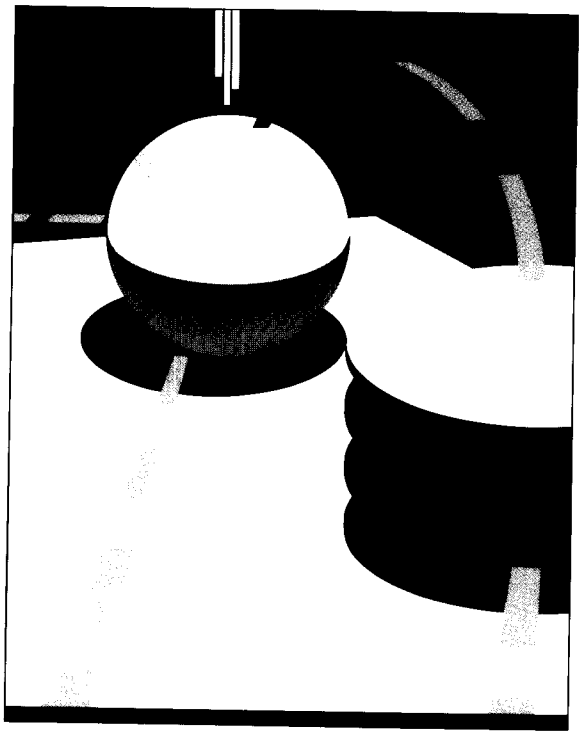
T A N D E M

SYSTEMS REVIEW

VOLUME 2, NUMBER 1

FEBRUARY 1986



Credit-authorization Benchmark for High Performance and Linear Growth

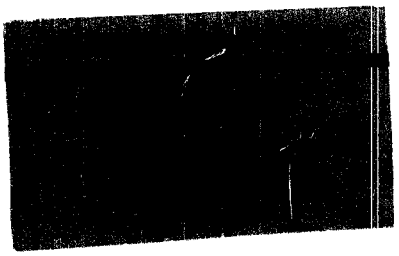
Buffering for Better Performance

DP2 Conversion ▪ TACL ▪ COBOL85

Managing System Time

New Products ▪ Manual Subscriptions

Index



Volume 2, Number 1, February 1986

Editor
Carolyn Turnbull White

Technical Advisor
Dick Thomas

Associate Editors
Kent Madsen
Ellen Marielle-Tréhoüart

Assistant Editor
Sarah Rood

Art Director
Carol Schaffer

Production and Layout
Laurie Menden
David Thompson

Cover Art
Stephen Stavast

Typesetting
Tandem Typesetting Group

The *Tandem Systems Review* is published by Tandem Computers Incorporated.

Purpose: The *Tandem Systems Review* publishes technical information about Tandem software releases and products. Its purpose is to help programmer-analysts who use our computer systems to plan for, install, use, and tune Tandem products.

Subscription additions and changes: Subscriptions are free. To add names or make corrections to the distribution data base, requests within the U.S. should be sent to Tandem Computers Incorporated, Sales Administration, 19191 Vallco Parkway, MS 4-05, Cupertino, CA 95014. *Requests outside the U.S. should be sent to the local Tandem sales office.*

Comments: The editor welcomes suggestions for content and format. Please send them to the *Tandem Systems Review*, 1309 So. Mary Avenue, Sunnyvale, CA 94087.

Copyright © 1986 by Tandem Computers Incorporated. All rights reserved.

No part of this document may be reproduced in any form, including photocopying or translation to another language, without the prior written consent of Tandem Computers Incorporated.

The following are trademarks or servicemarks of Tandem Computers Incorporated: BINDER, CROSSREF, DYNAMITE, ENABLE, ENCOMPASS, ENCORE, ENSCRIBE, EXPAND, FAXLINK, FOX, GUARDIAN, GUARDIAN 90, GUARDIAN 90XF, INSPECT, NonStop II, NonStop TXP, PCFORMAT, PS MAIL, PS TEXT EDIT, TACL, TAL, Tandem, TMF, TRANSFER, XRAY.

IBM and IBM PC are trademarks of International Business Machines Corporation. UNIX is a trademark of AT&T Bell Laboratories.

Credit-authorization Benchmark for High Performance and Linear Growth

In benchmark tests conducted for a major U.S. retailer interested in building a nationwide credit-authorization system, Tandem™ NonStop TXP™ systems demonstrated a linear increase in processing power as additional processor modules were added. In the tests, an 8-processor Tandem system processed twice as many transactions per second as a 4-processor system, and a 32-processor system processed twice as many as a 16-processor system.

The benchmark tests also demonstrated the high performance of Tandem NonStop TXP systems: on the 32-processor system, 149 transactions were processed per second, with a CPU utilization of 80.6%. Response time was less than two seconds for at least 90% of the transactions.

This article discusses the importance of linear growth in processing power and then describes the retailer's proposed credit-authorization application, the hardware and software configurations used in the tests, and the results of the performance measurements.

Importance of Linear Growth in Processing Power

The results indicating the linear growth in processing power are significant. They mean that users can expand their NonStop TXP systems to meet growing transaction-processing needs without incurring the nonlinear increase in system costs encountered when most other computer systems are expanded. Also, Tandem systems are expandable in small steps so that the amount of processing power available need never greatly exceed that required.

The expansion of the test system stopped at 32 processors, as that was the total needed to satisfy the retailer's requirements. There is no indication that the linear behavior of the NonStop TXP system stops there, however. It conceivably extends to 224 NonStop TXP processors, the maximum number that can be linked by FOX™, Tandem's fiber optic extension.

Project Overview

The retailer, interested in meeting stringent requirements for its credit-authorization system, asked Tandem to run simulations of the application on Tandem hardware. The key requirements were a fast response time and a high transaction volume.

Specifically, the retailer proposed to create a network consisting of three host systems and to divide the national transaction volume among them. This credit-authorization network would receive transactions from the retailer's existing SNA environment through an IBM 3705/25 Communications Controller. Initially, the peak transaction volume for each host system would be 60 transactions per second (tps), with a potential for growth to 120 tps. The retailer required a response time under two seconds for at least 90% of the transactions.

Two benchmark evaluations were conducted to validate the system design. The first, referred to as B1, was conducted at the Tandem Performance Center in Sunnyvale, California. The second, a retailer-developed volume test referred to as B2, was conducted at the Tandem manufacturing facility in Santa Clara, California.

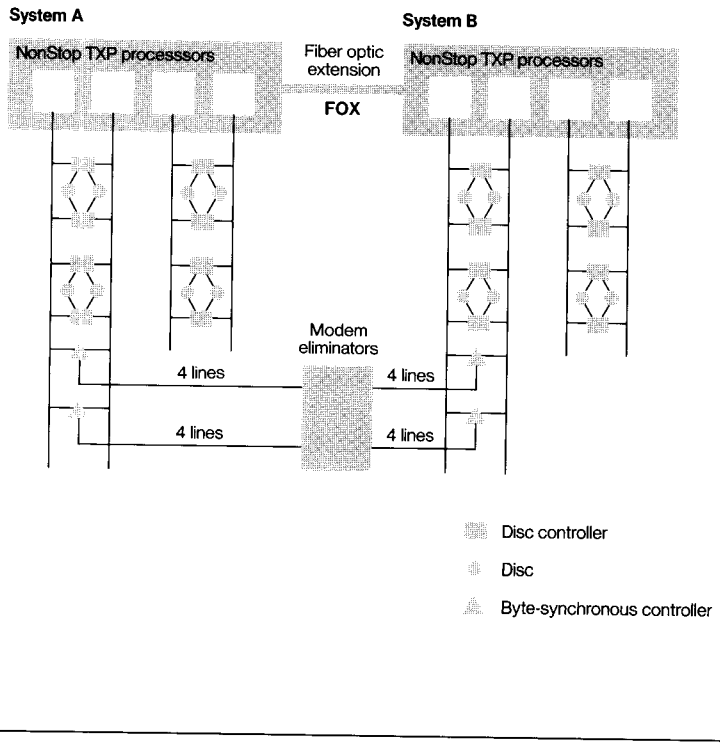
The Benchmarks

Benchmark B1

In B1, a standard benchmarking application was modified by Tandem to simulate the SNA interface and data-base I/O requirements for the proposed credit-authorization system. This application was first run on a 4-processor NonStop TXP system and then on an 8-processor NonStop TXP system.

Figure 1 illustrates the hardware configuration used in the 8-processor B1 tests. Two systems (A and B), each consisting of 4 NonStop TXP processors, were connected with FOX. The two systems were identically configured, with the exception that some application files were partitioned across disc volumes resident in both systems. The credit-authorization files and negative files for other national credit cards were divided into eight partitions, based on the primary key. Four of the eight partitions resided on system A and four on B. The byte-synchronous lines and modem eliminators connecting the two machines were used to simulate the processing of credit-authorization requests for national credit cards.

Figure 1



In the 4-processor tests, only system A was used, and the files were partitioned over the four disc volumes resident on that system. Credit-authorization requests for national credit cards were simulated by programs within system A.

Benchmark B2

In B2, the retailer provided an application similar to that used in B1. This application was stress-tested on 16- and 32-processor NonStop TXP systems.

Figure 1.
Hardware configuration used in the 8-processor B1 tests. In the 4-processor tests, only system A was used. (For simplicity, the asynchronous controllers and terminals, a printer controller and printer, and tape controllers and tape drives are not shown.)

Figure 2.

Hardware configuration used in the 32-processor B2 tests. (a) An overview. In the 16-processor tests, the same configuration was used, with half the number of CPUs and SNA lines. (b) The distribution of disc drives, byte-synchronous controllers, and bit-synchronous controllers in the 16-processor systems. (For simplicity, asynchronous controllers and terminals, a printer controller and printer, and tape controllers and tape drives are not shown.)

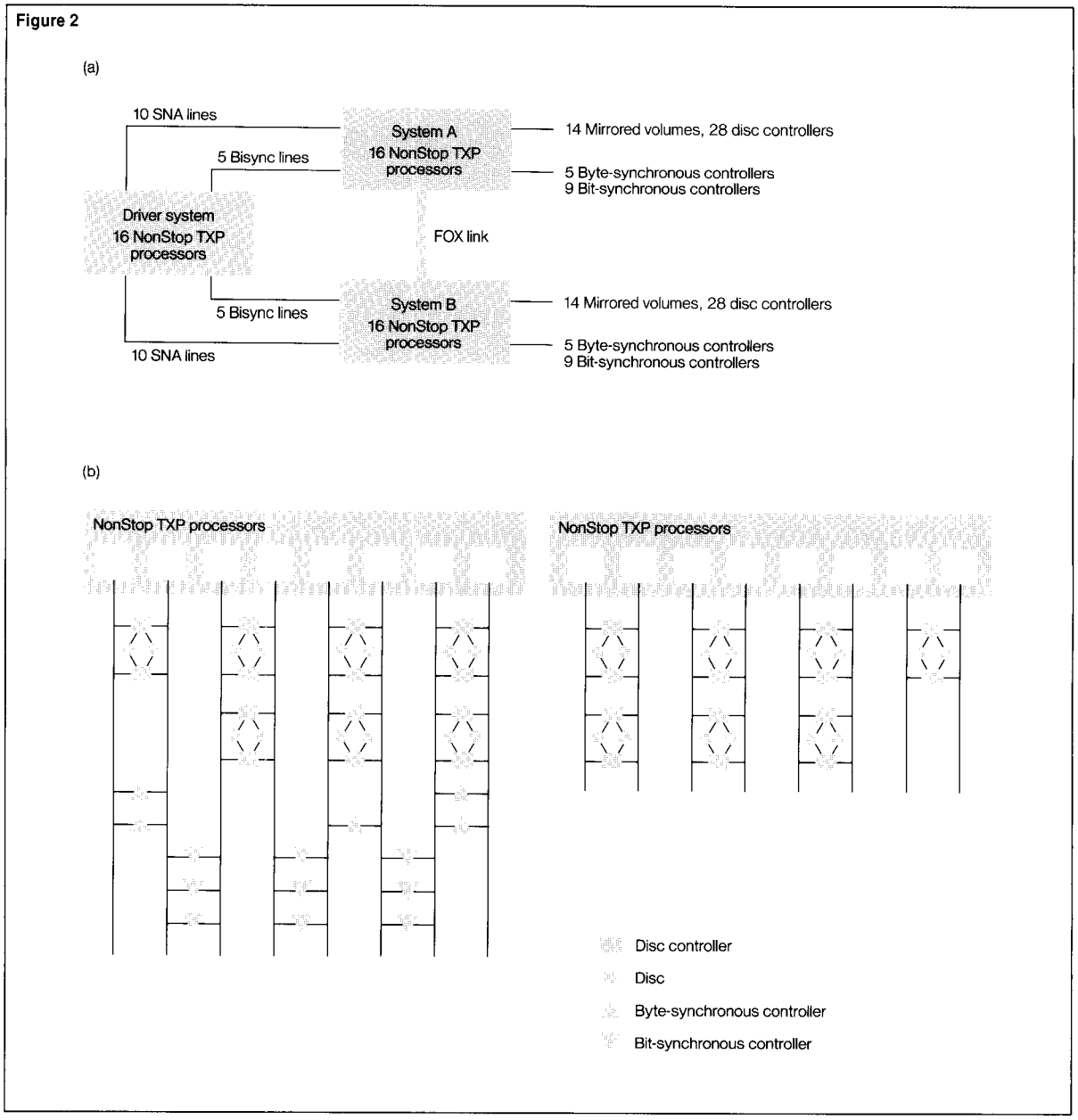


Figure 2 provides a high-level overview of the hardware configuration used in the 32-processor B2 test. This time, FOX was used to connect two fully configured, 16-processor NonStop TXP systems. The credit-authorization and bankcard negative files were again partitioned across both nodes; this time ten partitions were used (five per system).

In the 16-processor tests, the same configuration was used except that two 8-processor systems were linked together instead of two 16-processor systems. Again, the files were partitioned across both nodes. There were ten partitions (five per system).

It was the goal of B2 to provide as realistic a test environment as possible. Thus, a driver system consisting of 16 NonStop TXP processors was used in both the 16- and 32-processor tests. The driver system sent transactions to the benchmark nodes using 20 lines driven by SNAX, Tandem's standard SNA interface (ten lines were used in the 16-processor tests).

SNAX was used because, as explained above, transactions would come to the proposed Tandem credit-authorization system from the retailer's existing SNA environment through an IBM 3705/25 Communications Controller.

In addition to the SNA lines (which used the 3650 protocol), ten bisynchronous lines were evenly distributed between the two nodes in both the 16- and 32-processor tests. These lines were used to simulate the transmission and servicing of transactions to national bankcard-authorization centers.

Application Overview

As explained above, two separate application designs were used in the evaluation. The first (B1) was an application simulation developed by Tandem, and the second (B2) was a more realistic customer-written simulation.

The design used in B1 is summarized in Figure 3. As shown, both the terminal simulators and an SNA 3650 simulator ran in the same system as the application. The major differences in B2 were that (a) a separate driver system and a real SNA interface replaced the B1 terminal simulator and SNA 3650 simulator shown in Figure 3, and (b) the servers in B2 were provided by the customer. Except for these differences, the structure of the B2 application was the same as that shown in Figure 3.

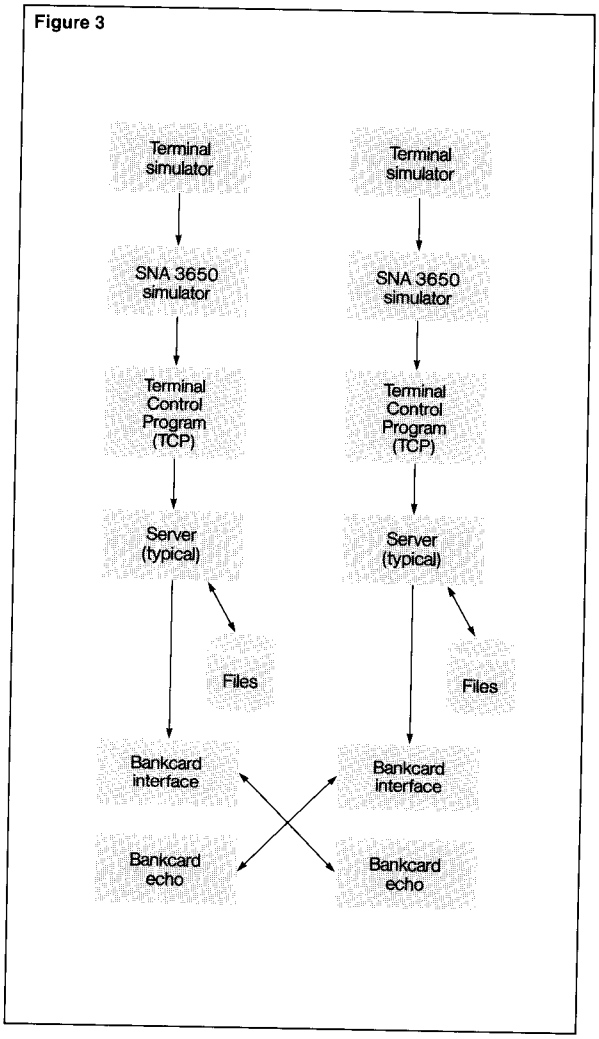


Figure 3.
Structure of the application run in benchmark B1. The B2 application had essentially the same structure, except that a driver system and a real SNA interface were used in place of the terminal and SNA 3650 simulators.

Application Components

SNA 3650 Simulation. The B1 terminal simulators (resident in the same node as the application software) and the B2 terminal simulators (resident in a separate driver system) both generated transactions containing random data at specified time intervals and captured response-time statistics. The B1 simulators accounted for expected SNA communications-software overhead by consuming CPU cycles, however, while the B2 terminal simulators used SNAX.

Terminal Control Process (TCP). The Terminal Control Process (TCP) is a multithreaded process supplied by Tandem to control multiple terminals and terminal types. In both B1 and B2, one TCP resided in each CPU at each node. Application programs executed by the TCP were written in Screen COBOL.

Table 1.
The transaction mix used in the B1 and B2 benchmarks.

Transaction	Percentage of mix	I/O requirements
Read-only of customer's credit record	1%	1 terminal read 5 table lookups 1 read of Auth file 1 write to Log file 4 interprocess I/Os 1 reply to terminal
Processing for charge-authorization and update	73%	1 terminal read 5 table lookups 1 read of Auth file 1 update of Auth file 1 write to Log file 4 interprocess I/Os 1 reply to terminal
Out-of-area authorization (2-second delay)	1%	1 terminal read 2 table lookups 1 read of Auth file 1 read of Index file 1 write to pipeline 7 interprocess I/Os 1 reply to terminal
Out-of-area authorization (3-second delay)	1%	(Same as above)
Out-of-area authorization request (4-second delay)	1%	(Same as above)
Bankcard authorization (1-second delay)	4%	1 terminal read 3 table lookups 1 write to bankcard line 1 read of bankcard line 1 write to bankcard log 5 interprocess I/Os 1 reply to terminal
Bankcard authorization (2-second delay)	3%	(Same as above)
Bankcard authorization (3-second delay)	3%	(Same as above)
Customer file inquiry	1%	1 terminal read 4 table lookups 1 read of Auth file 1 write to Log file 3 interprocess I/Os 1 reply to terminal
Customer file inquiry and update	2%	1 terminal read 4 table lookups 1 read of Auth file 1 update of Auth file 1 write to Log file 3 interprocess I/Os 1 reply to terminal
Authorization from remote system	2%	1 pipeline read 1 table lookup 1 read of Auth file 1 update of Auth file 1 write to Log file 3 interprocess I/Os 1 reply to pipeline
Authorization from catalog process/system	8%	1 terminal read 1 table lookup 1 write to Catalog file 1 interprocess I/O 1 reply to terminal

Servers. The functions listed below are representative of typical servers used in the tests (although not all servers performed every function listed):

- Edit and reformat incoming transactions.
- Make yes/no decisions for local requests.
- Determine the need for remote or bankcard authorizations.
- Perform fallback processing.
- Log transactions.
- Format responses.
- Simulate going to a remote ADC for authorization.
- Format messages to the bankcard interface.
- Log remote authorizations.
- Log catalog requests for later processing.
- Log operational (as opposed to application) exception conditions.
- Provide table-lookup services to other servers.

Bankcard Interface. In both B1 and B2, the bankcard interface provided the multithreaded interface to various bankcard-authorization networks. It isolated application servers from communications-protocol concerns.

Bankcard Echo. In both B1 and B2, the bankcard echo simulated a bankcard-authorization network. It imposed response-time delays based on transaction type and resided in a separate node (or, in the 4-processor tests, in a different CPU) from the node containing the bankcard interface.

The Transaction Mix

The transaction mix used in our tests is shown in Table 1. It reproduced the retailer's requirements as closely as possible, incorporating a specified percentage of each type of transaction that the production system would be required to process.

As shown in Table 1, all of the transactions used table lookups. These tables were loaded into extended data segments within each processor's memory. The application then calculated which table it should reference to read the necessary information. (The use of in-memory tables is a high-performance design alternative to storing data tables on disc; when this technique is used, table access can be at memory speed rather than at I/O speed.)

Benchmark Results

One way of comparing the capacities of multiple-processor computer systems of various sizes is to measure the average CPU utilization at different transaction rates. Given transaction rates and corresponding CPU utilization averages for a system with *n* processors, the system's performance behavior is considered linear if, with twice as many processors and I/O peripherals, it can handle twice as many transactions per second at the same level of CPU utilization. The results of B1 and B2 show that Tandem systems behave in this way.

Figure 4 and Table 2 summarize the performance of the systems in B1. The data shows that, at given levels of CPU utilization, the 8-processor system was consistently able to handle twice as many transactions per second as the 4-processor system.

Table 2.
B1 transaction rate versus CPU utilization (NonStop TXP processors).

Transactions per second	CPU utilization (%)
4 processors	
19.0	93.7
14.6	72.7
10.1	50.0
7.3	36.3
5.7	28.3
4.7	23.4
8 processors	
38.0	93.6
29.2	72.5
20.2	50.0
14.7	36.2
11.5	28.3
9.4	23.3

Figure 4

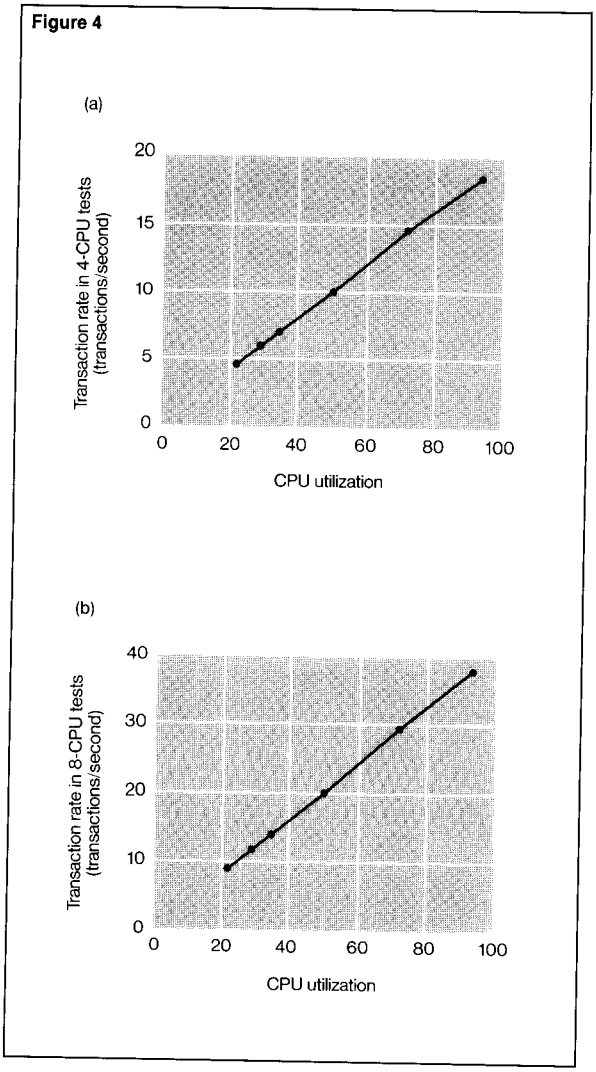


Figure 4.
Summary of performance data obtained in benchmark B1 for (a) the 4-processor system and (b) the 8-processor system. The 8-processor system consistently handled twice as many transactions per second as the 4-processor system at identical levels of CPU utilization.

