ESTTA Tracking number: **ESTTA321629**
Filing date: **12/11/2009**

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE TRADEMARK TRIAL AND APPEAL BOARD

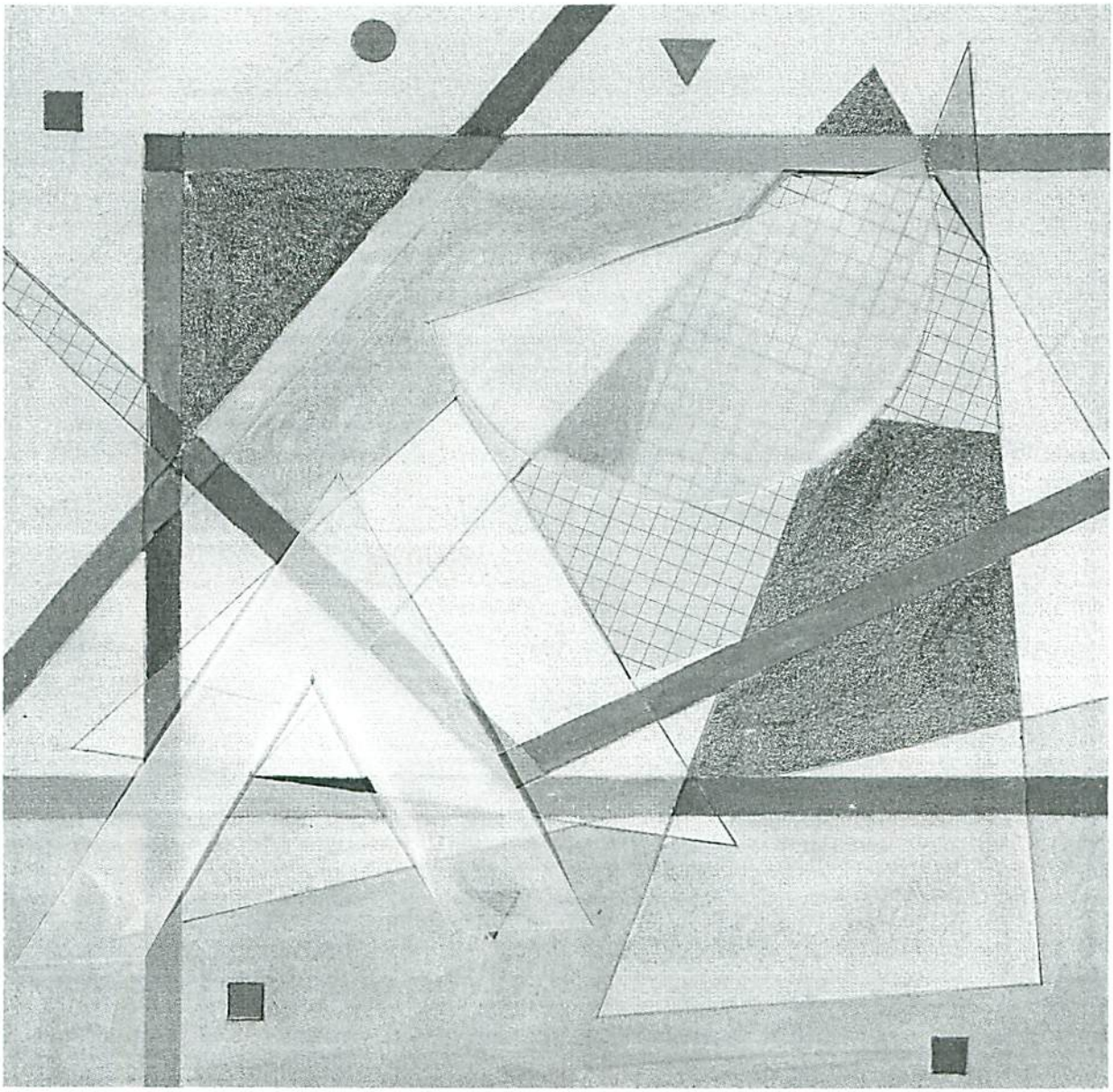
Proceeding	91179897
Party	Plaintiff Information Builders, Inc.
Correspondence Address	Howard F. Mandelbaum Levine & Mandelbaum 222 Bloomingdale Road, Suite 203 White Plains, NY 10605 UNITED STATES mail@levman.com
Submission	Testimony For Plaintiff
Filer's Name	Howard F Mandelbaum
Filer's e-mail	mail@levman.com
Signature	/Howard F Mandelbaum/
Date	12/11/2009
Attachments	PX44 of 87 - Part1.pdf (10 pages)(695766 bytes) PX44 of 87 - Part2.pdf (10 pages)(481550 bytes) PX44 of 87 - Part3.pdf (10 pages)(525497 bytes) PX44 of 87 - Part4.pdf (10 pages)(520090 bytes) PX44 of 87 - Part5.pdf (10 pages)(461330 bytes) PX44 of 87 - Part6.pdf (10 pages)(398213 bytes) PX44 of 87 - Part7.pdf (10 pages)(410937 bytes) PX44 of 87 - Part8.pdf (10 pages)(413657 bytes) PX44 of 87 - Part9.pdf (8 pages)(199895 bytes)

FOCUS SYSTEMS JOURNAL

A Magazine For The FOCUS Professional

July 1989 Volume 2 Number 4

PENGAD-Bayonne, N. J.
PLAINTIFF'S EXHIBIT
44 (W)
11-13-09



FOCUS SYSTEMS JOURNAL

Editor-In-Chief
Merv Adrian

Managing Editor
Amanda Regan

Art Director
Bernadette DeCelle

Circulation Manager
Mary Guiffre

The *FOCUS Systems Journal* is published bimonthly by Information Builders, Inc, New York, N.Y. 10001.

Subscription rate: $60.00 per calendar year (single copies $12.00). Foreign subscriptions must be paid in U.S. funds with additional airmail postage of $40 per year ($8 for Canadian subscribers). Multiple copy corporate subscription rates on request.

Address manuscript submissions and correspondence to the Managing Editor.

Opinions expressed in the Journal are solely those of the authors. Information Builders, Inc. assumes no responsibility or liability for damages resulting from errors or misuse of information contained herein.

Vendor names and products in this issue are trademarked by their providers.

All published Journal material is copyrighted © 1989 by Information Builders, Inc. All rights reserved.

Cover art, "Melange Avec Orange", by Elizabeth Gilliland. All reproduction rights are reserved by the artist.

From The Editor-In-Chief...

With this issue, the Journal takes a predominantly large-system approach to the concerns of FOCUS users. In recent months, there have been a number of developments in this arena: IBI's agreement with IBM to sell 9370 and 4381 computers as an IBM Business Partner, ports of FOCUS to IBM's AS/400 as well as Honeywell Bull and Hewlett-Packard computers, the shipment of the MVS version of LEVEL5 and the imminence of a LEVEL5-to-DB2 interface, our agreement to sell the Microsoft/Ashton-Tate/Sybase SQL Server, interfaces to Digital's CIT and Wang's WIIS, and more. There are more exciting developments to come: PM/FOCUS and Release 2 of LEVEL5 come to mind.

Some of the critical issues surrounding large-scale FOCUS use include: efficiency and its measurement in Bernard Blick's Measuring Code Efficiency and David Bland's Performance Tuning with the FOCUS-to-DB2 Interface; command executive and batch processing issues in Peter Van Der Lee's Using Dialogue Manager, Steven St. John's Using Session Monitor to Store Variables For Batch Processing, and Patricia Davis' Controlling FOCUS Read Loops and Restarts; and environmental issues in Larry Breindel's Isolating Operating System Commands – OSFEX and David Lovelace's Logical File Names.

We are delighted to have Nick Brown's thoughful and cogent letter in reply to Gregory Dorman's recent article on Dialogue Manager. We hope to see increasing use of the Journal as a medium for such exchanges.

I am pleased to announce that Amanda Regan has assumed the title of Managing Editor, a reflection of the outstanding contribution she has made to the publication. Russ Lipton has moved to other assignments in IBI, and we have moved into IBI ourselves as Information Builders Periodicals. We expect that our publications will benefit from the closer ties to IBI.

Finally, in our travels to FUSE meetings and Top Gun Schools, we have found increasing numbers of readers asking for back issues and a back issue index. We are pleased to present that index in this issue; it will be updated periodically. Take note of the discounted price for multiple copies and call us to complete your collection!

Performance Tuning With The FOCUS To DB2-SQL/DS Interface

David Bland
Pacific Software, Inc.

The FOCUS to DB2–SQL/DS Interface enables almost transparent access to DB2 or SQL/DS databases. David Bland discusses what types of requests are handled by the FOCUS processor and what types are handled by DB2 or SQL/DS, and how to write code to effectively use these processors.

FOCUS programmers now have available almost transparent access to DB2 (MVS) or SQL/DS (VM) databases. From an application programmer's point of view, it is possible to maintain a DB2 or SQL/DS application from within the FOCUS environment with little or no knowledge of the SQL language.

FOCUS programmers whose responsibilities are limited to reporting and simple updating could easily carry out their duties without even knowing whether the data was stored by FOCUS, DB2, or SQL/DS. However, as with all computer systems, performance problems may arise unexpectedly, particularly when an application's processing load increases during production cycles.

As more and more DB2 and SQL/DS shops take advantage of the power of FOCUS, it becomes important for FOCUS programmers to be aware of some of the performance implications of their coding practices and of the way that DB2 and SQL/DS handle certain types of common programming requests.

This article is not intended as a tutorial on the Interface, and not all of the factors affecting performance in a FOCUS to DB2 or SQL/DS environment will be discussed. The principal areas to be covered involve topics that are peculiarly related to accessing DB2 and SQL/DS data via FOCUS that may not be obvious, even to the experienced FOCUS or SQL programmer:

- Determining the Interface release level.
- Making sure that the DBMS does the JOIN, including some ways around Interface restrictions.
- FOCUS JOINs vs. SQL views – why it is not always such a good idea to rely upon SQL views.
- Optimizing tests done on DEFINEd fields.
- Understanding why RECORDLIMIT and READLIMIT tests do not always return results quickly.
- Determining when to COMMIT WORK in MODIFY processing.

Note: The term DBMS as used in this article refers generically to both DB2 and SQL/DS.

Determining The Release Level Of The Interface

The Interface to DB2 and SQL/DS is in its fourth release, but many programmers are not aware of what release they are running or what new features are available. The blue *SQL/DS and DB2 READ/WRITE Interface* manual, with which most people are familiar, describes the features of Release 1 of the Interface and is woefully out of date. Releases 2 and 3, which contain important new features, have been described in New Features Bulletins NFB204.2 and NFB204.3, which many programmers have never seen.

Users of FOCUS Release 5.5 should be on Interface Release 3; it is possible to verify this by issuing these two commands:

```
[CMS/TSO] SQL SET OPTIMIZATION ON
```
Valid in Release 3 only

```
[CMS/TSO] SQL SET SQLJOIN ON
```
Valid in Releases 2 or 3 only

In Release 3, both statements are synonymous; under Release 1, neither works. The techniques discussed in this article assume the use of Release 3.

FOCUS JOINs Vs. SQL Views

The optimization of JOINs is a principal goal of performance tuning in a FOCUS to DB2 or SQL/DS environment. Broadly speaking, there are three ways that DB2 or SQL/DS data can be JOINed from FOCUS:

- Use the FOCUS JOIN command.
- Create a multi-segment MFD and FOCSQL.
- Create an SQL view of the JOIN and reference the view (i.e., the MFD and FOCSQL are of the view, not the base tables).

In the first two cases, the optimization setting determines how FOCUS carries out the JOIN. With optimization off (the default), FOCUS reads each table individually, carrying out the JOIN from within FOCUS. With optimization on, FOCUS attempts to formulate an SQL JOIN to let the DBMS do the JOIN. This is almost always the more efficient method, as both DB2 and SQL/DS have their own optimizers whose function is to pick the most efficient access path that satisfies an SQL request. SQL requests do not specify how to access the data; it is strictly up to the DBMS to decide in what order to access tables and whether to use any indexes that may be present. The more information the DBMS' optimizer has about what is wanted, the better able it is to make a good decision about how to retrieve the data. When FOCUS does the JOIN by reading each table individually, the optimizer is not given a chance to optimize the access path selection.

For example, consider the tables COURSE and EMPLOYEE, described in the SQL/DS and DB2 Read/Write Interface manual (the MFDs are shown in Figure 1 on page 4). The following TABLE request is issued:

```
JOIN EMP_NO IN COURSE TO ALL EMP_ID IN EMPLOYEE AS J1
TABLE FILE COURSE
PRINT  COURSE_NAME
IF LAST_NAME EQ 'JOHNSON'
END
```

With optimization off, FOCUS will first read a row (record) from the COURSE table, then read the EMPLOYEE table for the matching EMP_NO, and lastly test the value of LAST_NAME. This process is then repeated for every row in COURSE. The actual SQL statments that would be issued are:

```
SELECT COURSE_NAME, EMP_NO
        FROM "USERID"."COURSE"

SELECT LAST_NAME
        FROM "USERID"."EMPLOYEE"
        WHERE EMP_ID = ?
```

where the "?" in the second SQL SELECT statement is resolved to the value of EMP_NO obtained from the first SQL SELECT statement.

Figure 1

```
FILENAME=EMPLOYEE, SUFFIX=SQLDS,
SEGNAME=EMPLOYEE, SEGTYPE=S0,
  FIELD=EID,        ALIAS=EMP_ID,        A9,    A9,    MISSING=OFF,$
  FIELD=LN,         ALIAS=LAST_NAME,     A15,   A15,   MISSING=OFF,$
  FIELD=FN,         ALIAS=FIRST_NAME,    A10,   A10,   MISSING=OFF,$
  FIELD=HDATE,      ALIAS=HIRE_DATE,     I4,    I4,    MISSING=OFF,$
  FIELD=DEPT_CODE,  ALIAS=,              A3,    A3,    MISSING=ON,$
  FIELD=CUR_SAL,    ALIAS=,              P9.2,  P5,    MISSING=OFF,$
  FIELD=JOB_CODE,   ALIAS=JOB_CODE,      A3,    A3,    MISSING=OFF,$
  FIELD=ED_HRS,     ALIAS=ED_HRS,        I4,    I4,    MISSING=ON,$
  FIELD=BONUS_PLAN, ALIAS=BONUS_PLAN,    I2,    I2,    MISSING=OFF,$

FILENAME=COURSE,  SUFFIX=SQLDS,
SEGNAME=COURSE,   SEGTYPE=S0,
  FIELD=CNAME,      ALIAS=COURSE_NAME,   A15,   A15,   MISSING=OFF,$
  FIELD=WHO,        ALIAS=EMP_NO,        A9,    A9,    MISSING=OFF,$
  FIELD=GRADE,      ALIAS=GRADE,         A1,    A1,    MISSING=ON,$
  FIELD=YR_TAKEN,   ALIAS=YR_TAKEN,      A2,    A2,    MISSING=OFF,$
  FIELD=QTR,        ALIAS=QUARTER,       A1,    A1,    MISSING=OFF,$
```

With optimization on, this same TABLE request would be satisfied by issuing this SQL statement:

```
SELECT COURSE_NAME, LAST_NAME
FROM "MYID"."COURSE" T1
   ,"MYID"."EMPLOYEE" T2
WHERE T1.EMP_NO = T2.EMP_ID
   AND T2.LAST_NAME = 'JOHNSON'
```

Since the DBMS would be responsible for all record and access path selections, it might decide to first look up the employe whose name is 'JOHNSON' and then find all the matching employees in COURSE. This is potentially a much more efficient process, especially if there are indexes on LAST_NAME and EMP_NO. Overhead would also be reduced as communications traffic between the DBMS and FOCUS would be lessened, since fewer rows would have to be returned to FOCUS' address space.

Even if a multi-segment structure had been employed instead of the JOIN command to tell FOCUS to relate EMPLOYEE and COURSE as parent and child, FOCUS would have performed identically. With optimization on, FOCUS would issue an SQL JOIN; with optimization off, it would read each table individually.

Part of the general optimization of SQL calls is the ability to offload sorts and aggregations to the DBMS, as well as joins. This is important principally because if the DBMS knows that the data is needed in sorted order, it may choose to access it via an index on the sort key (should one be present), thereby avoiding the overhead of a sort. Since the offloading of sorts and aggregations is really a function of whether optimization is on or off, keep in mind that all of the points concerning the optimization of joins also apply to sorts and aggregations.

The third method of joining tables is to create an SQL view and let FOCUS TABLE off the view. This guarantees that the DBMS will perform the JOIN regardless of optimization, since FOCUS will not even know that there are tables to be JOINed (the MFD and FOCSQL for the SQL view EMPCOURSE, as well as the SQL statement that created the view, are shown in Figure 2). Because of this fact and because, under Interface Release 1, it was the only method available for getting the DBMS to do the join (Release 1 could not construct SQL joins), many shops have come to rely upon this method of getting the DBMS to do the JOIN. However, now that the Interface is capable of issuing SQL joins, there is a good reason not to rely upon views as the principal method of getting the DBMS to do the JOIN; when a SQL view is read, all tables mentioned in the view are accessed, even if fields from only some of the tables are asked for. Consider this FOCEXEC:

Figure 2

```
FILENAME=EMPCOURSE, SUFFIX=SQLDS,
SEGNAME=EMPCOURSE, SEGTYPE=S0,
FIELD=EID,          ALIAS=EMP_ID,      A9,    A9,    MISSING=OFF,
FIELD=COURSE_NAME,  ALIAS=,            A15,   A15,   MISSING=OFF,
FIELD=LN,           ALIAS=LAST_NAME,   A15,   A15,   MISSING=OFF,
FIELD=FN,           ALIAS=FIRST_NAME,  A10,   A10,   MISSING=OFF,
FIELD=HDATE,        ALIAS=HIRE_DATE,   I4,    I4,    MISSING=OFF,
FIELD=DEPT_CODE,    ALIAS=DEPT_CODE,   A3,    A3,    MISSING=OFF,
FIELD=CUR_SAL,      ALIAS=CUR_SAL,     P9.2,  P5,    MISSING=OFF,
FIELD=JOB_CODE,     ALIAS=JOB_CODE,    A3,    A3,    MISSING=OFF,
FIELD=ED_HRS,       ALIAS=ED_HRS,      I4,    I4,    MISSING=OFF,
FIELD=BONUS_PLAN,   ALIAS=BONUS_PLAN,  I2,    I2,    MISSING=OFF,
FIELD=EMP_NO,       ALIAS=,            A9,    A9,    MISSING=OFF,
FIELD=GRADE,        ALIAS=,            A1,    A1,    MISSING=ON,
FIELD=YR_TAKEN,     ALIAS=,            A27,   A2,    MISSING=OFF,
FIELD=QUARTER,      ALIAS=,            A1,    A1,    MISSING=OFF,
END
SEGNAME=EMPCOURSE, TABLENAME=MYID.COURSE, WRITE=NO, KEYS=2,
END
CREATE VIEW EMPCOURSE
AS (COURSE_NAME, EMP_NO, GRADE, YR_TAKEN, QUARTER,
(EMP_ID, LAST_NAME, FIRST_NAME, HIRE_DATE, DEPT_CODE, CUR_SAL,
JOB_CODE, ED_HRS, BONUS_PLAN) AS
SELECT COURSE_NAME, EMP_NO, GRADE, YR_TAKEN, QUARTER
(EMP_ID, LAST_NAME, FIRST_NAME, HIRE_DATE, DEPT_CODE, CUR_SAL,
JOB_CODE, ED_HRS, BONUS_PLAN)
FROM EMPLOYEE T1, COURSE T2
WHERE T1.EMPNO = T2.EMP_ID
```

```
TABLE FILE EMPCOURSE
PRINT LAST_NAME
END
```

Even though LAST_NAME is contained only in the EMPLOYEE table, the full join of EMPLOYEE and COURSE will be carried out by the DBMS; for each row of EMPLOYEE, the corresponding row of COURSE will be retrieved. This can be a significant source of overhead, especially if a small number of large views are created for general reporting purposes, since even the most limited query will be forced to access all the underlying tables. If, on the other hand, a multi-table structure was used (the structure having been created either with a JOIN or through a multi-segment MFD/FOCSQL), no unnecessary tables would be accessed. When FOCUS reads from such a structure, only the tables containing referenced fields will be read from.

Additionally, in the SQL/DS environment only (not DB2), such large, general-purpose views can create other problems. First, a view in SQL/DS is limited as to the number of columns that it can have; it may not be possible to include all the fields that might be needed. Secondly, queries against large, multi-table views are generally more complicated, and shops still running SQL/DS 1.3.5 and below have a well-known risk of getting SQLCODE -101, "query too complex", in such situations. (This was corrected in Version 2, but many shops have not yet gone to this release.)

Automatic Disabling Of Optimization

There are four conditions under which FOCUS will automatically turn optimization off:

- If an "outer join" is requested (SET ALL=ON).
- If the TABLE request references data from more than a single path of the logical data structure.
- Where SUM or COUNT operations might return incorrectly replicated data, such as in the case of a SUM on a parent field or a child field of a many-to-many relationship.

- If the FST or LST operators are used and KEYS=0 in FOCSQL. This fact is not documented, but IBI has confirmed that these conditions will disable optimization.

While these circumstances may be unavoidable, the potential impact on performance is so great that their use should be minimized unless an actual test demonstrates that the effect on performance is minimal. Here are some ways to work around these restrictions:

1. Outer joins. An outer join is the same as a short path, and it is enabled by SET ALL=ON. The ability to do outer joins is a large benefit of using FOCUS with DB2 or SQL/DS data, since the SQL language does not fully support this type of join. In order to avoid the performance penalty caused by setting optimization off, however, it is advisable to use SET ALL=OFF as the default and only SET ALL=ON when the outer JOIN is absolutely needed.

   Of particular concern are situations involving three or more tables where the missing data is in the lowest child only, since OPTIMIZATION OFF will completely disable FOCUS' ability to issue an SQL join of the top two segments. In this case, it is advisable to create an SQL view of the top segments and use FOCUS to JOIN the view with the child (with ALL=OFF). For example, suppose that in addition to the COURSE and EMPLOYEE tables, there was a third table, DEPENDANTS, with information about each employee's dependants. This would be a good candidate for an outer join, since not all employees have dependants. What one would like to see happen is for FOCUS to let the DBMS join COURSE and EMPLOYEE but do the look-ups against DEPENDANTS itself:

```
A.  SELECT COURSE_NAME, LAST_NAME
    FROM "MYID"."COURSE" T1
        ,"MYID"."EMPLOYEE" T2
    WHERE T1.EMP_NO = T2.EMP_ID
        AND T2.LAST_NAME= 'JOHNSON'

B.  SELECT SPOUSE_NM
    FROM DEPENDANTS
      WHERE EMPLOYEE = ?
```

where the "?" in B is resolved to the value of EMP-NO in A.

Unfortunately, setting ALL=ON will prevent FOCUS from issuing the SQL join statement in A; instead, FOCUS would read the COURSE and EMPLOYEE tables separately:

```
A.    SELECT COURSE_NAME, EMP_NO
      FROM "USERID"."COURSE"

B.    SELECT LAST_NAME
      FROM "USERID"."EMPLOYEE"
      WHERE EMP_ID = ?

C.    SELECT SPOUSE_NM
      FROM DEPENDANTS
      WHERE EMPLOYEE = ?
```

Because this may result in unacceptable performance, it would be a good idea to create a view of the join of COURSE and EMPLOYEE (e.g., EMPCOURSE), and issue a FOCUS JOIN between EMPCOURSE and DEPENDANTS (shown in Figure 3).

2. Multiple Paths. Assume that the COURSE table, which contains the EMP_NO and COURSE_NAME for all courses and employees, is used to associate EMPLOYEE with a third table, CLASSES, containing a one row description for each course. In SQL, the many-to-many relationship would be expressed as:

```
SELECT LAST_NAME,
EMP_NO, COURSE_DESCR
FROM EMPLOYEE T1,
```

Figure 3

```
SET ALL=ON
JOIN EMP_NO IN EMPCOURSE TO ALL EMPLOYEE IN DEPENDANTS
TABLE FILE EMPCOURSE
PRINT COURSE_NAME LAST_NAME
SPOUSE_NM
IF LAST_NAME EQ 'JOHNSON'
```

```
    COURSE  T2,
    CLASSES T3
WHERE T2.EMP_NO = T1.EMP_ID
    AND T2.COURSE_NAME=T3.COURSE_NM
```

Note: Since FOCUS Release 6.0 will accept SQL syntax, this could be issued from within the TABLE command. In FOCUS, the three tables could be JOINed together in a similar manner, as shown in Figure 4.

However, doing so creates a multi-path structure, and hence will turn optimization off if fields from all three tables are requested. In this instance, it would make sense to create an SQL view of the three tables and TABLE off the view. Another way to handle this situation would be to create an SQL view of one of the paths and JOIN this view to the remaining path with OPTIMIZATION ON (see Figure 5).

3. SUM and COUNT Operations. The use of these operators on fields that might be replicated in an SQL join (the multiplicative effect) will disable optimization. More specifically, optimization will be disabled when a SUM or COUNT is used on (a) any parent or child field in a many-to-many relationship, (b) a parent field in a one-to-many relationship, or (c) a child field in a many-to-one relationship. How FOCUS determines the nature of this relationship between segments is not documented, but IBI has confirmed that it is at least partly dependent upon the use of the KEYS parameter inthe FOCSQL. For this reason, it is important to make sure that, where possible, all tables have a unique index and that the KEYS parameter is correctly coded.

Unfortunately, there are no good ways around these restrictions. The creation of a view would not solve the problem, since that would just create the problem that FOCUS was trying to stay away from in the first place. About all that one can do is analyze each application on a case-by-case basis to see if the resulting performance penalty is so severe as to preclude the use of these operators.

4. FST and LST Operators. If the FST. or LST. operators are used, and the FOCSQL contains KEYS=0, then optimization will be disabled (verified under FOCUS 5.5, Interface Release 3, VM environment). The only way around this is to make sure that all tables have a unique index and code the KEYS parameter accordingly. Unfortunately, not all applications allow for this, and in such cases (as with the SUM and COUNT operators), there is no easy way around the restriction. Once again, the utility of these functions must be weighed against the possible performance problems that may ensue as a result of their use.

Tests On DEFINEd Fields

IF tests made on DEFINEd fields will not be passed to the DBMS under FOCUS 5.5. This is among the most important limitations of the Interface, since it directly affects the ability of the DBMS' optimizer to pick a good access path. For example:

Figure 4

```
JOIN EMP_NO IN COURSE TO ALL EMP_ID IN EMPLOYEE AS J1
JOIN COURSE_NAME IN COURSE TO ALL COURSE_NM IN CLASSES AS J2
TABLE FILE COURSE
PRINT LAST_NAME EMP_NO COURSE DESCR
END
```

Figure 5

```
JOIN EMP_ID IN EMPCOURSE TO ALL COURSE_NM IN CLASSES
TABLE FILE EMPCOURSE
PRINT LAST_NAME EMP_NO COURSE DESCR
END
```

```
DEFINE EMPLOYEE
OK/I1=IF JOB_CODE EQ 'XXX' AND
BONUS_PLAN GT 30 THEN 1 ELSE IF
JOB_CODE EQ 'BBB' AND BONUS_PLAN GT
60 THEN 1 ELSE IF JOB_CODE EQ 'CCC'
AMD BONUS_PLAN GT 80 THEN 1 ELSE 0 ;
END
TABLE FILE EMPLOYEE
PRINT LAST_NAME
IF OK IS 1
END
```

The Interface will not pass any of the JOB_CODE or BONUS_PLAN tests to the DBMS, so that FOCUS would end up having to read the entire EMPLOYEE table! A better way to code the FOCEXEC would be to include redundant IF tests directly on DBMS fields, in addition to those applied against DEFINEd fields:

```
DEFINE EMPLOYEE
OK/I1=IF JOB_CODE EQ 'XXX' AND
BONUS_PLAN GT 30 THEN 1 ELSE IF
JOB_CODE EQ 'BBB' AND BONUS_PLAN GT
60 THEN 1 ELSE IF JOB_CODE EQ 'CCC'
AMD BONUS_PLAN GT 80 THEN 1 ELSE 0 ;
END
TABLE FILE EMPLOYEE
PRINT LAST_NAME
IF OK IS 1
-* Redundant IF tests on DBMS fields
-*follow
IF JOB_CODE EQ 'XXX' OR 'BBB' OR 'CCC'
IF BONUS_PLAN GT 30
END
```

In this example, the redundant IF tests are incomplete by themselves since they do not fully capture the logical relationships that are needed. Nevertheless, they can be translated into SQL, and this may allow the DBMS' optimizer to do a much better job than if it knew nothing about what JOB_CODEs and BONUS_PLAN values were needed. For instance, if there happened to be a compound index built on JOB_CODE and BONUS_PLAN, access could become very efficient indeed.

