

Entire System Server At Work - Examples

This section covers the following topics:

- General Information
 - IEBCOPY Utility
 - Disc Maintenance
 - File Maintenance
 - Job Handling
 - Spool File Handling
 - Imagination Is the Limit
-

General Information

The principle of operation behind the Entire System Server is surprisingly simple.

Just as a Natural program would access conventional data, Entire System Server views can be called from a Natural program using the Natural statements Process or Find, depending on the view involved. Just as a database identifier is required for a standard call to a database, the call to the Entire System Server is identified by a node number.

The Entire System Server recognizes the node number and processes the call, returning the requested operating system service or information to the program. Each operating system in the computer network is identified by the node number, thus enabling access to any system from anywhere within the network.

This section illustrates the use of the Entire System Server with some example program coding. The examples illustrate how simple, easy-to-write Natural programs can be used to display system information on the terminal screen, requiring only a minimum of input from the user. However, this is not the only use of the Entire System Server.

The examples should be seen as a starting point for more powerful applications that can process system information automatically, invisible to the user.

IEBCOPY Utility

In OS/390 systems, the IEBCOPY utility can be invoked using only a few lines of code in a Natural program:

```
*
* The INPUT statement is a standard Natural statement which
* defines a map that is displayed on the terminal screen when the
* program is run. The map prompts the user to specify the source * member to be
* copied and the destination member:
*
  INPUT      // ' Input Dsname ...:' IEBCOPY.IN-DSNAME
             / ' Volser .....:' IEBCOPY.IN-VOLSER
             // ' Output Dsname ...:' IEBCOPY.OUT-DSNAME
             / ' Volser .....:' IEBCOPY.OUT-VOLSER
             // ' Member .....:' IEBCOPY.IN-MEMBER
             / ' New Name .....:' IEBCOPY.OUT-MEMBER
             // ' Replace .....:' IEBCOPY.REPLACE '(yes/no) '
  ....
```



```

                                IEBCOPY utility

Input Dsname ...: MY.OLD.DATASET_____
Volser .....: _____

Output Dsname ..: MY.NEW.DATASET_____
Volser .....: _____

Member .....: OLDNAME_____
New Name .....: NEWNAME_____

Replace .....: ____ (yes/no)

```

By running such a simple Natural program, operating system utilities can be used without any special knowledge of utility-specific syntax on the part of the user.

Disc Maintenance

- Example 1
- Example 2

Example 1:

The following example program allows the user to perform certain disk maintenance functions in any operating system environment:

```

....
* When this program is run, the user is presented with the map
* defined by the following INPUT statement. Possible functions to * maintain a
catalog entry are: RENAME, SCRATCH, PURGE:
*
  INPUT // ' Function ...:' VTOC-UPDATE.FUNCTION
        // ' Volume ....:' VTOC-UPDATE.VOLSER
        /  ' Dsname ....:' VTOC-UPDATE.DSNAME
        // ' New Name ..:' VTOC-UPDATE.NEWNAME
....
*
* The disc is accessed by addressing the corresponding fields on * the view
VTOC-UPDATE using the PROCESS statement. Within a
* multi-CPU environment, the NODE variable allows the program to * access the
disc on another computer:
*
  PROCESS VTOC-UPDATE USING NODE      = ##NODE
                                ,      DSNAME   = VTOC-UPDATE.DSNAME
                                ,      VOLSER   = VTOC-UPDATE.VOLSER
                                ,      FUNCTION = VTOC-UPDATE.FUNCTION
                                ,      NEWNAME  = VTOC-UPDATE.NEWNAME

```

Again, this example shows that no special knowledge of operating system structures is required to write and use such a program; indeed, the same program can be used in different operating systems.