Figure 5.
Summary of performance data obtained in benchmark B2 for (a) the 16-processor system and (b) the 32-processor system. The 32-processor system consistently handled twice as many transactions per second as the 16-processor system at identical levels of CPU utilization.

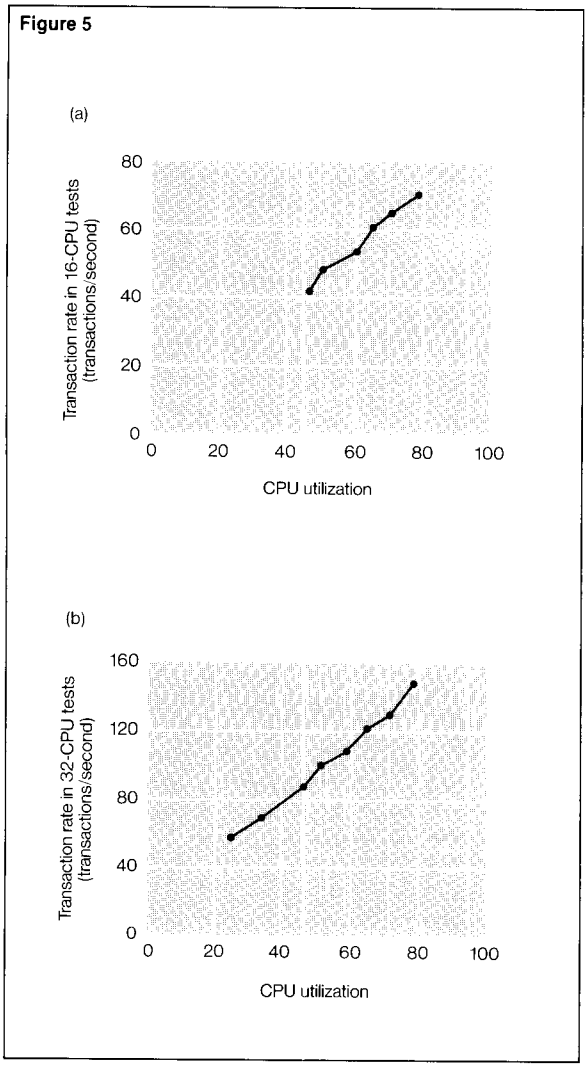


Figure 5 and Table 3 summarize the performance of the systems in B2. The data shows that, at given levels of CPU utilization, the 32-processor system was able to handle twice as many transactions per second as the 16-processor system. In the 32-processor tests, the system easily exceeded the benchmark goals. Since 149 tps was obtained with the first benchmark test, it was not necessary to fine-tune the application to meet the goal of 120 tps.

Table 3.
B2 transaction rate versus CPU utilization (NonStop TXP processors).

Transactions per second	CPU utilization (%)
16 processors	
72.0	80.2
66.0	73.7
61.0	68.2
55.0	61.1
49.0	54.4
43.0	48.9
32 processors	
149.0	80.6
124.0	67.4
110.0	59.6
90.0	49.0
70.0	34.3
57.0	24.0

Acknowledgments
Many Tandem employees contributed their time and energy to the success of the B1 and B2 performance studies. The authors would like to recognize the staff and management of the New York Uptown Branch and South Central District, who contributed the bulk of the software to the benchmarks; Jolly Young, Stephen Dudley, and John Haverland for managing the benchmarks; Richard Vnuk of the Large Systems Support Performance Group for his assistance in tuning the PATHWAY system; Gary Hugo for his contribution of the Benchmark Monitor (BMON); and the entire staff and management of the Santa Clara manufacturing facility for their assistance throughout the benchmarks.

Tony Chmiel is a senior staff analyst in the Tandem Performance Center and has been with Tandem since January of 1984. Tony has 11 years experience in data processing, with more than half of that on Tandem NonStop systems.

Tom Houy has been with Tandem for over five years and is currently a performance advisor for the Tandem Performance Center. Before this, Tom worked on another major mainframe developing performance measurement systems and performance enhancements for the operating system.

Buffering for Better Application Performance

Sequential block buffering and buffered cache are GUARDIAN 90™ File System options that can significantly improve the performance of online and batch applications that process structured files sequentially. In addition to improving the performance of specific applications, they reduce the per-transaction utilization of CPU and disc resources, thus indirectly improving the performance of all other applications that share those resources.

In this article, the following topics are discussed:

- The implementation of sequential block buffering.
- The implentation of buffered cache.
- The B00 COBOL enhancements that make both features easy to use.
- The use of sequential block buffering in a read-only environment.
- Concurrency issues relating to sequential block buffering.
- Considerations for using sequential block buffering and/or buffered cache when records are updated.

Sequential block buffering is available with both GUARDIAN™ and GUARDIAN 90 operating systems, both Disc Process 1 (DP1) and Disc Process 2 (DP2), and a variety of programming languages. Some performance and implementation details vary from one Tandem hardware and software environment to another. In B00 DP1, buffered cache is available for TMF™ audited files only; in DP2 it is available for all files.

To simplify the discussion, this article assumes a processing environment composed of NonStop TXP processors, the B00 GUARDIAN 90 operating system, B00 DP2, and B00 COBOL.[1] It presents a detailed view of the use of sequential block buffering and buffered cache in this environment. Tandem systems analysts can help users to apply the information to other processing environments.

How Sequential Block Buffering Works

ENSCRIBE™ structured files are a set of data records. To provide an efficient means of moving the records between disc and memory, the records are grouped into fixed-length data structures called blocks. Blocks can be as large as 4096 bytes and can hold as many records as space permits (minus room for control information).

[1]Tandem will release a new COBOL compiler and run-time library in the first calendar quarter of 1986. COBOL85, described in the accompanying article, "Tandem's New COBOL85," will be based on the new ANSI COBOL 1985 standard. Its use in the sequential block buffering and buffered cache methods described in this article will be identical with those of B00 COBOL.

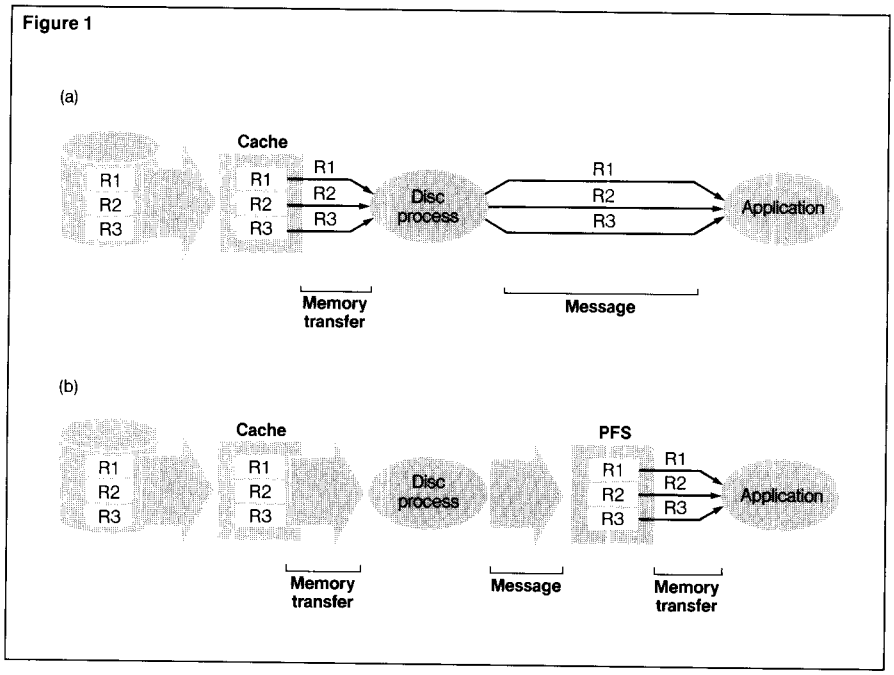


Figure 1.

Data access (a) without sequential block buffering and (b) with it. This File System option allows a process to read a structured file one block at a time instead of one record at a time, while retaining the convenience of automatic record deblocking. Since there are often many records in a structured block, reading a file with sequential block buffering reduces the number of requests for service that must be sent to the disc process.

Sequential block buffering is a File System option that allows a process to read a structured file one block at a time instead of one record at a time, while retaining the convenience of automatic record deblocking (see Figure 1). Since a structured block often has many records (the ratio of records to blocks is known as the blocking factor), reading a file with sequential block buffering reduces the number of requests for service that must be sent to the disc process.

While many disc-process requests for sequential reads are likely to be satisfied from cache, the requests still must enter a queue for disc-process services. The more applications there are contending for disc service, the greater the opportunity for queuing and the greater the chance of a cache "miss."

Also, each time the process sends a request to the disc process, it incurs the overhead associated with an interprocess message and enters an I/O wait state. This means that it must give up the CPU to any other processes waiting for it. When the I/O completes, the process must enter the ready list and wait for the CPU to become available to resume execution.

When sequential block buffering is in effect, the File System requests service from the disc process only when a new block is needed, not for every record logically read by the program. As shown in Figure 1, the File System maintains a buffer in the process file segment (PFS), a private data area established for every process. When a running program issues a sequential read request, the File System satisfies the request by deblocking the next record from the buffer and moving the information into the data area of the process. Thus, no messages have to be sent to the disc process, and the requester does not have to wait for a reply.

If update operations are performed on a file opened for sequential block buffering, the update does not simply change the record in the program's private sequential block buffer. The update is also passed directly to the disc process.

The buffer is discarded when an update occurs, so the next sequential read results in a message to the disc process, and possibly a physical I/O to retrieve the next block. The performance is no worse than when a program reads without sequential block buffering, but no benefit is received either.

With the exception of some concurrency issues (discussed later), sequential block buffering is transparent to programmers.

If a program uses sequential block buffering to open a file with alternate keys, the alternate-key file is also opened with sequential block buffering. When a file is accessed sequentially by alternate key, the alternate-key file can be read sequentially. The primary file must then be read by key, however, a nonsequential access. Overall performance is thus improved, but the disc process must still be involved with every request to read a new record. The same file can also be accessed sequentially by primary key and receive the normal benefits of sequential block buffering.

How Buffered Cache Works

The B00 software release offers a new File System and disc-process feature known as buffered cache, which must not be confused with sequential block buffering. Buffered cache allows data-base updates to be written to cache without immediately being written to disc. This is a significant performance advantage for an application that writes sequential data. Instead of writing each record to disc separately, the application can build blocks of records in cache without having to do any physical I/O until after the block is complete. The total number of physical I/Os is thus reduced by a factor that approaches the blocking factor of the file. Of course, this assumes that enough cache is available to let the block stay in cache without disturbance until the block is finished and that records are written in the sequence in which they are organized.

Updated cache blocks are written when the disc process goes idle, when they are forced out by a least recently used algorithm, or when periodic (every five minutes) control points are processed by the disc process. The longer a data block stays in cache, the more opportunity there is to update it multiple times in memory and post all of the updates with a single write to the disc. The performance benefit of buffered cache comes from the application's ability to write to buffered cache without waiting for the mechanical delay of the disc drive, and from the batching of multiple updates (cache-write hits).

If a file is audited by TMF, buffered cache is automatically used for all updates to the file. TMF ensures file consistency by using audit trails to back out aborted transactions or to recover inconsistent files. The File System permits applications to request buffered cache for DP2 unaudited files as well, however.

Application designers must carefully consider the use of buffered cache for unaudited files because, if it is used, a CPU failure that causes the loss of a primary disc process is likely to result in loss of the updates made to the file, if the sync depth is zero. Loss of buffered cache data can occur if a volume is brought down incorrectly, or if a double failure causes loss of the disc-process pair. If this happens, the File System returns Error 122, FEDATALOSS, to the application. Operations procedures should be established or the application should be written to implement a "restore and rerun" type of recovery when this error is encountered.

Sequential block buffering and buffered cache are separate and independent functions. Sequential block buffering is designed to improve read performance, while buffered cache is intended to improve write performance.

B00 COBOL Enhancements

Sequential block buffering has been supported by the File System for many years, but not by COBOL. Some programmers have called File System procedures directly from COBOL to take advantage of sequential block buffering. Although this works, the method is somewhat cumbersome. The B00 version of COBOL has been enhanced to fully support the feature.

Sequential block buffering is now selected through the RESERVE *n* AREAS clause in the FILE-CONTROL entry. When *n* is greater than 1 and the open mode is INPUT or I-O, sequential block buffering is selected. The number specified as *n* does not vary the number or size of buffers. A number greater than 1 simply selects the feature. The file opened must be a structured disc file, and the access mode must be SEQUENTIAL. Organization can be SEQUENTIAL, INDEXED, or RELATIVE, however. If for any reason sequential block buffering cannot be invoked, normal I/O is used, and no diagnostic is issued.

RESERVE *n* AREAS is also used to select the buffered-cache feature, so programmers should take care to select the correct open mode. Sequential block buffering is selected when the open mode is INPUT or I-O; buffered cache is selected when the open mode is I-O or OUTPUT. Thus, buffered cache and sequential block buffering are both selected when the open mode is I-O. The only way to read a file with sequential block buffering and update it without the risk involved with unaudited buffered cache is to use two separate file definitions (FDs), one open for INPUT with RESERVE 2 AREAS and another open for I-O with no RESERVE *n* AREAS clause.

The same COBOL verbs, READ and START, used for normal sequential I/O are used for sequential block buffering. The fact that sequential block buffering or buffered cache is turned on is transparent to programmers.

The following program is a simple example of the new COBOL implementation:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SBB-EXAMPLE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. TANDEM/16.
OBJECT-COMPUTER. TANDEM/16.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT TEST-SBB ASSIGN TO
          TESTFILE
          ORGANIZATION IS SEQUENTIAL
          ACCESS MODE IS SEQUENTIAL
          RESERVE 2 AREAS.
DATA DIVISION.
FILE SECTION.
FD   TEST-SBB
        LABEL RECORDS ARE OMITTED.
01   TEST-SBB-RECORD   PIC X(100).
WORKING-STORAGE SECTION.
01   EOF-FLAG                PIC 9   VALUE 0.
PROCEDURE DIVISION.
MAIN-LINE.
        OPEN INPUT TEST-SBB.
        PERFORM PROCESS-FILE UNTIL
          EOF-FLAG = 1.
        CLOSE TEST-SBB.
        STOP RUN.
PROCESS-FILE.
        READ TEST-SBB AT END MOVE 1
          TO EOF-FLAG.
```

Reading with Sequential Block Buffering

The main reason Tandem offers sequential block buffering is to improve the performance of programs that read structured files sequentially. This feature is most commonly used in batch processing. Sequential block buffering can improve the performance of most batch programs substantially. The performance improvement varies with the blocking factor of the files read: the bigger the block and the smaller the record, the better the performance improvement.

Without sequential block buffering, in tests on a NonStop TXP system running B00 software, DP2, and code written in COBOL, it took 11.99 seconds to read 2500 records (each of which was 100 bytes long) from a key-sequenced file with a 5-byte key, a block size of 4096, and an average blocking factor of 38.5. That same operation can be completed in 3.77 seconds with sequential block buffering. Thus, sequential block buffering provides better than a factor-of-three improvement.

It is important to note that sequential reading can be important to on-line programs as well as batch programs. Many on-line programs present lists of information or conduct brief file scans. For example, an order-processing application may contain a screen with order-header information and room for nine detail lines. The application can use sequential block buffering to read the detail lines and improve response. Thus, an order display that used to require ten disc-process services (one for the header and nine for the details), can be changed to require only two (one for the header and one for the block of detail records). Sequential block buffering can and should be used to improve the performance of this type of read-only operation.

The only issue that requires consideration when sequential block buffering is being evaluated for a read-only application is whether or not other processes are updating the file while it is being read.

Concurrency Issues Related to Sequential Block Buffering

Because records are read from a private buffer with sequential block buffering, there is always a chance that the buffer may be out of date when another process updates a file that is being read with sequential block buffering. For this reason, the *ENSCRIBE Programming Manual* warns against using sequential block buffering with access modes other than "read-exclusive" or "read-protected." The File System allows other types of access, however. The following considerations are important if shared access is allowed with sequential block buffering.

Sequential block buffering cannot detect record or key locks. This can be both a benefit and a burden. The benefit is that the process reading with sequential block buffering is not impeded by another process' locks. It views the record in its current state as of the time the block is read from disc, and the process does not have to wait to see it. The problem is that if the record is locked, it probably is involved in an update. The process cannot tell if the record image it read is the before image or the after image. It may view inconsistent data, and there is no way to know if it has.

If one process deletes, adds, or updates records while another reads with sequential block buffering, an additional problem may occur. The process using sequential block buffering may see data that is recently deleted or skip records recently added. Depending on the application, these anomalies may be acceptable.

Use sequential block buffering to improve the performance of read-only on-line operations.

In a key-sequenced file, the above operations are potentially multiblock operations. This means that new blocks are added to make room for new or larger records, or that old blocks are emptied and returned to the pool of free blocks. If one of these multiblock operations is performed at or near the same file position as sequential block-buffered reads, one might be concerned about the structural continuity of the file. Sequential block buffering is not confused by block splits, however, as it passes positioning information along with requests to read a next block.

Updating with Sequential Block Buffering and Buffered Cache

There are many applications that perform update operations as they process a file sequentially. One example is an application that purges a key-sequenced customer-order history file. Assume the application is designed to keep 12 months of history on file and delete everything older. Since the primary key is the customer number followed by the order number, and there are no alternate keys, a program must search sequentially for records to purge. Since the job runs monthly, it deletes about 8% of the records in the file each time it is run.

It is possible to take advantage of sequential block buffering and buffered cache in this type of application. The best method to use depends on the type of access allowed to the file. If the file can be opened for exclusive (or protected) access, there is no need to consider the effect of other processes updating the file concurrently. If the file must be opened for shared access, however, programmers should select a method of processing that protects the process from concurrent updates.

The Single-file, Single-read Method

A single-file, single-read method is best when exclusive access is possible. The method is quite straightforward: simply read the record, decide if it should be processed, and then process the record. The same file open used to read the file is used to update it. A sample implementation follows:

```
MAIN-LINE.
        OPEN I-O TEST-SBB PROTECTED.
        PERFORM PROCESS-FILE UNTIL EOF.
        CLOSE TEST-SBB.
        STOP RUN.

PROCESS-FILE.
        READ TEST-SBB NEXT RECORD
          AT END MOVE 1 TO EOF-FLAG.
        IF NOT EOF
          IF UPDATE-NEEDED
            REWRITE TEST-SBB-RECORD.
```

The file is opened for protected I/O access, and RESERVE 2 AREAS is specified in FILE-CONTROL. Thus, both sequential block buffering and buffered cache are invoked. The former speeds up the reads, while the latter speeds up the updates.

On a NonStop TXP system with B00 software, DP2, and code written in COBOL, tests were run to measure the time required to process a file sequentially (with updates of varying percentages of the records) by various methods.[2] In these tests, a key-sequenced file with 2500 records and a 5-byte key was used. Each record was 100 bytes long, the block size was 4096, and the average blocking factor was 38.5. Elapsed time was measured by a COBOL program, and each test was run at least twice on a dedicated system.

The design of the tests provides a worst case scenario, because updates are evenly distributed throughout the file. If 5% of the records are reported updated, every 20th record is changed. This means there is no "clumping" of updates in a block, i.e., some blocks receiving multiple updates while resident in buffered cache (cache-write hits) and others remaining untouched. In a real application clumping would occur, allowing for better performance.

Figure 2 shows how the time required to process the 2500 records varied (under the single-file, single-read method) depending on the percentage of records that were updated during the sequential pass. The dashed curve shows the time required when sequential block buffering and buffered cache were used. The solid curve shows the time required when normal reads and updates were performed (without sequential block buffering or buffered cache).

In Figure 2, the processing time increases (in both cases) as the percentage of records updated increases. If every record read is updated, the entire operation takes 2.5 times as long with normal reads and updates than if the reads and updates were performed with sequential block buffering and buffered cache.

[2]The performance numbers presented in this article should not be viewed as absolute values, valid in any application environment. They only indicate the relative performance that might be expected from various file-access methods. Performance questions relating to specific applications should be resolved through the testing of various file-access techniques in the environment in which they are to be used.

If every 20th record is updated (5% of the total), the entire operation takes 1.8 times as long with normal reads and updates than if the reads and updates were performed with sequential block buffering and buffered cache. As mentioned earlier, if no updates are performed, a threefold improvement is realized with sequential block buffering.

The Double-file, Double-read Method

A double-file, double-read method serves two purposes. First, it makes sequential block buffering easier to use in a shared update environment because concurrency protection can be provided with selective record locking. Second, it allows sequential block buffering to be used without buffered cache. The second file, used for the update and locking operations, is opened for normal I/O.

The method involves first reading through the file with sequential block buffering and selecting records. Then, the key from the selected record is used to reread the record through a second open of the same file. The FILE-CONTROL for this second file open specifies RESERVE 1 AREA (or no RESERVE clause at all), so sequential block buffering and buffered cache are not used. The second read may or may not specify WITH LOCK, depending on the need for concurrency protection. If shared access is specified in the open statement, record locking should be used, and the record image retrieved in the second read should be verified as current; otherwise, protected access should be specified in the open of the second file. The record can then be rewritten, and any lock can be released.

Although this method is less efficient than the single-file, single-read method (in terms of the code that is executed when a record is updated), it eliminates the possibility of overlaying another process' update because a noncurrent record image was used from a sequential block buffer. It also eliminates any risk involved in using buffered cache.

Figure 2

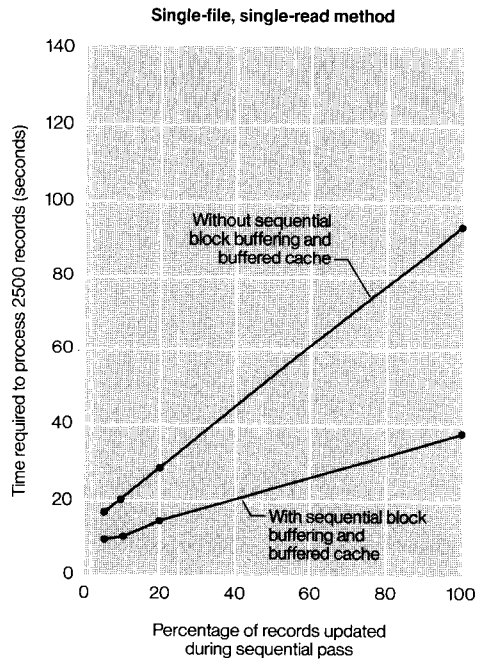


Figure 2.
Processing time for 2500 records (each 100 bytes long) with the single-file, single-read method. The time varies, depending on the percentage of records that must be updated during the sequential pass. A key-sequenced file with a 5-byte key was used. The block size was 4096, and the average blocking factor was 38.5. A NonStop TXP processor, B00 software, DP2, and COBOL code were used.

A sample implementation of the double-file, double-read method follows:

```
MAIN-LINE.
        OPEN INPUT TEST-SBB SHARED.
        OPEN I-O TEST-FILE SHARED.
        PERFORM PROCESS-FILE UNTIL EOF.
        CLOSE TEST-SBB.
        CLOSE TEST-FILE.
        STOP RUN.

