Entire System Server

Concepts and Facilities

Version 3.7.2

October 2022

This document applies to Entire System Server Version 3.7.2 and all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Copyright © 1987-2022 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA, Inc. and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at http://softwareag.com/licenses.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at http://softwareag.com/licenses/ and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software AG Products / Copyright and Trademark Notices of Software AG Products". These documents are part of the product documentation, located at http://softwareag.com/licenses and/or in the root installation directory of the licensed product(s).

Use, reproduction, transfer, publication or disclosure is prohibited except as specifically provided for in your License Agreement with Software AG.

Document ID: NPR-CONCEPTS-372-20221014

Table of Contents

1 About this Documentation 1
- Document Conventions 2
- Online Information and Support 2
- Data Protection 3

2 Concepts and Facilities 5

3 What is Entire System Server? 7
- General Information 8
- Advantages of 4th Generation Technology 9
- Entire System Server in a Distributed Processing Environment 13
- Tasks the Entire System Server Can Perform 14
- Security 16
- Easy Installation and Maintenance 16

4 Benefits of Using the Entire System Server 17
- General Information 18
- Increased Flexibility and Productivity 18
- Cost Reduction 20
- Improved Machine Performance 21

5 Entire System Server at Work - Examples 23
- General Information 24
- IEBCOPY Utility 24
- Disk Maintenance 26
- File Maintenance 28
- Job Handling 33
- Spool File Handling 36
- Receiving Emails 37
- Access to z/OS UNIX Files 41
- Imagination Is the Limit 43

1 About this Documentation

- Document Conventions 2
- Online Information and Support 2
- Data Protection 3

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
`Monospace font`	Identifies service names and locations in the format *`folder.subfolder.service`*, APIs, Java classes, methods, properties.
`Italic`	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Product Documentation

You can find the product documentation on our documentation website at **https://documentation.softwareag.com**.

In addition, you can also access the cloud product documentation via **https://www.softwareag.cloud**. Navigate to the desired product and then, depending on your solution, go to “Developer Center”, “User Center” or “Documentation”.

Product Training

You can find helpful product training material on our Learning Portal at **https://knowledge.softwareag.com**.

Tech Community

You can collaborate with Software AG experts on our Tech Community website at **https://tech-community.softwareag.com**. From here you can, for example:

- Browse through our vast knowledge base.
- Ask questions and find answers in our discussion forums.
- Get the latest Software AG news and announcements.
- Explore our communities.
- Go to our public GitHub and Docker repositories at **https://github.com/softwareag** and **https://hub.docker.com/publishers/softwareag** and discover additional Software AG resources.

Product Support

Support for Software AG products is provided to licensed customers via our Empower Portal at **https://empower.softwareag.com**. Many services on this portal require that you have an account. If you do not yet have one, you can request it at **https://empower.softwareag.com/register**. Once you have an account, you can, for example:

- Download products, updates and fixes.
- Search the Knowledge Center for technical information and tips.
- Subscribe to early warnings and critical alerts.
- Open and update support incidents.
- Add product feature requests.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

2 Concepts and Facilities

This documentation covers the following topics:

What is Entire System Server?	Gives a definition of the Entire System Server, discusses some of the major advantages of using 4th generation technology, explains how Entire System Server can play a vital role in distributed processing, lists the kind of tasks the Entire System Server can perform and takes a brief look at security issues.
Benefits of Using the Entire System Server	Illustrates the benefits of using the Entire System Server as the basic technology in corporate data processing, and deals with some of the considerations concerning the switch to this new technology.
Entire System Server at Work - Examples	Briefly explains the principle of operation behind the Entire System Server, and illustrates the use of the Entire System Server with some detailed example coding.

3 What is Entire System Server?

- General Information 8
- Advantages of 4th Generation Technology 9
- Entire System Server in a Distributed Processing Environment 13
- Tasks the Entire System Server Can Perform 14
- Security 16
- Easy Installation and Maintenance 16

General Information

The Entire System Server is a Software AG product that makes operating system information and system services available to the user, whether it be an application developer, system programmer, or computer operator.

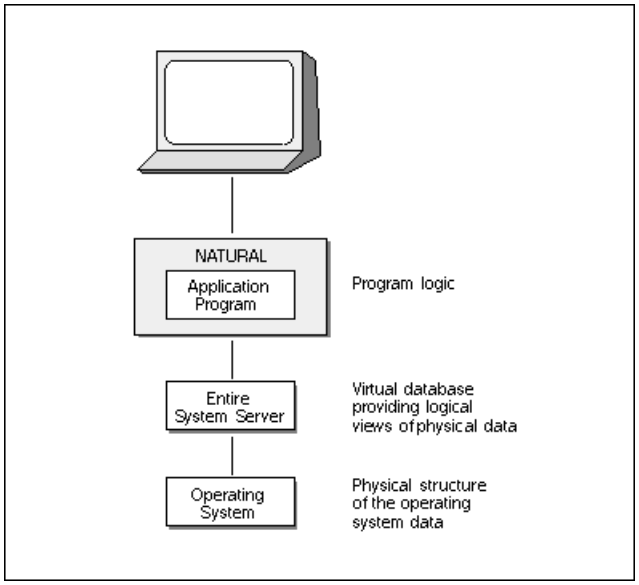
The Entire System Server does this by providing a logical view of the operating system in much the same way as a database management system provides access to a database such as Adabas or DB2. For this reason, the Entire System Server is often referred to as a virtual database for system data management. With the Entire System Server, a comprehensive number of views are made available, the following table lists only a sample selection:

