April 2003

In this issue

3  Structured design approach to program messages
9  Using high-level Assembler (HLASM) SYSADATA for SORT SYMNAME processing
17 Query allocated datasets
21 Using PF keys – a shortcut to running CLISTs
25 Exploring IPCS exit services for customization
50 Parsing strings in Assembler programs
74 MVS news

© Xephon plc 2003
Editor
Trevor Eddolls

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

Contributions
When Xephon is given copyright, articles published in MVS Update are paid for at the rate of £170 ($260) per 1000 words and £100 ($160) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of £50 ($80) per 100 lines. In addition, there is a flat fee of £30 ($50) per article. To find out more about contributing an article, without any obligation, please download a copy of our Notes for Contributors from www.xephon.com/nfc.
Structured design approach to program messages

One of the simplest but most often overlooked aspects of program design is the use of messages, both for informational purposes and when error conditions occur. As a program designer, you have two sets of design criteria that you would like to address. The first set of criteria comes from the end user of the program or utility. They need messages that provide meaningful information concerning the progress of the program as well as meaningful information that can help them address any errors that may occur during the course of its execution. From the program developer’s perspective, we need a way to easily create and maintain the messages that the program will produce. With these criteria in mind, we have attempted to create a control section structure that can address both sets of requirements. Our belief is that if we can make the programming aspect easier to implement, the programmer is more likely to increase the number of messages issued, as well as the quality of the messages.

We first turned our attention to an individual message and asked ourselves, what are the requirements for an individual message? We made a couple of base assumptions concerning a message. The first is that the message will be output to a file that can ultimately be printed. Based on that assumption, each message will contain a carriage control byte at the beginning. The second assumption is that all of the messages will be the same maximum length. This is not a technical requirement, but more of a stylistic requirement to facilitate a structured design. The last requirement for a message is that it supports static fields as well as dynamic fields that can be populated during the program execution.

From these simple requirements, we set out to design a macro that could be used to create a message structure. The results of our efforts are shown below in the $EDFMDFN macro. If you look at the coding in the macro, you will see that it produces a standard structure for each message that it is invoked for. Three fields are always provided for each message. Each of these fields is
defined as a Y-style address constant. The three fields are the length of the message, the displacement to the actual beginning of the message, and the number of dynamic fields that are in the message. If dynamic fields are present in the message, the displacement to each of these fields will be defined next as consecutive Y-style address constants. Following the displacements will be the actual fields that comprise the message.

Let’s take a closer look at the macro and the invoking parameters so that we understand what we need to specify. The $EDTMDFN macro uses a combination of keywords as well as a freeform style of parameter specification. The keywords that are defined in the macro prototype are ID, MAXL, and TRACE. ID can be used to help structure the messages into a sequence of sorts. You can use it based on whatever your program design requirements dictate. We will demonstrate how we used it a little later in this article. The MAXL parameter is used to specify the maximum length of the message. This value will usually be related to the size of the buffer specified for a print or listing file. The TRACE parameter can be used if you want additional information placed into the Assembler listing so that you can see what the various values are within the macro. The actual message or message pieces are specified in a freeform style. They are freeform in the sense that you simply enclose them within quotes. The information within the quotes must begin with specific pieces of information because the macro logic is checking for specific patterns of information. The macro checks to see whether the freeform fields begin with the literals TXT= or DYN=. These two literals represent text and dynamic fields respectively. Please note that when we refer to a field as dynamic, what we really mean is that we want to create a placeholder within the message that we will potentially populate with information at a later time.

A couple of simple examples will help illustrate how to use the macro. Our first example will be a simple message with no dynamic fields. In this example we will let the maximum length default to 133 characters:

```assembly
$EDTMDFN ID=1,
```
Now that we have the macro $EDTMDFN for message definition, how can we use it to make our programming task easier? We have opted to use the macro to further facilitate a structure to any programs or utilities that we may develop. We have made a design decision that all our program messages will be defined in a separate control section. This keeps all of the messages compartmentalized in a single structure, and also allows us to easily adapt a message module over and over again. To maintain consistency, we have decided that the messages control section will always be named ME$AGE$. Let’s take a look at a simple example of what a ME$AGE$ module might look like:

```
MESSAGES CSECT                         CSECT NAME
MESSAGES AMODE 31                      SPECIFY AN ADDRESSING MODE
MESSAGES RMODE ANY                     SPECIFY THE RESIDENCY
     SPACE 1
     DC  AL4(A_NEXT-A_FIRST)     SIZE OF AN ENTRY
     DC  AL4((A_END-A_FIRST)/(A_NEXT-A_FIRST)) NUMBER OF ENTRIES
     SPACE 1
*----+----+----+----+----+----+----+----+----+----+----+----+----+----*
* EACH ENTRY IN THE TABLE CONSISTS OF THE MESSAGE NUMBER, AND THE *
* ADDRESS OF THE MESSAGE IN THE CSECT. THE TABLE STRUCTURE CAN *
* ACCOMMODATE 255 MESSAGES.                                     *
*----+----+----+----+----+----+----+----+----+----+----+----+----+----*
     SPACE 1
A_FIRST   DC  AL1(Ø1),AL4(MSG_1_B)
A_NEXT    DC  AL1(Ø2),AL4(MSG_2_B)
       DC  AL1(Ø3),AL4(MSG_3_B)
A_END    EQU   *
     SPACE 1
$EDTMDFN ID=1,
     'TXT=BCDSINVT-Ø1I ',
```

Here is a slightly more extensive invocation, again using the default length of 133 characters:

```
$EDTMDFN ID=2,
     'TXT=PROGMSG-Ø2(W) ',
     'TXT=The number of records written to the file = ',
     'DYN=123,456'
```

In this example, we have provided for a dynamic field that we can place the appropriate information into during the program execution.
Notice that we have coded the RMODE of the module as ANY. We have done this so that it assumes the residency mode of the program it is included in at linkage edit time. We have also used the Assembler to create a simple table structure that comprises the message number or id. The header of the table has two entries that provide us with the address of the first entry and the total number of entries in the table respectively. We have done this so that we can utilize a simple table look-up to find a message based on the message id. As can be seen in the example, we simply start with a message id of one and then just increment the id for each successive message. In a future article, we will discuss the simple routine that we use to locate a message in the ME$$AGE$ table. We hope that this discussion of program messages and potential ways to program for them in a structured manner has provided some useful insight and some coding techniques potentially of general use.

$EDTMDFN MACRO

MACRO
$EDTMDFN &ID=,                                                -
&MAXL=133,                                              -
&TRACE=NO
**********************************************************************
.*       $EDTMDFN IS A SIMPLE MACRO THAT CAN BE USED TO CREATE MES-   *
.*       SAGES IN A STANDARD LAYOUT STRUCTURE. IT WILL SUPPORT A     *
.*       COMBINATION OF STATIC AND DYNAMIC FIELDS. THE MINIMUM RE-   *
.*       QUIREMENTS FOR INVOKING THE MACRO ARE THE SPECIFICATION OF   *
.*       THE MESSAGE ID, AND AT LEAST ONE FIELD.                   *
.*       *                                                            *
.*       THE GENERATED STRUCTURE WILL ALWAYS CONTAIN THE FOLLOWING:  *
.*       *                                                            *
.*       DC Y(LENGTH OF THE MESSAGE)                                *

END ME$$AGE$
.* DC Y(DISPLACEMENT TO THE START OF THE MESSAGE)
.* DC Y(NUMBER OF DYNAMIC FIELDS)
.* IF THE NUMBER OF DYNAMIC FIELDS IS NON-ZERO, THEN ADDITIONAL
.* Y-TYPE CONSTANTS WILL BE GENERATED WHICH REPRESENT THE DISPLACEMENT TO EACH DYNAMIC FIELD.
.* DC Y(DISPLACEMENT TO DYNAMIC FIELD 1)
.* .
.* .
.* .
.* DC Y(DISPLACEMENT TO DYNAMIC FIELD N)
.* MESSAGE FIELD(S) WILL BE DEFINED NEXT
.*
.* DEFINE THE LOCAL SYMBOLS WE WILL NEED
*****************************************************************************
LCLA &DYN_CNT                NUMBER OF TEXT/DYNAMIC FIELDS
LCLA &EL                     USED FOR LENGTH CALCULATIONS
LCLA &NE                     NUMBER OF ENTRIES
LCLA &NI                     INDEX VARIABLE
LCLA &TLEN                   USED TO TEST TOTAL LENGTH
LCLC &DYN(1Ø)                USED TO SAVE TEXT/DYN ELEMENTS
LCLC &DSP(1Ø)                DISPLACEMENT INDICATOR
LCLC &LBL_B                  LABEL
LCLC &LBL_E                  LABEL
LCLC &LBL_M                  LABEL
*****************************************************************************
* INITIALIZE SOME OF OUR VARIABLES
*****************************************************************************
&DYN_CNT SETA Ø
&TLEN     SETA Ø
*****************************************************************************
* CHECK TO SEE WHETHER WE HAVE AT LEAST ONE ENTRY DEFINED
*****************************************************************************
&NE SETA N’&SYSLIST
AIF (&NE GT Ø).OKTST1
MNOTE 12,'*** ERROR $EDTMDFN MUST HAVE AT LEAST ONE TEXT ENTRY-DEFINED ***'
AGO .MEND
.OKTST1 ANOP
*****************************************************************************
* PROCESS THE PARMS AND DETERMINE WHETHER WE HAVE TXT OR DYN ENTRIES
*****************************************************************************
&NI SETA 1
.Loop1 ANOP
AIF ('&SYSLIST(&NI)'(2,4) EQ 'TXT=').PARMG
AIF ('&SYSLIST(&NI)'(2,4) EQ 'DYN=').EQDYN
MNOTE 12,'*** ERROR ENTRY MUST BEGIN WITH TXT= OR DYN= ***'
AGO .MEND
.EQDYN ANOP
&DYN_CNT SETA &DYN_CNT+1
&DSP(&NI) SETC 'MSG'.'_'.'&ID'.'_'.'&NI'
.PARMG ANOP
&EL SETA K'SYSLIST(&NI) - 6
&TLLEN SETA &TLLEN+&EL
&DYN(&NI) SETC 'SYSLIST(&NI)'(6,&EL)
&NI SETA &NI+1
AIF (&NI LE &NE).LOOP1
**********************************************************************
* TEST THE TOTAL LENGTH OF OUR MESSAGE. DO NOT WANT TO EXCEED MAX *
**********************************************************************
AIF (&TLLEN LE &MAXL-1).GOTRAC
MNOTE 12,'*** SPECIFIED LENGTH OF &TLLEN PLUS CARRIAGE CONTROL -
BYTE EXCEEDS THE MAX LENGTH OF &MAXL ***'
AGO .MEND
.GOTRAC ANOP
**********************************************************************
* TRACE LOOP TO DISPLAY INFORMATION IN THE ASSEMBLER LISTING         *
**********************************************************************
AIF ('&TRACE' EQ 'NO').NOTRAC
&NI SETA 1
.TRACE ANOP
MNOTE *
,'DYN(&NI) --> &DYN(&NI)'
&NI SETA &NI+1
AIF (&NI LE &NE).TRACE
.NOTRAC ANOP
&LBL_B SETC 'MSG'.'_'.'&ID'.'_''.B'
&LBL_E SETC 'MSG'.'_'.'&ID'.'_''.E'
&LBL_M SETC 'MSG'.'_'.'&ID'
&LBL_B DS ØH
DC Y(&LBL_E-&LBL_B) COMPUTE LENGTH OF THE MESSAGE
DC Y(&LBL_M-&LBL_B) COMPUTE DISP. TO START OF MSG.
DC Y(&DYN_CNT) SPECIFY NUMBER OF DYNAMIC FLDS.
AIF (&DYN_CNT EQ Ø).ELOOP2
**********************************************************************
* LOOP TO GENERATE THE NEEDED ADDRESS CONSTANTS                        *
**********************************************************************
&NI SETA 1
.LOOP2 ANOP
AIF ('&DSP(&NI)' EQ '').NODSP
DC Y(&DSP(&NI) - &LBL_B) DEFINE DISPLACEMENT
.NODSP ANOP
&NI SETA &NI+1
AIF (&NI LE &NE).LOOP2
.ELOOP2 ANOP
**********************************************************************
* LOOP TO GENERATE CONSTANTS                                           *
**********************************************************************
Using high-level Assembler (HLASM) SYSADAT A for SORT SYMNAMES processing

I frequently use the SORT utility to process data by means of its INCLUDE/OMIT facilities. In particular, I find myself dealing with SMF data quite regularly. For years, I have been using INCLUDE/OMIT to extract data from SMF, for fields which the IBM-supplied SMF dump/extract utility (IFASMFDP) did not provide extraction keywords. This works well for SMF records where the fields in question are not part of variable sections, and has sometimes been usable even in certain SMF record variable sections.

A short time ago, I found a feature of SORT (both DF/Sort and Syncsort for z/OS) that allows the creation of a symbol file. This symbol file, referred to with the DDname of SYMNAMES, is used to define record layouts so that the user can refer to fields and constants by their symbolically-defined names, rather than by their offset, length, and data types. I also came across a program on one of my favourite Web sites, www.planetmvs.com, that was written as an example of processing HLASM SYSADAT A files. Between the two discoveries came the idea of using HLASM
SYSADATA files, generated by assembling SMF record mapping macros, to build SORT SYMNAMES files.

The JCL shown below assembles a group of SMF mapping macros for the sole purpose of generating a SYSADATA file. This file is used as input to a REXX EXEC, called SCNADATA, shown following the JCL. SCNADATA makes use of two SYSADATA records types, called Symbol records and DC/DS records. This was required because I found that, after beginning to code what I thought was going to be a very easy REXX EXEC using only the symbol records, I found a snag that necessitated the use of the DC/DS records. Symbol records are generated only for fields that have symbols, as their name implies. Looking initially at the types of symbol record, I thought that unlabelled DC/DS statements would wind up with one of the 15 defined symbol types. This was an error of judgement. I found that unlabelled DC/DS statements would generate only DC/DS records, and that I would have to match up the records by their statement numbers in order to account for unlabelled fields.

The symbol names generated by SCNADATA will be the label names from the macros that get assembled, with some caveats. While Assembler labels can be up to 63 bytes in length, SORT SYMNAMES labels can only be up to 50 bytes. Therefore SCNADATA will truncate any label longer than 50 bytes to be only 50 bytes in length, and write comment statements in its output indicating the label truncation. Be aware that this can cause duplicate labels to be generated. Additionally, when processing DC/DS records, there are no associated labels for their fields. For these records, SCNADATA will generate its own label names in the form of $OFFnnnnn, where nnnnn is a 5-digit decimal number that corresponds to the offset of the DC/DS field within the assembled program.