PROCESS-FILE.
        READ TEST-SBB NEXT RECORD
          AT END MOVE 1 TO EOF-FLAG.
        IF NOT EOF
          IF UPDATE-NEEDED
            MOVE TS-KEY TO TF-KEY
            READ TEST-FILE RECORD
               WITH LOCK
               KEY IS TF-KEY
            REWRITE TEST-FILE-RECORD
               WITH UNLOCK.
```

Figure 3.

Processing time for the double-file, double-read method. When 100% of the records read are updated, the benefit of sequential block buffering is insignificant. When only 5% of the records read are updated, however, a 44% throughput improvement is realized. (The test environment and file were identical to those for Figure 2.)

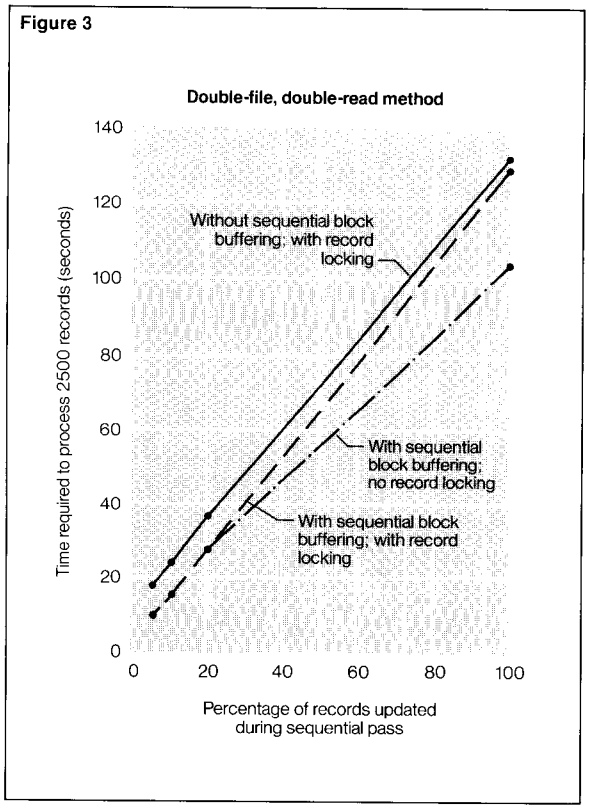
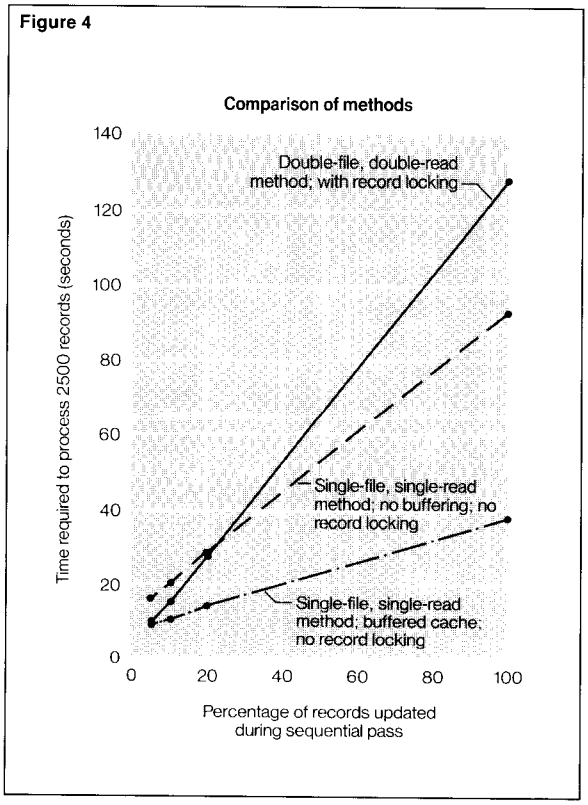


Figure 4.

A comparison of the processing times for various double-file and single-file methods. When only 5% of the file is updated in a sequential pass, the double-file method is about 11% slower than the single-file method with buffered cache and sequential block buffering. (Under those same circumstances, however, the double-file method is still about 37% faster than the single-file method using normal I/O.) As the percentage of records updated increases past 20%, the performance of the double-file method degrades. (The test environment and file were identical to those for Figure 2.)



TEST-FILE is opened for shared I/O access, and RESERVE *n* AREAS is *not* specified in FILE-CONTROL. TEST-SBB is opened for shared input access, and RESERVE 2 AREAS is specified in FILE-CONTROL to invoke sequential block buffering.

Figure 3 summarizes the results of a performance test of a NonStop TXP system doing sequential block buffering by the double-file, double-read method. The test was conducted in the same environment and with the same file as that described earlier. When 100% of the records read were updated, the benefit of sequential block buffering was insignificant. When only 5% of the records read were updated, however, a 44% throughput improvement was realized. (Note: The test also showed that elimination of record locking when protected access is available is slightly more efficient.)

Comparing the Various Methods

As the tests described so far have shown, both sequential block buffering and buffered cache consistently improve performance, although the level of improvement varies from one situation to another.

Figure 4 compares the performance of various double-file and single-file methods. When only 5% of the file is updated in a sequential pass, the double-file method is approximately 11% slower than the single-file method with buffered cache and sequential block buffering. (Under those same circumstances, however, the double-file method is about 37% faster than the single-file method when normal I/O is used.)

The advantage of the double-file method is that there is no risk from using buffered cache, and shared update access is allowed.[3] The 11% additional cost (in which 5% of the file is updated in a sequential pass) is relatively small in light of these advantages. As the percentage of records updated increases past 20%, however, the performance of the double-file method degrades. This degradation is caused by the additional read and record lock requests sent to the disc process.

[3]The risk associated with buffered cache is only considered with unaudited files. Buffered cache is always used for audited files (as of the B00 software release), but the files are protected by TMF recovery mechanisms.

This means that the fastest way to process a file sequentially is to rely on the single-file method (using both sequential block buffering and buffered cache and opening the file for exclusive or protected access). The choice of a next-best alternative (from the standpoint of performance) depends on the characteristics of the file and the percentage of records updated. If a low percentage of records is to be updated, the double-file method with sequential block buffering is probably best. If a large number of records is to be updated, however, the single-file, unbuffered method is best. The exact "break-even" percentage varies from application to application and can only be determined through testing.

Conclusion

Sequential block buffering and buffered cache are important File System features for improving application performance. The new B00 enhancements to COBOL make the features easy for programmers to use.

While it is generally intended for read-only applications, sequential block buffering can help other applications as well. Buffered cache improves the performance of most applications that write, delete, and update. Before selecting the techniques for accessing a file, application designers should (1) analyze the file's characteristics, (2) determine the possibility of concurrent updates to it, and (3) calculate the percentage of its records that are updated.

References

B00 Software Documentation (Softdoc) on COBOL. 1985. B00 software release Site Update Tape (SUT). Part no. T9251B00. Tandem Computers Incorporated.

COBOL Reference Manual. 1985. Section 9, Procedure Division—Verbs. Part no. 82589 A00. Tandem Computers Incorporated.

ENSCRIBE Programming Manual. 1985. Section 4, File Access. Part no. 82083 B00. Tandem Computers Incorporated.

Welsh, R. 1984. Optimizing Sequential Processing on the Tandem System. *Tandem Journal*. vol. 2, no. 3.

Acknowledgments

The author would like to thank Mike Noonan for his helpful suggestions for the technical content of the article. Thanks also to Jim Enright, Jim Gray, Guy Haas, Rich Lynn, Chris Ohland, Emile Roth, Harold Sammer, Dick Thomas, and Rob Welsh for their help in reviewing the article.

Randy Mattran is a senior staff analyst in the Minneapolis District. His activities include application-design support, support for data-base products, performance, and capacity planning. Before joining Tandem in 1981, he spent five years with a consulting firm, designing and developing distributed on-line transaction processing applications. Randy holds a B.B.A. in business computer systems from Eastern Michigan University.

DP1-DP2 File Conversion: An Overview

When systems are converted from Tandem's Disc Process 1 (DP1) to Disc Process 2 (DP2), the disc volumes must be converted to DP2 format. This is because the volume label, directory, and internal structure of structured files on a DP2 volume are different from those on a DP1 volume.[1]

Tandem's DP1-DP2 file-conversion utilities support all conversion requirements, from the simplest to the most complex. Note that while this article emphasizes the more complex file-conversion issues, most DP1-DP2 file conversions will not be complex. An understanding of the issues explained here, however, will enable those responsible for DP1-DP2 conversion to fully plan for the conversion of their files.

[1]A conversion from DP1 to DP2 does not require changes to the applications that used DP1. All programs that access structured and unstructured files in ordinary ways are fully compatible with DP2. As the block structure has changed for structured files, however, those few programs that read structured files with unstructured access will require changes to accommodate this. Also, as DP1 and DP2 audit-trail formats differ, any programs that directly access the audit trail will require modification.

Conversion Utilities

The following Tandem utilities are available for file conversion from DP1 to DP2 (and from DP2 to DP1):

- *BACKUP* has two new options (DP1FORMAT and DP2FORMAT) that write a file to tape in the specified format. If neither option is specified, the format of the tape file will be the same as that of the source disc file.
- *RESTORE* converts files automatically if the format of the tape file differs from that of the destination disc file. Since RESTORE can read all tapes created by BACKUP, it can be used to convert any file.
- The *File Conversion Program (FCP)* is a new utility designed to convert multiple files and volumes in parallel. It converts files from disc to disc, which allows it to convert volumes faster than BACKUP and RESTORE. It *should* be used to convert mirrored volumes, and *can* be used to convert nonmirrored volumes. (For a discussion of the time required to convert files with FCP, refer to the accompanying article, "Determining FCP Conversion Time.")
- The *File Utility Program (FUP) DUP command* converts files automatically if the format of the destination file differs from that of the source file.

File Conversion Assistance Program (FCAP)

The File Conversion Assistance Program (FCAP) automates DP1-DP2 file conversion. It is similar in function to INSTALL, a utility that provides an automated means of generating and installing the GUARDIAN operating system. FCAP invokes the DP1-DP2 conversion utilities at the appropriate time during the conversion process.

FCAP may be used for conversion planning as well. It generates a set of reports from data produced by an FCP ADVISE operation. The reports categorize the FCP data, making it much easier to identify files that require special consideration before starting their conversion.

As FCAP's documentation is an integral part of the program itself, no separate hardcopy manual accompanies it. Instead, one of the options on FCAP's initial menu is to print the user's guide. Refer to the B20 Software Documentation (Softdoc) for further information about FCAP and the user's guide.

DP2 Resource Requirements

Tests have shown that processors containing DP2 typically require more than 2 Mbytes of memory. DP1 requires 80 Kbytes for its code space, whereas DP2 needs 200 Kbytes. Also, DP2 requires additional memory to support a potentially larger cache size.

Before installing DP2, use XRAY™ to evaluate DP1's memory utilization. If the XRAY results indicate memory pressure, add more memory before installing DP2. Excessive page faults can significantly degrade performance.

For DP2 TMF, a volume can contain either audited files or audit-trail files, but not both. This restriction was made to make DP2 software more reliable than that of DP1. The DP1 TMF practice of "cross-auditing" is not allowed with DP2. If all DP1 volumes contain audited files, an additional mirrored volume is required unless all the audited files on a mirrored volume can be moved to other volumes.

Restrictions on Mixing DP1 and DP2 Volumes

As there is no requirement for all volumes on a node to have the same format, a single node can contain both DP1 and DP2 volumes. This means that, if appropriate, one or two volumes can be converted at a time, as opposed to all volumes being converted at once. The following are the restrictions associated with mixing DP1 and DP2 volumes on the same node:

- All volumes connected to the same disc controller must have the same disc-process type. A controller string must not contain a mix of DP1 and DP2 volumes.
- If a file has alternate keys, the primary file and the alternate-key files must have the same disc-process type.
- All partitions of a file must have the same disc-process type. This includes files that are partitioned across nodes.
- For TMF, audited files and audit-trail files must be on volumes with the same disc-process type. This usually means that all volumes on a node that uses TMF must be converted at the same time.

Changes in File Characteristics

DP2 introduces several changes in file characteristics. In some instances, described below, these changes will require special consideration and action before conversion.

Fewer Valid Block Sizes

DP2 block sizes are limited to power-of-two multiples of the sector size (512, 1024, 2048, or 4096 bytes). This means DP2 does not support four DP1 block sizes: 1536, 2560, 3072, and 3584 bytes. The conversion utilities adjust the block size of any structured file having one of the invalid DP1 sizes to the next highest DP2 block size. (Thus, block size 1536 is adjusted to 2048, and block sizes 2560, 3072, and 3584 are adjusted to 4096.)

This adjustment may not be optimum for a particular file. To avoid this condition, run the FCP ADVISE command on all files first to identify those whose block sizes will change (or use FCAP; the planning reports that it generates identify these files). Using a valid DP2 block size, restructure those files whose block size would be adjusted inappropriately during conversion; then convert the files.

Index and Data Block-size Requirements for Key-sequenced Files

DP2 requires index and data blocks in a key-sequenced file to be the same size, while DP1 does not.

DP1 Block-size Requirements. A DP1 file uses different index and data block sizes primarily to achieve optimum performance. DP1 was designed to use write-through cache, in which every write operation causes an immediate disc I/O. (With the B00 software release, DP1 began using buffered cache for audited files only.) Optimum block sizes for a DP1 file are determined by its write-through cache requirements, as discussed below. (For a comparison of DP1 and DP2 cache, see Schachter, 1985.)

For random access of a DP1 file for which write-through cache is used, a small data block size is desirable because few, if any, read cache hits are expected. Less of the finite cache resource is used to hold the block, and the number of bytes written to disc for a write operation is minimized. If the index block size is also small, however, an excessive number of index levels have to be accessed when a data block is retrieved. In this case, a large index block size is appropriate.

For sequential writes to DP1 files for which write-through cache is used, a small block size minimizes the number of bytes written to disc for each write operation.

DP2 Block-size Requirements. In DP2, because all of a file's blocks are cached in the same buffer and a separate cache buffer is used for each block size, block sizes for DP2 index and data files must be identical.

Also, DP2 cache can be either write-through or buffered. The default is buffered for audited and write-through for nonaudited files. Nonaudited files may be buffered, if appropriate, however. A buffered file that is written to sequentially should use a large block size to take advantage of cache write hits. Even files that are written to randomly benefit from using buffered cache because the disc I/O does not have to be performed immediately.

Conversion Considerations. Because DP2 requires the index and data blocks of key-sequenced files to have the same block size, the conversion utilities automatically change the block sizes of DP1 files with differing index and data block sizes.[2] During the conversion, the block size becomes the larger of the DP1 data and index block sizes rounded up to a valid DP2 block size. (For example, a DP1 file with an index block size of 1536 and a data block size of 512 would have a DP2 index and data block size of 2048.)

Before converting files from DP1 to DP2, use FCAP to identify all files whose index and data block sizes are different. For each of these, determine which is optimum for the DP2 cache scheme: the block size it will automatically be given by the conversion utility or one of the other valid DP2 block sizes.

It is best to restructure, before conversion, those files whose post-conversion block size would be inappropriate. For a large file, however, it may be more practical to restructure the file while converting it. Use the FCP DUP command's BLOCKSIZE option for this.

[2] All DP1-DP2 file-conversion utilities (BACKUP, RESTORE, FCP, and FUP) follow the same rules when making block and extent size adjustments. Thus, regardless of the conversion utility used, the adjustments made are the same.

Preserving a File's Address Space

Bit-map Blocks and Address Space. DP2 relative and key-sequenced files contain bit-map blocks in addition to index (key-sequenced) and data blocks. Bit-map blocks are used for free-space allocation within the file. With the addition of these bit-map blocks, it becomes necessary to distinguish between address space and total file space.

Address space can be defined as the total amount of space in a file that is available for the storage of data and index (key-sequenced) information when all the file's extents are allocated. It excludes any space required for bit-map blocks. *Total file space* can be defined as the total amount of space available in a file when all the file's extents are allocated, including address space and bit-map block space. Within these definitions, DP1 address space equals the total file space, as it does not use bit-map blocks. Thus, for DP2

address space = total file space - bit-map block space,

while for DP1

address space = total file space.

Conversion Considerations. For relative and key-sequenced files, the file conversion utilities attempt to preserve address space by adjusting the size of the total file space to compensate for the presence (DP1 to DP2) or absence (DP2 to DP1) of bit-map blocks in the converted file. They increment a file's primary extent size when converting it from DP1 to DP2 or decrement its primary extent size when converting it from DP2 to DP1.

The adjustment factor is a multiple of the file's block size. The smallest unit of allocation in a disc file is one page (2048 bytes). Thus, if the adjustment factor is not a 2048-byte multiple (possible for block sizes of 512 or 1024), the conversion utility rounds up the factor to the next highest page value for DP1-to-DP2 conversions and the next lowest page value for DP2-to-DP1 conversions. After an extent adjustment, a converted file has the same amount of address space as the source file or slightly more.

It is evident that DP2 relative and key-sequenced files require slightly more disc space than their DP1 counterparts. If few of these files reside on a volume, the additional space requirement is minimal (1% or less). If a volume contains a large number of these files, however, the additional space requirement could be significant, especially if the volume is almost filled to capacity. The additional space required could be as much as 5% but typically is in the range of 1% to 2%. Use the FCP ADVISE command as an aid in estimating the additional space needed for conversion. One or more additional disc volumes may be needed if the additional space is not available.

In conversion, block size becomes the larger of the DP1 data and index block sizes rounded up to a valid DP2 block size.

DP2 Blocks Must Reside in the Same Extent

There are other reasons that a file's extent sizes may be adjusted during conversion. For DP2, a block must reside in the same extent, while for DP1, a block may be split between two extents, such that the first half of a block can fall at the end of one extent and the last half reside at the beginning of the next. This occurs if, for a file whose block size is 4096, either the primary or secondary extent size is an odd number. For any structured file in this condition, the DP2 conversion utilities increment the extent size to the next even value.

Difference in Number of Extents Allowed

A DP1 file may only have a maximum of 16 extents allocated (1 primary and 15 secondary). This limit is not adjustable. Thus, for DP1,

total file space (in pages)
= primary extent size
+ (15 * secondary extent size).

The maximum number of extents for a DP2 nonpartitioned file is dynamically alterable and is limited by the space available in the file label. This allows for over 900 extents in most instances. A new file characteristic, MAXEXTENTS, dictates the maximum number of extents allocatable for a DP2 file. Thus, for DP2,

total file space (in pages)
= primary extent size
+ ((MAXEXTENTS - 1)
* secondary extent size)

To accommodate this difference, when converting a DP2 file whose MAXEXTENTS value is greater than 16 back to DP1, the conversion utilities adjust the primary and secondary extent sizes so that the total file space fits into 16 extents. For example, Table 1 shows the characteristics of a DP2 unstructured file whose MAXEXTENTS value is greater than 16 and the new values for these characteristics after the file has been converted to DP1. Note that the total file space has been maintained but the extent sizes have changed considerably.

Table 1.
Characteristics of a DP2 unstructured file whose MAXEXTENTS value is greater than 16, before and after it is converted to DP1.

Characteristics	Under DP2	After conversion to DP1
Primary extent size	10 pages	20 pages
Secondary extent size	10 pages	132 pages
MAXEXTENTS	200	16
Total file space	2000 pages or [10 + (200 - 1) * 10]	2000 pages or (20 + 15 * 132)

Partitioned Files

While the parts of any key-sequenced partitioned file can be converted individually, for entry-sequenced and relative partitioned files, all parts might have to be converted as a unit.

Key-sequenced Files

Key-sequenced files are partitioned at file creation when the primary key values for the range of records that are to reside in each part of the file are specified. It is not absolutely essential that address space be preserved in each part during file conversion. Records destined to reside in one part before file conversion reside in the same part after conversion, even if that part's address space is incremented slightly by the conversion process. Thus, regardless of any changes in a key-sequenced file's characteristics, it is always possible to convert each part individually.

Entry-sequenced and Relative Files

For an entry-sequenced or relative partitioned file, the position of a record in the file is dependent on each part's address space. If an individual part's address space were not preserved during conversion, records in one part might fall into another part. If this would occur for any part of a file, all parts of the file must be converted as a unit; the conversion utilities will not convert each part individually.

Also, the parts of an entry-sequenced or relative partitioned file whose block size would change as a result of conversion must be converted together. An individual part may not be converted separately because it is highly probable that some records would fall into different parts after conversion.

Conversion Example 1. A part of a DP1 entry-sequenced partitioned file has these characteristics:

Characteristics	Value
Primary extent size	12 pages
Secondary extent size	15 pages
Block size	4096 bytes

For this file, a 15-page secondary extent size is not valid for DP2 because the block size is 4096 (as explained earlier). This extent size must be an even number, but if it were adjusted to 16 pages, the address space would not be preserved. Thus, the part cannot be converted individually.

Conversion Example 2. A part of a relative partitioned file has a block size of 512 bytes. When it is converted, space must be added (DP1 to DP2) or subtracted (DP2 to DP1) to compensate for bit-map blocks. The block size is not a multiple of a disc page (2048 bytes), however. In all probability, address space would not be preserved if the primary extent were adjusted; therefore, the part cannot be converted individually. This is also true for a relative partitioned file with a block size of 1024.

Converting Files Whose Parts Must Be Converted as a Unit

Use FCAP to identify those files whose parts can be converted individually with FCP and those that must be converted as a unit with BACKUP and RESTORE or FUP DUP. (FCP is not capable of converting all parts as a unit.)

Files That Cannot Be Converted

FCAP produces reports identifying all files that cannot be converted. (It runs the FCP ADVISE command and generates the reports upon completion of the ADVISE operation.) The following describes the files that cannot be converted.

Broken Files

A file must have structural integrity before it can be converted, as none of the conversion utilities convert broken files. (The FCP ADVISE command's VERIFY option identifies such files. Note that this option checks only for errors that would prevent a file from being converted; it does not check for all possible structural errors.)

TMF Audit-trail Files

The internal format for TMF audit-trail files is different for DP1 and DP2; also, after a conversion, the required TMF initialization invalidates all previous audit trails. For these reasons, audit-trail files (those with a file code of 134) must not be converted. (FCP does not convert these files. The other conversion utilities do, but the contents of the converted files are useless.)

DP2 Records Longer Than 2035 Bytes

DP2 key-sequenced files may have records longer than 2035 bytes, the maximum record length for DP1. These files cannot be converted to DP1.

DP2 Primary Files Having More Than 26 Alternate-key Files

A DP2 file label is larger than a DP1 file label, allowing the specification of more alternate keys and alternate-key files than are allowed under DP1. As a rule, a DP2 primary file with more than 26 alternate-key files is not convertible to DP1.

Conclusion

An understanding of DP1-DP2 file-conversion issues is essential for the successful conversion of a data base. While not all file conversions will be complex, it is important that those responsible for a conversion understand the file-conversion process and the changes in the physical implementation of the data base that may result.

References

B20 Software Documentation (Softdoc) on the File Conversion Assistance Program (FCAP). 1985. B20 software release Site Update Tape (SUT).

Carlyle, K. and McGowan, L. 1985. DP2 Highlights. *Tandem Systems Review*. vol. 1, no. 2. Tandem Computers Incorporated.

DP1-DP2 File Conversion Manual. 1985. Part no. 82407 B00. Tandem Computers Incorporated.

DP2 Class. 1985. Course no. 38448-A00. Tandem Computers Incorporated.

Schachter, T. 1985. DP2's Efficient Use of Cache. *Tandem Systems Review*. vol. 1, no. 2. Tandem Computers Incorporated.

Jim Tate developed the File Conversion Program (FCP) and the File Conversion Assistance Program (FCAP). He joined Tandem in 1979 as an instructor and course developer in Hardware Training. After that, he became a course developer in Software Education and, then, District Systems Manager in Phoenix. He is currently an advisory staff analyst for the Large Systems Support Group. Before joining Tandem, Jim supported automated warehousing systems. He has 17 years of computing experience.

Determining FCP Conversion Time

The File Conversion Program (FCP) is a Tandem utility for converting files from Disc Process 1 (DP1) format to Disc Process 2 (DP2) format (and vice versa).[1] This article explains how to determine the amount of time it will take to convert files with FCP. It provides a model that can be used to estimate the amount of time that will be required to convert any volume. The model was derived from conversion tests also described in the article.

[1]For a description of the DP1-DP2 conversion utilities and an overview of conversion considerations, see the previous article, "DP1-DP2 File Conversion: An Overview," also by Jim Tate.

[2]For a complete list of DP1-DP2 file-conversion steps, see the *DP1-DP2 File Conversion Manual*.

Basic File Conversion Steps

Below is an abbreviated list of DP1-to-DP2 file-conversion steps, containing those steps that take the most time to perform.[2] This article focuses on Step 8. (Note that this step may take as little as a quarter of the time needed to perform all of the conversion steps listed.)

The list below assumes that a B00 or later release of the GUARDIAN 90 operating system has been installed and that the SYSGEN for DP2 has been performed. It also assumes that all volumes to be converted are mirrored, although FCP can be used on nonmirrored volumes if an extra disc drive is available as the destination disc.

1. Shut down all applications and subsystems.
2. Back up all files (usually to tape, but a disc can be removed to accomplish this step).
3. Run FCP ADVISE, VERIFY on all volumes being converted.
4. Use BACKUP to back up files not convertible by FCP. (This will be a small subset of the files backed up in Step 2.)

5. Shut down the "old" system.
6. Copy the "new" system-image tape to disc and cold load the system.
7. Convert the files that must be converted by RESTORE.
8. Use FCP to CONVERT all other files.
9. Run INSTALL and perform the REPSUBSYS phase.
10. Run FILCHECK to insure the structural integrity of all structured files.
11. Start up all subsystems and applications.
12. Revive the volumes.

Main Factors Affecting FCP Conversion Time

The main factors that may influence the amount of time needed for an FCP conversion are:

- Processor type (NonStop II™ or NonStop TXP) and disc-controller type (3106 or 3107).
- Number of files on the volume.
- Average file size on the volume.
- File type.

Processor and Disc-controller Types

The types of processor (NonStop II or NonStop TXP) and disc controller (3106 or 3107) are the primary hardware factors. They dictate the conversion transfer rate. The amount of time required to create and update the converted files is dependent on the processor type.

Number of Files on the Volume

The number of files on a volume correlates directly with the time required to convert the volume. The greater the number of files, the longer it will take to convert the volume. For every file, FCP must create the destination file, allocate disc space, and update the file label after conversion.

Average File Size on the Volume

Large files convert at a higher rate than do small files (28 Kbytes or less), for two reasons:

1. For small files, the time required for file creation, disc-space allocation, and the update of the file label is a significant portion (perhaps 50% or more) of the total time required to convert the file. This lowers the file's overall conversion rate.
2. Each file is converted by a pair of FCP processes (FCP1 and FCP2). The two processes are designed to overlap reading from the source file and writing to the destination file. For small files (28 Kbytes or less) there is no overlap, however, because of the small amount of data involved. Thus, the file's overall conversion rate is lower.

File Type

Unstructured files are the fastest to convert because only their file labels need to be modified. Data is simply copied from the source file into the destination file without changes.

For a DP1 entry-sequenced or relative file, the logical and physical block positions within the file are the same. The file can thus be read sequentially, simplifying conversion. Multiple blocks can be read in a single read operation.

A key-sequenced file is the slowest to convert because the source file must be read via the index so that records can be extracted in logically ascending order. This means only one block can be read with each read operation. For this reason, a key-sequenced file with a 4096-byte block size converts at a higher overall rate than a key-sequenced file with a 512-byte block size.

Table 1.
Characteristics of the file sets generated for the DP1-DP2 File Conversion Program (FCP) tests.

	File set 1	File set 2	File set 3	File set 4
Records/file	250	2500	25,000	125,000
Average file size	24 KB	220 KB	2.2 MB	10.4 MB
Total number of files	2000	200	20	4
Number of unstructured files	500	50	5	1
Number of entry-sequenced files				
512-byte block	125	12	1	—
1024-byte block	125	12	1	—
2048-byte block	125	13	1	—
4096-byte block	125	13	2	1
Number of relative files				
512-byte block	125	12	1	—
1024-byte block	125	12	1	—
2048-byte block	125	13	1	—
4096-byte block	125	13	2	1
Number of key-sequenced files				
512-byte block	125	12	1	—
1024-byte block	125	12	1	—
2048-byte block	125	13	1	—
4096-byte block	125	13	2	1
File extent sizes (both primary and secondary) in pages	6	60	600	6000

FCP Conversion Tests

The FCP conversion tests were designed to take into account the main factors that affect FCP conversion time. The B00 versions of DP2 and FCP were used.

File Description

Four different file sets were generated (see Table 1), with average file sizes ranging from small (24 Kbytes) to large (10.4 Mbytes). Each file set was individually loaded onto a DP1 mirrored volume, which was then converted to DP2.

Each file set consisted of 500,000 80-byte records distributed evenly among the four different types of file. The total size of each file set was approximately 45 Mbytes. This was deemed to be sufficiently large to yield meaningful FCP conversion-time data.

All files in a file set contained the same number of 80-byte records. This included the unstructured files, whose end-of-file value equalled the number of records per file multiplied by 80 bytes. Only valid DP2 block sizes were used for the structured files (512, 1024, 2048, and 4096 bytes). All key-sequenced files were generated with a block SLACK value of 10%.

Configuration

Figure 1 represents the hardware configuration for the test system. Only volumes $DATA1 and $DATA2 were used for the conversion tests. They were both mirrored 4114/4115 volumes.

For each of the four file sets, the FCP CONVERT operation was run twelve times, measuring the elapsed conversion times for one, two, and three FCP1-FCP2 process pairs, on either a NonStop II or a NonStop TXP processor connected to either a 3106 or a 3107 disc controller.

Only one volume was converted at a time, and no other activity was present on the system during the tests. All FCP output data was directed to a disc file on $SYSTEM. Volume $DATA1 was used for the tests of file sets 1 and 3, and $DATA2 for the tests of file sets 2 and 4.

Test Results

The elapsed time required to convert a file set with FCP for each hardware configuration is listed in Table 2. These results show that there is almost a linear relationship between the average file size and the average conversion rate. As the average file size increases, the average conversion rate also increases (elapsed conversion time decreases).

Figure 1

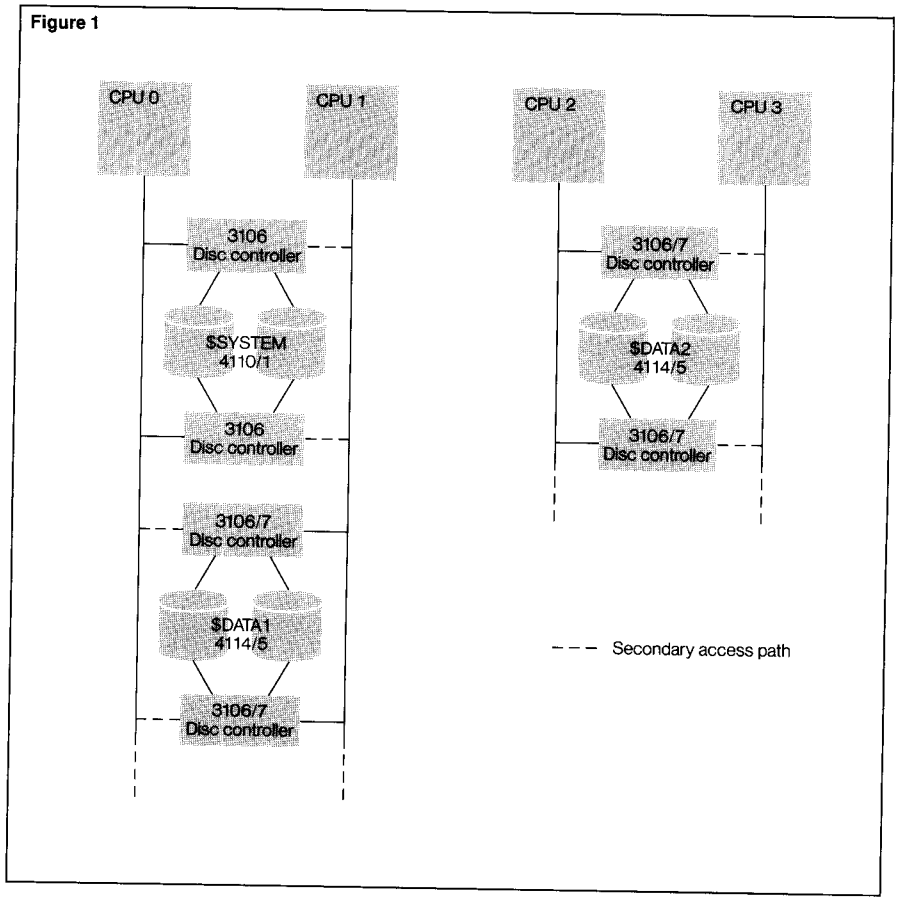


Table 2.
Elapsed time (in minutes:seconds) required to convert four file sets from DP1 to DP2 format with the File Conversion Program (FCP).

File set	Number of FCP1-FCP2 process pairs	NonStop II processor		NonStop TXP processor	
		3106 disc controller	3107 disc controller	3106 disc controller	3107 disc controller
File set 1 (24 Kbytes, 2000 files)	1	50:47	45:31	35:22	31:29
	2	42:28	39:27	31:34	28:58
	3	42:21	39:33	31:57	29:14
File set 2 (220 Kbytes, 200 files)	1	17:58	15:11	11:06	10:39
	2	15:50	14:27	11:31	11:06
	3	16:16	14:50	12:27	11:29
File set 3 (2.2 Mbytes, 20 files)	1	14:41	11:23	9:48	8:00
	2	14:12	11:37	11:15	9:24
	3	14:25	11:33	11:36	9:07
File set 4 (10.4 Mbytes, 4 files)	1	10:05	7:40	5:32	5:20
	2	9:10	7:31	7:24	5:26
	3	10:10	7:29	6:45	5:30

Figure 1.
The hardware configuration used in the DP1-DP2 File Conversion Program (FCP) tests.

Table 3.
Hardware-dependent values for variables *f1*, *f2*, and *f3* of FCP conversion-time model.

Variable	NonStop II processor		NonStop TXP processor	
	3106 disc controller	3107 disc controller	3106 disc controller	3107 disc controller
f1 (file creation disc-space allocation, and file-label update time, in secs/file)	0.7	0.7	0.5	0.5
f2 (FCP overhead)	17.8	16.9	24.5	16.5
f3 (transfer rate in Kbytes/sec)	50.7	61.5	86.4	91.2

Figure 2

