structure of the record’s data part is:

- Byte(1) = the object area position identifier (the same as in byte(3) in the triplet X'43' above).
- Byte(2) = X'17' – total length of this field and the subsequent fields in data part (23).
• Byte(3-5) = X coordinate of the origin of the object area in L-units.
• Byte(6-8) = Y coordinate of the origin of the object area in L-units.
• Byte(9-10) = rotation of the X axis of the object area relating to the X axis of the reference coordinate system (X'0000' – 0 degrees, X'2D00' – 90 degrees, X'5A00' – 180 degrees, X'8700' – 270 degrees).
• Byte(11-12) = rotation of the Y axis of the object area relating to the X axis of the reference coordinate system.
• Byte(13) = X'0000' – reserved.
• Byte(14-16) = X coordinate, in L-units, of the origin of the object presentation space within the object area.
• Byte(17-19) = Y coordinate, in L-units, of the origin of the object presentation space within the object area.
• Byte(20-21) = rotation of the X axis of the object presentation space relating to the X axis of object area.
• Byte(22-23) = rotation of the Y axis of the object presentation space relating to the X axis of object area.
• Byte(24) = reference coordinate system (in our case X'01' for page coordinate system with standard origin).

Bar Code Data Descriptor (BDD,D3A6EB)

The Bar Code Data Descriptor record contains the descriptor data for a bar code data object.

The data portion of the structured field has the following format:

• Byte(1) = the unit base (X'00' – 10 inches, X'01' – 10cm).
• Byte(2) = X'00 – reserved.
• Byte(3-4) = the number of units per unit base for Xbc axis (X'0960'=2400).
• Byte(5-6) = the number of units per unit base for Ybc axis (X'0960'=2400).
• Byte(7-8) = the width of the bar code presentation space in L-units.
• Byte(9-10) = the length of the bar code presentation space in L-units.
• Byte(11-12) = X'0000' – reserved.
• Byte(13) = bar code type (for Code 39, with check digit X'02', without X'01').
• Byte(14) = bar code modifier (the code depends on check digit presence, algorithm, and placement for the specified symbology).
• Byte(15) = font for HRI (human-readable interpretation of bar code characters) (X'FF' default).
• Byte(16-17) = colour of the bar code symbol elements (X'FFFF' device default).
• Byte(18) = module width in mils (thousandths of an inch) of the smallest defined bar code element.
• Byte(19-20) = height of bar code elements in L-units.
• Byte(21) = height multiplier (multiplied by height of bar code elements makes total height of bar code).
• Byte(22-23) = ratio of the wide-element dimension to the narrow- element dimension for bar code symbologies when only two different size elements exist. This parameter is the binary representation of a decimal number of the form n.nnnn; the decimal point follows the first significant digit. (X'00C8'=2.00).

End Object Environment Group (EOG,D3A9C7)

The End Object Environment Group record terminates the definition of an Object Environment Group initiated by a Begin
Object Environment Group record. If a name is specified, it must match the name in the most recent Begin Object Environment Group record.

Bar Code Data (BDA,D3EEEB)
The Bar Code Data record contains the parameters to position the bar code symbol within a bar code presentation space and the data to be represented by the bar code symbol. The structure of a data part follows:

- Bit(1-8) = the parameters of the bar code symbol (like presence and location of HRI and presence of asterisk).
- Byte(2-3) = the Xbc coordinate of the bar code symbol origin in the bar code presentation space in L-units.
- Byte(4-5) = the Ybc coordinate of the bar code symbol origin in the bar code presentation space in L-units.
- Byte(6-n) = data to be encoded, depending on the symbology.

End Bar Code Object (EBC,D3A9EB)
The End Bar Code Object record terminates the current bar code object initiated by a Begin Bar Code Object record. A matching Begin Bar Code Object record must appear within the containing structure at some location preceding the End Bar Code Object record.