View Name	Description
`READ-FILE`	Read data from any specific file.
`WRITE-FILE`	Write data to any specific file.
`VTOC-UPDATE`	Perform VTOC maintenance functions.
`ACTIVE-JOBS`	Access active job information.
`SPOOL-QUEUE`	Access spool queue information.
`CONSOLE`	Access operator console.
`SEND-MESSAGE`	Send message to a specific user.
`SUBMIT`	Submit a batch job.

With its large array of views, the Entire System Server constitutes a major enhancement to sites that use Natural, the proven and powerful 4th generation development environment from Software AG which enjoys world-wide renown. This yields an important two-way benefit:

- Operating system information and services are made available to application programmers using standard Natural statements (Find, Process);
- Full Natural functionality is extended to data center and operations personnel.

The following figure illustrates the Entire System Server as the virtual database of the operating system.



As outlined in the following subsections, the Entire System Server can be used in almost any computing setup. It can provide operating system data and services in a single-computer shop, in heterogeneous, multi-CPU setups, and it can also serve distributed computing environments: client programs from anywhere within a (heterogeneous) computer network can request a system service available anywhere else within the same network, and Entire System Server can provide that service.

Advantages of 4th Generation Technology

Greater Flexibility

One of the most important distinguishing features of Natural's 4th generation technology is the strict distinction it makes between the logical level and the physical level of data processing, that is, between the application logic and the structure of the database on which corporate information is physically stored. For example, entries in a personnel file are assigned fixed attributes such as surname, first name, age, and sex. Natural programs can refer to these attributes that are defined as fields in the logical view of the file, and the information is then extracted from the database regardless of the file structure.

The Entire System Server mirrors this approach by separating the application logic from the internals of the operating system on which the application runs. For example, a Natural program can access active operating system job information through the Entire System Server by referring to such assigned fixed attributes defined in the logical view of active jobs as job-name, job-number, type and priority, as well as operating-system-specific attributes such as status or step-name (z/OS, z/VSE), and account number or the predefined maximum time the job can use (BS2000).

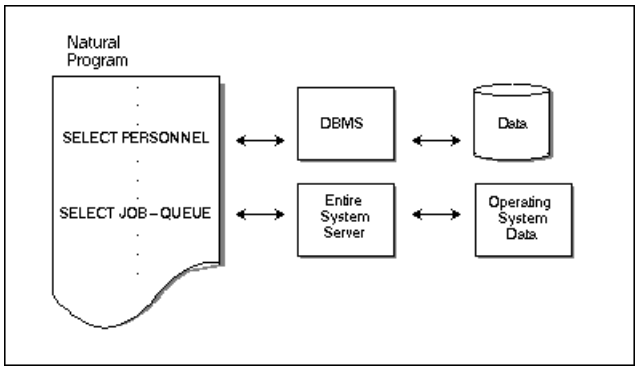
With this in mind, the implications of introducing the Entire System Server to a Natural environment are immense and constitute a major technological advance.

Whereas before, application programmers could only access the database from Natural programs without knowledge of the physical structure of the database, Entire System Server now also allows them to access operating system information and services using the same syntax and without detailed knowledge of the operating system and coding languages such as Assembler or COBOL. At the same time, the introduction of 4th generation technology to the data center means that system programmers need not program in Assembler any more, but can use Natural, which is distinguished by its flexibility and ease of use.

Again, the twin advantage of this technology becomes apparent immediately:

- Natural programmers now have the capability of managing their own files and accessing other operating system services, enabling them to write much more powerful applications without the need for special knowledge of and/or further training in operating system issues;
- System programmers and computer operators can now write their own site-specific (Natural) applications based on Entire System Server to automate or simplify their tasks. For example, programs are easily written for operations such as disk and dataset maintenance, file and catalog management, job and printout information retrieval. Programs can even be written for resource management, for example, ensuring optimal use of disk space.

The following figure illustrates the use of the Entire System Server:



Both data center personnel and application developers benefit from the Entire System Server, as it allows them to react much more independently and flexibly to their work requirements. Using Natural, program development is much faster, while the run-time performance of Natural programs compares favorably with programs written using 3rd generation technology.

Always willing to prove its point, Software AG itself has already realized ready-for-work application solutions based on the Entire System Server in the following areas of data center tasks:

- **Production management:**
  Planning and automatic scheduling of batch jobs and online tasks (Entire Operations, formerly Natural Operations);

- **Event management:**
  Console message filter and automatic response to system events (Entire Event Management, formerly Natural Console Management);
- **Output management:**
  Automatic bundling, separation, distribution and archiving of output (Entire Output Management, formerly Natural Output Management).

For more details on these products, see the Software AG publications **System Automation: The Road to Operation Center Management** (Order Number DCS-111-006), and **The System Management Solution** (Order Number RZS11B1E062 680).

