

Getting Started

This section contains information necessary to communicate with the various operating system services provided by the Entire System Server. The examples included show the use of the Entire System Server in heterogeneous environments running operating systems such as OS/390, VSE/ESA, and BS2000/OSD. The examples consist of Natural programs that reference Entire System Server views, and show the resulting output.

An example program for every available view is contained in an online tutorial delivered with the Entire System Server (see Section Online Tutorial for more information).

This section covers the following topics:

- Syntax Used to Access Entire System Server
- Using Entire System Server in a Multi-Computer Environment
- Using Entire System Server Update View Processors
- Examples

Syntax Used to Access Entire System Server

Entire System Server views can be accessed from any Natural program using the Natural statement PROCESS or FIND. Which statement you will use depends on the type of view accessed, the access mode required, and the number of records selected.

- The PROCESS statement is intended for use only for views related to performing an activity;
- The FIND statement can be used to retrieve information, which may consist of a set of records.

PROCESS Statement

The PROCESS statement is used to request a function (for example, allocate, update, or delete system data, issue operator commands,) using the appropriate operating system service.

```
PROCESS view-name USING field-name=value
      GIVING field-name
```

Example:

```
PROCESS FILE-ALLOCATE USING
      DSNAME = 'USER.123',
      VOLSER = 'SAG001',
      NODE   = 151
      GIVING ERROR-CODE
```

Using the view FILE-ALLOCATE, a dataset named USER.123 is allocated on a volume with the serial number SAG001, on the machine identified by the Entire System Server node ID 151, and a code is returned.

FIND Statement

The FIND statement can be used to display data from views that require specification of fields. Selectable fields are indicated in the view descriptions by a **D** in the Descriptor column (see Section View Descriptions).

```
FIND ... view-name WITH ... search criteria
```

Data is selected based on the specified search criteria (see next subsection).

Example:

```

FIND ACTIVE-JOBS WITH NODE      = 151
                        AND JOB-NAME = COMPLETE
      . . . .
END-FIND
```

Using the ACTIVE-JOBS view, the job named COMPLETE on the machine identified by the Entire System Server node ID 151 is selected for display.

Search Criteria with the FIND Statement

When FIND is used to access the Entire System Server, search criteria can be used to specify values for alphanumeric fields. Two search criteria are available:

- * Acts as placeholder for one or more characters in the position;
- _ Acts as placeholder for one character.

The following examples demonstrate the use of these search criteria:

Example 1:

```

FIND VTOC WITH VOLSER = 'V3380A' AND DSNAME = 'L99*LOAD*'
```

All datasets on volume V3380A whose names start with L99 followed by anything, followed by LOAD, followed by anything, are selected; for example, datasets

- L99COM.LOAD.NPR,
- L99.SAG.LOAD.DOCS and
- L99NPROC.LOAD

Example 2:

```

FIND VTOC WITH VOLSER = 'V3380A' AND DSNAME = '*SOURCE'
```

All datasets on volume V3380A whose names end with SOURCE are selected; for example, datasets

- A1234.SOURCE
- SAG.PP.SOURCE and
- AB.MYSOURCE.

Example 3:

```

FIND ACTIVE-JOBS WITH JOB-NAME = 'L_ _AB*'
```

All jobs whose first characters are **L**, followed by any 2 characters, followed by **AB**, followed by anything are selected; for example, jobs

- L12ABJOB
- LAAABX and
- LXXAB2YC.

Example 4:

```
FIND NATPROC-USERS WITH USER-ID = ' _ _ _ _ _ '
```

All users of the Entire System Server whose identifiers contain exactly 5 characters are selected.

The FIND statement does not create an ISN list as in the case of Adabas. Therefore, all functions related to ISN lists are not supported. (for example, RETAIN, *NUMBER, etc.)

Using Entire System Server in a Multi-Computer Environment

If you have more than one computer at your installation, you may have more than one Entire System Server node installed (ask your system programmer). Note that the node ID identifies each Entire System Server node uniquely.

It is possible to direct a Entire System Server request from a Natural program to a specific node. To do this, specify NODE=*nnn* in the appropriate FIND statement. For example, the statement:

```
FIND VTOC WITH VOLSER='DISK01' AND NODE=151
```

is executed in Node 151. If the NODE field is not specified, the default node is used (DBID specified in the DDM).