EXAMPLE PL/I PROCEDURE

PRINTBarcode: PROC;
/*******************************************************************/
/* PRINTING OF BARCODE ON LASER PRINTER FROM PL/1 PROGRAM */
/*******************************************************************/
/* DCL FILEØ1 FILE OUTPUT; Declared in calling procedure */
/*/ FIRST RECORD << BBC >> BEGIN BAR CODE OBJECT */
DCL BBC_SF_RED CHAR(133) DEF BBC_SFI;
DCL 1 BBC_SFI,
    2 BBC_START CHAR(1), /* 5A */
    2 BBC_LENGTH CHAR(2), /* LENGTH OF OUTPUT RECORD */
    2 BBC_SF_ID CHAR(3), /* IDENT OF EVERY RECORD*/
2 BBC_FLAG CHAR(1),
2 BBC_RESERV CHAR(2), /* RESERVED FIELD */
2 BBC_DATA CHAR(124);
DCL BBC_PADLEN CHAR(1) DEF BBC_SFI POS(133);
DCL BBC_SF_FL BIT(8) BASED(ADDR(BBC_SFI.BBC_FLAG));

/* SECOND RECORD << BOG >> BEGIN OBJECT ENVIRONMENT GROUP */
DCL BOG_SF_RED CHAR(133) DEF BOG_SFI;
DCL 1 BOG_SFI,
  2 BOG_START CHAR(1), /* 5A */
  2 BOG_LENGTH CHAR(2),
  2 BOG_SF_ID CHAR(3),
  2 BOG_FLAG CHAR(1),
  2 BOG_RESERV CHAR(2),
  2 BOG_DATA CHAR(124);
DCL BOG_PADLEN CHAR(1) DEF BOG_SFI POS(133);
DCL BOG_SF_FL BIT(8) BASED(ADDR(BOG_SFI.BOG_FLAG));

/* THIRD RECORD << OBD >> OBJECT AREA DESCRIPTOR */
DCL OBD_SF_RED CHAR(133) DEF OBD_SFI;
DCL 1 OBD_SFI,
  2 OBD_START CHAR(1), /* 5A */
  2 OBD_LENGTH CHAR(2),
  2 OBD_SF_ID CHAR(3),
  2 OBD_FLAG CHAR(1),
  2 OBD_RESERV CHAR(2),
  2 OBD_DATA,
  3 OBD_TRIPLET_X43,
    4 OBD_X43_LENGTH CHAR(1),
    4 OBD_X43_ID_TRI CHAR(1),
    4 OBD_X43_ID_OBP CHAR(1),
  3 OBD_TRIPLET_X4B,
    4 OBD_X4B_LENGTH CHAR(1),
    4 OBD_X4B_ID_TRI CHAR(1),
    4 OBD_X4B_MEASUR_UNITS_X CHAR(1),
    4 OBD_X4B_MEASUR_UNITS_Y CHAR(1),
    4 OBD_X4B_DOT_PER_X CHAR(2),
    4 OBD_X4B_DOT_PER_Y CHAR(2),
  3 OBD_TRIPLET_X4C,
    4 OBD_X4C_LENGTH CHAR(1),
    4 OBD_X4C_ID_TRI CHAR(1),
    4 OBD_X4C_SIZE_OBJ CHAR(1),
    4 OBD_X4C_X_DIM_OBJ CHAR(3),
    4 OBD_X4C_Y_DIM_OBJ CHAR(3),
  3 OBD_REST CHAR(104);
DCL OBD_PADLEN CHAR(1) DEF OBD_SFI POS(133);
DCL OBD_SF_FL BIT(8) BASED(ADDR(OBD_SFI.OBD_FLAG));