Additionally, Software AG offers a solution for comprehensive application development:

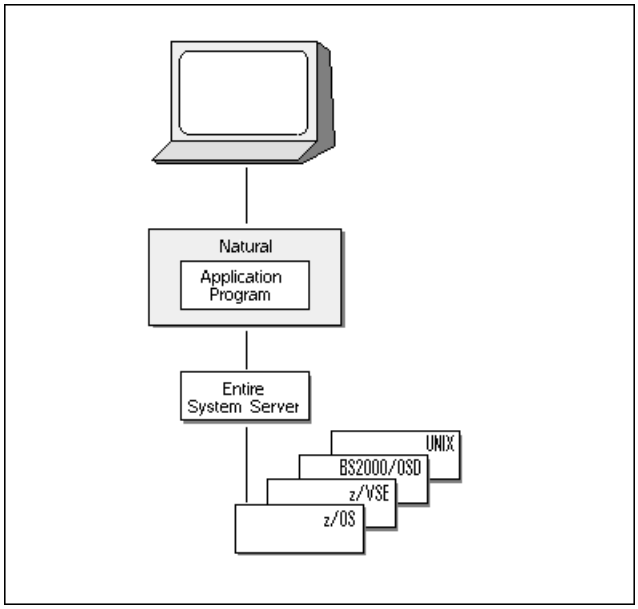
- **Application development:**
  Software AG's application development toolkit Natural ISPF provides a single, comfortable system image of an installation's data processing resources. The Entire System Server extends Natural ISPF to allow access not only to Natural objects, but also to a wide range of other objects such as JCL, JCS, objects written in other programming languages such as COBOL and Assembler, and, of course, operating system items.

For a more detailed overview of Natural ISPF, see the Software AG publication **Natural ISPF Concepts and Facilities** (Order Number ISP-141-006).

Platform Independence and Remote Access

Another major advantage of Natural's separation of the logical from the physical level is the resulting independence of operating platform. Applications written in Natural can be ported to a wide range of operating platforms without any changes being required. With the Entire System Server installed, all operating-system-specific information and services become available to the Natural environment, whether the platform is z/OS, FACOM, z/VSE, z/VSE, BS2000 or UNIX. For example, Natural applications running on a UNIX platform can access host data, and vice versa. Investments in Entire System Server-based application software are therefore protected, should your data processing department be rescaled during future corporate reorganization or restructuring exercises, or if the underlying operating platform is modified or upgraded.

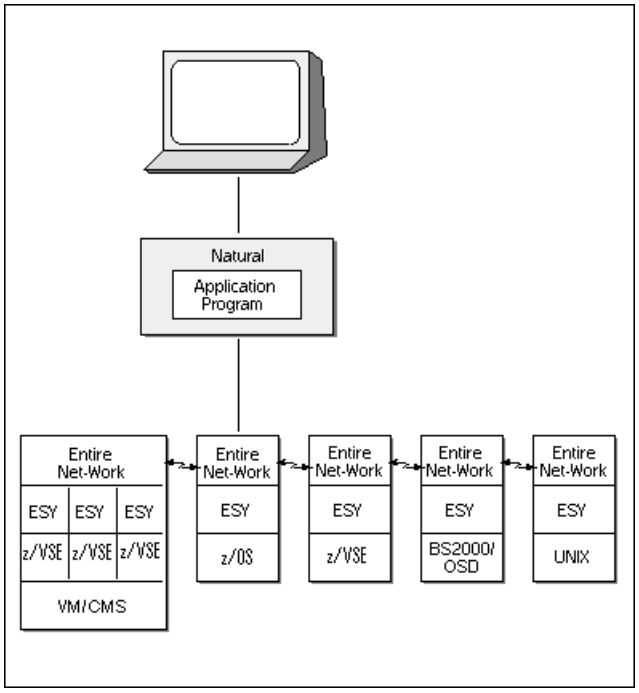
The following figure illustrates the Entire System Server as platform-independent interface to operating system information and services:



However, the wider implication of operating system independence is that it allows multi-CPU support within a corporate network of interconnected computers. With Software AG's communication product Entire Net-Work installed on each computer within the network, operations across computers and even across operating systems in a heterogeneous network are possible.

For example, given an IBM mainframe running VM and several z/VSEs, the combination of the Entire System Server and Entire Net-Work makes communication among the various z/VSEs possible. It allows the exchange of system data between computers in the network running, for example, z/OS operating systems, BS2000, or even UNIX. The whole computer network can be monitored and controlled from a single location, and Natural applications can access data and operating system information and services on any computer within the network.

The only requirement is that an Entire Net-Work component and an Entire System Server (ESY) nucleus be installed on each computer, as illustrated in the following figure:



Note:

The abbreviation ESY in the figure stands for the Entire System Server.

Regardless of operating system or location within a computer network, applications based on the Entire System Server can provide a uniform Application Program Interface (API) for a variety of tasks that in the past required a number of different, operating system specific tools, as well as special knowledge of the operating environment the tools were designed for.