IBI has announced that, in FOCUS 6.0, the Interface will support the translation into SQL of many types of tests on DEFINEd fields. Though a complete description of this enhancement is beyond the scope of this article (not to mention premature), it will not support DEFINE-based fields that rely upon IF...THEN...ELSE constructs or a number of other types of constructs. For this reason, it is highly advisable that the FSTRACE facilities always be used to verify that the SQL calls produced by the Interface are as information rich as possible and to add redundant IF tests when they are needed to improve performance.

DEFINEd Fields And Sorts

There is one unusual set of circumstances in which tests on DEFINEd fields combined with OPTIMIZATION ON may result in terrible performance. Suppose that the following conditions are met:

1. The FOCEXEC contains IF tests on DEFINED fields that do not make it into the SQL call. As a result, a very large number of rows are returned to FOCUS, even though just a few of them will actually pass the IF tests on the DEFINED fields.

2. The FOCEXEC contains BY fields, translated into SQL ORDER BY clauses.

In such a situation, FOCUS will offload the sort to the DBMS, forcing it to sort all the rows satisfying the SQL call, though just a few of them are really part of the answer set. If the number of rows that the DBMS is sorting is very large, but the number of rows satisfying the IF tests is very small, it might be cheaper to turn OPTIMIZATION OFF since that would prevent the DBMS from having to sort a large number of rows. Of course, whether the performance gain of eliminating the large sort is worth the penalty of OPTIMIZATION OFF is dependant on the particular applications and must be judged carefully.

RECORDLIMIT And READLIMIT Tests

One of the most frustrating experiences for the FOCUS programmer is to TABLE off DB2 or

SQL/DS data and then have to wait a long time for a small number of records specified in the RECORDLIMIT or READLIMIT to be reached. The secret to understanding why this happens is to have an understanding of when the DBMS is able to return the first row of data to FOCUS. DB2 and SQL/DS process data a full set at a time, and there is no way to tell the DBMS to produce only a subset, N rows long, of the full answer set. RECORDLIMIT tests in FOCUS are implemented by only reading (FETCHing) the first N rows of the answer set, even though the DBMS has the full answer set available. If the DBMS can produce the answer set without having to produce any intermediate tables, this will not be a problem, since the DBMS can and will satisfy each FETCH by reading the data directly. The problem arises when intermediate tables must be produced as a result of sorting or complex joins, since these intermediate tables must be produced before the DBMS can satisfy the first FETCH command from FOCUS.

Sorts occur when the DBMS needs an ordered representation of the data and the optimizer cannot, or does not, use an index to accomplish this. In such cases, the DBMS must retrieve all the rows satisfying any WHERE conditions first, then sort them. An ordered representation of the data is needed whenever an ORDER BY or GROUP BY (used in aggregation) is specified, and may also be required for certain types of joins. Recall that with optimization on, FOCUS will offload sorts and aggregations to the DBMS whenever possible.

Even without using ORDER BY/GROUP BY, other types of intermediate tables may have to be produced in order to resolve *n*-way joins, where *n* is greater than two. Both DB2 and SQL/DS do *n*-way joins as a series of two-way JOINs; the intermediate tables, called composites, may have to be produced in full before the DBMS is ready to hand over the first row.

As long as none of the underlying tables are really multi-table views, the best way around this problem is, surprisingly, to set optimization off. With optimization off, FOCUS will do the reads of the underlying tables individually, and the RECORDLIMIT/READLIMIT will apply against the individual reads (see the FOCUS Users Manual for a precise description of the differences between RECORDLIMIT and READLIMIT). Of course, if one of the underlying tables is really a muliti-table SQL view and the resolution of this JOIN requires that the DBMS produce intermediate tables, then setting optimization off will not solve the problem, though it might still help. This is yet another reason to avoid relying on views as the main way of getting the DBMS to carry out joins.

Use Of COMMIT/ROLLBACK WORK In MODIFY Processing

The use of MODIFY with DB2 or SQL/DS data includes two special commands, SQL COMMIT WORK and SQL ROLLBACK WORK. Normally, these commands are used to COMMIT or ROLLBACK on a transaction-by-transaction basis. For example:

```
ON MATCH UPDATE
ON MATCH SQL COMMIT WORK
ON NOMATCH SQL ROLLBACK WORK
ON NOMATCH REJECT
```

However, regardless of whether these commands are included in the FOCEXEC, the Interface will always issue a COMMIT at the end of the MODIFY. When more than one transaction is to be handled, using a COMMIT or ROLLBACK after each transaction requires a tremendous amount of overhead; allowing the COMMIT to be issued by FOCUS just once at the end of the MODIFY does not.

FOCUS uses dynamic SQL to communicate with the DBMS; in dynamic SQL, every SQL statement (SELECT, UPDATE, INSERT, DELETE) must undergo two processing steps prior to actually doing any work, first a PREPARE step and then either an OPEN (for SELECT statements) or EXECUTE (for non-SELECT statements). The PREPARE step tells the DBMS' optimizer to find an access path. This can entail a significant amount of overhead, as it involves accessing the catalogs and comparing alternative access paths. The OPEN or EXECUTE is what actually tells the DBMS to run the PREPAREd statement.

Figure 6

```
-RUN
-TYPE ... SQL ...
MODIFY MFILE ...
FIXFORM ... DEPTNUM/A3 ...
MATCH DEPT ...
ON MATCH UPDATE DEPTNUM ...
ON MATCH SQL COMMIT WORK ...
ON NOMATCH SQL ROLLBACK WORK ...
ON NOMATCH REJECT ...
DATA ON MODESTAT ...
END
```

In MATCH processing, at least two SQL statements will always have to be PREPAREd: a SELECT statement (to read the data, looking for a MATCH) and a non-SELECT statement (UPDATE, INSERT, DELETE) to carry out the update. The important thing to know is that all PREPAREd statements are destroyed whenever a COMMIT or ROLLBACK is issued and must be re-PREPAREd in order to be used again. For example, if a transaction file of 100 transactions is to be applied against the database and a COMMIT or ROLLBACK is issued after each MATCH/NOMATCH, there will be 100 PREPAREs of the select statement and 100 PREPAREs of the update statement. If, on the other hand, the COMMIT is issued just once at the end of the MODIFY, there would only be one PREPARE of the select statement and one PREPARE of the update statement.

The effect of issuing COMMIT or ROLLBACK after each transaction can be seen quite clearly in the following example. A FIXFORM of 100 records is used to update a small database, and each transaction record contains the same update to the same record (i.e., all 100 transactions are identical). The FOCEXEC is shown in Figure 6. The data file, INDATA, contains 100 of these records:

Figure 7

With COMMIT or ROLLBACK after each transaction	Without COMMIT or ROLLBACK after each transaction
17.0	8.9
16.0	6.9
15.9	7.3
15.8	7.4
16.6	9.4
16.6	6.8

20 20
20 20
20 20 etc.
(Note: All 100 records MATCHed a row in the database.)

Testing was done in a VM environment using FOCUS 5.5.3, Interface Release 3, and SQL/DS 1.3.5. Six trials were done with COMMIT/ROLLBACK after each transaction and without (i.e., FOCUS issues the COMMIT just once at the end of the MODIFY). The results, reported in elapsed seconds, are shown in Figure 7. Quite clearly, the effect of having to re-PREPARE the SELECT and UPDATE statements repeatedly is very severe.

Whether or not it is actually feasible to issue the COMMIT at the end of many transactions is entirely dependent upon the application. If transactions are being entered online by a CRT operator, it would probably not be acceptable to group them in this way; doing so would cause the DBMS to hold locks for long periods of time, and a ROLLBACK done for any reason would cause the loss of all input. On the other hand, if the transactions were from an input file, it probably would be appropriate to hold the COMMIT until the end.

Suggestions For Future Enhancements

There are three enhancements that would be of immediate help to FOCUS programmers.

1. Enhance the FSTRACE facility so that it explains why OPTIMIZATION has been disabled. There is currently no way to tell exactly why FOCUS has set OPTIMIZATION OFF except by one's own analysis. This can be maddening when FSTRACE3 says that aggregation was not performed because OPTIMIZATION was disabled, without explaining why it was disabled in the first place.

2. Allow the use of FST. and LST. and KEYS=0 without disabling OPTIMIZATION.

Finally, and most importantly:

3. Let SQL statements be coded directly in the FOCEXEC, and have these statements be passed to the DBMS without translation. Version 6.0 of FOCUS does allow the programmer to include SQL statements explicitly, but they will not be passed directly to the DBMS. Instead, they will first be translated into the FOCUS language, and then passed to the Interface for translation into SQL, subject to the regular FOCUS-SQL translation restrictions. Allowing SQL statements to be passed directly to the DBMS would, by itself, solve most, if not all, of the current deficiencies of the Interface.

Conclusion

Performance tuning in any environment can be time consuming and tedious, but it is not a black art. The important thing to understand is that, in almost all cases, the goal is to get the DBMS to do the work, and this means monitoring what the DBMS is being called upon to do. The Interface provides three FSTRACE facilities to do this (FSTRACE, FSTRACE3,and FSTRACE4 in FOCUS 6.0), and they should always be used to verify that the DBMS is getting the best SQL call that can be formulated, with particular attention paid to these issues:

- If tables are being joined, who is joining them, the DBMS or FOCUS, and if FOCUS is doing the join, what is preventing FOCUS from issuing an SQL join?
- Are important IF tests making it into the SQL call?
- When SQL views are being used, are any of the underlying tables being accessed unneccessarily? Can a FOCUS multi-segment structure be used instead?
- In MODIFY processing, is it possible to COMMIT WORK at the end of the MODIFY?

By following these guidelines, one can isolate performance issues to either the FOCUS side or to the DBMS side, and take the appropriate actions, thereby combining the best features of both FOCUS and the DBMS.

Measuring Code Efficiency

Bernard J. Blick
The Bank of New York Company, Inc.

Several solutions to a coding problem are always available; however, it is not always readily apparent which solution will be the most efficient. This article demonstrates several ways to quantitatively measure and compare the different programs possible in a given situation.

Many FOCUS users are aware of the substantial benefits that FOCUS provides in gaining access to files created by an external DBMS. In most cases, programmers of modest experience levels are able to use native FOCUS code to access logical files composed of mixed database types (FOCUS, IMS, DB2, VSAM, IDMS, etc.) and produce sophisticated reports with relative ease. What is not so obvious is the manner in which FOCUS navigates through the underlying DBMS, accesses the data, aggregates, sorts, and does the myriad other tasks necessary to produce the desired output from the non-procedural code that we give it. We want to determine the best ways to process in these complex environments, especially in cases where programs will be executed numerous times each day as production queries. Is there some way that we can obtain quantitative measurements for the relative efficiency of the alternate coding structures that can be created to produce our desired requests?

The &FOCCPU Variable

Fortunately, a solution to this dilemma is facilitated through a new Dialogue Manager variable that first appeared in Release 5.5 of the IBM mainframe product. The variable, &FOCCPU, gives the elapsed CPU time for the current session in milliseconds (under MVS). When two different FOCEXECs that produce the same result are sandwiched between Dialogue Manager –TYPE statements that reference &FOCCPU, we can essentially measure the relative CPU efficiency of the two programs. Figure 1 depicts this 'bare bones' approach.

For the purposes of this discussion, two FOCEXECs, ONESTEP and TWOSTEP (Figures 2 and 3, respectively on pages 13 and 14), will be studied. They represent alternate ways to code the same report query. These particular FOCEXECs access data stored in four DB2 tables, but it should be noted that the method described here will work in any FOCUS environment and with any combination of external interfaces. (Of course, the appropriate FOCUS allocations and set up commands for the external interfaces should be issued prior to attempting to execute the programs and take measurements.) In addition, it should be noted that while this method is particularly useful in complex interface situations, it is not limited to them and can be used for simple native FOCUS requests.

Figure1

```
-TYPE &PROGRAM1 START = &FOCCPU
-RUN
EX &PROGRAM1
-RUN
-TYPE &PROGRAM1 END   = &FOCCPU
-RUN
-TYPE &PROGRAM2 START = &FOCCPU
-RUN
EX &PROGRAM2
-RUN
-TYPE &PROGRAM2 END   = &FOCCPU
-RUN
-*------------------------------------
EX RUNTEST PROGRAM1=ONESTEP, PROGRAM2=TWOSTEP
-*------------------------------------
ONESTEP START = 10335
ONESTEP END   = 124121  /  113786 ms.

TWOSTEP START = 124123
TWOSTEP END   = 125914  /    1791 ms.
```

The program ONESTEP JOINs four DB2 tables, DEFINEs two temporary fields against the JOINed file, and then extracts the data and produces a formatted report. The program TWOSTEP takes a different approach. First, it JOINs only three tables that comprise the bulk of the data to be extracted. The data is then extracted from DB2 via a TABLE that uses no FOCUS functionality for which there is no exact SQL equivalent. Only summation, sorting, and simple selection are used. Features such as DEFINEd fields, COMPUTEd fields, multiple verbs, report formatting, and the like are avoided. The result is stored in a HOLD file. The remaining DB2 table, which in this case is a small code translation table, is JOINed to the HOLD file. The two DEFINEd fields are then created against the much smaller HOLD file, which serves as the object file for a second TABLE request that produces the final formatted report. Program TWOSTEP is certainly longer, takes more steps, and is more complex a request than is ONESTEP, but how do they stack up against each other in terms of efficiency?

Figure 1 also shows the execute statement required to run the 'bare bones' program RUNTEST against our two alternative coding structures and displays the results of the test. RUNTEST supplies only the start and end CPU times for the programs; it is up to us to perform

Figure 2

```
-*PROGRAM ONESTEP - PRODUCES A REPORT FROM DB2 TABLES
JOIN CLEAR
JOIN GLOBAL_ID IN JGLOB TO ALL GLOBAL_ID IN JCOMP AS J1
JOIN IDIV_CD   IN JGLOB TO DIVN_COD      IN JDIVN AS J2
JOIN CUST_ID10 IN JGLOB TO ALL CUST_ID   IN JACNTS AS J3
JOIN CATEG_ID  IN JGLOB TO CATEG_ID      IN JCATS AS J4
-*
DEFINE FILE JGLOB
CATEG_GNM1/A14 = EDIT(CATEG_GNM1,'99999999999999');
INST_NAME/A24  = EDIT(INST_NAME,'999999999999999999999999');
END
TABLE FILE JGLOB
HEADING CENTER
"<110 07/25/88"
"<DIV_NAME"
"INSTITUTION SUMMARY REPORT"
"IN ALPHABETICAL ORDER"
" "
SUM COMPUTE CONTR/P10C = INTR_INC + FEE + BALCMP + ERN_CR - INTR_EXP
                         PROD_EXP - FDIC + TX_EX_REV AS ' '
AND ROW-TOTAL AND COLUMN-TOTAL
ACROSS CATEG_GNM1 AS ' '
BY INST_NAME BY GLOBAL_ID AS 'GLOBAL,ID'
IF DIVN_COD EQ 050
END
```

Figure 3

```
-*PROGRAM TWOSTEP
-*PRODUCES THE SAME REPORT AS PROGRAM ONESTEP
JOIN CLEAR
JOIN GLOBAL_ID IN JGLOB TO ALL GLOBAL_ID IN JCOMP AS J1
JOIN CUST_ID10 IN JGLOB TO ALL CUST_ID IN JACNTS AS J3
JOIN CATEG_ID IN JGLOB TO CATEG_ID IN JCATS AS J4
-*
TABLE FILE JGLOB
SUM  INTR_INC  FEE  BALCMP  ERN_CR  INTR_EXP
     PROD_EXP  FDIC  TX_EX_REV
BY CATEG_GNM1 BY INST_NAME
BY GLOBAL_ID BY IDIV_CD
IF IDIV_CD EQ 050
ON TABLE HOLD
END
-RUN
-*
JOIN IDIV_CD IN HOLD TO IDIVN_CD IN JGLOBDIV AS J2
-*
DEFINE FILE HOLD
CATEG_GNM1/A14 = EDIT(CATEG_GNM1,'99999999999999');
INST_NAME/A24  = EDIT(INST_NAME,'999999999999999999999999');
END
-*
TABLE FILE HOLD
HEADING CENTER
"<110 07/25/88"
"<IDIV NAME"
"INSTITUTION SUMMARY REPORT"
"IN ALPHABETICAL ORDER"
" "
SUM
COMPUTE CONTR/P10C = INTR_INC + FEE + BALCMP + ERN_CR - INTR_EXP -
                     PROD_EXP - FDIC + TX_EX_REV; AS ' '
AND ROW-TOTAL AND COLUMN-TOTAL
ACROSS CATEG_GNM1 AS ' '
BY INST_NAME BY GLOBAL_ID AS 'GLOBAL_ID'
END
-RUN
```

the subtraction to obtain the elapsed CPU milliseconds for each run. The results demonstrate, to our amazement, that the more complex request, TWOSTEP, ran in only 1.7 CPU seconds against 113.7 CPU seconds for ONESTEP. Another way of looking at it is that we could run 63.5 query transactions using TWOSTEP for every one transaction using ONESTEP, an obvious boon if this were a popular production request.

A Better Solution: The CPU Efficiency Comparator

The simple approach described above works very well, but it does not supply much in the way of the amenities that we have come to expect in the days of user-friendly programs. Data entry could be facilitated by provision for full screen CRTFORMs. On the output side, there is no reason why the computer cannot compute the elapsed CPU times and offer benefit comparisons between the measured coding alternatives. Additionally, we want to minimize any bias that may arise from the sequence in which the programs are tested, costs only encountered the first time a FOCEXEC is run, and the expected variations that may be obtained across multiple executions of the programs.

In order to accomplish all of the above, we can create an easy-to-use shell program, the CPU Efficiency Comparator (the CPUCOMP FOCEXEC shown in Figure 4).

Screen 1

```
                    FOCEXEC CPU EFFICIENCY COMPARATOR

PLEASE ENTER:

      THE NAME OF THE 1ST TEST FOCEXEC:

      THE NAME OF THE 2ND TEST FOCEXEC:

      THE NUMBER OF TEST RUNS TO PERFORM:    01

      DO YOU WISH INTERMEDIATE RESULTS:      N

THE FOCEXECS THAT ARE ENTERED SHOULD BE ALTERNATE CODING STRUCTURES
 THAT PROVIDE THE SAME END RESULTS FOR THIS TEST TO BE MEANINGFUL.
        THE NUMBER OF TEST RUNS MAY BE BETWEEN 1 AND 10
```

CPUCOMP provides an input screen that prompts for the names of the two alternate FOCEXECs. Two additional fields prompt for the number of test runs to be applied to each program and whether the intermediate result for each run should be presented after each pair of comparison runs is completed or only at the end as part of an overall report. The default for the number of runs is '1', and the default for displaying intermediate results is 'N' for no (see Screen1). The valid range for the number of test runs is from 1 to 10 per execution of CPUCOMP. The input fields are checked for validity, and the appropriate message is moved to the input screen as necessary.

To compensate for any bias due to startup conditions, the first test program is executed and the results are discarded. Thus, if one of the programs is suspected to be vastly more efficient than another, some execution time could be saved by specifying that program as the first test FOCEXEC. Next, the pair of programs is put through code similar to the 'bare bones' example, and the CPU elapsed times and percentage benefits are calculated. The FOCEXEC name and the associated times are written out to a file via the -WRITE statement, and the file is used to update a simple temporary FOCUS database (Figure 5 on page 19). This database will be used to produce the final report. If intermediate results are requested, then a screen will be presented that will show the CPU results and computations for that particular run of the pair of FOCEXECs. Execution of the next test run and/or the final report

(continued on page 18)

Figure 4

```
-CPUCOMP PROGRAM - ONLINE VERSION
-A*****************************************************
-HOUSEKEEPING
-*****************************************************
-INITS
TSO ALLOC F(SAVEX) DS(TEMPFEXC.DATA(SAVEX)) SHR REUSE
CREATE FILE TIMESTAT
SET MSG OFF, PRINT=OFFLINE
-DEFAULTS &RUNS=01,&CTR=01,&TEST1='          &TEST2=
-DEFAULTS &ERROR='
-DEFAULTS &X=' ',&FOUND=' ',&INTER='N'
-RUN
-*****************************************************
-MAIN INPUT SCREEN (IDENTIFY PROGRAMS, # OF RUNS, AND NEED FOR
-                  INTERMEDIATE RESULTS FOR EACH RUN)
-*****************************************************
-DEFSCR
```

(continued)

Figure 4 (continued)

```
-CRTCLEAR
-CRTFORM LINE 1
-" "
-"                    FOCEXEC CPU EFFICIENCY COMPARATOR"
-" "
-" "
-" PLEASE ENTER:"
-" "
-"        THE NAME OF THE 1ST TEST FOCEXEC:   <T.&TEST1>"
-" "
-"        THE NAME OF THE 2ND TEST FOCEXEC:   <T.&TEST2>"
-" "
-"        THE NUMBER OF TEST RUNS TO PERFORM: <T.&RUNS>"
-" "
-"        DO YOU WISH INTERMEDIATE RESULTS:   <T.&INTER>"
-" "
-" "
-"  THE FOCEXECS THAT ARE ENTERED SHOULD BE ALTERNATE CODING STRUCTURES"
-"   THAT PROVIDE THE SAME END RESULTS, FOR THIS TEST TO BE MEANINGFUL."
-"            THE NUMBER OF TEST RUNS MAY BE BETWEEN 1 AND 10"
-" "
-"                  <24 <.HD.&ERROR> <.CD.&X>"
-RUN
-*********************************************************************
-* TESTS FOR VALID SCREEN INPUT
-*********************************************************************
-IF &TEST1   IS ' ' THEN GOTO EFFERR1;
-IF &TEST2   IS ' ' THEN GOTO EFFERR2;
-IF &RUNS.TYPE NE N THEN GOTO EFFERR3;
-IF &RUNS LT 01 OR &RUNS GT 10 THEN GOTO EFFERR3;
-IF &INTER NE 'N' OR 'Y' THEN GOTO EFFERR4 ELSE GOTO TESTFIND;
-EFFERR1
-SET &ERROR = 'PLEASE FILL IN PGM NAME 1';
-GOTO EFFSCR1
-EFFERR2
-SET &ERROR = 'PLEASE FILL IN PGM NAME 2';
-GOTO EFFSCR1
-EFFERR3
-SET &ERROR = 'RUNS MUST BE FROM 1 TO 10';
-GOTO EFFSCR1
-EFFERR4
-SET &ERROR = 'INTER RESULTS MUST BE Y/N';
-GOTO EFFSCR1
-TESTFIND
-TSO RUN FINDMEM,CHKFOCU,&TEST1,&FOUND
-RUN
-IF &FOUND IS Y THEN GO TO NEXTFIND;
-SET &ERROR = 'PGM 1 NOT IN YOUR LIBRARY';
-GOTO EFFSCR1
-NEXTFIND
-TSO RUN FINDMEM,CHKFOCU,&TEST2,&FOUND
-RUN
-IF &FOUND IS Y THEN GO TO BEFOREIT;
-SET &ERROR = 'PGM 2 NOT IN YOUR LIBRARY';
-GOTO EFFSCR1
-BEFOREIT
-TYPE PROCESSING ........ PLEASE BE PATIENT!
-RUN
-*********************************************************************
-* EXECUTE PROGRAM ONCE & THROW AWAY RESULTS (MIN. STARTUP CONDITIONS)
-*********************************************************************
EX &TEST1
-RUN
-GOTOIT
-TYPE PROCESSING ....... PLEASE BE PATIENT!
```

(continued)

Figure 4 (continued)

```
-RUN
-*********************************************************************
-* MAIN PART OF PROGRAM (DOES THE ACTUAL MEASUREMENT FOR BOTH PROGRAMS)
-*********************************************************************
-SET &START1 = &FOCCPU;
-RUN
EX &TEST1
-RUN
-SET &END1 = &FOCCPU;
-RUN
-SET &START2 = &FOCCPU;
-RUN
EX &TEST2
-RUN
-SET &END2 = &FOCCPU;
-RUN
-*********************************************************************
-* COMPUTE ELAPSED CPU TIME, PCT BENEFIT, AND WRITE TO SAVE FILE
-*********************************************************************
-SET &CPURUN1 = &END1 - &START1;
-SET &CPURUN2 = &END2 - &START2;
-SET &BENEFIT1 = (&CPURUN1 - &CPURUN2) * 100 / &CPURUN1;
-SET &BENEFIT2 = (&CPURUN2 - &CPURUN1) * 100 / &CPURUN2;
-WRITE SAVEX &TEST1&COMMA&CPURUN1
-WRITE SAVEX &TEST2&COMMA&CPURUN2
-RUN
-*********************************************************************
-* UPDATE TEMP DATABASE WITH RESULTS OF RUN
-*********************************************************************
MODIFY FILE TIMESTAT
FREEFORM PGM CPU
MATCH PGM CPU
ON MATCH INCLUDE
ON NOMATCH INCLUDE
DATA ON SAVEX
END
-RUN
-*********************************************************************
-* TEST IF INTERMEDIATE RESULTS FOR EACH RUN WAS REQUESTED
-*********************************************************************
-IF &INTER IS 'N' THEN GOTO TESTCNTR;
-RUN
-*********************************************************************
-* CONDITIONALLY PRODUCE SCREEN CONTAINING INTERMEDIATE RESULTS FOR RUN
-*********************************************************************
-EFFSCR2
-CRTCLEAR
-CRTFORM BEGIN LINE 1
-" "
-"                    FOCEXEC CPU EFFICIENCY COMPARATOR"
-" "
-" "
-"    TEST RUN #  <D.&CTR "
-" "
-" "
-"        <D.&TEST1 , START TIME, CPU MILLISECONDS = <D.&START1 "
-"        <D.&TEST1 , END TIME  , CPU MILLISECONDS = <D.&END1 "
-"        <D.&TEST1 , ELASPSED  , CPU MILLISECONDS = <D.&CPURUN1 "
-" "
-"        <D.&TEST2 , START TIME, CPU MILLISECONDS = <D.&START2 "
-"        <D.&TEST2 , END TIME  , CPU MILLISECONDS = <D.&END2 "
-"        <D.&TEST2 , ELASPSED  , CPU MILLISECONDS = <D.&CPURUN2 "
-IF &BENEFIT1 GE 0 THEN GOTO BENE1 ELSE GOTO BENE2;
-BENE1
```

(continued)

Figure 4 (continued)

```
-"BENEFIT OF <D.&TEST2 TO <D.&TEST1 ,IN PERCENT CPU TIME = <D.&BENEFIT1"
-GOTO EXITBENE
-BENE2
-"BENEFIT OF <D.&TEST1 TO <D.&TEST2 ,IN PERCENT CPU TIME = <D.&BENEFIT2"
-EXITBENE
-" "
-"                    PRESS ENTER TO CONTINUE"
-CRTFORM END
-RUN
-*********************************************************************
-* DETERMINE IF NUMBER OF REQUESTED RUNS HAS BEEN REACHED, LOOP IF NOT
-*********************************************************************
-TESTCNTR
-SET &CTR = &CTR + 1;
-IF &CTR GT &RUNS THEN GOTO STATS ELSE GOTO GOTOIT;
-*********************************************************************
-* PRODUCE REPORT FOR THE RESULTS OF ALL RUNS, AND BENEFIT ACHIEVED
-*********************************************************************
-STATS
TABLE FILE TIMESTAT
SUM   AVE.CPU BY PROGRAM
LIST      CPU BY PROGRAM
ON TABLE HOLD AS STATXXX
END
TABLE FILE STATXXX
HEADING CENTER
"SUMMARY OF RELATIVE CPU EFFICIENCY BETWEEN TWO FOCEXECS"
"CPU TIME IN MILLISECONDS"
SUM AVE.E02 AS 'AVG,CPU'
COMPUTE PCTBENX/I5=((LAST AVE.E02 - AVE.E02)/ LAST AVE.E02)*100; NOPRINT
COMPUTE PCTBENE/I5S=IF PCTBENX GE 0 THEN PCTBENX ELSE 0;AS 'PERCENT BENEFIT'
BY HIGHEST E02 NOPRINT BY E01 AS 'FOCEXEC'
PRINT E04
BY HIGHEST E02 NOPRINT BY E01 AS 'FOCEXEC'
BY E03 AS 'RUN,NUMBER'
ON E01 SKIP-LINE
ON TABLE SET PRINT ONLINE AND PAUSE OFF
END
```

will not take place until the ENTER key is pressed. Whether or not intermediate results are displayed, a counter is tested in order to determine if the requested number of run cycles has been performed. If not, the program branches back to run another test cycle; otherwise, the program runs its final report and ends.