/* FOURTH RECORD << OBP >> OBJECT AREA POSITION */
DCL OBP_SF_RED CHAR(133) DEF OBP_SFI;
DCL 1 OBP_SFI,
   2 OBP_START CHAR(1), /* 5A */
   2 OBP_LENGTH CHAR(2),
   2 OBP_SF_ID CHAR(3),
   2 OBP_FLAG CHAR(1),
   2 OBP_RESERV CHAR(2),
   2 OBP_DATA,
      3 OBP_ID CHAR(1),
      3 OBP_MEANINGFUL_LENGTH CHAR(1), /* LENGTH FROM HERE */
      3 OBP_X_START_OBJ CHAR(3),
      3 OBP_Y_START_OBJ CHAR(3),
      3 OBP_X_ROTATION_OBJ CHAR(2),
      3 OBP_Y_ROTATION_OBJ CHAR(2),
      3 OBP_RESERVED CHAR(1),
      3 OBP_X_START_BARC CHAR(3),
      3 OBP_Y_START_BARC CHAR(3),
      3 OBP_X_ROTATION_BARC CHAR(2),
      3 OBP_Y_ROTATION_BARC CHAR(2),
      3 OBP_REF_SYST CHAR(1),
      3 OBP_REST CHAR(100);
DCL OBP_PADLEN CHAR(1) DEF OBP_SFI POS(133);
DCL OBP_SF_FL BIT(8) BASED(ADDR(OBP_SFI.OBP_FLAG));

/* FIFTH RECORD << BDD >> BAR CODE DATA DESCRIPTOR */
DCL BDD_SF_RED CHAR(133) DEF BDD_SFI;
DCL 1 BDD_SFI,
   2 BDD_START CHAR(1), /* 5A */
   2 BDD_LENGTH CHAR(2),
   2 BDD_SF_ID CHAR(3),
   2 BDD_FLAG CHAR(1),
   2 BDD_RESERV CHAR(2),
   2 BDD_DATA,
      3 BDD_MEASUR_UNITS CHAR(1),
      3 BDD_RESERVED1 CHAR(1),
      3 BDD_DOT_PER_X CHAR(2),
      3 BDD_DOT_PER_Y CHAR(2),
      3 BDD_WIDTH_BARCODE CHAR(2),
      3 BDD_LENGTH_BARCODE CHAR(2),
      3 BDD_RESERVED2 CHAR(2),
      3 BDD_TYPE_BARCODE CHAR(1),
      3 BDD_BARCOD_MODIF CHAR(1),
      3 BDD_LABEL_FONT CHAR(1),
      3 BDD_BARCODE_COLOR CHAR(2),
      3 BDD_WIDTH_MIN_ELEM CHAR(1),
      3 BDD_HEIGHT_ELEM CHAR(2),
      3 BDD_MULTPLI_KOEF CHAR(1),
      3 BDD_RATIO CHAR(2),
      3 BDD_REST CHAR(101);
DCL BDD_PADLEN CHAR(1) DEF BDD_SFI POS(133);
DCL BDD_SF_FL BIT(8) BASED(ADDR(BDD_SFI.BDD_FLAG));

/* SIXTH RECORD << EOG >> END OBJECT ENVIRONMENT GROUP */
/* END OF << BOG >> */

DCL EOG_SF_RED CHAR(133) DEF EOG_SFI;
DCL 1 EOG_SFI,
   2 EOG_START CHAR(1), /* 5A */
   2 EOG_LENGTH CHAR(2),
   2 EOG_SF_ID CHAR(3),
   2 EOG_FLAG CHAR(1),
   2 EOG_RESERV CHAR(2),
   2 EOG_DATA CHAR(124);
DCL EOG_PADLEN CHAR(1) DEF EOG_SFI POS(133);
DCL EOG_SF_FL BIT(8) BASED(ADDR(EOG_SFI.EOG_FLAG));