Example 2:

The following example shows how a Natural program can retrieve storage unit information and display it to the user. This example is taken from an OS/390 environment:

```

...
*
* The FIND statement addresses the view UNIT-ATTRIBUTES to read
* the desired device. The NODE parameter is used when reading the
* information from a different machine within the computer
* network:
*
      FIND UNIT-ATTRIBUTES WITH CLASS = 'DASD'
                          AND NODE = ##NODE
...
*
* Using the DISPLAY statement, the output is presented to the
* user when the program is run:
*
      DISPLAY (ZP=OFF)
        'Unit/Adr'           UNIT-ATTRIBUTES.UNIT
        'Unit/Type'          SERIES
        'avail./Unit'        DEVICE-STATUS
        'OPEN/DCB' 'S'       DCB-COUNT
        5X
        'mounted/Volume'     VOLSER
        'mount/Attributes'   MOUNT-STATUS
        'avail./of Volume'   VOLUME-STATUS
        'FREE/CYL'           FREE-CYLINDERS
        'FREE/TRACKS'        FREE-TRACKS
        'FREE/EXTENTS'       FREE-EXTENTS
      ADD 1 TO #NUMBER
      END-FIND
...

```

Running the program with the above code produces output similar to the following:

Unit Adr	Unit Type	avail. Unit	OPEN DCB'S	mounted Volume	mount Attributes	avail. of Volume	FREE CYL	FREE TRACKS	FREE EXTENTS
300	3380	ONLINE	102	SYSF06	RESIDENT	PRIVATE	367	50	20
310	3380	ONLINE	9	NAT002	RESIDENT	PRIVATE	70	138	48
320	3380	ONLINE	5	NDM001	RESIDENT	PRIVATE	96	13	16
330	3380	ONLINE	65	XKGS01	RESIDENT	PRIVATE	293	280	58
350	3380	ONLINE	21	USR8A6	RESIDENT	STORAGE	140	1080	236
360	3380	ONLINE	7	DCN002	RESIDENT	PRIVATE		131	19
370	3380	ONLINE	55	DBDC06	RESIDENT	PRIVATE	1018	63	19
380	3380	ONLINE	6	EUP003	RESIDENT	PRIVATE	29	141	23

File Maintenance

- Example 1
- Example 2
- Example 3
- Example 4
- Example 5
- Example 6

The file maintenance group of views allow users to allocate files, display file information, and rename, delete or copy files using small Natural programs.

Example 1:

The following example works in an OS/390 and BS2000/OSD environment. The program prompts the user for the name of a dataset to be compressed (in BS2000/OSD terms, unused space to be released).

```
.....
*
* The prompt is defined using the INPUT statement, allowing the
* user to specify the dataset to be compressed:
*
INPUT          // ' Dataset ....:' FILE-MAINTENANCE.DSNAME
               / ' Volume ....:' FILE-MAINTENANCE.VOLSER
*
* Compression is performed by addressing the FILE-MAINTENANCE
* view using the PROCESS statement.
* The NODE parameter must be used
* when compressing a dataset that resides on a different node within
* the computer network:
*
PROCESS FILE-MAINTENANCE USING FUNCTION='COMPRESS'
,
,          DSNAME = FILE-MAINTENANCE.DSNAME
,          VOLSER = FILE-MAINTENANCE.VOLSER
,          NODE   = ##NODE
.....
```

Libraries can thus be maintained easily using only a few lines of Natural code.

Example 2:

The following example shows how a simple Natural program can perform a file transfer operation from one VSE/ESA system to another in a network. Note that in this example, up to three Entire System Server nodes are involved: the program runs on one machine, but can copy a file residing on a second machine to a third machine within the computer network.

```

*
* The INPUT statement defines an input mask to be displayed
* when the program is run, in which the user can specify the
* source and target datasets.
* The NODE parameter specifies the node, if
* different from the node on which the program runs.
*
  INPUT          // '   Dataset...:' COPY-FILE.FROM-DSNAME
                / '   Sublib...:'  COPY-FILE.FROM-SUB-LIBRARY
                / '   Member typ:' COPY-FILE.FROM-MEMBER-TYPE
                / '   Member...:'  COPY-FILE.FROM-MEMBER
                / '   Volser...:'  COPY-FILE.FROM-VOLSER
                / '   Node.....:'  COPY-FILE.FROM-NODE
                // ' to' (I)
                // '   Dataset...:' COPY-FILE.TO-DSNAME
                / '   Sublib...:'  COPY-FILE.TO-SUB-LIBRARY
                / '   Member typ:' COPY-FILE.TO-MEMBER-TYPE
                / '   Member...:'  COPY-FILE.TO-MEMBER
                / '   Volser...:'  COPY-FILE.TO-VOLSER
                / '   Node.....:'  COPY-FILE.TO-NODE
                / '   Replace...:' #REPLACE

*
* The copy operation is performed by the PROCESS call to the
* COPY-FILE view, specifying the source and target dataset
* characteristics:
*
  PROCESS COPY-FILE USING FROM-DSNAME = COPY-FILE.FROM-DSNAME
                        , FROM-SUB-LIBRARY = COPY-FILE.FROM-SUB-LIBRARY
                        , FROM-MEMBER-TYPE = COPY-FILE.FROM-MEMBER-TYPE
                        , 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-SUB-LIBRARY = COPY-FILE.TO-SUB-LIBRARY
                        , TO-MEMBER-TYPE = COPY-FILE.TO-MEMBER-TYPE
                        , TO-MEMBER = COPY-FILE.TO-MEMBER
                        , TO-VOLSER = COPY-FILE.TO-VOLSER
                        , TO-NODE = COPY-FILE.TO-NODE
                        , NODE = ##NODE
                        , REPLACE = #REPLACE '(Yes/No)'

```

Example 3:

The following example shows how a simple Natural program can be used to perform a file transfer operation from an OS/390 to a VSE/ESA node in a network:

```

...
*
* The INPUT statement defines an input mask to be displayed
* when the program is run, in which the user can specify the
* source and target datasets.
*
  INPUT          // ##TITLE (AD=OI IP = OFF)
                 // '   Dataset....:' COPY-FILE.IN-DSNAME
                 / '   Member....:' COPY-FILE.IN-MEMBER
                 / '   Volser....:' COPY-FILE.IN-VOLSER
                 / '   Node.....:' COPY-FILE.IN-NODE
                 // ' to' (I)
                 // '   Dataset....:' COPY-FILE.OUT-DSNAME
                 / '   Sublib....:' COPY-FILE.OUT-SUB-LIBRARY
                 / '   Member typ:' COPY-FILE.OUT-MEMBER-TYPE
                 / '   Member....:' COPY-FILE.OUT-MEMBER
                 / '   Volser....:' COPY-FILE.OUT-VOLSER
                 / '   Node.....:' COPY-FILE.OUT-NODE
                 // '   Replace....:' #REPLACE

.....
*
* The copy operation is performed by the PROCESS call to the
* COPY-FILE view, specifying the source and target dataset
* characteristics. The different operating systems involved in
* the copy operation are identified by the node number:
*
  PROCESS COPY-FILE USING IN-DSNAME = COPY-FILE.IN-DSNAME
                        ,          IN-MEMBER = COPY-FILE.IN-MEMBER
                        ,          IN-VOLSER = COPY-FILE.IN-VOLSER
                        ,          IN-NODE   = COPY-FILE.IN-NODE
                        ,          OUT-DSNAME = COPY-FILE.OUT-DSNAME
                        ,          OUT-SUB-LIBRARY = COPY-FILE.OUT-SUB-LIBRARY
                        ,          OUT-MEMBER-TYPE = COPY-FILE.OUT-MEMBER-TYPE
                        ,          OUT-MEMBER = COPY-FILE.OUT-MEMBER
                        ,          OUT-VOLSER = COPY-FILE
                        ,          OUT-NODE   = COPY-FILE.OUT-NODE
                        ,          REPLACE   = #REPLACE
                        ,          NODE      = ##NODE

```

Comparing this example with the previous one, note how similar the syntax to perform the file transfer is, even though the second example involves a different operating system. No special system-specific knowledge is required on the part of the programmer. All required information is provided by the Entire System Server's logical view of the operating systems involved, and standard sample programs are easily and quickly modified to access specific system information and services in heterogeneous networks.