The input screen for an actual test run of ONESTEP and TWOSTEP is shown in Screen 2 . The request is for two run cycles for which intermediate results will be displayed. Results from the two test cycles are shown by the intermediate results screens, Screen 3 and Screen 4 on page 20, respectively. Finally, Screen 5 depicts the output report that shows both the individual run statistics as well as the average for all the runs.

Some Other Considerations

- If one or both of the test FOCEXECs consumes considerable CPU resources and/or many test run cycles are being requested, then it is possible that the on-line session may time out due to maximum session CPU limits for the testing environment. In this case, it is suggested that a batch version of the CPU Efficiency Comparator program be created by eliminating the input screen and intermediate results screen. An appropriate JCL may be constructed, and the comparator can be invoked from the input stream via the following sample command:

```
EX CPUCOMPB
TEST1=TWOSTEP,TEST2=ONESTEP,RUNS=05
```

Screen 2

```
                    FOCEXEC CPU EFFICIENCY COMPARATOR

PLEASE ENTER:

      THE NAME OF THE 1ST TEST FOCEXEC:      TWOSTEP

      THE NAME OF THE 2ND TEST FOCEXEC:      ONESTEP

      THE NUMBER OF TEST RUNS TO PERFORM:    02

      DO YOU WISH INTERMEDIATE RESULTS:      Y

THE FOCEXECS THAT ARE ENTERED SHOULD BE ALTERNATE CODING STRUCTURES
 THAT PROVIDE THE SAME END RESULTS FOR THIS TEST TO BE MEANINGFUL.
          THE NUMBER OF TEST RUNS MAY BE BETWEEN 1 AND 10
```

- While the examples presented herein use the FOCUS/DB2 Read Interface, the CPU Efficiency Comparator program itself makes no such assumption. It is valid for any mainframe FOCUS environment (Version 5.5 or higher), whether it uses native FOCUS, DB2, IMS, or other external interfaces. Those differences should be represented in the CLIST or JCL used to invoke FOCUS.

- As stated previously, it is essential that both coding alternatives produce the same result from identical input data. It is equally important that all differences between FOCEXECs be incorporated in the individual FOCEXECs themselves. This includes not only differences in coding technique, but also differences in environmental setup commands, such as SET commands, TSO SQL SET commands, etc.

- This technique measures CPU efficiency only, and while we can assume that there is some correlation between CPU and I/O efficiency, this method does not try to address that. For those who wish to investigate I/O relationships, there are longstanding statistical variables in FOCUS that report the number of reads, database I/O, and sort I/O.

A Final Word

The use of this method has proved invaluable in providing insights on how to write efficient FOCUS code in complex situations. In particular, with the FOCUS/DB2 interface, a pattern began to become evident as to how to structure queries in general. Vast amounts of CPU resources could be saved with no loss of actual work produced. When running at its most efficient, FOCUS can gain important user acceptance among critical production department teams. In addition, training in how to use the system to its best advantage is graphic as opposed to theoretical. This represents powerful leverage for such a simple technique.

Figure 5

```
MASTER FILE DESCRIPTION FOR TIMESTAT FILE
USED IN CPUCOMP PROGRAM

FILE=TIMESTAT, SUFFIX=FOC,$
  SEGNAME=STAT, SEGTYPE=S0,$
    FIELD=PROGRAM, PGM, A8, A8,$
    FIELD=CPU_TIME, CPU, I9, I9,$
```

Screen 3

```
                    FOCEXEC CPU EFFICIENCY COMPARATOR

TEST RUN #    01

    TWOSTEP  , START TIME, CPU MILLISECONDS =  8699
    TWOSTEP  , END TIME  , CPU MILLISECONDS =  10237
    TWOSTEP  , ELAPSED   , CPU MILLISECONDS =  1538

    ONESTEP  , START TIME, CPU MILLISECONDS =  10238
    ONESTEP  , END TIME  , CPU MILLISECONDS =  122629
    ONESTEP  , ELAPSED   , CPU MILLISECONDS =  112391

BENEFIT OF  TWOSTEP  TO  ONESTEP  ,IN PERCENT CPU TIME =  98

                    PRESS ENTER TO CONTINUE
```

Screen 4

```
                    FOCEXEC CPU EFFICIENCY COMPARATOR

TEST RUN #    02

    TWOSTEP  , START TIME, CPU MILLISECONDS =  122936
    TWOSTEP  , END TIME  , CPU MILLISECONDS =  124572
    TWOSTEP  , ELAPSED   , CPU MILLISECONDS =  1636

    ONESTEP  , START TIME, CPU MILLISECONDS =  124576
    ONESTEP  , END TIME  , CPU MILLISECONDS =  234146
    ONESTEP  , ELAPSED   , CPU MILLISECONDS =  109570

BENEFIT OF  TWOSTEP  TO  ONESTEP  ,IN PERCENT CPU TIME =  98

                    PRESS ENTER TO CONTINUE
```

Screen 5

```
PAGE      1

SUMMARY OF RELATIVE CPU EFFICIENCY BETWEEN TWO FOCEXECS
            CPU TIME IN MILLISECONDS

                  AVG      PERCENT    RUN
FOCEXEC           CPU      BENEFIT    NUMBER   CPU TIME

ONESTEP         110908                 1        112391
                                       2        109570

TWOSTEP           1587       98        1          1538
                                       2          1636

                                   END OF REPORT
```

Controlling FOCUS Read Loops And Restarts

Patricia Davis
Independent Consultant

Many systems use a file to store control records needed for processing and use the –READ command to read the values at runtime. A common problem with this technique is that if the control file is closed for any reason (such as a –RUN), processing starts again at the first record of the file. In her article, Patricia Davis discusses several solutions to this problem.

A common system requirement is to use a file of control records as parameters in a set of repeated processing steps. However, when using the Dialogue Manager -READ command to establish the next control record to be processed, the FOCSTACK is run and the input control file is closed if the program invokes a -RUN. Any subsequent -READ processing will start again from the first record in the file.

Various techniques have been developed to handle this problem. In addition to two methods best used for small control files, this article describes several solutions involving operating system functions and text editors that, as well as being more suitable for larger control files, have the added feature of restart processing facilities.

FOCSTACK Solution

One solution suitable for a small number of control records is to keep filling the stack (i.e., no -RUN) until EOF (End of File) is reached on the control file. The end of the control file is determined by testing the value of &IORETURN, which returns a value not equal to 0 if the end of the file has been reached. This solution (an example is shown in Figure 1 on page 22) may involve a very large FOCSTACK; if operating system commands are being used, care must taken to ensure that commands are stacked correctly (e.g. -CMS FILEDEF will be run immediately, CMS FILEDEF will be stacked for later execution). The use of some system variables requiring I/O, such as &RETCODE, cause the stack to be run anyway. In this technique, any -QUIT command would have the effect of abandoning the whole job stream.

Iterative -READ Solution

Another solution, also suitable for a small number of control records, is to restart each process at the beginning of the control file and loop on the -READ until the *n*th + 1 record is reached. Two counters are used: the first, &NEXT, is reset to 0 after each process has been run; the second, &LOOP, is incremented each time a process is run. As successive records are read, &NEXT is incremented until its value equals that of &LOOP. In this manner, the required record is reached and the process executed. Depending on the size of the file, this

Figure 1

```
-* Define the control file
-CMS FILEDEF INPUT DISK [illegible] (or appropriate system command)
-* Read the next control record
-READNEXT
-READ INPUT variables [illegible]
-IF RC.IORETURN NE 0 GOTO READEOF
-* Start [illegible]
-* Determine if the control record is required for processing
-IF record not required GOTO READNEXT
-* [illegible]
-EXEC procedure parameters
-GOTO READNEXT
-* [illegible]
-* End of control file [illegible] FOCSTACK contains several processes
-READEOF
-RUN (Runs the whole stack)
```

solution may cause a significant increase in processing time. Care must also be taken to handle the next record correctly when control records are to be skipped. A FOCEXEC demonstrating this solution is shown in Figure 2.

Delete Record Solution

A better solution might be to remove the records that have been processed from the control file. The FOCEXEC shown in Figure 3 (on page 24) involves the use of a CMS XEDIT profile to remove the processed record. In this example, XEDIT does not delete the entire file; when the last record has been deleted, it is replaced with a single end record that can be recognized as EOF (End of file). If the file were completely deleted, the final -READ would fail. If it is required that the file be completely deleted, then 'INPUT EOF' should be replaced with 'ERASE filename filetype filemode' and 'QUIT'. Additionally, the test for EOF should be replaced with a -CMS STATE command prior to the -READ command to check if the control file still exists, and the test for 'EOF' should be removed.

A similar solution (shown in Figure 4 on page 24) is available within PC/FOCUS using TED. (Note: any existing TED profile may need to be saved.) On the PC, there does not appear to be a problem when TED deletes the final control record, and FOCUS will correctly processes a -READ on the empty file. Unfortunately, although the TED method works under CMS Release 5.5, FOCUS issues error messages when processing the TED commands, and this is therefore not considered a viable solution. This is unfortunate, since the TED method would have provided some operating system independence.

CMS XEDIT Restack Entry Solution

If the input control file is required to be saved, this can be done by adding an EOF record and restacking the processed record at the end of the file. Processing stops when the EOF record is reached. (See Figure 5 on page 25.)

If the control file is to be reused, any EOF marker(s) must be removed. Care should be taken to ensure that the control file is not too large to be handled by XEDIT or TED. However, if this is a consideration, then a problem with a large FOCSTACK or excessive -READ loops would also present a problem. The example shown in Figure 6 (on page 26) explores the possibility of using a CMS EXECIO procedure to restart processing if any break should occur.

Conclusion

The advantage of the text editor and direct read solutions is that the next control record to be

Figure 2

```
-* Defaults control file
CMSFILEDEF INPUT DISK ...... (Or appropriate system command)
-SET &LOOP = 0 ;

-* Control file restarted. Record counter is reset
-READLOOP
-SET &NEXT = 0 ;

-* Next control record (nth+1)
-READSKIP
-SET &LOOP = &LOOP + 1 ;

-* Read the next control record
-READNEXT
-READ INPUT variables......
-IF &IORETURN NE 0 GOTO READEOF ;

-* Check if required record reached
-SET &NEXT = &NEXT + 1 ;
-IF &NEXT LT &LOOP GOTO READNEXT ;

-* Determine if control record is required for processing
-IF record not required GOTO READSKIP ;

-* Processing
.
.
.
-END
-RUN (Runs the single process)
-GOTO READLOOP ;

-* End of control file processing
-READEOF
```

processed is always the first record to be read, and there is no added complication of determining if the FOCSTACK has been -RUN and the control file reset. An added benefit is that processing can be restarted from the current control record following an error or if a system abend has occurred and that processing will stop when an 'EOF' record is read or the end of the file is detected.

With the added feature of restart processing, a more powerful facility is available for breaking suitable large processing steps into smaller complete steps using a high-level key for controlling processing via one of the restart –READ techniques. This approach may help to alleviate, for example, a large job going down with disk full errors, or it may save completed processing steps from unnecessary re-running during a period of machine outage or unreliability. Experience in this area seems to indicate that, in certain suitable situations, there is an added efficiency when a single large job step is split into several smaller processes.

Provided that true errors can be trapped, this type of technique opens up design possibilities for automatic detection and restart of any unfinished processing steps as soon as the system is re-entered.

Figure 3

```
-* Create an XEDIT Profile to delete the last record processed
-*(i.e. first record)
-CMS FI FIXIT DISK FIXIT XEDIT A (RECFM FB LRECL 80 BLKSIZE 80
-WRITE FIXIT :1
-WRITE FIXIT DELETE 1
-* &| is used in the following code to inhibit valuation of the &
-WRITE FIXIT &|IF &|RETCODE EQ 0 &|GOTO -DONE
-WRITE FIXIT INPUT EOF processing
-WRITE FIXIT -DONE
-WRITE FIXIT FILE

-* Define the control file
-CMS FILEDEF INPUT DISK filename filetype filemode

-* Read the next control record
-READNEXT
-READ INPUT &TEST.A3. variables.......
-IF &TEST EQ 'EOF' GOTO READEOF ;

-* Determine if control record is required for processing
-IF record not required GOTO READSKIP ;

-* Processing
 .
 .
 .
END
-*Run the single process
-RUN

-* Label for control record to be ignored
-READSKIP

-* After processing delete the processed control record
-CMS XEDIT filename filetype filemode (PROFILE FIXIT
-GOTO READNEXT ;

-* End of control file processing
-READEOF
```

Figure 4

```
-* Save any existing TED profile
-DOS RENAME A:PROFILE.TED A:PROFILE.SAV

-* Create a TED Profile to delete the last record processed
-*(i.e. first record)
-DOS FILEDEF FIXIT DISK A:PROFILE.TED
-WRITE FIXIT :1
-WRITE FIXIT DELETE 1
-WRITE FIXIT FILE

-* Define the processing Control file
-DOS FILEDEF INPUT DISK drive:filename.extension

-* Read the next control record
-READNEXT
-READ INPUT variables......
-IF &IORETURN NE 0 GOTO READEOF ;
```

(continued on next page)

Figure 4 (continued)

```
-- Determine if record is required for processing
-IF record not required GOTO READSKIP ;

-- Processing

END
-Run the single process
-RUN

-- Label for control record to be removed
-READSKIP
-LABEL

-- After processing delete the processed control record
-XEDIT filemode:filename.filetype
-RUN
-GOTO READNEXT ;
```

Figure 5

```
-- Example

-- Create an XEDIT Profile to add the IPOF record
-CMS FI FIXIT DISK FIXIT XEDIT A (RECFM FB LRECL 80 BLKSIZE 80)
-WRITE FIXIT BOT
-WRITE FIXIT INPUT EOF
-WRITE FIXIT FILE
-CMS XEDIT filename filetype filemode (PROFILE FIXIT

-- Create an XEDIT Profile to restack the processed record
-CMS FI FIXIT DISK FIXIT XEDIT A (RECFM FB LRECL 80 BLKSIZE 80)
-WRITE FIXIT BOT
-WRITE FIXIT GET filename filetype filemode A
-WRITE FIXIT :1
```

Figure 6

```
-PROC1B PCOPEN BOOK = used to restart after abend
-DEFINE = DESTINATION procedure
-CMS FT1 RESTART DISK RESTART PROCREC A= (FREQ=1 SB=(TRUE,80) BLKSIZE 80)

-* The next statement will restore any unfinished processing
-* from the RESTART PROCREC. Should contain:
-INCLUDE RESTART

-* Processing

-* REPEAT PROCREC = controls processing
-DEFAULT BIG/START/COUNT number
-DEFAULT NEXT = 0 ?

-* Set the restart condition
-READNEXT
-IF/TH RESTART EXEC REPEAT NEXT=NEXT

-* Read the next control record (if any)
-SET CNT = CNT + 1
-CMS PIPE(OUT ) LOCKSTK INPUT DATA A (NEXT
-IF STORE=TRUE NEXT GOTO READFOR ?
-READ BYTEN variable

-* Determine if control record is required for processing
-IF record not required GOTO READNEXT ?

-* Processing

END
-Run the single process
-RUN

-GOTO READNEXT

-* Exit if control file processing
-READFOR

-* Process is complete, restart is not required
-WRITE RESTART LEAVE
```

Using Session Monitor To Store Variables For Batch Processing

Steven St. John
HCA Information Services, Inc.

The ability to retain FOCUS variables for future use is often desirable when coding an application. However, these variables are erased when the user exits FOCUS. Steve St. John uses the Session Monitor to store these variables in a batch file.

Variables are powerful tools for FOCUS programmers. They are frequently used to allow end-users to enter screening criteria to be used in FOCUS programs. One limitation of FOCUS variables is that they are only active during a FOCUS session. When a user exits FOCUS, the variables disappear.

This article presents a technique for retaining interactively-generated variables for future use by passing the values from an interactive session to a batch file. The technique uses the Session Monitor currently available only in the VM/CMS and MVS/TSO environments.

Retaining Interactively-Generated Variables

The CARMENU FOCEXEC shown in Figure 1 (on page 28) begins by initializing a counter and the &&variables to be used in the subsequent batch job. The following numbered paragraphs correspond to the numbers in Figure 1.

1. The end user is prompted via –CRTFORM to enter values for the variables. In this example, the variables ending with 'COND' (i.e. &&COUCOND, &&CARCOND, &&MODCOND) are populated with valid FOCUS operators, such as EQ, GE, or LT. For those variables ending with 'VAR', the user enters a value such as 'U.S.A.' for &&COUVAR or 'FORD OR PEUGOT' for &&CARVAR.

2. After the user has entered the information, the value of the counter directs control to the proper Dialogue Manager label, initially CARAMPS1.

3. The file CARAMPS1 is then allocated and the Session Monitor, which stores terminal input and output in a stack in core, is switched on. The FOCUS command '? &&' is issued, displaying all &&variables and their values on the screen and thus placing them in the Session Monitor stack.

4. The Session Monitor command /TOP moves the pointer to the top of the stack, and /SAVE CARAMPS1 stores the stack in the sequential file CARAMPS1.

5. The Session Monitor is switched off and the batch job, CARJOB1, is submitted. CARJOB1 will use the variables from the CARAMPS1 file. Dialogue Manager routes control back to the top of the FOCEXEC,

Figure 1

```
     -*  INITIALIZE VARIABLES
     -SET &&COUNT = 0;
     -SET &&COUCOND='  ';
     -SET &&COUVAR='                                        ';
     -SET &&CARCOND='  ';
     -SET &&CARVAR='                                        ';
     -SET &&MODCOND='  ';
     -SET &&MODVAR='                                        ';
     -TOP
     -*  INCREMENT COUNTER
     -SET &&COUNT = &&COUNT + 1;
     -*  PROMPT USER FOR VARIABLES
     -CRTCLEAR
1.   -CRTFORM LINE 1
     -"                          CAR REPORT
     -"  SCREENING                                        <78 "
     -"  CRITERIA     COND    VARIABLES                   <78 "
     -"  --------     ----    ------------                <78 "
     -"  COUNTRY      <.I.&&COUCOND   <.I.&&COUVAR        <78 "
     -"  CAR          <.I.&&CARCOND   <.I.&&CARVAR        <78 "
     -"  MODEL        <.I.&&MODCOND   <.I.&&MODVAR        <78 "
     -"                                                   <78 "
     -"  PRESS [ENTER] TO CONTINUE OR [ALT + 3] TO QUIT

2.   -IF &&COUNT.EQ.1 THEN GOTO CARAMPS1 ELSE
     -IF &&COUNT EQ 2 THEN GOTO CARAMPS2 ELSE GOTO EXIT;

3.   -CARAMPS1
     TSO FREE F(CARAMPS1)
     TSO ALLOC F(CARAMPS1) DA('FOCUS.HOLD.CARAMPS1') SHR
     SET SM=ON
     ? &&
4.   /TOP
     /SAVE CARAMPS1
5.   SET SM=OFF
     TSO SUB (JCLLIB.JCL(CARJOB1))
     -RUN
     -GOTO TOP

     -CARAMPS2
     TSO FREE F(CARAMPS2)
     TSO ALLOC F(CARAMPS2) DA('FOCUS.HOLD.CARAMPS2') SHR
     SET SM=ON
     ? && .
     /TOP
     /SAVE CARAMPS2
     SET SM=OFF
     TSO SUB (JCLLIB.JCL(CARJOB2))
     -RUN
     -GOTO TOP

     -EXIT
```

the counter is incremented, and the –CRTFORM is redisplayed for user input.

The use of a counter to direct Dialogue Manager flow allows the use of multiple files for storing variables and multiple batch jobs. This prevents file contention between the online and batch jobs and enables the batch jobs to execute concurrently.

Transferring The Variables To Batch Jobs

Assume that the user entered 'EQ' and 'U.S.A.' for the country screening criteria and 'EQ' and 'FORD' for car. The resultant CARAMPS1 file is displayed in Figure 2. Note the algorithm used to list the variables in response to the '? &&' query. The variable names are listed in the

Figure 2

```
&&COUNT        1
&&CARVAR       'FORD
&&COUVAR       'U.S.A.
&&MODVAR       '
&&CARCOND      'EQ'
&&COUCOND      'EQ'
&&MODCOND      '  '
```

order of fewest characters. Thus, &&COUNT, which contains 7 characters, is listed first. The variables names with 8 characters are listed next, followed by those with 9 characters. When more than one variable name has the same number of characters, they are secondarily sorted alphabetically. The batch job (CARJOB1 shown in Figure 3) using these variables relies on knowledge of the order.

CARJOB1 begins by initializing the variables, then performs a –READ on CARAMPS1. Since variables with names of 8 characters will be read first, the field &VALUE is defined as A40, corresponding to the variable's format in the online CARMENU job.

Other variables, which establish the IF conditions in the upcoming TABLE request, are determined based on the values of the variables. After the last variable, &&MODVAR, is read, Dialogue Manager passes control to the second –READ, which reads the FOCUS operators and sets the format of their values as A2.

When the last record of CARAMPS1 has been read, &IORETURN is set to a value other than 0, and Dialogue Manager directs control to the label –REPORT.

The TABLE request is executed using the variables as screening criteria. A section of the output (Figure 4 on page 30) shows that the variables have been determined and inserted into the FOCEXEC.

Figure 3

```
-*  CARJOB1

-* INITIALIZE VARIABLES
-SET &&COUVAR = '
-SET &&CARVAR = '
-SET &&MODVAR = '
-SET &&COUCOND = '  ';
-SET &&CARCOND = '  ';
-SET &&MODCOND = '  ';

-*  READ 'CARAMPS1' TO OBTAIN OPERANDS
-READVAR
-READ CARAMPS1 &SPACE1.A2. &AMPER.A9. &SPACE.A6. &VALUE.A40.
-IF &IORETURN NE 0 GOTO REPORT;
-IF &AMPER EQ '&&COUVAR' GOTO COUVAR ELSE
-IF &AMPER EQ '&&CARVAR' GOTO CARVAR ELSE
-IF &AMPER EQ '&&MODVAR' GOTO MODVAR ELSE GOTO READVAR;

-COUVAR
-SET &&COUVAR = &VALUE;
-SET &&COUIF = IF &&COUVAR EQ ' ' THEN ' ' ELSE 'IF COUNTRY ';
-GOTO READVAR;

-CARVAR
-SET &&CARVAR = &VALUE;
-SET &&CARIF = IF &&CARVAR EQ ' ' THEN ' ' ELSE 'IF CAR ';
-GOTO READVAR;

-MODVAR
```

(continued)

Figure 3 (continued)

```
-SET &MODVAR = &VALUE;
-SET &MODIF = IF &MODVAR EQ 1 THEN 1 ELSE 1 IN MODIF;
-GOTO READCOND;
-READ
-READ CARAMPST AT TO OBTAIN OPPRATOR
-READCOND
-READ CARAMPST &SPACE1(12) CAMPER(A9) &SPACE2(A6) &VALUE(A2)
-IF &RETURN NE 0 GOTO REPORT
-IF CAMPER EQ '&&COUCOND' THEN GOTO COUCOND ELSE
-IF CAMPER EQ '&&CARCOND' THEN GOTO CARCOND ELSE
-IF CAMPER EQ '&&MODCOND' THEN GOTO MODCOND ELSE GOTO READCOND;
-COUCOND
-SET &COUCOND = &VALUE;
-GOTO READCOND;
-CARCOND
-SET &CARCOND = &VALUE;
-GOTO READCOND;
-MODCOND
-SET &MODCOND = &VALUE;
-GOTO READCOND;
-REPORT
-TABLE FILE CAR
PRINT COUNTRY CAR MODEL
WHERE COUNTRY EQ &&COUCOND AND CAR EQ &&CARCOND AND MODEL EQ &&MODCOND
END
-EXIT
```

Figure 4

```
TABLE FILE CA
PRINT COUNTRY CAR MODEL
IF COUNTRY EQ U.S.A.
IF CAR EQ FORD
END
NUMBER OF RECORDS IN TABLE = 3 LINES = 3
-PAUSE - PRESS ISSUE CARRIAGE RETURN WHEN READY
PAGE 1

COUNTRY      CAR         MODEL
--------     ---         -----
U.S.A.       FORD        MUSTANG
U.S.A.       FORD        PINTO
U.S.A.       FORD        THUNDERBIRD
```

An Automated Method For Verifying FOCUS Database Integrity

Eric Heilner
Federal Reserve Bank of New York

It is often a good idea to adopt precautionary measures to guard against occurrences of system problems. Eric Heilner demonstrates an automated technique to check for integrity problems in a FOCUS database.

In any data management system that uses list structures to store information, it is essential that pointers are properly maintained. If a pointer becomes damaged, every record following the damaged pointer may be lost, since there is no *a priori* method to determine the next record and restore the chain. While this is a very rare occurrence in FOCUS, there are several scenarios where pointers can become damaged.

- System failure. If a computer crashes in the middle of a multi-segment transaction, it is possible that only part of a record will be correctly entered. This only occurs on certain operating systems (e.g., MVS without the Shadow Paging option).

- Discrepancies between the master file description (MFD) and the physical file. While FOCUS does an excellent job of preventing such discrepancies, there are at least two exceptions:

  (i) The SCAN FILE facility does not perform run-time checks to verify that the segment size in the MFD is equal to the actual segment size in the physical database. Thus, if a new field is added to a segment or the size of an existing field is increased and a REORG is not done, FOCUS will overwrite the next physical segment instance on a SCAN update and destroy the pointers.

  (ii) It is possible to create a file with recursive segments, add information to that file, remove the recursive segments from the MFD without doing a REORG, then do a MODIFY against the file. I have observed one situation involving a complex COMBINEd MODIFY where the recursive pointers were overwritten with data.

A simple way to check for integrity problems in a FOCUS database is to compare segment counts in physical order (obtained from a ? FILE) with segment counts in pointer order (using TABLEF FILE). If these are not identical, the database has been damaged. Unfortunately, this process can be very tedious if done manually, especially if a large number of databases need to be checked daily as a precaution. This technique uses a FOCEXEC that will do the comparison for any FOCUS database.

Analysis Of Procedure Code

The design of this procedure presented several interesting challenges in coding, namely:

(continued on page 33)

Figure 1

```
-* &1 is the name of the FOCUS database being checked.  It must reside in
-* the current MASTER
-*********************
SET ALL = ON
-*********************
-PART1
-* Use Session Monitor to put the results from ? FILE into FILEST1.  In the
-*PC or VAX environment, this could also be accomplished by FILEDEFing
-*SYSPRINT to a file and then doing a ? FILE.
-*
-* CMS FI FILEST1 DISK FILE FILEST1 A (LRECL 80 BLKSIZE 6160 RECFM FB)
-* TSO ALLOC F(FILEST1) SP(2) BLOCK(6160) LRECL(80) RECFM(FB)
SET SM=ON
? FILE &1
/TOP
/PO
/SAVE 100 FILEST1
SET SM=OFF
-RUN
-*********************
-PART2
CHECK FILE &1 HOLD
-****
-*  Determine the segment names being checked and put in file SEGNAMES.
-* Ignore unique and cross referenced segments.
TABLE FILE HOLD
PRINT SEGNAME
IF SEGTYPE EQ ' ' OR 'S$*'
ON TABLE SAVE AS SEGNAMES
END
-RUN
-SET &SEGCOUNT = &LINES ;
-*********************
-* Generate the TABLE FILE code that will count the SEGMENT instances.
-*PR_FIELD must be 80 bytes to ensure that the sure file CRTFIELD will
-*have the proper LRELL to be included in Part 3.
DEFINE FILE HOLD
APOS/A1 WITH FIELDNAME      = HEXBYT(125, APOS) ;
F_COUNT/I5  WITH FIELDNAME  = F_COUNT + 1 ;
PR_FIELD/A80 WITH FIELDNAME = IF (F_COUNT EQ 1) THEN
-IF (&SEGCOUNT EQ 1) GOTO ONE_SEG_ONLY ;
      'WRITE CNT.' | FIELDNAME || '/I08 AS ' | APOS | SEGNAME | APOS ELSE
      'OVER CNT.' | FIELDNAME || '/I08 AS ' | APOS | SEGNAME | APOS ;
-GOTO END_DEFS
-ONE_SEG_ONLY
   'WRITE CNT.' | FIELDNAME || '/I08 AS ' | APOS | SEGNAME | APOS
   'OVER CNT.' | FIELDNAME || '/I08 AS ' | APOS | APOS | ' NOPRINT' ;
-END_DEFS
END
TABLE FILE HOLD
WRITE FST.PR_FIELD
BY SEGNO  NOPRINT
BY SEGNAME NOPRINT
IF SEGNAME EQ (SEGNAMES)
IF SEGTYPE EQ ' ' OR 'S$*'
ON TABLE SAVE AS CNTFIELD
END
-RUN
-*****************************************
-PART3
-* Count the SEGMENT instances in pointer order and put the results into
-* file FILEST2
TABLEF FILE &1
```

(continued)

Figure 1 (continued)

```
-INCLUDE CNTFIELD.FTM
ON TABLE SET PAGE NOPAGE
ON TABLE SAVE AS FILEST2 FORMAT WP
END
-RUN
-*****************************************
-PART4
-* Compare the SEGMENT counts in physical order (FILEST1) vs the
-* SEGMENT counts in pointer order (FILEST2) and hold any which do not
-* match in file RESULTS
MATCH FILE FILEST1
PRINT SEGNAME QFIL ACTIVE_QFIL
BY SEGNAME_QFIL AS 'SEGNAME'
BY ACTIVE_QFIL AS 'ACTIVE_COUNT'
IF SEGNAME_QFIL EQ (SEGNAMES)
RUN
FILE FILEST2
PRINT SEGNAME_TAB ACTIVE_TAB
BY SEGNAME_TAB AS 'SEGNAME'
BY ACTIVE_TAB AS 'ACTIVE_COUNT'
IF SEGNAME_TAB NE ' '
AFTER MATCH HOLD AS RESULTS OLD-NOR-NEW
END
-RUN
-*********************************
-PART5
-IF (&LINES NE 0) GOTO BAD_STUFF ;
TABLE FILE RESULTS
HEADING
"*** FILE &1 IS OK ***"
WRITE FST.SEGNAME NOPRINT
END
-EXIT
-****
-BAD_STUFF
TABLE FILE RESULTS
HEADING
"THERE IS A PROBLEM WITH &1 DATABASE - &DATE"
-
PRINT SEGNAME_QFIL AS '-SEGNAME,FROM ? FILE'
      ACTIVE_QFIL AS '  COUNT, FROM ? FILE'
      SEGNAME_TAB AS '-SEGNAME,FROM TABLEF'
      ACTIVE_TAB  AS '  COUNT,FROM TABLEF'
END
```

- How can a TABLEF be generated against a database if the fieldnames or the structure of the file are not known?