/* SEVENTH RECORD << BDA >> BAR CODE DATA */
DCL BDA_SF_RED CHAR(133) DEF BDA_SFI;
DCL 1 BDA_SFI,
   2 BDA_START CHAR(1), /* 5A */
   2 BDA_LENGTH CHAR(2),
   2 BDA_SF_ID CHAR(3),
   2 BDA_FLAG CHAR(1),
   2 BDA_RESERV CHAR(2),
   2 BDA_DATA,
   3 BDA_LABEL_EXISTS CHAR(1),
   3 BDA_X_START_OF_SYMBOL CHAR(2),
   3 BDA_Y_START_OF_SYMBOL CHAR(2),
   3 BDA_BARCODE_LABEL,
       4 VV CHAR(2),
       4 VBROJ CHAR(5),
       4 VIRGULE CHAR(1),
       4 RBRVB CHAR(3),
       4 YU CHAR(2),
   3 BDA_REST CHAR(106);
DCL BDA_PADLEN CHAR(1) DEF BDA_SFI POS(133);
DCL BDA_SF_FL BIT(8) BASED(ADDR(BDA_SFI.BDA_FLAG));
DCL BDA_OZNIKA BIT(8) BASED(ADDR(BDA_SFI.BDA_DATA));

/* EIGHTH RECORD << EBC >> END BAR CODE OBJECT */
DCL EBC_SF_RED CHAR(133) DEF EBC_SFI;
DCL 1 EBC_SFI,
   2 EBC_START CHAR(1), /* 5A */
   2 EBC_LENGTH CHAR(2),
   2 EBC_SF_ID CHAR(3),
   2 EBC_FLAG CHAR(1),
   2 EBC_RESERV CHAR(2),
   2 EBC_DATA CHAR(124);
DCL EBC_PADLEN CHAR(1) DEF EBC_SFI POS(133);
DCL EBC_SF_FL BIT(8) BASED(ADDR(EBC_SFI.EBC_FLAG));

BBC_SF_RED = (133)' ';
BOG_SF_RED = (133)' ';
OBD_SF_RED = (133)' ';
OBP_SF_RED = (133)' ';
BDD_SF_RED = (133)' ';
EOG_SF_RED = (133)' ';
BDA_SF_RED = (133)' ';
EBC_SF_RED = (133)' ';

/* RECORD << BBC >> BEGIN BAR CODE OBJECT */
BBC_START = '5A'X;       /* 9Ø */
BBC_RESERV = 'ØØØØ'X;
BBC_SF_FL = 'ØØØØ1ØØØ'B; /* 5. BIT MUST BE 1 */
BBC_SF_ID = 'D3A8EB'X; /* RECORD ID */
/* YOU CAN CHANGE THIS FIELD BUT IT HAS TO BE THE SAME AT THE END */
BBC_DATA = 'KOVBARCO'; /* Name of the Bar Code Object */
BBC_LENGTH = 'ØØ84'X; /* 132 */
BBC_PADLEN = '74'X; /* 116 LENGTH SFDATA, LENGTH OF "KOVBARCO" =8 */

WRITE FILE(FILE01) FROM(BBC_SF_RED);

/* RECORD << BOG >> BEGIN OBJECT ENVIRONMENT GROUP */
BOG_START = '5A'X; /* 9Ø */
BOG_RESERV = 'ØØØØ'X;
BOG_SF_FL = 'ØØØØ1ØØØ'B; /* 5. BIT MUST BE 1 */
BOG_SF_ID = 'D3A8C7'X; /* RECORD ID */
/* YOU CAN CHANGE THIS FIELD BUT IT HAS TO BE THE SAME AT THE END */
BOG_DATA = 'OEGKOVER';
BOG_LENGTH = 'ØØ84'X; /* 132 */
BOG_PADLEN = '74'X; /* 116 */

WRITE FILE(FILE01) FROM(BOG_SF_RED);

/* RECORD << OBD >> OBJECT AREA DESCRIPTOR */
OBD_START = '5A'X;
OBD_RESERV = 'ØØØØ'X;
OBD_SF_FL = 'ØØØØ1ØØØ'B; /* 5. BIT MUST BE 1 */
OBD_SF_ID = 'D3A66B'X; /* RECORD ID */
OBD_TRIPLET_X43.OBD_X43_LENGTH = '03'X; /* 3 */
OBD_TRIPLET_X43.OBD_X43_ID_TRI = '43'X; /* 67 */
OBD_TRIPLET_X43.OBD_X43_ID_OBP = '01'X; /* 1 */

