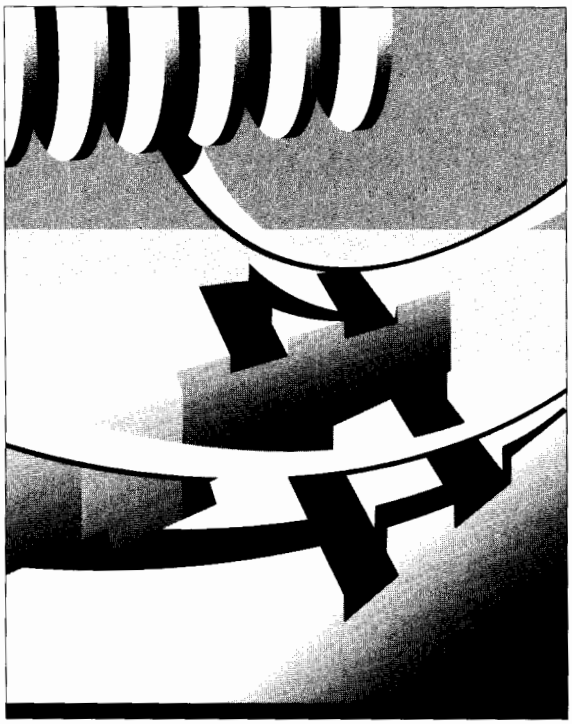
T A N D E M

# SYSTEMS REVIEW

VOLUME 6, NUMBER 1 MARCH 1990

BRANDIFINO



This issue of the *Tandem Systems Review* was produced entirely on a Macintosh. Text files were downloaded from the Tandem host and formatted in Microsoft Word. Art figures were created in Adobe Illustrator. The text and art were imported into Aldus PageMaker for the page layout process. The cover graphic was drawn by hand, traced using Adobe Illustrator, and separated in Adobe Separator. Films were output on a Linotronic 300.

**Volume 6, Number 1, March 1990**

**Editorial Director**
Susan Wayne Thompson

**Editor**
Anne Lewis

**Associate Editors**
Steven Kahn
Mark Peters

**Technical Advisors**
Mark Anderton
Bart Grantham

**Assistant Editor**
Sarah Rood

**Electronic Publishing**
Annie F. Valva

**Art Director**
Janet Stevenson

**Cover Art and Illustrations**
Niklas Hallin

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:** As of the March 1990 issue, subscriptions to the *Tandem Systems Review* must be approved by a Tandem representative. Complete the first portion of the order form at the back of this copy and send the form to your local Tandem sales office.

**Comments:** The editors welcome suggestions for content and format. Please send them to the *Tandem Systems Review*, LOC 216-05, 18922 Forge Drive, Cupertino, CA 95014.

# Editor's Preface

Open Systems Interconnection (OSI) is the international seven-layer reference model that sets standards for the connection of heterogeneous computer systems. Tandem has developed two communication software products that adhere to OSI standards and thus enable Tandem™ systems to operate with OSI-standard subsystems from other vendors. The opening article by Smith describes how the OSI/AS (Application Services) and OSI/TS (Transport Services) products use OSI Session and Transport Layers to allow a direct dialogue with applications in a multivendor environment. The author discusses the capabilities of the products with reference to the appropriate OSI standards and describes how they are implemented in the Tandem environment.

The article by Earle and Wakashige is the third in a series of articles published by the *Tandem Systems Review* (April 1989 and September 1989) on batch processing on Tandem systems. This article describes the NetBatch™-Plus product, the enhanced version of the NetBatch batch scheduler. The NetBatch-Plus software provides a full-screen interface and database that integrates with the original NetBatch software. The authors describe how the NetBatch-Plus features meet the scheduling requirements of batch processing and suggest ways to exploit these features to take advantage of Tandem system architecture.

Because of the increasing interest in Structured Query Language (SQL), a growing number of Tandem users are considering converting from the ENSCRIBE database record manager to a NonStop™ SQL relational database management system. Weikel explores the issues that must be weighed when deciding to convert to NonStop SQL and discusses the considerations that help to make a conversion successful. The article also describes the considerations for a successful conversion and emphasizes the importance of redesigning and rewriting the application during conversion.

The Tandem online transaction processing (OLTP) architecture allows concurrent transactions to operate on shared data and, at the same time, maintain transaction integrity and high performance. When more than one transaction operates on the same data, the database record locking mechanisms ensure transaction isolation and optimum user response times. NonStop SQL and the Transaction Monitoring Facility (TMF™) product provide locking options that give programmers flexibility when regulating contention among concurrent transactions. The article by Senf describes how the NonStop SQL options can be used to regulate concurrency. The discussion also includes techniques that application designers can use to anticipate transaction wait times and determine the chances of locks occurring.

Finally, this issue includes an index of *Tandem System Review* articles. The purpose of this index is to allow readers to have a list of all articles published on each subject or product. If you would like to order back issues of the *Tandem Systems Review,* submit the order form to your Tandem representative.

Susan W. Thompson

# Building Open Systems Interconnection with OSI/AS and OSI/TS

The Tandem Open Systems Interconnection (OSI) networking products OSI/AS (Application Services) and OSI/TS (Transport Services) enable Tandem™ systems to interwork with other vendors' systems and workstations, thus allowing cooperative processing between these systems. Two fundamental aspects of the Tandem approach to networking are openness, or the ability to interwork with heterogeneous equipment, and the support of standards. OSI/AS and OSI/TS allow a wide range of users easy access to database applications that run on Tandem systems, including those using the Tandem NonStop™ SQL relational database management system.

This article discusses the capabilities of the Tandem OSI products, with reference to the appropriate OSI standards, and shows how the various OSI products fit together. It also describes how the OSI products are implemented in a Tandem environment.[1]

This article assumes the reader is familiar with OSI concepts and nomenclature. Readers unfamiliar with OSI are urged to read "The OSI Model: Overview, Status, and Current Issues" in the April 1989 issue of the *Tandem Systems Review* (Dunn).

## Overview of OSI/AS and OSI/TS

OSI/AS is an implementation of the OSI Session Layer. OSI/TS is an implementation of the OSI Transport Layer and functions of the OSI Connectionless Network Protocol. These products can use different wide area networks (WANs) and local area networks (LANs). One advantage of the Tandem approach to OSI is that OSI applications need not concern themselves about the type of network they are using. (See Figure 1.)

[1]The OSI/AS and OSI/TS products described in this article have the features and functionalities of the initially released OSI core services products, which were announced in April 1989.

Two separate Tandem products, the X.25 Access Method (X25AM) and the Tandem LAN Access Method (TLAM), provide the network connection. X25AM is a WAN-oriented product and TLAM is a LAN product. TLAM supports two different types of LAN, carrier sense multiple access with collision detection (CSMA/CD) and Token Bus. These products, together with OSI/AS and OSI/TS, implement the OSI layers 1 through 5 operating under the Tandem GUARDIAN™ 90 operating system. (See Figure 2.)

## Using OSI/AS and OSI/TS

The Tandem OSI products are meant to connect heterogeneous systems to allow cooperative processing. Users can provide OSI-defined applications like FTAM (File Transfer, Access, and Management) or X.400 that use OSI/AS or OSI/TS; these applications take advantage of OSI products supported by Tandem that can use WANs and LANs. Users can also write their own applications utilizing OSI/AS or OSI/TS and use the Session Layer or Transport Layer connections as "pipes" through which data can be passed.

Users that require high-speed data exchange over a restricted area can use the OSI/AS and OSI/TS products on a LAN. If a network has to cover a wide geographical area at medium speed, the users can choose a WAN. Users often have both requirements and use a combination of LANs and WANs.

## Accessing OSI Networks

OSI/TS can use two types of network service, a Connectionless Network Service supporting LAN operation and a Connection-oriented Network Service supporting WAN operation. Users of OSI/AS need not be aware of whether the actual network being used is a WAN or a LAN.

The configuration of OSI/TS, TLAM, and X25AM differs depending on whether LAN or WAN operation is used. If LANs are selected, Transport Protocol Class 4 is mandatory, as is the use of the Connectionless Network Service and Logical Link Control Type 1 (LLC1). The choice of LAN type is limited to CSMA/CD or Token Bus. For WAN operation, the user has a choice of all five Transport Protocol classes above the X.25 WAN. The user can select a 1980 standard X.25 or a 1984 standard X.25; X25AM supports both. Table 1 shows the OSI standards associated with the various OSI layers.