- How can the results of the ? FILE and the TABLEF be formatted so they can be compared (presumably by a MATCHing report)?

The solution to the first question is to use CHECK FILE HOLD to generate the actual code used by the TABLEF. The answer to the second question is to put the results of both the ? FILE and the TABLEF into external SAVE files and create MASTER descriptions for each. The code is shown in Figure 1 and is broken down into five major parts:

1. Put the results from the ? FILE into FILEST1.

2. Generate FOCUS code that will be used by the TABLEF FILE.

3. Run the TABLEF FILE using the code generated by part 2 and SAVE the results in FILEST2.

4. MATCH FILEST1 and FILEST2 and HOLD any differences.

5. Print the results.

The master file descriptions (MFDs) for FILEST1 and FILEST2 are shown in Figure 2.

The best way to understand the methodology is to proceed backwards from the MATCH in Part 4. Since the output from a ? FILE is already in an accessible format (one record per line, with each record containing the segment name and count, plus some extraneous headings that can be easily filtered out), an obvious strategy is to force the output of the TABLEF into a similar format. FOCUS provides a means of doing this with the OVER option. Ideally, the TABLEF code would look something like:

```
TABLEF FILE filename
WRITE CNT.a-field-from-segment1 AS
'segname1' OVER CNT.a-field-from-
segment2 AS 'segname2'
OVER
etc.
ON TABLE SAVE AS FILEST2 FORMAT WP
END
```

In this way, the results of the TABLEF will be one record per line, with each record containing the SEGNAME and a count of a field in that segment. Parts 2 and 3 generate precisely this code. (Note: This is similar to the VALIDATE FOCEXEC shipped with FOCUS.)

In Part 2, CHECK FILE HOLD determines which fields to count. One field from each segment is selected, and a line of TABLEF coding is placed in the 80-byte DEFINED field PR_FIELD. WRITE CNT,[illegible] is used instead of a simple COUNT since the results of the COUNT verb are always in A8 format, which would limit the use of the procedure to segments with fewer than 100,000 instances. The purpose of the apostrophes in the DEFINE is to insure that spacing in the output of the TABLEF will be identical for fields/segments. If there is only one segment in the database, an additional dummy segment is added; otherwise, the segment name will appear over the count instead of on the same line. Unique segments are not counted; they are always present.

The FOCUS code generated by [illegible] SAVEd in file CNTFIELD for use [illegible] COUNTing with the OVER option [illegible] count on a separate line, but the [illegible] be SAVEd as FORMAT WP, since [illegible] SAVE reformats the OVERs onto one [illegible] creates one final problem to be [illegible] FILEST1 can be compared with [illegible] least on TSO and CMS). The SAV[illegible] duced by the FORMAT WP option [illegible] record length file with a blank [illegible]

Figure 2

```
$FILEST1 is used to report against the results of the ? FILE
FILE=FILEST1, SUFFIX=FIX,$
SEGNAME=FL1INFO,$
  FIELD=FILLER, ,            A02,        A02,$
  FIELD=SEGNAME_QFIL, ,      A08,        A08,$
  FIELD=FILLER, ,            A04,        A04,$
  FIELD=ACTIVE_CNT, ,        A08,        A08,$
  FIELD=FILLER, ,            A58,        A58,$
DEFINE
  ACTIVE_QFIL/A8=IF (ACTIVE_CNT EQ '  ') THEN '      0' ELSE ACTIVE_CNT;
```

```
$FILEST2 is used to report against the HOLD file produced by the TABLEF
$It may need modifications for PCs or non-IBM operating systems,
$depending on the layout of the file produced when doing a
$SAVE FORMAT WP.
FILE=FILEST2, SUFFIX=FIX,$
SEGNAME=TOPSEG,$
  FIELD=FILLER, ,            A02,        A02,$
SEGNAME=FL2INFO, PARENT=TOPSEG, OCCURS=VARIABLE,$
  FIELD=SEGNAME_TAB, ,       A08,        A08,$
  FIELD=FILLER, ,            A02,        A02,$
  FIELD=ACTIVE_TAB, ,        A08,        A08,$
```

appearing first; a one segment MASTER cannot handle this situation. The solution is to use a two-segment MFD for FILEST2. The root segment picks up the initial record, while the child (with OCCURS=VARIABLE) picks up the remaining information.

Implementation

When this procedure is run, two ? FILE listings (one from the initial ? FILE, the second from the Session Monitor) will be displayed before the actual results from Part 5. This creates a very cluttered look and makes the procedure impractical to use with many databases at once.

For this reason, it is recommended that the procedure be run in batch mode, with SYSPRINT allocated to a temporary dataset that is deleted at the end of the job and OFFLINE allocated to wherever is most convenient (typically the screen output listing). By doing this, the output from the ? FILE listings is split off from the results (a sample JCL is shown in Figure 3) . Of course OFFLINE mode must be set for this to work.

Two additional caveats should be noted. Spurious inconsistencies can be generated if the database is updated after the ? FILE is generated, but before the TABLEF is completed. A database should always be checked twice before assuming a problem exists.

Also, if there are no records in a file, the TABLEF will not work, and an incorrect result will be reported. This can be easily detected and should pose no problem for most programmers.

Fixing Bad Pointers

FOCUS provides at least two methods of correcting damaged pointers. The REBUILD REBUILD option cleans up all pointer-related discrepancies (at least all problems that this author has encountered). However, as previously mentioned, segment instances following the bad pointer may be lost. If the bad pointer is on the root segment, a better approach is to do a TABLE FILE, dump the entire file, then FIXFORM the information back into the file. This method is superior because TABLE FILE retrieves the top segment in physical order, thus circumventing any bad pointers.

Thanks to Doug Lee for providing useful suggestions and verifying that the code worked under CMS.

Figure 3

```
//job card
//
//
//FOCUS EXEC PGM=FOCUS
//
//  allocate SYSPRINT to dummy dataset
//SYSPRINT DD DUMMY,DSN=&&SYS,SPACE=(CYL,(1,1),RLSE),
//
//
//
//  allocate OFFLINE to screen display
//OFFLINE  DD  SYSOUT=*
//
//
//  allocate remaining FOCUS files and database
//
//
//SYSTSIN  DD
OFFLINE
SET PAGE=OFF
EX database_check_routine
FIN
//
```

Using SORTHOLD With CRTFORM

Joseph Szpiro
Szpiro Computers

Frank Roberts
API

The FOCUS 5.5 feature SORTHOLD reorders records in the Scratch Pad Area. Joe Szpiro and Frank Roberts have developed this technique using SORTHOLD to dynamically resort records displayed on a CRTFORM.

The SORTHOLD subcommand in MODIFY provides a mechanism to reorder the records in the scratch pad area (SPA). CRTFORMs are often used to update or display multiple records on one screen. The order in which these records are shown on the CRTFORM can be controlled through the use of PF keys on mainframes.

For example, in Screen 1, the user is shown a list of employees, their social security numbers, names, and current salaries from a small payroll system. The list, by default, is displayed in social security number order. However, some users may prefer to view the list in name or salary order; some may wish to select ascending or descending order.

As shown in Screen 1, the user can select an order by choosing a PF key from the list on the bottom of the screen. When the PF key is pressed, the list is resorted in the desired order.

The MODIFY request in Figure 1 shows [illegible] accomplish this resorting. If a PF key [illegible] lected requesting an alternative sort, [illegible] program branches to a CASE in which [illegible] a SORTHOLD subcommand. Thus, [illegible] occurs in six CASEs - SSNLOW, SSN[illegible] NAMELOW, NAMEHIGH, SALELOW, [illegible] SALEHIGH. In each the syntax is:

```
SORTHOLD BY (HIGHEST) FIELD1
         BY (HIGHEST) FIELD2
```

After the resorting, the program branches [illegible] to CASE DISPLAY to display the list [illegible] order requested.

Screen 1

```
                    PAYROLL DISPLAY SCREEN

SOC  SEC NO  NAME                                  SALARY
-----------  ----------------------------------    ------
123-45-6789  Salvador, Michael                     2000.00
567-45-1212  Albertson, John                       1500.00
678-11-1111  Barton, Kim                           [illegible]
789-11-1344  Willy, John                           1200.00
876-12-1234  Masters, Elizabeth                    4300.00
-----------  ----------------------------------    ------

SORT BY LOWEST -> PF04=SOCSECNO   PF05-NAME   PF06-SALARY
     BY HIGHEST-> PF14=SOCSECNO   PF15-NAME   PF16-SALARY
     OR         -> PF03=EXIT      PF07-BACK   PF08-[illegible]
```

Figure 1

```
MODIFY FILE PAYROLL
COMPUTE PFKEY/A4=;SCREENINDEX/I5=;
NEXT SOC_SEC_NO
  ON NEXT  HOLD SOC_SEC_NO NAME SALARY
  ON NEXT  GOTO TOP
  ON NONEXT GOTO DISPLAY

CASE DISPLAY
CRTFORM LINE 1
"                PAYROLL DISPLAY SCREEN"
" "
"SOC SEC NO.  NAME                                SALARY     "
"_____ _________________ _____"
"<D.SSN(1)    <D.NAME(1)>                         <D.SALARY(1)>"
"<D.SSN(2)    <D.NAME(2)>                         <D.SALARY(2)>"
"<D.SSN(3)    <D.NAME(3)>                         <D.SALARY(3)>"
"<D.SSN(4)    <D.NAME(4)>                         <D.SALARY(4)>"
"<D.SSN(5)    <D.NAME(5)>                         <D.SALARY(5)>"
"_____ _________________ _____"
" "
" SORT BY LOWEST -> PF04=SOCSECNO  PF05=NAME   PF06=SALARY "
"      BY HIGHEST-> PF14=SOCSECNO  PF15=NAME   PF16=SALARY "
"      OR         -> PF03=EXIT      PF07=BACK   PF08=FORW   "

IF PFKEY EQ 'PF03' GOTO EXIT     ELSE
IF PFKEY EQ 'PF07' GOTO BACKONE  ELSE
IF PFKEY EQ 'PF08' GOTO FORWONE  ELSE
IF PFKEY EQ 'PF04' GOTO SSNLOW   ELSE
IF PFKEY EQ 'PF14' GOTO SSNHIGH  ELSE
IF PFKEY EQ 'PF05' GOTO NAMELOW  ELSE
IF PFKEY EQ 'PF15' GOTO NAMEHIGH ELSE
IF PFKEY EQ 'PF06' GOTO SALLOW   ELSE
IF PFKEY EQ 'PF16' GOTO SALHIGH  ELSE DISPLAY ;
ENDCASE

CASE BACKONE
COMPUTE TEMPINDEX= SCREENINDEX - 5 ;
        SCREENINDEX = IF TEMPINDEX LT 0 THEN 0 ELSE TEMPINDEX ;
GOTO DISPLAY
ENDCASE

CASE FORWONE
COMPUTE TEMPINDEX= SCREENINDEX + 5 ;
        SCREENINDEX = IF TEMPINDEX GT (HOLDCOUNT - 5)
        THEN HOLDCOUNT - 5 ELSE TEMPINDEX ;
GOTO DISPLAY
ENDCASE

CASE SSNLOW
SORTHOLD BY SSN BY NAME BY SALARY
GOTO DISPLAY
ENDCASE

CASE SSNHIGH
SORTHOLD BY HIGHEST SSN BY NAME BY SALARY
GOTO DISPLAY
ENDCASE
```

Interview: Dave Kaplan

SQL Server is one of a new crop of client-server database products currently engaging the imagination of developers. Since FOCUS will be one of the client products which will operate with SQL Server, we discussed the product with Dave Kaplan, Microsoft's product manager. The conversation took place prior to IBI's designation as a gold reseller.

Could you briefly describe SQL Server as a product, its goals in the marketplace, and the class of applications at which it is pointed?

SQL Server is a database engine with the quality, performance, and power previously found only on mainframes and minicomputers. It supports the full range of business applications, from mission-critical transaction processing systems all the way to flexible decision support systems.

For example, an SQL Server workgroup can support transaction-oriented users accessing accounting or inventory databases via FOCUS, dBASE, or C language applications, at the same time other users access those databases through Lotus 1-2-3, Paradox or Excel for reporting and business analysis.

From the PC network point of view, SQL Server is a dramatic upgrade from what people have done with the back-end file sharing model that's current among PC database products. If you look at it from the mainframe or transaction processing point of view, it's a way to accomplish, in a much more flexible environment, mission-critical applications that have been relegated to a host-terminal setup in the past.

We have worked closely with our part[illegible] Ashton-Tate and Sybase, to bring SQL [illegible] to the PC LAN environment. We feel that [illegible] Server combines the best of minicomputer [illegible] computing with the best of the PCs [illegible] SQL Server, along with its wide range [illegible] end applications (dBASE IV, FOCUS [illegible] dox, DataEase, Revelation, 1-2-3, [illegible] others) represents the first delivery [illegible] promise of client-server computing.

SQL Server is expensive in dollars [illegible] to your typical PC DBMS; it's also expe[illegible] terms of resources–30 megabytes of [illegible] 6 megs of memory. A recent PC Tech[illegible] story by Richard Finkelstein ran a table [illegible] ing how much more your product requi[illegible] some of the alternatives. Is that [illegible] cause we're looking at a more powerful [illegible] with a higher aim?

Other database server products will [illegible] approximately the same amount in terms [illegible] memory, disk, etc. to get comparable performance. If you run engines that were developed initially under DOS in any environment [illegible] requires connecting a large number of us[illegible] they will either have to go with an exten[illegible] memory scheme under DOS or move into [illegible] OS/2 environment. In that case, you're in [illegible]

the same ballpark as you are with SQL Server. When you look at the 6 meg total, that's the base configuration for OS/2 plus LAN Manager plus SQL Server, and anybody else running in the same environment will still require the operating system and the network to be there. SQL Server itself, depending on cache and buffer size, is going to account for perhaps 2 or 2 1/2 meg of that.

So in a very real sense, we compare favorably to other products, and in terms of retail pricing, that's also true; we're probably at about the middle of the range as far as competitive products are concerned.

A related resource issue is not only what is the base memory configuration, but how much memory do you require per user connected. SQL Server is quite efficient in that respect, taking in the neighborhood of about 40K per connection. In a typical environment, the first 6 meg for entry configuration will get you the first five or ten user connections, depending on transaction volumes. At 40K per connection, every meg added to the system beyond the first 6 enables you to connect another 25 users. Other products literally require as much as 512K per user, so for every meg you add to the system you can add another two users. That's a pretty dramatic difference.

Do you see the marketplace opportunity as being the sizable firms porting mission-critical systems?

Yes, but not to the exclusion of, say, decision support systems connecting a workgroup of spreadsheet users. Many early adopters of SQL Server are corporate MIS departments, sophisticated systems integrators, and network-oriented VARs. If you look at the latter two categories, the end customer is still typically the Fortune 500 or 1000 company. These are the people who are blazing the trails; they realize that with the type of software represented by SQL Server and the type of hardware it can run on, mission-critical applications can be done for a fifth or one-tenth or even one-twentieth the cost of the way they're implemented today.

A lot of SQL Server's strengths really recommend it in the mission-critical environment in a way that other competing product offerings are not nearly as likely to – sheer transaction throughput, the robustness of the database engine itself, the ability to enforce data integrity at the database level. These are things that go beyond ordinary SQL implementations, and, in our experience, they are things that seem to be important to corporate users.

There seems to be a trend associated with what Microsoft and IBM are doing to expand the mission of the operating system itself. If you look at IBM's OS/2 Extended Edition, suddenly the network/communications functions and the database functions are being perceived not as separate applications, but as extensions of the operating system itself. Is this purely due to the fact that these demands are typically heavy users of system resources; is it that this is where the money is going to come from? What is the rationale?

It's part of the ongoing evolution in computing that, as services are recognized as being generic with succeeding generation of computing technology, it makes more and more sense to incorporate those services at the system level. To turn back the clock, it was only 30 years ago that people were programming in binary, but compilers and fourth-generation languages like FOCUS came along and changed all that. The database and communications areas certainly make sense as services that, especially in a connected environment, belong in the system level of the software. IBM's direction reflects this fact. Their architecture makes very good sense for the IBM environment *per se*, and especially for the environment with a lot of PS/2 machines connected principally to IBM hosts.

SQL Server complements the IBM products, offering SQL capabilities that have good interoperability with the IBM offerings, and that also maximize what you can get out of the LAN itself. We will also offer connectivity to a heterogeneous world, including IBM and non-IBM systems alike. Applications such as PC databases, spreadsheets, accounting systems,

and the like, all need to access data from a variety of remote data sources and still have a local high performance data source available to them over the LAN. That's where we see LAN Manager and SQL Server providing a systems-level foundation.

There was a time when access methods (ISAM, VSAM) removed the need to program physical record access. Soon we couldn't live without them. The level of power in a single command available to the programmer has been raised substantially, and 4GLs, like FOCUS are a good example of that. Now we can just say, PRINT this.

In a similar fashion, we're extending it now by putting APIs into the environment and saying, "code the call like this and we'll do all the work for you." It seems to me that the danger in this is that we standardize on whatever is there, regardless of how efficient it may be in a given situation or configuration. You have system tables in SQL Server containing physical information that may be used in query optimization. Can you discuss that a little?

SQL Server's query optimizer is very efficient, making use of index and table size information that is dynamically maintained by SQL Server itself. But besides the query side of things, SQL Server is the first relational DBMS that has been designed from the ground up to handle heavy-update transaction systems at the same time.

With Sybase, we have ported this technology to the OS/2 environment and added a number of extensions that we think are quite important in the PC world, like browse mode with update, large text, and image data types that are now also offered by Sybase on its other platforms.

This makes for a scalable server strategy – an OS/2 SQL Server in the PC world that will also be sold by Microsoft OEMs, and SQL servers from Sybase on other platforms (Sun UNIX, parallel UNIX systems, VAX systems, etc.).

A lot of the relational engines on the market have core technologies that go back to the late 70's and early 80's when people thought that the most you could really do with relational was an *ad hoc* query system, or a decision support system, with a few users at best. SQL Server certainly has all those capabilities but is also aimed at a larger spectrum of applications, including the really high performance ones as well.

> *"Most of the implementations that people are talking about are single vendor, or homogeneous, solutions... the major issue is distributed update."*

Today, "scalable" means that if a database we've implemented, let's say, on a 386 server is reaching its limit, we can jump to another level of power by moving to a mini. In the not too distant future, I would guess we'd hope to add another 386 server that and the system would be intelligent enough to break up the work so that it does work additively. How far are we from being able to scale by addition, then, rather than by expansion? How much of that depends on Microsoft's new installable file system for OS/2?

To scale by replication really hinges on the capability of transparent distributed data, which in some sense is the holy grail that the market is aiming at. There tends to be a lot of confusion in the market right now about the whole area. Many vendors, in fact, claim to have it today. On closer examination, two things emerge. One is that nobody is worried about the problem of multi-vendor distributed databases yet; most of the implementations that people are talking about are single vendor or homogeneous, solutions. Secondly, it appears that most people are approaching the problem through the query or read side of the problem. In other words, how you do distributed reads against multiple databases at the same time and present an interface that makes it look as though they come from a single place?

It turns out that the major issue is distributed update, especially since a DBMS can justify itself in the business environment by supporting mission-critical applications. We have built a number of capabilities into SQL Server for

that support distributed database processing. One of those is the ability to do distributed updates against multiple SQL Servers using a two-phase commit protocol.

Today?

Exactly. For example, the first release of the product supports a scenario that has savings accounts on server A and checking accounts on server B, and a teller issues a transaction to debit savings, and credit checking. You want that transaction boundary to be respected, so that either both updates take place or neither one of them takes place, and you can roll back if necessary. SQL Server, I believe uniquely among relational databases, supports this distributed update capability.

Transact-SQL (the SQL language implementation used in SQL Server) has a syntax that defines Begin Transaction and End Transaction so you can define logical transaction boundaries.

Right. Now, the way distributed update is done today, it's not transparent to the programmer. It can be quite transparent to the end user, but the programmer has to explicitly code the logic that says savings is on A and checking is on B. Once that's done, though, the servers themselves cooperate at run time in a dynamic fashion to make sure that both updates took or that neither one of them took. In fact, that could apply whether it was two updates or ten updates; the servers themselves cooperate to make sure that it happens.

Another capability of SQL Server is the ability to do JOINs across multiple databases on a single server. So even though a database is a logically separate entity, you could have a JOIN that transparently links Customer table in database A and Order table in database B and presents a unified view to the application. The next step, obviously, is to evolve to the point where there is location information in front of the database information that is distributed around the network.

There has been considerable discussion of the performance of LAN Manager; Gordon Lettwin (the architect of OS/2) has been quoted to the effect that it has not been that competitive. It seems that Microsoft feels that the benefits of LAN Manager seem to emerge more clearly when multiple users are added to a network, and Novell tends to show low-level numbers that are very favorable even under small loads. Can you comment on that?

I don't think it's accurate to say that LAN Manager is slower than Novell across the board; in fact, we've done benchmarks that indicate LAN Manager is considerably faster in many areas.

Novell seems to tune up well at the low end of the processing spectrum. When you do tests that send small data packets across the network, Netware seems to do well. On large chunks of data, LAN Manager does better.

The related issue is that it's not just the LAN, it's what you use it for. In particular, if you look at the combination of LAN Manager and SQL Server as compared to Netware and anything else, whether it is Novell's own Btrieve engine or a a database server running on Netware, we think the side-by-side comparison would be very stark, with a very strong performance edge for the LAN Manager/SQL Server combination. This is partly on the strengths of SQL Server and partly on the fact that you're using the database server to get around what has been the wholedifficulty with multiuser PC database processing until now, and that is having to do explicit back-end record locking, navigating databases, network calls, importing entire files to the front end, and that sort of thing.

Also, as we've said all along, the direction for OS/2 is toward fast installable file systems. The nice thing about SQL Server being just an OS/2 application–we are, literally, a plain vanilla OS/2 application–is that we're able do deal with multiple users and networks through LAN Manager and with file systems through OS/2. When you slide in a new high-performance file system under OS/2, SQL Server sits there unchanged, except that it runs a lot faster because it's a disk-intensive application. So we will be one of the first big winners with a high-speed installable file system.

At this time, you don't support catalogued views?

Yes, we do. There is a system table in SQL Server called `sysobjects` that shows everything on the system – tables, views, triggers, stored procedures – including views (see Figure 1). A query on that table, where type equals V, will get you all the views in the system.

The one major thing you've been taken to task for most is the lack of the UNION verb...

We're adding that in; it's real obvious. We should have UNION in SQL Server during 1990. Meanwhile, we support temporary tables, which allow you to do a UNION-type function quite efficiently.

There are number of attractive features that are fairly unique in SQL Server: triggers, rules, defaults in the catalog, stored procedures. All these things are a significant extension to what is perceived as the benefits of an SQL-based database architecture. Can you address some of them?

Let's look first at the business requirements that call for these sorts of things. If you look at any business environment, there is so much interdependency between the pieces of information that a business deals with. For example, you have an accounting system that consists of a general ledger, an accounts payable module, a purchasing system, a human resource system, and an inventory management system.

All of these things have implicit rules or business policies underlying them. How do you enforce, for example, a straightforward and obvious business policy company-wide that says "Don't delete a customer if that customer has open orders?"

Referential integrity, in other words.

Yes. You can think of your own company's business life and realize that there are dozens or even hundreds of these business policies that constrain the way data is used. Otherwise, even though individual pieces of data may look fine, they're really getting corrupted or out of sync when you look at it globally across the whole data resource.

What this all boils down to is the need to enforce data integrity at the level of the database itself. That has been recognized as a need, even in the mainframe world, for years. Take the environment I was just describing, where you have separate applications all talking to what should be the same underlying resource of data–general ledger, accounts payable, inventory, and so on. You certainly have a need to maintain consistency and keep things in sync at the database level, but, at least in mainframe systems, the information resource was controlled by one organization, typically the corporate MIS group. The computer itself was probably in a locked room that was accessed by only a few people; the people who could get at information through the system were highly expert.

> *"The need to enforce data integrity at the level of the database itself has been recognized as a need for years."*

But over the next several years, many of these applications, just on the basis of sheer cost per MIP and cost per transaction, will migrate to the LAN environment. In order for that environment really to work on a plug and play basis, you have to be able to distribute the information resource out there using, let's say, SQL Server on an OS/2 LAN, and get at that information from a wide variety of front-end applications. These might be very sophisticated front ends such as application development tools; others may be the more casual user tools, like spreadsheets, that still have a need to get at that information.

In the near future, every major spreadsheet on the market will have the capability not only to query but also to update SQL databases over a LAN. How does that fit if one of those statements tries to delete Bill Jones from the Customer table when Bill Jones has $10,000 worth of outstanding orders?

You can't build the handling into the client piece; it has to be server-based.

That's the whole issue. In the current environment, using ordinary SQL technology, the only choice you have is to build it into the client, which is really an untenable situation in a LAN environment. So SQL Server contains a number of extremely important capabilities, including some of the ones you enumerated, that allow you to handle this sort of situation. One of the most important is a feature called a trigger.

A trigger is an implicit stored procedure; it is a way to enforce that referential integrity rule or business policy within the database itself. So even though you may have completely separate and disconnected tables for customer information and order information, if somebody comes in and issues a delete request against the customer table, this trigger fires and goes off and checks the order table.

It could actually go off and do any one of a number of other things, but the most straightforward example, to get back to the scenario we gave, is that it goes off and checks the order table and basically reports back–does Bill Jones have outstanding orders on this table? If the answer is yes, it disallows the delete and sends a message back to the application to let it know that the delete can't happen and the reason why. If the answer is no, then it goes ahead, assuming that this is the business policy, and deletes. This is a very robust way for ensuring referential integrity.

It doesn't matter where the delete request came from. This could be a single SQL Server being accessed by 50 different applications at the same time. It could have been that single spreadsheet user who issued a delete request, and SQL Server would stop it and trap it right at the level of the database itself.

When you think about the business policies that we were describing before, some of them boil down to the simple classic referential integrity problem. This almost just a set theory problem that occurs in relational databases, i.e., how do you make sure that the integrity of reference between the primary key (the customer table) and the foreign key (the order table) gets maintained.

It turns out, though, that if you step back to the business issues at hand, there are many other business problems that can't be reduced simply to a referential integrity check. An example might be something like a payroll system in which you want to determine the age bracket of the employee, where they live, social security number, number of dependents, things of that nature. Then, based on that information, a complex lookup process is performed against a variety of different tax tables and policies and finally a single figure is computed as the withholding amount on the employee's paycheck.