OBD_TRIPLET_X48.OBD_X48_LENGTH = '08'X; /* 8 */
OBD_TRIPLET_X48.OBD_X48_ID_TRI = '4B'X; /* 75 */
OBD_TRIPLET_X48.OBD_X48_MEASUR_UNITS_X = '00'X; /* 0 10 INCHES */
OBD_TRIPLET_X4B.OBD_X4B_MEASUR_UNITS_Y = 'Ø0'X; /* 0 10 INCHES */
OBD_TRIPLET_X4B.OBD_X4B_DOT_PER_X = '0960'X; /* 2400 */
OBD_TRIPLET_X4B.OBD_X4B_DOT_PER_Y = '0960'X; /* 2400 */

OBD_TRIPLET_X4C.OBD_X4C_LENGTH = '09'X; /* 9 */
OBD_TRIPLET_X4C.OBD_X4C_ID_TRI = '4C'X; /* 76 */
OBD_TRIPLET_X4C.OBD_X4C_SIZE_OBJ = '02'X; /* 2 */
OBD_TRIPLET_X4C.OBD_X4C_X_DIM_OBJ = '00021C'X; /* 540=2,25 INCHES */
OBD_TRIPLET_X4C.OBD_X4C_Y_DIM_OBJ = '00000F'X; /* 240=1 INCHES */

OBD_LENGTH = '0084'X; /* 132 */
OBD_PADLEN = '68'X; /* 104 */

WRITE FILE(FILEØ1) FROM(OBD_SF_RED);
/* RECORD << OBP >> OBJECT AREA POSITION */
OBP_START = '5A'X;
OBP_RESERV = 'Ø000'X;
OBP_SF_FL = '00001000'B; /* 5. BIT MUST BE 1 */
OBP_SF_ID = 'D3AC6B'X; /* RECORD ID */
BP_LENGTH = '0084'X; /* 132 */
OBP_PADLEN = '64'X; /* 100 */

OBP_DATA.OBP_ID = '01'X; /* THE SAME AS 3. BYTE U X43 TRI */
OBP_DATA.OBP_MEANINGFUL_LENGTH = '17'X; /* 23 */
OBP_DATA.OBP_X_START_OBJ = '00000CC'X; /* =204=0.85 INCHES */
OBP_DATA.OBP_Y_START_OBJ = '000744'X; /* =1860=7.75 INCHES */
OBP_DATA.OBP_X_ROTATION_OBJ = '8700'X; /* 30 DEGREES */
OBP_DATA.OBP_Y_ROTATION_OBJ = '0000'X; /* 0 DEGREES */
OBP_DATA.OBP_RESERVED = '00'X;
OBP_DATA.OBP_X_START_BARC = '000000'X;
OBP_DATA.OBP_Y_START_BARC = '000000'X;
OBP_DATA.OBP_X_ROTATION_BARC = '0000'X; /* 0 DEGREES */
OBP_DATA.OBP_Y_ROTATION_BARC = '2000'X; /* 90 DEGREES */
OBP_DATA.OBP_REF_SYST = '01'X;

WRITE FILE(FILEØ1) FROM(OBP_SF_RED);
/* RECORD << BDD >> BAR CODE DATA DESCRIPTOR */
BDD_START = '5A'X;
BDD_RESERV = '0000'X;
BDD_SF_FL = '00001000'B; /* 5. BIT MUST BE 1 */
BDD_SF_ID = 'D3A6EB'X; /* RECORD ID */