Example 4:

The following example program displays a library directory according to specified characteristics:

```

...
*
* The INPUT statement defines an input mask in which the user can
* specify member characteristics according to which the directory * is to be
* composed:
*
  INPUT // ' Dataset.....:' LIB-DIRECTORY.DSNAME
         / ' Element.....:' LIB-DIRECTORY.ELEMENT
         / ' -type.....:' LIB-DIRECTORY.ELEMENT-TYPE
         / ' -version.....:' LIB-DIRECTORY.ELEMENT-VERSION
.....
*
* The requested information is provided by the
* view LIB-DIRECTORY, called with the FIND statement.
* The node number specifies the operating system in the computer * network from
* which the information is to be read:
*
  FIND LIB-DIRECTORY WITH NODE = 31
      AND DSNAME                = LIB-DIRECTORY.DSNAME
      AND ELEMENT                = LIB-DIRECTORY.ELEMENT
      AND ELEMENT-TYPE          = LIB-DIRECTORY.ELEMENT-TYPE
      AND ELEMENT-VERSION       = LIB-DIRECTORY.ELEMENT-VERSION
.....
*
* The DISPLAY statement is used to present the desired
* information to the user at his terminal:
*
*
  DISPLAY LIB-DIRECTORY.ELEMENT-TYPE (AL=1)
          LIB-DIRECTORY.ELEMENT (AL=40)
          LIB-DIRECTORY.ELEMENT-VERSION (AL=10)
  END-FIND
.....

```

Example 5:

The following example illustrates the use of a Natural program to copy all files with a certain prefix and suffix as list elements (type P) to an LMS library in a BS2000/OSD system:

```

.....
*
* An INPUT statement defines the input mask in which the user
* specifies the target LMS library, as well as the search
* criteria prefix and suffix. The replace option is used to
* specify whether datasets of the same name in the target
* library are to be overwritten.
*
      INPUT (AD=MI'_' ) 'NAME OF LMS LIBRARY .....:' #LMS-LIB
                'PREFIX.....:' #PREFIX
                'SUFFIX.....:' #SUFFIX
                'REPLACE.....:' #REPLACE
.....
*
* The names of the datasets to be searched consist of three
* parts: the prefix, element, and suffix. For the search
* operation, they can be compressed into one string, where the
* wildcard symbol asterisk (*) selects any element:....
*
      COMPRESS #PREFIX '*' #SUFFIX INTO #DSNAME LEAVING NO SPACE
.....
*
* All the datasets matching the search criteria are selected using
* the CATALOG view:
*
      FIND CATALOG WITH NODE      = #NODE
                        AND DSNAME = #DSNAME
.....
*
* The parts of the dataset name, compressed into DSNAME, are now
* separated:
*
      MOVE CATALOG.DSNAME TO #DSNAME
      EXAMINE #DSNAME FOR #PREFIX AND REPLACE WITH '*'/* NOT IN DSNAME
      EXAMINE #DSNAME FOR #SUFFIX AND REPLACE WITH '*'
      SEPARATE #DSNAME INTO #DSNAME-PARTS(*) WITH DELIMITER '*'
      MOVE #DSNAME-PARTS(2) TO #ELEMENT
      MOVE CATALOG.DSNAME TO #DSNAME
.....
*
* The datasets can now be copied using the COPY-FILE view:
*
      PROCESS COPY-FILE USING NODE = #NODE
                , FROM-DSNAME      = #DSNAME
                , TO-DSNAME        = #LMS-LIB
                , TO-PRODUCT       = 'M'
                , TO-ELEMENT       = #ELEMENT
                , TO-ELEMENT-TYPE  = 'P'
                , REPLACE          = #REPLACE
      END-FIND
.....