Entire System Server in a Distributed Processing Environment

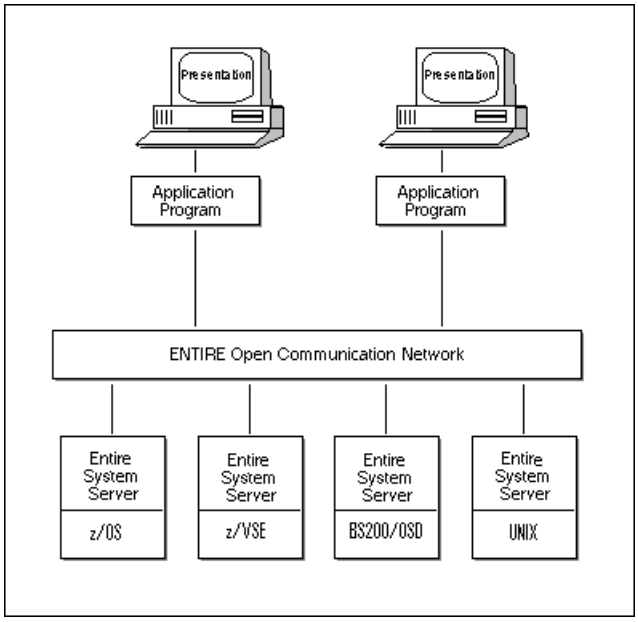
The ever-spreading use of the UNIX operating system and the increasing integration of workstations in computer networks has created a rising demand for system management tools that support a distributed processing environment, in which all components are defined in terms of clients and servers.

Client/server computing is more than remote database access of the type that can be achieved in a multi-CPU environment as described above. In the client/server environment, applications can be "cut up" and distributed among the computers within the network as appropriate. For example, the presentation component may run on a workstation client, while other parts of the application may run on an operating system most suited to their needs. All application components are inter-connected with appropriate communication software.

Based on Natural, which allows pieces of applications to be easily ported to different computers within a heterogeneous network, the Entire technology of Software AG allows the evolutionary transition of purely mainframe-oriented software systems to a client/server environment that spans mainframes, mid-range computers, UNIX systems and workstations.

The Entire System Server, in combination with Entire Net-Work as part of the communication software, allows operating system services and information to be tied into the Entire solution in a modern, heterogeneous computer network.

The following figure illustrates the use of the Entire System Server in a client/server setup:

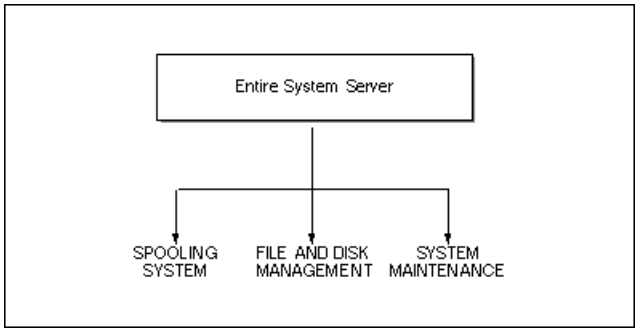


This approach cuts the Gordian knot of integrating an operating system server into a corporate data processing environment that provides all types of users with a single image of all available distributed information and services. Using existing methods and syntax, the Entire System Server provides a relatively simple solution to an apparently complex problem.

Tasks the Entire System Server Can Perform

This subsection gives a brief overview of the tasks that programs based on the Entire System Server can perform. The tasks mentioned are only a selection from a wide array of possible implementations. *Entire System Server at Work - Examples* gives some typical examples of programs that call the Entire System Server to access operating system items.

The primary use of the Entire System Server lies in the areas of spool management, file and disk management and system maintenance, as illustrated by the following figure:



- **Spooling System:**
  Using the spool management group of views, programs can be written that allow the user to write procedures to submit jobs anywhere in the computer network, track each job's progress and inspect return codes and output files. One good example of the use of this functionality is in applications that issue long requests to the operating system: the online application can trigger a batch job that performs the request. The application can continue work, and it can look into the spool to check the job's progress and check its result. The views used for this example are Submit, Spool-files and Read-spool.
- **File and Disk Management:** The views that provide file management functionality allow the allocation, deletion and editing of files, as well as the copying of files from one library into another. In a computer network, files can be maintained anywhere in the network, and file transfer can be performed from one computer to another, regardless of the operating system. For example: whereas before, supplying data to a remote location involved creating a tape, taking it to the remote location and reading it there, the Entire System Server makes file transfer a simple online operation using the Copy-File view.

  Useful disk management operations using the Vtoc-update view include the renaming, scratching and purging of catalog entries, as well as releasing unused space.
- **System Maintenance:**
  System maintenance functions offered by the Entire System Server include the Console view which provides an image of the operator console and allows the user to issue operator commands to the operating system. The Send-Message view allows messages to be distributed to users around the network.

  Another important use of system maintenance views, is the ability to execute operating system functions (for example, `IEBCOPY`, `IDCAMS`, `ARCHIVE`) with standard Natural syntax. Operating system utilities can be given a uniform user interface, and no special knowledge of operating system internals is required.

  On the hardware side, if a new computer is added to the network, no changes to the applications of the installation are required. All that is needed is a new Entire System Server node on the new machine.