BDD_DATA.BDD_MEASUR_UNITS = '00'X; /* 10 INCHES */
BDD_DATA.BDD_RESERVED01 = '00'X;
BDD_DATA.BDD_DOT_PER_X = '0960'X; /* =2400 */
BDD_DATA.BDD_DOT_PER_Y = '0960'X; /* =2400 */
BDD_DATA.BDD_WDT_H_BARCODE = '021C'X; /* =540=2,25 INCHES */
BDD_DATA.BDD_LENGTH_BARCODE = '00F0'X; /* =240=1 INCHES */
BDD_DATA.BDD_RESERVED02 = '0000'X;
BDD_DATA.BDD_TYPE_BARCODE = '01'X; /* BAR CODE 39 */
BDD_DATA.BDD_BARCODE_MODIF = '01'X; /* WITHOUT CHECK DIGIT */
BDD_DATA.BDD_LABEL_FONT = 'FF'X; /* 255 DEFAULT FONT */
BDD_DATA.BDD_BARCODE_COLOR = 'FFFF'X; /* 65535 DEFAULT */
BDD_DATA.BDD_WIDTH_MIN_ELEM = '07'X; /* BIN */
BDD_DATA.BDD_HEIGHT_ELEM = '003C'X; /* =60=0,25 INCHES */
BDD_DATA.BDD_MULTIPLI_KOEF = '01'X; /* BIN */
BDD_DATA.BDD_RATIO = '00C8'X; /* =200 */

BDD_LENGTH = '0084'X; /* 132 */
BDD_PADLEN = '65'X; /* 101 */

WRITE FILE(FILEØ1) FROM(BDD_SF_RED);
/

/* RECORD << EOG >> END OBJECT ENVIRONMENT GROUP */
/* END OF << BOG >> */
EOG_START = '5A'X;
EOG_RESERV = '0000'X;
EOG_SF_FL = '00001000'B; /* 5. BIT MUST BE 1 */
EOG_SF_ID = 'D3A9C7'X; /* RECORD ID */
EOG_DATA = 'OEGKOVER';
EOG_LENGTH = '0084'X; /* 132 */
EOG_PADLEN = '74'X; /* 116 */

WRITE FILE(FILEØ1) FROM(EOG_SF_RED);
/

/* RECORD << BDA >> BAR CODE DATA */
BDA_START = '5A'X;
BDA_RESERV = '0000'X;
BDA_SF_FL = '00001000'B; /* 5. BIT MUST BE 1 */
BDA_SF_ID = 'D3EEEB'X; /* RECORD ID */
BDA_LABEL_EXISTS = '00000000'B;
BDA_DATA.BDA_X_START_OF_SYMBOL = '003C'X; /* =60=0,25 INCHES */
BDA_DATA.BDA_Y_START_OF_SYMBOL = '003C'X; /* =60=0,25 INCHES */

BDA_DATA.BDA_BARCODE_LABEL.VV = 'VV';
BDA_DATA.BDA_BARCODE_LABEL.VBROJ = '35789';
BDA_DATA.BDA_BARCODE_LABEL.VIRGULE = '/';
BDA_DATA.BDA_BARCODE_LABEL.RBRVB = '003';
BDA_DATA.BDA_BARCODE_LABEL.YU = 'YU';

BDA_LENGTH = '0084'X;
BDA_PADLEN = '6A'X;

WRITE FILE(FILE01) FROM(BDA_SF_RED);
/~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~/

/* RECORD << EBC >> END BAR CODE OBJECT */
EBC_START = '5A'X;
EBC_RESERV = '0000'X;
EBC_SF_FL = '0001000'B; /* 5. BIT MUST BE 1 */
EBC_SF_ID = 'D3A9C'B; /* RECORD ID */
EBC_DATA = 'KOVBARCO';
EBC_LENGTH = '0084'X;
EBC_PADLEN = '74'X;

WRITE FILE(FILE01) FROM(EBC_SF_RED);
/~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~/
END PRINT_BARCODE;