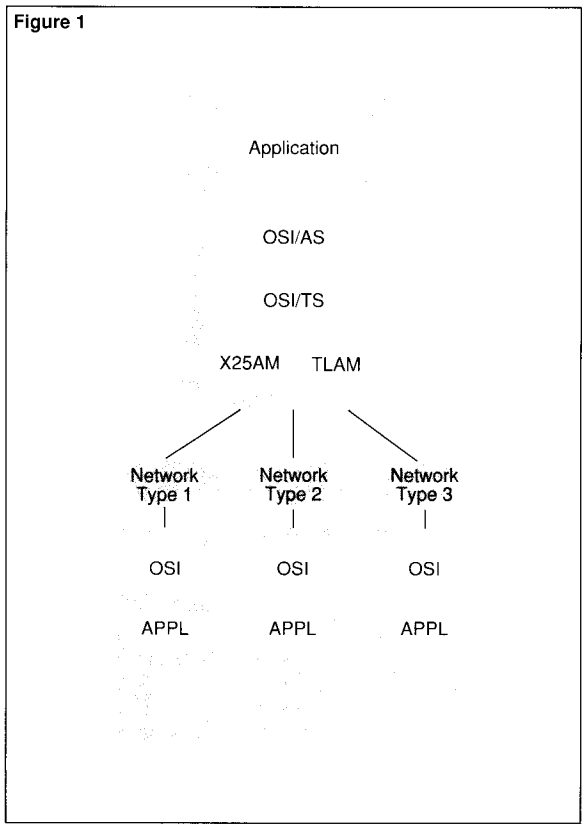



**Figure 1.**
*OSI access to Tandem applications.*

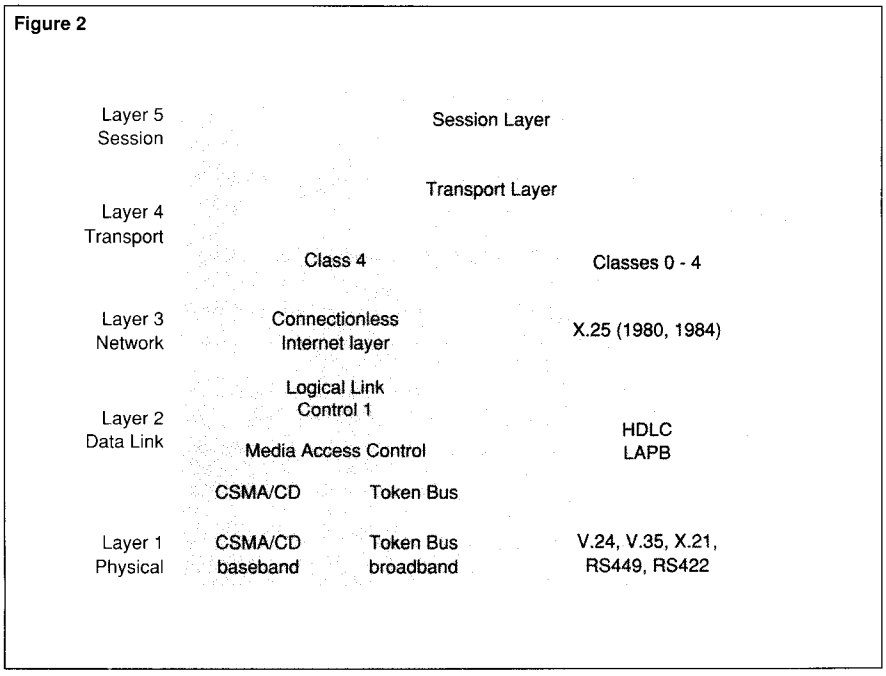



**Figure 2.**
*OSI layers 1 through 5.*

Figure 3

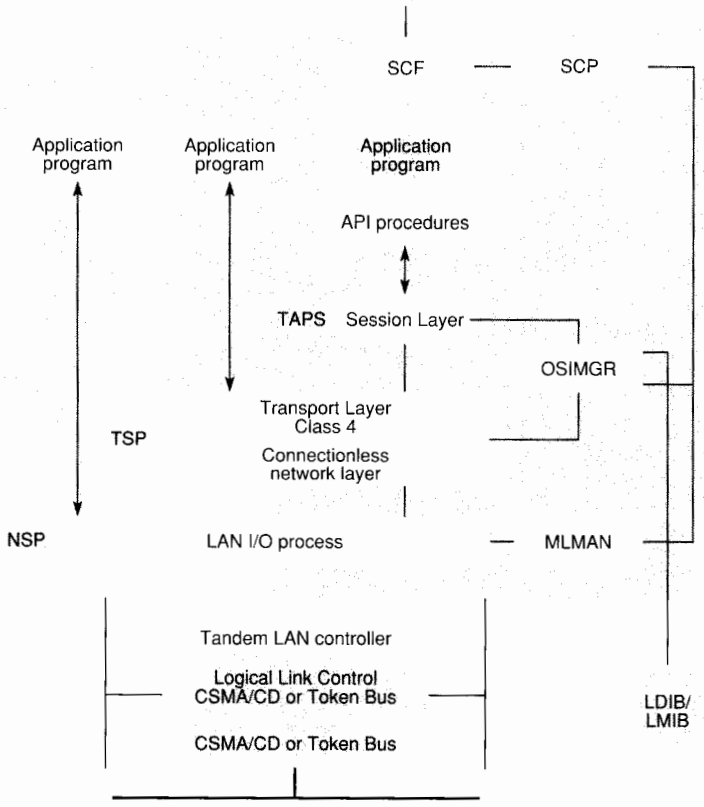



**Figure 3.**
*OSI implementation on the Tandem system for LANs.*

**Table 1.**
The standards associated with the various OSI layers.

| Layer | Standard |
|---|---|
| Session | ISO 8326, 8327 |
| Transport | ISO 8072, 8073 |
| Connectionless Network Service | Connectionless Network layer corresponding to ISO 8348/AD 1, 8473 |
| Data Link (LANs) | Logical Link Control 1 corresponding to ISO 8802/2. Media Access Control: if CSMA/CD then ISO 8802/3; if Token Bus then ISO 8802/4. |
| Physical (LANs) | If CSMA/CD then ISO 8802/3; if Token Bus then ISO 8802/4 |
| Network (X25) | CCITT X25 (1980) and some CCITT X25 (1984) features |
| Data Link (X25) | CCITT X25 HDLC LAPB |
| Physical (X25) | Many types of electrical interface are supported. CCITT V24, CCITT V35, CCITT X21, EIA RS449, EIA RS442 |

## The Tandem OSI Implementation

As shown in Figures 3 and 4, the OSI layers up through the Session Layer (OSI/AS) are implemented on the Tandem system. Figure 3 shows the configuration for LANs, and Figure 4 shows the configuration for WANs. The Tandem processes are as follows:

- The *Tandem Session Service provider* (TAPS) process implements the OSI Session Layer.
- The *Transport Service provider* (TSP) process implements the OSI Transport Layer. If LANs are used, the TSP process also provides the OSI Connectionless Network Service.
- The *Network Service provider* (NSP) process contains the LAN I/O process TLAM or the Tandem X25AM process.
- The *OSI Manager* (OSIMGR) process enables the management and configuration of OSI/AS, OSI/TS (when used with OSI/AS), and the local directories.

For a LAN, the Tandem I/O controllers include the LLC1 capability as well as the media access control (MAC) for CSMA/CD or Token Bus. For an X.25 network, the Tandem controllers usually implement the link access procedure—balanced (LAPB) protocol and provide the physical connectivity.

Typically, a system contains many copies of each of the three process types (TAPS, TSP, and NSP) and one copy of the OSIMGR for each OSI end system. (See Figure 5.) The main advantage of splitting the layers is flexibility. The OSI subsystem is merely another user of the I/O processes and uses their published programmatic interfaces, which allows the I/O products to be developed and updated independently of the OSI products. This flexibility makes it much easier to add new I/O processes and thus new subnetworks. For example, users can easily add a new LAN access manager supporting a new LAN type.

OSI standards can be divided into two logical groups:

- *Application-related services* include the user application, Application Layer, Presentation Layer, and Session Layer.
- *Transport-related services* cover the Transport Layer to the Physical Layer.

Because these two groups are logically separate, it makes good design sense to separate them physically as well. This allows the Transport Services and Application Services to work autonomously and aids the distribution of the OSI subsystem across processors.

The split structure facilitates multiplexing transport connections onto X.25 virtual circuits. The transport protocol allows multiple transport connections to be carried over one network connection (virtual circuit). Thus, one can have many TSP processes using the network connections from only one NSP process. For example, each of the 255 virtual circuits supported by an X25AM (NSP) process could carry ten transport connections. This can be achieved by having ten TSP processes above a single X25AM process, each TSP process supporting 255 transport connections.

## System Configuration and Management

OSI/AS and OSI/TS operate within the Tandem Distributed Systems Management (DSM) environment. The OSIMGR process, which is part of OSI/AS, implements some OSI subsystem management functions as well as the OSI directory database management functions. If only OSI/TS is used, the OSIMGR is not used for OSI system management. The OSI subsystem (OSI/AS, OSI/TS, X25AM, and TLAM) is configured through the DSM Subsystem Control Facility (SCF) user interface that supplies input data to OSIMGR. In this way the user, together with SCF, sets up the OSI end-system addressing structure in a directory and configures the processes needed to implement the OSI end system.

A local directory information base (LDIB) contains OSI addressing information through which applications can select sessions to other OSI applications simply by specifying their names. The name-to-OSI address mapping is done in the LDIB. In addition, a local management information base (LMIB) holds information describing the TAPS, TSP, and NSP process structure and supports the OSI addresses set up in the LDIB. The LDIB and LMIB are both logical structures and are required only when the Session Layer is being used and not when the application interfaces directly to the Transport Layer or the Network Layer.

Figure 4

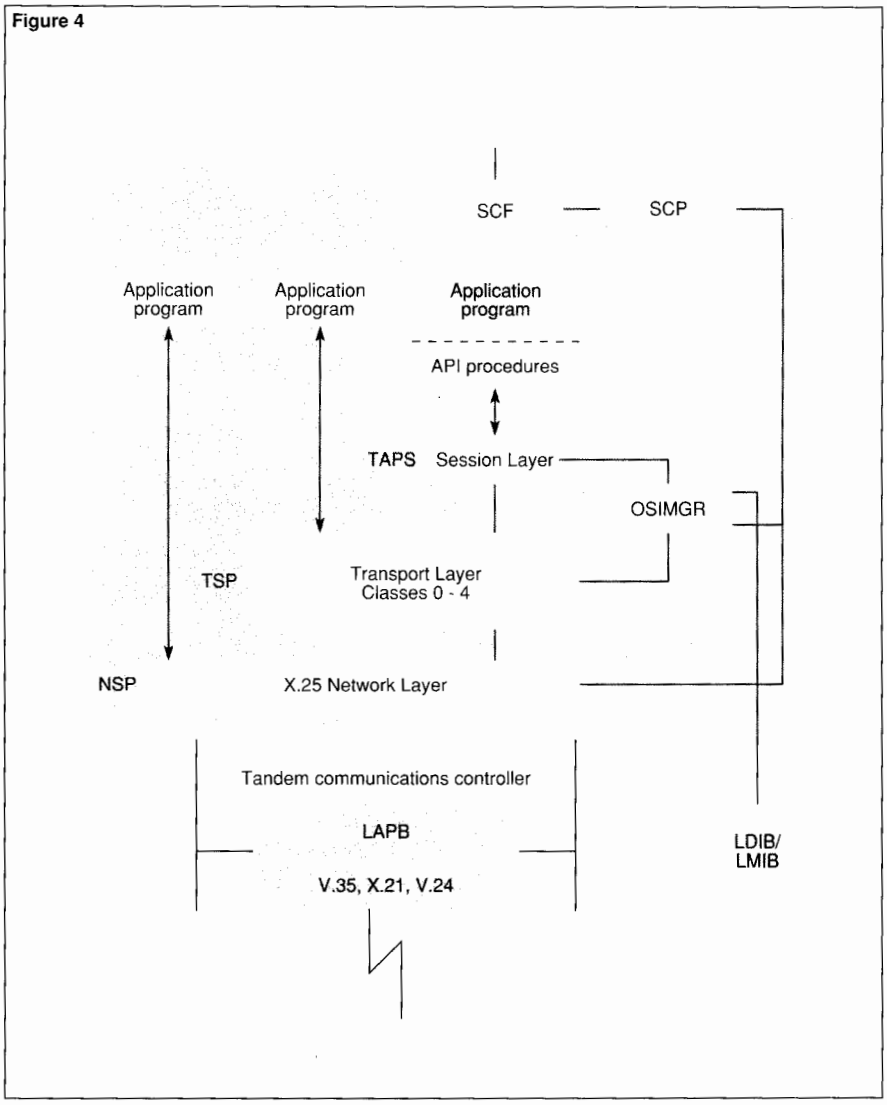



**Figure 4.**
*OSI implementation on the Tandem system for WANs.*

Figure 5

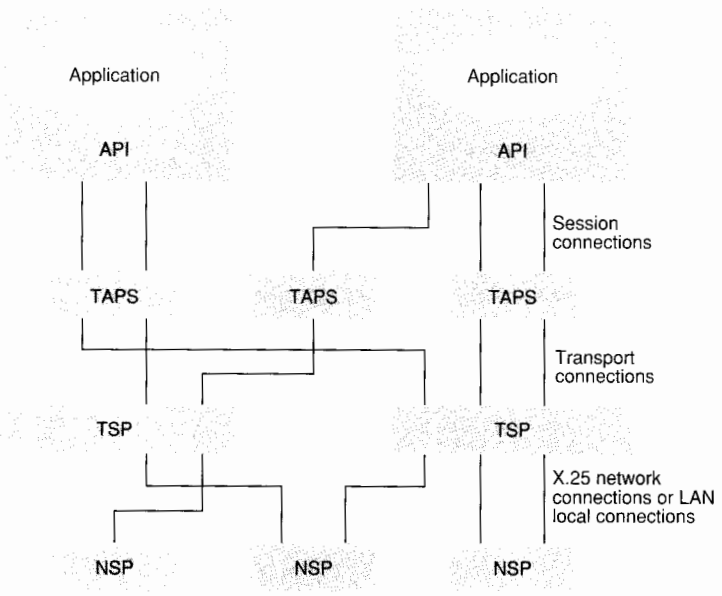



**Figure 5.**
*A typical system containing multiple copies of TAPS, TSP, and NSP processes.*

Management functions for the NSP processes are handled through their own management interfaces. Thus, the LAN manager process, MLMAN, handles management functions for the LAN I/O process and LAN controllers. X25AM handles management functions for the X.25 I/O process. SCF provides the user interface for X25AM, MLMAN, and OSIMGR, and it interfaces to these processes through the DSM Subsystem Control Process (SCP).

A number of completely separate OSI end systems, each with its own OSIMGR process and its own unique addressing domain, can operate within the same Tandem system. Furthermore, a Tandem OSI end system can be distributed over a Tandem EXPAND™ data communication software network.

## Programmatic Interfaces

The application interfaces to the Transport Layer, the LAN I/O process, and the X.25 I/O process provide a message interface based on Tandem GUARDIAN 90 file system procedures. These procedures operate on subdevices representing transport connections, X.25 virtual circuits, or LAN link-level ports.

In contrast, the Session Layer application programming interface (the OSI/AS API) is implemented using specialized OSI Session Layer procedures contained in the Tandem system library. This complex subsystem looks simple to the application programmer using the API. The user need not be concerned about the complexities of OSI addressing or the underlying OSI subsystem process structure. The user simply has to ask that a session be set up between the local application and a remote application identified by name. The OSIMGR resolves the name into an OSI address and appropriately configures the various TAPS, TSP, and NSP processes.