Security

Making operating system services and information available to multiple types of user raises security issues. The Entire System Server supports a variety of standard security packages such as RACF, TOP SECRET and ACF2. The introduction of the Entire System Server to your installation has no effect on the authorization levels in place at your site: before the user is given access to any operating system data or service, authorization is derived from the external security system. This means that the user has the same authorization level for these items with or without the Entire System Server.

Additionally, with Natural Security, access to operating system resources and information can be controlled in the same way as the logical views of Natural of the database are protected: each Entire System Server view can be given a specific authorization profile.

Easy Installation and Maintenance

During the installation of the Entire System Server, no restart of the operating system is necessary. There are no “hooks” into the operating system. The clearly defined and stable interfaces provided by the operating system (SSI, SAF, SMS, CMS, UCON, LMS, etc.), together with the modifiable startup parameters make the Entire System Server easy to customize to changing conditions at your site. No maintenance work is required, the Entire System Server provides you with a 24-hour a day, 7-day a week service.

4 Benefits of Using the Entire System Server

- General Information 18
- Increased Flexibility and Productivity 18
- Cost Reduction 20
- Improved Machine Performance 21

General Information

The benefits of using the Entire System Server as basic technology in a heterogeneous corporate data processing environment can be summed up as:

- Increased flexibility and productivity for all personnel involved in data processing;
- Reduced costs in system maintenance and training;
- Improved machine performance.

This section analyses these benefits in more detail.

Increased Flexibility and Productivity

- *Application Programming*
- *Power to the User*
- *Data Center*

Use of the Entire System Server increases the flexibility and productivity of all personnel involved in data processing: from the system programmer managing the storage media and optimizing machine performance,by way of the application developer including operating system information and services in his or her programs, right up to the end-user retrieving an output file stored in an automated office system.

Application Programming

Incorporating Entire System Server functionality in applications written in Natural, application programmers are provided with capabilities that include Read and Write access to files, file allocation and maintenance, job status display and spool queue display.

This relieves data center personnel of much routine work, and enables the development of applications that automate certain time-consuming procedures and functions, signalling an end of the distinction between online and batch processing.

For example, a batch job can be built using program logic, and can be submitted, its progress monitored, and its output data read into the program without any interruption.

This integration of operating system functions and information in application development allows the simple and quick realization of much more powerful applications. In particular, application developers are given more control over system resources.

Like the data center, the application development department becomes much more productive and efficient at no extra training or labor cost, as the added capabilities provided by the Entire System Server are available to the application programmer without leaving the Natural environment.

Power to the User

At the high end of data processing, applications based on the Entire System Server allow the selective delegation of more processing power to the end-users, who can use standardized interfaces written for them to access the operating system to perform such functions as allocate their own files, display the spool queue, and track the progress of jobs.

The Entire System Server thus answers a basic need in modern enterprise computing. An increasing number of corporations are following the general trend of decentralizing their data processing activities. They are moving away from the large, central mainframe towards mid-range computers and workstations installed in the departments, all interlinked within a computer network to provide selective processing power where it is needed most: in the hands of the user.

In combination with Entire Net-Work, applications based on the Entire System Server can serve such a distributed environment and provide a single system image across the whole network. Users can control their own environments without troubling the data center, which can get on with the logistical tasks of network management, event management, software distribution and workload balancing.

Data Center

The Entire System Server allows Natural programs to be written to satisfy unique, site-specific requirements in the areas of system maintenance (disk and file), job submission and management, resource management, accounting data (SMF) and system console operations. This enables data center staff to respond quickly and flexibly to problems and questions arising from the user community:

- All system queries have a common syntax, meaning that the system programmer or computer operator can concentrate on getting the job done, rather than worrying about how to do it. There is no need to become embroiled in the details of the operating system structure and use complicated Assembler routines;
- Required information is easily accessed using simple but powerful statements or commands, allowing system programmers to deal quickly with ad hoc requests and react to problems before they become critical;
- The use of Entire Net-Work in a multi-computer environment allows access to information located at remote sites. Additionally, a whole heterogeneous computer network can be controlled from a single location.

All these points make system programmers and computer operators much more efficient and productive members of staff. With the use of the Entire System Server, the data center becomes the nerve center of distributed corporate data processing.

Cost Reduction

Whichever way corporate decisions concerning their data processing methods go, whether it be towards a distributed environment or back to central mainframes, the Entire System Server protects investments in existing applications and thus helps ensure that the time and costs involved in changes to the hardware are kept at a minimum.

In the day-to-day running of data processing operations, a range of different software vendor products were traditionally required, which do not communicate with each other and often have no programming interface (API). The use of the Entire System Server alleviates this problem and can help cut costs in a number of areas:

- **Labor and training costs:**

  To support the increasing number of heterogeneous computer networks using traditional methods of system programming, expert knowledge of the various hardware and operating system configurations had to be condensed in a small team of staff. Such system programmer expertise is rare and expensive, as is the cost of training to acquire such expertise.

  Additionally, the typical tools for data center operations are Assembler programs, which are complicated and expensive to code. The use of the Entire System Server eliminates the need to know the details of hardware and operating system structures. Through the automated facilities provided by Natural, fewer lines of code need to be generated for a Natural application than for applications written in traditional languages. Simple Natural programs can therefore replace complex Assembler routines, and system programmers can implement tools with a minimum of training and effort. The Entire System Server can thus reduce labor and training costs significantly.