An alternative method is to use the NODE-NAME field when referencing Entire System Server nodes on other machines. NODE-NAME is a character field and allows programs to be written without regard to a specific node number. If a particular machine needs to have its node changed, the only update that is required is to the mapping module ESYNODTB (see also Entire System Server Installation Documentation). No Natural programs need to be changed and restowed.

An example of NODE-NAME is

```
FIND VTOC WITH VOLSER='SMS236' AND NODE-NAME = 'PROD'
```

If both NODE and NODE-NAME are specified, the NODE specification takes precedence.

A Natural program can even access multiple nodes. For example, using the COPY-FILE view, you can copy a file from one node to another.

Using Entire System Server Update View Processors

General

Most of the views provided by ESY obtain data from the Operating System and return these data to the Natural program. Those views belong to the group of retrieval views. Another group of views allows you to modify Operating System objects in a certain way. These views belong to the group of update views.

All multi-record views with an ADABAS file number greater than or equal to 200 belong to update views. These views require a special programming technique. A number of other update views support a single record request only (eg. CATALOG-UPDATE, VTOC-UPDATE) and do not need special programming considerations.

The field FUNCTION is provided in all update views. It should contain blanks while creating the object, and the value CLOSE if the object has been properly created. If no CLOSE has been requested, the object is still in open state and not completely built.

You are recommended to use the Natural statement PROCESS to request update view services.

```
PROCESS update_view USING
      NODE = #NODE
      , FUNCTION = #FUNCTION
      ...
```

A Session with Multiple PROCESS Statements

If more than one PROCESS statement is implemented in a Natural program and the requests are related to one session only, e.g., to create a single dataset, the PROCESS statements must be indicated as belonging together.

A typical example is the **PROCESS WRITE-FILE USING FUNCTION=' '** in one subroutine to create several records and **PROCESS WRITE-FILE USING FUNCTION='CLOSE'** in a different subroutine. The field IDENTIFIER must be used and filled with the same 8-byte character string to indicate a session dealing with the same dataset in different locations of a Natural program.

Another issue is implementing nested loops requesting update view services. A separate IDENTIFIER must be used in every loop level to make the calls linked to several sessions if for example WRITE-FILE is used in different loop levels. If no IDENTIFIER is provided, unpredictable results might occur in nested loops.

Segmenting Data

If records have to be written to a dataset, the view WRITE-FILE must be used to do it. The field RECORD is defined as an alphanumeric field with a maximum length of 253 bytes only. Datasets probably contain records larger than 253 bytes. Therefore, the pieces of such records have to be delivered in segments.

Setting fields SEGMENT-NUMBER and SEGMENT-LENGTH allows you to create records longer than 253 bytes. Assuming a record length of 500 bytes, two view calls are needed.

SEGMENT-NUMBER=1, SEGMENT-LENGTH=250, RECORD bytes 1-250 filled with data, create the first part of the record, SEGMENT-NUMBER=2, SEGMENT-LENGTH=250, RECORD bytes 1-250 the second part of the record.

Smaller segments could also be used (e.g., 5 segments each 100 bytes long) but this would increase the number of calls and reduce the performance.