```
:DSAP /OUT $S/ $CAT, BYSUBVOL
PAGE 0      DSAP — $CAT on \SUPPORT —
Disc Space Analysis Program — T9074B00 - (28JAN85)
Volume $CAT is logical device 9
Device type is 3, subtype 3 ( 4104 — 240MB )
      114,026 pages (2048 bytes) on volume
  233,525,248 bytes on volume

Summary of space use on $CAT
      10,908 free pages in 642 extents (9.5%).
     100,258 allocated pages in 8,771 files in 17,088 extents (87.9%).
      12,297 unused pages in 3,131 files (10.7%).
       1,949 deallocatable extent pages in 33 files (1.7%).

Space Allocation Consistency Analysis:
No space allocation anomalies.

Media Failure Analysis:
Primary disc has no unspared defective sector(s).
Mirror disc has no unspared defective sector(s).

PAGE 1      DSAP — $CAT on \SUPPORT — Subvol Summary Report
```

	Files	Total Pages	Unused Pages	Dealloc Pages	Large File	Min Age
FREE SPACE		10908				
DISC DIRECTORY		2860				
TEMPORARY FILES	6	198	54	48	64	0
ABUDPARM	4	372	350	0	93	446
ABURUN84	1	93	90	0	93	446
ACCESS	6	106	7	0	80	607
ACONFIG	3	16	1	0	8	607
ADVENT	16	305	31	0	62	328
ADVENT2	2	74	3	0	72	614
AIDDEV	59	1005	92	0	228	14
ALGORITH	4	34	6	0	16	614

Figure 2.
Example Disc Space Analysis Program (DSAP) report used in determining the values of as *(average file size) and* nf *(number of files) for the conversion-time model.*

Estimating File-conversion Time

Conversion-time Model

Based on the test results, the following model to estimate the amount of time required to convert a volume was developed:

Conversion time in seconds

$$= \left(f1 + \frac{as + f2}{f3}\right) * nf$$

where $f1$ = file creation, disc-space allocation, and file-label update time in seconds/file

$f2$ = an FCP overhead factor

$f3$ = transfer rate in Kbytes/second

as = average file size in Kbytes/file

nf = number of files

The values of *f1*, *f2*, and *f3* depend on the hardware configuration, i.e., the processor and disc-controller types. Use Table 3 to determine these for a specific hardware configuration.

In determining the value of *f1*, a modified version of FCP1 was used to capture the actual time required to create the destination file, allocate disc space, and update the file label. The value proved to be processor dependent.

In calculating the values of *f2* and *f3*, the elapsed times for file sets 1 and 2 with one FCP1-FCP2 process pair were used.

Other Factors Affecting Conversion Time

Many other factors, besides those previously mentioned, can affect conversion time. If a volume has a predominance of unstructured files, it will probably convert in less time than the model would indicate. Conversely, if key-sequenced files predominate, more time will probably be required.

If FCP is to be used to convert multiple volumes in parallel (as it was designed to do), resource contention may cause the conversion time for a volume to increase. Thus, the model-based estimate should be viewed as an approximation with an accuracy of ±25%.

Using the Model

The values for the average file size (*as*) and number of files (*nf*) can be determined with the Disc Space Analysis Program (DSAP), a GUARDIAN 90 utility. For this explanation, the DSAP output example in Figure 2 is used as the basis for determining the values of *as* and *nf*.

Page 0 of the DSAP example indicates 100,258 pages are allocated to 8771 files. Of this total, however, 12,297 pages are unused; i.e., they do not currently hold any data. To find out the number of disc pages that contain data, subtract the unused pages from the allocated pages (in this example, 100,258 - 12,297 = 87,961).

Then determine the average file size in Kbytes. (Set aside the six temporary files mentioned on page 1 of the report for later consideration.) To calculate the average file size, multiply the total data pages by 2 Kbytes (the size of a disc page) and divide the result by the number of files on the volume. For this example, the average file size is

$$\frac{87{,}971 \text{ pages} * 2 \text{ Kbytes}}{8771 \text{ files}} = 20.1 \text{ Kbytes/file.}$$

Now consider the temporary files mentioned on page 1. As FCP does not convert temporary files, subtract them and the space they use from the totals. In this example, the total number of files is 8771 - 6, or 8765 files, and the total space used is 87,961 - (198 - 54), or 87,817 pages. Thus, the average file size is

$$\frac{87{,}817 \text{ pages} * 2 \text{ Kbytes}}{8765 \text{ files}} = 20.0 \text{ Kbytes/file.}$$

To calculate the time required to convert the volume, use 20 Kbytes/file as the average file size (*as*) and 8765 as the number of files (*nf*). Also, for this example, assume the system has NonStop TXP processors and 3106 disc controllers. Thus, the approximate time required to convert the volume is

$$\text{Time} = \left(0.5 + \frac{20.0 + 24.5}{86.4}\right) * 8765$$

or 8897 seconds (148.3 minutes or 2.5 hours).

If this volume were on a NonStop II system using 3106 controllers, the approximate time required for conversion would be

$$\text{Time} = \left(0.7 + \frac{20.0 + 17.8}{50.7}\right) * 8765$$

or 12,670 seconds (211.2 minutes or 3.5 hours).

Conclusion

The amount of time required to convert a volume from DP1 to DP2 is dependent on many factors. This model for estimating FCP conversion time takes into consideration the main factors affecting FCP conversion. It should be helpful for calculating the amount of time it will take FCP to convert a specific volume.

References

DP1-DP2 File Conversion Manual. 1985. Part no. 82407 B00. Tandem Computers Incorporated.

Jim Tate wrote this article, as well as the accompanying article, "DP1-DP2 File Conversion: An Overview."

TACL, Tandem's New Extensible Command Language

For some time, users of Tandem systems have asked for an interface to the GUARDIAN operating system that is more flexible and powerful than COMINT, Tandem's command interpreter. To answer this need, Tandem has developed a new integrated command language that can be used to perform simple interactive functions as well as to automate complex procedures. TACL™, the Tandem Advanced Command Language, is available for use with GUARDIAN 90 in the B20 software release.

TACL's basic command-interpreter features include:

- Support of COMINT commands.
- Support of user-defined alternate command names (aliases).
- A command history, allowing reexecution and/or modification of previously entered commands.
- Function-key definitions.
- Prompts containing status information.

TACL's advanced command-language features include:

- Extensibility, allowing user-written commands with full functionality.
- A "help" facility that describes the syntax expected next.
- Support of wild cards for file naming.
- Support of macro files (files containing a series of commands in the order and format they would be typed in interactively).
- An implicit RUN command, allowing programs or macro files to be invoked by file name only.
- Support of functions that return a value, allowing the results of one command to be used as the arguments of another (similar to UNIX pipes).

TACL's extensibility is achieved through the following features traditionally available only in programming languages:

- Transparent type conversion between numeric and string data.
- Arithmetic and logical expressions.
- Variables (which can be used as stacks).
- Procedural constructs (macro, text, and routine functions).
- Control structures (IF, labeled CASE, recursion, WHILE-DO, and DO-UNTIL loops).
- Exception handling.
- A debugging facility which allows step-by-step or breakpoint debugging.

- Sequential I/O.
- GUARDIAN 90 interface procedures.
- Support of variables used for process communication.
- Text-editing primitives.
- Aids for parsing complex argument strings.

The basic command-interpreter features are described in the TACL manuals listed at the end of this article. The more advanced features and the programming features are described in the following sections. Examples of how they can be used are included.

Advanced Command-language Features

Extensibility

All commands in TACL are implemented as functions, the TACL equivalent of procedures. Each TACL user's environment is initialized with a standard set of functions, the TACLBASE functions, that implement commands compatible with COMINT. Users can add to or replace these functions at any time by creating new functions that use existing commands and built-in functions. The new functions can be as simple as COMINT commands or as complex as programs.

The built-in functions are TACL's predefined building blocks. Many of these functions provide a high-level interface to GUARDIAN 90 procedures such as FILEINFO and PROCESSINFO. Others allow new, more flexible ways of using the system, such as selecting sets of files using wild-card notation (TACL's file-name templates).

Help Facility

TACL provides three facilities for aiding interactive users. First, users can display a list of the available built-in functions by typing the command BUILTINS. They can also press the predefined "help" key, F16, to display the syntax options of any command, including that of user-defined commands. Finally, as TACL evaluates an incorrectly typed command, it issues an error message indicating the syntax it was expecting. Users can then correct the command without having to refer to a manual. (This feature is also available to TACL programs.)

Figure 1

```
?SECTION print ROUTINE
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = TGALs all files matching the file-name template
= = passed to the routine.
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

#FRAME              {Make it easy to clean up PUSHed variables.}
#PUSH template      {Prepare a variable to hold the argument.}

#PUSH arg_type, filename      {Prepare variables used by DO_EACH routine.}

SINK [#ARGUMENT /VALUE template/ TEMPLATE]      {Get the template argument.}

= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = Define a routine that TGALs each file.
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
[#DEF do_each ROUTINE
|BODY|
    [#LOOP      {Loop once for each file name passed to this routine.}
    |DO|
    = = Find out if the argument is a file name (1)
    = = or the end of the argument string (2)
    = = and save the value of it
        #SET arg_type [#ARGUMENT /VALUE filename/ FILENAME END]

        [#IF (arg_type = 1)      {If a file name, TGAL it.}
        |THEN|
            TGAL /in [filename], out $s.#tgal, nowait/
        ]       {if}

    |UNTIL|
        (arg_type = 2)      {When no more file names, quit.}
    ]       {loop}
]   {def}

= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = Execute the routine that TGALs each file, passing it
= = the complete list of file names that match the template
= = we were given.
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
do_each [#FILENAMES [template]]

#UNFRAME     {Clean up all variables we created here.}
```

File-name Templates (Wild Cards)

Users of Tandem systems have requested a way to perform an operation, such as purging or printing, on a set of files without having to type each file name in the set. With TACL, users can do this by giving a file-name template as an argument to a function. For instance, the command FILENAMES accepts a file-name template and prints the names of the files the template specifies in a format similar to that used by the familiar COMINT FILES command. The example in Figure 1 illustrates the contents of a routine named PRINT that accepts a file-name template as an argument and TGALs each file name that matches it.

Figure 1.
This PRINT function demonstrates the argument-handling capabilities of a TACL routine.

Macro Files

A desirable trait of any command language is the capability of executing commonly used sequences of commands without typing them each time. Some of the tools used in the Tandem environment to accomplish this are OBEY, STREAM, and EXEC. TACL allows users access to these tools, but introduces macro files as a simpler means of accomplishing this goal.

A macro file is simply an edit file that contains one or more commands. These commands are executed in sequence when users invoke the macro file by its file name. A macro file can contain a few commands typed in the same format as they would be typed interactively or a few thousand commands utilizing some of TACL's more sophisticated features. It can also contain definitions of new functions to be used as subroutines.

Users can define dummy arguments in macro files and pass arguments to the macros at start-up time. They can achieve even more sophisticated argument handling in macro files by using routines within the macro files.

Finally, through macro files, TACL can determine which actions to take dynamically, based on the results of previous actions and the characteristics of the current environment, rather than basing its actions solely on the environment that existed when a process started.

Implicit RUN or Function Invocation

TACL users can execute a macro file or program (object) file simply by typing the file's name. If the name is not fully qualified, TACL searches a user's current search list of subvolume names. The default search list contains \$SYSTEM.SYSTEM only (simulating the way COMINT handles program file names). Users can modify their individual search lists, however, to have TACL look for programs and macros in other subvolumes before or after searching \$SYSTEM.SYSTEM. The list can contain a user's current subvolume and/or any other locations.

For program files, all RUN options (including DEBUG) are available with this implicit RUN feature. The search feature is not available when the RUN command is explicitly used.

Explicit Invocation and Pipe-like Usage

The *TACL User's Guide* and *TACL Quick Start* explain how users can load libraries of commands to customize their individual environments. They then execute these commands by typing the command name and command arguments on a single line and pressing RETURN. (This method of invoking commands or functions can be considered implicit invocation.)

Explicit invocation provides additional functionality. Users can invoke commands explicitly by surrounding them in square brackets ([ ]). TACL evaluates any command surrounded by square brackets as soon as all left and right square brackets match. This allows users to place multiple commands on a single line and to spread a single command over multiple lines.

For instance, the commands

```
1> [time] [status *, user]
```

cause the time and then the status information to be displayed.[1] Similarly, TACL does not evaluate the command

```
2> [ status
2>    *,
2>  user ]
```

until the square brackets match.

Explicit invocation also allows users to nest commands so that the output of one function can be passed as input directly to another. Before TACL was available, it was necessary to store the results of a command in an intermediate process, variable, or file so that it could be altered into acceptable input for another function.

[1]The TACL prompt is a "greater than" sign (>). The number appearing to the left of the prompt is the count of the command in the sequence of commands the user has typed (e.g., a number 1 shows to the left of the prompt for the first command typed in, a 2 for the second command typed in, etc.).

For example, to obtain a timestamp and display it in a format of month, day, year, and time, users had to call TIMESTAMP, save the result, pass it to CONTIME, save CONTIME's results, and convert them to date and time format. In TACL the timestamp conversion can be done in a single command, as illustrated below:

```
3> #OUTPUT [_Contime_To_Text
3> [#CONTIME
3> [#TIMESTAMP]]]
June 23, 1985 12:35:06
```

In the above example, TACL executes the function #TIMESTAMP and passes the value returned as an argument to #CONTIME. It then executes #CONTIME and passes its value as an argument to the TACLBASE function _Contime_To_Text. TACL then passes the result to #OUTPUT and executes it, displaying the formatted date on the user's terminal.

Programming Facilities

Transparent Data Type Conversions

From a TACL user's point of view, all data is textual. In fact, TACL recognizes integers as well and can perform arithmetic and logical operations on integers. TACL makes any conversions between numeric values and ASCII that might be required. In addition, whenever users need to supply a number as an argument to a TACL routine, they can use the name of a variable containing a number.

Arithmetic and Logical Expressions

TACL also allows the use of an arithmetic expression wherever a number or numeric variable name is expected. Such an expression must be enclosed in parentheses, and can include other arithmetic expressions, integer numbers, numeric variables, and operators. Operators can be arithmetic (+ , -, *, /) or logical (NOT, AND, OR, <, >, =, <=, >=, <>). Parentheses can be used to control the order of evaluation. The value of a logical expression is either -1 (true) or 0 (false).

Variables

In most programming languages, variables are used to store values. A variable's current value is substituted for its name each time it is encountered. Some languages also allow a variable to be a function or procedure that is executed whenever its name is used. Generally, a variable is defined to be of a specified type, for example, integer or string. All these capabilities are also true of TACL variables. In TACL, however, a variable's value and type and the manner in which it is used can be quite different from those in other programming languages.

In TACL, variables are actually stacks, and the number of levels in a variable is limited only by the fact that they must fit into the user's data area. Any existing level of a variable can be referenced. Variables can be created, assigned values, and destroyed either interactively or from within a TACL program. By default, all variables are global within a TACL program, although it is possible to create local variables for a particular procedure.

With TACL, variables can be altered while the process is running so that concurrent processes can be managed flexibly.

The method TACL uses to substitute a variable's value for its name is somewhat different from that of most languages. The value that is substituted for the variable name depends on its type: alias, text, macro, routine, or delta. Text, macro, and routine variables can contain commands or function invocations as well as text; they then can be viewed as procedures.

An *alias* variable is used as another name for a word. The word may be the name of a built-in function, TACLBASE command, user-defined command, or file name. When the variable name is used, the word for which it is an alias is substituted.

Text, the default, is the type most similar to types found in other languages. The value of a text variable is simply the value of the text to which the variable is set or defined. It can be numeric or character.

Like a text variable, a *macro* variable can contain numeric or character text. Macros allow the substitution of dummy arguments based on the positions of their actual arguments. A macro variable's value then becomes its textual content plus the argument substitutions. (Macros are most often used to execute other commands, however, in the same way that routines are used.)

A *routine* variable is used to execute other commands, much like a procedure is used in other languages. It determines its own value through calls to the built-in function #RESULT. A routine has the option of not returning a value.

Delta variables are discussed in the section, "TACL Text Editor (#DELTA)."

TACL variables can be used in ways not commonly offered in other programming languages. For example, variables can be used as input and output files to one or more other processes. They can be used strictly as files; that is, the entire input contents are contained in the input variable when the process is started and the output variable is examined when the process' output is complete.

These variables can also be altered dynamically, however; their contents can be changed while the process is running. This provides great flexibility in managing one or more concurrent processes. TACL variables can also be used easily to perform sequential I/O to files, as elaborated upon in the section, "Sequential I/O."

Procedural Constructs

Text, macro, and routine TACL variables can be used as procedures because they themselves can contain command or function invocations. All three can be constructed with functions of any type as building blocks; each can be used to fill particular needs.

Text functions, the simplest of the three to build and use, can contain collections of commands and/or built-in functions that are invoked as a unit; thus the name of the function becomes a shorthand notation for the sequence of functions within it. The invocation of a text function produces only those results generated by its constituents.

A macro can contain different functions or function arguments every time it is invoked, through the argument-substitution scheme mentioned in the previous section. The results of a macro invocation consist of the results of the functions contained within it, after the actual arguments have replaced the dummy arguments.

Routines are the most versatile of the procedure types in TACL. Users can vary the contents of a routine (in the same way they can vary the content of a macro) by having TACL pass arguments to it. In addition, TACL can check routine arguments automatically for syntactic (and in some cases, semantic) correctness through the built-in function #ARGUMENT. Note that the writer of the routine determines syntactic correctness; the syntax rules for routine arguments need not be identical to those for TACL commands.

The writer of the routine also has complete control over the values (or results) produced by the invocation of the routine. If needed, values must be generated explicitly with the built-in function #RESULT.

Control Structures

All of TACL's procedural function types can make use of the control built-in functions to perform different actions, based on the values of control expressions (which can be the results of function invocations).

TACL supports block IF-THEN-ELSE statements, labeled CASEs, and two loop types (WHILE-DO and DO-UNTIL). Recursion is also possible.

Routines alone can be programmed to #RETURN at any time to their invoking function (or simply terminate, if they were invoked directly from the keyboard).

Exception Handling

Another facility unique to routines is the ability to detect and handle exception conditions programmatically. Exceptions are generated ("raised") by TACL or by functions when an unexpected event prevents normal processing. Routines may cause (#RAISE) exceptions at any time they are being executed.

If a routine needs to handle exceptions itself (and this can include exceptions raised by it or by any routine it invokes), it uses the #FILTER built-in function to name the exceptions for which it will accept responsibility.

When an exception is raised, TACL ceases invoking the current routine and checks whether the exception is filtered by it. If not, TACL cancels execution of the routine that invoked the current one (if there is such a routine) and checks that routine's filters. It passes the exception "up" the chain of routine invocations in this manner until it finds a filtering routine.

When TACL finds the routine that filters the desired exception, it reinvokes that routine. The routine must use #EXCEPTION to find out whether it is being invoked normally or as the result of its filtering a raised exception.

In some instances, TACL handles exceptions for users. One frequently encountered exception is called _ERROR, which is raised when TACL detects an error. One cause of a raised _ERROR is the attempt to invoke a routine with arguments not recognized as correct by its #ARGUMENT processing. If such an argument error occurs when no routine has declared it will handle _ERROR problems, TACL responds by returning the error message "expecting..." followed by a list of the argument types expected.

Debugging Facility

TACL supplies a debugging facility for TACL code. Users have the option of stepping line by line or setting breakpoints at an invocation. The debugger itself is written in TACL code and resides in TACLBASE. Users may create a modified version by copying it from TACLBASE, making the desired changes, and LOADing it into their individual environments. Any TACL command can be evaluated at a debugger prompt; thus, users can obtain information about the TACL process or any of its variables.

Sequential I/O

TACL functions can pass information to and use information from other processes, devices, and files of all types, including edit files. The #INPUT (and #INPUTV) and #OUTPUT (and #OUTPUTV) functions can be used to read from and write to the IN and OUT files of a TACL process. The function #REQUESTER can be used to open other files for reading or writing, perform the operations, and close the files. The function #REQUESTER I/O can occur asynchronously; that is, other functions can be invoked while TACL completes the I/O. It is also possible to wait for the operation to finish.

The TACL variables can be altered while the process is running so that concurrent processes can be managed flexibly.

Figure 2

```
?SECTION program_timer ROUTINE
#FRAME

#PUSH program_in          {Set up variables for controlling}
#PUSH program_status      {the process to be timed.}

#PUSH source_file
#SET source_file smallobj

#PUSH inspect_in          {Set up variables for controlling INSPECT process.}
#PUSH inspect_out
#PUSH inspect_server

[#SET inspect_in &
b#bo^build^object^file + %247
b#bo^build^object^file + %434
b#bo^build^object^file + %461
b#bo^build^object^file + %471
b#bo^build^object^file + %504
resume
]

#PUSH breakpoint_name     {Set up variables for data relating}
#SET breakpoint_name Start    {to timing and breakpoints.}

#PUSH start_time, stop_time, elapsed_time

= = Get a server file name to use for INSPECT
= = (the TERM of the timed process)
#SET inspect_server [#SERVER /IN inspect_in, OUT inspect_out/]

= = Start the process to be timed under INSPECT
RUND $system.system.bind/STATUS program_status, NOWAIT, &
  INV program_in DYNAMIC, OUT $s.#temp, &
  TERM [inspect_server]/

= = Wait for it to be started up (ready for input)
sink [#WAIT program_in]

= = Send it the commands to start work
#APPEND program_in add * from [source_file]
#APPEND program_in build testobj
#EOF      program_in
= = Send it an end of file

#SET start_time [#timestamp]      {Record the starting time.}

= = Print headers, first "breakpoint" name (Start)
#OUTPUT

#OUTPUT /COLUMN 5, HOLD/          breakpoint
#OUTPUT /COLUMN 40, HOLD/         start
#OUTPUT /COLUMN 55, HOLD/         end
#OUTPUT /COLUMN 70/               elapsed

#OUTPUT /COLUMN 5, HOLD/          name
#OUTPUT /COLUMN 40, HOLD/         time
#OUTPUT /COLUMN 55, HOLD/         time
#OUTPUT /COLUMN 70/               time

#OUTPUT

#OUTPUTV/COLUMN 5, HOLD/ breakpoint_name

= = Begin loop which waits for breakpoints to be hit and
= = records the time spent between each pair
wait_for_breakpoints

#UNFRAME


?SECTION print_breakpoint_times TEXT
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = Function to print the breakpoints encountered and the
= = time elapsed between each pair of breakpoints
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

#SET elapsed_time [#COMPUTE stop_time - start_time]

= = Print elapsed time for code AFTER last breakpoint,
= = then name of the breakpoint we just hit
#OUTPUT /COLUMN 40, HOLD/[contime_to_text_time
    [#contime [start_time]]]
#OUTPUT /COLUMN 55, HOLD/[contime_to_text_time
    [#contime [stop_time]]]
#OUTPUT /COLUMN 70/ [contime_to_text_time
    [#contime [elapsed_time]]]

#OUTPUTV/COLUMN 5, HOLD/ breakpoint_name
```

Figure 2.
PROGRAM_TIMER can be used to make elapsed-time performance measurements on processes whose actions are controlled through INSPECT breakpoints. This version of the routine was used to measure BIND performance. The routine's implicit #SERVER is the IN file of the process (supplying commands to it). The explicit server-file name is used as the TERM of the process so that INSPECT's input and output can be manipulated. PROGRAM_TIMER also has a subfunction (get_breakpoint_routine) that uses #DELTA to extract information from INSPECT's output.

GUARDIAN 90 Interface Functions

TACL implements many of the most useful GUARDIAN 90 procedures as built-in functions. These functions are easy to use, as TACL converts the values supplied as input (numeric or plain text) to the proper formats, fills in required parameters, calls GUARDIAN 90, and converts the returned values to text. On GUARDIAN 90 calls that return multiple values (FILEINFO, for example), TACL obtains only the items users specify. Some of the GUARDIAN 90 procedures TACL supports are shown in Table 1.

Using Variables for Process I/O

When users start a process with TACL (by explicit or implicit RUN, or by using #NEWPROCESS), the process can use TACL variables as its IN, OUT, and/or TERM files. In addition, if the program uses logical file names, users can direct the program to use TACL variables in place of other physical files, as well. This is the server-file feature of TACL.

When server files are used, TACL can determine the contents of the process' input variable programmatically while the process is running, and it can examine the contents of the process' output variable at any time; that is, TACL is in complete control of the process.

An example of the use of both explicit and implicit server files to control a program running under INSPECT is shown in Figure 2. Figure 3 demonstrates how a program that gets its file location by reading an ASSIGN message might use a TACL variable instead of a disc file.

```
?SECTION wait_for_breakpoints TEXT
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = Recursive function that waits for breakpoints to be hit and
= = reports on them
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

= = Wait for INSPECT to be ready for input (meaning a
= = breakpoint has been hit) OR for the program to
= = stop
[#CASE [#VARIABLEINFO /VARIABLE/
    [#WAIT inspect_in program_status]]

|inspect_in|
    #SET stop_time [#timestamp]

    = = get breakpoint location from INSPECT output
    = = and print statistics
    get_breakpoint_name
    print_breakpoint_times

    = = throw away INSPECT output
    #set inspect_out

    = = resume execution and wait for next breakpoint
    #SET start_time [#timestamp]
    #APPEND inspect_IN resume

    wait_for_breakpoints

|program_status|
    #SET stop_time [#timestamp]

    #SET breakpoint_name Stop          {Print statistics on}
    print_breakpoint_times              {last breakpoint.}

    #OUTPUT
    #OUTPUT Killing server [inspect_server]: &
        [#SERVER /KILL/ [inspect_server]]      {Delete the server}
                                               {file for INSPECT.}

] {case}
```

```
?SECTION get_breakpoint_name ROUTINE
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = #DELTA commands to extract the breakpoint location
= = from the INSPECT output
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

#FRAME
#PUSH delta_commands
#SET /TYPE DELTA/delta_commands &
Ginspect_out$ &                 = = Get the text from inspect_out
0J &                            = = Go to the beginning
:S-BREAKPOINT-$ &               = = Search for -BREAKPOINT-
?N Xbreakpoint_name$' &         = = If found, put the rest of the text
HK                              = = into breakpoint_name,
                                = = clear the buffer.

SINK [#DELTA /COMMANDS delta_commands/]

#UNFRAME
```

Table 1.
GUARDIAN 90 procedures and the equivalent TACL built-in functions.

GUARDIAN 90 procedure	TACL built-in function
File system	
create (partially supported)	#createfile
deviceinfo	#deviceinfo
fileinfo	#fileinfo
nextfilename	#nextfilename
processfilesecurity	#processfilesecurity
rename	#rename
open, writeread/read, and close	READ #requester
open, write, and close	WRITE #requester
open and close	#in
writeread	#input, #inputv
open and close	#out
write	#output, #outputv
Process control	
activateprocess	#activateprocess
alterpriority	#alterpriority
createprocessname	#createprocessname
createremotename	#createremotename
debugprocess	#debugprocess
lookupprocessname or getppdentry	#lookupprocess
newprocess or newprocessnowait	#newprocess
processinfo	#processinfo
stop	#stop
Process control (continued)	
suspendprocess	#suspendprocess
TMF	
aborttransaction	#aborttransaction
begintransaction	#begintransaction
endtransaction	#endtransaction
Other	
contime	#contime
convertprocesstime	#convertprocesstime
locatesystem (number only)	#systemnumber
mom	#mom
mypid	#mypid
mysystemnumber plus getsystemname	#mysystem
myterm plus setmyterm	#myterm
setmode 28	#initterm
shiftstring	#shiftstring
getsystemname	#systemname
timestamp	#timestamp
tosversion	#tosversion
usernametouserid	#userid
useridtousername	#username
verifyuser (log on only)	#changeuser

Figure 3

```
?TACL MACRO

#FRAME

#PUSH prog_name, prog_stat, data_var        {Create variables to run the program.}
#SET / IN datafile / data_var               {Load data into the data variable.}

= = Set up variables to control the servers simulating logical files
#PUSH server_in_name, server_out_name, server_in, server_out

= = Get server file names for 2 logical files, the program will
= = use one as an input file and the other as an output file
#SET server_in_name      [#SERVER / IN server_in / ]
#SET server_out_name     [#SERVER / OUT server_out / ]

= = Save a copy of the current ASSIGNs, then assign the logical
= = files to the server names
#PUSH #ASSIGN
ASSIGN ft005, [server_in_name]
ASSIGN ft006, [server_out_name]

= = Run the program using a status variable to tell when it finishes
p1foro/ STATUS prog_stat, nowait /

= = Append the contents of the data file, now contained in data_var,
= = to the IN variable of the server simulating the logical file being
= = used for input. This allows the program access to the data.
#APPENDV server_in data_var

= = When all the data has been read from the IN variable (server_in) or
= = the program terminates (prog_stat), we can examine the OUT variable
= = (server_out), to see the results of the program.
SINK [#WAIT server_in prog_stat]

#OUTPUT
#OUTPUT The results of the program are:
OUTVAR server_out

= = Stop the servers which were running and #POP the current logical file
= = assigns. These commands and the #UNFRAME will leave the environment
= = as it originally was.

SINK [#SERVER/ KILL / [server_in_name]]
SINK [#SERVER/ KILL / [server_out_name]]

#POP     #ASSIGN

#UNFRAME
```

Figure 3.

This macro demonstrates two TACL server files simulating I/O files for a FORTRAN program. The server names are passed to the program with ASSIGNs. The IN server-file variable is filled with the data for the program, and the OUT variable receives the program's output.

TACL Text Editor (#DELTA)

Although it is possible to use the #ARGUMENT feature of routines to interpret textual argument strings, TACL has a much more powerful text manipulation facility called #DELTA. This facility is a programmable text editor that allows the use of IFs, loops, and macros. It can read and write TACL variables as well as files of all types (using sequential I/O). The usual editing functions, such as insert, delete, and search, are also supported, along with upshifting and downshifting.

The #DELTA facility can be used interactively or as a low-level tool in the creation of higher-level multipurpose (or specialized) text editing functions. (The latter is done by storing #DELTA commands in a variable of type DELTA.)

Figure 2 contains a routine using #DELTA (get_breakpoint_name) which extracts a breakpoint name from INSPECT process output and places it in a variable for use in other routines.

Conclusion

TACL has many features that make it well suited for implementing complex procedures, especially those requiring process control or access to the GUARDIAN 90 operating system. While it is not a replacement for compiled languages, as an interpreted high-level language it is ideal for quick prototyping. For the development of applications whose performance is not critical but whose flexibility is (such as a command processor), TACL provides a complete solution.

Julia Campbell has worked in Tandem's Languages and Tools Quality Assurance Group for two years, supporting PATHWAY, the Product Development Tools (PDT), the FORTRAN compiler, and TACL. Before working in Software Development, Julia worked in Tandem's Manufacturing MIS Group as a programmer/analyst for the PATHWAY application EMPACT.

Robin Glascock joined Tandem in 1983 as a member of the Languages and Tools Quality Assurance Group of Software Development. Since then she has been responsible for the QA and performance evaluation of several products, including TACL. She has recently moved into the Work Group Software Quality Assurance project, where she is writing tools to facilitate the testing of screen-based interactive software. Robin spent four years in software development at other companies before coming to Tandem.

In the first calendar quarter of 1986, Tandem will release a new COBOL compiler and run-time library called COBOL85. COBOL85 will not immediately replace the current COBOL compiler and run-time library (referred to in this article as COBOL74). Both products will be available for the next few years, after which COBOL74 will gradually be phased out.

COBOL85 runs only on the GUARDIAN 90 operating system. It is based on the new American National Standards Institute (ANSI) COBOL 1985 standard. It supports all of the required modules in the revised American National Standard Programming Language COBOL, X3.23-1985, and has extensions to provide access to standard Tandem facilities.

COBOL85 supports the following ANSI standard modules: nucleus, table-handling, sequential I/O, relative I/O, indexed I/O, sort/merge, interprogram communication, and source text manipulation. Level 1 of the optional debug module (which allows paragraph traces) is also supported by COBOL85. Two optional modules of the ANSI standard have not been implemented: report writer and communications. The segmentation module is almost entirely implemented.

COBOL85 and the New Standard

The new COBOL standard has been in the making for some time. The previous standard was approved in 1974; work on the new one began in 1978, and it was approved in September 1985. Most of the problems in completing the new standard had to do with its incompatibilities with the previous standard. Several review cycles were needed to resolve the problems, and there are still several areas in which the two are incompatible.

Most COBOL programmers feel that the changes are necessary, however, and that they will cause few (if any) conversion problems. This is especially true for Tandem COBOL, since Tandem implemented COBOL74 in a logical fashion in the main areas affected by the changes. Also, Tandem tended to follow the clarified specifications as they were placed in the CODASYL COBOL Committee Journal of Development (JOD). Unfortunately, some other implementors did not, and almost all of the complaints came from their users. If Tandem COBOL users have any conversion problems at all, they should be minor ones.

One might well ask why there should be a new standard. The answer lies in the simple fact that COBOL has existed for 25 years. As a result, it lacks many of the aids for "structured programming" that other languages have. This has caused maintenance nightmares and long development times for applications written in COBOL.

The new standard includes most of these missing facilities, which aid in program design, implementation, and maintenance. Digital Equipment Corporation (DEC) and Control Data Corporation (CDC) presently offer compilers containing many of the new features, and some of their users estimate up to a 50% reduction in implementation time and maintenance costs. A cost/benefit study published by the U.S. Department of Commerce (NBSIR 83-2639) indicates that the federal government could save approximately $90 million over a ten-year period by adopting the new standard, primarily because of such reductions.

In the 1974 standard, there were many undefined areas and rarely used features. In the new standard, the undefined areas have been defined and the rarely used features made obsolete (although not deleted). These obsolete features will be deleted when the next standard is completed (in the 1990s). COBOL85 flags obsolete features upon request.

New Features in the COBOL 1985 Standard

The most important changes are those commonly called "the structured programming features." These comprise:

- Explicit scope terminators.
- NOT options for the "one-legged" branches, such as AT END.
- In-line PERFORM.
- Nested programs.
- The EVALUATE statement.

Explicit scope terminators are reserved words that can be used to terminate conditional statements. There is one for every such statement. The form is END-verb, where "verb" is IF, ADD, READ, and so on. When an explicit terminator is specified, the statement becomes an imperative statement and can be used anywhere an imperative statement can be used. The following example illustrates the use of explicit scope terminators (and two other minor new features):

```
IF Action - "Delete" THEN
  DELETE Trans-file RECORD
    INVALID KEY
      CALL Inv-key-process
    NOT INVALID KEY
      SET Some-deleted TO TRUE
      ADD 1 TO Records-deleted
  END-DELETE
END-IF
```

The explicit scope terminators in this example are END-DELETE and END-IF.

Note also that no periods are used to terminate the conditional statements. One of the biggest problems with COBOL has been the period terminator. It is hard to see, it terminates everything, and it is a source of many program bugs. If a period were inserted after "ADD 1 TO Records-deleted," COBOL74 would terminate the IF and DELETE and cause a syntax error. The only periods necessary in the Procedure Division in COBOL85, however, are after section and paragraph headers and at the end of a paragraph.

The previous example also illustrates the use of a new NOT branch that has been provided for phrases such as SIZE ERROR, INVALID KEY, and AT END. (These were formerly one-legged branches, but now, in each case, a NOT branch is available.) Also illustrated is the use of the optional word THEN after the condition in the IF statement, and the use of SET to set the conditional variable associated with a condition-name to a value that makes the condition-name true. In the example, "SET Some-deleted TO TRUE" moves the value that makes "Some-deleted" true to the associated conditional variable.

The in-line PERFORM is similar to a "DO loop" in other languages. An example is:

```
PERFORM WITH TEST AFTER
   VARYING I1 FROM 1 BY 1 UNTIL I1 = I2
        ADD 1 TO Counter-1
        CALL Something
END-PERFORM
```

Obviously, this is much easier than creating a paragraph to contain the performed code. Note the TEST AFTER phrase. This indicates that the loop test is to take place after the loop. The default is before the loop (the COBOL74 method), and the words TEST BEFORE are available if the programmer wants to be more explicit.

A nested program is one that is embedded in some other program. Other languages have offered this facility for years, and now COBOL does too. Nested programs enable the programmer to structure the task easily. They are superior to performed paragraphs since the programmer can prevent unwanted side effects such as the changing of a variable that was not meant to be changed. A paragraph can reference everything in the Data Division of the performing program. A nested program cannot.

A simplified example of a nested program follows:

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. Containing-Program.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*   Note that the following has a global
*   name. It can be referenced in a
*   contained program.
 01   F1   GLOBAL   PIC XXX.
*   The following does not have a global
*   name. It cannot be referenced in a
*   contained program.
 01   F2   PIC XXXXX.
 PROCEDURE DIVISION.
 STARTT.
        CALL Contained-1.
        STOP RUN.

 IDENTIFICATION DIVISION.
 PROGRAM-ID. Contained-1.
 WORKING-STORAGE SECTION.
 01   An-item   PIC 99.
 PROCEDURE DIVISION.
 STARTT.
        MOVE "xxx" TO F1.
 EXITT.
        EXIT PROGRAM.
 END PROGRAM Contained-1.

 END PROGRAM Containing-Program.
```

This example illustrates the use of a GLOBAL name. If GLOBAL is not specified for a name, it cannot be referenced in a contained program. Thus, that data can be protected.

Note that the structure is top-down, not bottom-up as it is in other languages, such as Pascal and Ada. This makes for easier reading and construction.

Finally, note that no ENVIRONMENT DIVISION is needed in any program, contained or containing. The nesting limit is seven, but using more than two or three levels is unpractical.