The sample JCL can be used to create a single SYSADATA file corresponding to a single SMF mapping macro, or every defined IBM and user SMF record can be defined in a single assembly. I elected to store the SYSADATA files as PDS members, in some cases with one SMF mapping macro for one PDS member, in
other cases multiple related SMF mapping macros to a single
PDS member (such as SMF types 70 to 79, which are all the RMF
records). Either way, whenever there might be changes to any
SMF record mapping, either the individual SMF mapping macro
can be reassembled and processed by SCNADATA, or all 256
possible macros can be reassembled and processed in one
invocation. Of course, some offset/length/datatype coding would
still be required for tasks such as extracting data for jobnames
that begin with certain characters, such as ‘TSO’. This is
because the SMF mapping macros define the jobname fields as
eight characters. However, you can get around this, and still use
symbol names, if you add Assembler EQU statements for the
necessary fields and lengths prior to running the complete
jobstream. For example, to be able to extract SMF type 14 data
for jobs that begin with the characters ‘TSO’, you would need to
add only the following code within the Assembler input, preferably
after the IFASMFIR 14 statement in the Assembler deck:

```
SMF14JBN1_3 EQU SMF14JBN,3,C'C'
TSOPFX    DC CL3'TSO'
TYPE14X   EQU X'0E',1,C'X' alternative 1 (no storage needed)
TYPE14A   DC AL1(14) alternative 2 (decimal numbers)
```

This would define the field SMF14JBN1_3 as a 3-byte field, with
a data type of character, and the field TSOPFX as a 3-byte field
with a value of ‘TSO’. The TYPE14 fields can be coded in either
format, depending on the comfort, readability, and maintainability
of using decimal or hexadecimal values. The above statements
could also be hand-coded as SYMNAMES statements and
concatenated with any prior generated SCNADATA output. The
results either way would be the same, and fields could then be
referenced as follows:

```
INCLUDE COND=(SMF14RTY,EQ,TYPE14A,AND,
               SMF14JBN1_3,EQ,TSOPFX)
```

While this process works fine for SMF records that do not have
variable sections, those that do pose some problems. However,
I have found myself able to ‘cheat’ at handling such records. This
stems from the fact that, although the sequence of sections
cannot be guaranteed, they do tend to come out in somewhat
repeatable order. The order tends to be the same as that listed for the SMF records in the appropriate level of the IBM SMF manual. This works for the first variable section that follows a fixed section, as well as when there is only one repetition of a variable section, rather than multiple occurrences. By printing raw SMF records in dump format, you can begin to see the patterns that some SMF records take. Such printing can be performed using the IDCAMS utility PRINT statement, or DASD utilities such as DF/DSS or FDR/DSF. This information can then be used to gauge the effectiveness of the SYMNAMES record layouts being generated.

Please be aware that I have not tested the use of every possible type of SMF record being properly handled by SCNADATA, nor has it been used against other types of record mapping macros. Neither has it been coded for all possible permutations of data, because I wanted to keep the REXX code fairly simple. For instance, it is possible to code the following, which would not be correctly handled:

```
FIELD1 DC F'1,2,3,4'
FIELD2 DC 2F'1,2,3,4',2H'5,6,7,8'
```

This is because the SYSADATA DC/DS records get more complex when multiple operands and/or values are coded. SCNADATA will detect the presence of both multiple operands and/or multiple values. If any such fields are found, the program will issue a warning message and terminate. However, for most of the SMF records I have come across, this has not presented a problem.

Sample JCL to assemble a program and process SYSADATA output follows:

```bash
//ADATA    PROC M=
//HLASM    EXEC PGM=ASMA90, PARM='ADATA, NOOBJ ECT'
//SYSLIB   DD DISP=SHR, DSN=SYS1.MACLIB
//         DD DISP=SHR, DSN=SYS1.MODGEN
//         DD DISP=SHR, DSN=SYS2.MACLIB
//SYSUT1   DD DSN=&&SYSUT1, SPACE=(4096,(120,120),,,ROUND), UNIT=VIO
//SYSPRINT DD SYSOUT=*  
//SYSADATA DD DISP=SHR, DSN=userid.SYSADATA(&M)  
//SYSIN    DD DISP=SHR, DSN=userid.ASM(&M)
```
// I KJEFT01 EXEC PGM=I KJEFT01, DYNAMNBR=99,
// PARM='%SCNADATA ' 'userid.SYSADATA(&M)' '
// SYSPROC DD DISP=SHR, DSN=userid.EXEC
// SYSTSPRT DD SYSOUT=* 
// SYSTSIN DD DUMMY 
// PEND 
// ADATA1 EXEC ADATA, M=SMFSAMP 
// HLASM.SYSIN DD * 
$SMF000 DSECT
 IFASMFR 0
$SMF002 DSECT
 IFASMFR 2
$SMF003 DSECT
 IFASMFR 3
$SMF004 DSECT
 IFASMFR 4
$SMF005 DSECT
 IFASMFR 5
$SMF006 DSECT
 IFASMFR 6
$SMF007 DSECT
 IFASMFR 7
$SMF008 DSECT
 IFASMFR 8
$SMF009 DSECT
 IFASMFR 9
$SMF010 DSECT
 IFASMFR 10
$SMF011 DSECT
 IFASMFR 11
$SMF014 DSECT
 IFASMFR 14
$SMF017 DSECT
 IFASMFR 17
$SMF018 DSECT
 IFASMFR 18
$SMF019 DSECT
 IFASMFR 19
END
/*
// SORT EXEC PGM=SORT 
// SYSOUT DD SYSOUT=* 
// SORTOUT DD DISP=(NEW, PASS), UNIT=3390, SPACE=(CYL, (5, 1)), DSN=&&TT 
// SORTIN DD DISP=SHR, DSN=userid.SMFDATA 
// SYMNOUTS DD SYSOUT=* 
// SYMNAMES DD DISP=SHR, DSN=userid.SYSADATA(&M) 
// SYSIIN DD * 
OPTION VLSHRT
SORT FIELDS=(SMF14DTE, A, SMF14TME, A, SMF14SID, A), FILESZ=E20000
RECORD TYPE=V, LENGTH=(32756, 32756, 32756)
INCLUDE COND=(SMF14RTY,EQ,X'ØE',AND,
SMF14JBN,EQ,C'useridX ')

/*
SCNADATA REXX EXEC follows:

/*  rexx comment *** start standard header
SCNADATA Scan HLASM adata file to build SYMNAMES statements for SORT
  rexx comment *** end  standard header */
parse upper arg dsn
a = sysdsn(dsn)
if a ¬= "OK" then do
  say "Dataset" dsn "checking failed ("a")"
  exit 8
  end /* if a */
"ALLOC DD(SYSADATA) DA("dsn") SHR REUSE"
alloc_rc = rc
if alloc_rc ¬= Ø then do
  say "Dataset" dsn "allocation failed, rc="alloc_rc"
  exit alloc_rc
  end /* if alloc_rc */
"EXECIO * DISKR SYSADATA (STEM RECIN. FINIS"
execio_rc = rc
if execio_rc ¬= Ø then do
  say "Dataset" dsn "read failed, rc="execio_rc"
  exit alloc_rc
  end /* if allocio_rc */
"FREE DD(SYSADATA)"
/* process the SYSADATA file for Symbol and DC/DS records */
symrec = 'ØØ42'x /* Symbol record indicator */
dcdsrec = 'ØØ34'x /* DC/DS record indicator */
ordinary = 'ØD'x /* symbol type value for Ordinary */
equate = 'ØC'x /* symbol type value for EQU */
sym42. = "" /* Symbol record array */
sym34. = "" /* Symbol record array */
reco. = "" /* SYMNAMES record array */
sym42cnt = Ø /* Symbol record counter */
sym34cnt = Ø /* Symbol record counter */
reccnt = Ø /* SYMNAMES record counter */
do i = 1 to recin.Ø
  rectyp = substr(recin.i,2,2)
  if rectyp = symrec | rectyp = dcdsrec then nop
  else iterate
  if rectyp = symrec then do /* process Symbol records */
    symtype = substr(recin.i,21,1)
    /* only use symbol types Ordinary and EQU */
    if symtype = ordinary | symtype = equate then nop
    else iterate
    stmtnum = substr(recin.i,17,4)
end /* for i */
stmtnum = c2d(stmtnum)
sym42cnt = sym42cnt + 1
sym42cnt = max(stmtnum, sym42cnt)
sym42.stmtnumd = recin.i
iterate
end /* if rectyp */
else do
  /* process DC/DS records */
stmtnum = substr(recin.i, 25, 4)
stmtnum = c2d(stmtnum)
sym34cnt = sym34cnt + 1
sym34cnt = max(stmtnum, sym34cnt)
sym34.stmtnumd = recin.i
iterate
end /* else */
end /* do i */
sym42.Ø = sym42cnt
sym34.Ø = sym34cnt
/* process both the resulting Symbol and DC/DS record arrays */
maxrecs = max(sym34.Ø, sym42.Ø)
do i = 1 to maxrecs
  symlen = length(sym42.i)
dcdslen = length(sym34.i)
  select /* decide which record to use to build SYMNAMES statements */
    when symlen = Ø &,
      dcdslen = Ø then iterate /* format no record */
    when dcdslen = Ø | ,
      (dcdslen > Ø & symlen > Ø) then do /* format Symbol record */
        call proc_sym
        iterate
    end /* when dcdslen */
    when symlen = Ø then do /* format DC/DS record */
      call proc_dcds
      iterate
  end /* select when symlen/dcdslen */
end /* do i */
"EXECIO * DISKW SYMNAMES (STEM RECO. FINIS"
exit

proc_sym:
symtype = substr(sym42.i, 21, 1)
fldfmt = substr(sym42.i, 22, 1)
flddupx = substr(sym42.i, 23, 4)
fldlenx = substr(sym42.i, 27, 2)
length = max(x2d(c2x(flddupx))), 1) * x2d(c2x(fldlenx))
offsetx = substr(sym42.i, 33, 4)
select
  when symtype = ordinary then ,
    offset = x2d(c2x(offsetx)) + 1
when symtype = equate then do
    symflg = substr(sym42.i,37,1)
    if symflg = 'Ø'x then do
        offset = x2d(c2x(offsetx)) + 1
        offlen = ((length(offset) + 1) % 2) * 2
        end /* if symflg */
    else do /* make it an even # of digits */
        offset = strip(c2x(offsetx),"L","Ø")
        offlen = ((length(offset) + 1) % 2) * 2
        offset = "X"||right(offset,offlen,"Ø")||"" /*
        end /* else do */
    end /* when symtype = equate */
otherwise nop
end /* select when symtype */
lablen = x2d(c2x(substr(sym42.i,45,2)))
labtrunc1 = ""
labtrunc2 = ""
if lablen > 5Ø then do
    laborig = substr(sym42.i,47,lablen)
    labtrunc1 = "** Above label was truncated in length",
                "from" lablen "to 5Ø (original label below)"
    labtrunc2 = "** " laborig
    lablen = 5Ø
    end /* if lablen */
label = substr(sym42.i,47,lablen)
select /* set sort field format (default = BI) */
    when fldfmt = "C" then format = "CH"
    when fldfmt = "P" then format = "PD"
    when fldfmt = "Z" then format = "ZD"
    otherwise format = "BI" /* default to binary data type */
end /* select when fldfmt */
/* format the data line to be displayed */
select
    when symtype = ordinary then do /* Symbol ordinary label */
        recctnt = recctnt + 1
        reco.reccnt = label","offset","length","format
    end /* when symtype = ordinary */
when symtype = equate then do /* Symbol EQU name */
    recctnt = recctnt + 1
    if symflg ¬= '8Ø'x then /* Symbol absolute EQU */
        reco.reccnt = ""||label","offset","length","format
    else reco.reccnt = label","offset","length","format
    end /* when symtype = equate */
otherwise do
    recctnt = recctnt + 1
    reco.reccnt = label","offset "**OTHER-UNKNOWN**"
end /* otherwise */
end /* select when symtype */
if labtrunc1 ¬= "" then do
The following program was created to act as a subroutine for other Assembler programs in order to get allocation information for a particular dataset or DDname. For example, you are running a program and you need to know the datasetname for a given DDname, or you know the dsname but need the DDname.
The program issues an SVC99 (dynalloc) with function number 7 – get information allocation. It has only two parameters: the first is always the DDname, and the second is the dsname. If you know the DDname and want to know the dsname, supply the DDname and leave the dsname filled with spaces or low-values. Upon return, the dsname will contain the desired information. Or – the other way around – if you know the dsname and need the DDname, supply a blank DDname and fill in the dsname.

If you supply both parameters filled or both blank, the program returns 1 in register 15 (and 0 in register 0). If there is some other error, for example the dataset is not allocated, both R0 and R15 will contain the codes as set by SVC99. For details on error codes, refer to the dynalloc macro in the Application Development Guide: Authorized Assembler Language Programs, GC28-1645.

DYNALOC7 SOURCE CODE

*===================================================================*
* DYNALOC7 - DYNALOC FUNCTION 7 - Retrieve allocation information *
*            Returns datasetname for a DDname or vice versa.         *
*===================================================================*
* PARM1: DDNAME(8)    One parm is given, the other is returned. The *
* PARM2: DSNAME(44)   parm to be returned must be spaces or low val *
*                     uses upon entry. The return codes set are:     *
*                     R0=0, R15=0. Function completed.             *
*                     R0=0, R15=1. No parm or both parms supplied. *
*                     Others: as set by Dynalloc call.              *
*===================================================================*

&PROGRAM SETC 'DYNALOC7'
&PROGRAM CSECT
&PROGRAM AMODE 31
&PROGRAM RMODE 24
SAVE  (14,12)
LR    R12, R15
USING &PROGRAM, R12
ST    R13, SAVEA+4
LA    R11, SAVEA
ST    R11, 8(R13)
LR    R13, R11
B    MOVEPARM
DC    CL16' &PROGRAM 1.1'
DC CL8' &SYSDATE'

MOVE PARM DS ØH
LR R2, R1
L R3, Ø(0, R2)  R3: PARM1 address
MVC DDNAME, Ø(R3)  move DDname to dynalloc area
L R4, 4(0, R2)  R4: PARM2 address
MVC DSNAME, Ø(R4)  move Dsname to dynalloc area
MVI FLAG, X'ØØ'  Initialize flag and
MVC DYDDLENG, =X'ØØØ8'  default lengths
MVC DYDSLENG, =X'ØØ2C'

* TESTDD EQU *
CLC DDNAME, =CL8' '  DDname spaces or low-values?
BE TESTDSN  Yes, jump
CLC DDNAME, =XL8'ØØ'  BE TESTDSN
OI FLAG, C'1'  Set flag DD specified

* TESTDSN EQU *
CLC DSNAME, =CL44' '  DSname spaces or low-values?
BE TESTBOTH  No, jump
CLC DSNAME, =XL44'ØØ'  BE TESTBOTH
OI FLAG, C'2'  Set flag DSN specified

* TESTBOTH EQU *
CLI FLAG, C'1'
BE EXECDYN1
CLI FLAG, C'2'
BE EXECDYN2
L R15, =F'1'  Error: no param or both parms
XR R0, R0
B EXITØ  Return

* EXECDYN1 EQU *
MVC DYDDNAME, =X'ØØ01'  DDname given
MVC DYDSNAME, =X'ØØ05'  Ask for Dsname
LA R5, DDNAME  String address
XR R9, R9  Clear character counter
LH R6, =H'8'  Max length
BAL R10, FINDSPC  Find DDname length
STH R9, DYDDLENG  and store it for dynalloc
BAL R10, EXECDYN  Call dynalloc subroutine
LA R5, DSNAME  Load answer address
LH R6, =H'44'  and length
BAL R10, CLEARLOW  turn low-values to spaces
MVC Ø(44, R4), DSNAME  Move answer to parameter
B EXITØ

EXECDDY2 EQU *
MVC DYDDNAME, =X'0004' Ask for DDname
MVC DYDSNAME, =X'0002' Dsname given
LA R5, DSNAME String address
XR R9, R9 Clear character counter
LH R6, =H'44' Max length
BAL R10, FINDSPC Find dsname length
STH R9, DYDSLENG and store it for dynaloc
BAL R10, EXECDDY Call dynaloc subroutine
LA R5, DDNAME Load answer address
LH R6, =H'8' and length
BAL R10, CLEARLOW turn low-values to spaces
MVC Ø(8, R3), DDNAME Move answer to parameter
*
EXITØ EQU *
L R13, SAVEA+4 R0 (reason code) are kept as
LA R14, 12(R13) set by dynalloc.
LM R1, R12, 24(R13) If everything ok, R15 is zero.
BR R14
*
*===================================================================*
*        Subroutines                                                *
*===================================================================*
*
FINDSPC EQU *
CLI Ø(R5), X'40' This routine returns in R9
BE FINDSPC the number of characters in a
CLI Ø(R5), X'00' string, up to the first space
BE FINDSPC or low-value.
LA R5, 1(Ø, R5) The string is addressed by R5.
LA R9, 1(Ø, R9)
BCT R6, FINDSPC R6 is the string length.