Sample WRITE-FILE Program

The following sample program reads an LMS element on one node and copies the data to another LMS element on another node. It deals with segments returned by the view READ-FILE. If an error occurs, the copying is stopped immediately.



```

DEFINE DATA LOCAL
1 READ-FILE VIEW OF RECORDS IN 0
2 ERROR-TEXT READ-FILE.ERROR-TEXT
2 ERROR-CODE READ-FILE.ERROR-CODE
2 SYSTEM-CODE READ-FILE.SYSTEM-CODE
2 SYSTEM-MESSAGE-TEXT READ-FILE.SYSTEM-MESSAGE-CODE
2 DSNAME #CLOSE-NEEDED EQ FALSE
2 ELEMENT ESCAPE ROUTINE
2 ELEMENT-TYPE
2 ELEMENT-VERSION
*2 RECORD
2 RECORD-LENGTH END-OF-FILE EQ 'YES' OR
2 RECORD-LENGTH ERROR-CODE NE 0
2 SEGMENT-LENGTH FUNCTION = 'CLOSE'
2 SEGMENT-NUMBER
*2 END-OF-FILE
2 PROGRAM WRITE-FILE USING
2 KEY NODE = #O-NODE
1 WRITE-FILE VIEW OF WRITABLE FILE = #O-DSNAME
2 ERROR-CODE , DISP = #O-DISP
2 ERROR-TEXT , FUNCTION = #O-FUNCTION
2 SYSTEM-CODE , PRODUCT = #O-PRODUCT
2 SYSTEM-MESSAGE-CODE ELEMENT = #O-ELEMENT
2 DSNAME , ELEMENT-TYPE = READ-FILE.ELEMENT-TYPE
2 ELEMENT , ELEMENT-VERSION = READ-FILE.ELEMENT-VERSION
2 ELEMENT-TYPE , RECORD = READ-FILE.RECORD
2 ELEMENT-VERSION , RECORD-LENGTH = READ-FILE.RECORD-LENGTH
2 RECORD , RECORD-NUMBER = READ-FILE.RECORD-NUMBER
2 RECORD-LENGTH , SEGMENT-LENGTH = READ-FILE.SEGMENT-LENGTH
2 RECORD-NUMBER , SEGMENT-NUMBER = READ-FILE.SEGMENT-NUMBER
2 SEGMENT-LENGTH , KEY = READ-FILE.KEY
*2 SEGMENT-NUMBER
2 PROGRAM WRITE-FILE.ERROR-CODE NE 0
2 DISWRITE WRITE-FILE.ERROR-CODE
2 KEY WRITE-FILE.ERROR-TEXT
2 FUNCTION WRITE-FILE.SYSTEM-CODE
* WRITE-FILE.SYSTEM-MESSAGE-CODE
1 #I-DSNAME ESCAPE ROUTINE INIT <'$NPR.NPR311.DEV'>
1 #ELEMENT (A64) INIT <'XCOMMMAIN'>
1 #I-ELEMENT-TYPE (A8) INIT <'P'>
1 #I-READ-FILE.ERROR-CODE CODE LINE 13
1 #I-ESCAPE ROUTINE (R1) INIT <'M'>
* END-IF
1 #O-DISP (A3) INIT <'NEW'>
1 #ASSIGN #CLOSE-NEEDED IN TRUE $PRD.NPR311.DEV'>
1 #O-FUNCTION (A8) INIT <' ' >
END OF ELEMENT (A64) INIT <'XCOMMMAIN'>
1 #O-NODE (N3) INIT <-114>
END OF PRODUCT (A1) INIT <'M'>
*
1 #CLOSE-NEEDED (L) INIT <FALSE>
*
END-DEFINE
*
* Main loop reading the segments of the input file to
* write the data to output on the target node.
*
FIND READ-FILE WITH NODE = #I-NODE
AND DSNAME = #I-DSNAME
AND ELEMENT = #I-ELEMENT
AND ELEMENT-TYPE = #I-ELEMENT-TYPE
AND PRODUCT = #I-PRODUCT

```

Examples

Example programs and their results are shown below for each of the functional areas of Entire System Server. The programs are taken from the Entire System Server online tutorial. A full list of field names for the various operating systems is contained in the view descriptions in Section View Descriptions.

Note:

Programs whose names start with **M** are taken from an OS/390 environment, those beginning with **D** from a VSE/ESA environment, and those beginning with **B** from BS2000/OSD.

File Management: COPY-FILE

Program MCOPIFYI uses the COPY-FILE view to copy files from one node to another within a computer network:

```

* Program      MCOPIFYI
* View        COPY-FILE
*
* Function     Copy files from node to node
*
* -----
*
DEFINE DATA
  GLOBAL USING TUTO
  LOCAL USING COPYFI-L
END-DEFINE
*
REPEAT
  INPUT (AD=MI'_' ZP=OFF)
          // ##TITLE (AD=OI IP = OFF)
          // ' from' (I)
          // ' Dataset...:' COPY-FILE.FROM-DSNAME
          // ' Member...:' COPY-FILE.FROM-MEMBER
          // ' Volser...:' COPY-FILE.FROM-VOLSER
          // ' Node.....:' COPY-FILE.FROM-NODE
          // ' to' (I)
          // ' Dataset...:' COPY-FILE.TO-DSNAME
          // ' Member...:' COPY-FILE.TO-MEMBER
          // ' Volser...:' COPY-FILE.TO-VOLSER
          // ' Node.....:' COPY-FILE.TO-NODE
  PROCESS COPY-FILE USING FROM-DSNAME = COPY-FILE.FROM-DSNAME
                        , FROM-MEMBER = COPY-FILE.FROM-MEMBER
                        , FROM-VOLSER = COPY-FILE.FROM-VOLSER
                        , FROM-NODE   = COPY-FILE.FROM-NODE
                        , TO-DSNAME  = COPY-FILE.TO-DSNAME
                        , TO-MEMBER  = COPY-FILE.TO-MEMBER
                        , TO-VOLSER  = COPY-FILE.TO-VOLSER
                        , TO-NODE    = COPY-FILE.TO-NODE
                        , NODE       = ##NODE
  REINPUT ERROR-TEXT
*
END-REPEAT
END