The new EVALUATE statement allows the testing of one or more items and the selection of different paths depending on various criteria. It is similar to the CASE statement in other languages, but is much more powerful. By using EVALUATE, the programmer can avoid very complex nested IF statements. An example of the EVALUATE statement is shown in Figure 1.

The example illustrates the use of two "selection subjects." The first is the conditional expression "Balance NEGATIVE," and the second is the data name "Customer-type." Each WHEN phrase must contain the same number of "selection objects" as there are selection subjects, and these objects are paired with the subjects positionally. In the example, the selection objects in the first WHEN phrase are the truth condition FALSE and the match-anything word ANY.

Figure 1

```
EVALUATE Balance NEGATIVE   ALSO   Customer-type
      WHEN     FALSE          ALSO   ANY
          CONTINUE
      WHEN     TRUE           ALSO   Preferred
          PERFORM Dunn-preferred-customer
      WHEN     TRUE           ALSO   Always-late
          PERFORM Dunn-late-customer
      WHEN OTHER
                DISPLAY "Error"
                GO TO Abort-run
END-EVALUATE
```

Figure 1.
An example of the EVALUATE statement that uses two selection subjects.

The first WHEN phrase is selected if the data item referenced by Balance is positive or zero and the data item referenced by "Customer-type" has any value. The word FALSE indicates that the corresponding selection subject must evaluate to a false condition. The word ANY indicates that the corresponding selection subject is ignored; that is, any value at all is considered to match. Note that the action taken is null. The word CONTINUE is a no-op instruction. The execution continues after END-EVALUATE.

The second WHEN phrase is selected if the data item referenced by "Balance" is negative and the value of "Customer-type" is the value "Preferred." "Preferred" can be another data item, or it can be a constant defined with the REPLACE statement (for example, "REPLACE == Preferred == BY == 1 ==.").

The third WHEN phrase selection is similar to the second. If no WHEN phrase is selected, the WHEN OTHER phrase is selected.

An equivalent IF statement in COBOL74 would be:

```
IF Balance NOT NEGATIVE
  NEXT SENTENCE
ELSE
  IF Balance NEGATIVE
   AND Customer-type = 1
     PERFORM Dunn-preferred-customer
  ELSE
     IF Balance NEGATIVE
      AND Customer-type = 2
        PERFORM Dunn-late-customer
     ELSE
        DISPLAY "Error"
        GO TO Abort-run.
```

Note that this is harder to read than the EVALUATE statement. When many WHEN phrases and selection subjects and objects are used, the equivalent nested IF becomes quite complex. From 1 to 255 selection subjects and a corresponding number of selection objects can be used.

Some of the other major changes in COBOL85 are summarized below:

- INSPECT CONVERTING enables the programmer to convert one character string to another. This feature is commonly used to convert lowercase to uppercase.
- Reference modification (commonly referred to as substring or byte slicing in other languages) allows the programmer to reference a part of a data item. Although it can be misused, reference modification can be a very powerful and useful feature.
- External files and data enable the programmer to share data and files among programs without passing the files or data as parameters.
- CALL has been enhanced to allow the passing of any elementary item as a parameter and to allow the protection of parameters by specifying that they are passed by content.
- The INITIALIZE statement allows the programmer to set items to predefined values. For example, by referencing a group item, the programmer can set each elementary item to an appropriate predefined value.
- Variable-length records can be written under explicit control, and the length can be determined when the record is read.
- The REPLACE statement allows the programmer to replace one or more words with another. This feature is often used to define constants, such as the length of a table.

Incompatibilities

In the detailed discussion below, COBOL85's incompatibilities with COBOL74 are grouped according to whether they:

- Are likely to cause problems.
- May cause problems.
- Are unlikely to cause problems.

In each case, first the incompatibility is described. Then an action is recommended to help programmers avoid future problems caused by that incompatibility when they write COBOL74 programs.

Incompatibilities Likely to Cause Problems

1. COBOL85 has 49 new reserved words.
2. SEARCH ALL now does a binary search.
3. Many new I-O status codes have been added.
4. Numeric exceptions may abort a run.
5. Arithmetic results may differ (because of greater precision in COBOL85).
6. Short records on fixed-length files do not abort a run.
7. Subscript evaluation differs in STRING and UNSTRING.
8. Multiple source programs in a compiler input file now require terminators.
9. OPEN I-O or EXTEND on a nonexisting file does not create the file.

New Reserved Words. The 49 new reserved words are listed at the right. Of them, 19 are END-xxx statements (where xxx is a verb like IF) and the rest are other words. TEST, ANY, TRUE, and FALSE are probably the most likely to cause problems. Diagnostics are given when these words are misused, but the diagnostics may be confusing (since incorrect syntax is being diagnosed).
Action. This incompatibility has proven to be a minor problem. Avoid using the new words. When the transfer to COBOL85 is made, the REPLACE statement can be used to help alleviate the problem. Also, several conversion programs should be available from various software vendors to change reserved words (and make other changes) automatically.

SEARCH ALL. COBOL74 does a serial search, so an item may be found even if the items in the table are in incorrect order. A compatibility warning diagnostic is provided. Also, if the standard SEARCH ALL rules are not followed in the syntax of the statement, a serial search is done.
Action. Make sure that the table is in order and that all rules are followed for SEARCH ALL.

New I-O Status Codes. COBOL74 produces status codes "00," "30," "90," and "91," instead of the new codes. Two situations are most likely to cause problems:

- Opening an optional file that is not present produces I-O status code 05 when status code 00 was produced in COBOL74. (Also, for an optional file opened for I-O or EXTEND, I-O status code 05 is returned by COBOL85 if the file was created.)

- Executing an OPEN or CLOSE statement with options such as NO REWIND, REEL/UNIT, or FOR REMOVAL for a device that does not support the options results in status code 07 rather than status code 00.

In both cases, the operation is successful. A list of the differences is provided in the *COBOL85 Reference Manual*. No diagnostics can be provided.
Action. Take care in testing for specific codes of 00, 30, 90, and 91.

New reserved words in COBOL85.

ALPHABET	END-DIVIDE	FALSE
ALPHABETIC-LOWER	END-EVALUATE	GLOBAL
ALPHABETIC-UPPER	END-IF	INITIALIZE
ALPHANUMERIC	END-MULTIPLY	NUMERIC-EDITED
ALPHANUMERIC-EDITED	END-PERFORM	ORDER
ANY	END-READ	OTHER
BINARY	END-RECEIVE	PACKED-DECIMAL
CLASS	END-RETURN	PADDING
COMMON	END-REWRITE	PURGE
CONTENT	END-SEARCH	REFERENCE
CONTINUE	END-START	REPLACE
CONVERTING	END-STRING	STANDARD-2
DAY-OF-WEEK	END-SUBTRACT	TEST
END-ADD	END-UNSTRING	THEN
END-CALL	END-WRITE	TRUE
END-COMPUTE	EVALUATE	
END-DELETE	EXTERNAL	

Numeric Exceptions. Numeric exceptions (such as an arithmetic overflow) may abort a run if SIZE ERROR is not specified. COBOL74 does not detect these, and thus, incorrect values may be produced. Also, invalid data in a numeric data item (such as being initialized to spaces) may cause an abort. COBOL74 would process the bad data, giving undefined results (spaces would be treated as zeros, however). No diagnostics can be provided.

COBOL85 ensures more accuracy for arithmetic operations than does COBOL74, and it never truncates digits from the left.

Action. Use care in calculations, and use SIZE ERROR when exceptions are possible. Make sure numeric data items are initialized correctly.

Arithmetic Results. Arithmetic results may differ. COBOL74 does not produce as many digits to the right as COBOL85 does, and it sometimes truncates significant digits from the left without any indication, if SIZE ERROR is not specified.

COBOL85 ensures more accuracy for arithmetic operations, and it never truncates digits from the left. Note that a run may be aborted as the result of an arithmetic overflow condition in conditions that previously resulted in left truncation or right zero padding. No diagnostics can be provided.

Action. If the accuracy in an arithmetic expression is questionable (especially for division), use individual ADD, SUBTRACT, MULTIPLY, and DIVIDE statements to control the accuracy. Use the SIZE ERROR clause to detect any possible left truncation, if necessary. (Note that it involves more overhead. Note also that exponentiation with fractional exponents, e.g., 0.5 for a square root, exists in COBOL85.)

The READ Statement. READ allows short records to be read from fixed-length files, while COBOL74 aborts a run if it encounters these. No compiler diagnostic can be provided, but a file status value is defined for such an operation.

Action. None can be taken. Do not assume the run will be aborted if a file with short records is read. COBOL85 allows a check for file status "04." COBOL74 programs can be modified at any time to check for status code "04," since it has no effect until the program is run on COBOL85. In any case, there should never be such records.

UNSTRING and STRING. UNSTRING and STRING evaluate all subscripts at the start of a statement. COBOL74 defers some subscript evaluations until before their use. A compatibility warning diagnostic is provided.

Action. Do not rely on the deferment of the evaluation. In general, using values changed during the execution of a statement as subscripts within the statement (except in SEARCH and PERFORM) is poor programming practice.

Multiple Source Programs Per Compilation. Multiple source programs in one compiler input file that are not separated by ?ENDUNIT directives are perceived differently by the two compilers. COBOL74 views them as separately compiled programs. COBOL85 assumes them to be nested within the first program. Since the END PROGRAM headers are not there, a diagnostic is produced. Also, other diagnostics may be produced for constructs banned from contained programs.

Action. Place an ?ENDUNIT directive after each program. This is a good practice for any COBOL74 program or COBOL85 program that is separately compiled. In COBOL85, do not place an ?ENDUNIT directive in front of any contained program, since the directive terminates all nesting. There are no contained programs in COBOL74, so using ?ENDUNIT directives does not cause problems.

OPEN I-O or EXTEND on a Nonexistent File. OPEN I-O or EXTEND on a nonexistent file results in an unsuccessful open if OPTIONAL is not specified in the SELECT clause. COBOL74 creates the file and does not allow OPTIONAL in the SELECT clause for indexed or relative files. COBOL85 creates the file if OPTIONAL is specified. No diagnostic can be provided.
Action. Add OPTIONAL to the SELECT clause for the file, if it is sequential. For indexed and relative files this cannot be done in COBOL74, so it will have to be added when the program is converted to COBOL85.

Incompatibilities That May Cause Problems

1. A store to a group with an OCCURS DEPENDING ON differs, based on whether or not the "depending-on" item is in that group.
2. ALPHABET should appear in front of an alphabet clause.
3. The initialization order of multiple VARYING identifiers in PERFORM differs.

OCCURS DEPENDING ON. A store to a group with an OCCURS DEPENDING ON (ODO) uses the maximum size if the group contains the depending-on item, and it uses the specified size if the group does not contain that item. COBOL74 uses the maximum size except in UNSTRING. A compatibility warning diagnostic is provided.
Action. Do not use a group containing an OCCURS DEPENDING ON as a receiver in UNSTRING. This operation would not be useful and would be very misleading to a maintenance programmer. If it is used, make sure the depending-on item has the maximum value before UNSTRING is executed. Also, do not assume that any items with subscripts greater than the resulting value in the depending-on item contain useful information.

ALPHABET. The word ALPHABET should appear before an alphabet clause. This requires manual conversion, since COBOL74 doesn't recognize ALPHABET. Although the standard requires ALPHABET in all instances, COBOL85 requires it only when "ALPHABET alphabet-name IS system-name" is specified. (Currently, the only system-name in COBOL85 is EBCDIC. NATIVE, STANDARD-1, and STANDARD-2 are not system-names.) A diagnostic is given for "alphabet-name IS EBCDIC" if ALPHABET does not precede it.
Action. Since COBOL85 accepts all constructs that are legal in COBOL74, no action is necessary. It is recommended, however, that ALPHABET be inserted in such clauses when programs are converted, in order to make them compatible with the standard.

Multiple VARYING Identifiers in PERFORM. The initialization order of multiple VARYING identifiers in PERFORM has changed. This only affects a program using such an identifier in a FROM or BY phrase (e.g., PERFORM P1 VARYING X FROM 1 BY 1 UNTIL X = 3 AFTER Y from X BY 1 UNTIL Y = 3). A compatibility warning diagnostic is provided.
Action. Do not use constructs like this. It is poor programming practice, and the results are nonobvious.

Incompatibilities Unlikely to Cause Problems

1. "ALL literal" produces different results if associated with a numeric item.
2. A figurative constant is not allowed in the CURRENCY SIGN clause.
3. "P" is not allowed in PIC strings for a relative key.
4. LINAGE cannot be specified for a file opened with EXTEND.
5. CLOSE REEL/UNIT WITH NO REWIND is no longer legal.
6. Changes have been made in READ or RETURN INTO.
7. ADVANCING PAGE and AT EOP are not allowed in the same WRITE statement.
8. Independent segments have been deleted.
9. An index data item is four bytes rather than two.
10. ON OVERFLOW in a CALL is taken if the program cannot be found.
11. The size of LINAGE-COUNTER has changed from PIC 9(5) to PIC 9(4).

ALL Literal. "ALL literal," as in

ALL "9"

may produce different results when associated with a numeric or numeric-edited data item. In COBOL85, the literal is repeated; in COBOL74 it is not. In COBOL74,

MOVE ALL "9" TO PIC 99V9

produces 09.0. Some implementors produce 99.9 and some 99.0. COBOL85 produces 99.0. A compatibility warning diagnostic is provided.
Action. Don't use "ALL literal" with such items. It is misleading and of no use. Also, it is an obsolete item and will be deleted from the next standard.

Figurative Constants in CURRENCY SIGN Clauses. A figurative constant is not allowed in a CURRENCY SIGN clause. For example,

CURRENCY SIGN IS ALL "L"

is invalid, and a diagnostic is given.
Action. Do not use this type of construct. Since ALL means nothing in this context, it is confusing and redundant.

"P" in PIC Strings. "P" is not allowed in PIC strings for a relative key data item (e.g., PIC 99PP to access every 100th record). A diagnostic is given.
Action. Do not use a construct of this sort. It is misleading, and the results are not defined.

LINAGE Clause. The LINAGE clause cannot be specified for a file opened with EXTEND. A diagnostic is given.
Action. Do not use LINAGE for files opened with EXTEND. The results are undefined and not what one would expect.

CLOSE REEL/UNIT WITH NO REWIND. This construct is not allowed. A diagnostic is given.
Action. Do not use this construct. In COBOL74, it leaves the reel at the end during a reel swap, requiring the operator to rewind the reel manually. This makes no sense and is an extra burden on the operator.

INTO Phrase in READ and RETURN. READ and RETURN now allow an INTO phrase if only one record description is subordinate to the file-description entry, or if all subordinate record-description entries are alphanumeric or group entries and the INTO item is also an alphanumeric or group entry. For example, multiple record descriptions with an edited INTO item are no longer allowed. A diagnostic is given.
Action. Do not use the INTO phrase in such instances. The results are not what would be expected, anyway, as no editing or scaling takes place.

ADVANCING PAGE and EOP with WRITE. WRITE no longer allows ADVANCING PAGE and AT EOP in the same statement. COBOL74 always takes the EOP. A diagnostic is given.
Action. Do not use this construct. Since the EOP is always executed, the AT EOP phrase is redundant.

Independent Segments. Independent segments have been deleted. This affects only the targets of ALTER statements. A diagnostic is produced for ALTER statements that reference paragraphs in independent segments.
Action. Do not use ALTER. It is extremely poor programming practice to do so. It is obsolete and will be deleted from the next standard, as will segmentation.

Index Data Items. An index data item is now 4 bytes rather than 2. A compatibility warning diagnostic is provided.
Action. Do not use index data items. (They are defined by USAGE IS INDEX.) Such items are not useful and can easily cause nonobvious bugs. Note that an index defined by the INDEXED BY phrase within an OCCURS clause is not the same as an index data item. No compatibility problem exists for indexes.

ON OVERFLOW/EXCEPTION. The ON OVERFLOW/EXCEPTION branch is taken if a CALL identifier references a program that cannot be found. COBOL74 aborts the run. A compatibility warning diagnostic is provided.
Action. Do not use the ON OVERFLOW phrase, since the conditional code is never executed. If it is used, for complete compatibility, specify STOP RUN along with it (or some other means to abort the run in the conditional code). When COBOL85 is used, the code is executed in the indicated case.

PICTURE for LINAGE-COUNTER. The implied PICTURE for LINAGE-COUNTER has changed from 9(5) to 9(4). The maximum allowable number is now 9999 rather than the previous value of 32767. No diagnostic is given.
Action. Make sure the LINAGE value specified in the FD does not exceed 9999. Since any numbers greater than 66 or so make little or no sense, it is doubtful that a problem will occur.

Conclusion

COBOL85 will help to reduce the development and maintenance costs associated with COBOL programming. The new features are not hard to learn, better programs will result from their use, and conversion from COBOL74 to COBOL85 is simple (probably about 80% of the COBOL74 programs will run on COBOL85 with no changes). For further information about the new COBOL standard or other more advanced COBOL developments, contact Don Nelson at Tandem Computers Incorporated, 10555 Ridgeview Court, Cupertino, CA 95014.

Reference

COBOL85 Reference Manual, vols. 1 and 2. Part nos. 82520 A00 and 82521 A00. Tandem Computers Incorporated.

Don Nelson has worked at Tandem for three years, the last two of which were devoted to the writing of the code-generation phase of the Tandem COBOL85 compiler. Before joining Tandem he spent 18 years with another mainframe vendor, working on compilers and operating systems. While there, he worked on five different COBOL compilers. He has been on the CODASYL COBOL Committee since 1971 and has been its chairman since 1977.

Managing System Time Under GUARDIAN 90

The timekeeping services offered by the GUARDIAN operating system were significantly enhanced in the B00 software release. As explained in the article, "New GUARDIAN 90 Timekeeping Facilities" (*Tandem Systems Review*, June 1985), GUARDIAN 90 now supports:

- Four-word, microsecond-resolution timestamps based on the Julian date.
- CPU clock-rate averaging.
- Clock-rate adjustment.
- Automatic Daylight Savings Time (DST) adjustments.
- Julian-date conversion routines.
- A callable procedure to set system clocks.
- An optional IN file for the cold-load Command Interpreter.

This article focuses on techniques for the accurate and reliable initialization of system time on Tandem computers using the GUARDIAN 90 operating system. Familiarity with the timekeeping terminology defined in the previous article is assumed.

System Time

Software designers and users of most computer systems usually assume that system time is always sufficiently accurate for their purposes. Implicit assumptions are that system time is monotonically increasing (i.e., that the clock never runs backwards), that system time is kept accurately by the computer, and that the system clock is somehow always initialized accurately.

It is important to understand how particular systems keep time in order to verify whether these assumptions are valid.

The Tandem System Clock

In Tandem computer systems, there is no "system clock" *per se*; instead, each processor has its own hardware clock. Because all clocks are kept synchronized, programs can be designed as if there were a single system clock.

The operating system is responsible for keeping these clocks synchronized. GUARDIAN 90 accomplishes this task by averaging the values of all processor clocks and adjusting the individual clocks to agree with the average. By averaging the processor clocks, GUARDIAN 90 keeps system time more accurately than pre-B00 versions of GUARDIAN did. Measurements indicate that processor clocks in GUARDIAN 90 systems are usually synchronized to within 5 ms of each other; however, even with the averaging mechanism, clock times fluctuate, and differences of 15 ms between processors are sometimes present. Thus, applications should be designed so as not to rely on perfect synchronization of clocks in all processors.

The GUARDIAN 90 clock-rate-adjustment algorithm requires that clocks running faster than the average be slowed down. This is accomplished in a manner transparent to all programs. Successive calls on the JULIANTIMESTAMP procedure within the same processor always yields monotonically increasing values (unless, of course, the clock is reset). The same is true of the RCLK instruction, with one exception: because the RCLK instruction returns the Local Civil Time (LCT), it "jumps" whenever a Daylight Savings Time (DST) transition occurs. Application designers are therefore encouraged to use the JULIANTIMESTAMP procedure. It returns the Greenwich Mean Time (GMT), which is not subject to DST fluctuations.

Microsecond-resolution Timekeeping

Some applications use timestamps as unique identifiers of transactions. In such a situation, it is important to note that the resolution of the clock may be more important than the accuracy of the clock. If more than one event can occur within a clock "tick," the resolution of such a timestamp prohibits its use as a unique identifier. For example, in most computer systems, a timestamp having a resolution of one second is not sufficient for use as a unique identifier, because several events may occur within one second.

The timestamps provided by GUARDIAN 90 timekeeping services are four-word timestamps, based on the Julian date, and they have microsecond resolution (which, as suggested above, is not the same as microsecond accuracy).

Recovery of Clocks after a Power Failure

As mentioned earlier, each processor in a Tandem system has its own clock, supported by the operating system and microcode and relying upon the processor hardware. If power to the processor is lost, the clock stops. When power is restored, the clock starts running again.

If power is restored before the battery backup is exhausted, a Tandem system automatically performs power-failure recovery. If power remains off for so long that the battery backup is unable to preserve the contents of memory, however, it is impossible to recover from the power failure, and a cold load of the system is required.

When power is restored (assuming the battery backup was not exhausted), the clock in each processor takes up exactly where it left off when the power went down. As part of the power-failure recovery process, the operating system then resynchronizes the clock.

If all processors lose power and power-failure recovery is performed, GUARDIAN 90 synchronizes all clocks to the fastest clock in the system. In this case, the clocks are synchronized, but system time is incorrect by an amount equal to the duration of the power outage. The front panel lights indicate that a power-failure recovery has occurred.

If the power loss is transient, it is possible that only some processors lose power (and subsequently undergo power-failure recovery). In this case, as long as at least one processor in the system does not lose power, the operating system synchronizes the clock of each processor that lost power with the clocks of the processors that continued to run.

Setting the Clock—An Operations Headache

The operations staff traditionally is responsible for initializing the system clock. With some exceptions, discussed later, system time is set by an operator at cold-load time, after a power failure recovery, or when someone notices that the system time is incorrect. Unfortunately, every time someone enters the date and time manually, it is possible that the system clock is being set incorrectly. At best, the clock is being set within a few seconds of the wall clock time or the operator's wristwatch. At worst, the operator may enter the wrong date.

One new feature provided by GUARDIAN 90 is the SETSYSTEMCLOCK procedure. Another useful feature is the ability to specify an input file for the initial (cold-load) Command Interpreter. Together, these features provide several alternatives to the familiar method of requiring the operator to set the system clock. (See the GUARDIAN 90 Software Documentation, or Softdoc, and Nellen, 1985, for details on the SETSYSTEMCLOCK procedure and the cold-load Command Interpreter IN file.)

The COMINT SETTIME command checks the syntax of date and time specifications, requiring only that they be reasonable (i.e., not impossible) and unambiguous. For example, it does not allow the system clock to be set to a Local Civil Time that is within a Daylight Savings Time (DST) transition period, because such a time specification is ambiguous. If it is necessary to set the system clock to a time that is within a DST transition period, the operator must specify the time as Local Standard Time (LST) or Greenwich Mean Time (GMT), which are not ambiguous.

If it is not acceptable for an operator to set the system clock (because of the inaccuracies inherent in this approach), there are two basic alternatives. One method is to allow the operator to set the clock initially, during the cold load, and then run a program to verify and possibly adjust the time after the cold load is complete (but before applications are allowed to start). The other method is to set the system clock programmatically, using an external clock.

Checking the System Clock

Even if a system does not have an external clock, there are ways of checking the system time as set by the operator. Ideally, one would like to validate the time when the SETTIME command is entered by the operator. SETTIME is performed during the cold load, however, and there are complications that make this impractical. Thus, it is necessary to write a program (call it CLOKCHEK for purposes of discussion) that compares the system time against some other time reference after the cold load has been completed.

CLOKCHEK would have to get a timestamp from some reference source, obtain the current system time by calling JULIANTIMESTAMP, compare the two values, and then determine whether or not the current system time was reasonable. Further protection could be afforded by having CLOKCHEK inhibit the start-up of applications if it found the system time to be in error.