If you think about the process from beginning to end, it's a fairly complex set of programming logic...

It's a program.

Exactly. It's got branches, it's got loops, it's got flow control and the rest–but at the same time, it's the type of program that has implicit business integrity built in to it. If you don't do it properly, you're going to end up withholding too much or too little, you're going to have the government on your neck, or you're going to have the employee on your neck. It's the type of thing that ought be executed in a consistent fashion whether the request to issue the check came from a FOCUS program, a dBASE program, an Excel spreadsheet, or anywhere else.

Transact-SQL, the language supported by SQL Server, is a significant superset of ANSI-standard SQL that includes a number of important extensions. These include the ability to control branching and program flow right within a SQL procedure itself. That capability also enables what we call stored procedures, the ability to build programs that reside right on the server along with the data to which they refer. Those programs can be any combination of SQL statements, ANSI-standard as well as these extensions, like flow control, IF-THEN-ELSE, DO-WHILE. Stored procedures are also a mechanism that handles more arbitrarily

complex data integrity or business integrity logic than you could do with simple referential integrity checks. So we handle not only referential integrity but also data integrity of an arbitrary complexity.

The other nice thing about stored procedures is that they are pre-compiled and pre-optimized, yet you still have the flexibility of feeding it parameters at run time. So the procedure could be a generic way of handling payroll records, say, but you could feed it a specific employee name on a particular run. That also means that, in a transaction environment, this is one of a number of important reasons that our performance is so much better than ordinary SQL implementations.

Because you want to be able to deal with lots of short bursts of processing in a transaction environment, stored procedures can typically improve performance by a factor anywhere from five to ten times, compared to just executing the corresponding set of SQL statements each time you go through that transaction. We have just started benchmarking runs on the production release of SQL Server, and we're extremely impressed with the numbers we're seeing.

It seems clear that we are ready, from the software perspective, to provide tools that convert data models to schemas and programs. You have a very extendable product with lots of sockets, and FOCUS will be plugging in to it. Are you preparing design tools like that?

There are a number of third parties already doing this. Most of the CASE players in the marketplace have bought a SQL Server developer's kit, and I think a number of them are working on versions that will support SQL Server from the database design perspective.

At this time, you're not supporting record-level locking. Do you expect to in the future?

We support page-level locking. If you look at the sort of classic tradeoff curves of flexibility vs. response time, one thing that seems to be at the optimum point on both curves is page-level locking. If you lock at the table level, you limit concurrency. Record-level locking, on the other hand, requires much more overhead because records have variable lengths.

Page-level locking and stored procedures are two important reasons for the kind of performance we get. In our discussions with customers, nobody has told us that page locking limits their flexibility, so we expect to stick with the current architecture.

What about bypassing the lock entirely if the query is clearly read-only?

We support that too. SQL Server has two kinds of locks, shared and exclusive. Shared locks permit as many concurrent reads as you like.

Getting back to your question on querying, we also eliminate what are called livelocks, which occur when you have so many reads coming and queued up against the database that the one guy who wants to UPDATE is not able to get in. In a case like that, the underlying algorithm gets a little more sophisticated; basically, we figure out which of the reads bump to the bottom of the queue to allow the update to get through.

We haven't covered rules yet....

Well, this will wrap up the discussion on business policies and data integrity. There are a couple of other entities in SQL Server that are extremely useful. One is called defaults, which basically means just what it says. A certain field could automatically be initialized as the current date, for example, until a person entering data overrides that.

And this is done in the database language?

Right in the database itself. Getting back to the discussion of triggers and procedures, the key importance here is that all these things are centralized in the database in a place where they're controllable, as opposed to being dispersed throughout the many different front-end applications that might talk to the database. The other entity you asked about is a rule, or in relational terms, a domain constraint. If it refers

to a numeric value, for instance, the rule could ensure that the price has to fall between $5 and $10. Or, for discrete-valued fields, the rule could ensure that rate-type has to be L,E, or B, which could be codes meaning Labor, Equipment, or Both. Anybody accessing the database from FOCUS or DataEase or C or any other front end need not worry about that constraint in their application; it's been enforced at the database level.

Journalling is obviously a significant issue; it affects your ability to do rollbacks and deal with things like the logical transaction boundary. How does SQL Server control the size of the journal to minimize the use of space? Will the administrator have to go in and clean it up?

That's the typical environment we envision. Again, being a transaction engine, we never go in and blow away the journal, or the log as we call it, without being instructed to do so. In our extended SQL, there is a command to dynamically back up the database or the log online. So you can snapshot the log or the database, even while it's in use–there's a great deal of flexibility there.

We perform disk journalling, and in fact, if you walked over and pulled the plug on SQL Server while it's in the middle of running 50 transactions, the first thing it does when you bring power back is look at its log, run its recovery process, and rebuild all of its databases to the last point of good data integrity.

How do you prevent the file from growing too large?

The log file is pre-allocated, so it can't grow beyond its defined size. There are also commands to erase the log file up to any particular checkpoint. It'll typically be a very routine thing, a few SQL statements that could be executed once a week or daily, or whatever the installation requires.

Analyzing Master File Descriptions

Rodger Williams
JCPenney Financial Services

Master File Descriptions contain a wealth of information that can be used for a number of purposes. Rodger Williams demonstrates several methods for utilizing this information.

There are occasions when analyzing a Master File Description (MFD) would greatly benefit an application. For example, a common problem when designing a FOCUS application is the use of reserved words. This is particularly true when the reserved word has been used in an MFD for a field name or an alias. This technique demonstrates several programs used to analyze MFDs. The first checks for reserved words, the second provides a list of DBA users, and the third determines the number of full words in each segment and then calculates the number of filler bytes available.

The MFD used as a basis for analyzing other MFDs is shown in Figure 1. This is a copy of the COMASTER MASTER, which is included with all versions of FOCUS (except for PC). A slight modification has been made to change the field name at the bottom of the FIELDID segment from RESERVED to TITLE so that the TITLEs could be printed and also to prevent errors when TITLE is encountered in the MFD. Because this technique works on comma-delimited data, a successful way to prevent errors on DEFINEs in the MFD without commenting them out before running the programs has not been found. If there are just a few DEFINEs, the error messages are bearable. Additional segments can be added to prevent errors on new features, such as ACCEPT clauses.

The FOCEXEC in Figure 2 checks for reserved words. The first line prompts for the MFD name. The CMS STATE command checks all disks for the presence of the MFD before running the program, or the asterisk in the STATE command can be replaced with a specific disk letter if desired. FILEDEFs are set up for the MFD shown in Figure 1 and for the list of reserved words (distributed at the 1988 National FUSE in Orlando). The external file is called RESERVED WORDS, and one reserved word is placed on each line. If LET statements are used, consider adding any special abbreviations to the reserved word list (see Figure 3 on page 48 for the reserved word list).

The program shown in Figure 4 (on page 49) prints an alphabetized list of users on MFDs that have a Data Base Administrator (DBA). This FOCEXEC is similar to the DBATABLE FOCEXEC supplied with FOCUS.

The final utility (shown in Figure 5 on page 49) examines the MFD for alpha fields, sums the

Figure 1

```
FILENAME=COMASTER,SUFFIX=COM,
SEGNAME=FILEID
  FIELDNAME=FILENAME      ,FILE      ,A8,,$
  FIELDNAME=FILE_SUFFIX   ,SUFFIX    ,A8,,$
SEGNAME=RECID
  FIELDNAME=SEGNAME       ,SEGMENT   ,A8,,
  FIELDNAME=SEGTYPE       ,SEGTYPE   ,A4,,$
  FIELDNAME=SEGSIZE       ,SEGSIZE   ,I9,A1,$
  FIELDNAME=PARENT        ,PARENT    ,A8,,$
  FIELDNAME=CRKEY         ,VKEY      ,A12,,$
SEGNAME=FIELDID
  FIELDNAME=FIELDNAME     ,FIELD     ,A12,,$
  FIELDNAME=ALIAS         ,SYNONYM   ,A12,,$
  FIELDNAME=FORMAT        ,USAGE     ,A8,,$
  FIELDNAME=ACTUAL        ,ACTUAL    ,A8,,$
  FIELDNAME=AUTHORITY     ,AUTHCODE  ,A8,,$
  FIELDNAME=FIELDTYPE     ,INDEX     ,A8,,$
  FIELDNAME=TITLE         ,RESERVED  ,A44,,$
SEGNAME=DEFREC
  FIELDNAME=DEFINITION    ,DEF       ,A80,,$
SEGNAME=PASSREC,PARENT=FILEID
  FIELDNAME=READ/WRITE    ,RW        ,A32,,$
SEGNAME=CRSEG,PARENT=RECID
  FIELDNAME=CRFILENAME    ,CRFILE    ,A8,,$
  FIELDNAME=CRSEGNAME     ,CRSEGMENT ,A8,,$
  FIELDNAME=ENCRYPT       ,ENCRYPT   ,A4,,$
SEGNAME=ACCSEG,PARENT=DEFREC
  FIELDNAME=DBA           ,DBA       ,A8,,$
  FIELDNAME=USER          ,PASS      ,A8,,$
  FIELDNAME=ACCESS        ,ACCESS    ,A8,,$
  FIELDNAME=RESTRICT      ,RESTRICT  ,A8,,$
  FIELDNAME=NAME          ,NAME      ,A8,,$
  FIELDNAME=VALUE         ,VALUE     ,A80,,$
```

length of those fields, and divides that length by 4 to determine the number (in decimal format) of full words. Non-zero digits to the right of the decimal indicate that the segment has not been extended to the next full word boundary. (Note DATETEST in the DEFINE to check for old alpha date formats.) The data is placed in a HOLD file so that the decimal values may be converted to integer for evaluation in a DEFINE. For example, if the remainder is .25, there is 1/4 of a full word used, which leaves 3/4 of a full word (3 bytes) available at the end of the segment. The data from the HOLD file is then printed to display filler bytes available for each segment.

The MFD utilities that can be created by using these examples are limited only by imagination.

Figure 2

```
*&1 A8 Enter Master File name you want to check
CMS STATE &1 MASTER *
-IF &RETCODE NE 0 GOTO NOFILE;
CMS FILEDEF RESERVED DISK &1 MASTER
CMS FILEDEF RRR DISK RESERVED WORDS
TABLE FILE RESERVED
"List of Segnames That are Reserved Words"
PRINT SEGNAME BY SEGNAME NOPRINT
IF SEGNAME EQ (RRR)
END
-*****************************************
TABLE FILE RESERVED
"List of Fieldnames That are Reserved Words"
PRINT FIELDNAME ALIAS TITLE BY FIELDNAME NOPRINT
IF FIELDNAME EQ (RRR)
END
-*****************************************
TABLE FILE RESERVED
"List of Aliases That are Reserved Words"
PRINT ALIAS FIELDNAME TITLE BY ALIAS NOPRINT
IF ALIAS EQ (RRR)
END
-EXIT
-NOFILE
-TYPE Master file &1 not found.
```

Figure 3

Reserved word list

A	EXCLUDES	LOC	REPLOT
ABS	EXEC	LOCATE	RESTRICT
ACCESS	EXIT	LOCATION	RETYPE
ACROSS	EXPLAIN	LOG	ROW
ADD	EXSMO	LST	ROW-TOTAL
AGAIN	E01	LT	ROWS
ALL	E01	MARK	RPCT
ANALYSE	E02	MATCH	SAVB
AND	E03	MAX	SAVE
ANOVA	E04	MDISC	SCAN
AS	E05	MDY	SEG
ASQ	E06	MDYY	SEGMENT
AVE	E07	MIN	SEGNAME
B	E08	MODIFY	SET
BACK	E09	MOVE	SHOW
BIN	FACTO	MSG	SKIP-LINE
BINS	FILE	MT	SORT
BY	FIN	MTDY	ST
CDN	FINISH	MTRDY	STATSET
CHANGE	FIXFORM	MTRDYY	STOR
CHECK	FML	MULTR	STORE
CLEAR	FOCUS	N	SUB-TOTAL
CNT	FOOTING	NE	SUBFOOT
CO	FOR	NEXT	SUBHEAD
COLUMN	FORM	NOMATCH	SUBTOTAL
COLUMN-TOTAL	FORMAT	NOPRINT	SUM
COLUMNS	FREEFORM	NOT-FROM	SUMMARIZE
COMBINE	FROM	NOTOTAL	SUPPRINT
COMPILE	FST	OFFLINE	TABLE
COMPUTE	GE	OM	TABLEF
CONTAINS	GOTO	OMITS	TEO
CONTINUE	GRAPH	ON	TEMP
CORRE	GT	ONLINE	TLOCATE
COUNT	GTREND	OR	TO
CRTFORM	HEADING	OVER	TOP
CT	HIST	PAGE	TOT
C1	HOLD	PAGE-BREAK	TOTAL
C2	I	PANEL	TRACE
C3	IF	PASS	TYPE
DBA	IN	PAUSE	UNDERLINE
DECODE	INCLUDE	PCT	UP
DECRYPT	INCLUDES	PICKUP	UPDATE
DEFINE	INDEX	PIE	USAGE
DEL	INPUT	POLRG	USE
DELETE	INT	POST	USER
DIS	INVALID	PRINT	VALIDATE
DISPLAY	IS	PROMPT	WITHIN
DMTY	IS-LESS-THAN	QUIT	WRITE
DMY	IS-MORE-THAN	RANKED	XFER
DMYY	IS-NOT	READLIMIT	X1
DUPL	JOIN	REBUILD	X2
ECHO	JUMP	RECAP	X3
EDIT	LAST	RECOMPUTE	Y
ENCRYPT	LE	RECORDLIMIT	YMD
END	LET	RECTYP	YMTD
EQ	LINES	RECTYPE	YMTRD
EX	LINK	REORG	YYMTRD
EXCEEDS	LIST	REPLACE	

Figure 4

```
&1 A8 Enter name of master file you want to check
CMS STATE &1 MASTER *
IF &RETCODE NE 0 GOTO NOFILE;
CMS FILEDEF RESERVED DISK &1 MASTER *
TABLE FILE RESERVED
"DBA Access Information    "
"FILENAME &1 ON &DATE AT &TOD "
PRINT ACCESS AND RESTRICT AND NAME AND VALUE
BY USER UNDER-LINE
END
EXIT
NOFILE TYPE
TYPE Master file &1 not found
```

Figure 5

```
&1 A8 Enter master file name you want to check
CMS STATE &1 MASTER
IF &RETCODE NE 0 GOTO NOFILE;
CMS FILEDEF RESERVED DISK &1 MASTER *
DEFINE FILE RESERVED
FTYPE/A1=EDIT(USAGE,'9$$$$$$$');
DATETEST/A1=EDIT(USAGE,'$$9$$$$$');
SIZE/A3=IF DATETEST EQ 'Y' OR 'M' OR 'D' THEN EDIT(USAGE,'$9$$$$$$')
ELSE EDIT(USAGE,'$$99$$$$');
BYTES/I5=EDIT(SIZE);
WORDS/D7.2=BYTES/4;
END
TABLE FILE RESERVED
SUM BYTES WORDS BY SEGNAME IF FTYPE EQ 'A'
ON TABLE HOLD AS BYTES1
END
DEFINE FILE BYTES1
IWORD/I6=WORDS*100;
REMAINDER/A2=EDIT(IWORD,'$$$$99');
FILLER/A25=IF REMAINDER EQ '25' THEN '3 filler bytes available'
ELSE IF REMAINDER EQ '50' THEN '2 filler bytes available'
ELSE IF REMAINDER EQ '75' THEN '1 filler byte available'
ELSE 'No filler Bytes available';
END
TABLE FILE BYTES1
"Check of Available Filler Bytes"
"Filename &1 On &DATE at &TOD "
PRINT SEGNAME BYTES AS 'Alpha Bytes' FILLER AS 'Filler Available'
END
EXIT
NOFILE TYPE
TYPE MASTER FILE &1 NOT FOUND
```

Isolating Operating System Commands - OSFEX

Lawrence M. Breindel
BioSystems Integration

At the 1989 National FUSE, Gerry Cohen distinguished the FOCUS kernel from "presentation services" – those platform-specific elements that make each environment different. In this piece, Larry Breindel suggests an approach to enhanced portability aimed at minimizing those differences.

FOCUS itself is completely portable across all supported operating systems and platforms; however, operating system (OS) differences most frequently cause problems when porting code between environments. A complete FOCUS application can be viewed as a composite of database activity and a small envelope of OS specifics that allow the FOCUS application to interact with its environment. The FOCUS-specific components are indeed truly portable across all environments. The envelope of system specifics are, by definition, not portable at all.

Mainframe aficionados talk about 'virtual devices' – a virtual printer, for instance. Virtual devices share one nice property: their characteristics are theoretical and thus under the control of the system programmer. FOCUS fits this description of 'virtual devices' when an on-line table request is executed, for instance; the output device (the CRT screen) is so generic that, for the purposes of alphanumeric output, it can be ignored.

With the simple command OFFLINE, however, one crosses from the realm of the virtual output device (the CRT) to the world of real printers, with all their idiosyncrasies. The topic of printer management is complex even under a single OS; it is even more so when multiple printers are supported. When one attempts to support a VM/CMS system printer and then port to a UNIX environment, the issues multiply geometrically.

Another system-specific area involves associating physical, disk-resident files with the ddname (or file handle) used in FOCUS. For example, if there is data coming in each day to a CMS-resident FOCUS application from some sort of COBOL system, the FOCUS application could import from a file referred to as DAILY. On Monday, it is a physical file called MONDAY DATA A; on Tuesday, it is a different file, TUESDAY DATA B. The association between the physical file and its ddname is accomplished with a FILEDEF command.

Using the example above, two versions of this command are issued from within FOCUS. FILEDEF is a CMS command, so on VM/CMS, the syntax would be slightly different each day.

On Monday:

```
CMS FILEDEF DAILY DISK MONDAY DATA A
```

On Tuesday:

```
CMS FILEDEF DAILY DISK TUESDAY DATA B
```

In the PC environment, the FILEDEF command is a PC/FOCUS command, not a DOS command, so the syntax is slightly different:

```
FILEDEF DAILY DISK A:MONDAY.DAT
FILEDEF DAILY DISK B:TUESDAY.DAT
```

Indeed, if the application is writing data at the end of a file (for a transaction log, for example) it gets worse.

```
CMS FILEDEF MYLOG DISK MYLOG A (DISP MOD
```

must become:

```
DOS FILEDEF MYLOG DISK A:MY.LOG APPEND
```

As another example, the application might erase SAVE and HOLD files before the FILEDEF and actual data processing to prevent any contamination from prior runs. This may be accomplished from within FOCUS by the following commands:

```
CMS ERASE SAVEIT FOCTEMP A
```

or

```
DOS ERASE A:SAVEIT.FTM
```

Though the bulk of the FOCEXEC will port with absolutely no changes, some slight 'tweaking' will still be necessary. If there are only one or two FOCEXECs to deal with, this is a minor effort. If dozens or hundreds are involved, it becomes a major nightmare. Worse, there are now two versions of every FOCEXEC to be maintained. If they are different, which one is correct?

A Solution: Metalanguage

Coding FOCUS in an OS-neutral fashion can be achieved by isolating the system dependencies in a 'meta routine'. This term is analogous to a graphics 'metalanguage', which achieves portability across graphic environments as diverse as color monitors, laser printers, and pen plotters. In a portable graphics system, no routine writes to the hardware directly; rather, calls are made to hardware specific device-drivers. Graphics data is stored on disk as a series of device-independent codes that can be displayed on diverse output devices.

By analogy, no application FOCEXEC should interact directly with the true operating system. Instead, a FOCEXEC, called OSFEX, serves as an OS buffer. The application FOCEXECs make calls to OSFEX using the metalanguage, which consists of OS-independent terms for the common operations. OSFEX can then translate these into system-specific syntax. When the time comes to port the application, it contains no system dependencies. OSFEX achieves true portability across platforms.

Some key terms in this proposed metalanguage:

- &&OPSYS. Used to specify an operating system; initialized by a call to OSFEX using the SET OPSYS command.

- &&ALPHAFMT. Specifies an output format for word processing. Regardless of the operating system, the FOCUS code would be `ON TABLE SAVE &&ALPHAFMT`. &&ALPHAFMT can be FORMAT ALPHA for VM/CMS, FORMAT WP under DOS, and FORMAT DOC for VAX/VMS. The user should never need to specify this global variable; selection of &&SETOPSYS sets it automatically.

- SETPATH. Equates a single 'mode letter' with a path. Only VM/CMS has mode letters (omit for the moment the mode number, which is too arcane for this discussion). VMS, DOS, and UNIX use both letters and path names.

- ERASE and PRINT. As expected. This avoids system-specific equivalents, such as the DOS DEL command.

- FILEDEF. Associates the logical or virtual file handle with its physical counterpart.

- DISKFILE, TAPEFILE, and PRNTFILE tell FILEDEF where the output is to be routed: disk, tape, or printer.

Usage And Examples

Only initialization calls should be OS-specific; they appear only once within a FOCUS session. The calls to OSFEX for SETOPSYS and SETPATH should appear in the PROFILE FOCEXEC, enabling all other application FOCEXECs to ignore the issue and deal with a "virtual" OS. This concept is simply an extension of using the PROFILE to define the environment by specifying an OS. An alternative is to write a stand-alone FOCEXEC that can then be explicitly called from within an application. Whichever method is used, all application FOCEXECs are OS-neutral because all OS dependencies are isolated in calls to OSFEX.

Here are some examples:

- Set up the global variable &&OPSYS to initialize OSFEX for a default operating system by including this line in the PROFILE:

  ```
  EX OSFEX SETOPSYS CMS
  ```

  OSFEX translates this to:

  ```
  -SET &&OPSYS = CMS;
  -SET &&ALPHAFMT=ALPHA;
  ```

  When erasing a file is necessary, the line `EX OSFEX ERASE MY LOG A` could be included. Since &&OPSYS is set to CMS, OSFEX translates this to `CMS ERASE MY LOG A`.

- Set up DOS as the default OS, designate a PATH translation, and then erase a file. The following lines are inserted in the PROFILE:

  ```
  EX OSFEX SET OPSYS DOS
  EX OSFEX SET PATH A
  C:\USER\LARRY\BREINDEL
  ```

  OSFEX translates these to:

  ```
  -SET &&OPSYS=DOS  ;
  -SET &&PATH_A=C:\USER\LARRY\BREINDEL;
  ```

  At this point, the line `EX OSFEX ERASE MY LOG A` in a file translates to `DOS ERASE C:\USER\LARRY\BREINDEL\MY.LOG`.

- With DOS set as the operating system, FILEDEF for a running transaction log to disk or printer by using the following lines:

  ```
  EX OSFEX DISKFILE MYLOG MYLOG A APPEND
  EX OSFEX PRNTFILE MYLOG
  ```

  OSFEX would translate these to:

  ```
  FILEDEF MYLOG DISK
   C:\USER\LARRY\BREINDEL\MY.LOG APPEND
  FILEDEF MYLOG PRINTER
  ```

A version of OSFEX is shown in Figure 1; a flow chart is shown in Figure 2 (on page 57). The following numbered paragraphs correspond to the numbers in Figure 1.

1. The type of command to be executed is determined. In this example, the recognized commands are SET, ERASE, PRINT, and DISKFILE; other OS-specific commands could be included. This is a bit more complicated than simply passing variables in from the command line; the subroutine GETTOK is used to read the input line. The procedure then branches according to which type of command is to be executed.

2. If the input is SET OPSYS, OSFEX then reads the third word in the input string, which is the type of operating system. It then branches according to which operating system is specified and sets &&OPSYS and &&ALPHAFMT.

3. If the input is SET PATH, OSFEX reads in the path specified in the input string and sets the variable accordingly. The syntax shown enables several paths to be set at once.

4. The procedure branches to the label –SETUP_E_P_F if ERASE, PRINT, or DISKFILE was specified in the input string. GETTOK is then used to set the filename, filetype, etc. Once the file variables have

been set, they are concatenated together with the drive variable to set &FILEID.

5. &FILEID is used to implement the final ERASE, PRINT, or FILEDEF command.

Figure 3 (on page 57) shows a FOCEXEC that makes use of OSFEX to set up its environment parameters.

Figure 1

```
-* ==========================================================
-* OSFEX.FEX V2.0   4/17/89   Lawrence M. Breindel
-*
-* This is a demonstration FEX, not production code.
-* Use it as food for thought. Most especially, neither true error
-* handling nor efficient code have been attempted.
-* Also, FILETYPE is truncated to 3 bytes for all values of &&OPSYS.
-*
-* ==========================================================
-*   Echo and parse input
-*
1. -SET &WORD1 = GETTOK (&1, 30, 1, ' ', 8, 'A8') ;
-*
-* ==========================================================
-*   Command case statement
-*
-IF &WORD1 EQ 'SET'      GOTO DO_SET      ;
-IF &WORD1 EQ 'ERASE'    GOTO SETUP_E_P_F ;
-IF &WORD1 EQ 'PRINT'    GOTO SETUP_E_P_F ;
-IF &WORD1 EQ 'DISKFILE' GOTO SETUP_E_P_F ;
-*
-TYPE INITIAL COMMAND &WORD1 IS NOT LEGAL
-GOTO BAD_COMMAND ;
-RUN
-* ==========================================================
-*   Parse the set command
-*
-DO_SET
-SET &WORD2 = GETTOK (&1, 30, 2, ' ', 5, 'A5') ;
-IF &WORD2 EQ 'OPSYS'   GOTO DO_OPSYS   ;
-IF &WORD2 EQ 'PATH'    GOTO DO_PATH    ;
-*
-TYPE Operand to SET command  is not OPSYS or PATH
-GOTO BAD_COMMAND ;
-* ==========================================================
-*   Implement operating system initialization.
-*
2. -DO_OPSYS
-SET &WORD3 = GETTOK (&1, 30, 3, ' ', 3, 'A3') ;
-IF &WORD3 EQ 'DOS'     GOTO DO_DOS     ;
-IF &WORD3 EQ 'VMS'     GOTO DO_VMS     ;
-IF &WORD3 EQ 'CMS'     GOTO DO_CMS     ;
-*
-TYPE OPSYS is not specified as DOS, VMS or CMS.
```

(continued)

Figure 1 (continued)

```
     -GOTO  BAD_COMMAND  ;
     -RUN
     -*.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-
     -DO_DOS
     -SET &&OPSYS        =       'DOS'        ;
     -SET &&ALPHAFMT     = 'FORMAT WP'        ;
     -GOTO DONE_GOOD                          ;
     -RUN
     -* - - - - - - - - - - - - - - - - - - - -
     -DO_CMS
     -SET &&OPSYS        =       'CMS'        ;
     -SET &&ALPHAFMT     = 'FORMAT WP'        ;
     -GOTO DONE_GOOD                          ;
     -RUN
     -* - - - - - - - - - - - - - - - - - - - -
     -DO_VMS
     -SET &&OPSYS        =       'VMS'        ;
     -SET &&ALPHAFMT     = 'FORMAT DOC'       ;
     -GOTO DONE_GOOD                          ;
     -RUN
     -* =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-*
     -*   Implement PATH substitution (much like the DOS SUBSTR command).
     -*
3.   -DO_PATH
     -SET &WORD4 = GETTOK (&1, 40, 4, ' ' , 1, 'A1') ;
     -IF &WORD4 EQ 'K' GOTO DRIVE_K ;
     -IF &WORD4 EQ 'L' GOTO DRIVE_L ;
     -IF &WORD4 EQ 'M' GOTO DRIVE_M ;
     -RUN
     -* - - - - - - - - - - - - - - - - - - - -
     -DRIVE_K
     -SET &&DRIVEK = GETTOK (&1, 40, 3, ' ' , 20, 'A20') ;
     -GOTO DONE_GOOD ;
     -RUN
     -DRIVE_L
     -SET &&DRIVEL = GETTOK (&1, 40, 3, ' ' , 20, 'A20') ;
     -GOTO DONE_GOOD ;
     -RUN
     -DRIVE_M
     -SET &&DRIVEM = GETTOK (&1, 40, 3, ' ' , 20, 'A20') ;
     -GOTO DONE_GOOD ;
     -RUN
     -* =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-*
4.   -*   Setup FILEID for Erase, Print and Filedef commands.
     -*
     -SETUP_E_P_F
     -IF &WORD1 EQ 'ERASE'     GOTO MODE_1        ;
     -IF &WORD1 EQ 'PRINT'     GOTO MODE_1        ;
     -IF &WORD1 EQ 'DISKFILE'  GOTO MODE_2        ;
     -RUN
     -* - - - - - - - - - - - - - - - - - - - -
     -MODE_1
     -DOS_FILEID_1
     -VMS_FILEID_1
     -CMS_FILEID_1
     -SET &FILENAME = GETTOK (&1, 40, 2, ' ' ,  8, 'A08') ;
     -SET &FILETYPE = GETTOK (&1, 40, 3, ' ',   3, 'A03') ;
     -SET &FILEMODE = GETTOK (&1, 40, 4, ' ' ,  1, 'A01') ;
     -GOTO USE_FILEID ;
     -RUN
     -* - - - - - - - - - - - - - - - - - - - -
```