```

This program prompts you for specification of the source and destination names, and notifies you of the successful copy function with a message:

ESY5000 COPY COMPLETED SUCCESSFULLY

System Maintenance: NET-OPER

Program MNETOPR executes certain operator commands and displays system response:

```

* Program      MNETOPR
* View        NET-OPER
*
* Function     Execute NET operator commands and display response
*
* -----
*
DEFINE DATA
  GLOBAL USING TUTO
  LOCAL  USING NETOP-L
END-DEFINE
*
REPEAT
  INPUT (AD=MI'_' )
  // ##TITLE (AD=OI IP=OFF)
  // 'Command:' / ' ' NET-OPER.COMMAND (AL=79)
  // 'Purge previous messages ?' NET-OPER.PURGE-PREVIOUS '(y/n)'
*
  FIND NET-OPER WITH COMMAND          = NET-OPER.COMMAND
                        AND NODE        = ##NODE
                        AND PURGE-PREVIOUS = NET-OPER.PURGE-PREVIOUS
*
  IF ERROR-CODE > 0
    ASSIGN ##MSG-NR = 1000
    ASSIGN ##MSG-TXT1 = ERROR-TEXT
    STOP
  END-IF
*
  IF LINE-STATUS NE 'YES'
    NEWPAGE
  END-IF
*
  WRITE NOTITLE TIME-STAMP LINE (AL=70)
*
  END-FIND
END-REPEAT
END

```

Example output from the program MNETOPR using input "D NET,LINES":

```

09:02:12 D NET,LINES
09:01:38 IST097I DISPLAY ACCEPTED
09:01:38 IST350I VTAM DISPLAY - DOMAIN TYPE= LINES
09:01:38 IST354I PU T4/5 MAJOR NODE = ISTPUS
09:01:38 IST170I LINES:
09:01:38 IST080I 050-L ACTIV----I 052-L ACTIV----I
09:01:38 IST231I CA MAJOR NODE = FCHAN
09:01:38 IST170I LINES:
09:01:38 IST232I FACAL , ACTIV----E, CUA = 930
09:01:38 IST354I PU T4/5 MAJOR NODE = NCPF00
09:01:38 IST170I LINES:
09:01:38 IST080I NATL1 RESET-N--- SIML1 RESET-N--- BRUL1 RESET-N---
09:01:38 IST080I BERL1 RESET-N--- HANL1 RESET-N--- AMSL1 RESET-N---
09:01:38 IST080I MUEL2 RESET-N--- STUL2 RESET-N--- STUL3 RESET-N---
09:01:38 IST080I DEML1 RESET-N--- HAML2 RESET-N--- NIKL1 RESET-N---
09:01:38 IST080I NIKL3 RESET-N--- NIKL4 RESET-N--- HAML1 RESET-N---
09:01:38 IST080I STUL1 RESET-N--- WIEL1 RESET-N--- DEML2 RESET-N---
09:01:38 IST080I LNKRESL ACTIV----E
09:01:38 IST354I PU T4/5 MAJOR NODE = NCPE01
09:01:38 IST170I LINES:
09:01:38 IST080I VXEL1 RESET-N--- NIKL5 RESET-N--- NIKL6 RESET-N---
09:01:38 IST080I NETL1 RESET-N--- NUEL1 RESET-N--- FRIL1 RESET-N---
09:01:38 IST080I FRIL2 RESET-N--- FRIL3 RESET-N--- FRIL4 RESET-N---