FINDSPCF EQU *
BR R10 Return
*
CLEARLOW EQU *
CLI Ø(R5), X'00' This routine replaces
BNE CLEARLO2 low-values by spaces.
MVI Ø(R5), X'40' The string is addressed by R5
LA R5, 1(Ø, R5) R6 is the string length.
BCT R6, CLEARLOW
BR R10 Return
*
EXECDDYN EQU *
LA R1, DYNADDR Dynalloc subroutine
DYNALLOC Address parameters
Call SVC99
BR R10 Return
*
*===================================================================*
Using PF keys – a shortcut to running CLISTs

The use of PF keys allows you to achieve results directly with a single key (PFK) press. To avoid a series of repetitive commands, you can set up a PFK with the name of a TSO CLIST, which can be executed immediately.
KEYS

Keys are visible from all ISPF panels using the command KEYS.

Warning: many KEYLISTs exist in ISPF. The modifications that we will make will affect only that specified KEYLIST, eg:

ISR Keylist ISRSAB Change

It is not advisable to modify the structure of PFKs from 1 to 12; it is better to act on the alternative keys 13 – 24. On a PC keyboard, in order to use the alternative keys, it is necessary to press the shift key at the same time as the PF key.

For example, if we want to use PF19, simultaneously press the shift key and F7.

Suppose we want to modify key PF13 and associate it with the command SDSF LOG in order to see the syslog with a single keystroke:

1. On the screen from which we want to operate the command, enter the command:

   Command ===>    KEYS

The following screen will appear:

```
File PRIVATE ISR Keylist ISRSxxx Change
Command ===> Scroll ===> PAGE
Make changes and then select File action bar.
Keylist Help Panel Name . . . ISRSxxx

Key       Definition     Format  Label
F1 . . .  HELP           SHORT  Help
F2 . . .  SPLIT          LONG   Split
F3 . . .  EXIT           SHORT  Exit
F4 . . .                        
F5 . . .  RFIND          SHORT  Rfind
F6 . . .                        
F7 . . .  UP             LONG   Up
F8 . . .  DOWN           LONG   Down
F9 . . .  SWAP           LONG   Swap
```
I need to browse the SYSLOG quite often and control my jobs’ output too. Rather than continuously press Enter and PF2/PF3/PF9, until arriving at the SDSF menu, I want to set up two small commands, which will save me time.

2 Entering PF8, we go to the second set of keys (PF 13-24); we enter the command `tso %sflog` in place of the predefined F13 key, and `tso %sfst` on the F16 line, as in the following screen:

```
PRIVATE ISR Keylist ISRxxxx Change Row 13 to 24 of 24
Command ===> Scroll ===> PAGE
Make changes and then select File action bar.

Key       Definition                           Format  Label
F13  ;tso %sflog                             SHORT   Help
F14  SPLIT                                  LONG    Split
F15  END                                    SHORT   End
F16  ;tso %sfst                              SHORT   Return
F17  RFIND                                  SHORT   Rfind
F18  RCHANGE                                SHORT   Rchange
F19  UP                                     LONG    Up
F20  DOWN                                   LONG    Down
F21  SWAP                                   LONG    Swap
F22  LEFT                                   SHORT   Left
F23  RIGHT                                  SHORT   Right
F24  CRETRIEV                                SHORT   Cretriev
```

3 Exit from this screen by pressing PF3. On the top right of the screen will be displayed **Keylist saved.**

4 The **SFLOG** and **SFST** commands correspond to the two CLISTS, which are two members of a partitioned library concatenated to //SYSPROC DD in the logon procedure.

Clist SFLOG will contain the following line:
ISPEXEC SELECT PGM(ISFISP) PARM(LOG) NOCHECK NEWAPPL(ISF)

The result will be the display of the SYSLOG, as follows:

```
137 00000081 IST1051I EVENT CODE = 02
137 00000081 IST1062I EVENT ID = 0000
137 00000081 IST314I END
```

REQUEST F

*************** BOTTOM OF DATA ***************

Clist SFST will contain the following line:

ISPEXEC SELECT PGM(ISFISP) PARM(ST &SYSUID.*) NOCHECK NEWAPPL(ISF)

The result will be the display of my user-prefixed job status. For example, if my TSO SYSUID is L041105, I'll see:

```
NP   JOBNAME  JobID    Owner    Prty Queue     C  Pos  SAff  ASys Status
L0411052 J0802082 L041105      1 PRINT     B    53
L0411053 J0802088 L041105      1 PRINT     B    54
L041105P J0802089 DB2UT        1 PRINT     B    55
L041105C J0802090 L041105      1 PRINT     B    56
L041105C J0802103 L041105      1 PRINT     B    54
L0411056 J0802104 L041105      1 PRINT     B    65
L041105P J0802105 DB2UT        1 PRINT     B    66
```

5 To obtain the results above, press F13 (shift + F1) and F16 (shift + F4). This is now sufficient to enter SDSF panels.

So, think of the many possibilities there are in associating your personal command utilities with one key. Enjoy!

Alberto Mungai  
Senior Systems Programmer (Italy)  © Xephon 2003
Exploring IPCS exit services for customization

Have you ever used IPCS (Interactive Problem Control System) to assist in diagnosing a problem on OS/390 or an associated OS/390 subsystem and marvelled at what IPCS can provide? Maybe you’ve even grudgingly used IPCS, fumbled through its ISPF interface, and wrestled with its cryptic commands, eventually getting what you wanted, but never being sure why or how. Perhaps you’ve simply made use of IPCS as a tool without regard to what was happening in the background – you were just happy that it made you look like a hero again to your manager. Rest assured, if any of these describes you, you are not alone.

Out of the box, IPCS provides an extremely powerful diagnostic interface for OS/390 (any reference to OS/390 throughout the article implies z/OS as well). What’s particularly amazing is that virtually all of the components that make up the ‘out of the box’ IPCS offering are also available for use by the creative IPCS user. This starts right from the ability to create simple control block models, extends through to function-specific exits such as those that would be used for the ASCB exit or the CBSTAT exit, and culminates with a general purpose VERBEXIT exit that offers a very high degree of flexibility.

IPCS customization as it relates to exits comes in three basic flavours:

- Function-specific exits
- Stand-alone service routines
- Exit services router functions.

FUNCTION-SPECIFIC EXITS

IPCS provides a number of different function-specific exits that can be used to augment the standard IPCS offering. Function-specific exits fall into three broad categories:
• Exits that are invoked in a sequential manner based on the issuance of a specific IPCS subcommand. An example would be the ANALYZE exits that are executed when the **ANALYZE IPCS** subcommand is invoked.

• Subcommand-specific exits that are invoked through a subcommand request. Examples of exits that would fall into this category would be exits invoked through the ASCBEXIT or TCBEXIT subcommands.

• Exits that are invoked through the use of a subcommand operand. Examples of these exits include exits specified for FORMAT, CBSTAT, FIND, or SCAN subcommand operands.

The **BLSCECT** or **BLSCUSER** members from the SYS1.PARMLIB concatenation can contain definitions that cause some of the exits to be automatically invoked based on usage. *Writing IPCS Exit Routines*, chapter 8 of the **OS/390 MVS IPCS Customization** manual, provides a more detailed discussion on the creation and use of function-specific exits.

**STAND-ALONE SERVICE ROUTINES**

IPCS also provides a number of stand-alone service routines. These routines are typically invoked using the LINK macro or a LOAD/CALL sequence. Some of the services that fall into this category include:

• **BLSUSTOP** – this service provides the ability to quiesce an IPCS transaction.

• **BLSUXTID** – this service provides the ability to convert an 8-byte TOD clock value into a 26-character timestamp value (mm/dd/yyyy hh:mm:ss:fffff).

• **BLSUMTOD** – this service provides the ability to convert a 17-character timestamp value (mm/dd/yy hh:mm:ss) into its corresponding 8-byte TOD clock value.

• **BLSUXTOD** – this service provides the ability to convert a 26-character timestamp value (mm/dd/yyyy hh:mm:ss:ffffff) into its corresponding 8-byte TOD clock value.
Although these services would be used primarily from within IPCS exit applications, services like BLSUXTID, BLSUMTOD, and BLSUXTOD can be used externally as well. *IPCS Exit Services*, chapter 10 of the *OS/390 MVS IPCS Customization* manual, provides a detailed discussion on the use of these service routines.

**EXIT SERVICES ROUTER FUNCTIONS**

The IPCS exit services router provides a number of internal IPCS functions that really allow for exploiting the capabilities of IPCS. Some of the services provided by the exit services router include:

- Add symptom service
- Control block formatter service
- Control block status service
- Equate symbol service
- Expanded print service
- Format model processor service
- Get symbol service
- Storage access service
- Symbol service
- Table of contents service

Chapter 10, *IPCS Exit Services* in the *OS/390 MVS IPCS Customization* manual, provides a complete list of exit services router functions.

The anchor control block for using IPCS exit services invoked through the IPCS exit services router is the ABDPL (ABDUMP Parameter List). It is mapped by the BLSABDPL macro and, depending on which IPCS exit services you will be making use of, the BLSABDPL macro invocation can be used to expose various DSECTs as required. The address of the ABDPL is passed as a parameter to the VERBEXIT invoked program and
it can be referenced as necessary. IPCS exit services are invoked by passing the ABDPL address, an access service code, and an access service specific parameter list (as required) to the IPCS exit service router. The exit services router will perform the specified request and will return a return code to the caller, indicating the status of the request.

IPCS CUSTOMIZATION EXAMPLE

The rest of this article focuses on exit services router functions. A simple but practical IPCS VERBEXIT exit routine, SSCVTCHK, shows example usage for the following exit services router functions:

- Equate symbol service
- Expanded print service
- Format model processor service
- Storage access service
- Table of contents service.

Once these services have been demonstrated it shouldn’t be difficult to create custom exits for your own purposes to use within IPCS.

The SSCVTCHK exit example supplied with this article provides the ability to examine an OS/390 SSCVT (SubSystem Communication Vector Table) chain and it displays information about the subsystems that have been defined. In the absence of an exit parameter, the exit will list all defined SSCVTs and the status of the subsystem (whether it is inactive or active and, if it is active, the function codes that are supported, along with the addresses of the function routines). Expect to see output similar to that shown below when you issue the following IPCS subcommand from IPCS option 6.

Example VERBEXIT invocation:

```
--------------------- IPCS Subcommand Entry ----------------------------
Enter a free-form IPCS subcommand or a CLIST or REXX exec invocation
```
Sample SSCVTCHK output:

**SSCVT100I · SSCVT for SubSystem JES2**

**SSCVT: 00C53988**

- **ID.** SSCT SCTA... 00C53940 SNAM..... JES2
- **+00C FLG1.** A0 SS1D..... 02 RSV1..... 00
- **+0010** SSVT..... 00C35138 SUSE..... 00C35D18 SYN..... 00000000
- **+001C** SUS2..... 00C35688 RSV3..... 00000000

**SSCVT102I · Active SubSystem JES2 supports 028 function(s) with 028 routine(s)**

**SSCVT103I · Supported function codes and routine addresses follow:**

- **Func code:** 001 Rtn addr: 87632C60
- **Func code:** 002 Rtn addr: 87628E00
- **Func code:** 003 Rtn addr: 8762C0A0
- **Func code:** 004 Rtn addr: 87644470
- **Func code:** 005 Rtn addr: 87646AA0
- **Func code:** 006 Rtn addr: 87653580
- **Func code:** 007 Rtn addr: 87657780
- **Func code:** 008 Rtn addr: 87645178
- **Func code:** 009 Rtn addr: 876319D8
- **Func code:** 010 Rtn addr: 87630CF8
- **Func code:** 011 Rtn addr: 87633788
- **Func code:** 012 Rtn addr: 87649610
- **Func code:** 013 Rtn addr: 87649420
- **Func code:** 016 Rtn addr: 87650540
- **Func code:** 017 Rtn addr: 87651BE8
- **Func code:** 018 Rtn addr: 87652C08
- **Func code:** 019 Rtn addr: 87652430
- **Func code:** 020 Rtn addr: 87648DA8
- **Func code:** 021 Rtn addr: 876491F0
- **Func code:** 053 Rtn addr: 87635938
- **Func code:** 054 Rtn addr: 87636C10
- **Func code:** 064 Rtn addr: 87630480
- **Func code:** 070 Rtn addr: 87627F38
- **Func code:** 071 Rtn addr: 87635E70
- **Func code:** 075 Rtn addr: 87636080
- **Func code:** 077 Rtn addr: 87659A80
- **Func code:** 079 Rtn addr: 87638758
- **Func code:** 080 Rtn addr: 8762C528

**SSCVT100I · SSCVT for SubSystem MSTR**
SSCVT: 00C53940
    +0000 ID. ....... SSCT    SSTA. .... 00C53964 SNAM. .... MSTR
    +000C FLG1. .... 00 SSID. .... 00 RSV1. .... 00
    +0010 SSVT. .... 00C53970 SUSE. .... 00000000 SYN. .... 00C53A60
    +001C SUS2. .... 00000000 RSV3. .... 00000000

SSCVT102I - Active SubSystem MSTR supports 020 function(s) with 006 routine(s)
SSCVT103I - Supported function codes and routine addresses follow:
    Func code: 004 Rtn addr: 83ECC000
    Func code: 005 Rtn addr: 80E34000
    Func code: 006 Rtn addr: 83E0FB40
    Func code: 008 Rtn addr: 83ECC000
    Func code: 009 Rtn addr: 83ECC000
    Func code: 010 Rtn addr: 83ECC000
    Func code: 012 Rtn addr: 00CA1000
    Func code: 014 Rtn addr: 83ECC000
    Func code: 015 Rtn addr: 80CA7000
    Func code: 032 Rtn addr: 83ECC000
    Func code: 033 Rtn addr: 83ECC000
    Func code: 048 Rtn addr: 83ECC000
    Func code: 050 Rtn addr: 83ECC000
    Func code: 054 Rtn addr: 85B06EC0
    Func code: 063 Rtn addr: 83ECC000
    Func code: 068 Rtn addr: 83ECC000
    Func code: 072 Rtn addr: 83ECC000
    Func code: 073 Rtn addr: 83ECC000
    Func code: 078 Rtn addr: 83ECC000
    Func code: 080 Rtn addr: 83ECC000

SSCVT100I - SSCVT for SubSystem SMS
SSCVT: 00C53964
    +0000 ID. ....... SSCT    SSTA. .... 00C53964 SNAM. .... SMS
    +000C FLG1. .... 00 SSID. .... 00 RSV1. .... 00
    +0010 SSVT. .... 00C53028 SUSE. .... 00000000 SYN. .... 00000000
    +001C SUS2. .... 00000000 RSV3. .... 00000000

SSCVT102I - Active SubSystem SMS supports 005 function(s) with 003 routine(s)
SSCVT103I - Supported function codes and routine addresses follow:
    Func code: 008 Rtn addr: 84386D40
    Func code: 015 Rtn addr: 84386D40
    Func code: 016 Rtn addr: 844FCF28
    Func code: 017 Rtn addr: 844FB300
    Func code: 055 Rtn addr: 84386D40
The output presented above is an excerpted list. Expect several more subsystems to be displayed in your environment.

You can limit the display to a specific subsystem by passing a parameter to the SSCVTCHK exit. Examples include:

```
VERBX SSCVTCHK 'SUBSYS=JES2'
VERBX SSCVTCHK 'SUBSYS=X''E2D4E2'''
```

The latter example demonstrates that the subsystem name can be supplied in hex format for those subsystems that have a subsystem name that is not EBCDIC-readable. The paired single quotes preceding and following the subsystem name are required to indicate that quotes are part of the parameter value and that they do not represent delimiters for the parameter value itself.

Once you have used the SSCVTCHK exit, issue an IPCS LISTSYM subcommand. This subcommand will display the list of active symbols for the current default source data. You should see symbols listed for the subsystems that were located. The symbol names will start with SSCVT and end with the subsystem name. For example, a symbol name of SSCVTJES2 should exist if JES2 is a defined subsystem. If you restrict your display to a specific hex byte subsystem name, SSCVTCHK will create a symbol name that starts with SSCVTX and ends with the hex bytes for the subsystem name. For example, if you specify

```
VERBX SSCVTCHK 'SUBSYS=X''E2D4E2''
```

SSCVTCHK will create a symbol with a name of SSCVTXE2D4E2 if that subsystem has been defined.