## Tandem OSI Components

The remainder of this article examines the various parts of the OSI/AS and OSI/TS products in detail. It discusses the Tandem application programming interface (API), the concept of OSI service primitives, and the use of local directories, and it examines a session establishment. It also describes the functions of OSI layers 1 through 5 as defined by the OSI standards and discusses which options and profiles are implemented by Tandem in the OSI/AS, OSI/TS, X25AM, and TLAM products.

## OSI/AS Application Programming Interface

The API under consideration is the Tandem interface between applications and the Session Layer. It is based on a set of procedures resident in the system library that are called by the application program. These API procedures perform interprocess I/O operations to the OSI processes on behalf of the application while hiding the I/O from the application. In fact, one procedure call can cause many interprocess messages to be sent between the processes in the OSI subsystem, without the application's knowledge. Table 2 contains examples of API procedure calls.

The application programmer need not be aware of the processes in the OSI subsystem, as would be necessary with the more common file system type of programmatic interface. For example, when an application issues a session connect request (APS_ASSOC_CONNECTREQ_) to set up a session connection with a remote application, it can supply local and remote application names in the call to the API procedure. The API procedure obtains the OSI address information from the LDIB through the OSIMGR and then the OSIMGR configures the relevant TAPS, TSP, and NSP processes to allow processing of the connect request by the OSI system.

To a large extent, the API procedures mirror the OSI Session Service primitives[2] and allow operation on the Session Layer connection end point identifier (CEPI) once the connection has been set up. As defined by OSI, the CEPI is similar in function to the Tandem file number used after a file name has been opened.

A feature of the API is the ability to perform nowait I/O operations on a CEPI. For example, a programmer who issues an APS_ASSOC_CONNECTREQ_ call to initiate a session connection does not have to wait for it to be completed, but can proceed with other operations. The programmer can then call the MFM_AWAITIOX_ procedure to wait for the CONNECT or any previously initiated I/O operation to be completed. Whenever a program uses the API, it must use the MFM_AWAITIOX_ procedure to wait for completions in place of the AWAITIO or AWAITIOX procedures.

**Table 2.**
Examples of names of API procedure calls.

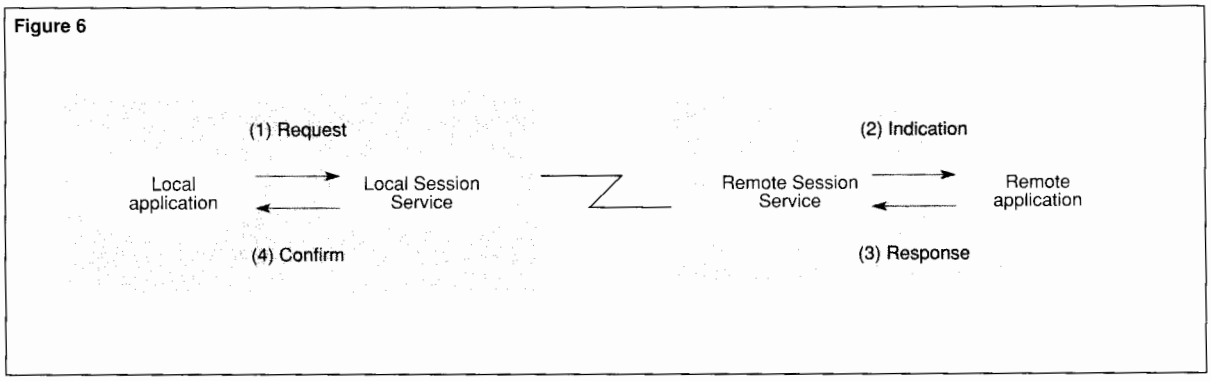
| API procedures | Session Service primitives | |
|---|---|---|
| **Examples of procedures for initiating requests** | | |
| APS_ASSOC_CONNECTREQ_ | S-CONNECT | REQ |
| APS_ACTIVITY_STARTREQ_ | S-ACTIVITY-START | REQ |
| APS_ACTIVITY_ENDREQ_ | S-ACTIVITY-END | REQ |
| APS_RESYNCREQ_ | S-RESYNCHRONIZE | REQ |
| APS_TOKEN_PLEASEREQ_ | S-TOKEN-PLEASE | REQ |
| **Procedures for receiving indications** | | |
| APS_EVENT_RECEIVE_ | | |
| APS_STATUS_ | | |
| Also other procedures depending on primitive received | | |
| **Examples of procedures for sending responses** | | |
| APS_ACTIVITY_ENDRSP_ | S-ACTIVITY-END | RSP |
| APS_SYNC_MINORRSP_ | S-SYNC-MINOR | RSP |
| APS_ASSOC_CONNECTRSP_ | S-CONNECT | RSP |
| **Procedures for receiving confirm primitives** | | |
| APS_EVENT_RECEIVE_ | | |
| APS_STATUS_ | | |
| Also other procedures depending on primitive received | | |

The new OSI/AS API has several advantages:

- The configuration of the OSI subsystem is hidden from the application.
- The application programmer does not have to worry about complicated OSI addresses and need only use OSI application names.
- The API procedures closely match the OSI Session Service primitives, so that a programmer familiar with these primitives will find the API easy to use.
- Because the API procedures closely reflect the OSI Session Service primitives, it is likely that third-party software already written to access the Session Layer will be structured to call procedures similar to the API procedures. This facilitates the porting of such software to the Tandem system.
- The OSI subsystem structure can be changed without changing the user's application programs.

[2]Primitives are the means of communication between applications and the OSI Session Layer.

Figure 6.
*The four types of Session Layer primitives.*




## Session Primitives

As shown in Figure 6, the primitives that provide communication between applications and the OSI Session Layer are of four types:

- A *request primitive* is invoked by API when a local application initiates a Session Layer operation.
- An *indication primitive* indicates to the remote application that the local application has initiated an operation. In some cases it indicates that the remote Session Service provider has initiated an operation.
- A *response primitive* carries the response to an indication primitive. Not all indications require a response.
- A *confirm primitive* completes the cycle by providing the remote application's response to the request that was sent. Confirm primitives can carry a positive or negative response.

Information is sent to the remote application by invoking request or response primitives. Information is received from the remote application or Session Service provider (TAPS) by indication or confirm primitives.

There are several classes of API procedures, including those that generate request and response primitives, those that detect the arrival of indication and confirm primitives, and those that allow the application to recover information from indication or confirm primitives. An API procedure exists for every request and response primitive. Calling one of these API procedures generates the primitive and causes it to be sent to the remote application.

To detect the arrival of an indication or confirm primitive, the application must call APS_EVENT_RECEIVE_ using the CEPI to identify the connection. When the APS_EVENT_RECEIVE_ operation is complete, the application is alerted to the fact that information has arrived. The application must then call APS_STATUS_ to identify the information. The next stage of operation involves calling a specific API procedure to recover details of the indication or confirm primitive and any user data.

## Operations below the API Level

To understand how the API shields the application from the complexities of OSI addressing and the OSI subsystem structure, it is helpful to examine what happens when an application issues an APS_ASSOC_CONNECTREQ_ call with the local and remote application names as parameters.

The process is as follows. The API issues a request to OSIMGR that includes the application names. The OSIMGR uses the LDIB to convert the local and remote application names into OSI addresses. (See Figure 7.) The LDIB also provides the subnetwork point of attachment (SNPA), which specifies the point at which a system is attached to a real subnetwork.

The OSIMGR uses the LMIB to select the appropriate TAPS, TSP, and NSP processes, depending on the corresponding OSI addresses specified. Thus, the TAPS process selected depends on the local transport selector (TSEL), the TSP process depends on the local Network Service access point (NSAP), and the NSP process depends on the local SNPA.

The system manager specifies these assignments by means of the SCF ADD ENTRY command at the time the system is configured. A one-to-many relationship exists between TAPS and the local TSEL. That is, one TAPS process can serve many TSELs, but each TSEL can only be served by one TAPS process. The same one-to-many relationship also exists between TSP and the local NSAP, and between NSP and the local SNPA.

The LMIB also provides connection-specific configuration parameters. These parameters are specified by means of a PROFILE object at the time the system is configured. A PROFILE is a group of parameters whose values typically depend on either the local or destination OSI address. Many different PROFILEs can be configured, which allows the use of different values (such as timers), depending on which remote system is addressed on a particular connection.

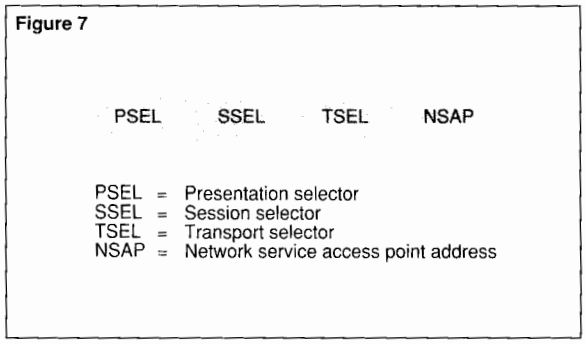



**Figure 7.**
*OSI addresses provided by LDIB.*

PROFILEs are currently defined for the Transport Layer (Layer 4) and Network Layer (Layer 3). PROFILEs are defined by the SCF ADD PROFILE command. The following is an example of a Layer 4 PROFILE:

ADD PROFILE $OMGR.#L4.*profile-name*,

CLASS 4, CONNECTTIMEOUT 180,

DISCONNECTTIMEOUT 60,

MULTIPLEX ON, TPDUSIZE 1024 ...

Layer 4 PROFILEs are assigned to local and remote NSAP addresses; Layer 3 PROFILEs are assigned to local and remote SNPAs.

At the time a connection is established, the OSIMGR selects a PROFILE. If a PROFILE is defined for the remote address, that PROFILE is used on the connection. If no PROFILE is defined for the remote address but a PROFILE is defined for the local address, the latter PROFILE is used on the connection. If neither the remote nor the local address has a PROFILE assigned to it, the layer-specific default parameter values are used on the connection.

After all processes and connection-specific parameters are selected, the OSIMGR configures a TSP subdevice using the Layer 4 and Layer 3 configuration parameters. The OSIMGR also allocates a TAPS subdevice name, but it does not configure the TAPS subdevice. The TAPS subdevice is created when the API opens it.

Next, the OSIMGR returns the OSI addresses, the TAPS subdevice name, and the TSP subdevice name to the API procedure in the application process. The user request and the OSIMGR-supplied information is then passed to the TAPS process, which requests a transport connection from the TSP.

If necessary, the TSP configures an NSP subdevice and, if X.25 is being used, requests a network connection. The NSP process establishes a network connection and returns confirmation to the TSP process that the connection has been made. (If multiplexing is being used and a suitable network connection exists, the existing network connection is used.) If LANs are used, the Network Service is connectionless.

After the TSP confirms the connection, TAPS writes the session connect request to the now open transport connection. When this operation is completed, control is returned to the application.

Finally, the remote system receives the connect request as a connect indication and generates a connect response that will be returned to the local TAPS process. If the local application issued an APS_EVENT_RECEIVE_ call, the arrival of the connect confirmation causes the APS_EVENT_RECEIVE_ call to be completed. This indicates to the application that the connect has been accepted.

**Session Layer**

The Session Layer is implemented in the TAPS process. The Session Layer is application oriented, unlike Layers 1 through 4, which are oriented toward the transfer of data across networks. The Session Layer allows cooperating applications to organize and synchronize their dialogue and to manage data exchange. The application can divide work into logical groupings called *activities* and can manage those groupings. The Session Layer also provides mechanisms to synchronize and resynchronize streams of data, and it establishes the manner in which data is exchanged; that is, whether in a two-way alternate (half-duplex) mode or a two-way simultaneous (full-duplex) mode.