EXAMPLE JCL

//JOB0001 JOB CLASS=A,MSGCLASS=J,MSGLEVEL=(1,1),NOTIFY=&SYSUID
/**
* PROGRAM PRINT001 prints 17 lines per page and invokes
* PRINT_BARCODE procedure to print bar code symbol on each page
*/
/**
* STEP0001 EXEC PGM=PRINT001
**/SYSPRINT DD SYSOUT=* 
/** OUT01 OUTPUT CLASS=A,OUTDISP=KEEP,PAGEDEF=PD01 
/** FILE01 DD SYSOUT=(,,KOSL),OUTPUT=(*.OUT1), 
/** DCB=(BLKSIZE=133,RECFM=FBA,LRECL=133)

EXAMPLE PAGEDEF

//JOB0002 JOB CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1),NOTIFY=&SYSUID
/** JCL TO CREATE A "PAGEDEF" PD01 DIRECT=UP LANDSCAPE
/** 17 LINES ON PAGE
/** 6 LINES PER INCH
/** FONT L2CROE (COURIER LATIN, 10 PITCH 12 POINT)
/** STEP1 EXEC PGM=AKQPFFA,REGION=2048K
/** PAGELIB DD DSN=SYSP.PPFA.PAGELIB,PAGELIB,DISP=SHR 
/** SYSPRINT DD SYSOUT=* 
/** SYSIN DD *
/** SETUNITS 1 IN 1 IN
/** LINESP 6 LPI;
/** PAGEDEF PD01
/** WIDTH 8
/** HEIGHT 6
/** DIRECTION UP
Why not share your expertise and earn money at the same time? *MVS Update* is looking for REXX EXECs, macros, CLISTs, program code, JavaScript, etc, that experienced MVS practitioners have written to make their life, or the lives of their users, easier. We will publish it (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 at any of the addresses shown on page 2, or e-mailed to trevore@xephon.com.

A free copy of our *Notes for contributors* is available from our Web site at www.xephon.com/nfc.
MacKinney Systems has announced Version 4.0 of its Job and Syslog Facility (JSF).

This new version of JSF includes: date format used in most JSF panels and reports changed to Gregorian; improved performance when parameter CHKDIRBK = Y is being used; expanded security options including interface with a site’s ESM; improved support for z/OS six-digit job numbers; a new dynamic REFRESH command for the Jobname Browse List panel; ability to use a wildcard (*) for the Jobname on the Job History Restore panel; and added FBM support for JSF job reprints.

For further information contact:
MacKinney Systems, 2740 S Glenstone Ave, Suite 103, Springfield, MO 65804-3737 USA.
Tel: (417) 882 8012.

IBM has announced WebSphere Application Server for z/OS Version 5, which now has Java Enterprise Edition (J2EE) V1.3 compatibility, support for key Web services standards, deep integration with the z/OS operating system, and enhanced scalability.

Version 5 is designed to be functionally equivalent to the programming model of the WebSphere Application Server Network Deployment. This allows for greater flexibility in development and simplified management capabilities, while maintaining the tight integration and optimization for the z/OS and zSeries programs and products.

WebSphere Application Server for z/OS V5 is said to provide comprehensive utilization of resources, from hardware to IT personnel, through the use of open standards and integration with zSeries processes and procedures. It contains improvements to both deployment and management functions to help provide a friendlier and more flexible environment in a manner consistent with WebSphere Application Server Network Deployment V5.

WebSphere Application Server for z/OS V5 will be priced on the Value Unit metric, said to allow for more flexible pricing and to align more closely with other zSeries software charges, and there’s a new entry price option.

For further information contact your local IBM representative.

IBM has announced Tivoli Storage Manager, S/390 Edition V5.2, which adds LAN-free clients to Tivoli Storage Manager Server on z/OS. It is said to reduce the impact of back-ups by utilizing snapshot back-ups and the LAN-free enhancements include tape mount retention and device configuration.

There’s automatic control of Tivoli Storage Manager Recovery Log utilization, improved HSM usability, faster access time for AIX GPFS files systems during HSM recall, extended retention time for archived data, and back-up of open files or files locked from back-up.

For further information contact your local IBM representative.