```

System Maintenance: CONSOLE

Program MCONSOL displays the operator console:

```

* Program   MCONSOL
* View     CONSOLE
*
* Function  Operator Console
*
* -----
*
DEFINE DATA
  GLOBAL USING TUTO
  LOCAL USING CONSOLEL
END-DEFINE
*
SET KEY PF12 NAMED 'Node'
*
REPEAT
  RESET #LINE (*)
  RESET #CV-LINE (*)
  FIND CONSOLE WITH NODE      = ##NODE
                        AND FUNCTION = 'DISPLAY'
    PERFORM CHECK-ERROR
    ASSIGN #LINE (20) = TEXT
    ASSIGN #LINE (1:19) = #LINE(2:20)
  END-FIND
*
  FIND CONSOLE WITH NODE      = ##NODE
                        AND FUNCTION = 'DIS-WTOR'
    PERFORM CHECK-ERROR
    ASSIGN #I = *COUNTER
    ASSIGN #LINE (#I) = TEXT
    ASSIGN #CV-LINE (#I) = (AD=I)
  END-FIND
*
  RESET #LINE (20)
  #CV-LINE (20)
  FOR #I 1 19
    FOR #J = #I 20
      IF #LINE (#I) = ' '
        ASSIGN #LINE (#I:19) = #LINE (#I+1:20)
        ASSIGN #CV-LINE (#I:19) = #CV-LINE (#I+1:20)
        RESET #LINE (20)
        #CV-LINE (20)
      ELSE
        ESCAPE BOTTOM
      END-IF
    END-FOR
  END-FOR
*
  PERFORM SCREEN-IO
*
END-REPEAT
*

```

```

DEFINE SUBROUTINE SCREEN-IO
  INPUT WITH TEXT *##MSG-NR,##MSG-TXT1, ##MSG-TXT2
  USING MAP 'CONSOLE&'
  RESET ##MSG-NR ##MSG-TXT1 ##MSG-TXT2
  IF #COMMAND-LINE NE ' '
    ASSIGN #FUNCTION = 'OP-CMD'
    PERFORM ISSUE-OPERATOR-COMMAND
    RESET #COMMAND-LINE
  END-IF
  IF *PF-KEY = 'PF12'
    CALLNAT 'TUTODB' ##NODE ##MSG ##TUTO
  END-IF
END-SUBROUTINE
*
DEFINE SUBROUTINE ISSUE-OPERATOR-COMMAND
  PROCESS CONSOLE USING NODE      = ##NODE
                        , FUNCTION = ##FUNCTION
                        , TEXT     = #COMMAND-LINE
                        GIVING ERROR-CODE ERROR-TEXT
  PERFORM CHECK-ERROR
END-SUBROUTINE
*
DEFINE SUBROUTINE CHECK-ERROR
  IF CONSOLE.ERROR-CODE > 0
    ASSIGN ##MSG-TXT1 = CONSOLE.ERROR-TEXT
    ASSIGN ##MSG-NR   = 1000
  END-IF
END-SUBROUTINE
*
END

```

Output from the program MCONSOL:

```

----- Operator-Console ----- Node 148
- STC 2141 NET0120 - VTAM LINK LNKA      TO NODE ANODE      STAT=ACTIVE
- STC 2141 NET0120 - VTAM LINK LNKVM     TO NODE UNKNOWN   STAT=OPEN
- STC 2141 NET0120 - VTAM LINK LNKKOP   TO NODE UNKNOWN   STAT=OPEN
- JOB 2344 IEF404I IMSGEN05 - ENDED - TIME=10.22.44
- JOB 2345 IEF403I IMSGEN06 - STARTED - TIME=10.22.45
  STC 2112 F FNETWK,CONN LNKU           --> UQ K CMD FROM HRO
- STC 2141 NET0137 - LINK LNKU          CONNECT INITIATED
- STC 2141 NET0137 - LINK LNKR          CONNECT INITIATED
  STC 2124 IST663I CDINIT REQUEST TO EHOST FAILED, SENSE=08010000
  IST664I REAL OLU=SAGNET.FNETWK        ALIAS DLU=SAGNET.UNETWK
  IST889I SID = CB6722CE854CF71C
  IST314I END
  STC 2124 IST663I CDINIT REQUEST TO EGAT FAILED, SENSE=08010000
  IST664I REAL OLU=SAGNET.FNETWK        ALIAS DLU=SAGNET.RNETWK
  IST889I SID = CB6722CE854CF71E
  IST314I END
  STC 2112 P NPR123                      --> UQ K CMD FROM WKK
  STC 2112 IEE341I NPR123      NOT ACTIVE
00- STC 2231 === INACTIVE USER HAL        HAS BEEN PURGED ===