**ACTIVATING THE SSCVTCHK VERBEXIT EXIT**

In order to make the SSCVTCHK VERBEXIT exit available to your IPCS session, linkedit SSCVTCHK into a load library that resides somewhere in the search order for your active session – the link list or STEPLIB are two options.

The source dump data for using SSCVTCHK should include
CSA, SQA, and NUC, as the data areas that are perused by the exit can reside in those areas of virtual storage. Set the default source to ACTIVE in your IPCS session if you want to look at the SSCVT chain on your running system.

CONCLUSION

The customization interfaces for IPCS can provide a very useful tool set for determining system status and diagnosing system and application problems. There may be a standard list of system control blocks that you examine for every dump you investigate regardless of the error or anomaly. Creating a customization exit that formats that information in a standard presentation could be helpful to your diagnostic effort. I hope this article has provided you with insight for your own IPCS customization exercises.

SSCVTCHK ASM

SSCVTCHK CSECT
SSCVTCHK AMODE 31
SSCVTCHK RMODE ANY

*---------------------------------------------------------------------*
*   SSCVTCHK is designed to be used as an IPCS VERBX exit routine     *
*   that can be used to display the information regarding the         *
*   MVS subsystems that were defined at the time the dump data        *
*   that is currently being processed was created.                    *
*                                                                    *
*   To that end, this VERBX exit routine will produce the best        *
*   results for dumps that contain CSA, SQA, and NUC SDATA. The       *
*   data for the SSCVT chain may terminate prematurely if these       *
*   data areas are not present in the dump dataset.                   *
*                                                                    *
*   SSCVTCHK can be used to dump all existing SSCVTs and the          *
*   corresponding supported function codes and function routine      *
*   addresses or a parm can be passed to SSCVTCHK to display this     *
*   same information for only a specified subsystem. For example,     *
*   some valid invocations would include:                            *
*                                                                    *
*   VERBX SSCVTCHK                                                    *
*   VERBX SSCVTCHK 'SUBSYS=SMS'                                      *
*   VERBX SSCVTCHK 'SUBSYS=X''D1C5E2F2''' (JES2 in hex format)        *
*                                                                    *
*   The first example would list all defined subsystems. The         *
*   displayed information would include a formatted SSCVT, an
indication whether or not the subsystem is active, and for active subsystems the supported function codes and their corresponding function routine addresses will be presented.

The second example would list information only about the subsystem named 'SMS'.

The third example would list information only about the subsystem named 'JES2'. This is useful because subsystem names do not need to contain readable hex characters.

As much as possible, the SSCVTCHK VERBX exit will accommodate for hex subsystem names. For example, if you request 'SUBSYS=X'D1C5E2F2''', an internal symbol of SSCVTXD1C5E2F2 will be created instead of symbol SSCVTJES2. This occurs only for subsystem specific requests. If the entire SSCVT chain is presented, all internal actions are attempted using EBCDIC readable characters only.

The following IPCS exit services are demonstrated in this program:

- Storage Access (IPCS service code ADPLSACC)
- Format Model Processor (IPCS service code ADPLSFMT)
- Expanded Print (IPCS service code ADPLSPR2)
- Equate Symbol (IPCS service code ADPLSEQS)
- Table of Contents (IPCS service code ADPLSNDX)

Chapter 10 in the OS/390 MVS IPCS Customization manual discusses the various IPCS exit services in detail. This exit example offers usage demonstration for only a handful of the available services.

In order to use the SSCVTCHK VERBX exit ensure that it is linkededited somewhere into the load module search order for your active IPCS session. Linkedit JCL similar to the following can be used:

```
//IEWL EXEC PGM=HEWLH96,PARM='XREF,LIST,MAP,RENT'
//SYSPRINT DD SYSOUT=* 
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(2,1))
//OBJECT DD DSN=object.code.pds,DISP=SHR
//SYSLMOD DD DSN=load.library,DISP=SHR
//SYSLIN DD *
INCLUDE OBJECT(SSCVTCHK) 
ENTRY SSCVTCHK
NAME SSCVTCHK(R) 
```

STM R14,R12,12(R13) Save incoming registers
LR R12,R15 Copy module address
LA R11,4095(,R12) Set up second ...
LA R11,1(,R11) base register
USING SSCVTCHK, R12, R11  Set module addressability
LR  R2, R1  Copy parameter address
LR  R3, R13  Copy savearea address
STORAGE OBTAIN, LENGTH=WORKLEN, LOC=ANY
LR  R0, R1  Copy working storage address
LR  R14, R1  Again
LR  R13, R1  Again
L  R1, =A(WORKLEN)  Get length
XR  R15, R15  Set fill byte
MVCL  R0, R14  Clear the storage
USING WORKAREA, R13  Set addressability
ST  R3, SAVEAREA+4  Save incoming savearea address
LA  R9, WORKPACC  Get ADPLPACC address
USING ADPLPACC, R9  Set addressability
LR  R8, R2  Get ABDPL address
USING ABDPL, R8  Set addressability
MVC  ASID(2), ADPLASID  Save the ASID
MVC  CVTADDR(4), ADPLCVT  Save the CVT address

*---------------------------------------------------------------------*
* The ADPLEXT contains the address of the extension pointer. If you  *
* want to process any input parameters passed to the VERBX program  *
* they can be captured at this point and processed.                   *
* +Ø from the ADPLEXT address contains the parameter address.         *
* +4 from the ADPLEXT address contains the CPPL address.              *
* See comments earlier for the format of valid parameters.           *
*---------------------------------------------------------------------*
MVC  SSNMPARM(4), =4C' '  Clear the area
MVC  SNAMHXSV(8), =C' 40404040'  Set default
L  R7, ADPLEXT  Get extension address
LTR  R7, R7  An extension?
BZ  NOPARM  No - unusual, but nothing to do
USING ADPLEXTN, R7  Set addressability
L  R15, ADPLOPTR  Get parm buffer address
LTR  R15, R15  A parameter?
BZ  NOPARM  No - nothing to do
S  R15, =F' 4'  Point to length
CLC  0(2, R15), =AL2(8)  Enough data?
BL  BADPARM  No - issue message
OC  4(6, R15), =6C' '  Set parm keyword to uppercase
CLC  4(7, R15), =C' SUBSYS='  Proper keyword?
BNE  BADPARM  No - issue message

*---------------------------------------------------------------------*
* A SUBSYS= parm has been supplied. Check to see whether the         *
* subsystem name has been supplied in character or hex format. If    *
* the first two characters following the SUBSYS= are X' this         *
* indicates a hex format subsystem name.                             *
*---------------------------------------------------------------------*
CLC  11(2, R15), =C' X''  Hex indicator?
BE  HEXSS  Yes - process as hex characters
MVC SSNMPARM(4),11(R15) Copy subsystem name
B NOPARM Go on

HEXSS DS ØH

* If we get here, the subsystem name is to be interpreted as a *
* hex value. Make sure the supplied characters are valid hex *
* characters and make sure the resulting length is correct. *

XR R14,R14 Clear R14
ICM R14,B'Ø011',Ø(R15) Get max buffer length
S R14,=F'9' Subtract length of SUBSYS=X'
LA R3,13(,R15) Point to first subsys name byte
XR R4,R4 Clear a length counter register
LTR R14,R14 Some data?
BZ BADPARM No - that's a problem

SSHXCHLP DS ØH

CLI Ø(R3),C' ''' End indicator?
BE SSHXDN Yes - go check the counters
CLI Ø(R3),C'A' A valid hex character?
BL BADPARM No - that's a problem
CLI Ø(R3),C'F' A valid hex character?
BNH SSHXOK Yes - check next one
CLI Ø(R3),C'Ø' A valid hex character?
BL BADPARM No - that's a problem
CLI Ø(R3),C'9' A valid hex character?
BH BADPARM No - that's a problem

SSHXOK DS ØH

LA R3,1(,R3) Point to next byte
LA R4,1(,R4) Add one to count
B SSHXCHLP Check next character

SSHXDN DS ØH

C R4,=F'2' Valid byte count?
BE SSHXPRM Yes - we've got a valid parm
C R4,=F'4' Valid byte count?
BE SSHXPRM Yes - we've got a valid parm
C R4,=F'6' Valid byte count?
BE SSHXPRM Yes - we've got a valid parm
C R4,=F'8' Valid byte count?
BNE BADPARM No - that's a problem

SSHXPRM DS ØH

MVC DBL1(8),=8C Space Clear the target area
LR R5,R4 Copy the length
BCTR R5,Ø Reduce by one for EX
EX R5,SSHXMVC Copy the subsystem name
TR DBL1(8),TRTABLE Scrub unwanted info
LR R5,R4 Copy the length
SRL R5,1 Divide by two
LR R6,R5 Copy length for later use
SLL R5,4 Shift length to high order nibble
STC R5, DBL2+8 Save length
STC R4, DBL2+9 Save length

OC  DBL2+8(1), DBL2+9 OR the two length values
IC  R5, DBL2+8 Get lengths for EX
EX  R5, SSHXPACK PACK to valid hex data
BCTR R6, Ø Reduce length by one for EX
EX  R6, SSNMMVC Copy the subsystem name
ST  R4, SNAMHXLN Save the length
BCTR R4, Ø Reduce length by one for EX
EX  R4, SSNMMVC2 Copy the subsystem name
B NOPARM We should be done

*--------------------------------------------------------------------*
BADPARM  DS  ØH
LA  RØ, PRMMSG1L Get message length
LA  R1, ParmMsg1 Get message address
BAL R14, PRINTLN Go print the line
LA  RØ, 1 Set message length
LA  R1, =C' ' Get message address
BAL R14, PRINTLN Go print a blank line
DROP R7

NOPARM  DS  ØH

*---------------------------------------------------------------------*
*   Obtain the CVT.                                                   *
*---------------------------------------------------------------------*
MVC ADPLPAAD(4), CVTADDR Set address to the CVT
MVC ADPLDLEN(2), =AL2(CVTOSLVF+1-CVT) Set get length
OI ADPLPRDP, ADPLVIRT+ADPLSAMK Indicate virtual 24-bit addr
L R15, ADPLSERV Get service routine address
CALL (15), X
((R8), X
CODEACC, X
(R9)), MF=(E, CALLLST)
LTR R15, R15 Were things ok?
BNZ NOSTORE1 No - issue storage not found msg

*---------------------------------------------------------------------*
*   Obtain the JESCT.                                                 *
*---------------------------------------------------------------------*
L R1, ADPLPART Get buffer location address
USING CVT, R1
MVC ADPLPAAD(4), CVTJESCT Get JESCT address
MVC ADPLDLEN(2), =AL2(128) Set get length
DROP R1
L R15, ADPLSERV Get service routine address
CALL (15), X
((R8), X
CODEACC, X
(R9)), MF=(E, CALLLST)
LTR R15, R15 Were things ok?
BNZ NOSTORE2 No - issue storage not found msg

*---------------------------------------------------------------------*
*   Obtain the first SSCT.                                            *
*---------------------------------------------------------------------*
L R1, ADPLPART Get buffer location address
USING JESCT, R1

MVC ADPLPAAD(4), JESSSCT Get SSCT address

GETSSCVT DS ØH

MVC ADPLDLLEN(2), =AL2(32) Set get length

NI ADPLPRDP, 255-ADPLSAMK Indicate virtual 31-bit addr

DROP R1

L R15, ADPLSERV Get service routine address

CALL ((15), (R8), CODEACC, (R9)), MF = (E, CALLLST)

LTR R15, R15 Were things ok?

BNZ NOSTORE3 No - issue storage not found msg

*--------------------------------------------------------------------*

L R1, ADPLPART Get buffer location address

USING SSCT, R1

MVC SNAMSAVE(8), =8C ' ' Clear the area

MVC SNAMSAVE(4), SSCTSNAME Save the subsystem name

MVC SSCTNEXT(4), SSCTSCTA Save the address of the

CLC SSNMPARM(4), =4C ' ' Display all SSCVTs?

BE NOTJ USTL Yes - format this SSCVT

CLC SSNMPARM(4), SNAMSAVE A match?

BNE NEXTSSCT No - bypass

NOTJ USTL DS ØH

MVC LINEBUF( L'MSG1), MSG1 Copy the message

MVC LINEBUF+32(4), SNAMSAVE Copy the subsystem name

LA R0, L'MSG1 Get message length

CLC SNAMHXS(8), =C'40404040' Hex name specified?

BE NOHEX1 No - bypass hex specific stuff

MVC LINEBUF+32(2), =C'x''' Move in prefix

MVC LINEBUF+34(8), SNAMHXS Copy the name

L R14, SNAMHXLN Get length of name

LA R14, LINEBUF+34(R14) Point to end of name

MVI 0(R14), C' ' ' Set end quote

LA R0, 1 (R14) Set ending address

LA R14, LINEBUF get starting address

SR R0, R14 Set message length

NOHEX1 DS ØH

LA R1, LINEBUF Get message address

BAL R14, PRINTLN Go print the line

LA R0, 1 Set message length

LA R1, =C' ' ' Get message address

BAL R14, PRINTLN Go print a blank line

MVC LINEBUF( L'TOCCSSCVT), TOCCSSCVT Copy the TOC message

MVC LINELEN(4), =A( L'TOCCSSCVT) Get the TOC message length

MVC LINEBUF+30(4), SNAMSAVE Copy the subsystem name

CLC SNAMHXS(8), =C'40404040' Hex name specified?

BE NOHEX2 No - bypass hex specific stuff

MVC LINEBUF+30(2), =C'x''' Move in prefix

MVC LINEBUF+32(8), SNAMHXS Copy the name

L R14, SNAMHXLN Get length of name
LA R14,LI NEBUF+32(R14) Point to end of name
MVI Ø(R14),C'''' Set end quote
LA R0,1(R14) Set ending address
LA R14,LI NEBUF get starting address
SR R0,R14 Set message length
ST R0,LIN ELEN Save the length

NOHEX2 DS ØH
BAL R14,TOCENTRY Add a TOC entry

* Format and print the SSCVT.