(continued)

Figure 1 (continued)

```
-MODE 2
-DOS_FILEID 2
-VMS_FILEID 2
-CMS_FILEID 2
-SET &HANDLE   = GETTOK (&1, 40, 2, ' ' ,  8, 'A08') ;
-SET &FILENAME = GETTOK (&1, 40, 3, ' ' ,  8, 'A08') ;
-SET &FILETYPE = GETTOK (&1, 40, 4, ' ' ,  3, 'A03') ;
-SET &FILEMODE = GETTOK (&1, 40, 5, ' ' ,  1, 'A01') ;
-SET &OPTION   = GETTOK (&1, 40, 6, ' ' ,  6, 'A06') ;
-GOTO USE_FILEID ;
-RUN
-* - - - - - - - - - - - - - - - - - - - - - -
-USE_FILEID
-RUN
-IF &&OPSYS EQ 'DOS'    GOTO DOS_BUILD  ;
-IF &&OPSYS EQ 'VMS'    GOTO VMS_BUILD  ;
-IF &&OPSYS EQ 'CMS'    GOTO CMS_BUILD  ;
-*
-TYPE OPSYS is not specified as DOS, VMS or CMS.
-GOTO BAD_COMMAND ;
-RUN
-* - - - - - - - - - - - - - - - - - - - - - -
-DOS_BUILD
-VMS_BUILD
-* - - - - - - - - - - - - - - - - - - - - - -
-IF &FILEMODE EQ 'K' GOTO USE_K ;
-IF &FILEMODE EQ 'L' GOTO USE_L ;
-IF &FILEMODE EQ 'M' GOTO USE_M ;
-GOTO BAD_COMMAND ;
-RUN
-* - - - - - - - - - - - - - - - - - - - - - -
-USE_K
-SET &FILEID = &&DRIVEK || '\' || &FILENAME || '.' || &FILETYPE ;
-GOTO DO_E_P_F
-* - - - - - - - - - - - - - - - - - - - - - -
-USE_L
-SET &FILEID = &&DRIVEL || '\' || &FILENAME || '.' || &FILETYPE ;
-GOTO DO_E_P_F
-* - - - - - - - - - - - - - - - - - - - - - -
-USE_M
-SET &FILEID = &&DRIVEM || '\' || &FILENAME || '.' || &FILETYPE ;
-GOTO DO_E_P_F
-* - - - - - - - - - - - - - - - - - - - - - -
-CMS_BUILD
-SET &FILEID = &FILENAME || '.' || &FILETYPE || ' ' || &FILEMODE ;
-GOTO DO_E_P_F
-RUN
-* - - - - - - - - - - - - - - - - - - - - - -
-DO_E_P_F
-IF &&OPSYS EQ 'DOS' AND &WORD1 EQ 'ERASE'  GOTO DOS_ERASE
-IF &&OPSYS EQ 'VMS' AND &WORD1 EQ 'ERASE'  GOTO VMS_ERASE
-IF &&OPSYS EQ 'CMS' AND &WORD1 EQ 'ERASE'  GOTO CMS_ERASE
-*
-IF &&OPSYS EQ 'DOS' AND &WORD1 EQ 'PRINT'  GOTO DOS_PRINT
-IF &&OPSYS EQ 'VMS' AND &WORD1 EQ 'PRINT'  GOTO VMS_PRINT
-IF &&OPSYS EQ 'CMS' AND &WORD1 EQ 'PRINT'  GOTO CMS_PRINT
-*
-IF &&OPSYS EQ 'DOS' AND &WORD1 EQ 'DISKFILE'  GOTO DOS_DISKFILE
-IF &&OPSYS EQ 'VMS' AND &WORD1 EQ 'DISKFILE'  GOTO VMS_DISKFILE
-IF &&OPSYS EQ 'CMS' AND &WORD1 EQ 'DISKFILE'  GOTO CMS_DISKFILE
-*
-GOTO BAD_COMMAND ;
-RUN
```

(continued)

Figure 1 (continued)

```
-* - - - - - - - - - - - - - - - - - - - -
-DOS_ERASE
DOS ERASE  &FILEID
-GOTO DONE_GOOD    ;
-RUN
-*
-VMS_ERASE
VMS DELETE &FILEID
-GOTO DONE_GOOD    ;
-RUN
-*
-CMS_ERASE
CMS ERASE  &FILEID
-GOTO DONE_GOOD    ;
-RUN
-* - - - - - - - - - - - - - - - - - - - -
-DOS_PRINT
DOS TYPE  &FILEID >LPT1:
-GOTO DONE_GOOD    ;
-*
-VMS_PRINT
VMS TYPE  &FILEID >LPT1:
-GOTO DONE_GOOD    ;
-*
-CMS_PRINT
CMS PRINT  &FILEID
-GOTO DONE_GOOD    ;
-RUN
-* - - - - - - - - - - - - - - - - - - - -
-DOS_DISKFILE
DOS FILEDEF &HANDLE DISK &FILEID &OPTION
-GOTO DONE_GOOD    ;
-*
-VMS_DISKFILE
VMS TYPE  &FILEID >LPT1:
-GOTO DONE_GOOD    ;
-*
-CMS_DISKFILE
CMS PRINT  &FILEID
-GOTO DONE_GOOD    ;
-RUN
-* =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
-BAD_COMMAND
-TYPE
-TYPE ***********************************************************
-TYPE ARGUMENT TO OSFEX WAS &1
-TYPE THIS IS NOT ON THE LIST OF LEGAL COMMANDS. OSFEX TERMINATES
-TYPE ***********************************************************
-TYPE
-GOTO DONE ;
-* - - - - - - - - - - - - - - - - - - - -
-DONE_GOOD
-* ANY HOUSE KEEPING CAN GO HERE
-GOTO DONE ;
-* - - - - - - - - - - - - - - - - - - - -
-DONE
-TYPE =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-  OSFEX COMPLETE  =-=-=-=-=-=-=
-EXIT
```

Figure 2

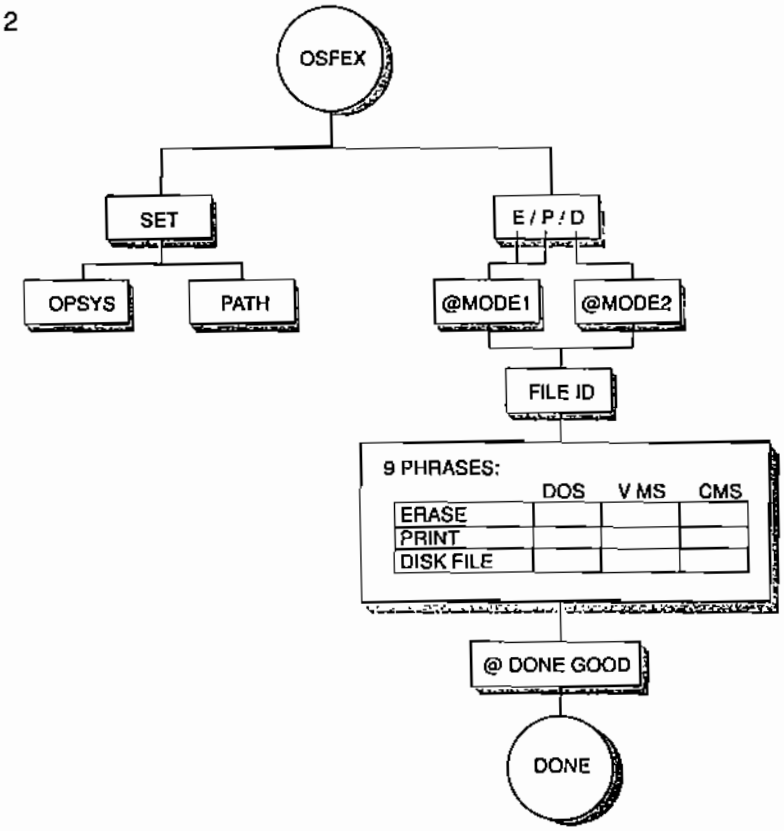


Figure 3

```
OSFEX OSFEX V2.0  V17/89  Lawrence M. Brandt
THIS IS A DEMONSTRATION ONLY, not production code!
EX C:\OSFEX SET OPSYS DOS
EX C:\OSFEX SET PATH C:\USER\TEST1
EX C:\OSFEX SET PATH C:\USER\TEST2
EX C:\OSFEX SET PATH C:\USER\TEST3
EX C:\OSFEX ERASE TEST*.DAT K
RUN
EX C:\OSFEX PRINT TEST\TEXT.C
RUN
EX C:\OSFEX DISK file2 MONDRIA TEST\DAC1
RUN
DELETE
RUN
```

Logical File Names

David Lovelace
Information Builders, Inc.

The concept of ddnames is useful in system development, as it facilitates porting code across environments. David Lovelace takes a look at the ways FOCUS associates logical file names with physical files.

Under all operating systems, FOCUS uses the concept of a logical file name, or data definition name (ddname). A ddname is the name by which either a data file or a device, such as a printer, is referenced in a FOCEXEC or an ad-hoc FOCUS session. It may be thought of as an alias, temporary name, or synonym for the data file name or device.

FOCUS reads or writes to a data file or device by referring to the associated ddname. TSO and VAX users find ddnames especially useful. On TSO, dataset names can be up to 44 characters long, not including membername specification when referencing a member of a PDS. VAX names (filespec) can be up to 255 characters long. It is much easier to remember and refer to an eight-character synonym.

ddnames are assigned to data files and devices via the FILEDEF command (short for File Definition) on all operating systems (except TSO, where the TSO ALLOCATE or FOCUS DYNAM command is used). The only files in most environments not requiring ddnames are FOCUS files, referenced via the FOCUS USE command; in TSO and WANG, however, FOCUS files are ALLOCATEd and FILEDEFed respectively. Data files can be either new files or previously existing ones.

Why FILEDEF?

FOCUS was originally developed in the CMS environment, where FILEDEF is a CMS command. When FOCUS was converted to other operating systems, either the operating systems did not have the ddname concept (PC DOS, UNIX, and OS/2) or the native operating system commands did not have the functionality needed by FOCUS (VAX and WANG). Consequently, IBI wrote its own FILEDEF command for those systems. The only exception was TSO, which already had a command, ALLOCATE, similar to the CMS FILEDEF command.

In the TSO environment, IBI wrote the FOCUS command DYNAM, which is functionally equivalent to ALLOCATE. It was originally written for the CICS interface and will be documented for general FOCUS use for the first time in the IBM FOCUS 6.0 manual. DYNAM has several advantages over ALLOCATE; it may be used to replace most TSO ALLOCATE, ATTRIB and FREE commands used from within FOCUS. One command instead of three makes life easier. DYNAM may also replace most DD statements in a batch job. In the past, users had to write two versions of a FOCEXEC, one for TSO and one to be run in batch, replacing any TSO ALLOCATE commands

with DD statements (TSO commands are ignored in batch). Now, in most cases, a single FOCEXEC will suffice because DYNAM may be used in either mode.

FOCUS could have been re-engineered for other operating systems to avoid the use of ddnames, but then FOCUS code would not be easily transportable across systems. The ddname concept is the same for all operating systems, allowing for more consistent FOCUS coding. The syntax for assigning ddnames is similiar across operating systems. Also, the FILEDEF command provides a level of flexibility not otherwise available. For instance, one type of functionality needed at times is the ability to define a file with fixed length records (RECFM F). UNIX, PC DOS, and OS/2 use variable length exclusively.

Another example is a report from a sales database for which the data files change monthly (JAN, FEB, MAR , etc.). To run reports for a particular month, the ddname SALES can be assigned to the relevant data file. When changing the data file name, only the ddname assignment statement needs changing. MODIFY and TABLE requests do not need to be changed, since they refer to files via ddnames. This is useful when porting FOCUS code between operating systems, as each operating system specifies file names differently. The ddname used on one operating system can be used under another operating system; FIXFORM ON JOBCO, where JOBCO is the ddname, is the correct FOCUS MODIFY syntax on all operating systems.

Syntax

The general syntax for FILEDEF, ALLOCATE, and DYNAM is shown in Figure 1.

The device most often is DISK (DA under TSO) to indicate that the ddname is to be assigned to a disk file. Other common devices are the terminal and printer; for TSO, CMS, WANG, and VAX, tape devices may also be specified.

There are many options available to enhance the power of FILEDEF and ALLOCATE:

LRECL, for Logical Record Length, specifies how long the records in the file can be.

RECFM, for Record Format, specifies the format of the file. RECFM is F or FB for fixed length records or V for variable length.

APPEND indicates that new records will be appended to an existing file. (It is specified as MOD for TSO ALLOCATE and FOCUS DYNAM, and DISP MOD for WANG and CMS FILEDEF.)

Normally it is not necessary to specify options such as LRECL for the printer or terminal devices, since the system defaults are adequate. In addition, many other options are available, some of which are operating-system dependent. For instance, one TSO option indicates whether a file is new or old; another specifies how much disk space is required.

FOCUS Use

Many FOCUS statements use ddnames. Commonly used ones are:

For MODIFY

```
FIXFORM ON ddname
FREEFORM ON ddname
DATA ON ddname
TYPE ON ddname *
LOG ON ddname *
```

Figure 1

CMS FILEDEF, UNIX:	`FILEDEF ddname device fileid (options`
DOS, OS/2, VAX:	`FILEDEF ddname device fileid options`
WANG:	`FILEDEF ddname device filename library volume (options`
TSO:	`TSO ALLOCATE F(ddname) DA(fileid) options`
	or
	`DYNAM ALLOC FILE ddname DA fileid options`

For TABLE

```
            [SAVE]
ON TABLE [SAVB] AS ddname
            [HOLD]
DECODE fieldname (ddname) *
IF fieldname EQ (ddname) *
```

For Dialogue Manager

```
-READ ddname *
-WRITE ddname *
-TYPE ON ddname *
```

The asterisk (*) indicates that the user MUST issue a FILEDEF or ALLOCATE for the ddname; in the other cases, FOCUS attempts a default assignment for the needed ddname and data file. Most defaults may be overridden by the user; FOCUS only tries to default if the user has not issued the ddname assignment.

Defaults

FOCUS assigns certain ddnames on an as-needed basis; work files such as FOCSTACK are one example. As another example, for the ON TABLE subcommand, the parameters SAVE, HOLD, and SAVB (without the "AS" parameter) are treated as ddnames themselves; FOCUS sets up a default FILEDEF for them and creates a data file named SAVE, HOLD or SAVB (except for TSO, which writes to a temporary dataset). When ON TABLE is used with "AS", the word following "AS" is taken as the ddname instead of SAVE, HOLD or SAVB, and the output is written to a file of the same name.* Most work files are referred to as FOCUS temporary files, i.e., FOCTEMP or .FTM files; others include LOTUS and WP files.

FOCUS also issues a default FILEDEF for the data file referenced by the Master File Description (MFD) the TABLE request uses if the suffix is FIX or COM. FOCUS first looks for a data file with the same name as the MFD with a filetype (extension) of DATA. If not found, it looks for a FOCTEMP file with the same name. The exception again is TSO, which looks for a dataset name of prefix.ddname.DATA.

In MODIFY FIXFORM and FREEFORM statements using ON DDNAME, FOCUS under CMS by default looks for a filetype of DATA with a filename the same as the ddname. Under TSO, FOCUS looks for a dataset named prefix.ddname.DATA. Other operating systems look for ddname.DAT.

FOCUS ddname defaults shows two reasons why the concept of ddname is valuable. A user who writes a FOCEXEC using only defaults does not have to worry about data file names, since FOCUS handles them. Secondly, converting a FOCEXEC to another operating system can be done much more easily, since, again, the user does not have to worry about the data file names.

For example, if the two TABLE requests shown below are issued consecutively, the second request would find the data in COUTYPE.

```
TABLE FILE CAR
PRINT COUNTRY BODYTYPE
ON TABLE HOLD AS COUTYPE
END

TABLE FILE COUTYPE
PRINT COUNTRY
END
```

Some ddnames are automatically assigned when FOCUS is entered. For example, users will notice that reports come to the terminal, since by default FOCUS writes the report to the ddname assigned to the terminal. The ddname to which the report is written can be changed by issuing the command OFFLINE, which tells FOCUS to use the ddname OFFLINE (assigned by default to the system printer.). (To see the initial FOCUS default ddname assignments, refer to the section *Querying ddname Assignments*.)

Under TSO, FOCUS also issues default ddname assignments for the MASTER, FOCEXEC and FOCCOMP datasets, as well as FOCUS files, if certain naming conventions are followed. In addition, WANG FOCUS issues default FILEDEFs for many standard ddnames (MASTER, FOCEXEC, FOCCOMP, FOCTEMP, etc.). The ddnames direct FOCUS to specific libraries and volumes where the files reside when the user enters FOCUS.

User ddname Assignments

In most cases, the user will issue ddname assignments, either because a FOCUS command (such as TYPE ON ddname) requires it or to override FOCUS defaults.

A user may wish to override FOCUS defaults because the default file names may not meet desired naming conventions or conform with those set up by the installation. For example, the default assignment for the ddname OFFLINE is the system printer. The user may wish to override the default to redirect a report to a different printer or a disk file. Another example is that the user does not desire ON TABLE SAVE to write the report output to a file named SAVE. This may be overridden by assigning SAVE to a file with a different name.

It should be noted that whenever a user assigns a ddname to an output file, especially a new one, file attributes such as LRECL and RECFM should be indicated. If these attributes are not specified, FOCUS will either try to default them or give an error message when the ddname is written to, depending on the operating system. If FOCUS defaults them, the format of the file may not be necessarily what is expected. As a general rule of thumb, always supply these attributes, especially when the data file will be used as input later, such as for a MODIFY. It should also be noted that LRECL and RECFM are always assigned by FOCUS for all HOLD files, overriding any user-supplied values.

Following are two examples using ddnames across operating systems.

1. This TABLE request writes a SAVE file to the ddname LSTCO.

```
TABLE FILE CAR
PRINT COUNTRY
ON TABLE SAVE AS LSTCO
END
```

FOCUS will default the ddname assignment to the FOCTEMP files that follow.

CMS:
```
LSTCO FOCTEMP
```

UNIX,DOS,OS/2,VAX:
```
LSTCO.FTM
```

WANG:
```
LSTCO (in the FTM library)
```

TSO:
a temporary dataset with the ddname LSTCO

2. The same TABLE request as above, except that the name of the output data file is changed to OUT1. The statements in Figure 2 assign the ddname LSTCO to the file OUT1.

Notice the option LRECL to specify the logical record length of the file; it should equal exactly the length of the TABLE output record. BLKSIZE is an option under

Figure 2

```
CMS:   CMS FILEDEF LSTCO DISK OUT1 DATA A (LRECL 10 RECFM F
UNIX:  FILEDEF LSTCO DISK OUT1.DAT (LRECL 10 RECFM F
DOS:   FILEDEF LSTCO DISK OUT1.DAT LRECL 10 RECFM F
OS/2:  FILEDEF LSTCO DISK OUT1.DAT LRECL 10 RECFM F
VAX:   FILEDEF LSTCO DISK OUT1.DAT LRECL 10 RECFM F
WANG:  FILEDEF LSTCO DISK OUT1 FOCDAT SYSTEM (LRECL 10 RECFM F
TSO:   TSO ATTR MYDCB LRECL(10) RECFM(F) BLKSIZE(10) CATALOG USING(MYDCB)
       TSO ALLOCATE F(LSTCO) DA(OUT1.DATA) NEW SPACE(5) TRACKS
       or
       DYNAM ALLOC FILE LSTCO DA MYID.OUT1.DATA NEW CATLG -
       SPACE 5 TRACKS LRECL 10 RECFM F BLKSIZE 8
```

TSO and should be specified for a new dataset. FOCDAT library under WANG is an arbitrary name.

This example shows the major difference between TSO assignments and those under other operating systems. When specifying new datasets on TSO, it is necessary to state how much disk space the dataset will use, dataset disposition, whether it is new or old, and dataset attributes such as LRECL and RECFM.

It is interesting to note that ALLOCATE and DYNAM have a purpose besides assigning a ddname. If the specified file did not previously exist, these two commands actually create it on disk, whether or not the file is ever used. This is in contrast to the other operating systems, where a new file assigned to a ddname is not created until it is written to.

Clearing ddnames

All ddnames stay assigned until they are cleared or freed either by the user or FOCUS. It is recommended that ddnames be cleared when no longer needed, especially those explicitly assigned by the user. Following is an example of clearing a ddname, where JOBCO is the ddname.

For CMS:

```
CMS FILEDEF JOBCO CLEAR
```

For DOS, OS/2, WANG, UNIX, VAX:

```
FILEDEF JOBCO CLEAR
```

For TSO: either

```
TSO FREE F(JOBCO)
or
DYNAM FREE FILE JOBCO
```

For the FILEDEF commands, an asterisk (*) specified instead of a ddname will clear all ddnames assigned at the time.

What happens to those ddnames not cleared by the user before exiting FOCUS depends upon the operating system. Under CMS and TSO, they are retained. Under DOS, OS2, WANG, UNIX, and VAX, they do not remain in effect, since FILEDEF was written for FOCUS use only under these operating systems. This includes the situation where a non-FOCEXEC program, such as a FORTRAN HLI program, is executed from within FOCUS: the ddname assignment to an input data file, for example, will not be known to the program. Other languages will not recognize ddname assignments made with FILEDEF. This is especially important to VAX users. If a COBOL or FORTRAN program is to be run under FOCUS, any input or output files must be assigned ddnames (logical names) via a native DCL command, such as ASSIGN. Using FILEDEF here will cause the program to return an error.

There is no problem with using FILEDEF and VMS ASSIGN together within FOCUS. But just as assignments made via FILEDEF are not known by other languages, so too are assignments made via ASSIGN not know by FOCUS.

Querying ddname Assignments

The ddname assignments currently in effect may always be displayed by issuing the relevant query command shown below.

For CMS:

```
CMS QUERY FILEDEF
```

For WANG:

```
VS QFI
```

For DOS, OS/2, UNIX, VAX:

```
? FILEDEF
```

For TSO:

```
? TSO DDNAME
```

For example, issuing the query command upon entering FOCUS shows the default ddnames initially used. Note that these listings may vary slightly from site to site (see Figure 3).

Notice how OFFLINE is always assigned to the printer. SYSIN is used under all operating systems, except for WANG (where INPUT is used), to specify where FOCUS commands will come from; SYSIN is normally assigned to

Figure 3

For CMS

```
SYSIN TERMINAL
SYSPRINT TERMINAL
OFFLINE PRT
```

For WANG

```
MASTER DISK * AMLMAS SYSTEM
FOCEXEC DISK * AMLFEX SYSTEM
ERRORS DISK * FOCERR QA000B
FOCTEMP DISK * AMLFTM SYSTEM
HOLDMAST DISK * AMLHMAST
   SYSTEM (LRECL 80 RECFM FB)
FOCCOMP DISK * AMLFCM SYSTEM
   (LRECL 2020 RECFM V)
FMU DISK * AMLFMU SYSTEM
   OFFLINE PRINTER
   INPUT TERM
   OUTPUT PRINTER
```

For DOS and OS/2

```
SYSIN       TERMINAL
SYSPRINT    TERMINAL
OFFLINE     PRINTER
FOCSTACK    C:FOCSTACK.FTM
```

For UNIX

DDNAME	DEVICE	RECFM	LRECL	FILENAME
SYSIN	TERMINAL	V	80	UPPER
SYSPRINT	TERMINAL	V	80	UPPER
OFFLINE	PRINTER	V	160	offline.ftm
FOCSTACK	DISK	F	102	focstack.ftm

For VAX

```
SYSIN       TERMINAL
SYSPRINT    TERMINAL
OFFLINE     PRINTERL
FOCSTACK    DISK$FPG:[FPGBSP]FOCSTACK.FTM
```

For TSO

DDNAME	OCCURRENCES	DSNAME
ERRORS	1	YIPFOC.PROD55.ERRORS.DATA
OFFLINE	1	JES2.TSU54666.S0000104
SYSPRINT	1	NULLFILE
SYSIN	1	NULLFILE

the terminal. SYSPRINT is the normal destination of FOCUS output and defaults to the terminal; issuing the FOCUS command OFFLINE overrides the default.

A number of default library names are assigned under WANG. UPPER or LOWER is displayed under FILENAME for WANG if the ddname is assigned to a device other than DISK; this indicates whether incoming data will be in upper case or lower. Note: LOWER and UPPER are additional options that may be specified with FILEDEF under all operating systems.

This TSO listing is an abbreviated list, since TSO allocates many other ddnames as needed. Those shown are those needed by FOCUS at entry time.

Using Dialogue Manager

Peter Van Der Lee
Information Builders, Inc.

In the March 1989 issue of the Systems Journal, *Gregory Dorman discussed the internals of the Dialogue Manager processor. In this article, Peter Van Der Lee uses an application to demonstrate how these Dialogue Manager principles work.*

Dialogue Manager is easy to use and has powerful features that allow for a great deal of programming flexibility, which places it among FOCUS' most used development tools. This article discusses some of the abilities of Dialogue Manager that can be used in applications, particularly those that are menu-driven. Very often we are faced with a situation where we want to present a value list from which to select one item in order to process a MODIFY request. However, a TABLE request cannot be issued from within a MODIFY procedure.

The two techniques discussed in this article demonstrate the relationship between Dialogue Manager and the FOCUS command processor. The first method simulates the execution of a TABLE request within a MODIFY procedure, using Dialogue Manager and HOTSCREEN. The second technique passes variables between –CRTFORM and TABLE.

The application used for both examples is that of a tracking system for technical memos, which contain technical information about a subject and are sent out to IBI's agents worldwide. The primary master file description (MFD), with a brief description of the fields, is shown in Figure 1. Note that the MFD contains a key sorted in descending order. Moreover, this MFD contains the ACCEPT clause for most of its fields in order to protect itself against ghost instances; the corresponding MFDs are shown in Figure 2 (on page 66).

Example One: The Search And Pick-Up Method

The first technique (shown in the FOCEXEC in Figure 3 on page 67) takes advantage of Dialogue Manager's flexibility, together with HOTSCREEN's save feature (the PF6 key). The technique allows TABLE requests to be executed as a subprocess of -CRTFORM. HOTSCREEN's save facility is used to pass on information from the TABLE request to the –CRTFORM. This will be referred to as the search and pick-up method. Moreover, values can be directly entered on the screen.

The great advantage of this method is that it allows users who are unsure about a specific value to obtain the entire list of possible values and choose one from this list. Values can be selected from four lists, and the PFKEYs are used to select a specific list. Once data values are collected, they aree processed in a MODIFY procedure. A CRTFORM is simulated by

Figure 1

```
FILENAME=INTTECH,    SUFFIX=DOC,
SEGNAME=DOCID,       SEGTYPE=SH1,
  FIELD=DOCNUM,  DNUM,  I7,  DMYD, INDEX=I, $
  FIELD=DOCTYP,  DTYP,  A1, ACCEPT=T' OR 'Z' OR 'A' OR 'B' OR 'N' $
  FIELD=ENVIRNMT, ECODE,  A6, ACCEPT=FIND(ECODE AS MACHCODE IN ENVIRNMT), $
  FIELD=INTFACE, IFACE,  A6, ACCEPT=FIND(INTFACE IN INTERFAC), $
  FIELD=PROTYP,  PTYP,  A6, ACCEPT=FIND(PROTYP IN PRODUCT), $
  FIELD=FOCREL,  FREL,  A5, ACCEPT=FIND(FREL IN FOCREL), $
  FIELD=XMI,     XMI,   A1, ACCEPT=N' OR 'Y' $
  FIELD=ADATE,   ADT,   DMY,
  FIELD=TITLE,   TITLE, A50,
  FIELD=KEY1 $,  AA12, ACCEPT=FIND(KEY1 AS KWORD IN KEYTECH), $
  FIELD=KEY2 $,  AA12, ACCEPT=FIND(KEY2 AS KWORD IN KEYTECH), $
  FIELD=KEY3 $,  AA12, ACCEPT=FIND(KEY3 AS KWORD IN KEYTECH), $
SEGNAME=LINETXT, SEGTYPE=S1, PARENT=DOCID, $
  FIELD=LINE#,   LIN#,  I4,
  FIELD=CONTENT, CONT,  A78,
```

```
DOCNUM  : Document number, key is sorted in descending order,
DOCTYP  : Document type, T = Technical Document, Z = Zap,
ENVIRNMT : Environment (PC, VAX, UNIX, ...)
INTFACE : Interface (SQL, DB2, ...)
PROTYP  : Product Type
FOCREL  : FOCUS release (5.5.5, 3.1, ...)
XMI     : Cross Machine Interface ?
ADATE   : Article Date
TITLE   : Title of article
KEY1    : First keyword (an article can have as many as three)
KEY2    : Second keyword
KEY3    : Third Keyword
LINE#   : Line number of article
CONTENT : TEXT of a particular line
```

–CRTFORM, enabling the Dialogue Manager variables to be the means of communication between CRTFORM and –CRTFORM.