```

Example 6:

The following example is taken from a simple Natural program to print a file.

```

.....
*
* When the program is run, the user is prompted for details of
* the dataset to be printed by the input mask defined by the INPUT
* statement:
*
INPUT /'NAME OF FILE TO PRINT.....: ' #FILENAME
      /'SPACE-PARAMETER.....: ' #CONTROL '(E OR BLANK)'
      / 'JOB-NAME.....: ' #JOB-NAME
      / 'DEVICE.....: ' #DEVICE
      / 'STARTNO.....: ' #STARTNO
      / 'ENDNO.....: ' #ENDNO
.....
*
* Printing is performed by the call to the WRITE-SPOOL view:
*
PROCESS WRITE-SPOOL USING NODE          = #NODE
                                , DSNAME = #FILENAME
                                , JOB-NAME = #JOB-NAME
                                , CONTROL = #CONTROL
                                , DEVICE = #DEVICE
                                , STARTNO = #STARTNO
                                , ENDNO = #ENDNO

```

Job Handling

- Example 1
- Example 2

The Entire System Server provides a number of views that allow users to retrieve system information from a Natural program. Views are available for display of address space, main storage, as well as active tasks.

Example 1:

The following lines of Natural code call the ACTIVE-JOBS view and specify the information items required for display. The example can be used in an OS/390, FACOM, VSE/ESA and BS2000/OSD system. The NODE parameter is used to access a different machine in the computer network.

```

FIND ACTIVE-JOBS WITH NODE          = ##NODE
                                AND JOB-NAME = #JOB-NAME
                                AND TYPE = #TYPE
                                AND CPU-USED = #CPU-USED
                                AND STATUS = #STATUS
.....
END-FIND

```

The resulting output from this program is presented in the following format in an OS/390 system:

Job-Name	Type	Status	Cpu used	Region	JobNr.	ProcName	StepName
*	*	*					
MASTER	STC	NON-SWAP	2226.21	268	3513		
PCAUTH	STC	NON-SWAP	0.01	164			PCAUTH
TRACE	STC	NON-SWAP	0.01	104			TRACE
GRS	STC	NON-SWAP	0.03	1016			GRS
DUMPSRV	STC	NON-SWAP	2.66	92		DUMPSRV	DUMPSRV
CONSOLE	STC	NON-SWAP	305.78	204			CONSOLE
ALLOCAS	STC	NON-SWAP	0.01	148			ALLOCAS
SMF	STC	NON-SWAP	6.57	152		IEFPROC	SMF
LLA	STC	NON-SWAP	1.69	332		LLA	LLA
ACF2	STC	NON-SWAP	1.61	172		IEFPROC	ACF2
JES2	STC	NON-SWAP	1607.18	936		JES2	JES2
RMF	STC	NON-SWAP	10.72	76	3525	IEFPROC	FRMF
TMON8DLS	STC	NON-SWAP	281.03	568	3202	TMON8DLS	TMON8DLS
TMDBDLS	STC	NON-SWAP	33.66	248	3122	TMDBDLS	TMDBDLS
TMONMVS	STC	NON-SWAP	79.22	152	4013	TMONMVS	TMONMVS

The same program produces the output in the following format in a BS2000/OSD system (all jobs beginning with "N" are displayed):

Job-Name	Type	JobNr.	Cpu used	Acct-Nr	Cpu-max	Sta-Typ
N*	*					
NATV21	BATCH		0.68	E	32767.00	2
NATV21	BATCH		0.67	E	32767.00	2
NATISPF	BATCH		0.85	1	32767.00	2
NAT220	BATCH		0.78	1	32767.00	2
NETWORK	TP		451.26	1	32767.00	2
NCL	BATCH		203.40	1	NTL	2

Example 2:

The following example illustrates the use of a Natural program to handle job variables in a BS2000/OSD environment.

```

.....
*
* The INPUT statement defines an input mask which is displayed on * the
terminal screen at run time, together with the options for * the FUNCTION
field:
*
      INPUT
      /'Name of Job-Variable: ' #JV-NAME
      /'Node : ' #NODE
      / 'Function (READ / WRITE / ALLOC / ERASE / END): ' #FUNCTION
// '          only for function WRITE:'
      / 'Date: ' #DATA (AL=20)
      / 'Substring-start: ' #SUBSTR-START (NL=3 SG=OFF)
      / '(Substring-)length: ' #SUBSTR-LENGTH (NL=3 SG=OFF)
      / 'Value-length: ' #VALUE-LENGTH (NL=3 SG=OFF)
      / 'Password, if required: ' #PASSWORD
.....
*
* The view JOB-VARIABLES can then be addressed by the program,
* the fields used depending on the specified function.
* Below are examples for READ and WRITE:
*
      VALUE 'READ'
* -----
      RJV. FIND JOB-VARIABLES WITH NAME = #JV-NAME
          AND NODE = #NODE
          AND FUNCTION = #FUNCTION
          AND READ-PASSWORD = #PASSWORD
.....
      VALUE 'WRITE'
* -----
      WJV. FIND JOB-VARIABLES WITH NAME = #JV-NAME
          AND NODE = #NODE
          AND FUNCTION = #FUNCTION
          AND DATA = #DATA
          AND WRITE-PASSWORD = #PASSWORD
          AND LENGTH = #VALUE-LENGTH
          AND SUBSTRING-START = #SUBSTR-START
          AND SUBSTRING-LENGTH = #SUBSTR-LENGTH

```

Spool File Handling

The following example illustrates how job output can be read from the spool in a VSE/ESA system and written to a file.

```

.....
*
* An input mask is defined with the INPUT statement:
*
INPUT
  / ' Job name.....:' #JOB
  / ' Job number.....:' #JOBN
  // ' Mark spool type...:' #SEL(1) 'CC' (I) 'completion codes'
  / 21X #SEL(2) 'RD' (I) ' reader queue'
  / 21X #SEL(3) 'LS' (I) ' list queue'
  / 21X #SEL(4) 'PU' (I) ' punch queue'
  / 21X #SEL(5) 'XM' (I) ' transmit queue'
  / ' Dataset number...:' #DS
  // 'to' (I)
  / ' Library.....:' WRITE-FILE.LIBRARY
  / ' Sub library....:' WRITE-FILE.SUBLIB
  / ' Member.....:' WRITE-FILE.MEMBER
  / ' Member type....:' WRITE-FILE.MEMBER-TYPE
  / ' VSAM catalog...:' WRITE-FILE.VSAM-CAT

...
*
* To read the job output, the READ-SPOOL view is called,
* identifying the job and output file required:
*
      FIND READ-SPOOL WITH JOB-NAME      = #JOB
                          AND JOB-NUMBER = #JOBN
                          AND TYPE       = #TYPE
                          AND DATA-SET  = #DS
                          AND NODE       = ##NODE

.....
*
* To write the output file to a file, the WRITE-FILE view is
* called, specifying the destination member:
*
      PROCESS WRITE-FILE USING LIBRARY = WRITE-FILE.LIBRARY
                          ,   SUBLIB   = WRITE-FILE.SUBLIB
                          ,   VSAM-CAT  = WRITE-FILE.VSAM-CAT
                          ,   MEMBER    = WRITE-FILE.MEMBER
                          ,   MEMBER-TYPE = WRITE-FILE.MEMBER-TYPE
                          ,   RECORD    = READ-SPOOL.RECORD
                          ,   NODE      = ##NODE

      END-FIND
.....

```

With the Entire System Server, different types of system data can thus be accessed and stored as conventional files using easy Natural programs. If such programs are implemented in applications, even users with no special computer training can use this advanced technology.

Imagination Is the Limit

The above examples illustrate the kind of tasks the Entire System Server can be used for, but they are not exhaustive. Small programs can be written for specific tasks, but more elaborate site-specific applications can be built to automate whole areas of data center tasks. As already mentioned, Software AG provides ready applications based on the Entire System Server in the area of operations scheduling, event management, and output handling (see the section What is Entire System Server).

Additionally, the Entire System Server installation tape provides a comprehensive online tutorial consisting of sample programs for every Entire System Server view. These programs not only serve as a useful online training guide, but can also be customized to meet the requirements of the installation, and can be used as a starting point for the development of more complex applications.

Experienced application developers and system programmers will readily recognize the potential of using the Entire System Server as a powerful aid to build tools for their work. If we can say that the Entire System Server provides the brush and the colors, then the application developers and system programmers paint the picture. As with all creative artists, their ingenuity is the limit.