The OSI Session Layer protocol is organized into a set of functional units that group together related services. When an OSI session connection is established, negotiation takes place with the remote application to determine which functional units to use. Only functional units that have been negotiated can be used. Eleven functional units are supported by the Tandem OSI/AS product:

- *Kernel* contains functionality to make and release connections and to transfer and receive data.
- *Half Duplex* allows operation in half-duplex mode over a connection. In this mode, the data token indicates which side can send data.
- *Duplex* allows operation in full-duplex mode. No data token is needed. A particular connection can use half-duplex or full-duplex mode, but not both.
- *Typed Data* allows the sending of data over a connection, ignoring any token restrictions.
- *Capability Data Exchange* allows the sending and receiving of data while not in an activity. It can be used only if activity services are available but no activity is in progress. It is subject to data-token restrictions if operation is to be in half-duplex mode.
- *Minor Synchronize* permits the application to define minor synchronization points in the communication flow to a remote application. In this case, the sender of the request does not have to stop data transfer while waiting for a confirmation.
- *Major Synchronize* enables the application to define major synchronization points in the communication flow to a remote application. In this case, the sender will stop data transfer while waiting for a confirmation.
- *Negotiated Release* allows negotiation with the remote application to ensure an agreed-upon release of a connection. Without this functional unit, orderly release is still possible, but a release request cannot be refused.
- *Resynchronize* allows the sending of resynchronize requests. Such a request resets the connection to a defined state after an error or lack of response by the local application, the remote application, or the service provider.
- *Exceptions* permit applications and the service provider to issue exception reports.
- *Activity Management* allows the local application and the remote application to define logical pieces of work as activities and to organize the flow of work accordingly.

At present, the OSI/AS does not support two features, the Session Layer Expedited Data functional unit and the Extended Concatenation capability. In practice, few Tandem installations require Session Layer Expedited Data and even fewer require Extended Concatenation.

The Session Layer functional units provide mechanisms that applications can use in the conversation between the local and remote application. For example, the Session Layer itself cannot provide completely secure message transmission for the CCITT-defined X.400 messaging service, but a specialized X.400 application (the Reliable Transfer Service) can provide secure message transfer by using functional units (such as Activity Management and Resynchronize) in the Session Layer.

**Table 3.**
The five classes of service defined by the OSI Transport Layer protocol.

| Class of service | Recovery |
|---|---|
| **Non-multiplexing** | |
| 0 | No recovery from Network-signaled errors. No recovery is attempted; the connection is released. |
| 1 | Error recovery from signaled errors. |
| **Multiplexing** | |
| 2 | No recovery from Network-signaled errors. No recovery is attempted; the connection is released. |
| 3 | Error recovery from Network-signaled errors. |
| 4 | Error recovery from Network-signaled and unsignaled network errors. |

## Transport Layer

The Transport Layer provides for the reliable transfer of data between Transport Service users, who may be interconnected through a number of intervening subnetworks. To the Transport Service user, the Transport Layer provides a completely reliable data transfer capability, regardless of the quality of service provided by the Network Layer. The Transport Layer contains procedures to recover data lost in transmission, received out of sequence, or corrupted; it can recover from certain types of network errors, both signaled and unsignaled. This layer also handles flow control.

The OSI Transport Protocol defines five classes of service that provide different types of multiplexing capability, error detection, and recovery. (See Table 3.) The Tandem OSI/TS product supports the following Transport Service primitives:

T-CONNECT REQUEST

T-CONNECT INDICATION

T-CONNECT RESPONSE

T-CONNECT CONFIRM

T-DATA REQUEST

T-DATA INDICATION

T-EXPEDITED DATA REQUEST

T-EXPEDITED DATA INDICATION

T-DISCONNECT REQUEST

T-DISCONNECT INDICATION

The OSI service primitives cannot necessarily be mapped onto a single GUARDIAN 90 file system call. The subdevice is the application's basic unit of access, and each subdevice has the potential of being mapped onto a transport connection.

Before a subdevice can be used, the application must open it by name. SETMODE and SETPARAM commands can then be issued to alter its operational characteristics. WRITE, READ, WRITEREAD, or CONTROL commands can be issued to initiate or release a connection and to perform data-transfer operations on the subdevice. When the application no longer needs access to a subdevice, it can terminate the conversation with a CLOSE command.

An application can open multiple subdevices, and multiple applications can use the same subdevice if it is opened with shared access. Subdevice usage can be divided up into four phases:

- *Configuration*. Operational parameters are established through DSM or by SETMODE and SETPARAM commands.
- *Connection Establishment*. The subdevice is mapped onto a transport connection (T-CON) and the T-CON is established.
- *Data Transfer*. Data is transferred on the transport connection.
- *Connection Release*. The connection is cleared and the subdevice freed.

The following conformance statements can be made for the Tandem Transport Layer implementation. First, it supports classes 0 through 4. If a LAN is used with a Connectionless Network Service, only Class 4 can be used. Second, splitting is not performed in Class 4. Third, the system is capable of both initiating and responding to Connect Request TPDUs (transport protocol data units). Finally, the maximum TPDU size implemented is 8192 octets.

Several optional procedures are not implemented in the Tandem OSI products. The Transport process always employs the flow control procedure in Class 2. Class 1 does not use receipt confirmation; the Transport process will send Data Acknowledgment TPDUs to acknowledge outstanding TPDUs. Class 1 does not use network expedited; the Transport Layer will send TPDUs as normal data requests to the Network Layer.

### Connectionless Network Service in the Network Layer

When a LAN is used, the Connectionless Network Service is provided as part of the TSP process. The Connectionless Network Service has no API. The Tandem system acts as a Network Layer end system and not as an intermediate system; the Connectionless Network Protocol is responsible for end-to-end communication across any number of intervening subnetworks. The Connectionless Network Service allows data-unit transfer without the establishment of a network connection.

The N-UNITDATA, the OSI service element that performs this data transfer, provides for the data transfer of a discrete Network Service data unit (NSDU). No other service elements are required for any additional functionality. For example, NSDUs can be delivered out of sequence; thus, the Transport Layer Class 4 is used above the Connectionless Network Service to provide the data integrity required.

Tandem supports the following Connectionless Network Protocol functions:

- Protocol data unit (PDU) composition.
- PDU decomposition.
- Header format analysis.
- PDU lifetime control.
- Route PDU.
- Discard PDU.
- Error reporting.
- Header error detection.

This implementation supports the Inactive subset but not the Nonsegmenting subset. The use of the Inactive subset is included to allow interworking with certain existing OSI implementations from other vendors. Whenever possible, the Inactive subset should not be selected.

## OSI Acronyms

| | |
|---|---|
| API | Application Programming Interface |
| CCITT | International Telegraph and Telephone Consultative Committee |
| CEPI | Connection end point identifier |
| CSMA/CD | Carrier sense multiple access with collision detection |
| DSM | Distributed Systems Management |
| FTAM | File Transfer, Access, and Management |
| LAN | Local area network |
| LAPB | Link access procedure—balanced |
| LDIB | Local directory information base |
| LLC1 | Logical Link Control Type 1 |
| LMIB | Local management information base |
| MAC | Media access control |
| MLMAN | LAN manager process |
| NSAP | Network Service access point |
| NSDU | Network Service data unit |
| NSP | Network Service provider |
| OSI | Open Systems Interconnection |
| OSI/AS | Open Systems Interconnection/Application Services |
| OSI/TS | Open Systems Interconnection/Transport Services |
| OSIMGR | OSI Manager |
| PDU | Protocol data unit |
| SCF | Subsystem Control Facility |
| SCP | Subsystem Control Process |
| SNPA | Subnetwork point of attachment |
| TAPS | Tandem Session Service provider |
| TLAM | Tandem LAN Access Method |
| TPDU | Transport Protocol data unit |
| TSEL | Transport selector |
| TSP | Transport Service provider |
| UI | Unnumbered information |
| WAN | Wide area network |
| X25AM | X.25 Access Method |
| XID | Exchange identification |

### LLC1 in the Data Link Layer

The OSI/TS product uses the TLAM product to provide the necessary LAN connectivity. The LAN I/O process allows connection of the LAN controller to the Tandem system; the LAN controller runs LLC1 and MAC and supports the Physical Interface. The LAN I/O process also provides a programmatic interface to allow applications to access the LLC1 layer. For more information on this user interface, refer to the Tandem MULTILAN™ and TLAM manuals.

The purpose of any Data Link Layer is to provide a reliable communication path between two devices sharing a transmission medium. The LLC1 allows for the sending and receiving of LLC frames with no form of acknowledgment needed to ensure delivery. It supports point-to-point, multipoint, and broadcast addressing and multiplexing. LLC1 allows for two service primitives, L-DATA REQUEST and L-DATA INDICATION. The protocol uses three unnumbered frame formats: unnumbered information (UI), exchange identification (XID), and the TEST frame.

### MAC in the Data Link Layer

The Tandem TLAM product supports two types of media access control: CSMA/CD and Token Bus. MAC manages the access of the media so that controlled communication can take place.

***CSMA/CD.*** The simplest form of MAC is CSMA/CD, which operates on a bus topology. This scheme requires that a station trying to send data must first determine if the medium is busy. If it is, the station does not send but tries again a little later. If the medium is not busy, the station sends. With this scheme, it is possible for two stations to send at the same time, thus causing a collision. If a collision occurs, both stations abort the send operation, wait a random amount of time, and resend.

***Token Bus.*** In this scheme, the stations on the bus, or tree, form a logical ring; that is, the stations are assigned to logical positions in an ordered sequence with the last member of the sequence followed by the first. A control frame known as the *token* regulates the right of access. The station receiving the token is granted control of the medium for a specified time. The station may then transmit frames, poll stations, and receive responses. Once the station has completed its communication, it passes the token to the next station in the logical sequence.

### LANs in the Physical Layer

The CSMA/CD LAN uses baseband, 50-ohm, coaxial cable; digital signals are transmitted using Manchester encoding at 10 megabits per second (Mbps). Tandem supports thick or thin Ethernet cabling. CSMA/CD LANs are connected to Tandem systems through type 5600, 3611, and 3613 controllers.

The Token Bus standard specifies three alternative physical options. All three use 75-ohm, CATV coaxial cable and analog (RF) signalling. Tandem systems connect to broadband coaxial cable and transmit at 10 Mbps.

### X.25

The Tandem OSI products use X25AM for connecting OSI end systems over a WAN. A WAN is a network that can interconnect geographically dispersed systems, terminals, and workstations over distances greater than a few kilometers. Such a network can span the globe.

X.25 provides a reliable, connection-oriented Network Service. For communication to occur, virtual circuits must be established with the network data circuit terminating equipment. The Tandem implementation of X.25 contains 1980 and some 1984 functionality. Most important, X25AM supports NSAP addressing. Note that a GUARDIAN 90 WRITEREAD-type programmatic interface exists to allow applications to interface directly to X25AM.

The X.25 Data Link Layer is high-level data link control LAPB. It provides a reliable data path between the two Data Link Layer entities. X.25 supports the following electrical interfaces: V.24, V.35, X.21, RS442, and RS449.

## Conclusion

OSI/AS and OSI/TS allow heterogeneous workstations and systems to interwork with Tandem systems using software that supports OSI standards. The OSI software can use WANs or LANs, allowing users to geographically distribute their processing or set up systems that interwork over a limited distance using high-speed connections. With OSI/AS and OSI/TS, a larger number of users can take advantage of unique Tandem software capabilities, including the high-performance NonStop SQL database system, transaction processing, and query processing.

**Acknowledgments**

The author would like to thank the following people for their valuable help in the preparation of this article: Chris Annetts, Doug Bergh, Huy Truong, Wilfried Kruse, John Lemon, Andrew Dunn, and Mike Farrant.

**References**

CCITT X.25. 1984. Interface between Data Terminal Equipment (DTE) and Data Circuit Terminating Equipment (DCE) for terminals operating in the Packet Mode and connected to public data networks by dedicated circuit.

Dunn, A. 1989. The OSI Model: Overview, Status, and Current Issues. *Tandem Systems Review*. Vol. 5, No. 1. Part no. 18662.

ISO 8072. 1986. Open Systems — Transport Service Definition.

ISO 8073. 1986. Connection Oriented Transport Protocol Specification.

ISO 8073. 1987. PDAD2 Connection Oriented Transport Protocol Specification — Addendum 2: Class 4 Operation over the Connectionless Network Service.

ISO 8326. 1987. Basic Connection Oriented Session Service Definition.

ISO 8327. 1987. Basic Connection Oriented Session Protocol Definition.