The coding technique to accomplish this is based on the following steps:

1. Declare Dialogue Manager variables for corresponding MFD fieldnames.

2. Create a separate CASE for each list that can be brought up for consultation.

3. Use PFKEYs to branch to a specific list.

4. Ask the user to use HOTSCREEN's PF06 key to select a specific value, which will be written to a SAVE FOCTEMP file.

5. Read the value from this SAVE file and assign it to the corresponding Dialogue Manager variable.

6. Branch back to the first screen.

Steps 1 through 6 describe the search and pick-up data selection method. After these steps, a MODIFY procedure is used to process the transactions. Within the MODIFY procedure, the fieldnames inherit the values of their Dialogue Manager equivalents. At this point, all fieldnames values are known, and processing could be finished without any user action. The primary key, DOCNUM, is generated automatically. If no document number is entered on the data entry screen, the MODIFY procedure generates one automatically by incrementing the last existing document number.

Incrementation of the document numbers is done in the SNUM case. Because the numbers are stored in a decreasing order (SH1 segtype), the procedure simply repositions to the beginning of the database, takes the first number it finds (and thus last included one), and increments it. Based on whether the document number existed or not, the record needs to be updated or included. Inclusion is done in the INSERT case; updating is done in the

CHANGE case. Both cases give a summary of the values that were passed on before doing the related database actions.

If we have a careful look at the procedure, the following general flow can be seen. The line numbers correspond to those in Figure 3.

- Line 1 – Initialization of the Dialogue Manager variables by –SET.
- Line 2 – The starting point of the data entry screen.
- Line 3 – The screen itself.
- Line 6 – A –IF test on &PFKEY is used to branch to the TABLE requests.
- Line 7 – The actual MODIFY procedure (note the use of the SET ACCBLN syntax, which allows blanks and zeros for all ACCEPT clauses).
- Line 9 – Mandatory statement to flush the FOCSTACK (see following discussion).
- Line 10 – After processing a specific document, all is reinitialized.
- Line 11 – Label for a specific TABLE request
- Line 12 – The TABLE request itself.
- Line 14 – Mandatory statement to flush the FOCSTACK (see following discussion).
- Line 15 – Branching to the data entry screen, bringing in the selected values (no reinitialization).

Figure 2

```
FILENAME=ENVIRNMT,   SUFFIX=FOC,$
 SEGNAME=TYPE,   SEGTYPE=S1,$
  FIELD=MACHCODE, ECODE, A6, INDEX=I,$
  FIELD=MACHDESC,   ENAME, A15,$
```

```
FILENAME=RELATION,   SUFFIX=FOC,$
 SEGNAME=TYPE,SEGTYPE=S1, $
  FIELD=INTFACE, IFACE, A6,FIELDTYPE=I,$
  FIELD=IN_DESC, INAME, A15, $
```

```
FILENAME=PRODUCT,    SUFFIX=FOC , $
  SEGNAME=TYPE, SEGTYPE=S1,$
   FIELD=PROTYP ,PTYP, A6, FIELDTYPE=I,$
   FIELD=PRODESC, PNAME,A15,$
```

```
FILENAME=KEYTECH,SUFFIX=FOC
 SEGNAME=KEYINDEX,SEGTYPE=S1
  FIELDNAME=KWORD,ALIAS=KW, FORMAT=A12,INDEX=I,$
```

Dialogue Manager is actually nothing more than a preprocessor. When a line starts with the special character "–", it is considered a Dialogue Manager statement and is processed as such. Lines beginning with a character other than "–" are processed by the command processor. In this example, interaction between the two processes is as follows: the Dialogue Manager processor starts processing the "–" lines and continues until a –RUN, –EXIT, –QUIT or an End Of File is encountered. During such a session, Dialogue Manager executes lines starting with a "–" and writes all others to FOCSTACK. The –RUN indicates the end of a Dialogue Manager session.

At this point, Dialogue Manager writes a special line to FOCSTACK and dumps the session's Dialogue Manager values into this line. The special line is recognized by the FOCUS command processor, at which point it executes (flushes) the FOCSTACK. Then a second Dialogue Manager session is started, beginning at the line following the –RUN command. The next paragraph explains how this mechanism applies to the example shown in Figure 3.

Assume this sequence of events occurs: the user presses PF04 to obtain the environment list and selects the value "TSO" from the displayed list using HOTSCREEN's SAVE facility. The user then presses ENTER to process the MODIFY request.

Based on these assumptions, the first Dialogue Manager session begins at lines 1 through 6, continues at line 11, and finally finishes at line 14. At this point, the FOCSTACK contains line 12 and 13 (the TABLE request). The first Dialogue Manager session terminates with the –RUN statement; the special line is written to the FOCSTACK, which signals the start of execution of the commands that have been written to the FOCSTACK. In this example, this means that the TABLE request gets executed. Specific values can then be selected from the displayed report using Hotscreen's PF06 key, and the selected value is saved to a temporary file.

A second Dialogue Manager session is started, beginning at line 14. The saved value is read

(continued on page 70)

Figure 3

```
1.  -INITIAL
    -SET &DOCVAR = '          ';
    -SET &ECODE  = '          ';
    -SET &PROTYP = '          ';
    -SET &DTYP   = ' '        ;
    -SET &IFACE  = '          ';
    -SET &FREL   = '          ';
    -SET &XMI    = ' '        ;
    -SET &ADATE  = &DMY       ;
    -SET &KEY1   = '                ';
    -SET &KEY2   = '                ';
    -SET &KEY3   = '                ';
    -SET &TITLE  = '                                        ';
2.  -START
    -SET &PFKEY='     ';
3.  -CRTFORM
    -" ENTER DOCUMENT NUMBER OF EXISTING MEMO / LEAVE BLANK FOR NEW MEMO"
    -" "
    -"  <15 DOCUMENT NUMBER: <.UT.&DOCVAR>      "
    -" "
    -" PROVIDE INFORMATION (NEW MEMOS ONLY)     "
    -" "
    -"ENVIRONMENT..: <.UT.&ECODE> <40 INTERFACE ...: <.UT.&IFACE> "
    -"PRODUCTTYPE..: <.UT.&PROTYP><40 FOCUSRELEASE.: <.UT.&FREL>(5.5.0/3.1)"
    -"CROSS MACHINE: <.UT.&XMI>   <40 DOCUMENT TYPE: <.UT.&DTYP>  "
    -"KEYWORD 1....: <.UT.&KEY1>  <40 KEYWORD 2....: <.UT.&KEY2>  "
    -"KEYWORD 3....: <.UT.&KEY3>  <40 MEMO DATE....: <.UT.&ADATE> "
    -"TITLE........: <.UT.&TITLE> "
    -" "
4.  -"PRESS ENTER TO PROCESS                    "
    -"      PF03  TO QUIT                       "
5.  -"      PF04  TO SEE LIST OF ENVIRONMENTS   "
    -"      PF05  TO SEE LIST OF INTERFACES     "
    -"      PF06  TO SEE LIST OF PRODUCT TYPES  "
    -"      PF09  TO SEE LIST OF KEYWORDS       "
    -"      PF10  TO SEE LIST OF DOCUMENT TYPES"
6.  -IF &PFKEY EQ PF04 THEN GOTO EVIRN ;
    -IF &PFKEY EQ PF05 THEN GOTO INTER ;
    -IF &PFKEY EQ PF06 THEN GOTO PRODU ;
    -IF &PFKEY EQ PF09 THEN GOTO KWORD ;
    -IF &PFKEY EQ PF10 THEN GOTO DOCTP ;
    -SET &DOCNUM= IF &DOCVAR EQ '          ' THEN 0 ELSE &DOCVAR ;

    SET ACCBLN=ON
7.  MODIFY FILE DOCUMENT
    COMPUTE CURSOR/A12=;
            DOCNUM    =  &DOCNUM  ;
    MATCH DOCNUM
      ON MATCH GOTO CHANGE
      ON NOMATCH IF DOCNUM EQ 0 THEN GOTO SNUM;
      ON NOMATCH TYPE "THIS DOCNUM DOES NOT EXIST LEAVE BLANK FOR NEXT DOCNUM"
      ON NOMATCH GOTO EXIT

    CASE SNUM
    REPOSITION DOCNUM
    NEXT DOCNUM
      ON NEXT COMPUTE DOCNUM = D.DOCNUM + 1;
      ON NONEXT COMPUTE DOCNUM = 1;
      ON NEXT/NONEXT GOTO INSERT
    ENDCASE

    CASE INSERT
    COMPUTE ECODE  = '&ECODE'  ;
            PROTYP = '&PROTYP' ;
```

(continued)

Figure 3 (continued)

```
        DTYP    = '&DTYP'   ;
        IFACE   = '&IFACE'  ;
        FREL    = '&FREL'   ;
        XMI= IF '&XMI' EQ 'Y' THEN 'Y' ELSE 'N';
        ADATE   = '&ADATE'  ;
        KEY1    = '&KEY1'   ;
        KEY2    = '&KEY2'   ;
        KEY3    = '&KEY3'   ;
        TITLE   = '&TITLE'  ;
        IDT/DMY=IF D.IDT EQ 0 THEN &DMY ELSE D.IDT;
CRTFORM LINE 1
"*****************************************************************"
"***   PRESS ENTER TO CONFIRM THE ADDITION OF THIS NEW RECORD     ***"
"*****************************************************************"
" "
" <15 DOCUMENT NUMBER: <D.DOCNUM>   "
" "
" ENVIRONMENT..: <T.ECODE> <40 INTERFACE....: <T.IFACE>"
" PRODUCTTYPE..: <T.PROTYP><40 FOCUS RELEASE: <T.FREL>"
" CROSS MACHINE: <T.XMI>   <40 DOCUMENT TYPE: <T.DTYP> "
" KEYWORD 1....: <T.KEY1>  <40 KEYWORD 2....: <T.KEY2> "
" KEYWORD 3....: <T.KEY3>  <40 MEMO DATE....: <T.IDT> "
" TITLE........: <:A.T.TITLE>"
" "
" PRESS ENTER TO PROCESS              "
"       PF03  TO QUIT WITHOUT SAVING "
COMPUTE CURSOR=':A';
VALIDATE
  TITTEST = (TITLE NE ' ');
  ON INVALID TYPE "A DOCUMENT TITLE MUST BE SUPPLIED   "
   ON INVALID GOTO INSERT
MATCH  DOCNUM
 ON NOMATCH INCLUDE
 ON MATCH/NOMATCH GOTO EXIT
ENDCASE

CASE  CHANGE
COMPUTE ECODE   = D.ECODE    ;
        PROTYP  = D.PROTYP   ;
        DTYP    = D.DTYP     ;
        IFACE   = D.IFACE    ;
        FREL    = D.FREL     ;
        XMI     = D.XMI      ;
        ADATE   = D.ADATE    ;
        KEY1    = D.KEY1     ;
        KEY2    = D.KEY2     ;
        KEY3    = D.KEY3     ;
        TITLE   = D.TITLE    ;
         IDT/DMY=IF D.IDT EQ 0 THEN &DMY ELSE D.IDT;
CRTFORM LINE 1
"*****************************************************************"
"***   PRESS ENTER TO CONFIRM THE UPDATE OF THIS EXISTING RECORD  ***"
"*****************************************************************"
" "
" <15 DOCUMENT NUMBER: <D.DOCNUM>   "
" "
" ENVIRONMENT..: <T.ECODE> <40 INTERFACE....: <T.IFACE>"
" PRODUCTTYPE..: <T.PROTYP><40 FOCUS RELEASE: <T.FREL>"
" CROSS MACHINE: <T.XMI>   <40 DOCUMENT TYPE: <T.DTYP> "
" KEYWORD 1....: <T.KEY1>  <40 KEYWORD 2....: <T.KEY2> "
" KEYWORD 3....: <T.KEY3>  <40 MEMO DATE....: <T.IDT> "
" TITLE........: <:A.T.TITLE>"
" "
" PRESS ENTER TO PROCESS "
```

(continued)

Figure 3 (continued)

```
     "        PF03  TO QUIT WITHOUT SAVING "
     COMPUTE CURSOR=':A';
     VALIDATE
       TITTEST = (TITLE NE ' ');
       ON INVALID TYPE "A DOCUMENT TITLE MUST BE SUPPLIED "
       ON INVALID GOTO CHANGE
     MATCH  DOCNUM
       ON MATCH UPDATE DTYP ECODE IFACE PROTYP FREL XMI IDT
       ON MATCH UPDATE TITLE KEY1 KEY2 KEY3
       ON MATCH/NOMATCH GOTO EXIT
     ENDCASE

     LOG INVALID MSG OFF
     LOG NOMATCH MSG OFF
     LOG DUPL    MSG OFF
     DATA VIA FI3270
  8. END
  9. -RUN
 10. -GOTO INITIAL

 11. -EVIRN
     -CMS ERASE SAVE FOCTEMP A
     -CMS FILEDEF SAVE DISK SAVE FOCTEMP A (RECFM F LRECL 80
     -SET &ECODE ='        ' ;
     -WRITE SAVE &ECODE
 12. TABLE FILE ENVIRNMT
     "SELECT THE ENVIRONMENT BY MOVING THE CURSOR TO THE BEGINNING OF"
     "THE WORD AND PRESSING PF6"
     " "
     PRINT MACHCODE AS 'ENVIRONMENT' MACHDESC AS 'DESCRIPTION'
 13. END
 14. -RUN
 15. -READ SAVE &ECODE.A6.
 16. -GOTO START

     -INTER
     -CMS ERASE SAVE FOCTEMP A
     -CMS FILEDEF SAVE DISK SAVE FOCTEMP A (RECFM F LRECL 80
     -SET &IFACE = '        ' ;
     -WRITE SAVE &IFACE
     TABLE FILE INTERFAC
     "SELECT THE INTERFACE BY MOVING THE CURSOR TO THE BEGINNING OF"
     "THE WORD AND PRESSING PF6"
     " "
     PRINT INTFACE AS 'INTERFACE' IN_DESC AS 'DESCRIPTION'
     END
     -RUN
     -READ SAVE &IFACE.A6.
     -GOTO START

     -PRODU
     -CMS ERASE SAVE FOCTEMP A
     -CMS FILEDEF SAVE DISK SAVE FOCTEMP A (RECFM F LRECL 80
     -SET &PROTYP = '        ' ;
     -WRITE SAVE &PROTYP
     TABLE FILE PRODUCT
     "SELECT THE PROTOTYP BY MOVING THE CURSOR TO THE BEGINNING OF"
     "THE WORD AND PRESSING PF6"
     " "
     PRINT PROTYP AS 'PROTOTYP' PRODESC AS 'DESCRIPTION'
     END
     -RUN
     -READ SAVE &PROTYP.A6.
     -GOTO START
```

(continued)

Figure 3 (continued)

```
-DOCTP
-CRTFORM LINE 1
-" "
-" THE FOLLOWING IS A LIST OF VALID DOCUMENT TYPES"
-" "
-" B : BULLETIN BOARD "
-" N : NEW FEATURE "
-" T : TECHNICAL MEMO "
-" Z : ZAPS "
-" "
-" "
-" PRESS ENTER TO CONTINUE "
-GOTO START

-KWORD
-CMS ERASE SAVE FOCTEMP A
-CMS FILEDEF SAVE DISK SAVE FOCTEMP A (RECFM F LRECL 80
-SET &DUMMY='
-WRITE SAVE &DUMMY
-RUN
SET PAUSE=OFF
SET MSG ON
TABLE FILE KEYTECH
"<TABPAGENO"
ON TABLE SUBHEAD
"SELECT UP TO 3 KEYWORDS BY MOVING THE CURSOR TO THE BEGINNING OF"
"THE WORD AND PRESSING PF6"
PRINT KWORD
END
-RUN
-DEFAULT &KEY1='                    ',&KEY2='                    '
-DEFAULT &KEY3='                    ',&KEYY='                    '
-SET &I=1;
-LOOP1
-READ SAVE &KEYY.A12.
-IF &IORETURN NE 0 GOTO START ;
-SET &KEY.&I=&KEYY;
-SET &I=&I+1;
-IF &I LE 3 GOTO LOOP1 ;
-GOTO START
```

into the Dialogue Manager variable &ECODE in line 15. Line 16 contains a –GOTO statement that forces Dialogue Manager to continue its process at the specified label. The –START label shows a –CRTFORM screen that will display the newly selected value.

When the –CRTFORM screens pops up again, the user presses the ENTER key, which causes the direction of lines 7 and 8 (the MODIFY request) to the FOCSTACK. Line 9, the important –RUN statement, indicates that the FOCSTACK will be flushed but not until after the Dialogue Manager values are dumped into it. This dump is an important communication possibility between Dialogue Manager and the command processor. Finally, after executing the FOCSTACK, a third Dialogue Manager session will be started and so on.

Other Methods Of Sharing Variables

There are other ways to share Dialogue Manager context with the command processor. The most common way is to use –INCLUDE. Instead of coding the MODIFY and TABLE requests within one FOCEXEC, a different FOCEXEC could be created for each TABLE and MODIFY request. The –INCLUDE FOCEXEC commands can then be used in the main program. The advantage of this method is obvious; it reduces the amount of code in the main program.

On the other hand, more FOCEXECs are used to accomplish the same effect. It is more a question of taste as to which method is preferable. Note that –INCLUDE cannot be re-

placed by EX FOCEXEC, since this starts a new session that will not share its context with the calling session unless parameters are passed on the calling line or global variables (&&) are used.

Example 2: –CRTFORM And TABLE

Another interesting FOCEXEC based on the interaction between the Dialogue Manager processor and the FOCUS command processor is shown in Figure 4 (on page 72). In order to trace the Tech Memos, the user should be given a great deal of flexibility. One user might remember a certain article date or have an idea in what range the requested document resides. Others might want to obtain any available information on a certain subject (keyword search) and/or screen on specific environments, interfaces, or other criteria.

This is what happens at execution time:

- A –CRTFORM menu appears, allowing the user to specify a document number range, a memo date range, environment, interface, product type, FOCUS release, cross machine interface, document type, and up to three keywords.
- The search and pick-up method demonstrated in Example 1 can be used to determine environments, interfaces, product types, and keywords.
- Based on the information supplied on the –CRTFORM, a list of document numbers and titles will appear. At this point, the user can decide which document(s) to view.
- The selected documents are then displayed.

From a logical point of view, the FOCEXEC flows as follows (the line numbers correspond to those in Figure 4).

- Line 1 – Initialization of Dialogue Manager variables.
- Line 2 – Start and returning point of data entry screen.
- Line 3 – Data entry screen.
- Line 4 – Branching to a specific label.
- Line 7 – Define statement to select key word in TABLEF request.
- Line 18 – End of define statement.
- Line 19 – Flushing the FOCSTACK.
- Line 20 – TABLEF request to determine a list of candidates to be consulted.
- Line 53 – End of TABLEF request.
- Line 54 – Flushing the FOCSTACK.
- Line 64 – TABLE request of selected candidates in 20.
- Line 82 – End of TABLE request.
- Line 83 – Flushing the FOCSTACK.
- Line 84 – Branching back to data entry screen.
- Line 85 – Label to show list of environments, product type, etc.
- Line 86 – TABLE request to show the list (search-pick up method).
- Line 87 – End of the TABLE request.
- Line 88 – Flushing the FOCSTACK.

Suppose we want to know whether any Tech Memos exist on the subject of memory considerations for PC/FOCUS–MultiUser Release 3.0. After executing the FOCEXEC, the data entry screen appears. The default document range is from 0 to 9999999; in this example, these default values are unchanged. The environment code is PCMU; the keyword is MEMORY. Both values can be found by the search and pick-up method in case the correct code names are not known. Moreover, screening can be done on the keyword MEMORY.

The rest of the fields are left blank because they are not applicable to the example (interface, product type, and cross machine interface) and because in order to obtain the largest possible data selection, no screening will be done on the field DOCUMENT TYPE. After typing in the data, the user presses ENTER to process.

At this point, the following values have been assigned to the variables:

```
&KEY1   = MEMORY
&KEY2   = '        '
&KEY3   = '        '
&DOCNUM1= '0     '
&DOCNUM2= '9999999'
&ECODE = PCMU
```

(continued on page 74)

Figure 4

```
1.  -SET &KEY1    = '              ' ;
    -SET &KEY2    = '              ' ;
    -SET &KEY3    = '              ' ;
    -SET &DOCNUM1= '0       '        ;
    -SET &DOCNUM2= '9999999'         ;
    -SET &ECODE   = '       '        ;
    -SET &PROTYP = '       '         ;
    -SET &DTYP    = ' '              ;
    -SET &IFACE   = '       '        ;
    -SET &FREL    = '       '        ;
    -SET &XMI     = ' '              ;
    -SET &ADATE1 = '       '         ;
    -SET &ADATE2 = '       '         ;
    -SET &TITLE   = '                                    ' ;
2.  -START
    -SET &PFKEY='    ';
3.  -CRTFORM LINE 1
    -"THIS PROCEDURE TRACES TECH MEMOS"
    -" "
    -"PROVIDE AVAILABLE INFORMATION TO IDENTIFY TECHMEMO"
    -"    "
    -"    "
    -"DOCUMENT RANGE : <.UT.&DOCNUM1 (FROM)<53 : <.UT.&DOCNUM2 (TO)"
    -"MEMO DATE RANGE: <.UT.&ADATE1  (FROM) (DMY EX 071288) <0X
    - <53 : <.UT.&ADATE2  (TO)"
    -"ENVIRONMENT     : <.UT.&ECODE  <40 INTERFACE     : <.UT.&IFACE  "
    -"PRODUCTTYPE     : <.UT.&PROTYP<40 FOCUSRELEASE : <.UT.&FREL (5.5.0/3.1)"
    -"CROSS MACHINE   : <.UT.&XMI    <40 DOCUMENT TYPE: <.UT.&DTYP  "
    -"KEYWORD 1       : <.UT.&KEY1   <40 KEYWORD 2     : <.UT.&KEY2   "
    -"KEYWORD 3       : <.UT.&KEY3      "
    -" "
    -"PRESS ENTER TO PROCESS              "
    -"  PF03  TO QUIT                 "
    -"  PF04  TO SEE LIST OF ENVIRONMENTS"
    -"  PF05  TO SEE LIST OF INTERFACES "
    -"  PF06  TO SEE LIST OF PRODUCT TYPES"
    -"  PF09  TO SEE LIST OF KEYWORDS     "
4.  -IF &PFKEY EQ PF04 THEN GOTO EVIRN ;
    -IF &PFKEY EQ PF05 THEN GOTO INTER ;
    -IF &PFKEY EQ PF06 THEN GOTO PRODU ;
    -IF &PFKEY EQ PF09 THEN GOTO KWORD ;

5.  -SET &ALLKEY=&KEY1 | &KEY2 | &KEY3 ;
6.  -IF &ALLKEY EQ ' ' GOTO NODEFINE;
7.  DEFINE FILE DOCUMENT
8.  IFKEY/I1=IF
9.  -IF &KEY1 EQ ' ' GOTO SKIP1 ;
9a.  ( (KEY1  EQ '&KEY1') OR (KEY2 EQ '&KEY1')
9b.  OR (KEY3 EQ '&KEY1') ) AND
10. -SKIP1
11. -IF &KEY2 EQ ' ' GOTO SKIP2 ;
12.  ( (KEY1  EQ '&KEY2') OR (KEY2 EQ '&KEY2')
     OR (KEY3 EQ '&KEY2') ) AND
13. -SKIP2
14. -IF &KEY3 EQ ' ' GOTO ENDTHIS  ;
15.  ( (KEY1  EQ '&KEY3') OR (KEY2 EQ '&KEY3')
     OR (KEY3 EQ '&KEY3') ) AND
16. -ENDTHIS
17.    ('A' EQ 'A') THEN 1 ELSE 0;
18. END
19. -RUN
```

(continued)

Figure 4 (continued)

```
     -NODEFINE
     -CMS ERASE SAVE FOCTEMP A
     -CMS FILEDEF SAVE DISK SAVE FOCTEMP A (RECFM F LRECL 80
     -SET &NUMBER = '0000000'  ;
     -WRITE SAVE &NUMBER
     SET ALL=ON
20. TABLEF FILE DOCUMENT
21. HEADING
22. "SELECT THE DOCUMENT(S) BY MOVING THE CURSOR TO THE BEGINNING OF"
23. "THE WORD AND PRESSING PF6"
24. " "
25. PRINT TITLE
26. BY DOCNUM AS 'NUMBER'
27. IF DOCNUM FROM &DOCNUM1 TO &DOCNUM2
28. -IF &DTYP EQ ' ' GOTO ENVIR  ;
29. IF DTYP EQ &DTYP
30. -ENVIR
31. -IF &ECODE EQ ' ' GOTO FACE;
32. IF ECODE EQ &ECODE
33. -FACE
34. -IF &IFACE EQ ' ' GOTO PRO;
35. IF IFACE EQ &IFACE
36. -PRO
37. -IF &PROTYP EQ ' ' GOTO RELEASE   ;
38. IF PROTYP EQ &PROTYP
39. -RELEASE
40. -IF &FREL EQ ' ' GOTO CROSS;
41. IF FREL EQ &FREL
42. -CROSS
43. -IF &XMI EQ ' ' GOTO DATE;
44. IF XMI EQ &XMI
45. -DATE
46. -IF (&ADATE1 EQ ' ') AND (&ADATE2 EQ ' ') GOTO REFT;
47. IF ADATE FROM &ADATE1 TO &ADATE2
48. -REFT
49. -IF &ALLKEY EQ ' ' GOTO NOKEYT;
50. IF IFKEY EQ 1
51. -NOKEYT
52. END
53. -RUN
54. -CMS FILEDEF TEMP CLEAR
55. -CMS FILEDEF TEMP DISK TEMP FOCTEMP A (LRECL 80 RECFM F
56. -SET &TEMP = '0000000' ;
57. -LOOP
58. -READ SAVE &TEMP
59. -IF &IORETURN EQ 1 GOTO ENDLOOP ;
60. -WRITE TEMP &TEMP
61. -GOTO LOOP
62. -ENDLOOP
63. TABLE FILE DOCUMENT
64. PRINT CONTENT AS ''
65. BY DOCNUM NOPRINT
66. BY LINE# NOPRINT
67. ON DOCNUM PAGE-BREAK
68. ON DOCNUM SUBHEAD
69. "DOCUMENT TITLE   <TITLE  "
70. " "
71. "DOCUMENT NUMBER <DOCNUM        <40 DOCTYPE          <DTYP"
72. "ENVIRONMENT     <ECODE         <40 INTERFACE        <IFACE "
73. "PRODUCTTYPE     <PROTYP        <40 FOCUSRELEASE     <FREL "
74. "CROSS MACHINE   <XMI           <40 MEMO DATE (DMY)  <ADATE "
75. "KEYWORD 1 <KEY1                <40 KEYWORD 2        <KEY2 "
76. "KEYWORD 3 <KEY3  "
77. "                PRESS PF3 TO QUIT"
78. "  "
```

(continued)

Figure 4 (continued)

```
79.  IF DOCNUM IS (TEMP)
80.  END
81.  -RUN
82.  -GOTO START

83.  -EVIRN
     -CMS ERASE SAVE FOCTEMP A
     -CMS FILEDEF SAVE DISK SAVE FOCTEMP A (RECFM F LRECL 80
     -SET &ECODE = &CODE
     -WRITE SAVE &ECODE
84.  TABLE FILE ENVIRNMT
     PSELECT THE ENVIRONMENT BY MOVING THE CURSOR TO THE BEGINNING OF"
     "THE WORD AND PRESSING PF6"

     PRINT MACHCODE AS 'ENVIRONMENT' MACHDESC AS 'DESCRIPTION'
85.  END
86.  -RUN
     -READ SAVE &ECODE A6
     -GOTO START

     -INTER
     (Note: this code is functionally the same as that shown after
     the -EVIRN label and is not shown for the sake of brevity.)

     -GOTO START

     -PRODUCT
     (Note: this code is functionally the same as that shown after
     the -EVIRN label and is not shown for the sake of brevity.)

     -GOTO START

     -KWORD
     (Note: this code is functionally the same as that shown after
     the -EVIRN label and is not shown for the sake of brevity.)
     -GOTO START
```

```
&PROTYP = '    '
&DTYP   = ' '
&IFACE  = '    '
&FREL   = '    '
&XMI    = ' '
&ADATE1 = 010188
&ADATE2 = 310188
&TITLE  = '                    '
```

Let's look at the part of the code that will get processed after pressing the ENTER key. It corresponds to a DEFINE (lines 5-19), TABLEF (lines 20-54), and TABLE (lines 14-83).