*---------------------------------------------------------------------* 
* Format and print the SSCVT.                                         *
*---------------------------------------------------------------------*
LA R7,WORKPFMT Get ADPLPFMT address
USING ADPLPFMT,R7 Set addressability
MVC ADPLPPTR(4),=A(SSCTMODL) Get control block model addr
MVI ADPLPVCI,X'03' Set viewing control to x'03'
MVC ADPLDLEN(2),=AL2(SSCTSIZE) Get control block length
MVC ADPLPBLS(2),=AL2(SSCTSIZE) Get control block length
MVC ADPLPBAV(4),ADPLPAAD Dumped address to access
MVC ADPLPCHA(8),=8C' ' Clear model name
OI ADPLPOPT,ADPLPOAC Set acronym check flag
MVC ADPLPBAS(4),ADPLPART Address of buffer
        L R15,ADPLSERV Get service routine address
        CALL (15),                                                   X
        ((R8),                                                  X
        CODEFMT,                                                X
        (R7)),MF=(E,CALLLST)                                      X
        LTR R15,R15 Were things ok?
        BNZ NOFRMAT1 No - just leave for now
        DROP R7

*---------------------------------------------------------------------*
* Create an IPCS symbol for this SSCVT. The symbol name will         *
* have the format of SSCVTssname where 'ssname' is the name of       *
* the subsystem providing the request is for all subsystems or       *
* a specific readable EBCDIC subsystem name. If the request was     *
* for a specific hex subsystem name, the symbol name will have the   *
* format of SSCVTXssnamehex where 'ssnamehex' is character          *
* representation of the hex value bytes.                            *
*---------------------------------------------------------------------*
NI FLAG1,255-SYMTRY Reset the SYMTRY flag
SYM1 DS ØH
LA R7,WORKESSY Get ESSY area address
MVC ESSYSYM-ESSY(32,R7),=CL32'SSCVT' Create required ...
MVC ESSYSYM-ESSY+5(4,R7),SNAMSAVE symbol name
CLC SNAMHXS(8),=C'40404040' Hex name specified?
BE NOHEX3 No - bypass hex specific stuff
MVC ESSYSYM-ESSY+5(4,R7),=C'X ' Set up for hex symbol
        L R14,SNAMHXLN Get length of name
        BCTR R14,Ø Reduce by one for EX
        EX R14,SSNMVC3 Move in the subsystem name

NOHEX3 DS ØH
MVC ESSYAST-ESSY(2,R7),=C'CV' Move in address space type
MVC ESSYALAD-ESSY(4,R7),ADPLPAAD Move in SSCVT address
MVC ESSYALDI-ESSY(4,R7),=A(SSCTSIZE) Move in SSCVT length
* MVC ESSYDTY-ESSY(1,R7),=C'M' Indicate type as STRUCTURE
MVC ESSYDTY-ESSY(1,R7),=C'U' Indicate type as AREA
MVC ESSYDTD-ESSY(32,R7),ESSYSYM-ESSY(R7) Move in data name
MVC ESSYRL-ESSY(2,R7),=AL2(31) Move in remark length
MVC ESSYRT-ESSY(31,R7),=C'SSCVT for subsystem xxxx
MVC ESSYRT-ESSY+2D(4,R7),SNAMSAVE Remark
CLC SNAMHXS(8),=C'40404040' Hex name specified?
BE NOHEX4 No - bypass hex specific stuff
L R14,SNAMHXLN Get length of name
BCTR R14,Ø Reduce by one for EX
EX R14,SSNMMVC4 Move in the subsystem name
LA R14,1+2+ESSYRT-ESSY+2D(R14,R7) Point past subsys name
MVI Ø(R14),C'''' Put in ending quote
NOHEX4 DS ØH

* SSCVT is in common storage so set ASID=1

MVC ESSyas2-ESSY+2D(2,R7),=AL2(1) Move in the ASID
OI ESSYFC-ESSY(R7),ESSYFCD Set NODROP attribute on symbol
L R15,ADPLSERV Load addr of exit services router
CALL (15), X (((R8), X CODEEqs,
(7)),MF=(E,CALLLST)
C R15,=F'12' Symbol equate ok?
BL SYMI Yes - go on
TM FLAGI,SYMTRY Is this the second try?
BO SYMI Yes - issue message
OI FLAGI,SYMTRY Set flag
B SYMI Try a second time
SYMl DS ØH

* Check to see whether this is an active subsystem entry and if it is, locate the SSVT.

L R1,ADPLPART Get buffer location address
CLC SSCTSSVT(4),=F'Ø' An active subsystem?
BE INACTIVE No - get the next SSCVT
MVC ADPLPAAD(4),SSCTSSVT Get SSVT address
MVC ADPLDLEN(2),=AL2(264) Set get length
MVC SSVTADDR(4),SSCTSSVT Save the SSVT address
L R15,ADPLSERV Get service routine address
CALL (15), X (((R8), X CODEACC,
(9)),MF=(E,CALLLST)
LTR R15,R15 Were things ok?
BNZ NOSTORE4 No - issue storage not found msg

DROP R1

*----------------------------------------------------------------------*
L R1,ADPLPART           Get buffer location address
USING SSVT,R1
MVC LINEBUF(MSG3L),MSG3 Copy the message
MVC LINEBUF+29(4),SNAMSAVE Copy the subsystem name
XR R15,R15               Clear R15
ICM R15,'ØØ11',SSVTFNUM Get number of function routines
ST R15,FNUMSAVE          Save number of function routines
CVD R15,DBL2              Convert to decimal
UNPK DBL1(8),DBL2(8) Unpack it
OI DBL1+7,X'FØ' Clear sign
MVC LINEBUF+64(3),DBL1+5 Copy number of function routines
XR R15,R15               Clear counter
LA R14,SSVTFCOD          Get function code matrix address
LA R3,256 Set loop count
FCODLP1 DS ØH
CLI Ø(R14),X'ØØ' Active function code?
BE FCODNXT1 No - go check next one
LA R15,1(,R15) Add one to counter

FCODNXT1 DS ØH
LA R14,1(,R14) Point to next indicator
BCT R3,FCODLP1 If more, go check it out
CVD R15,DBL2 Convert to decimal
UNPK DBL1(8),DBL2(8) Unpack it
OI DBL1+7,X'FØ' Clear sign
MVC LINEBUF+43(3),DBL1+5 Copy number of functions
LA R0,MSG3L Get message length
CLC SNAMHXSV(8),=C'40404040' Hex name specified?
BE NOHEX5 No - bypass hex specific stuff
MVC DBL1(3),LINEBUF+43 Save number of functions
MVC DBL2(3),LINEBUF+64 Save number of function routines
MVC LINEBUF+25(3),=C' x''' Move in prefix
MVC LINEBUF+28(8),SNAMHXSV Copy the name
L R14,SNAMHXLN Get length of name
LA R14,LINEBUF+28(R14) Point to end of name
MVC Ø(2,R14),=C''' Set end quote
LA R14,2(,R14) Point past end quote
MVC Ø(9,R14),=C'supports ' Set next part of message
LA R14,9(,R14) Point past it
MVC Ø(3,R14),DBL1 Copy number of function codes
MVI 3(R14),C' ' Move in separator
LA R14,4(,R14) Point past it
MVC Ø(17,R14),=C'function(s) with ' Set next part of message
LA R14,17(,R14) Point past it
MVC Ø(3,R14),DBL2 Copy number of function routines
LA R14,3(,R14) Point past it
MVC Ø(7,R14),=C'rtn(s)' Set next part of message
LA R14,7(,R14) Point past it
LR R0,R14 Copy end of message address
LA R14,LINEBUF Get starting address

© 2003. Xerphon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.
SR R0, R14                    Set message length

NOHEX5 DS ØH

LA R1, LINEBUF                Get message address
BAL R14, PRINTLN              Go print the line
LA R0, MSG4L                 Get message length
LA R1, MSG4                 Get message address
BAL R14, PRINTLN              Go print the line

*--------------------------------------------------------------------*
MVC ADPLPAAD(4), SSVTADDR     Get SSVT address
L R15, FNUMSAVE               Get number of function routines
SLL R15, 2                    Multiply by 4
LA R15, 264(R15)             Set proper size
STCM R15, B'0011', ADPLDLEN  Save the length
L R15, ADPLSERV              Get service routine address
CALL (15),                    X
((R8),                    X
CODEACC,                   X
(R9)), MF=(E, CALLLST)
LTR R15, R15                  Were things ok?
BNZ NOSTORE4                  No - issue storage not found msg
L R1, ADPLPART                Get buffer location address
LA R3, 256                    Set loop count
LA R4, SSVTFCOD              Get addr of function code matrix
LA R5, SSVTFRTN              Get addr of function rtn addr
XR R6, R6                    Clear counter
FCODLP2 DS ØH

CLI Ø(R4), X'00'             Active function code?
BE FCODNXT2                  No - go check next one
MVC LINEBUF(L'MSG5), MSG5    Copy the message
LA R15, 1(R6)               Get real function code #
CVD R15, DBL2                Convert function code # to decimal
UNPK DBL1(8), DBL2(8)        Unpack it
OI DBL1+7, X'F0'             Clear sign
MVC LINEBUF+12(3), DBL1+5    Copy function code number
XR R14, R14                  Clear R14
LC R14, Ø(R4)               Get function rtn index value
BCTR R14, Ø                   Reduce by one
SLL R14, 2                   Multiply by 4
L R15, Ø(R14, R5)            Get function rtn addr
BAL R14, HEXCNVT             Make it readable
MVC LINEBUF+27(8), DBL1      Copy to output line
LA R0, L'MSG5                Get message length
LA R1, LINEBUF              Get message address
BAL R14, PRINTLN             Go print the line
FCODNXT2 DS ØH

LA R4, 1(R4)                Point to next indicator
LA R6, 1(R6)                Add one to function code counter
BCT R3, FCODLP2             If more, go check it out

*--------------------------------------------------------------------*
LA R0, 1                    Set message length
LA R1, =C' '                Get message address
BAL R14, PRINTLN
LA R0, 1
LA R1, =C' ' 
BAL R14, PRINTLN
DROP R1
CLC SSNMPARM(4), SNAMSAVE
BNE NEXTSSCT
MVC LINEBUF(L'MSG7), MSG7
MVC LINEBUF+49(4), SSNMPARM
LA R0, L'MSG7
CLC SNAMHXSV(8), =C' 40404040' 
BE NOHEX6
MVC LINEBUF+49(2), =C' x'''
MVC LINEBUF+51(8), SNAMHXS
LA R14, SNAMHXLN
LA R14, LINEBUF+51(R14)
MVI Ø(R14), C''''
LA R0, LINEBUF
SR R0, R14
LA R1, LINEBUF
BAL R14, PRINTLN
B RETURN
*--------------------------------------------------------------------*
NEXTSSCT DS ØH
CLC SSCTNEXT(4), =F'Ø'
BE SSCTDONE
MVC ADPLPAAD(4), SSCTNEXT
B GETSSCVT
*--------------------------------------------------------------------*
INACTIVE DS ØH
MVC LINEBUF(L'MSG2), MSG2
MVC LINEBUF+22(4), SNAMSAVE
LA R0, L'MSG2
CLC SNAMHXSV(8), =C' 40404040' 
BE NOHEX7
MVC LINEBUF+22(2), =C' x'''
MVC LINEBUF+24(8), SNAMHXS
LA R14, SNAMHXLN
LA R14, LINEBUF+24(R14)
MVC Ø(13, R14), =C' '' is inactive' 
LA R0, 13(R14)
LA R14, LINEBUF
SR R0, R14
NOHEX7 DS ØH
LA R1, LINEBUF
BAL R14, PRINTLN
LA R0, 1
LA R1, =C' ' 
BAL R14, PRINTLN
© 2003. Xerphon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.
LA R0,1                   Set message length
LA R1,=C' '                Get message address
BAL R14,PRINTLN           Go print a blank line
CLC SSNMPARM(4),SNAMSAVE  A subsystem name match?
BNE NEXTSSCT              No - do next subsystem
B RETURN                  We're done

*--------------------------------------------------------------------*
SSCTDONE DS ØH
CLC SSNMPARM(4),=4C' '    Specific subsystem requested?
BE SSCTDON2               No - we're done
CLC SSNMPARM(4),SNAMSAVE  A subsystem name match?
BE SSCTDON2               Yes - we're done
MVC LINEBUF(L'NOSSMSG1),NOSSMSG1 Copy message model
MVC LINEBUF+22(4),SSNMPARM Copy subsystem name into message
LA R0,L'NOSSMSG1          Get message length
CLC SNAMHXSV(8),=C'40404040' Hex name specified?
BE NOHEX8                 No - bypass hex specific stuff
MVC LINEBUF+22(2),=C'x''' Move in prefix
MVC LINEBUF+24(8),SNAMHXSV Copy the name
L R14,SNAMHXLN            Get length of name
LA R14,LI NEBUF+24(R14)   Point to end of name
MVC Ø(13,R14),=C' ' ' not located' Set end of message
LA R0,13,(R14)            Set ending address
LA R14,LI NEBUF           get starting address
SR R0,R14                 Set message length
NOHEX8 DS ØH              *
LA R1,LI NEBUF            Get message address
BAL R14,PRINTLN           Go print the line
LA R0,1                   Set message length
LA R1,=C' '               Get message address
BAL R14,PRINTLN           Go print a blank line
B RETURN                  We're done

SSCTDON2 DS ØH
LA R0,L'MSG6              Get message length
LA R1,MSG6                Get message address
BAL R14,PRINTLN           Go print the line
B RETURN                  We're done

*---------------------------------------------------------------------*
RETURN DS ØH
L R3,SAVEAREA+4           Load incoming savearea address
LR R1,R13                 Get working storage address
STORAGE RELEASE,LENGTH=WORKLEN,ADDR=(R1)
LR R13,R3                 Restore incoming savearea address
LM R14,R12,12(R13)        Restore incoming registers
XR R15,R15                Set return code
BR R14                    Return

NOSTORE1 DS ØH
MVI LI NEBUF,C' '         Set fill byte
MVC LI NEBUF+1(131),LI NEBUF Clear the area
MVC LI NEBUF(STMSG1L),STOMSG1 Copy the message
BAL R14,HEXCNVT           Make the rc readable
MVC LINEBUF+50(2), DBL1+6 Copy rc into message
ICM R15, B'1111', ADPLPAAD Get CVT address
BAL R14, HEXCNVT Make it readable
MVC LINEBUF+36(8), DBL1 Copy CVT address into message
LA R0, STMSG1L Get message length
LA R1, LINEBUF Get message address
BAL R14, PRINTLN Go print the line
LA R0, 1 Set message length
LA R1, =C' ' Get message address
BAL R14, PRINTLN Go print a blank line
LA R0, TRMMSG1L Get message length
LA R1, TERMMMSG1 Get message address
BAL R14, PRINTLN Go print the line
B RETURN We're done

NOSTORE2 DS ØH
MVI LINEBUF, C' ' Set fill byte
MVC LINEBUF+1(131), LINEBUF Clear the area
MVC LINEBUF(STMSG2L), STORMSG2 Copy the message
BAL R14, HEXCNVT Make the rc readable
MVC LINEBUF+52(2), DBL1+6 Copy rc into message
ICM R15, B'1111', ADPLPAAD Get JESCT address
BAL R14, HEXCNVT Make it readable
MVC LINEBUF+38(8), DBL1 Copy JESCT address into message
LA R0, STMSG2L Get message length
LA R1, LINEBUF Get message address
BAL R14, PRINTLN Go print the line
LA R0, 1 Set message length
LA R1, =C' ' Get message address
BAL R14, PRINTLN Go print a blank line
LA R0, TRMMSG1L Get message length
LA R1, TERMMMSG1 Get message address
BAL R14, PRINTLN Go print the line
B RETURN We're done

NOSTORE3 DS ØH
MVI LINEBUF, C' ' Set fill byte
MVC LINEBUF+1(131), LINEBUF Clear the area
MVC LINEBUF(STMSG3L), STORMSG3 Copy the message
BAL R14, HEXCNVT Make the rc readable
MVC LINEBUF+52(2), DBL1+6 Copy rc into message
ICM R15, B'1111', ADPLPAAD Get SSCVT address
BAL R14, HEXCNVT Make it readable
MVC LINEBUF+38(8), DBL1 Copy SSCVT address into message
LA R0, STMSG3L Get message length
LA R1, LINEBUF Get message address
BAL R14, PRINTLN Go print the line
LA R0, 1 Set message length
LA R1, =C' ' Get message address
BAL R14, PRINTLN Go print a blank line
LA R0, TRMMSG1L Get message length
LA R1, TERMMMSG1 Get message address
BAL R14, PRINTLN Go print the line
B RETURN We're done

NOSTORE4 DS 0H
MVI LINEBUF,C' ' Set fill byte
MVC LINEBUF+1(131),LINEBUF Clear the area
MVC LINEBUF(STMSG4L),STORMSG4 Copy the message
BAL R14,HEXCNVT Make the rc readable
MVC LINEBUF+51(2),DBL1+6 Copy rc into message
ICM R15,B'1111',ADPLPAAD Get SSVT address
BAL R14,HEXCNVT Make it readable
MVC LINEBUF+37(8),DBL1 Copy SSVT address into message
LA R0,STMSG4L Get message length
LA R1,LINEBUF Get message address
BAL R14,PRINTLN Go print the line
LA R0,1 Set message length
LA R1,=C' ' Get message address
BAL R14,PRINTLN Go print a blank line
LA R0,TRMMSG1L Get message length
LA R1,TEMLMSG1 Get message address
BAL R14,PRINTLN Go print the line
B RETURN We're done