To implement a CLOKCHEK program, one must first find a reliable source of timestamps that can be compared with those provided by the JULIANTIMESTAMP call. The following might be used:

- A file containing the oldest and newest dates allowable. The CLOKCHEK program would compare the current date with entries in this file.
- The SYSGEN time obtained via the JULIANTIMESTAMP procedure (which is returned as a GMT timestamp). Assuming the time was correct at the time of the SYSGEN, the SYSGEN time could be used as a lower bound for the current time.
- Another node in the network. Currently, one can send a request to a server on another node for the current GMT from the other node. The requester must, of course, measure the amount of time it takes to get a reply from a remote server and adjust the GMT value by the transit time. This method should be accurate to within a few seconds, but it assumes that another node with a server is accessible and that the time is set correctly on the other node.
- An X.25 network. Some public X.25 packet-switching networks maintain a clock that can be read via a special request packet. The accuracy of the time received is influenced by several factors, such as the accuracy of the clock, the type of communication lines used, and the transmission delays involved. Generally, the time should be accurate to within a few seconds of the network clock's actual time, which may be sufficient for many applications. Prospective users should discuss these problems with the vendor of such network services.

External Clocks

External clocks provide a much more reliable way of initializing system time. By configuring a system to use the initial cold-load Command Interpreter IN file, one can run a program that obtains the time from an external clock and calls the system procedure SETSYSTEMCLOCK to initialize the system clock. Refer to the GUARDIAN 90 Software Documentation, or Softdoc, and Nellen, 1985, for details on the SETSYSTEMCLOCK procedure and the cold-load Command Interpreter IN file.

For purposes of this discussion, an external clock is a hardware device that has, minimally, the following characteristics:

- It contains a precision digital clock and calendar.
- It can be attached via a standard interface to a computer system.
- It can be interrogated for the date and time by programs running on that computer system.

Selecting an External Clock

Many external clocks are available. Selecting an external clock that works well at a specific site is not a trivial task. The following is a general discussion of some of the features one should consider.

Accuracy

The primary requirement for any clock is to keep time accurately. As a minimum, the clock should be accurate to within one second per day; however, some applications require greater accuracy.

Electronic clocks are commonly driven by one of three mechanisms, each of which offers a different level of accuracy:

- An internal oscillator, usually crystal-controlled and temperature-compensated for reasonable accuracy.
- Synchronization to the power-line frequency.
- Synchronization to radio broadcasts of a time standard.

Most crystal-controlled clocks are accurate to within 100 ms per day or better, depending on the quality of the crystal. Such clocks need a battery backup in case external power is lost.

The second type of clock phase-locks its oscillator to the line frequency. This is a simple and effective way of keeping time accurately because the utility companies must synchronize their power grid very closely in order to share power. The line frequency in the United States is maintained to within one cycle (1/60 of a second, or 6.7 ms) per day.

External clocks capable of synchronizing to the power-line frequency generally have the ability to switch to their internal oscillator automatically and run on an internal backup battery if the line power is lost. Then, when external power is available, they synchronize to the line frequency again.

If power for the computer system is supplied by an Uninterruptable Power Supply (UPS), one should determine whether the UPS output is synchronized to the commercial power-line frequency. If it is not, this may create problems for a clock that relies on the line frequency as a standard, because frequency regulation in the UPS system may not be as accurate as that of the commercial power grid. In such a case, one would want either to connect the external clock to the commercial power line or to disable the synchronization of the clock to the line frequency and allow it to use its internal clock.

Some clocks that rely on the power-line frequency can be fooled by transient noise spikes. They run fast because they detect the noise in addition to the line voltage peaks. If a particular clock is affected by such noise, a simple power-line noise filter may solve the problem.

External clocks are much more reliable for initializing system time than checking the system time set by the operator.

Distributed applications (i.e., those that must access several geographically separated computer systems connected in a network) often require that system times at each node be in agreement. That is, each system must be able to calculate the correct Greenwich Mean Time.

Some distributed applications may tolerate differences of a few seconds between nodes, in which case it is possible to send messages to remote nodes to request the time (or to request the time from public packet-switching networks). Some applications require greater accuracy than this, however. One solution is to use an external clock that is designed to receive and decode radio transmissions of standard time signals.

There are radio stations throughout the world that broadcast encoded time signals. Most of these stations are operated by governmental agencies. In the United States, the National Bureau of Standards (NBS) transmits the national time and frequency standard from station WWVB located in Fort Collins, Colorado. The NBS uses an atomic clock to keep the day-to-day deviation of their time signal to within 5 parts in 10^{12}. This is equivalent to 432 ns per day. The NBS also transmits a time signal from the National Oceanic and Atmospheric Administration's geostationary satellites, known as GOES. Some radio-receiver clocks are capable of detecting and decoding the GOES transmissions.

The carrier frequencies and encoding techniques used by the U.S. NBS are not an international standard. The British government operates radio station MSF, which broadcasts a standard time signal from Rugby, England. The West German government station DCF77 broadcasts the time from Mainflingen. The frequencies and codes used in the United States, Britain, and West Germany all differ from each other. A few stations in other parts of the world use the same frequencies and code as the U.S. NBS. Prospective purchasers of radio-receiver clocks would be well advised to determine which stations the clock can receive and decode, and what type of antenna is required for their specific location.

An important feature of any radio-receiver clock is a visible indicator, such as a light, that indicates that the device is receiving the station and has synchronized its clock. Some also provide protocols that allow the computer system to query the clock and determine whether it is synchronized to the radio signal.

Additionally, one should be sure that the clock automatically switches over to an internal crystal-controlled oscillator in the event of a reception failure (and to a battery backup in the event of a concurrent power failure).

Many radio-receiver clocks have an adjustment that permits compensation for the propagation delay. If the distance between the receiver and the transmitter is known, one can compute how long it took the signal to travel that distance and, with the propagation-delay adjustment, correct the clock to compensate for this delay.

Some radio-receiver clocks also have an adjustment that allows correction for Local Standard Time. For our purposes, this is not necessary.

Other Considerations

Resolution. A resolution in the range of one-tenth to one-hundredth of a second is useful.

Calendar. The clock should be able to correctly compute the date, even in leap years.

Human Interface. Non-radio-receiver clocks should have a simple control panel for setting the date and time (and a display to show the current date and time). It may also be desirable to have a lock and key to prevent unauthorized persons from setting the clock.

Computer Interface. A standard interface, such as RS-232 or current-loop, is required. A simple protocol for interrogating the clock is also desirable.

Some vendors of external clocks.*

Chrono-Log Corporation
2 West Park Road
Havertown, PA 19083
Phone: (215) 853-1130
Telex: 831579

Digital Pathways Incorporated
1060 East Meadow Circle
Palo Alto, CA 94303
Phone: (415) 493-5544
TWX 910 379-5034

Hayes Microcomputer Products, Inc.
705 Westech Drive
Norcross, GA 30092
Phone: (404) 449-8791
Telex: 703500 (Hayes USA)

Hopf Elektronik KG
Postfach 1847
Im Hasley 14 c
D-5880 Luedenscheid
West Germany
Phone: 2351/22201
Telex: 826693

Kinemetrics/True Time
3243 Santa Rosa Avenue
Santa Rosa, CA 95401
Phone: (707) 795-2220
Telex: 675402 (Kinemetrics PSD)

Spectracom Corporation
101 Despatch Drive
East Rochester, NY 14445
Phone: (716) 381-4827
Telex: 9103509587

*This list has been compiled from advertisements in various electronics and computer trade publications. Tandem makes no recommendation for, or endorsement of, any of these devices, nor does Tandem intend to imply anything by the absence of any company from this list.

Price. Last, but not least, one should consider how much the clock costs and what the warranty provisions are.

Commercially Available External Clocks

The list on page 52 lists companies that sell external clocks. As the list is not comprehensive, customers should consider it only a starting point when researching sources for external clocks. Also, before purchasing any device, they should consult a Tandem customer engineer and a Tandem systems analyst about the feasibility of using that device with a Tandem computer system.

Configuring an External Clock

Most external clocks are configured in the same way as an asynchronous terminal. The following example is typical:

```
$CLOCK    TATM.7    ASYNCTERM
                    TYPE 6, SUBTYPE 0,
                    RSIZE 80,
                    BAUD9600,
                    NOECHO,
                    CL;
```

Note that this is not a universal example. SYSGEN configuration options should reflect the characteristics of the specific device as described in the manufacturer's installation manual.

The CLOCK Program

Tandem provides a sample program that can be used to initialize the system clock. The source is distributed in a file called SCLOCK in the GUARD2 distribution subvolume of GUARDIAN 90 Site Update Tapes (SUTs). Please note that SCLOCK is an example only. Although it works correctly, it may not be ideal for a particular site. Also, the sequence of characters that it transmits to an external clock is device-dependent; not all external clocks use the same codes. Note too that SCLOCK assumes that the external clock is set to the correct Greenwich Mean Time.

If SCLOCK is modified for a specific user site and compiled into an object file called CLOCK, and the basic logic of the program has not been altered, it can be used in one of two ways. The first is to run it as follows:

```
:RUN CLOCK /IN $CLOCK/
```

where $CLOCK is the device name of the external clock. When run in this mode, CLOCK reads the external clock twice and then, if there are no I/O errors, calls SETSYSTEMCLOCK to set system time. It reads the external clock twice in order to eliminate the effect of potential page faults in the user program. This mode of execution within the initial cold-load Command Interpreter IN file can be used to perform the initial setting of system time.

If the time is to be set via the external clock at some time other than during the cold load, CLOCK should be run at a priority high enough to avoid competition with other processes.

The alternate way to run CLOCK is:[1]

```
:RUN CLOCK /IN $CLOCK, &
                PRI 160, NOWAIT, CPU x / y
```

Specification of a backup CPU number (y) causes CLOCK to behave differently than in the first example. In this case, it reads the external clock and calls SETSYSTEMCLOCK immediately and also every five minutes. (It does not terminate.) CLOCK also sets the system clock whenever it receives a POWERON system message, which indicates that power-failure recovery has occurred.

This alternate mode of execution can also be used within the initial cold-load Command Interpreter IN file to perform the initial setting of system time and to maintain the synchronization of system time with the external clock.

[1]The ampersands (&) used in this example and the following one denote the continuation of a command string that is broken across two lines. They are not needed if the command is entered on an 80-character line.

The following commands could be placed at the beginning of the initial cold-load Command Interpreter IN file:

```
RUN CLOCK /NAME $TIME, IN $CLOCK, &
          PRI 160, NOWAIT, CPU 0 / 1
RUN CLOCK /NAME $TIME, IN $CLOCK, &
          PRI 160, NOWAIT, CPU 1 / 0
DELAY 4 SECONDS
SYSTIMES
```

This example assumes one can cold load via either CPU 0 or 1. The DELAY is present in order to allow the CLOCK program to initialize the system time before anything else is allowed to run.

Note that calling SETSYSTEMCLOCK every five minutes should not result in a reset of the system clock every five minutes. Instead, if the time difference between the system clock and the requested time is small, as one would expect, GUARDIAN 90 uses the time difference to adjust the processor clocks over a ten-second interval. This adjustment algorithm makes small adjustments transparent and facilitates synchronization to an external clock.

Conclusion

The GUARDIAN 90 operating system provides a rich procedural interface to facilitate retrieval of system time, transformations of timestamps, initialization of system time, and retrieval of process execution time. By using an accurate and secure external clock, one can eliminate the possibility of human error in setting the system clock. For geographically distributed systems, the use of external clocks, which can receive and decode standard time broadcasts, provides a simple and reliable method for synchronizing system times closely across the nodes of a network.

References

Nellen, E. 1985. New GUARDIAN 90 Timekeeping Facilities. *Tandem Systems Review*. vol. 1, no. 2. Tandem Computers Incorporated.

Sharma, Sunil. 1985. New Process-timing Features. *Tandem Systems Review*. vol. 1, no. 2. Tandem Computers Incorporated.

System Description Manual. 1985. Part no. 82507 A00. Tandem Computers Incorporated.

System Procedure Calls Reference Manual. 1985. Part no. 82359 A00. Tandem Computers Incorporated.

Eric Nellen joined Tandem in February 1979 as a member of the Software Quality Assurance Group. He has worked in operating systems development for several years and is currently a member of the Operating Systems Kernel Group.

This is the first of a series of *Tandem Systems Review* columns devoted to new and enhanced Tandem products. Each column will briefly describe the new or enhanced software and hardware products that Tandem has recently announced to its users and the computer industry.

Product Overview

Tandem has recently released the following new or enhanced products:

- An 8-Mbyte memory board for NonStop TXP processors.
- The 6600 Intelligent Cluster Controller.
- A C compiler.
- COBOL and FORTRAN separate run-time libraries.
- A COBOL85 compiler (planned for release in the first part of 1986).
- DYNAMITE™ workstation color models 6548 and 6549.
- EM3270 terminal emulator (enhanced IBM 3270 emulation software).
- FASTSORT, a high-performance sort/merge program.
- Information Management Technology (IMT) products FAXLINK™, PC LINK, PS MAIL™, PS TEXT EDIT™, and PS TEXT FORMAT™.
- A Pascal compiler (planned for release in the first part of 1986).
- PATHWAY intelligent device support (IDS).
- TACL, a flexible command interpreter.
- An enhanced TAL compiler.

Literature is available for these products from Tandem sales representatives. The *Programmer Productivity Languages and Tools* product guide describes the languages and tools. Separate data sheets are available for FASTSORT, the 6600 Intelligent Cluster Controller, the EM3270 Terminal Emulator, and the DYNAMITE 6548 and 6549 Workstations. Information sheets are also available for the IMT products.

Throughout this article, the following terms are used to describe the software releases in which the new products are (or will be) available:

- B10, the release of the GUARDIAN 90 operating system made available in mid-1985.
- B20, a new release of GUARDIAN 90 made available in the last calendar quarter of 1985.
- B30, a release of GUARDIAN 90 planned for the first half of 1986.

Brief descriptions of the new or enhanced products follow, alphabetized by product name. (All the IMT products are located under the subheading of that name.)

8-Mbyte Memory Board

An 8-Mbyte memory board is now available for NonStop TXP processors. This product can increase the capacity of main storage to a maximum of 16 Mbytes per processor. For large, high-performance applications, the 8-Mbyte board allows a NonStop TXP processor to store large amounts of data in memory, thus minimizing or eliminating the need for disc access during a transaction.

6600 Intelligent Cluster Controller

The 6600 Intelligent Cluster Controller allows clustering of terminals and workstations to reduce communications line costs and to allow sharing of expensive communications resources such as phone lines and modems. The 6600 controls and helps manage communications between a Tandem host computer and up to eight terminals, workstations, or printers plus one additional dedicated printer. The 6600 can support any combination of Tandem 653X terminals, DYNAMITE 654X workstations, and IBM PCs or PC-compatible workstations. It is compatible with NonStop TXP, NonStop II, and NonStop EXT processors.

The 6600 controller communicates with a Tandem system via SNAX or SNAX6600. Those customers who do not use SNAX currently can benefit from the 6600. SNAX6600 is available for applications that do not need to communicate with IBM SNA controllers.

C Compiler

With the B20 release of GUARDIAN 90, the popular, portable C language became available on Tandem NonStop systems.[1] The Tandem C compiler and run-time library are as compatible as possible with those in other C environments. Since the ANSI X3J11 committee is still working on a C language standard, Tandem C follows the *de facto* standard defined in *The C Programming Language* by Kernighan and Ritchie.

The Tandem C compiler is derived from the Lattice C compiler currently available for the DYNAMITE workstation and other computers. This compiler makes it possible for C program modules to be developed on the DYNAMITE or a PC, transferred to a Tandem NonStop system, and recompiled for execution on the NonStop system. On a NonStop system, a C program can call TAL™ or GUARDIAN procedures to gain access to more functions. Small programs running on a NonStop system can be recompiled to run on a DYNAMITE or PC.

COBOL and FORTRAN Run-time Libraries

Before the B20 software release, the COBOL and FORTRAN compilers were only available as a set. The full set was unnecessary for production systems, as they only require the run-time library. With the B20 release, the run-time library can be ordered separately, allowing customers to order the lower priced run-time library for their production systems and the complete compiler package for their development system.

The new Pascal and C compilers do not have separate run-time libraries. The library routines are bound to the object program, and thus, customers do not have to order the compiler for their production systems.

Pricing for compilers has been revised. Compilers and run-time libraries are now charged on a per-system basis with a one-time initial license fee and a monthly license fee.

[1]The term "NonStop systems" refers to all Tandem processors and the software that runs on them except for NonStop 1+ processors and software.

COBOL85 Compiler

In the B30 software release, Tandem is offering the COBOL85 compiler. (See the accompanying article, "Tandem's New COBOL85.") COBOL85 supports all of the required modules in the American National Standards Institute (ANSI) revised COBOL standard, X3.23-1985, and has extensions for access to standard Tandem facilities. The ANSI standard provides many new features to increase programmer productivity and program maintainability. As mentioned above, Tandem COBOL85 contains a run-time library which is available separate from the compiler.

COBOL85 supports the following modules:

- Nucleus.
- Table handling.
- Sequential I/O.
- Relative I/O.
- Indexed I/O.
- Sort/merge.
- Interprogram communication.
- Source text manipulation.

It supports Level 1 of the optional debug module, also. Two optional modules of the ANSI standard have not been implemented in Tandem COBOL85: the report writer and communications. The segmentation module is almost entirely implemented.

DYNAMITE Color Workstations

The DYNAMITE workstation product line has been enhanced by the addition of two color models. Both color workstations have 14-inch color monitors. The model 6548 has two 360-Kbyte diskette drives; the model 6549 has one 360-Kbyte diskette drive and one 10-Mbyte hard disk drive.

When emulating a 653X terminal, the color models display system information in high-quality white characters on a black background (or, in reverse video, black on white). Color text applications can be developed locally with MS-DOS and BASIC. The bit-mapped graphics option is required to develop color graphics applications or to run third-party software for the IBM PC.

The color DYNAMITE workstation (with the graphics option) provides five ways of formatting information into color text, charts, and graphs. There are two color-text modes (40 × 25 and 80 × 25), two IBM-compatible graphics modes (320 × 200 and 640 × 200), and an extended high-resolution graphics mode (800 × 300) exclusive to the DYNAMITE.

The two color text modes display text in up to 16 different colors on one screen. The dual-mode design of the color monitor allows both alphanumeric and graphic information to be shown on the screen at the same time. Third-party color printers or color plotters can be supported if they have RS-232 serial interfaces and DTR flow control.

Also available is an upgrade option, which converts a DYNAMITE monochrome unit to a color unit. Finally, Information Xchange Facility (IXF) software is now included with all DYNAMITE models.

EM3270 Terminal Emulator

The enhanced EM3270 terminal emulator permits Tandem users, from a single terminal, to access IBM 3270 applications on up to six IBM host computers using either SNA or bisynchronous communications. EM3270 supports two-way (Tandem terminal to IBM system) and simultaneous three-way (Tandem terminal to both the Tandem and IBM systems) communications. Terminal users can switch between sessions by pressing the SWITCH key, or the switch can be made programmatically from the Tandem application.

Users of Tandem 653X terminals, DYNAMITE 654X workstations, or IBM PCs can run IBM 3270 and Tandem PATHWAY applications concurrently, and they can alternate between IBM SNA and bisynchronous hosts through menus activated by a HOST key.

EM3270 also allows Tandem printers to emulate the IBM 328X family of printers in both bisynchronous and SNA applications. Each EM3270 process can support a combination of 15 to 20 devices configured as terminals and printers.

If EM3270 is to be used in an SNA environment, the B10 release of the GUARDIAN 90 operating system is required. EM3270 can reside on a system that is not running SNAX and access SNAX on a separate system through an EXPAND™ network.

FASTSORT

FASTSORT is a high-performance sort/merge program for Tandem NonStop systems. Available with the B10 release of the GUARDIAN 90 operating system, it is an optional product and must be purchased separately. It provides all the functions of the standard SORT program, but it performs better and offers additional features. FASTSORT will eventually replace SORT.

When installed on a system, FASTSORT is used in:

- Conversational sorts.
- Sorts invoked from TAL or COBOL applications.
- FUP manipulation of alternate-key files.
- ENFORM sorts during report generation.

FASTSORT sorts faster serially than Tandem's standard SORT program, and it offers the high performance of parallel sorting as well. By sorting in parallel, FASTSORT significantly reduces the elapsed time of a sort run by distributing the work load among multiple processors and discs. Parallel sorting with FASTSORT is more than seven times faster than serial sorting with SORT. Optimum performance can be attained by using NonStop TXP processors, 3107 disc controllers, the DP2 disc process, and extended memory.

IMT Products

FAXLINK

FAXLINK, an image storage, forwarding, and retrieval facility made available in June 1985, allows users to move printed documents or pictures through a Tandem network using any CCITT group III facsimile machine. In addition, documents created on any 327X, conversational, or TTY terminal, IBM PC or PC-compatible workstation, or Tandem's own 653X terminals and DYNAMITE workstations can be sent to a remote facsimile device without the need for a terminal or workstation at the receiving site. Thus, electronic mail can be sent together with signed letters, diagrams, or other documents. FAXLINK is ideal for business operations that require the routine delivery of information such as orders, invoices, shipping instructions, or design changes.

FAXLINK delivers facsimiles via a Tandem network reliably and at low cost, integrates facsimiles with PS MAIL, simplifies document addressing, accepts input without a terminal, stores facsimiles on-line, offers flexible delivery options, achieves high performance without expensive devices, turns facsimile machines into remote printers, and includes all required hardware and software.

PC LINK

PC LINK is a collection of software programs, both host-resident and diskette-based, that allows IBM PCs and PC-compatible workstations that are connected to a Tandem system to emulate a Tandem 653X terminal or an IBM 327X terminal. PC LINK allows PC users to send and receive electronic mail, transfer files and manipulate information stored in a Tandem data base, and access both Tandem and IBM host applications. These services significantly increase a user's productivity.

PC LINK accesses Tandem and 3270 applications on-line, uses PS MAIL, takes advantage of system peripherals, and integrates system data with PC applications. PC LINK consists of four software tools (EM6530PC, IXF/PC, PCFORMAT™, and EM3270) made available in August 1985.

PS MAIL

PS MAIL is a distributed electronic mail system, designed to provide easy-to-use electronic communications among users of a wide variety of desktop devices. PS MAIL lets users of IBM 327X and TTY terminals, IBM PCs and PC-compatible workstations, and Tandem 653X terminals and DYNAMITE workstations to send and receive electronic mail and to store, forward, and file documents electronically. PS MAIL for TTYs was released in B10, and PS MAIL for IBM 3270s and Tandem terminals was released in B20. PS MAIL is built on the TRANSFER™ time-staged information delivery software, first shipped by Tandem in 1982.

PS MAIL features include built-in help, a directory of PS MAIL users, forward and reply features, user-defined distribution lists, assured delivery, delivery certification, facsimile and PC document delivery, filing and retrieval, efficient resource utilization, an easy-to-use editor, an automatic filing and response facility for users on vacation, and customized administration.

PS TEXT EDIT

PS TEXT EDIT, available in the B20 release, is an advanced, full-screen text-editing system for creating reports, documents, memos, letters, and computer programs. PS TEXT EDIT provides a complete set of powerful, built-in features for producing any kind of document quickly and easily. It offers on-line help for all PS TEXT EDIT functions, easy transfer of information between documents or within a single document (with a split-screen option), and function keys that can be redefined around your particular needs. PS TEXT EDIT can extend its power through several compatible programs, including the PS TEXT FORMAT print formatter.

PS TEXT FORMAT

PS TEXT FORMAT, available in the B10 release, gives users complete control over the layout of documents printed on any Tandem printer. A wide range of features, such as proportionally spaced characters, subscripts, and superscripts, lets users take full advantage of the capabilities of the Tandem 5530 daisy-wheel printer. PS TEXT FORMAT is a sophisticated text formatter, and it is very easy to use.

With PS TEXT FORMAT, users can translate command names and error messages from English to another language and change parameter defaults for paper sizes and margin widths. Users can also modify the appearance of type, design page layouts, alter letters and documents, format reports, store customized formats, time-stamp and extract information, perform arithmetic computations in text, customize PS TEXT FORMAT itself, merge information into text from a distribution list for mass mailings, and print and move files.