ISO 8348. 1987. AD1 Network Service Definition — Addendum 1: Connectionless Mode Transmission.

ISO 8473. 1988. Protocol for Providing the Connectionless-mode Network Service, with Addendum 1: Provision of the Underlying Service assumed by ISO 8473 integrated.

ISO 8802/2. 1987. Logical Link Control.

ISO 8802/3. 1987. Carrier Sense Multiple Access with Collision Detect.

ISO 8802/4. 1988. Token Passing Bus.

*MULTILAN/TLAM Management and Operations Manual*. Part no. 11757. Tandem Computers Incorporated.

*MULTILAN/TLAM Management Programming Manual, Vols. 1 and 2*. Part nos. 11756 and 16224. Tandem Computers Incorporated.

*MULTILAN/TLAM Programming Manual*. Part no. 11758. Tandem Computers Incorporated.

*Tandem OSI Management Programming Manual*. Part no. 13643. Tandem Computers Incorporated.

*Tandem OSI/AS Management and Operations Guide*. Part no. 18189. Tandem Computers Incorporated.

*Tandem OSI/AS Programming Manual*. Part no. 84274. Tandem Computers Incorporated.

*Tandem OSI/TS Manual*. Part no. 84142. Tandem Computers Incorporated.

*X.25 Access Method—X25AM, Vols. 1 and 2*. Part nos. 17464 and 17465. Tandem Computers Incorporated.

**Rhod Smith** joined Tandem as a communications and networking specialist in 1982 and has 18 years' experience in the industry working in development, support, and consultancy. He became a consulting analyst in 1985 and X.400 product manager in 1986. Today he works in the European Product Marketing Group.

# NetBatch-Plus: Structuring the Batch Environment

Batch processing provides important support for the data captured by online transaction processing (OLTP). NetBatch™-Plus, Tandem's batch scheduling software available with the C20 release of the GUARDIAN™ 90 operating system, offers users a powerful, easy-to-use, full-screen interface that structures and controls a batch scheduling environment. With its database of job scheduling information, NetBatch-Plus allows users to schedule complex, multiple job runs across multiple system nodes. It automates job submission, executing jobs without operator intervention, thus reducing demands on personnel. It also permits jobs to be submitted manually. NetBatch-Plus enables users to balance the batch processing load and control the impact of batch processing on OLTP. Finally, it enhances visibility of the entire batch environment, increasing the efficiency of job management.

NetBatch-Plus integrates a new full-screen interface and database, based on the Tandem™ PATHWAY transaction processing system, with the original NetBatch batch scheduling product, designed to control the actual submission and execution of jobs. NetBatch was discussed in the April 1989 issue of the *Tandem Systems Review* (Wakashige).

This article describes how the features of NetBatch-Plus meet the scheduling requirements of batch processing. It suggests ways to exploit those features to take advantage of the benefits of Tandem systems. It also offers practical tips on how to design and organize an effective NetBatch-Plus configuration. A checklist at the end of the article summarizes the important steps in creating a NetBatch-Plus scheduling database.

In a Tandem environment, batch scheduling is defined as the structuring, submission, monitoring, and control of job requests. This article focuses on the structuring and control of batch jobs. Performance of batch processing is largely a separate issue; refer to the September 1989 issue of the *Tandem Systems Review* (Keefauver).

## NetBatch-Plus Features

NetBatch-Plus has two main components, in addition to the NetBatch portion of the product. First, the NetBatch-Plus database maintains information about job definitions, their dependencies, and batch scheduling parameters. Second, the screen-interface software allows the user to create and manage the NetBatch-Plus database, communicate with other systems in the Tandem environment, generate reports, and submit jobs. The following NetBatch-Plus features help users structure and manage their batch processing environment.

- *Defaults sets* allow the user to group related jobs and define default attributes for all jobs in the group.
- *Parametric passing of information* enables job attachments, such as ASSIGNs, PARAMs, and DEFINEs, to be passed directly to a job's EXECUTOR program when the job starts. For greater control and easy maintenance, job attachments can be defined once and then used by a virtually unlimited number of batch jobs.
- *Versatile job selection* permits the user to submit jobs individually or in a group (according to various selection criteria).
- *System visibility* allows a single terminal screen to display information about each SCHEDULER[1] and provides a gateway to other Tandem subsystems without having to exit NetBatch-Plus.
- *Management reporting* provides information about job descriptions and other information in the NetBatch-Plus database and the status of batch processing jobs. Users can produce standard reports or generate their own reports tailored to their specific needs.
- *Individually defined security* enables a system manager to define, for each user, a different level of access to the NetBatch-Plus screens. Also, access to each defaults set, job, catalog, and job attachment can be restricted to different groups of users.
- *Extensive online help* provides information describing all screens on two levels. The user can obtain a general overview of each screen or, by positioning the cursor in a particular field of the screen, can get a detailed explanation of that field.

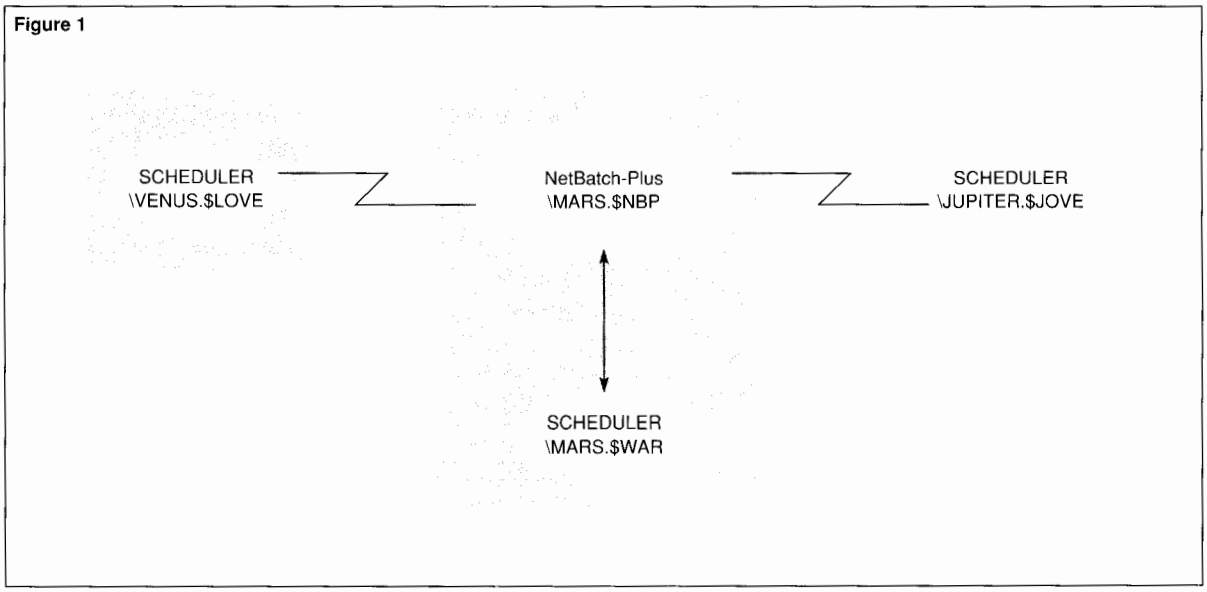
[1]The SCHEDULER is a fault-tolerant, process-pair server that maintains supervisory access to all batch processing components.

### NetBatch and NetBatch-Plus

Before the features of NetBatch-Plus are described in detail, it is useful to clarify what is new in NetBatch-Plus in relation to the NetBatch portion of the product. With NetBatch-Plus, the user defines and maintains job schedules, dependencies, and other job attributes by choosing options on the NetBatch-Plus screens. Jobs defined in NetBatch-Plus can be submitted and executed automatically, without operator intervention.

The actual execution of jobs is controlled by the NetBatch portion of the product. NetBatch SCHEDULERs control job execution according to their scheduling options. For example, a SCHEDULER can execute a job at midnight or after another job has finished processing. Each SCHEDULER is configured with JOBCLASSes and EXECUTORs. JOBCLASSes provide a convenient way to group jobs, such as payroll or high-priority jobs, that have similar processing functions. JOBCLASSes, in turn, are linked to one or more EXECUTORs, each of which is associated with a particular central processing unit (CPU). An EXECUTOR starts the job's initial process, called the EXECUTOR program, in its CPU. By designating batch jobs to run only in specified CPUs, a SCHEDULER allows the system manager to control the batch environment and ensure that batch processing does not adversely affect OLTP. See Wakashige, 1989, for a detailed discussion of NetBatch SCHEDULERs.

Figure 1.
*NetBatch-Plus can schedule batch processing across an entire network of GUARDIAN 90 systems.*




### Communicating with the SCHEDULERs

NetBatch-Plus can communicate with one or more SCHEDULERs, which can be spread over the entire Tandem network. For example, NetBatch-Plus, which is running on the system \MARS, can submit jobs to run on the SCHEDULERs named \VENUS.$LOVE, \MARS.$WAR, and \JUPITER.$JOVE. Figure 1 shows the relationship of NetBatch-Plus to three SCHEDULERs, each running on a different node.

## Defaults Sets

Defaults sets are the key to setting up a complete batch scheduling environment. A defaults set contains the default attributes for a group of batch jobs as well as static information common to their scheduling environment.

Many jobs can share a single defaults set. When creating a new job, the user can enter all the attributes of its defaults set with a single keystroke. NetBatch-Plus provides additional flexibility by allowing the user to override selected default attributes in an individual job definition. However, by defining an appropriate set of job attributes in a defaults set, all or most job attributes for an individual job can default, saving time and effort in entering new jobs.

By defining global job attributes such as a node or SCHEDULER, defaults sets make job definitions easier to maintain. When a user changes an attribute in a defaults set, in most cases the attribute is automatically changed in all the job definitions belonging to that defaults set. The attribute is not changed automatically in those job definitions that have individually redefined the attribute.

As a guideline, it is recommended that certain job attributes be maintained at the defaults set level; they should not be overridden in individual job definitions. These job attributes include the NetBatch-Plus node, SCHEDULER, JOBCLASS, job owner and security, and the EXECUTOR program that processes the job. Other recommended attributes to be maintained include a default job output (OUT) file; a default volume for job control files; job selection priority; the execution priority for the EXECUTOR program; and a start time, if appropriate. Depending on the choice of EXECUTOR program, startup parameters can also be made standard.

Normally, defaults sets can be set up for different groups of users. For example, programmers could use a defaults set for running COBOL compiles using COBOL85 as its EXECUTOR program. Another defaults set using the Tandem Advanced Command Language (TACL™) would be suitable for most operational tasks such as backup or housekeeping functions.

Figure 2 displays TRAVEL-UPD, a sample defaults set for jobs that update an example passenger-reservations database. All update jobs in this database are assigned to TRAVEL-UPD. When an update job is submitted, it inherits all default job attributes that were not redefined in the individual job definition.

**Figure 2**

```
NETBATCH-PLUS          DEFAULTS SET DETAILS                13Aug1991

SET          : TRAVEL-UPD
SCHEDULER    : \MARS.$WAR _____
JOBCLASS     : GALACTIC _____________
OWNER        : SOFTWARE.MGR ____ , _____/AOGO

COMMENT      : _____________________________
EXECUTOR     : $SYSTEM.SYSTEM.TACL _____
IN           : _____________________________
OUT          : $S.#UPDJOB _____________
VOLUME       : $DATA.DBASE _____
STARTUP      : _____________________________

SELPRI : 3              PRI    : 130
DRIVES : 2__            LINES  : _____      PAGES : ___
WAIT   : ___            AT/AF  : ___        TIME  : ___

RESTART          : _    HOLD       : ___    ANY USER SUBMIT: __
STOP ON ABEND    : Y    HOLD AFTER : ___
```

**Figure 2.**
*The defaults set shown above establishes commonly used attributes for similar groups of jobs, thereby saving time and reducing errors.*

## Creating Effective Defaults Sets

To create an effective framework for defining the batch scheduling environment, one should begin by organizing jobs into logical groups. Consider all regularly recurring tasks as well as jobs that run infrequently. Because a defaults set stores a permanent set of job descriptions, it is not appropriate to list unique jobs that will never be required again.