NOFRMAT1 DS 0H
MVI LINEBUF,C' ' Set fill byte
MVC LINEBUF+1(131),LINEBUF Clear the area
MVC LINEBUF(FMTMSG1L),FRMTMSG1 Copy the message
BAL R14,HEXCNVT Make the rc readable
MVC LINEBUF+49(2),DBL1+6 Copy rc into message
LA R0,FRMTMSG1L Get message length
C R15,=F'4' Additional error information?
BNE NOPRET1 No - bypass
XR R15,R15 Clear R15
ICM R15,B'0011',ADPLPRET-ADPLPFMT(R7) Get error flag info
BAL R14,HEXCNVT Make the flags readable
MVC LINEBUF+FMTMSG1L(15),=C' ADPLPRET(xxxx)' Copy error flag info
MVC LINEBUF+FMTMSG1L+10(4),DBL1+4 Get message length
LA R0,FMTMSG1L+15

NOPRET1 DS 0H
LA R1,LINEBUF Get message address
BAL R14,PRINTLN Go print the line
LA R0,1 Set message length
LA R1,=C' ' Get message address
BAL R14,PRINTLN Go print a blank line
LA R0,TRMMSG1L Get message length
LA R1,TEMLMSG1 Get message address
BAL R14,PRINTLN Go print the line
B RETURN We're done

NOSYM1 DS 0H
MVI LINEBUF,C' ' Set fill byte
MVC LINEBUF+1(131),LINEBUF Clear the area
MVC LINEBUF(SYDMSG1L),SYMDMSG1 Copy the message
BAL R14,HEXCNVT Make the rc readable
MVC LINEBUF+48(4),SNAMSAVE Copy subsystem name into message
MVC  LINEBUF+58(2), DBL1+6    Copy rc into message
LA  R0, SYDMSG1L              Get message length
LA  R1, LINEBUF               Get message address
BAL R14, PRINTLN              Go print the line
LA  R0, 1                     Set message length
LA  R1, =C' '                 Get message address
BAL R14, PRINTLN              Go print a blank line
B  SYM1E                      Go back for more

*---------------------------------------------------------------------*
*   Subroutines                                                      *
*---------------------------------------------------------------------*
PRINTLN  DS  ØH

*---------------------------------------------------------------------*
*   The PRINTLN subroutine generates a line of output using the       *
*   IPCS print service.                                              *
*   On entry:  R0 - contains the length of the output line           *
*              R1 - contains the address of the output line           *
*              R8 - contains the address of the ABDPL                 *
*   On exit:  R15 - contains the return code from the IPCS print      *
*              service                                               *
*---------------------------------------------------------------------*
STM  R0, R15, REGSAVE         Save the registers
LA  R7, WORKPPR2              Get BLSUPPR2 address
MVC  Ø(PPR2999-PPR2000, R7), PPR2 Copy the PPR2 model
MVC  PPR2BUF-PPR2(4, R7), ADPLBUF Copy print buffer address
ST  R0, PPR2BUF-PPR2(4, R7)   Save the message length
L   R3, PPR2BUF-PPR2(4, R7)   Copy the message length
L   R15, ADPLBUF              Get message buffer address
MVI  Ø(R15), =C' '            Set fill byte
MVC  1(131, R15), Ø(R15)      Clear message buffer area
L   R15, ADPLBUF              Get message buffer address
BCTR R3, Ø                    Reduce length by one for EX
EX   R3, MSGMVC               Copy the message
MVI  PPR2PFL1-PPR2(R7), PPR2MSG Indicate buffer contains a msg
L   R15, ADPLSERV             Get service routine address
CALL (15),                    X
   ((R8),                     X
    CODEPPR2,                 X
    (R7)), MF=(E, CALLLST)

PRINTLNE DS  ØH
LM  R0, R14, REGSAVE          Restore required registers
BR  R14                        Return

*---------------------------------------------------------------------*
HEXCNVT  DS  ØH

*---------------------------------------------------------------------*
*   The HEXCNVT subroutine converts the hex contents of R15 to        *
*   a human readable format in variable DBL1.                         *
*---------------------------------------------------------------------*
ST  R15, DBL2                 Save the value
UNPK DBL1(9), DBL2(5)         Unpack it
NC  DBL1(8), =8X'0F'          Turn off high nibble
TR  DBL1(8),=C'Ø123456789ABCDEF'  Make it readable
BR  R14  Return

* The TOCENTRY subroutine adds an entry to the IPCS table of contents.
* On entry, LINELEN contains the length of the TOC message
* (greater than Ø, less than 41). LINEBUF contains the value of the TOC message

STM   RØ,R15,REGSAVE  Save the registers
L     R15,ADPLBUF     Get message buffer address
L     R3,LINELEN      Get TOC message length
LA    R3,4(,R3)      Add in length of length word
BCTR  R3,Ø           Reduce by one for EX
EX    R3,TOCMVC       Copy the TOC message
L     R15,ADPLSERV    Get service routine address
CALL  (15),                                                   X
       ((R8),                                                  X
       CODENDX),                                               X
       MF=(E,CALLLST)
LM    RØ,R14,REGSAVE  Restore required registers
BR    R14             Return

* Executed instructions

MSGMVC MVC Ø(*-*),R15),Ø(R1)  Copy the message
TOCMVC MVC Ø(*-*),R15),LINELEN Copy the TOC message
SSHXMVC MVC DBL1(*-*),13(R15) Copy the hex subsystem name
SSHXPACK PACK DBL2(*-*),DBL1(*-*) Pack the hex characters
SSNMMVC2 MVC SNMPARM(*-*),DBL2 Copy the subsys name (hex format)
SSNMMVC3 MVC SNAMHXSV(*-*),13(R15) Copy the subsys name (hex format)
SSNMMVC4 MVC ESSYRT-ESSY+22(*-*),R7) Copy subsys name to rem

* Constants

CODEACC DC A(ADPLSACC)
CODEFMT DC A(ADPLSFMT)
CODEPR2 DC A(ADPLSPR2)
CODEEQS DC A(ADPLSEQS)
CODENDX DC A(ADPLSNDX)

ESSY BLSRESSY DSECT=NO  IPCS ES record buffer
PPR2 BLSUPPR2 DSECT=NO  IPCS expanded print parm list

MSG1 DC C'SSCVT1ØØI - SSCVT for SubSystem xxxx'
MSG2 DC C'SSCVT1Ø1I - SubSystem xxxx is inactive'

MSG3 DC C'SSCVT102I - Active SubSystem xxxx supports xxx ' 
   DC C'function(s) with xxx routine(s)' 
MSG3L EQU *-MSG3 
MSG4 DC C'SSCVT103I - Supported function codes and routine ' 
   DC C'addresses follow:' 
MSG4L EQU *-MSG4 
MSG5 DC C' Func code: xxx  Rtn addr: xxxxxxxx' 
MSG6 DC C'SSCVT199I - SSCVT chain complete' 
MSG7 DC C'SSCVT198I - SSCVT display complete for Subsystem xxxx' 
PARMMSG1 DC C'SSCVT110I - Invalid parm detected. Entire SSCVT ' 
   DC C'chain will be presented.' 
PRMMSG1L EQU *-PARMMSG1 
STORMSG1 DC C'SSCVT111I - Unable to locate CVT at XXXXXXXX - RC(xx)' 
STORMSG1L EQU *-STORMSG1 
STORMSG2 DC C'SSCVT112I - Unable to locate JESCT at xxxxxxxx ' 
   DC C'RC(xx)' 
STORMSG2L EQU *-STORMSG2 
STORMSG3 DC C'SSCVT113I - Unable to locate SSCVT at xxxxxxxx - ' 
   DC C'RC(xx)' 
STORMSG3L EQU *-STORMSG3 
STORMSG4 DC C'SSCVT114I - Unable to locate SSVT at xxxxxxxx - ' 
   DC C'RC(xx)' 
STORMSG4L EQU *-STORMSG4 
TERMMSG1 DC C'SSCVT189I - Format has terminated prematurely' 
TERMMSG1L EQU *-TERMMSG1 
FRMTMSG1 DC C'SSCVT121I - Error detected formatting SSCVT - ' 
   DC C'RC(xx)' 
FRMTMSG1L EQU *-FRMTMSG1 
SYMDMSG1 DC C'SSCVT131I - Error detected defining symbol SSCVTxxxx ' 
   DC C'RC(xx)' 
SYMDMSG1L EQU *-SYMDMSG1 
NOSSMSG1 DC C'SSCVT141I - Subsystem xxxx not located' 
TOCSSCVT DC C'Formatted SSCVT for SubSystem xxxx' 
*---------------------------------------------------------------------* 
TRTABLE DC 256X'80' 
ORG TRTABLE+0 
DC C'0123456789ABCDEF' 
ORG TRTABLE+193 
DC '0A0B0C0D0E0F' 
ORG TRTABLE+240 
DC '00010203040506070809' 
ORG , 
*---------------------------------------------------------------------* 
LTORG , 
*   Define the SSCVT as an IPCS model.                                 * 
*---------------------------------------------------------------------* 
SSCTMODL BLSEQMDEF BASELBL=SSCT, X 
   CBLEN=SSCTSIZEx, X 
   PREFIX=x=4, # of chars to remove from lbl nm X 
   ACRONYM=SSCT, X
ACROLBL=SSCTID,
HEADER=SSCVT
BLSQMFLD NAME=SSCTID, DTYPE=EBCDIC
BLSQMFLD NAME=SSCTSCCTA, DTYPE=HEX
BLSQMFLD NAME=SSCTSNAM, DTYPE=EBCDIC
* BLSQMFLD NAME=SSCTSNAM, DTYPE=HEX
BLSQMFLD NAME=SSCTFLG1, DTYPE=HEX
BLSQMFLD NAME=SSCTSSID, DTYPE=HEX
BLSQMFLD NAME=SSCTRSV1, DTYPE=HEX
BLSQMFLD NAME=SSCTSSVT, DTYPE=HEX
BLSQMFLD NAME=SSCTSNAM, DTYPE=HEX
BLSQMFLD NAME=SSCTSSVT, DTYPE=HEX
BLSQMFLD NAME=SSCTSYN, DTYPE=HEX
BLSQMFLD NAME=SSCTSUS2, DTYPE=HEX
BLSQMFLD NAME=SSCTRSV3, DTYPE=HEX
BLSQMFLD SHDR=BLNKLINE, NEWLINE
BLSQMDEF END
BLNKLINE BLSQSHDR ' '
* ***************************************************** *

WORKAREA DSECT
SAVEAREA DS 18F
CALLLST CALL ,(),(),(), MF=L
REGSAVE DS 16F
ASI D DS XL2
FLAG1 DS XL1
SYMTRY EQU X'80'
CVTADDR DS F
SSCTNEXT DS F
SSVTADDR DS F
FNUMSAVE DS F
WORKPACC DS ØD, CL(ADPLLACC)
WORKPFMT DS ØD, CL(ADPLLFORM)
WORKESSY DS ØD, CL(ESSYHRL)
WORKPPR2 DS ØD, CL(PPR2999-PPR2000)
LINELEN DS F
LINEBUF DS CL(132)
SNAMSAVE DS CL(8)
SNAMHXSV DS CL(8)
SNAMHXLN DS F
SSNMPARM DS CL(4)
DBL1 DS 2D
DBL2 DS 2D
WORKLEN EQU *. WORKAREA
R⁰ EQU Ø
R¹ EQU 1
R² EQU 2
R³ EQU 3
R⁴ EQU 4
R⁵ EQU 5
R⁶ EQU 6
R⁷ EQU 7
Parsing strings in Assembler programs

One of the great strengths of REXX is the ability to parse strings to extract substrings, words, and delimited arguments. The following macro and Assembler routines attempt to re-create some of the more common string handling functions for use in Assembler programs.

In order to take advantage of the parse functions, this article provides the following:

- **RDSPARID** – Assembler routine to provide INDEX functions.
- **RDSPARPT** – Assembler routine to provide PATTERN matching.
- **RDSPARST** – Assembler routine to provide STRIP functions.
• RDSPARVR – Assembler routine to provide PARSE VAR functions.
• RDSPARWI – Assembler routine to provide WORDINDEX functions.
• RDSPARWS – Assembler routine to provide WORDS functions.
• RDSPARWD – Assembler routine to provide WORD functions.
• RDSPARSE – Assembler program to provide linkage to the above Assembler routines.
• PARSE Assembler programming interface to the RDSPARSE program.

USING THE PARSE MACRO
The PARSE macro allows the Assembler programmer to easily invoke the RDSPARSE program for the string handling function desired.

Standard and execute form syntax
The standard and execute forms of the PARSE macro are written as follows:
• name – name: symbol. Begin name in column 1.
• PARSE – one or more blanks must precede PARSE. One or more blanks must follow PARSE.

Valid parameters (required parameters are underlined.)
• INDEX – STRING, SUBSTR, RESULT
• PATTERN – STRING, MASK, RESULT
• STRIP – STRING, SUBSTR, OPTION, RESULT
• WORD – STRING, WORDNUM, RESULT
• WORDS – STRING, RESULT
• WORDINDEX – STRING, WORDNUM, RESULT
• VAR – STRING, FIELDS
• ,STRING=source_data – RX-type address or register (2) – (12)
• ,SUBSTR=substring – RX-type address or register (2) – (12)
• ,OPTION=LEADING – default OPTION=BOTH
  ,OPTION=TRAILING
  ,OPTION=BOTH
• ,MASK=mask_data – RX-type address or register (2) – (12)
• ,WORDNUM=word_number – RX-type address or register (2) – (12)
• ,FIELDS=(field1, – RX-type address or register (2) – (12)
  field2, – RX-type address or register (2) – (12)
  …,
  fieldn) – RX-type address or register (2) – (12)
• ,RESULT=result_data – RX-type address or register (2) – (12)
• ,MF=S – standard form
  ,MF=(E,label) – execute form

STANDARD AND EXECUTE FORM PARAMETERS
The parameters are as follows.

INDEX
This function call emulates the REXX ‘INDEX’ function to return the position of ‘substring’ within ‘source_data’. If ‘substring’ is not found PARSE INDEX returns 0 in ‘result_data’. If ‘substring’ is found ‘result_data’ contains the offset relative to 1 in hex format of the first character of ‘substring’ within ‘source_data’.

PATTERN
This function call provides a generic mask-matching function.
The contents of ‘source_data’ are compared with a mask value in ‘mask_data’. If ‘source_data’ matches ‘mask_data’ the function returns a 1 in ‘result_data’, otherwise 0 is returned. Wildcards of ‘*’ and ‘%’ can be used to specify multiple or single characters respectively.

**STRIP**

This function call emulates the REXX ‘STRIP’ function to remove the leading and/or trailing characters from ‘source_data’. The default character to be stripped is a blank (X’40’) but this can be overridden by ‘substring’. The value of the OPTION keyword specifies whether leading and/or trailing characters are removed; the default is ‘BOTH’.

**WORD**

This function call emulates the REXX ‘WORD’ function to return a specific word from ‘source_data’. The words within ‘source_data’ must be separated from each other by blanks. The word that is returned is specified a 4-byte hex number in ‘word_number’. If successful the word is returned in ‘result_data’.

**WORDS**

This function call emulates the REXX ‘WORDS’ function to return the number of words in ‘source_data’ into ‘result_data’. The number of words returned is a 4-byte hex number.

**WORDINDEX**

This function call emulates the REXX ‘WORDINDEX’ function to return the position within ‘source_data’ of the first character of a specific word specified by ‘word_number’. If the word is not found, PARSE WORDINDEX returns 0 in ‘result_data’. If the word is found, ‘result_data’ contains the offset relative to 1 in hex format of the first character of ‘word_number’ within ‘source_data’.

**VAR**

This function call emulates the REXX ‘PARSE VAR’ function to
split the characters in ‘source_data’ into sub-strings, depending on the contents of ‘field1’ to ‘fieldn’. Individual ‘fieldn’ parameters can be either a separator field to specify the characters used to divide ‘source_data’ into component substrings, or a result field to hold the contents of the component substrings.