Pascal Compiler

The Pascal language was designed to support modern high-level programming techniques. It is well structured, easily understood, portable, and, yet, relatively efficient. Pascal programs tend to be correct and robust. The compiler actively assists in finding logic errors or interface errors at compilation time, and optional run-time checks help find any remaining errors. Pascal has especially good support for multilevel data structures that use pointers or nested records.

Tandem Pascal, available with the B30 software release, is a superset of the 1983 ANSI definition of Pascal, formally known as ANSI/IEEE 770 X3.97-1983. It also complies with Level 0 of the International Standards Organization (ISO) Pascal (ISO 7185). Compliance with both standards is measured by the Pascal Validation Suite of the British Standards Institution. Tandem Pascal is extended with features that facilitate large programs, business applications, and systems programming.

PATHWAY Intelligent Device Support (IDS)

Before the B10 software release, the PATHWAY transaction processing system supported only terminal-type devices (6530, 3270, conversational, 6520, and 6510 terminals and the DYNAMITE workstation). The user environment has evolved over the last several years, and PATHWAY applications now need to interface with intelligent workstations, ATMs, POS devices, and GUARDIAN processes.

Initially, programmers handled this requirement by implementing multithreaded front-end processes, which stood between the intelligent device and the Terminal Control Process (TCP). These front-end processes converted messages into the format required by the message receiver. Control and device-specific information was added or deleted as appropriate.

PATHWAY IDS, available with the B10 release of PATHWAY, eliminates the need for front-end processes by providing increased Screen COBOL support for intelligent devices. With it, the TCP transmits a message to an intelligent device when the Screen COBOL program requests the action. This action is translated into the appropriate GUARDIAN write, read, or writeread procedures to transmit the correct data. Screen COBOL programs are responsible for message resynchronization (error trapping, error recovery, I/O retries, and so on) and for adhering to the intelligent device's protocol.

TACL

TACL, the Tandem Advanced Command Language (pronounced *tackle*), is a flexible command interpreter that can be customized for a particular user or installation. TACL is a standard product for GUARDIAN 90 users (as are Tandem's other program development tools, INSPECT™, BINDER™, and CROSSREF™). These standard products are provided at no additional charge. TACL is automatically included in all B20 GUARDIAN 90 shipments.

TACL provides all the capabilities of Tandem's command interpreter, COMINT (and will eventually replace COMINT). In addition, it allows users to write macros to define frequently used commands. These macros and other functions can then be mapped to function keys. At its most advanced level, TACL becomes a powerful high-level interpreted language.

TAL Compiler

TAL, the Transaction Application Language, is Tandem's systems-programming language. As of the B20 software release, it has many new features:

- The elapsed time for compilations has been reduced as much as 20%.
- A labeled CASE statement makes programs much easier to write, debug, and maintain.
- In compilations with the ?ERRORFILE directive, syntax errors are written to a disc file, allowing programmers to use PS TEXT EDIT to display the source program in one window and the error messages in another.
- It has additional support for data declared in extended memory.
- A new data type UNSIGNED, for declaring bit fields, allows pointers to be declared within a structure and templates to be used as substructures.

Corinne Robinson is the Product Manager for Tandem's languages and tools. She joined Tandem in June 1983 as a software designer. Before joining Tandem, Corinne spent seven years working in microprogramming, diagnostics, and languages for another computer vendor. Corinne has a B.S. in Information and Computer Science from the University of California at Irvine.

Subscription Policy for Software Manuals

In May 1985 Tandem introduced a new subscription service and a new update service for its software manuals. Their purpose is to help Tandem customers keep their Tandem software documentation up-to-date with the latest software releases.

The *Software Manual Subscription Service* provides the sets of manuals (including binders) that describe Tandem software products. It includes one year of the *Manual Update Service*.

The Manual Update Service provides updates (replacement pages) and revisions (entire replacement manuals) for sets of Tandem software manuals (but no binders).

In addition, Tandem customers in the United States can now order manuals via a toll-free 800 phone number (if a blanket purchase order has already been submitted to Tandem Sales Administration in Cupertino, California). This procedure can be used for ordering individual manuals or additional subscriptions.

Software Manual Subscriptions

One software manual subscription now entitles a subscriber to one or more sets of manuals (describing specific software products, as selected by the subscriber) and one year of updates *for the manual sets ordered*.

A basic set of manuals is available for each Tandem operating system (GUARDIAN and GUARDIAN 90), as well as for the extended function combination, GUARDIAN 90XF™ (which includes GUARDIAN 90, ENCOMPASS™, EXPAND, and TRANSFER). For each optional software product or package of products, smaller manual sets are offered. If a software product is described in three manuals (e.g., a reference manual, a user's guide, and an operations guide), all three are included in the subscription service and update service. Table 1 (page 62) lists the sets available for NonStop systems (NonStop II and TXP processors).

Renewing the Manual Update Service

Renewals for the Manual Update Service are for a term of one year. Three months before the term ends, a renewal letter is sent to the subscriber, detailing the sets of manuals that will require updates.

Customers must send in a purchase order for the renewal; without it, update service expires. They can change quantities when they renew.

Table 1.
Product identification numbers for the Software Manual Subscription Service and Manual Update Service, NonStop software (NonStop II and NonStop TXP processors).[1]

Software product	Software Manual Subscription Service[2] product ID	Manual Update Service (only) product ID
Operating system software		
GUARDIAN 90 package	9072MS	9072MU
GUARDIAN 90XF package	9090MS	9090MU
Optional software		
EXCHANGE	9054MS	9054MU
EM3270	9059MS	9059MU
XRAY	9056MS	9056MU
EXPAND	9057MS	9057MU
X.25 Access Method	9060MS	9060MU
AM3270	9061MS	9061MU
TR3271	9062MS	9062MU
AM6520	9063MS	9063MU
SNAX	9064MS	9064MU
TMF	9066MS	9066MU
ATP6100	9075MS	9075MU
6100BSC	9076MS	9076MU
6100ADCCP	9077MS	9077MU
6100MS01	9078MS	9078MU
CP6100	9079MS	9079MU
ENCOMPASS	9116MS	9116MU
T-TEXT	9120MS	9120MU
TRANSFER	9160MS	9160MU
DDL	9150MS	9150MU
Spooler	9151MS	9151MU
ENFORM	9102MS	9102MU
PATHWAY	9103MS	9103MU
ENABLE	9155MS	9155MU
COBOL	9201MS	9201MU
FORTRAN	9202MS	9202MU
MUMPS	9203MS	9203MU
BASIC	9204MS	9204MU
COBOL (NonStop systems)	9251MS	9251MU
FORTRAN (NonStop systems)	9252MS	9252MU

[1]For the titles, descriptions, and Tandem part numbers of individual Tandem software manuals, see Tandem's *Catalog of Software Publications and Related Products*, part number 82552 B00.

[2]The subscription service includes all manuals for that software product or package and Manual Update Service for one year.

Ordering by Phone in the United States

As mentioned earlier, U.S. customers can now order individual software manuals and subscriptions through a toll-free 800 telephone number. Those who want this flexibility must send an open (or blanket) purchase order or equivalent document to:

Sales Administration, Manuals Group
Tandem Computers Incorporated
19191 Vallco Parkway, MS 4-05
Cupertino, California 95014

The open purchase order should specify the dollar amount and duration of the order. (This protects the customer and Tandem.) It should also specify other pertinent information, such as the names of customer personnel authorized to place phone orders. The Manuals Subscription Group will send the customer the 800 phone number with an acknowledgment of the purchase order and will keep the open purchase order on file for reference on all manual invoices.

Billing

Prices for Subscription and Update Services
Individual Tandem software manuals were repriced in April 1985. The current prices are available from Tandem sales representatives and also from the Manuals Group in Tandem Sales Administration mentioned above.

Prices for the software manual services are a fixed percentage of the list price of the manual set(s) ordered. The price of the Software Manual Subscription Service is 130% of the current list price of the manuals (that is, 20% less than if the manuals and the Manual Update Service were purchased separately), plus shipping and handling.

The charge for the Manual Update Service alone is 50% of the current list price of the manuals, plus shipping and handling. Tandem software manuals (especially the basic set of manuals for the operating system) are updated at least every two years. Although the frequency and size of the updates vary, subscribers receive substantial updates and revisions commensurate with price, on a timely basis.

Shipping and Handling

For both the initial subscription service and the Manual Update Service, shipping and handling charges are added to the orders and billed in advance. (When individual manuals are ordered, however, shipping and handling are billed at the time of shipment.)

Shipping and handling have not been incorporated into the price of the manuals because if each manual were priced to fully recover its individual shipping and handling costs, the total price would be unreasonably high for large orders.

Invoicing

Subscription service orders are invoiced in full, upon shipment of the initial set of manuals. Update service renewals are invoiced upon receipt of the renewal purchase order.

No Volume Discounts

No volume discounts are available, as comparatively little economy is achieved in filling an order for 50 sets of manuals as opposed to filling one set, especially when (as is common in large orders) the 50 are to be sent to 25 different addresses.

Billing for Additional Subscriptions

When subscribers place orders for additional subscriptions after placing an initial order (e.g., to get a set of manuals they did not previously order or to get a second set of some manuals they had ordered), they are billed for a full one-year subscription at the time of the order. Then when they renew their Manual Update Service, the rate for the renewal period is prorated (to account for the unused subscription service months included in their additional order) and a common renewal date is established.

For example, if an initial subscription were ordered in April and additional subscriptions were ordered in August, the renewal for the update service would include a year of update service for the April subscription and eight months of update service for the August subscription(s). This allows the update services for all subscriptions to come up for renewal at the same time.

Cancellation Policy

Subscription Service

If customers return all packages unopened and undamaged to Tandem's distribution point within 60 days of the order, credit is issued for the price of the subscription, less a 15% restocking charge. The shipping and handling charge is not refunded, and the customer must pay the return freight.

If a subscription is cancelled within the first six months, Tandem refunds 20% of the subscription price, excluding the shipping and handling charges. The customer can keep the initial set of manuals.

Update Service

If a customer cancels update service during the first six months, Tandem refunds half of the update service price, excluding handling charges. After six months, a refund cannot be issued, and the update service runs to completion.

B*nn* Manual Sets

Before the B00 release, the last general distribution of Tandem software manuals occurred with the A06/E07 release of the GUARDIAN operating system in February 1984. At that time "system manual kits," consisting of one of every manual, were sent to all customers. Until May 1985, subscriptions were available only for these manual kits. In the United States, the kit subscriptions were priced well below current printing costs.

Table 2.
B*nn* software manual sets for subscription and update services.[1]

Product ID	Description
9*nnn*MS	Software Manual Subscription Service for the software product "9*nnn*," including the initial set of current manuals describing it and one year of Manual Update Service.
9*nnn*MU	Manual Update Service for the software product "9*nnn*" for one year. Provides updates for an (assumed) existing set of manuals.
9B*nn*	Latest versions of all software manuals. No update service.
9B*nn*MS	Software Manual Subscription Service that includes the latest versions of all manuals and one year of Manual Update Service.
9B*nn*MU	Manual Update Service for all software manuals for one year. Provides the updates but not the initial set of manuals.

[1]In this table, "B*nn*" represents any software release in the B series, e.g., B10, B20, etc.

With the B00 software release, 65 of 83 NonStop manuals (describing NonStop II and NonStop TXP software) were changed, requiring either updated pages or complete revisions.[1] Table 2 lists the two B*nn* manual sets available. (B*nn* is used in this article to represent all releases of the "B series" software, e.g., B10, B20, etc.) Below, three B*nn* update situations are explained.

Updating Manuals Provided with the System
Customers who are licensed to use Tandem software, and who pay for software maintenance or pay the monthly license fee, automatically receive one set of B*nn* updates for the set of manuals Tandem provides with a Tandem system.

Those who want B*nn* updates for software for which they are not licensed can order them through the Manual Update Service.

[1]NonStop 1+ manuals are not included in the B00 software manual distribution.

Replacing A-Series Software Manual Sets
In addition to the updates for the set of manuals Tandem provides with a Tandem system, mentioned above, most customers have ordered additional manuals. Those who have a *current* subscription to A-Series manual sets (under the old subscription policy) receive B*nn* updates until the end of that subscription's term. To keep that set of B*nn* manuals up-to-date when the old subscription expires, these customers must order the 9*nnn*MU Manual Update Service for the manuals to be updated (where 9*nnn* corresponds to the Tandem software product numbers). They can keep complete sets current by ordering the 9B*nn*MU Manual Update Service.

Customers whose subscriptions are no longer current have several choices. They can:

- Order individual manuals by manual number, up to and including full sets. (Ordering full sets in this way would be the least economical alternative in the long run.)
- Order Software Manual Subscription Service products by Tandem software product number for the number of sets needed. This method allows specific customization of the manuals most needed. For example, ordering one 9103MS yields the current version of PATHWAY manuals plus update service for these manuals for one year.
- Order a complete replacement set of up-to-date B*nn* manuals, including manual updates for one year, with Software Manual Subscription Service product 9B*nn*MS.

Ordering Individual Manuals

Tandem customers who want to order individual manuals (as opposed to manual sets based on a software product) can order them by Tandem part number. (See the *Catalog of Software Publications and Related Products,* part number 82522 B00, for the titles, descriptions, and part numbers.) Orders for individual manuals are not viewed as subscription service orders and do not include update service.

Tim McSweeney is manager of the Pricing Analysis Group in Tandem's Marketing organization. He has also worked as a senior marketing analyst in the Competitive Analysis Group. Before joining Tandem in 1983, Tim was associated with a start-up software development company. Before that he worked for nine years for another computer vendor in several capacities, including international sales and computer support.

The *Tandem Journal* became the *Tandem Systems Review* in February 1985. Four issues of the *Tandem Journal* were published:

Volume 1, number 1	Fall 1983	Part No. 83930
Volume 2, number 1	Winter 1984	Part No. 83931
Volume 2, number 2	Spring 1984	Part No. 83932
Volume 2, number 3	Summer 1984	Part No. 83933

As of February 1986, three issues of the *Tandem Systems Review* have been published:

Volume 1, number 1	February 1985	Part No. 83934
Volume 1, number 2	June 1985	Part No. 83935
Volume 2, number 1	February 1986	Part No. 83936

The articles published in all seven issues are arranged by subject below. (*Tandem Journal* is abbreviated as TJ and *Tandem Systems Review* as TSR.) For those articles whose subject matter falls in more than one area, the title may be listed in more than one area (notably, those articles about system and application performance).

Article title	Author(s)	Publication	Volume, Issue	Season or Month and Year	Part Number
Operating system					
Changes in FOX	N. Donde	TSR	1,2	June 1985	83935
A Comparison of the DP1 and DP2 Disc Processes	T. Schachter	TSR	1,2	June 1985	83935
DP1-DP2 File Conversion: An Overview	J. Tate	TSR	2,1	Feb. 1986	83936
DP2 Highlights	K. Carlyle, L. McGowan	TSR	1,2	June 1985	83935
DP2 Key-sequenced Files	T. Schachter	TSR	1,2	June 1985	83935
DP2 Performance	J. Enright	TSR	1,2	June 1985	83935
DP2's Efficient Use of Cache	T. Schachter	TSR	1,2	June 1985	83935
Determining FCP Conversion Time	J. Tate	TSR	2,1	Feb. 1986	83936
The GUARDIAN Message System and How to Design for It	M. Chandra	TSR	1,1	Feb. 1985	83934
Improved Performance for BACKUP2 and RESTORE2	A. Khatri, M. McCline	TSR	1,2	June 1985	83935
Increased Code Space	A. Jordan	TSR	1,2	June 1985	83935
Introducing TMDS, Tandem's New On-line Diagnostic System	J. Troisi	TSR	1,2	June 1985	83935
Managing System Time Under GUARDIAN 90	E. Nellen	TSR	2,1	Feb. 1986	83936
New GUARDIAN 90 Timekeeping Facilities	E. Nellen	TSR	1,2	June 1985	83935
New Processing-timing Features	S. Sharma	TSR	1,2	June 1985	83935
NonStop II Memory Organization and Extended Addressing	D. Thomas	TJ	1,1	Fall 1983	83930
Optimizing Sequential Processing on the Tandem System	R. Welsh	TJ	2,3	Summer 1984	83933
Robustness to Crash in a Distributed Data Base: A Nonshared-memory Approach	A. Borr	TSR	1,2	June 1985	83935
TACL, Tandem's New Extensible Command Language	J. Campbell, R. Glascock	TSR	2,1	Feb. 1986	83936
The Tandem Global Update Protocol	R. Carr	TSR	1,2	June 1985	83935
Using FOX to Move a Fault-tolerant Application	C. Breighner	TSR	1,1	Feb. 1985	83934
Buffering for Better Application Performance	R. Mattran	TSR	2,1	Feb. 1986	83936
VIEWSYS: An On-line System-resource Monitor	D. Montgomery	TSR	1,2	June 1985	83935
Writing a Command Interpreter	D. Wong	TSR	1,2	June 1985	83935
Languages					
An Introduction to Tandem EXTENDED BASIC	J. Meyerson	TJ	2,2	Spring 1984	83932
TACL, Tandem's New Extensible Command Language	J. Campbell, R. Glascock	TSR	2,1	Feb. 1986	83936
Tandem's New COBOL85	D. Nelson	TSR	2,1	Feb. 1986	83936

Continued on next page.

Article title	Author(s)	Publication	Volume, Issue	Season or Month and Year	Part Number
Data management					
The ENABLE Program Generator for Multi-file Applications	B. Chapman, J. Zimmerman	TSR	1,1	Feb. 1985	83934
The ENCORE Stress Test Generator for On-line Transaction Processing Applications	S. Kosinski	TJ	2,1	Winter 1984	83931
Improvements in TMF	T. Lemberger	TSR	2,1	June 1985	83935
A New Design for the PATHWAY TCP	R. Wong	TJ	2,2	Spring 1984	83932
The PATHWAY TCP: Performance and Tuning	J. Vatz	TSR	1,1	Feb. 1985	83934
The Relational Data Base Management Solution	G. Ow	TJ	2,1	Winter 1984	83931
TMF and the Multi-threaded Requester	T. Lemberger	TJ	1,1	Fall 1983	83930
TMF Autorollback: A New Recovery Feature	M. Pong	TSR	1,1	Feb. 1985	83934
The TRANSFER Delivery System for Distributed Applications	S. Van Pelt	TJ	2,2	Spring 1984	83932
Understanding PATHWAY Statistics	M. Pong	TJ	2,2	Spring 1984	83932
Data communications					
The 6100 Communications Subsystem: A New Architecture	R. Smith	TJ	2,1	Winter 1984	83931
A SNAX Passthrough Tutorial	D. Kirk	TJ	2,2	Spring 1984	83932
SNAX/HLS: An Overview	S. Saltwick	TSR	2,1	June 1985	83935
Processors					
The High-Performance NonStop TXP Processor	W. Bartlett, T. Houy, D. Meyer	TJ	2,1	Winter 1984	83931
The NonStop TXP Processor: A Powerful Design for On-line Transaction Processing	P. Oleinick	TJ	2,3	Summer 1984	83933
Peripherals					
Introducing the 3207 Tape Controller	S. Chandran	TSR	2,1	June 1985	83935
The Model 6VI Voice Input Option: Its Design and Implementation	B. Huggett	TJ	2,3	Summer 1984	83933
The V8 Disc Storage Facility: Setting a New Standard for On-line Disc Storage	M. Whiteman	TSR	2,1	June 1985	83935
Workstations					
An Introduction to DYNAMITE Workstation Host Integration	S. Kosinski	TSR	2,1	June 1985	83935
The DYNAMITE Workstation: An Overview	G. Smith	TSR	2,1	June 1985	83935
Application development and performance					
Buffering for Better Application Performance	R. Mattran	TSR	2,1	Feb. 1986	83936
PATHFINDER—An Aid for Application Development	S. Benett	TJ	1,1	Fall 1983	83930
Optimizing Sequential Processing on the Tandem System	R. Welsh	TJ	2,3	Summer 1984	83933
System performance and tuning					
Credit-authorization Benchmark for High Performance and Linear Growth	T. Chmiel, T. Houy	TSR	2,1	Feb. 1986	83936
DP2 Performance	J. Enright	TSR	1,2	June 1985	83935
The High-Performance NonStop TXP Processor	W. Bartlett, T. Houy, D. Meyer	TJ	2,1	Winter 1984	83931
Improved Performance for BACKUP2 and RESTORE2	A. Khatri, M. McCline	TSR	1,2	June 1985	83935
The NonStop TXP Processor: A Powerful Design for On-line Transaction Processing	P. Oleinick	TJ	2,3	Summer 1984	83933
The PATHWAY TCP: Performance and Tuning	J. Vatz	TSR	1,1	Feb. 1985	83934
The Performance Characteristics of Tandem NonStop Systems	J. Day	TJ	1,1	Fall 1983	83930
VIEWSYS: An On-line System-resource Monitor	D. Montgomery	TSR	1,2	June 1985	83935
Manuals and courses					
B00 Software Manuals	S. Olds	TSR	1,2	June 1985	83935
New Software Courses	M. Janow	TSR	1,2	June 1985	83935
Subscription Policy for Software Manuals	T. McSweeney	TSR	2,1	Feb. 1986	83936
Miscellaneous					
Highlights of the B00 Software Release	K. Coughlin, R. Montevaldo	TSR	1,2	June 1985	83935
Tandem's New Products	C. Robinson	TSR	2,1	Feb. 1986	83936

TANDEM PUBLICATIONS ORDER FORM

The *Tandem Systems Review* and the Tandem Application Monograph Series are combined in one subscription. Use this form to subscribe, change a subscription, and order back copies.

For requests *within the U.S.*, send this form to:

Tandem Computers Incorporated
Sales Administration
19191 Vallco Parkway, MS 4-05
Cupertino, CA 95014-2599

For requests *outside the U.S.*, send this form to your local Tandem sales office.

Check the appropriate box(es):

- ☐ New subscription (# of copies desired _____)
- ☐ Subscription change (# of copies desired _____)
- ☐ Request for back copies. (Shipment subject to availability.)

Print your current address here:

_____
ADDRESS

_____

_____

_____

_____

_____
ATTENTION

_____
PHONE NUMBER (U.S.)

If your address has changed, print the old one here:

_____
ADDRESS

_____

_____

_____

_____

_____
ATTENTION

_____
PHONE NUMBER (U.S.)

To order back copies, write the number of copies next to the title(s) below.

NUMBER OF COPIES

Tandem Journal

_____ Part No. 83930, Vol. 1, No. 1, Fall 1983

_____ Part No. 83931, Vol. 2, No. 1, Winter 1984

_____ Part No. 83932, Vol. 2, No. 2, Spring 1984

_____ Part No. 83933, Vol. 2, No. 3, Summer 1984

Tandem Systems Review

_____ Part No. 83934, Vol. 1, No. 1, February 1985

_____ Part No. 83935, Vol. 1, No. 2, June 1985

_____ Part No. 83936, Vol. 2, No. 1, February 1986

Tandem Application Monograph Series

_____ Part No. 83900, *Developing TMF-Protected Application Software,* March 1983, AM-005

_____ Part No. 83901, *Designing a Tandem/Word Processor Interface,* March 1983, AM-006

_____ Part No. 83902, *Integrating Corporate Information Systems: The Intelligent-Network Strategy,* March 1983, AM-007

_____ Part No. 83903, *Application Data Base Design in a Tandem Environment,* August 1983

_____ Part No. 83904, *Capacity Planning for Tandem Computer Systems,* October 1984

_____ Part No. 83905, *Sociable Systems: A Look at the Tandem Corporate Network,* May 1985

_____ Part No. 83907, *Designing a Network-Based Transaction-Processing System,* April 1982, SEDS-002

TANDEM EMPLOYEES: PLEASE ORDER YOUR COPIES THROUGH YOUR MARKETING LITERATURE COORDINATOR.

02/86

TANDEMCOMPUTERS

Part No. 83936

400098 02/86 Printed in USA