Note the important characteristics of each job in the environment, and arrange the data into a matrix like the one shown in Figure 3. Characteristics to consider include the node, SCHEDULER, EXECUTOR program, OUT file, and job owner.

Users should consider other characteristics essential to their site. For example, jobs can be grouped according to departments within an organization or according to functions such as payroll and accounts payable. System managers may wish to nominate defaults sets that suit the requirements of their own organizations. Once the attributes of the jobs are listed, patterns should start to emerge.

To take advantage of the easy maintenance of global job attributes, define as many attributes as possible on the Defaults Set screen. In the example shown in Figure 3, most jobs on the same node and SCHEDULER have a common EXECUTOR program, OUT file, and owner. The few exceptions can be handled by overriding the defaults on the Job Definition screen.

**Figure 3.**
*An important step in structuring a batch environment is the determination of "sets" in which to file jobs. Use a matrix similar to the one shown here to decide how many sets and which jobs to assign to a set.*

**Figure 3**

| Job Name | Node | Scheduler | Out | Exec-Prog | GUARDIAN User | Set Name |
|---|---|---|---|---|---|---|
| A | \VENUS | $LOVE | $S | TACL | SOFTWARE.RHONDA | SET 1 |
| B | \MARS | $WAR | $G | BPROC | SOFTWARE.ANNA | SET 2 |
| C | \EARTH | $LOVE | $D | ENFORM | SOFTWARE.ANNA | SET 3 |
| D | \VENUS | $LOVE | $S | TACL | SOFTWARE.RHONDA | SET 1 |
| E | \JUPITER | $JOVE | $S | TACL | SOFTWARE.DAVID | SET 4 |
| F | \MARS | $WAR | $G | BPROC | SOFTWARE.BRUCE | SET 2 |
| G | \MARS | $WAR | $G | BPROC | SOFTWARE.BRUCE | SET 2 |
| H | \VENUS | $LOVE | $S | TACL | SOFTWARE.RHONDA | SET 1 |
| I | \JUPITER | $JOVE | $S | TACL | SOFTWARE.DAVID | SET 4 |
| J | \VENUS | $LOVE | $S | FUP | SOFTWARE.RHONDA | SET 1 |

In Figure 3, Jobs A, D, H, and J are scheduled to \VENUS.$LOVE, owned by the user SOFTWARE.RHONDA, and write to the collector $S. They have TACL as their EXECUTOR program (except Job J, which uses the File Utility Program). They have enough characteristics in common to form a logical defaults set. Similarly, a second defaults set can be formed with Jobs B, F, and G, which are scheduled to run under BPROC on \MARS.$WAR, write to the collector $G, and are owned by SOFTWARE.BRUCE (except Job B, owned by SOFTWARE.ANNA). A third defaults set schedules jobs to \JUPITER.$JOVE on behalf of SOFTWARE.DAVID. A final defaults set contains only Job C, which has almost no attributes in common with the others.

## Job Definitions

Once appropriate defaults sets have been defined, users may enter their individual jobs into the NetBatch-Plus database by using the Job Definition screen. A single keystroke enters all the scheduling attributes of the defaults set into the individual job definition. Users can then redefine any attributes that are unique to the individual job.

Having supplied the job name, a job's input (IN) file is the only required attribute for a job definition. However, to take advantage of the power of NetBatch-Plus, it is better to define as many attributes as possible by taking attributes from the defaults set and, if necessary, by defining attributes specific to the job. It is recommended that the user enter a comment into the job definition to make it easier to identify later.

The job's security attribute restricts certain users for read, write, or purge access. It is recommended that write access be limited to owner only or group at most, especially for jobs owned by SUPER users. Normally, only the owner of a job can schedule it. However, the job definition can be set to allow any GUARDIAN 90 user to submit the job.

## Parametric Passing of Information

Each job can have any number of ASSIGNs, PARAMs, and DEFINEs passed directly to its EXECUTOR program on startup. The user can define this parametric information, also referred to as job attachments, in a job definition or global catalog. This allows a single ASSIGN, PARAM, or DEFINE to be used by many jobs. When NetBatch-Plus submits the job, the information is passed automatically to its EXECUTOR program. At submission time, the user does not have to specify either the parametric information or its environment, such as the subvolume in which an assigned file resides; this information is passed to the job by NetBatch-Plus. The job does not have to rely on parameters from a user's current session at a particular terminal.

When NetBatch-Plus submits a job with PARAMs, ASSIGNs, or DEFINEs, it builds a new IN file that contains the parametric information followed by a RUN command, which executes the original IN file commands. As an example, suppose the job DBASE-REPORTS, which reports on daily database transactions using the Tandem ENFORM™ query language/report formatter, has the following attributes:

EXECUTOR program:
$SYSTEM.SYSTEM.ENFORM

IN file: $DATA.TRAVEL.DBREPT

OUT file: $S.#UPDRPT

ASSIGNs: DBMASTER DAYLOG

The IN file, $DATA.TRAVEL.DBREPT, contains ENFORM source statements. When the job is scheduled, a new IN file is built containing:

```
ASSIGN DBMASTER, $DATA.DBASE.MASTER
ASSIGN DAYLOG, $DATA.DBASE.LOG1
$SYSTEM.SYSTEM.ENFORM
  /IN $DATA.TRAVEL.DBREPT,
  OUT $S.#UPDRPT/
```

The new IN file is placed in the same subvolume as the original IN file, and is purged after the job finishes executing. The EXECUTOR program is also altered to TACL. If the EXECUTOR program is already TACL, a new IN file is built by inserting the parametric information before the original TACL commands, which are then written to the new file. In addition to ASSIGNs and PARAMs, the C20 release of NetBatch-Plus supports five types of DEFINEs:

- Map DEFINEs.
- Tape DEFINEs.
- Spool DEFINEs.
- Catalog DEFINEs.
- Defaults DEFINEs.

### Maintaining Job Attachments in Catalogs

The user can define job attachments locally in a job definition or globally in a catalog file. When the information is defined in a catalog, the system manager, or other users with write access to the catalog record, can modify catalog entries once, and the new values are passed to all jobs that invoke them. By using catalogs, the system manager significantly reduces the effort and increases the flexibility of maintaining batch jobs.

For example, suppose the reservations-system database files residing in the subvolume $DATA.DBASE are moved to the volume $TRANS. If DBMASTER and DAYLOG, the two files referenced by ASSIGNs, are catalog entries, the user only has to change each entry once to ensure that all jobs using these ASSIGNs are automatically updated.

Figure 4

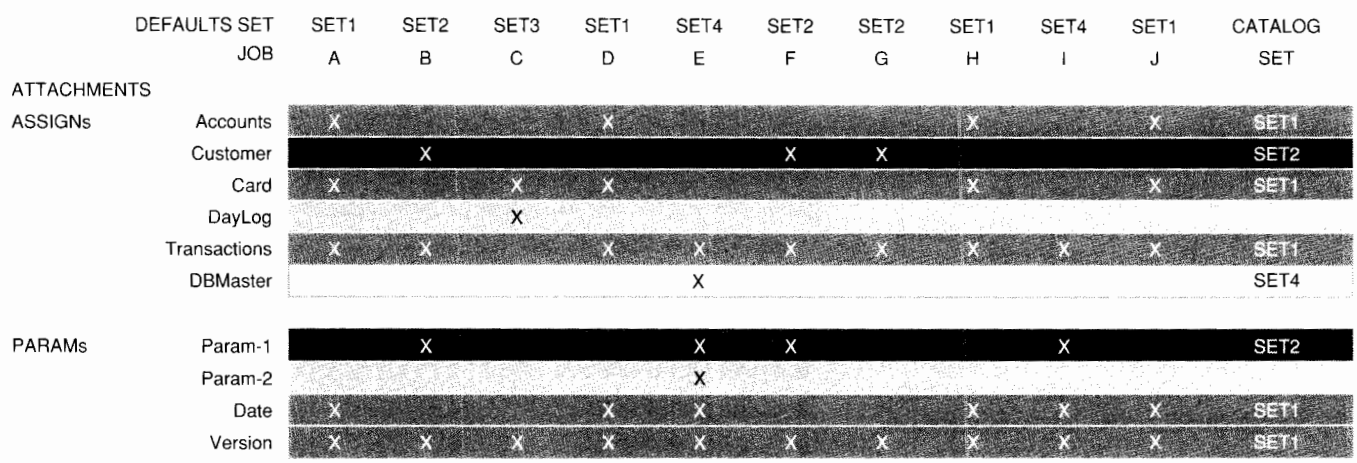



**Figure 4.**

*This matrix lists batch jobs horizontally and their job attachments vertically. Users can draw such a matrix to group job attachments and to determine which catalogs to create.*

## Creating Catalog Attachments

To determine which job attachments belong together in a catalog, one should begin by listing all the ASSIGNs, PARAMs, and DEFINEs attached to each job. The user can draw a matrix like the one shown in Figure 4, which lists all jobs horizontally and lists all the ASSIGNs and PARAMs belonging to those jobs vertically. The boxes containing an "x" identify each job attachment.

Patterns may emerge from the matrix of jobs and job attachments. The user can group shared attachments into catalog records. Most attributes in a catalog record should be identical. If there are one or two exceptions, they can be modified in individual job definitions.

Once the catalog records have been identified, the user must organize them into appropriate defaults sets. If all the jobs that share the attachments are in the same defaults set, choose that defaults set. If the attachment is an ASSIGN or MAP DEFINE, one could choose a defaults set belonging to the owner of the file referenced by the ASSIGN or DEFINE. Catalog attachments can be shared by jobs in other defaults sets.

In Figure 4, Jobs A through J have a variety of ASSIGNs and PARAMs. Some parameters are attached to jobs that belong to the same defaults set. For example, the ASSIGN ACCOUNTS is attached to Jobs A, D, H, and J, which all belong to DEFAULTS SET1. On the other hand, the PARAM VERSION is attached to all the jobs, and the decision to attach it to DEFAULTS SET1 is arbitrary. The ASSIGN DAYLOG and PARAM-2 are attached to only one job each and need not be placed in any catalog record.

## Dependencies

Job dependencies are easily entered and maintained in NetBatch-Plus. The release of a NetBatch-Plus job can depend on up to eight other jobs that must execute before the dependent job is allowed to execute. NetBatch-Plus submits the dependent job with a WAITON attribute set for each master job. The command that releases the dependent job must be inserted in each master job's IN file. This procedure is documented in the *NetBatch User's Guide*. In keeping with Tandem's commitment to transparent distributed processing, the master jobs do not have to execute in the same node or SCHEDULER as the dependent job.

## Versatile Job Selection

NetBatch-Plus offers three ways to select jobs for submission to a SCHEDULER. Two methods select jobs individually; the third selects jobs in bulk. As well as providing flexibility, this variety of job submission methods gives the system manager a way to restrict a user to a limited number of NetBatch-Plus functions and still allow that user to submit jobs.

### Submitting Jobs Individually

The first method allows a user with security access to display a job on the Job Definition screen and submit it directly to its SCHEDULER.

The second method allows a user to submit a job by selecting it from a list of job names. The Ad Hoc Job Selection screen lists jobs grouped by defaults set and provides wild-card selection mechanisms that make it easy to find jobs. The user can also choose to list only those jobs with no security restrictions; these are jobs that all NetBatch-Plus users with access to the corresponding job screen are permitted to run. These jobs are identified by setting the ANY USER SUBMIT flag to *Y*. A short description helps to identify each job listed on the screen.

### Submitting Jobs in Bulk

The BULK SUBMIT program, the third method of submitting jobs, allows the user to submit a large number of related jobs to various SCHEDULERs. After the user defines the overall batch schedule in the NetBatch-Plus database, and if the user wishes to select a daily bulk run, the BULK SUBMIT program automatically submits jobs on schedule by setting a control job to run daily at a specified time. Through its BULK SUBMIT program, NetBatch-Plus provides complete operator independence.

Users can select groups of jobs based on a range of criteria, including jobs that run on specific dates and those that run in special categories. Categories are a convenient way to group jobs such as payroll or administration jobs that are usually run together. They can also be associated with dates entered on a calendar.