'STRING='source_data’
Specifies the address of the source string to be parsed. The data must be constructed of a 1-byte length field followed immediately by the actual string data.

'SUBSTR='substring’
Specifies the address of a sub-string to be passed to the relevant PARSE function. It must be constructed of a 1-byte length field followed immediately by the actual sub-string data. When using PARSE STRIP, the length field must be set to 1 and followed by a single character.

'OPTION=LEADING,OPTION=TRAILING,OPTION=BOTH
Specifies the option to be used during PARSE STRIP:
• OPTION=LEADING – the leading characters only are removed from ‘source-data’.
• OPTION=TRAILING – the trailing characters only are removed from ‘source-data’.
• OPTION=BOTH – both the leading and trailing characters are removed from ‘source-data’. This is the default.

'MASK='mask_data’
Specifies the address of a generic mask pattern to be compared against ‘source_data’ during a PARSE PATTERN function call. It must be constructed of a 1-byte length field followed immediately by the mask characters.

A wildcard character of ‘*’ can be used to signify one or more
characters. A wildcard character of ‘%’ can be used to signify just one character.

,WORDNUM=‘word_number’
Specifies the address of the word number to be used in the PARSE WORD and WORDINDEX functions. It must be constructed of a 1-byte length field followed immediately by the 4-byte hex format number.

,FIELDS=(field1,field2,...,fieldn)
Specifies the list of fields to be used during the PARSE VAR function call. Each field must be the address comprising a 1-byte length field followed immediately by the field data. The fields can be one of two forms:

- Separator field – the field data specifies the characters used to divide the input string ‘source_data’ into its component sub-strings. The default separator is blanks. The 1-byte length field must be non-zero as this declares the field as a separator field to the PARSE VAR function.
- Result field – this field receives the component sub-strings that have been separated from ‘source_data’. The 1-byte length field must be set to X'00' as this identifies the field as a result field to the PARSE VAR function.

On return from the PARSE VAR function, any applicable sub-string will be copied to the result field data and its length copied into the 1-byte length field. If the 1-byte length field is X'00' then there is no applicable sub-string data for this field.

A special field value of ‘<.>’ can be used to indicate that any result data can be thrown away for this field.

,RESULT=‘result_data’
Specifies the address of the result data area for all PARSE functions except for PARSE VAR. It must be constructed of a 1-byte length field followed immediately by enough storage to
contain the result data. On return from the PARSE function, the 1-byte length field will contain the length of the data returned (if any).

,MF=S
Specifies the standard form of PARSE. The standard form places the parameters into an in-line parameter list.

,MF=(E,label)
Specifies the execute form of PARSE. The execute form generates code to put the parameters into the storage pointed to by ‘label’.

LIST FORM SYNTAX
The list forms of the PARSE macro are written as follows:
• name – name: symbol. Begin name in column 1.
• PARSE – one or more blanks must precede PARSE. One or more blanks must follow PARSE.

Valid parameters are:
• INDEX
• PATTERN
• STRIP
• WORD
• WORDS
• WORDINDEX
• VAR FIELDS
• ,FIELDS=(field1, RX-type address or register (2) - (12)
  field2, RX-type address or register (2) - (12)
  …,
  fieldn) RX-type address or register (2) - (12)
• ,MF=(L,label) List form
LIST FORM PARAMETERS

The parameters are explained as follows:

- **INDEX** – generate parameter list storage for a PARSE INDEX function call.
- **PATTERN** – generate parameter list storage for a PARSE PATTERN function call.
- **STRIP** – generate parameter list storage for a PARSE STRIP function call.
- **WORD** – generate parameter list storage for a PARSE WORD function call.
- **WORDINDEX** – generate parameter list storage for a PARSE WORDINDEX function call.
- **WORDS** – generate parameter list storage for a PARSE WORDS function call.
- **VAR** – generate parameter list storage for a PARSE VAR function call.
- **,FIELDS=(field1,field2,,..,fieldn)** – required only for the list form of the PARSE VAR function call. The number of fields must match the number of fields in the corresponding execute form to enable enough storage to be reserved for the parameter list.
- **,MF=(L,label)** – specifies the list form of PARSE. The list form generates code to reserve enough storage to contain the parameter list and assigns ‘label’ as the reference name.

EXAMPLES OF USING THE PARSE MACRO

1. Strip leading zeros from ‘000025.00’:

   ```
   LA R4, INPUT * Point to the input string
   PARSE STRIP, * Use PARSE STRIP X
   OPTION=LEADING, * Only remove leading chars X
   STRING=(R4), * This input text X
   SUBSTR=ZERO, * Strip char = '0' X
   RESULT=OUTPUT * and place output here
   ```
On return from PARSE, OUTLEN would contain X'05' and OUTDATA would contain '25.00'.

The Assembler code above equates to the following REXX to perform the same task:

\[
\begin{align*}
\text{INPUT} & = '0000025.00' \\
\text{OUTDATA} & = \text{STRIP}(\text{INPUT}, L, 0) \\
\text{OUTLEN} & = \text{LENGTH}(\text{OUTDATA})
\end{align*}
\]

2. Scan the following SYSIN text to retrieve the setting of NAME. Once retrieved, ensure that any blanks around the dataset name are removed.

' DEF NVSAM(NAME(A.B ) DEVT(3390) VOL(TSO001))'

\[
\begin{align*}
\text{LA R8, NAME} & \quad \text{Point to result field} \\
\text{PARSE VAR,} & \quad \text{Use PARSE VAR X} \\
\text{STRING=} & \quad \text{SYSIN,} \quad \text{On this input string X} \\
\text{FIELDS=} & \quad (\text{<.>}, \text{NAMESEP, (R8), BRACKET, <.>}) \\
\text{PARSE STRIP,} & \quad \text{Use PARSE STRIP X} \\
\text{STRING=} & \quad \text{NAME,} \quad \text{Input is NAME X} \\
\text{RESULT=} & \quad \text{NAME} \quad \text{Replace original with stripped}
\end{align*}
\]

On return from PARSE, NAMELEN would contain X'03' and NAMEDATA would contain 'A.B'.

The Assembler code above equates to the following REXX to perform the same task:

\[
\text{NAMESEP} = 'NAME('}
\]
Scan the following SYSIN text to retrieve the setting of VOL and check that it is of the form ‘TS%0*’. Assume the code would reside in a re-entrant program.

```
' DEF NVSAM(NAME(A.B ) DEVT(339Ø) VOL(TSOØØ1))'
```

```assembly
LA R7, VOLSEP * Point to the VOL separator
PARSE VAR, * Use PARSE VAR X
STRING=SYSIN, * On the SYSIN text X
FIELDS=(<.>, * Throw beginning away X
(R7), VOLSER, BRACKET, * Place result in VOLSER X
<.>), * Throw remainder away X
MF=(E, PARSEV1) * Use the PARSEV1 parm list
CLI VOLLEN, X'ØØ' * Did we get a value for VOL ?
BE ERROR * No - exit with error
PARSE PATTERN, * Use PARSE PATTERN X
STRING=VOLSER, * On the VOLSER text X
MASK=VOLMASK, * Use the VOLSER mask field X
RESULT=RESULT, * Place Result here X
MF=(E, PARSEP1) * Use the PARSEP1 parm list
ICM R15, B'1111', RESDATA * Load up result
BZ NOMATCH * Zero = nomatch
MATCH EQU *

SYSIN DC AL1(72)
DC CL72' DEF NVSAM(NAME(A.B ) DEVT(339Ø) VOL(TSOØØ1))'
VOLSEP DC AL1(4), C'VOL(' * Separator field
VOLMASK DC AL1(5), C'TS%0*' * Pattern mask
BRACKET DC AL1(1), C')' * Separator field

WORKAREA DSECT
RESULT DS OC * Result field for PARSE PATTERN
RESLEN DS AL1 * Result field length
RESDATA DS XL4 * Result field data
VOLSER DS OC * Result field to hold VOLSER
VOLLEN DS AL1 * Result field length
VOLDATA DS CL6 * Result field data
PARSE VAR, FIELDS=(,,,), MF=(L, PARSEV1)
PARSE PATTERN, MF=(L, PARSEP1)
WORKLEN EQU *-WORKAREA
```

The Assembler code above equates to the following REXX
to perform the same task:

```assembly
VOLSEP = 'VOL('
BRACKET = ')
SYSIN = 'DEF NVSAM(NAME(A.B) DEVT(339C) VOL(TSO001))'
PARSE VAR SYSIN. (VOLSEP) VOLDATA (BRACKET).
IF SUBSTR(VOLDATA,1,2) <> 'TS' THEN EXIT 4
IF SUBSTR(VOLDATA,4,1) <> 'O' THEN EXIT 4
VOLLEN = LENGTH(VOLDATA)
```

4 Retrieved the last word from the following text:

```plaintext
'THE BOY RAN AWAY CLUTCHING HIS ICE-CREAMS'
```

```assembly
PARSE WORDS, * Get number of words X
STRING=TEXT, * From this text X
RESULT=NUMWORDS * and place result in WORDNUM
PARSE WORD, * Get word X
STRING=TEXT, * From this text X
WORDNUM=NUMWORDS, * This word number = last X
RESULT=LASTWORD * and place result in LASTWORD
... ...
TEXT DC AL1(72)
DC CL72'THE BOY RAN AWAY CLUTCHING HIS ICE-CREAMS'
NUMWORDS DS OC * Result field for PARSE WORDS
NUMLEN DS AL1 * Result field length
NUMDATA DS XL4 * Result field data
LASTWORD DS OC * Result field for PARSE WORD
LASTLEN DS AL1 * Result field length
LASTDATA DS CL16 * Result field data
```

On return from PARSE WORD, LASTLEN would contain X'0A' and LASTDATA would contain 'ICE-CREAMS'.

The Assembler code above equates to the following REXX to perform the same task:

```plaintext
TEXT = 'THE BOY RAN AWAY CLUTCHING HIS ICE-CREAMS'
NUMWORDS = WORDS(TEXT)
LASTDATA = WORD(TEXT, NUMWORDS)
LASTLEN = LENGTH(LASTDATA)
```

### SOURCE CODE FOR THE PARSE MACRO

```assembly
MACRO

.*--------------------------------------------------------------------
.* MACRO NAME : PARSE
.*
.* FUNCTION : THE PARSE MACRO PROVIDES REXX TYPE PARSE FUNCTIONS

```
TO ASSEMBLER PROGRAMS. IT CREATES A PARAMETER LIST AND CALLS THE 'RDSPARSE' PROGRAM.

SYNTAX

PARSE parse_type,
STRING=source_data,
OPTION=option_data,
SUBSTR=substring,
MASK=mask_data,
WORDNUM=word_number,
FIELDS=(field1,field2...fieldn),
RESULT=result_data,
MF=S
MF=(L,label)
MF=(E,label)

KEYWORDS

parse_type
is the type of parse function required.

INDEX
specifies that the offset into 'source_data', relative to one, of the first character of the string specified by 'substring' is to be returned in 'result_data'.

PATTERN
specifies that the 'source_data' is to be compared to the generic pattern specified in 'mask_data'.
If there is a match, 'result_data' is set to 1, otherwise it is set to 0.

STRIP
specifies that the 'source_data' is to have its leading and/or trailing characters removed and the result placed in 'result_data'.
The character to be stripped is specified in 'substring' (Default is a space x'40').
The setting of 'option_data' specifies if the leading and/or trailing characters are to be removed.

WORD
specifies that the word whose number is specified in 'word_number' is to be copied from 'source_data' into 'result_data'.

WORDINDEX
specifies that the offset into 'source_data', relative to one, of the first character of the word number specified in 'word_number' is to be returned in 'result_data'.

WORDS
  specifies that the number of words in 'source_data'
  is to be returned in 'result_data'.

VAR
  specifies that 'source_data' is to be parsed
  according to the result and data fields specified
  in the FIELDS list.

: STRING=source_data
  is the name (RX-Type) or address in register
  (2)-(12) of the source data. The source data
  must be constructed of a 1 byte length field
  followed by the source data to be parsed.

: OPTION=option_data
  is the option passed to the PARSE function.

For STRIP
  LEADING = Strip leading characters ONLY
  TRAILING = Strip trailing characters ONLY
  BOTH     = Strip both leading and trailing
              characters.

: MASK=mask_data
  is the generic pattern to be used in the PARSE
  PATTERN function and must be RX-Type format or
  address in register (2)-(12). The mask-data is constructed of a 1
  byte length field followed by the mask data.
  A wild card of '*' can be used to match one or
  more characters.
  A placeholder of '%' can be used to match just
  one character.

: WORDNUM=word_number
  is the word number to be used in the PARSE WORD and
  WORDINDEX functions and must be RX-Type format or
  address in register (2)-(12). The word_number is constructed of a
  1-byte length field which must be 4, followed by a four byte hex
  word number.

: SUBSTR=substring
  is the substring data to be passed to the PARSE
  function and must be RX-Type format or address
  in register (2)-(12). The substring is constructed of a 1
  byte length field followed by the substring data.
For INDEX
    'substring' specifies the search string data.

For STRIP
    'substring' specifies the strip character.

: FIELDS=(field1,field2...fieldn)
    is the list of field names to be used with
    'source_data' during a PARSE VAR operation.
    Each field must be constructed of a 1 byte length
    field followed by the field data.
    Each field name must be RX-Type format or address
    in register (2)-(12).

Each field can take one of two forms:

(1) A Separator Field
    This is defined by a field whose field data
    length is set to the length greater than
    zero.

(2) A Result Field
    This is defined by a field whose field data
    length is set to zero. The field data area
    will be populated by the PARSE program.
    To indicate that a successful parse has
    taken place, the field data length will
    be set to the length of the data returned.
    A length of zero in the field data length
    on return from the PARSE program indicates
    no data has been returned in the result field.

Special Case :
    Specifying a field of <.> can be used to emulate
    the placeholder function in REXX. If specified, the
    result data that would enter this field is thrown
    away.

: RESULT=result_data
    is the name (RX-Type) or address in register
    (2)-(12) of the result data. The result data
    must be constructed of a 1 byte length field
    followed by enough bytes to contain the result
    returned from the PARSE function. This field is
    required for all function types except VAR.

: MF=S
    specifies the standard form of the macro. The "S"
    form generates code to put the parameters into an
    in-line parameter list and invoke the desired
    service
MF=(L,label)

specifies the list form of the macro. The "L" form
defines an area to be used for the parameter list.
All keywords applicable to the PARSE function
specified must be coded in order for the macro to
calculate the space required for the parameter list.

MF=(E,label)
specifies the execute form of the macro. The "E" form
generates code to put the parameters into the storage
pointed to by 'label'.