  Making operating system information and services available to application programmers using the Entire System Server also helps save training costs in the application development department, as the operating system can be accessed using Natural statements already familiar to Natural programmers.

- **Storage device costs:**

  Using the Entire System Server, flexible programs for managing storage devices (DASD) and datasets can be written and implemented. For example, programs can manage DASD space optimally by freeing unused space, generating reports on datasets and relocating datasets if necessary. Optimal management of such system resources using site-specific Natural programs helps reduce the number of storage devices required.

- **Service costs:**

  User queries to the data center concerning information such as the position of jobs in the job queue, the status of jobs or printouts are time-consuming and distract data center personnel from their system programming tasks. Though the Entire System Server can be used to satisfy such requests quickly and efficiently, the best solution is to build Natural applications with customized user interfaces to function as an online help desk that allows users to retrieve such information themselves. This eliminates the cost of having an information service provided by the data center.

Improved Machine Performance

The use of the Entire System Server can lead to an improvement in machine performance.

For example, running the Entire System Server and starting Natural programs under a TP-monitor (Com-plete, CICS, IMS, openUTM etc.) makes information available to the user that he or she could previously only obtain by running batch jobs or utility jobs (TSO, TIAM).

Operating system information accessed via Natural programs can be used either for pure retrieval purposes (display) or to control the system as components of a Natural application. This makes it possible to reduce the machine workload, as fewer TSO or TIAM users means less occupied storage, and it also helps prevent system shortages or failures.

5 Entire System Server at Work - Examples

- General Information 24
- IEBCOPY Utility 24
- Disk Maintenance 26
- File Maintenance 28
- Job Handling 33
- Spool File Handling 36
- Receiving Emails 37
- Access to z/OS UNIX Files 41
- Imagination Is the Limit 43

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 z/OS 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)'
  ....
```

```
*
* The PROCESS statement calls the IEBCOPY utility via the
* Entire System Server view IEBCOPY to perform the copy
* operation.
* The NODE variable is used if the destination member resides on
* a different node:
*
 ....
     PROCESS IEBCOPY USING IN-DSNAME  = IEBCOPY.IN-DSNAME
     ,                     OUT-DSNAME = IEBCOPY.OUT-DSNAME
     ,                     IN-MEMBER  = IEBCOPY.IN-MEMBER
     ,                     OUT-MEMBER = IEBCOPY.OUT-MEMBER
     ,                     REPLACE    = IEBCOPY.REPLACE
     ,                     NODE       = ##NODE
 ....
* In case of an error, the user can be reprompted with the input
* map and an error message:
 ....
          IF ERROR-CODE > 0
             REINPUT ERROR-TEXT
          END-IF
```

Running such a simple Natural program results in the following online prompt:

```
                              IEBCOPY utility

   Input Dsname ...: ____________________________________________________

   Volser .........: ______

   Output Dsname ..: ____________________________________________________

   Volser .........: ______

   Member .........: __________

   New Name .......: __________

   Replace ........: ___ (yes/no)
```

All the user has to do is fill in the required information and press Enter to perform the copy operation for example:

```
                                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.

Disk 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 z/OS 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  Unit  avail.  OPEN    mounted   mount      avail.   FREE   FREE   FREE
Adr   Type   Unit   DCB'S   Volume  Attributes of Volume   CYL  TRACKS 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    XKGSD1  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 z/OS and BS2000 environment. The program prompts the user for the name of a dataset to be compressed (in BS2000 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 z/VSE 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 z/OS to a z/VSE 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 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 z/OS, z/VSE and BS2000 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 z/OS 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 system (all jobs beginning with `N` are displayed):

```
Job-Name Type   JobNr.  Cpu used   Accnt-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 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 z/VSE 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.

Receiving Emails

Using view RECEIVE-EMAIL and the functions STATUS, LIST, READ and DELETE under statement FIND you can manage your incoming e-mails under z/OS. For communication the Internet Message Access Protocol (IMAP) is used. IMAP is defined in RFC-3501. The name of the mail server IMAP-HOST and the port number IMAP-PORT for access are specified by startup parameters. Other startup parameters specify the default number of mails to be listed by the LIST function and the maximum amount of mails to be read in. This helps to prevent sending unnecessary huge amounts of data for large mailboxes. See also LIST-NUM-MAILS and LIST-NUM-MAILS-MAX.

z/OS allows you to use Application Transparent Transport Layer Security (AT-TLS). This enables a secure access using port 993. Login credentials such as user ID and password are handed over in the Natural program.

We give some simple examples to explain the functions STATUS, LIST, READ and DELETE. Upon this we discuss more complex use cases:

- Status
- List
- Read
- Delete
- Read all Mails from a Specific Sender with a Specific Search Term in the Subject Line
- List the Latest 1000 Mails from a Mail Box Containing 10.000 Mails

Status

The following example illustrates a use case for the function STATUS in which we request

- the total number of mails (NUMBER-MAILS);
- the number of unread mails (NUMBER-NEW-MAILS).