NetBatch-Plus allows flexibility in choosing start times for jobs in bulk runs. The user specifies an overall start time and can also specify particular start times for individual jobs. If required, the user can override these individual jobs' start times and force all jobs in the bulk run to start at the time specified for the run.

**Figure 5**

NETBATCH-PLUS BULK JOB SELECTION CRITERIA 13Aug1991

JOB SET : TRAVEL-UPD JOB NAME: DBASE-SORT

| INclude/EXclude | CATEGORY | or | DATE | AT/AFter TIME |
|---|---|---|---|---|
| EX | | | 30Jul1991_Tue | |
| EX | | | 24Dec1991_Tue | |
| IN | | | 27Dec1991_Fri | AF 18:00 |
| EX | ENDMONTH | | | |
| EX | HOLIDAYS | | | |
| IN | TUESDAYS | | | AF 15:00 |

***Bulk Job Selection Criteria.*** For each job, the user defines the dates and categories in which the job executes when scheduled for a bulk run. A job can be assigned to as many dates and categories as necessary. The user can also explicitly exclude a particular date or category from a bulk run. An optional start time can be specified for each date and category.

For example, Figure 5 illustrates the bulk selection criteria of the batch job DBASE-SORT. It runs every Tuesday after 15:00, except on public holidays or the last day of the month. It has also been excluded from running on two Tuesdays, July 30 and December 24, 1991. On Friday, December 27, 1991, it will run after 18:00. The user has defined the categories TUESDAYS, ENDMONTH, and HOLIDAYS in the calendar, associating each category with the appropriate dates.

**Figure 5.**
*Multiple jobs can be selected to run by including or excluding categories, dates, and times.*

Figure 6

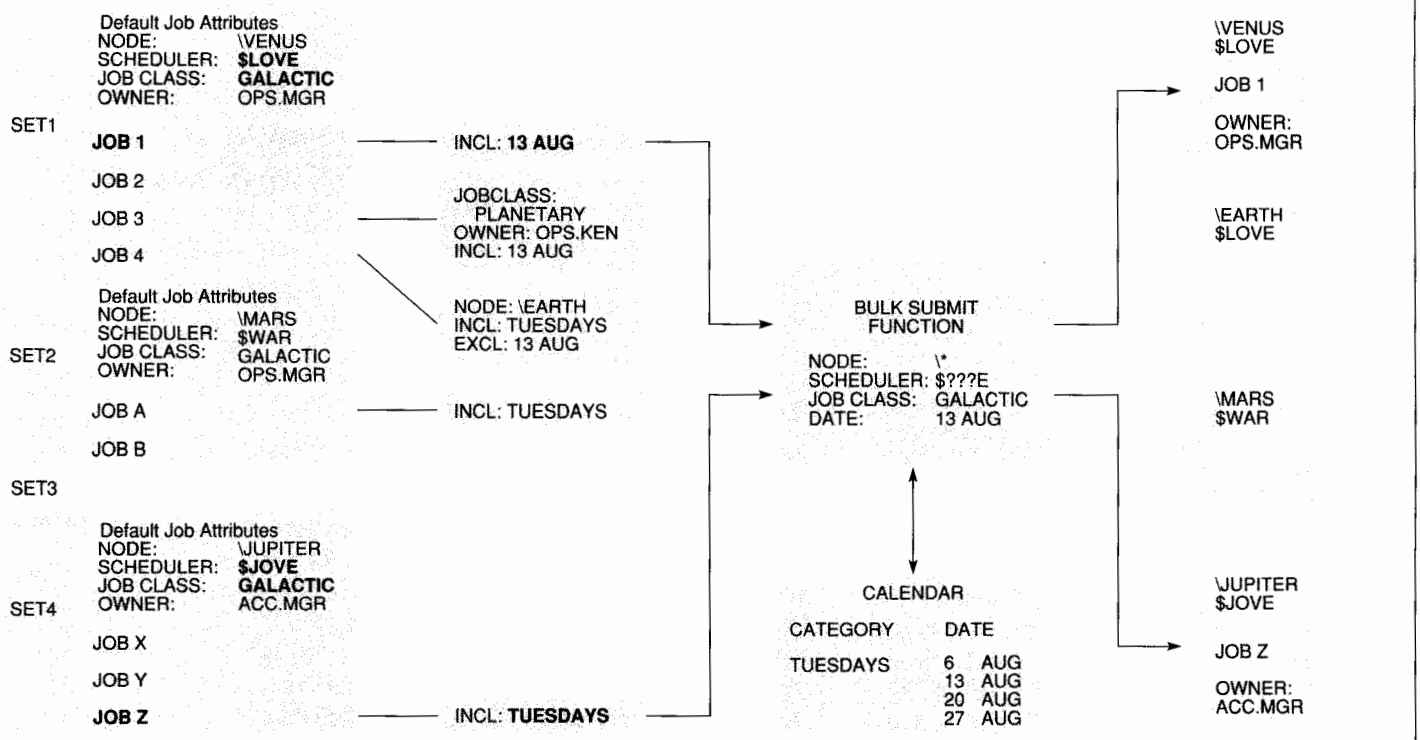



**Figure 6.**
*Multiple jobs can be selected to run based on node, SCHEDULER, JOBCLASS, and date. In the case shown above, two of nine jobs were selected to run.*

***Configuring Bulk Job Runs.*** When setting up large batch environments, users should consider carefully how they configure bulk runs of jobs. If jobs run regularly, begin by deciding on their scheduling criteria. List the dates when each job runs and the dates, such as holidays, when it should not run; list a category to which the job should belong; or list both dates and categories.

Once the run dates for each job are determined, patterns should emerge. The user can generate categories in the calendar based on those patterns. For example, certain jobs may run every Monday except public holidays. Define the category MONDAYS by using the automatic frequency generation in the calendar. Next, generate the category HOLIDAYS by explicitly defining all public holidays in the calendar.

A category does not have to map to dates in the calendar. It can identify a group of jobs that perform a similar function. For example, a category for payroll or monthly backups could be assigned to specific dates, such as the last day in each month, and generated automatically. However, a user may find it more convenient to schedule the run by explicitly selecting the category instead of having it be selected automatically in the BULK SUBMIT program's daily run. In that case, one would not enter the category in the calendar.

### Three Ways to Submit Bulk Jobs

NetBatch-Plus offers three ways to control the scheduling of bulk jobs. Once users decide to use the BULK SUBMIT program for job submission, users can combine all three types of bulk-job selection to fit their batch scheduling needs.

***Daily Run***. First, the BULK SUBMIT control job can run every day at a particular time, automatically selecting and submitting all jobs eligible to run on the current day. The control job submits each job to the specified SCHEDULER to start at the specified time. The control job itself needs to be submitted only once and continues to run at the same time every day.

The user can choose to select only jobs defined for certain SCHEDULERs or JOBCLASSes. For example, \VENUS.$* * specifies that only jobs belonging to any JOBCLASS in any SCHEDULER on the system \VENUS would be submitted. Similarly, *.$???E GALACTIC would select jobs for all nodes; each job would have a SCHEDULER with a four-character name ending in E and the JOBCLASS must be GALACTIC. The nodes and SCHEDULER names selected could include \VENUS.$LOVE and \JUPITER.$JOVE.

Figure 6 illustrates a daily bulk run for several jobs in three different defaults sets for Tuesday, August 13, 1991. Jobs 1 and 3 in SET1 explicitly include AUGUST 13 in their selection criteria. Jobs A and Z, in SET2 and SET4 respectively, include the category TUESDAYS, which has been defined to match AUGUST 13 in the NetBatch-Plus calendar. Job 4 also includes TUESDAYS but explicitly excludes AUGUST 13 and therefore cannot be selected.

Of the remaining jobs, Job 3 belongs to a JOBCLASS other than GALACTIC, which was specified for the BULK SUBMIT run. Similarly, Job A is assigned to a SCHEDULER with a three-character name of $WAR. In this run, only Jobs 1 and Z have the JOBCLASS, date, and SCHEDULER names that conform to the selection criteria. Therefore, only those two jobs are eligible for selection.

***Bulk Run by Category***. In the second method of bulk-run selection, the user can explicitly select all jobs that belong in a specific category, regardless of whether that category is defined in the calendar. Jobs can be selected by category when they must run at times controlled by the user rather than by predefined dates.

Suppose, for example, a user wants to manually initiate a complete batch run for the End of Financial Year Tax Reporting job after the accounts have been closed. All jobs to be included in the Tax Reporting run have the category TAX-REPORT among their selection criteria. This category has no calendar entry because it has no dates associated with it. When the user selects the category TAX-REPORT for a bulk run, all the Tax Reporting jobs will run at the specified start time.

The user can also select bulk runs for categories that have user-defined dates in the calendar. For example, one can select jobs in the category TUESDAYS. The jobs run at the specified start times on the day the selection is made. The dates specified in the calendar are ignored.

***Bulk Run by Date***. In the third method of bulk-run selection, the user can explicitly select all jobs due on a specific run date, regardless of category. Bulk jobs scheduled for a particular run date can be selected and submitted on a different date. This feature is particularly useful when one has to rerun jobs or run them in advance.

***Test Runs***. The user can perform a test run before actually submitting a group of jobs selected explicitly by date or category. The test run generates a report listing all jobs that would be submitted for that bulk run together with their start times. The test run allows users to modify the actual bulk run before submitting it, giving them better control over the batch environment.

## System Visibility

The NetBatch-Plus Scheduler Interface gives the user a full-screen interface to batch SCHEDULERS. It is a superset of the functions available in the BATCHCOM[2] component of NetBatch (Wakashige, 1989). The Scheduler Interface provides easier and faster control of the batch environment and offers greater power and flexibility.

### Monitoring the Batch Environment

From a single terminal screen, the user can obtain static and dynamic information on each SCHEDULER in the batch processing environment. With the proper security access and SUPER user ID, a system manager or operator can start, shut down, and update any SCHEDULER as well as add and configure new SCHEDULERs without having to use BATCHCOM. JOBCLASSes and EXECUTORs can also be configured and updated. There is no need to remember difficult keywords with the user-friendly screens in NetBatch-Plus.

Any user can inquire on the status of a job or can update its attributes. Jobs can be stopped; if required, they can be suspended and reactivated. Also, the user can monitor all the processes in any currently executing job.

### Invoking Utilities

The NetBatch-Plus Utility menu provides a gateway to Tandem subsystems, allowing the user to view and manage the entire system environment without having to exit NetBatch-Plus. From the Utility menu, the user can invoke Tandem utilities and user applications.

[2]BATCHCOM is a conversational interface to the SCHEDULER that allows a user to submit, monitor, and control jobs throughout a distributed environment.

For example, the user can add a job and submit it to run immediately on the Job Definition screen, monitor its progress on the Job Status screen, and then invoke the PERUSE utility of the SPOOLER program to check its output. If the job does not execute satisfactorily, the user can edit the IN file of that job by invoking the PS TEXT™ EDIT (TEDIT) utility, then resubmit the job from the Ad Hoc Job Selection screen. All this can be accomplished from within NetBatch-Plus.

From the Utility menu, the user can invoke up to 13 utilities, including BATCHCOM. To suit the requirements of an individual site, the system manager can easily replace up to 12 utilities in the menu with user-written applications or Tandem utilities. BATCHCOM is the only utility that is mandatory and cannot be replaced. Appendix C in the *NetBatch-Plus User's Guide* documents how to reconfigure the Utility menu.

## Management Reporting

The NetBatch-Plus reporting system provides a variety of critical information from the NetBatch-Plus database, enhancing the visibility of the batch environment. The reporting system is flexible and easy to use. The system manager can fully configure the Report menu, replacing standard reports with those written by users. Users are encouraged to modify the standard reports or write their own reports to suit the requirements of their installation. For details, refer to Appendix C of the *NetBatch-Plus User's Guide*.

NetBatch-Plus comes with 10 standard management reports that use the ENFORM query language/report formatter; sources are supplied as examples. In addition to the eight reports that appear on the Report menu, users can select their own ENFORM report simply by entering its name. Users can easily limit any report to particular dates, categories, defaults sets, and job names, by specifying start and end values for parameters on the menu. ENFORM selects only those records that fall within the specified range or ranges.