&LABEL PARSE &TYPE, X
&STRING=, x
&OPTION=, x
&SUBSTR=, x
&MASK=, x
&WORDNUM=, x
&FIELDS=, x
&RESULT=, x
&MF=S

Ensure that we have all required parms

&NUMFLDS SETA N'&FIELDS
&NUMMF SETA N '&MF
AIF ('&TYPE' EQ '').ERRORØ
AIF ('&TYPE' NE 'VAR').CHKRES
AIF ('&MF(1)' EQ 'L').CHKMF
AGO .CHKSTR
.CHKRES ANOP
AIF ('&RESULT' EQ '' AND '&MF(1)' NE 'L'). ERROR1
.CHKSTR ANOP
AIF ('&STRING' EQ '' AND '&MF(1)' NE 'L'). ERROR2
.CHKMF ANOP
AIF (&NUMMF EQ 1).CHKTYPE
AIF (&NUMMF NE 2).ERROR5
&MFLABEL SETC '&MF(2)'
.CHKTYPE ANOP

Now check which PARSE function is required

AIF ('&TYPE' EQ 'INDEX').TYPINDEX
AIF ('&TYPE' EQ 'PATTERN').TYPPAT
AIF ('&TYPE' EQ 'STRIP').TYPSTRIP
AIF ('&TYPE' EQ 'WORD').TYPWORD
AIF ('&TYPE' EQ 'WORDINDEX').TYPWINDX
AIF ('&TYPE' EQ 'WORDS').TYPWORDS
AIF ('&TYPE' EQ 'VAR').TYPVAR
AGO .ERROR3
.TYPSTRIP ANOP
.* Set up constants for the STRIP function call
.*
&WORKSZC SETC '40'
&PARSETYP SETC '40'
AGO .GETMF

.TYPINDEX ANOP
.* Set up constants for the INDEX function call
.*
&WORKSZC SETC '36'
&PARSETYP SETC '10'
AGO .GETMF

.TYPWINDX ANOP
.* Set up constants for the WORDINDEX function call
.*
&WORKSZC SETC '36'
&PARSETYP SETC '08'
AGO .GETMF

.TYPWORD ANOP
.* Set up constants for the WORD function call
.*
&WORKSZC SETC '36'
&PARSETYP SETC '04'
AGO .GETMF

.TYPWORDS ANOP
.* Set up constants for the WORDS function call
.*
&WORKSZC SETC '32'
&PARSETYP SETC '02'
AGO .GETMF

.TYPVAR ANOP
.* Set up constants for the VAR function call
.*
AIF ('&FIELDS' EQ '').ERROR7
&WORKSZ SETA (4*&NUMFLDS)+28
&WORKSZC SETC '&WORKSZ'
&PARSETYP SETC '01'
AGO .GETMF

* GETMF ANOP
*--------------------------------------------------------------------
* Examine the MF setting and decide what to do
*--------------------------------------------------------------------
AIF (' &MF(1)' NE 'S'). MFNOTS
&LABEL CNOP 0, 4   Align Fullword
B ' *+&WORKSZC+4 Branch round parameter list
DS  XL&WORKSZC   Parameter list
LA 1, '* .&WORKSZC Point to parameter list
AGO . GETTYPE

MFNOTS ANOP
AIF (' &MF(1)' NE 'L'). MFNOTL
AIF (' &LABEL' NE ''). ERROR12
AIF (' &MFLABEL'(1,1) EQ '). ERROR11
DS OF Align Fullword
&MFLABEL DS  XL&WORKSZC Parameter list
AGO . END

MFNOTL ANOP
AIF (' &MF(1)' NE 'E'). ERROR6
AIF (' &MFLABEL'(1,1) EQ '). MFREG
LA 1, &MFLABEL Point to parameter list
AGO . GETTYPE

MFREG ANOP
® SETC '&MFLABEL'(2, K' &MFLABEL-2)
LR 1, ® Point to parameter list

GETTYPE ANOP
MVI 0(1), X' &PARSETYP' Indicate TYPE
MVC 4(4,1), =X'01FF0000' Move in default settings
MVC 8(4,1), =X'0140004B' Move in default settings
LR 15, 1 Point to flag
ST 15, 12(1) Store in parameter list
AGO . STRING

STRING ANOP
* Process the STRING keyword
*--------------------------------------------------------------------
AIF (' &STRING'(1,1) EQ '). STRREG

* STRING=variable specified
*--------------------------------------------------------------------
LA 15, &STRING Point to the source string
AGO . STORESTR

STRREG ANOP
* STRING=(Rx) specified
*--------------------------------------------------------------------
® SETC '&STRING'(2, K' &STRING-2)
LR 15, ® Point to the source string

STORESTR ANOP
ST 15, 16(1) Store in parameter list
LA 15,24(1)  Point to other keywords
ST 15,20(1)  Store in parameter list

.RESTPARM ANOP

* Process the other keywords depending on the TYPE setting

&RESOFF SETA 24
&RESOFFC SETC &RESOFF
AIF ('&TYPE' EQ 'WORDS').RESULT
AIF ('&TYPE' EQ 'PATTERN').MASK
AIF ('&TYPE' EQ 'WORD').WORDNUM
AIF ('&TYPE' EQ 'WORDINDEX').WORDNUM
AIF ('&TYPE' EQ 'VAR').GETFLDS
AGO .SUBSTR

.GETFLDS ANOP

* Process the FIELDS keyword

&I SETA 1

.FLDLOOP ANOP

* Loop through all the FIELDS variables and store their addresses
* in the parameter list.

&OFF SETA &I-1
&FLDOFF SETA &OFF*4+24
&FLDNAME SETC &FIELDS(&I)
&FLDOFFN SETC &FLDOFF
AIF ('&FLDNAME' NE '<>').FLDNORM
LA 15,10(1)
AGO .STORFLD

.FLDNORM ANOP
AIF ('&FLDNAME'(1,1) EQ '(').FLDREG
LA 15,&FLDNAME         Get address of field entry
AGO .STORFLD

.FLDREG ANOP
&FREG SETC '&FLDNAME'(2,K'&FLDNAME-2)
LR 15,&FREG             Get address of field entry

.STORFLD ST 15,&FLDOFFN(1) Store in parameter list
&I SETA &I+1
AIF (&I GT &NUMFLDS).FLDLOOPE
AGO .FLDLOOP

.FLDLOOPE ANOP
&NULLOFF SETA &FLDOFF+4
&NULLOFFC SETC &NULLOFF
AGO .LINKPGM

* .SUBSTR ANOP

* Process the SUBSTR keyword

*--------------------------------------------------------------------
* Process the SUBSTR keyword
*--------------------------------------------------------------------

AIF ('&SUBSTR' NE '').SUBTEST
AIF ('&TYPE' NE 'STRIP').ERROR4
LA 15,8(1) Point to default character
AGO .STORESUB
.SUBTEST AIF ('&SUBSTR'(1,1) EQ '(').SUBREG

* SUBSTR=variable specified
*--------------------------------------------------------------------
LA 15,&SUBSTR Point to the substring
AGO .STORESUB
.SUBREG ANOP

* SUBSTR=(Rx) specified
*--------------------------------------------------------------------
® SETC '&SUBSTR'(2,K+'&SUBSTR-2)
LR 15,® Point to the substring
.STORESUB ANOP
ST 15,24(1) Store in parameter list
&RESOFF SETA 28
&RESOFFC SETC '&RESOFF'
AIF ('&TYPE' EQ 'STRIP').OPTION
AGO .RESULT
.MASK ANOP
AIF ('&MASK' EQ '').ERROR9

* Process the MASK keyword
*--------------------------------------------------------------------
AIF ('&MASK'(1,1) EQ '(').MASKREG

* MASK=variable specified
*--------------------------------------------------------------------
LA 15,&MASK Point to the mask
AGO .STOREMAS
.MASKREG ANOP

* MASK=(Rx) specified
*--------------------------------------------------------------------
® SETC '&MASK'(2,K+'&MASK-2)
LR 15,® Point to the mask
.STOREMAS ANOP
ST 15,24(1) Store in parameter list
&RESOFF SETA 28
&RESOFFC SETC '&RESOFF'
AGO .RESULT
.WORDNUM ANOP
AIF ('&WORDNUM' EQ '').ERROR10

* Process the WORDNUM Keyword
*--------------------------------------------------------------------
AIF ('&WORDNUM'(1,1) EQ '(').WRDNKREG
* WORDNUM=variable specified

LA 15, &WORDNUM Point to the word number
AGO .STOREWDN

* WORDNUM=(Rx) specified
® SETC '&WORDNUM' (2, K 'WORDNUM' - 2)
LR 15, @ Point to the word number

* Process the OPTION keyword

AIF ('&OPTION' EQ '').OPTDONE
.OPTLEAD AIF ('&OPTION' NE 'LEADING').OPTTRAIL
MVI 5(1), X'FØ' Indicate strip leading chars
AGO .OPTDONE
.OPTTRAIL AIF ('&OPTION' NE 'TRAILING').OPTBOTH
MVI 5(1), X'ØF' Indicate strip trailing chars
AGO .OPTDONE
.OPTBOTH AIF ('&OPTION' NE 'BOTH').ERROR8
.OPTDONE ANOP

* Process the RESULT keyword

AIF ('&RESULT'(1, 1) EQ '(').RESREG
.* RESULT=variable specified
.LA 15, &RESULT Point to the result field
AGO .STORERES

* RESULT=(Rx) specified
® SETC '&RESULT'(2, K '&RESULT' - 2)
LR 15, @ Point to the result field
.STORERES ANOP
SOURCE CODE FOR THE RDSPARSE PROGRAM

RDSPARSE TITLE 'ASSEMBLER ROUTINE TO PARSE STRINGS'

* Nname : RDSPARSE
* Function : This program acts as a 'stub' to pass control to the required RDSPARxx program. The type of PARSE

ST 15,&RESOFFC.(1) Store in parameter list
&NULLOFF SETA &RESOFF+4
&NULLOFFC SETC 'GNULLOFF'
AGO .LINKPGM
.
.LINKPGM ANOP
XR 15,15 Create null entry
ST 15,&NULLOFFC.(1) Store in parameter list
LA 1,12(1) Point to parameter list
LINK EP=RDSPARSE Link to RDSPARSE
AGO .END
.
* Error messages

.ERRORØ MNOTE 12,'PARSE type was not specified'
AGO .END
.ERROR1 MNOTE 12,'Required keyword RESULT was not specified'
AGO .END
.ERROR2 MNOTE 12,'Required keyword STRING was not specified'
AGO .END
.ERROR3 MNOTE 12,'Invalid value specified for PARSE type'
AGO .END
.ERROR4 MNOTE 12,'Required keyword SUBSTR was not specified'
AGO .END
.ERROR5 MNOTE 12,'Too many parameters in the MF keyword'
AGO .END
.ERROR6 MNOTE 12,'Invalid MF value specified - use L,E or S'
AGO .END
.ERROR7 MNOTE 12,'Required keyword FIELDS was not specified'
AGO .END
.ERROR8 MNOTE 12,'Invalid value for OPTION'
AGO .END
.ERROR9 MNOTE 12,'Required keyword MASK was not specified'
AGO .END
.ERROR10 MNOTE 12,'Required keyword WORDNUM was not specified'
AGO .END
.ERROR11 MNOTE 12,'Invalid use of register as label when MF=L'
AGO .END
.ERROR12 MNOTE 12,'Invalid use of Assembler label when MF=L'
AGO .END
.END MEND

© 2003, Xerphon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.
to perform is passed as the first parameter passed
to the program. This is examined and the
rest of the parameter list is relayed to the
appropriate routine:

<table>
<thead>
<tr>
<th>Function</th>
<th>Hex Code</th>
<th>Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE=STRIP</td>
<td>X'40'</td>
<td>RDSPARST</td>
</tr>
<tr>
<td>TYPE=PATTERN</td>
<td>X'20'</td>
<td>RDSPARP T</td>
</tr>
<tr>
<td>TYPE=INDEX</td>
<td>X'10'</td>
<td>RDSPARI D</td>
</tr>
<tr>
<td>TYPE=WORDINDEX</td>
<td>X'08'</td>
<td>RDSPARWI</td>
</tr>
<tr>
<td>TYPE=WORD</td>
<td>X'04'</td>
<td>RDSPARWD</td>
</tr>
<tr>
<td>TYPE=WORDS</td>
<td>X'02'</td>
<td>RDSPARWS</td>
</tr>
<tr>
<td>TYPE=VAR</td>
<td>X'01'</td>
<td>RDSPARVR</td>
</tr>
</tbody>
</table>

Attributes: Amode(31)
Rmode(Any)
REN T

Register Usage:
R1 - Parameters passed: +0 Address of Option
+4 Address of Source Data:
| LL| Source Data |
+--------+
R2 - Pointer to Option specified
R3 - Pointer to Source Data
R4 - Pointer to Template List
R5 -
R6 -
R7 - Address of parameters to be relayed
R8 -
R9 -
R10 - Branch and Link
R11 -
R12 - BASE
R13 - Savearea

------------------------------------------------------------------*

RDSPARSE CSECT
RDSPARSE AMODE 31
RDSPARSE RMODE ANY
BAKR R14,R0

linkage stack
LR    R12, R15            copy entry address to base
USING RDSPARSE, R12       address it
MODID
LR    R2, R1              protect parms
STORAGE OBTAIN,           grab some storage
LENGTH=WORKL,             this much
ADDR=(R13)                put address in r13
MVC 4(4, R13), =C'F1SA'  set acronym in save area

GETPARMS EQU *           
LR    R1, R2              restore parms
LA    R7, 4(R1)           r7 -> parms that are passed on
LM    R2, R4, Ø(R1)       copy parms passed
*                                     r2 -> options
*                                     r3 -> source data
*                                     r4 -> template list

TM    Ø(R2), WANT_RDSPARVR do we want parse var ?
BNO   CHKWORDS             no - check next option
L     R15, RDSPARVR_PGM    get address of RDSparvr pgm
LR    R1, R7               copy parm list
BALR R14, R15              branch to program
B     RETURNØØ             leave

CHKWORDS EQU *
TM    Ø(R2), WANT_WORDS   do we want words ?
BNO   CHKWORD              no - check next option
L     R15, WORDS_PGM      get address of words pgm
LR    R1, R7               copy parm list
BALR R14, R15              branch to program
B     RETURNØØ             leave

CHKWORD EQU *
TM    Ø(R2), WANT_WORD    do we want word ?
BNO   CHKWORDI             no - check next option
L     R15, WORD_PGM       get address of word pgm
LR    R1, R7               copy parm list
BALR R14, R15              branch to program
B     RETURNØØ             leave

CHKWORDI EQU *
TM    Ø(R2), WANT_WORDIDX do we want wordindex ?
BNO   CHKIIDX              no - check next option
L     R15, WORDIDX_PGM    get address of wordindex pgm
LR    R1, R7               copy parm list
BALR R14, R15              branch to program
B     RETURNØØ             leave

CHKIIDX EQU *
TM    Ø(R2), WANT_INDEX   do we want index ?
BNO   CHKPATTTN            no - check next option
L     R15, INDEX_PGM      get address of index pgm
LR    R1, R7               copy parm list
BALR R14, R15              branch to program
B     RETURNØØ             leave

CHKPATTN EQU *
**EDITOR'S NOTE:** *THE CODE WILL BE CONCLUDED IN THE NEXT ISSUE.*

Rob Scott  
MVS Consultant (USA)  © Rob Scott 2003
Serena Software is partnering with Relativity Technologies to sell a combined package that’s supposed to streamline the process of modernizing and maintaining legacy applications.

Specifically, Serena is leveraging its ChangeMan ZDD, which promotes desktop development on z/OS and OS/390 platforms, to work with Relativity’s RescueWare legacy modernization product. The combination is said to make it possible for sites to quickly retrieve data locked in legacy systems, and then update and maintain that information directly from desktop systems without having to use other tools like FTP and NDM.

RescueWare allows companies to leverage and reuse existing legacy application source code rather than having to manually reprogram the applications from scratch or lose them all together. ChangeMan ZDD allows access to legacy components while streamlining and improving the entire data conversion process. The combination apparently means RescueWare is more intuitive to operate and customers can achieve significant gains in overall productivity, development efficiency, and software quality.

For further information contact:
Serena Software, 2755 Campus Drive, 3rd Floor, San Mateo, CA 94403, USA.
Tel: (650) 522 6600.

* * *

IBM has announced DB2 UDB Version 8, a new re-engineered database for z/OS. New in this version are 64-bit virtual addressing, ‘extensive’ enhancements to SQL, and usability and portability enhancements through major catalogue changes.

There are major improvements in long object names, Unicode for worldwide support and improved SQL compatibility, DB2 family compatibility for portability of transaction applications from Unix and Windows environments, and enhanced data availability through on-line schema evolution.

For further information contact your local IBM representative.

* * *

ASG has announced that its ASG-TMON family of availability and performance monitoring tools provide support for z/OS V1R4.

For further information contact:
ASG, 1333 Third Avenue South, Naples, FL 34102, USA.
Tel: (239) 435 2200.

* * *

Embarcadero Technologies and Rocket Software have announced a joint venture whereby the former’s DBArtisan database administration product will be enhanced for DB2 for OS/390.

For further information contact:
Embarcadero Technologies, 425 Market Street, Suite 425, San Francisco, CA 94105, USA.
Tel: (415) 834 3131.