```
FIND    RECEIVE-EMAIL WITH   NODE = #NODE
        AND   FUNCTION    = 'STATUS'
        AND   USERID      = 'horst.mustermann@softwareag.com'
        AND   PASSWORD    = #PASSW
        PRINT '=' NUMBER-MAILS
        PRINT '=' NUMBER-NEW-MAILS
END-FIND
```

Note: It is useful to invoke this function from time to time to check if new mails have arrived. If so, you might then want to invoke other functions to list, read or delete mails.

List

Using function `LIST` you can display directory data of mails such as `STATUS`, `SENDER-MAIL`, `SENDER-NAME`, `SUBJECT`, `RECIPIENTS`, `MESSAGE-ID`, `UID` or `DATE`. This is illustrated in the following statement:

```
FIND    RECEIVE-EMAIL WITH   NODE = #NODE
        AND   FUNCTION    = 'LIST'
        AND   USERID      = 'horst.mustermann@softwareag.com'
        AND   PASSWORD    = #PASSW
        AND   LIST-NUM-MAILS  = 15
…
        PRINT '=' STATUS
        PRINT '=' SENDER-MAIL
        PRINT '=' SENDER-NAME
        PRINT '=' SUBJECT
        PRINT '=' RECIPIENTS
        PRINT '=' MESSAGE-ID
        PRINT '=' UID
        PRINT '=' DATE
END-FIND
```

Notes:

1. Using field `LIST-NUM-MAILS` you can limit the number of mails to be listed. If this field is omitted, a default defined in the startup parameters is assumed.
2. In field `STATUS` two different values can be displayed: `NEW` for a mail that yet has not been read and `SEEN` for a mail that has already been read.
3. Field `MESSAGE-ID` is a unique world-wide reference. A `MESSAGE-ID` is therefore long and complex. However, a `MESSAGE-ID` cannot be used to read or delete a mail using IMAP. Instead, IMAP uses the `UID` as identifier.
4. By adding `STATUS = 'NEW'` to the `FIND` statement only new mails will be listed.
5. By adding code such as `SENDER-MAIL = 'richard.wagner@softwareag.com'` to the `FIND` statement you can limit the list to mails from a specific sender. You can also use a search term embraced by stars (*) to limit your search to a sender that contains a specific substring, for example to list mails from senders that contain the string "richard" independent of its position specify `SENDER-MAIL='*Richard*'` or `SENDER-MAIL='*Wagner*'`.

Read

The next example illustrates the usage of function `Read`:

```
FIND     RECEIVE-EMAIL WITH   NODE = #NODE
         AND   FUNCTION = 'READ'
         AND   USERID   = 'horst.mustermann@softwareag.com'
         AND   PASSWORD = #PASSW
         AND   UID      =  #UID
         PRINT '=' RECORD
END-FIND
```

Notes:

1. The mail to be read is identified using a `UID` (unique identifier). UIDs are displayed using function `LIST`.
2. The contents of the mail to be read is displayed using field `RECORD`.

Delete

You can delete a mail using a `FIND` statement similar to the following example:

```
FIND      RECEIVE-EMAIL WITH   NODE = #NODE
          AND   FUNCTION = 'DELETE'
          AND   USERID   = 'horst.mustermann@softwareag.com'
          AND   PASSWORD = #PASSW
          AND   UID      =  #UID
          PRINT '=' ERROR-CODE
          PRINT '=' ERROR-TEXT
          PRINT '=' MAILSERVER-RESPONSE
END-FIND
```

If the call was successful, the following message is displayed:

```
ERROR-CODE:    0
ERROR-TEXT: ESY3000
MAILSERVER-RESPONSE: Delete successful ↩
```

Read all Mails from a Specific Sender with a Specific Search Term in the Subject Line

Assume we want to read all and only those mails with a matching term in the subject line from a specific sender:

```
FIND    RECEIVE-EMAIL WITH   NODE = #NODE
        AND   FUNCTION    = 'LIST'
        AND   USERID      = 'horst.mustermann@softwareag.com'
        AND   PASSWORD    = #PASSW
        AND   SUBJECT     = '*SALE*'
        AND   SENDER-MAIL = 'my.boss@company.com'
END-FIND
```

In this example we list mails from one sender that contain the word "SALE".

As already shown **above**, a term embraced by stars (*) matches any string in which the term occurs independent of its position. In this example case we get all messages in which the term `SALE` occurs somewhere in the subject line. For the sender, on the other hand, we require an exact match, that is we restrict our statement to mails with a sender that exactly matches the e-mail address of the boss.

List the Latest 1000 Mails from a Mail Box Containing 10.000 Mails

Assume a mailbox contains approximately 10.000 mails. We want to list the latest 1000. We could achieve this by one `LIST` call. However, this would imply an inadequately long response time. To reduce response time, we successively call the `LIST` command for a fraction (here 100) until the required amount of mails is listed. This is done by the following loop:

```
#START-MAIL-NR := #NUMBER-OF-MAILS
#LIST-NUM  := 100
FOR #I = 1 TO 10
FIND    RECEIVE-EMAIL WITH   NODE = #NODE
        AND   FUNCTION    = 'LIST'
        AND   USERID      = 'horst.mustermann@softwareag.com'
        AND   PASSWORD    = #PASSW
        AND   LIST-NUM-MAILS       = #LIST-NUM
        AND   LIST-NUM-MAILS-START = #START-MAIL-NR
END-FIND
#START-MAIL-NR := #START-MAIL-NR - #LIST-NUM
END-FOR
```

Before the loop is entered, variable `START-MAIL-NR` is set to `NUMBER-OF-MAILS`. This ensures that the list starts with the latest mail.

Note that for each iteration step the position from which to list mails, controlled by variable `START-MAIL-NR`, has to be decremented by the number of mails listed in one step (= `LIST-NUM` = `100`).

After the first iteration step, the starting point is decremented by `LIST-NUM-MAILS` (=`100`) to prepare the second iteration step, and so on until the nth step.

Access to z/OS UNIX Files

This section covers access to the z/OS UNIX file system within NATURAL applications. NATURAL itself already provides this in TSO or batchjobs but not in online monitors like CICS or Com-plete. Now, Entire System Server itself provides an access to the z/OS UNIX file-system. There are 3 new views called *WRITE-UNIX-FILE*, *READ-UNIX-FILE* and *UNIX-DIRECTORY*. A fully qualified path-name must be provided to locate the file you want to read or write to. Here is an example of how to write to a UNIX:

```
#RECORD-LENGTH := 80
#PATH   := '/u/nat/wkk/beispiele/lrec80 '
PROCESS WRITE-UNIX-FILE  USING NODE   = #NODE
   ,                     PATH   = #PATH
   ,                     RECORD  = #RECORD
   ,                     RECORD-LENGTH   = #RECORD-LENGTH
```

Note the fully qualified pathname for the file to be written to. It must start with a slash. Also note that we have set `RECORD-LENGTH` to 80 characters. Otherwise, a default record-length of 132 would be used. The maximum value for `RECORD-LENGTH` is 253. At the end the file must be closed as in other "update views" such as `WRITE-FILE` or `SUBMIT`:

```
PROCESS WRITE-UNIX-FILE  USING NODE   = #NODE
  ,                      FUNCTION     = 'CLOSE'
```

The DDM looks like this:

```
1  FN  FUNCTION                          A    8
       Function to be performed:
       blank   Write a record
       CLOSE   All records have been written, close file.                    ↩

1  A1  PATH                              A  128
       Fully qualified Path of file to be written, starting with /           ↩

1  A4  RECORD                            A  253
       Record to be written.
1  ID  IDENTIFIER                        A    8
       Needed if multiple update views are executing in                      ↩

       parallel, all requests for the same file must have same Identifier.   ↩

1  A7  RECORD-LENGTH                     N  3.0
       Length of record, default is 132.
```

How to z/OS UNIX file is demonstrated by the following example:

```
#PATH := '/u/nat/user/examples/lrec80 '
#RECORD-LENGTH := 80
FIND      READ-UNIX-FILE  WITH  NODE   = #NODE
          AND                   PATH   = #PATH
          AND                   RECORD-LENGTH = #RECORD-LENGTH
```

Note that you only need to specify `RECORD-LENGTH` for files that contains a record greater than 132 characters. Here, we want to read a file containing a record with 250 characters:

```
#PATH := '/u/nat/user/examples/lrec250'
#RECORD-LENGTH := 250
FIND    READ-UNIX-FILE  WITH  NODE   = #NODE
        AND                   PATH   = #PATH
        AND                   RECORD-LENGTH = #RECORD-LENGTH
```

The DDM looks like this:

```
A1  PATH                            A  128
    Path of file to be read.
A4  RECORD                          A  253
    Record to be read.
A7  RECORD-LENGTH                   N  3.0    D
```

By specifying the path of a directory you can get a list of all files and (sub)directories of that directory including a list of entries such as user ID, access-rights and file size. Here is an example:

```
#PATH   := '/u/nat/user/examples/ld'
FIND    UNIX-DIR   WITH  NODE   = #NODE
           AND          PATH  = #PATH
```

The DDM looks like this:

```
DB  Name                            F Leng
 A1  PATH                           A   54
     Path of directory to be read.
 A2  FILE                           A   54
     file name of directory
 A3  TYPE                           A    1
     D=Directory,3=standard file , see layout of BPXYFTYP
 A4  ACCESS-USER                    A    3
     Access rights of User, R=read,   W=write,   X=Execute.                    ↩

A5  ACCESS-GROUP                    A    3
     Access rights of group, R=read, W=write,   X=Execute.

A6  ACCESS-OTHER                    A    3
     Access rights for others, R=read, W=write,  X=Execute.
A7  OWNER                           A    8
```

```
     Owner of this file.
 A8  OWNER-GROUP                      A    8
     Group of owner.
 A9  FILE-SIZE                        N  8.0
     Size of file in bytes.
```

Field TYPE can be interpreted as defined in BPXYFTYP

```
** BPXYFTYP: File type definitions
**
FT_DIR               EQU   1      Directory File
FT_CHARSPEC          EQU   2      Character Special File
FT_REGFILE           EQU   3      Regular File
FT_FIFO              EQU   4      Named Pipe (FIFO) File
FT_SYMLINK           EQU   5      Symbolic link
*                    EQU   6      Reserved for Block Special
FT_SOCKET            EQU   7      Socket File
```

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 automize 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 medium 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.