First the DEFINE part will be discussed. The code is shown in Figure 4, lines 5 through 19. This DEFINE clause applies to the keywords assigned to the Dialogue Manager variables &KEY1, &KEY2 and &KEY3. &KEY1 might be equal to MFD fieldname values KEY1, KEY2 or KEY3; the same holds for &KEY2 and &KEY3. Therefore, we must determine which data entry field (i.e., &KEYi) corresponds to which data base field (i.e., KEYj). This is the purpose of the DEFINE. Let's see what happens to the FOCSTACK at execution time.

The first line checks to see if keywords were entered. If none were entered, the DEFINE part is skipped. In the example, MEMORY was entered in the &KEY1 field, so the DEFINE clause, starting at line 7, gets executed. Lines 7 and 8 are written to FOCSTACK as they do not start with "–". Lines 9a and 9b are written to the FOCSTACK as well, since the value of &KEY1 is MEMORY. The values for &KEY1, &KEY2 and &KEY3 are substituted before writing the line to the FOCSTACK. At this point, the FOCSTACK looks like this:

```
DEFINE FILE DOCUMENT
IFKEY/I1=IF  (  (KEY1  EQ  'MEMORY')
OR  (KEY2  EQ  'MEMORY')  OR  (KEY3  EQ
'MEMORY')  )  AND
```

Dialogue Manager continues processing at line 10. Since no &KEY2 value was entered, the condition `&KEY2 EQ ' '` holds and Dialogue Manager branches to the SKIP2 label in line 13. The FOCSTACK did not change during these steps. In line 14, Dialogue Manager is directed to the ENDTHIS label of line 16; again, the FOCSTACK did not change. Lines 17 and 18 are written to the FOCSTACK. Finally, the FOCSTACK created in this DEFINE clause by this process looks like this:

```
DEFINE FILE DOCUMENT
IFKEY/I1=IF ( (KEY1 EQ 'MEMORY') OR
(KEY2 EQ 'MEMORY') OR (KEY3 EQ
'MEMORY') ) AND ('A' EQ 'A') THEN 1
ELSE 0;
END
```

In line 19, Dialogue Manager encounters a –RUN and writes the special line to the FOCSTACK that will be interpreted by the FOCUS command processor as an execution order of the FOCSTACK. Finally, the IFKEY define field contains the following value:

```
IF ((KEY1 EQ 'MEMORY') OR (KEY2 EQ
'MEMORY') OR (KEY3 EQ 'MEMORY'))
AND ('A' EQ 'A') THEN 1 ELSE 0;
```

After the DEFINE, the TABLEF (lines 20 through 54) is executed. At the end of this TABLEF request, a list is shown containing the document number(s) and title(s) of all documents that correspond to screening information given in the form of Dialogue Manager variable values. From this, list the user can select documents to view by using HOTSCREEN's PF6 key; the values selected are then saved to the SAVE FOCTEMP file.

Lines 20 through 27 are written to the FOCSTACK.

Line 28 contains a Dialogue Manager command. Since &DTYPE was not referenced, Dialogue Manager branches to –ENVIR label. The &ECODE value is equal to 'PCMU', which means that the line IF ECODE EQ PCMU is written to the FOCSTACK. The values for &IFACE, &PROTYP, &FREL and &XMI are blank. Therefore, lines 35, 39, 41 and 44 are not written to the stack. In line 47, the line `IF ADATE FROM 010188 TO 310188` is written to the stack. Lines 50 and 52 are also written to the FOCSTACK. The –RUN in line 53 indicates the end of the Dialogue Manager session, and control is transferred to the FOCUS command processor.

This processor encounters the following FOCSTACK:

```
TABLEF FILE DOCUMENT
HEADING
"SELECT THE DOCUMENT(S) BY MOVING
THE CURSOR TO THE BEGINNING OF"
"THE WORD AND PRESSING PF6"
" "
PRINT TITLE
BY DOCNUM AS 'NUMBER'
IF DOCNUM FROM 0 to 9999999
IF ECODE EQ PCMU
IF ADATE FROM 010188 TO 310188
IF IFKEY EQ 1
END
```

This request lists all document titles and numbers that satisfy the selection criteria. HOTSCREEN's PF06 saving facility can be used in order to select the desired documents. After document selection, Dialogue Manager starts a new session at the point it passed control to the FOCUS command processor (line 54).

The next lines (54 through 62) in the FOCEXEC read the FOCTEMP file created during the document selection in HOTSCREEN. The TABLE request in lines 63 through 80 is entirely written to the FOCSTACK. The –RUN in line 81 passes control to the command processor that will, finally, show the document text related to the selected documents. Once control is passed back to Dialogue Manager, a new session is started. This session starts at line 82, where it encounters a –GOTO label that directs Dialogue Manager to the –START label at the beginning of the FOCEXEC.

Conference

In his article on Dialogue Manager Internals (March 1989 Systems Journal*), Gregory Dorman invited comments and suggestions on Dialogue Manager. This letter from Nick Browne begins a discussion we hope will continue.*

Dear *Systems Journal*:

I have been a user of FOCUS for over seven years. Greg Dorman's article on Dialogue Manager internals explained many of the features of the FOCUS Dialogue Manager (DM) language. I would like to put forward the following points in the discussion proposed by Mr Dorman.

1. Raising limits. In the context of the article, I assume Mr. Dorman is referring to raising the limit of the number of DM variables. This has often been raised by articles in the *Journal* but has always seemed unecessary; the following explains my reasoning.

- If the number of DM variables is increased, this will no doubt reduce the amount of memory available for other purposes and potential new features.

- I propose that any FOCUS system managing to use more than 350 or so variables is the result of poor program quality control and specification and a lack of standards.

Variables tend to be considered as consumable items within a system; that is, programmers will often use a different variable for a similar purpose in two or more programs. Examples of this are selection test values in TABLE requests; you will often find `IF PROD_CODE EQ '&PRD_CDE'` in one program, `IF PROD_CODE EQ '&PRDCODE'` in a second program, and `IF PROD_CODE EQ '&PROD_CODE'` in a third program. No doubt the system or program specification makes no mention of the naming of variables.

Variables in their simplest form are data items, as such they could be likened to fields within a Master File Description. Would anyone really create a file of 350 fields, many of which contain duplicate information and none of which are documented ? I would also propose that if so many variables are needed, Dialogue Manager is the wrong tool for the job; perhaps a MODIFY should be used.

The proliferation of variables within a systen can be reduced using the following rules :

- At the outset of programming, 10 'dummy' variables are set aside. These are used (and more importantly re-used) for miscellaneous purposes within FOCEXECs. Programs using these variables should make no assumptions about their existence, length or format (i.e., they should be explicitly assigned through a -SET).

- If a variable is used for selection tests, its name should be the same as the field. Suffixed variables can be used for multiple test values or a concatenated string built up from the dummy variables. For example:

```
-SET &PROD_CODE1 = 'LAAA001' ;
-SET &PROD_CODE2 = 'XBAB123' ;
IF PROD_CODE EQ '&PROD_CODE1'
OR '&PROD_CODE2'
```

or

```
-SET &PROD_CODE = &DUM1 | ' OR '
-                       | &DUM2 ;
IF PROD_CODE EQ '&PROD_CODE'
```

- One program should be called from another using -INCLUDE unless there is a good reason for using EX. The use of EX tends to increase the number of global variables needed.

- The use and naming of variables is documented as part of the system specification.

2. Database Access From Dialogue Manager. On the face of it, this would be a good idea. On reflection, though, I believe the opposite is needed, that is, the ability to assign variables within TABLE or MODIFY. Most systems will, at some point, use a TABLE or MODIFY followed by a -READ to assign data from the file to variables. I do not believe the addition of MODIFY updating facilities to Dialogue Manager justifies the coding effort, although this may seem simple. What about such issues as security, SU usage, SQL and other non-FOCUS file formats, and file definition?

Most database access requirements would be satisfied by implementing a Dialogue Manager version of the LOOKUP and FIND functions. I assume this would only be worth doing if there is a way of accessing an index without the need to build the internal tables required by MODIFY or TABLE and without the need to parse the Master File Description. An alternative strategy may be to add a new 'ON TABLE' option to TABLE, such as :

```
ON TABLE SAVEVAR [AS variable name]
```

Instead of a SAVE file being created, the first record would be assigned to a variable, perhaps with the variable identified through an AS phrase. If the record contained more than one field, individual variables could be easily be assigned by breaking up the new variable with the EDIT or SUBSTR functions. As an alternative to the syntax above, a new FORMAT option could be provided, such as:

```
ON TABLE SAVE FORMAT VARIABLE [AS
variable name]
```

A similar idea could be applied to MODIFY, perhaps as an extension of the 'TYPE ON ddname' command.

The above are simple extensions to the existing syntax and would probably cope with 95% of requirements. They avoid the need to use sequential files and the consequent FILEDEF or ALLOCate and -READ. They also use existing syntax so there would be no need to learn extra Dialogue Manager database access commands. As Dialogue Manager variables are stored in a predictable place and code already exists to set variables (for &LINES, &TRANS etc.), I presume these changes would not be too difficult to implement.

Perhaps another approach to the problem would be to add the 'FROM ddname' option to -READ, similar to 'FIXFROM FROM ddname', giving :

```
-READ FROM ddname
```

This would make the assignment of variables much easier, as there would be no need to worry about defining the length of variables. Taking this one stage further, how about :

`-READ FROM FOCSORT` coupled with `ON TABLE NOOUTPUT` or `TABLE FILE file NOOUTPUT`

The TABLE options would signal that FOCSORT should be created but not written to an extract file or formatted for the terminal. Another alternative would be the options :

```
ON TABLE NOMATRIX or
TABLE FILE file NOMATRIX
```

A TABLE of this form would simply set the values of &LINES and &RECORDS without attempting to create FOCSORT.

3. Total Increase in Dialogue Manager Syntax. Dialogue Manager is one of the most stable parts of FOCUS, and it would be shame to throw it away and start with something totally

new. On the other hand, Dialogue Manager can hardly be described as a 4GL in its current form. If anything should be done to the Dialogue Manager, it should be an increase in MODIFY features. In particular, PERFORM and REPEAT commands should be provided and the 'TYPE n' option given to -CRTFORM. Anyone who has used the PC/SCRIPT language will see the obvious benefits of a PERFORM command. This would allow a more structured, standardised and consistent approach to programming to be taken, which can only be good for system development and, just as importantly these days, the maintenance of systems.

4. Compiled Dialogue Manager. Is it worth the effort? Would compilation save machine resources or make FOCEXECs run faster? If so, why is the *Systems Journal* not full of articles describing methods to save CPU and I/O used by the Dialogue Manager? I do not believe a full compiled version is necessary; this would make system development and testing more time consuming. It would also lead to the inevitable problems that exist with some MODIFY programs, where there are slight differences between the COMPILEd and interpreted forms of the same FOCEXEC.

As I see it, the main problem with Dialogue Manager is the time it takes to branch to a label. Although Mr Dorman indicated that a table of labels is dynamically created, it often appears this table is lost. I presume this is because the table is only created for the currently executing FOCEXEC and overwrites the calling program's table. Slow branching usually manifests itself when returning from a –INCLUDEd FOCEXEC to the calling FOCEXEC (i.e., moving back up a menu level). This is where the called FOCEXEC has to branch to its end, followed by the calling FOCEXEC branching back up to a label before a -CRTFORM.

I would propose 2 alternatives to overcome this.

- A new Dialogue Manager command, –RETURN, should be implemented. This would work in the same way as –EXIT does, except that it would be used for -INCLUDEd FOCEXECs and would force an immediate return to the calling FOCEXEC.

- A preprocessor could be used to write a standard header to a FOCEXEC containing the line number of all labels within a FOCEXEC. When the FOCEXEC is executed, this header is used to build the table of labels. To reduce the amount of memory required, the table space each FOCEXEC occupies should be given up upon returning to the calling FOCEXEC. In theory, this strategy would only require at most five tables to be built, one for the EXecuted FOCEXEC and one for each level of -INCLUDE nesting (or should that be 10 if two levels of nested EX are used as well ?). As well as allowing faster branching, this would be a useful documentation feature. Obviously some way of coping with FOCEXECs that have been changed after preprocessing would be needed, something akin to the strategy used for out-of-date cross reference pointers on read-only files perhaps?

While I feel the Dialogue Manager has its limitations, it has largely been ignored as far as development is concerned. It would not take too many changes to enhance its usefulness and bring it into line with the features expected from a 4GL.

To finish, I would like to congratulate Mr Dorman on an excellent article; anything that explains the workings of FOCUS should be encouraged. This type of information can only be of benefit to the FOCUS community and Information Builders. As an aside, I wonder how many Dialogue Manager-related Hotlines can be closed now that a reasonable explanation has been made available!

Yours Sincerely,
Nick Browne
Director E.G.A. Ltd.
Basingstoke, Hampshire
England

About the Authors...

David Bland is a Vice-President at Pacific Software, Inc., which offers consulting services in New York and New Jersey. He may be reached at (201)792-5777.

Bernard J. Blick received his B.S. in physics from Rensselaer Polytechnic Institute in 1970. He has over eighteen years experience in data processing and has worked with FOCUS since 1981. Currently, he is an officer in the Technology Sector at The Bank of New York responsible for PC/FOCUS applications, developing and implementing new technologies, and supporting mainframe FOCUS product interfaces.

Lawrence M. Breindel is a long-time FOCUS user. His company, BSI, specializes in the delivery of networked medical software applications.

Patricia Davis is a Director of Central Solutions (U.K.) Ltd., a company specializing in consulting and FOCUS productivity tools. She has been in data processing for 25 years, 12 years with 4GLs and 8 years with FOCUS.

Eric Heilner is a Systems Analyst at the Federal Reserve Bank of NY. He has an MS degree in Computer Science and has been working with FOCUS for over 5 years. The views expressed are those of the author and do not necessarily reflect those of the Federal Reserve Bank of New York or the Federal Reserve System.

David Lovelace holds a B.S. in Physics. He has been in data processing for over 11 years and has worked with FOCUS for six years. Currently he is senior application support specialist with the FOCUS Products Group.

Frank Roberts is a FOCUS specialist in the Financial Systems Group of API (American Protection Industries), a large commercial alarm sales, service, and monitoring company. Frank has 20 years experience in data processing, with five of that in FOCUS.

Steven St. John has been a FOCUS programmer and consultant for the past three years. He is a consultant in the Information Technology Center of HCA Information Services, Inc., a subsidiary of HCA, the Health Care Company.

Joseph Szpiro is President of Szpiro Computers, Inc., a company specializing in consulting and educational services for advanced on-line systems. His background includes applications and systems programming, as well as information center experience. He has spent ten years with 4GLs and eight years with FOCUS.

Peter Van Der Lee is responsible for Technical Support to IBI's International Agents. Prior to his present position, he was a FOCUS consultant for IBI's French agent. He holds a Masters Degree in Computer Science from the University of Technology, Eindhoven, Netherlands.

Rodger Williams has worked for JCPenney Insurance since January 30, 1978 and has held several positions of increasing responsibility. He has a B.A. in English and has taken college courses in assembler language; his initial FOCUS training was completed May 22, 1985. In his current position with JCPenney, he is responsible for maintaining and designing personnel reporting systems for users in many locations throughout the United States.

BACK ISSUE INDEX

Volume 1, Number 1

SQL And FOCUS
Russell C. Lipton

Reportit! PC/FOCUS Lab Results Reporting
Lawrence M. Breindel

Benchmarking The Benchmark
Michael Howard

Maximizing FOCUS Efficiency
InThe VAX Environment
Melissa Webster
Sebastian Holst

Improving Simultaneous Processing
With Parallel Tables
Gerald D. Cohen

Tuning FOCUS Performance On MVS/TSO
Mark A. Simms

Interview: Dr. Ben Shneiderman

Book Review: Designing The User Interface

Tutorial: Introducing Knowledge-Based
Systems With LEVEL5
Cornelius Willis

Tutorial: FOCUS Windows
Jeffrey Kevelson

Techniques: Selecting Reports
With Hot Screen
Rick Zeimann

Techniques: Controlling Offline Output
Claude Goldman

Techniques: Tactics For Rounding
Numbers In PC/FOCUS
John Price

Volume 1, Number 2

Debugging FOCUS Code
Amanda Regan

Plexus File Structures Used In FOCUS
Gerald D. Cohen

CASE For FOCUS
Kenneth Newberger

Interview: Ken Orr

Exploring FOCUS Programming Styles
Ernie Ostic

Tutorial: CRTFORM Internals
Ed Wolfgram

Tutorial: LEVEL5 Reporting
Cornelius Willis

Reducing Paging In MODIFY
Mickey McIntire

Techniques: Using Alternate Database Keys
Joseph Szpiro

Techniques: Simulating Boolean Logic
For Report Screening
Robert Hoffman

Techniques: Hierarchical HOLD Files
Gerard Oberle

Techniques: Concatenating With Spaces
Joseph Yoklavich

BACK ISSUE INDEX

Volume 1, Number 3

Transaction Processing With COMMIT And ROLLBACK
Gerald D. Cohen

Measuring The Impact Of VMS Lock Management
Sebastian Holst

LEVEL5: New Opportunities For FOCUS Professionals
Russell C. Lipton

Extending The Entity-Relationship Model
Dr. Herbert J. Spencer

Interview: Dr. Peter Chen

Graphic Analysis Of Data With MODIFY
Mario Delgado

Scope Of The Testing Problem
Dr. Betty Mock

Technique: External Sorts From FOCUS
Art Greenhaus

Technique: Atypical Uses Of COMBINE To Update Key Fields
Jonathan Schulman

Volume 1, Number 4

SAA And FOCUS
Merv Adrian

PC/FOCUS And OS/2
Kevin Quinn
Discusses the changes from the DOS version of PC/FOCUS to the OS/2 version.

OS/2 And FOCUS
Bryce D. Segar
Includes specific information about the OS/2 operating system, as well as its meaning to the FOCUS user.

Interview: John Friedline

CUA - A New Toolkit
Merv Adrian

LEVEL5 Under Conditions Of Uncertainty
Don Barker

Relational Technology and SQL
Fabian Pascal

A FOCUS Program Generator For Custom Reports
Dan Hyman

Interactive Database Access Via Navigable Windows
Chris Gerrard

PC/FOCUS MultiUser: Hardware And Operating System Issues
John Hill

Power To The Printer
Terry Schwarz

Fine-Tuning The Unix Environment For SU
Reuven Schulman

CMS XEDIT From Dialogue Manager
Bill Myers

All At Once
Joseph Szpiro
A technique to correct all errors on a single CRTFORM screen with one message.

Remapping Keys In CMS
Larry Breindel

Preserving Core In PC/FOCUS
John Miller

Using LOAD MODIFY
Roland J. Baeslach
Robert G. Mathas

BACK ISSUE INDEX

Volume 1, Number 5

Bridging IMS And DB2 Via FOCUS
Bernard J. Blick
Marvin D. Schlanger

Entity-Relationship Modeling:
A Simplified Approach
Kurt Christoff

XWindows, DECWindows, and FOCUS
Merv Adrian

Interview: Bill Gates

Knowledge Systems With DBMS:
Tight Or Loose Coupling?
Peter H. Jones

The Impact Of FOCUS 5.5
Ian Bray

Tutorial: The Interactive
Chart Utility Interface
Vivian Perlmutter

Tutorial: The FOCUS To
Tell-A-Graf Interface
Michael R. Saeger

Generating Sorted Pie Charts
Michael R. Saeger

Improving Performance Using
Binary Files In PC/FOCUS
Robert G. Mathas
Roland J. Baeslach

Using DBA To Restrict Viewing In MODIFY
P. J. Matarese

FOCUS Control Of PROFS Directories
Label Klahr

Computing Medians In FOCUS
John Brady

Volume 1, Number 6

FOCUS/MSO: Many Users,
One Address Space
David Wolland

MVS Storage Management
Merv Adrian

Interview: Michael Stonebraker

Object Oriented Programming:
A New Paradigm
Ron Schultz

Tutorial: Letters To And From
Santa Claus With LEVEL5
Noreen Redden

Knowledge Engineering:
Building Expert Systems
George McKevitt

Constructing A LEVEL5 Knowledge
Base By Formalizing Arguments
Don Barker

Tutorial: A REBUILD Reference
Lynn Garst

Tuning A Large LAN Application:
A Case Study
John Hill

Technique: Tuning Of Existing
Extract Systems
Joseph Louis Yoklavich

Tutorial: Using Internal Formats
Joseph Szpiro

Technique: Relational Division In FOCUS
Edward Johnson

Technique: An Alternative To FOCUS
Global Variables
Michael Wacey

BACK ISSUE INDEX

Volume 2, Number 1

Application Generators For Managing Data With FOCUS
Amanda Regan

Technique: Rapid Screen Prototyping
Joseph Szpiro

Tutorial: Building FOCUS Window Applications
Pierre Castillo

Tutorial: Financial Reporting Language
Vivian Perlmutter

Technique: Sorting On Non-Key Fields In MODIFY
John Demetriades

Technique: Advanced Multi-Record Processing
John R. Moore

Interview: Karl Siler

Multi-Function Screen Processing
Hugh Lam

Automatic Generation Of Key Sequencers
Gregory Kiefer

Technique: An ISPF-Like TED Using FOCMAP
Michael R. Saeger

Technique: More Than Just An 'If' Test
Art Greenhaus

Technique: Using CURSOR To Highlight Invalid Entries
John Price

Volume 2, Number 2

Dialogue Manager Internals
Gregory Dorman

Technique: Reusing Space In The Scratch Pad Area
Chris Bell

Technique: Restructuring Databases To Optimize Retrieval
Williams Myers

Technique: Efficient Relational Files In PC/FOCUS
Roland J. Baeslach
Robert G. Mathas

Technique: Maintaining COMPILEd MODIFYs
Joseph Szpiro

Technique: Alternate Paths For MODIFY
James Whelehan

Technique: Passing Multiple Items From User-Written Subroutines
Patricia Franks

Automatically Starting And Stopping A VM Sink Machine
Geoffrey Allen Daniels

Interview: Raymond J. Noorda

Tutorial: Integrating LEVEL5 With FOCUS
Don Barker
Dan Cooper

OFFLINE To A PC FILE - Printing With LISTER
John Price

(continued)

BACK ISSUE INDEX

(Volume 2, Number 2 continued)

Technique: Pre-Validating TABLE Screening Criteria
Chris Le

Tutorial: Producing Multiple Report Lines From One Record
David Benbassat

Technique: If You Choose... Do Not MATCH FILE
Marta Campa Carvajal

Customizing FOCCALC
Ann Walker

Technique: Partial Key Retrieval From A Menu-Driven Application
Penny Clinch

Volume 2, Number 3

PC/FOCUS For OS/2 - Memory Access And Management
Kevin Quinn

The OSI Model
Merv Adrian

Network Design And Management
Daniel Collins

Fine-Tuning A PC/FOCUS-MultiUser Application
Susan Haran

The Evolution of PC/FOCUS Memory Management
David Cook

Tutorial: Report Painter
Chet Ensign

Understanding EQL
Laurie Boxer

Tutorial: FOCUS Vision
Terry Cosentino

File Lists From Within FOCUS
David Lovelace

CRTFORM Internals: Recomputed Fields and ACTIVATE/DEACTIVATE
Peter J. Natali

For those of you who do not have a complete set of the FOCUS Systems Journal, back issues are available at $12 an issue, three issues for $33, and five for $50. All issues are available with the exception of Volume One, Number One (January 1988).

If you are interested in purchasing back issues, you may either send in a check, along with a listing of the issues you would like, to: J.A.F. Box 1450, New York, NY 10117. Or call Paul Lin at 212-736-4433, ext. 3553, with your order and reference your American Express, VISA, or Master Card number.

Fold And Seal Before Mailing

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

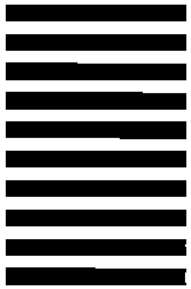
BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 00961 NEW YORK, NY

POSTAGE WILL BE PAID BY ADDRESSEE

FOCUS SYSTEMS JOURNAL

J.A.F BOX 1450
New York, N.Y. 10117



Fold And Seal Before Mailing

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 00961 NEW YORK, NY

POSTAGE WILL BE PAID BY ADDRESSEE

FOCUS SYSTEMS JOURNAL

J.A.F BOX 1450
New York, N.Y. 10117

Fold And Seal Before Mailing

To continue improving the Journal, we need to know the subjects that are of most interest to you. Please take a minute to check off one or more boxes below and to add any additional comments.

I'd like to see the following kinds of articles in the *Journal*:

- ☐ Thought pieces on leading-edge subjects for programming
- ☐ FOCUS performance optimization
- ☐ Expert system development with LEVEL5
- ☐ FOCUS architecture
- ☐ Relational database and database theory
- ☐ Emphasis on PC-FOCUS
- ☐ Porting FOCUS applications between machines
- ☐ FOCNET and connectivity
- ☐ Tips and Techniques

Comments:

FOCUS SYSTEMS JOURNAL

Please sign me up for a one year subscription (6 issues) at $60.00

Mr./Mrs./Ms._____ Title_____

Company_____ Department_____

Address_____

City_____ State____ Zip_____ Telephone_____

I currently run FOCUS on my: IBM Mainframe___ VAX___ WANG___ UNIX___ IBM-PC___

☐ Check enclosed

Charge my ☐ American Express ☐ MasterCard ☐ Visa

Account #_____ Exp Date_____

Signature_____

To Order Your Subscription By Telephone, Please Call (212) 736-4433 ext. 4409

Make checks payable to Information Builders, Inc.

Fold And Seal Before Mailing

To continue improving the Journal, we need to know the subjects that are of most interest to you. Please take a minute to check off one or more boxes below and to add any additional comments.

I'd like to see the following kinds of articles in the *Journal*:

- ☐ Thought pieces on leading-edge subjects for programming
- ☐ FOCUS performance optimization
- ☐ Expert system development with LEVEL5
- ☐ FOCUS architecture
- ☐ Relational database and database theory
- ☐ Emphasis on PC-FOCUS
- ☐ Porting FOCUS applications between machines
- ☐ FOCNET and connectivity
- ☐ Tips and Techniques

Comments:

FOCUS SYSTEMS JOURNAL

Please sign me up for a one year subscription (6 issues) at $60.00

Mr./Mrs./Ms._____ Title_____

Company_____ Department_____

Address_____

City_____ State____ Zip_____ Telephone_____

I currently run FOCUS on my: IBM Mainframe___ VAX___ WANG___ UNIX___ IBM-PC___

☐ Check enclosed

Charge my ☐ American Express ☐ MasterCard ☐ Visa

Account #_____ Exp Date_____

Signature_____

To Order Your Subscription By Telephone, Please Call (212) 736-4433 ext. 4409

Make checks payable to Information Builders, Inc.

FOCUS SYSTEMS JOURNAL

July 1989 Volume 2 Number 4

1 *From The Editor-In-Chief...*

2 *Performance Tuning With The FOCUS to DB2-SQL/DS Interface*
David Bland

12 *Measuring Code Efficiency*
Bernard J. Blick

21 *Controlling FOCUS Read Loops and Restarts*
Patricia Davis

27 *Using Session Monitor To Store Variables For Batch Processing*
Steven St. John

31 *An Automated Method For Verifying FOCUS Database Integrity*
Eric Heilner

36 *Using SORTHOLD With CRTFORM*
Joseph Szpiro
Frank Roberts

38 *Interview: Dave Kaplan*

46 *Analyzing Master File Descriptions*
Rodger Williams

50 *Isolating Operating System Commands - OSFEX*
Lawrence Breindel

58 *Logical File Names*
David Lovelace

64 *Using Dialogue Manager*
Peter Van Der Lee

76 *Conference*

79 *About The Authors...*

80 *Back Issue Index*

Information Builders, Inc.
J.A.F. Box 1450
New York, N.Y. 10117

Bulk Rate
U.S. Postage
PAID
New Haven, CT
Permit No. 130