```

Spool Management: READ-SPOOL

Program MRSPPOOL displays Information for a specified job:

```

* Program      MRSPool
* View        READ-SPOOL
*
* Function    Read SYSOUT records from JES
*
* -----
*
DEFINE DATA
  GLOBAL USING TUTO
  LOCAL USING RSPool-L
  LOCAL 1 #JOBN      (N5)
        1 #JOB       (A8)
        1 #TYPE     (A2)
        1 #DS       (N3)
        1 #STRING   (A50)
        1 #SEL      (A1/1:6)
        1 #CODE     (A2/1:6) CONST (1) <'SI'>
                                   (2) <'JL'>
                                   (3) <'SM'>
                                   (4) <'SO'>
                                   (5) <'CC'>
                                   (6) <'AL'>

        1 #I        (I1)
END-DEFINE
*
REPEAT
  INPUT WITH TEXT *##MSG-NR,
                ##MSG-TXT1,
                ##MSG-TXT2
        USING MAP 'MRSPool&'
  RESET ##MSG
*
  IF #STRING = ' '
    ASSIGN #STRING = '*'
  END-IF
*
  IF SELECTION NOT UNIQUE #SEL (*)
    REINPUT *1011
  ELSE
    FOR #I = 1 TO 6
      IF #SEL (#I) NE ' '
        ASSIGN #TYPE = #CODE (#I)
      END-IF
    END-FOR
  END-IF
*

```

```

FIND READ-SPOOL WITH JOB-NAME    = #JOB
                        AND JOB-NUMBER = #JOBN
                        AND TYPE      = #TYPE
                        AND DATA-SET  = #DS
                        AND NODE      = ##NODE
                        AND RECORD     = #STRING
      IF ERROR-CODE > 0
        REINPUT ERROR-TEXT
      END-IF
      DISPLAY NOTITLE NOHDR RECORD (AL=79)
    END-FIND
  END-REPEAT
*
END

```

Output from the program MRSPOOL:

The program displays the following output with all datasets specified in the prompt:

```

1                J E S 2  J O B  L O G  --  S Y S T E M  D A E F  --  N  -----  JOB
5812 IEF097I OPPLG181 - USER ACF2BAT ASSIGNED
14.03.58 JOB 5812 $HASP373 OPPLG181 STARTED - INIT 22 - CLASS W - SYS DAEF 14.03.58 JOB 5812
IEF403I OPPLG181 - STARTED - TIME=14.03.58
14.03.58 JOB 5812 *IEF233A M 803,P18111,,OPPLG181,RES518,DB181.PLOG07 14.10.51 JOB 5812
-
--TIMINGS
14.10.51 JOB 5812 -JOBNAME STEPNAME PROCSTEP RC EXCP CONN TCB S
14.10.51 JOB 5812 -OPPLG181 PLG181 RES518 00 7122 22575 .03 .
14.10.54 JOB 5812 $DRM007 WARNING - DRM DATABASE IS ACTIVE
14.11.03 JOB 5812 +ADAN02 00008 NUCLEUS-RUN WITHOUT PROTECTION-LOG
14.11.03 JOB 5812 +ADAN03 00008 ADABAS COMING UP
14.11.03 JOB 5812 +ADAN01 00008 A D A B A S IS ACTIVE
14.11.03 JOB 5812 +ADAN01 00008 MODE = SINGLE I S O L A T E D
14.11.05 JOB 5812 $DRM200 SDR PLG181 updated successfully.
14.11.05 JOB 5812 +DRM201 Tsn P18111
14.11.05 JOB 5812 +DRM202 Seq 1 Label 7 File 7 Used 53 Writes 0
14.11.05 JOB 5812 +ADAL01 00008 93.02.05 14:11:04 CLOG NOT ACTIVE
14.11.11 JOB 5812 -OPPLG181 DRMUPD 00 320 1034 .00 .
14.11.11 JOB 5812 IEF234E K 803,P18111,PVT,OPPLG181
14.11.11 JOB 5812 IEF404I OPPLG181 - ENDED - TIME=14.11.11
14.11.11 JOB 5812 -OPPLG181 ENDED. NAME- TOTAL TCB CPU T
14.11.11 JOB 5812 $HASP395 OPPLG181 ENDED
0----- JES2 JOB STATISTICS -----

```

Spool Management: JOB-SWITCHES

Program BSW allows the user to specify certain items in job switch handling:

Output from the program BSW:

```

                                BS2000/OSD Job switches
                                Function
..... READ_____ (READ/WRITE/ATTRIB)
Operation ..... _____ (INVERT/ON/OFF)
Switch-Type ..... USER_____ (USER/PROCESS)
BS2000/OSD-UserID ..... NPR_____

Switch Status      Switch Status      Switch Status      Switch Status
- - - - -
-   0   off        -   1   off        -   2   off        -   3   on
-   4   off        -   5   off        -   6   off        -   7   off
-   8   off        -   9   off        -  10   off        -  11   off
-  12   off        -  13   off        -  14   off        -  15   off
-  16   off        -  17   off        -  18   off        -  19   off
-  20   off        -  21   off        -  22   off        -  23   off
-  24   off        -  25   off        -  26   off        -  27   off
-  28   off        -  29   off        -  30   off        -  31   off

Please mark specific Switch(es) with 'X'

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
                                Exit
    
```

Disc and Catalog Management: VTOC

Program DVTOC lists catalog entries for a specific volume:

```

* List VTOC of a specific volume
*
DEFINE DATA
  GLOBAL USING TUTO
  LOCAL USING DVTOCL
END-DEFINE
*
ASSIGN VTOC.EXTENT-TYPE = '*'
*
INPUT (AD=MIT)
  ##TITLE (AD=OI IP=OFF)
  // 'List Vtoc of..:' VTOC.VOLSER
  / 'Extent type...:' VTOC.EXTENT-TYPE 3X '(* /USED /FREE) '
*
IF NOT VTOC.EXTENT-TYPE = 'USED' OR = 'FREE' OR = '*'
  REINPUT *1027 MARK *VTOC.EXTENT-TYPE
END-IF
*
WRITE TITLE ##TITLE (AD=I) /
           'Vtoc.....' (I) VTOC.VOLSER (AD=I) /
*
FIND VTOC WITH VOLSER      = VTOC.VOLSER
           AND  EXTENT-TYPE = VTOC.EXTENT-TYPE
           AND  NODE       = ##NODE
*
  IF ERROR-CODE > 0
    REINPUT ERROR-TEXT
  END-IF
*
END-ALL
SORT BY DSNAME USING TOTAL-TRACKS-ALLOCATED DSORG EXTENTS CREATION-DATE
*
  DISPLAY 'Name'      DSNAME (AL=30 IS=ON)
           'Tracks'   TOTAL-TRACKS-ALLOCATED
           'Dsorg'    DSORG
           'Extent'   EXTENTS (EM=H(12))
           'Created'  CREATION-DATE
END-SORT
*
END

```

Output of the program DVTOC for the volume SYSWK1:

```

MORE
                                List VTOC of a specific volume
                                Vtoc..... SYSWK1

```

Name	Tracks	Dsorg	Extent	Created

** VTOC EXTENT **	15	UN	013D0000013D000E00010000	*****
CICS.SYSTEM.LOG.A.TFS	45	SD	01BF000001C1000E00030000	29/06/90
CICS.SYSTEM.LOG.B.TFS	45	SD	01C2000001C4000E00030000	29/06/90
COM441.VSE.HISTORY	30	UN	000100000002000E00020000	20/03/91
DOS.LABEL.FILE.FF0000203081.AR	45	UN	024A0000024C000E00030000	16/06/89
ICCF.LIBRARY	1785	DA	013E000001B4000E00770000	04/12/87
INFO.ANALYSIS.DUMP.MGNT.FILE	10	SD	02480000024800090000000A	05/06/90
INFO.ANALYSIS.EXT.RTNS.FILE	5	SD	0248000A0248000E00000005	04/12/87
RMADA.DB001.DATAR1	855	DA	00030000003B000E00390000	20/10/89
VSE.DUMP.LIBRARY	600	SD	00D4000000FB000E00280000	04/12/87
VSE.HARDCOPY.FILE	30	UN	023E0000023F000E00020000	04/12/87
VSE.HARDCOPY.FILE.JW	30	UN	024D0000024E000E00020000	15/08/89
VSE.HARDCOPY.FILE.SHR2	30	UN	01B5000001B6000E00020000	04/11/88
VSE.HARDCOPY.FILE.TFS	30	UN	01BA000001BB000E00020000	09/11/88
VSE.POWER.ACCOUNT.FILE	60	DA	02390000023C000E00040000	11/11/88
VSE.POWER.DATA.FILE	1740	DA	01C500000238000E00740000	18/11/88
VSE.RECORDER.FILE	45	UN	024000000242000E00030000	04/12/87