### BULK SUBMIT Reports

Each bulk run generates a standard report containing details of all the jobs selected in the run. The report lists each job by defaults set, job name, node, SCHEDULER, JOBCLASS, and start time. It also reports any errors or warnings issued during the job submission.

A daily BULK SUBMIT run produces a report that lists the jobs submitted that day and predicts the jobs due on the next day. The user can examine this report daily and adjust the list for the next day to avoid any errors or omissions. The BULK SUBMIT program also creates a database file containing comprehensive information about the jobs selected and any errors or warnings issued during the run. Users can also write their own customized reports for the bulk run by using the record REPORT-BULK in the Tandem Data Definition Language (DDL) dictionary, supplied with NetBatch-Plus.

## Individually Defined Security

NetBatch-Plus permits both fine-tuned control over the security of the batch processing environment and flexible access to that environment. First, access to the NetBatch-Plus database is defined on a record-by-record basis, giving users separate control over each defaults set, catalog record, job, and job attachment. Second, the system manager can control the access to the NetBatch-Plus software and the batch environment by defining each user's profile, which determines the user's access to each NetBatch-Plus screen and to SCHEDULER and utility functions.

### Record Security

Each record in the NetBatch-Plus database is assigned a particular owner by use of GUARDIAN 90 user IDs. The record security determines which GUARDIAN 90 users can read, write, use, or purge that defaults set, catalog, job, or job attachment. Access to each record can be restricted to the owner of the record, to members of the owner's group, or to all users. These restrictions apply to all NetBatch-Plus users, including those on remote systems.

Read, write, and purge security apply to all record types in NetBatch-Plus. However, only defaults sets and catalog records can have use access meaningfully defined. The owner of the defaults set can restrict the users permitted to use it. By specifying a particular access type for use on a defaults set, the owner can limit the users who can attach job definitions to it. Similarly, the use access for a catalog determines which users may attach that catalog ASSIGN, PARAM, or DEFINE to their own jobs.

For each job definition, access may vary as to who can read, update, or delete it from the database. However, the owner of the job is usually the only one permitted to submit it to a NetBatch SCHEDULER. Alternatively, by setting the ANY USER SUBMIT flag on the job record, the job owner allows any user of NetBatch-Plus to submit the job on an ad hoc basis. In contrast, anyone who has access to the Bulk Submit screen may select eligible jobs for a bulk run, regardless of who owns them.

### Password Validation

A Password Validation screen allows the user to validate one or more GUARDIAN 90 user IDs in order to access different sets of NetBatch-Plus records. All validated passwords remain current until the end of the NetBatch-Plus session. To prevent unwanted access to the batch scheduling system, it is strongly recommended that users exit NetBatch-Plus when they finish their scheduling tasks. For additional security, a user-supplied time-out parameter automatically exits each NetBatch-Plus screen when no keyboard input occurs for the time-out period. The user can adjust the time-out period to any length above five minutes or can effectively disable the time-out feature.

Figure 7

| NetBatch-Plus User | Supervisor | Maintain jobs and parameters | Submit jobs | | Job environment limits | Change own password | Access to utilities | Access to SCHEDULER interface | |
|---|---|---|---|---|---|---|---|---|---|
| | | | Ad hoc | Bulk | | | | Job screens | Other |
| Manager | X | X | X | X | All | X | X | X | X |
| David | | X | X | X | All | X | X | X | X |
| Rhonda | | | X | | \VENUS.$LOVE | X | X | X | |
| Anna | | | X | | \MARS.$*, \EARTH.$* | X | | X | |
| Bruce | | X | X | | \MARS.$WAR | X | X | X | |
| Operator | | | X | X | All | | X | X | X |

**Figure 7.**
*Use a matrix of NetBatch-Plus users similar to the one shown above to determine security needs for the batch environment.*

## Screen Security

User profiles give the system manager a way to control each user's access to NetBatch-Plus. The user profile defines three distinct areas of access:

- Screen security defines the level of access the user has to each NetBatch-Plus screen. Access can be defined as update allowed, inquiry only, or no access.
- Job environment limits define the SCHEDULERs and JOBCLASSes available to jobs maintained or submitted by the user.
- Utility security defines the level of access the user has to the utilities, such as BATCHCOM, available in the Utility menu and to the Scheduler Interface.

## Bulk Submit Environment

Although the jobs selected in a bulk run can be submitted to many SCHEDULERs, the BULK SUBMIT control job always runs in the same SCHEDULER and JOBCLASS. It is recommended that only the system manager or database administrator have update access to the NetBatch-Plus screen that maintains the control job's environment and GUARDIAN 90 owner.

Security for the Bulk Submit environment is further controlled by automatic purging of temporary files created by NetBatch-Plus. A window parameter allows users to define the number of days temporary files remain in the database. It is recommended that the BULK SUBMIT program purge temporary files after the third day.

## Defining Security for NetBatch-Plus Users

To determine the security for NetBatch-Plus users, the system manager can create a matrix like the one shown in Figure 7. First, list all possible users; these users do not need to map directly to GUARDIAN 90 users. Next, list the major functions each user is permitted to perform.

The sample matrix in Figure 7 includes a manager; an operator, which can be represented by several people; and four other users listed by first name. The matrix does not list all the functions available in NetBatch-Plus, but it provides a basis on which to decide the users' access to individual screens. The matrix shows the individual requirements of each user, which can be implemented in NetBatch-Plus. The system manager still retains control of the total batch environment.

The user Manager is made a Supervisor, which automatically grants him or her full access to all screens and functions in NetBatch-Plus. The Operator can only submit jobs but has access to all job environments. Therefore, the Operator would be granted full access to the Ad Hoc Job Selection and Bulk Submit screens as well as inquiry access on the Job Definition screen. Also, the Operator would probably be granted full access to the Scheduler Interface and Utility screens, but access to most screens that maintain the NetBatch-Plus database would be denied.

The other four users in Figure 7 can submit ad hoc jobs, inquire on the jobs' status, and change their own passwords. However, most are limited to submitting jobs to certain SCHEDULERs only and may not submit bulk runs.

## Setting Up a NetBatch-Plus Database: A Checklist

In order to structure a batch environment to suit a particular installation, the system manager needs to do some preliminary planning. The following checklist summarizes the important steps required to design and implement a NetBatch-Plus database.

- First, list all batch tasks and give them unique names. These are the job definitions.
- Organize the jobs into logical groups or defaults sets. For an example, see Figure 3.
- For each job, list all its job attachments; that is, all ASSIGNs, PARAMs, and DEFINEs. For an example, see Figure 4.
- From a matrix of jobs versus job attachments, identify common or shared attachments that can be grouped into catalog records. Organize the catalogs into appropriate defaults sets.
- Next, specify the dependencies among the NetBatch-Plus jobs.
- If jobs run on a regular schedule, decide on their scheduling criteria: dates on which they run or don't run, or categories to which they should belong.
- List regularly recurring dates and organize them into categories. Enter the categories in the calendar.
- List all possible users, noting that they do not need to map directly to GUARDIAN 90 users. Next, list the major functions each user is permitted to perform. For an example, see Figure 7.

## Conclusion

By adding structure to the batch processing environment, NetBatch-Plus enhances the batch scheduling facilities provided by NetBatch and gives users the ability to handle complex bulk scheduling. With NetBatch-Plus, users can categorize job information, specify defaults sets for similar kinds of jobs, and utilize catalogs of batch scheduling parameters. NetBatch-Plus simplifies batch scheduling with its full-screen interface, versatile database, system visibility, and the easy access it provides to other Tandem subsystems and utilities. NetBatch-Plus implements the Tandem commitment to mainframe batch processing.

**References**

Keefauver, T. 1989. Optimizing Batch Performance. *Tandem Systems Review*. Vol. 5, No. 2. Part no. 28152.

*NetBatch User's Guide*. Tandem Computers Incorporated. Part no. 18097.

*NetBatch-Plus User's Guide*. Tandem Computers Incorporated. Part no. 19527.

Wakashige, D. 1989. NetBatch: Managing Batch Processing on Tandem Systems. *Tandem Systems Review*. Vol. 5, No. 1. Part no. 18662.

**Glenys Earle** is a quality assurance engineer in the Batch Research and Development Group in Melbourne, Australia. Before joining Tandem in 1988, she worked as a quality assurance engineer for a Tandem third-party vendor, as a support analyst, and as a technical programmer.

**Dean K. Kaumaakani Wakashige** is a senior staff analyst for the Customer Support Organization. Before joining Tandem in 1987, Dean was the support manager for a Tandem third-party vendor and a systems analyst for a major bank.

# Converting Database Files from ENSCRIBE to NonStop SQL

Because of the increasing interest in Structured Query Language (SQL) relational database management, a growing number of users of Tandem™ products are considering converting from the ENSCRIBE database record manager, the original Tandem data access system, to the NonStop™ SQL distributed relational database management system. There are many reasons to convert from an ENSCRIBE to a NonStop SQL system, but conversion can be a complex endeavor. Users should evaluate several critical factors before they decide to convert. In some installations, it is more practical to keep the ENSCRIBE system. In other installations, it may be more efficient to support some applications with ENSCRIBE files and other applications with NonStop SQL database tables.

It should be emphasized that the NonStop SQL file system is designed to coexist with the ENSCRIBE file system. For example, the ENSCRIBE and NonStop SQL systems share modules for the OPEN function. The ENSCRIBE file system supports unstructured files, object files, and edit files, whereas NonStop SQL only supports database files, which are called SQL tables. Therefore, the ENSCRIBE system will continue to be part of any Tandem environment, including one that uses only NonStop SQL databases.

This article initially explores the issues that must be weighed when one is deciding whether to convert and discusses several practical considerations that help to make a conversion successful. The article describes the benefits of NonStop SQL as compared to ENSCRIBE, and it discusses whether conversion is warranted in a few hypothetical scenarios. It emphasizes the importance of redesigning and rewriting the application during conversion. The article then describes the considerations for a successful conversion, including conversion methods, conversion tools, program-code conversion, and step-by-step programming changes. Tables at the end of the article compare the features of the NonStop SQL Report Writer to those of the ENFORM™ query language/report formatter, the report-writing system associated with ENSCRIBE.

The information presented in this article is based on NonStop SQL Release 2, and all references to and examples of programming code appearing in this article are based on COBOL85. Readers using other programming languages need to extrapolate the information for their particular environment.

## Determining Whether to Convert

Before deciding to convert to a new system, users should compare the benefits of NonStop SQL to those of the ENSCRIBE system, examine their reasons for converting, and consider the effort required to redesign the application as well as to convert the database files. Users should also consider the costs associated with conversion, which include staff time, staff training, CPU resources, and other resources such as disk space.

### Comparing the NonStop SQL and ENSCRIBE Systems

NonStop SQL is a relational database management system that uses the ANSI (American National Standards Institute) standard version of SQL to define and manipulate data. It is integrated with other Tandem system software so that it can be used for online transaction processing (OLTP) applications, batch applications, or both. NonStop SQL can operate centrally or be distributed in a network of systems.

The flexibility of the SQL language permits NonStop SQL code to be embedded in COBOL85, C, and Pascal programs. It can also be used for ad hoc queries and update functions. The SQL language was specifically designed for easy access to relational databases.

In contrast, the ENSCRIBE record management system stores and retrieves data on a record-by-record basis, either by key or sequentially. It is not a database management system.

***Performance.*** The NonStop SQL system delivers high performance for production OLTP applications. A benchmark demonstrating several aspects of NonStop SQL Release 1 produced a transaction rate of over 200 transactions per second (Tandem Performance Group, 1988). That benchmark was based on the debit-credit banking application also known as ET1 (*Datamation*, April 1985).

Although the ENSCRIBE system performs reasonably well compared to NonStop SQL, it is much more limited in scale. ENSCRIBE can have at most 16 partitions, so it cannot be expanded beyond 16 processors. NonStop SQL allows up to 100 partitions, supported by up to 100 processors. NonStop SQL, in general, offers better performance than ENSCRIBE. However, as of version C10 of the GUARDIAN™ 90 operating system, a few features, such as file loading, perform better in ENSCRIBE than in NonStop SQL.

NonStop SQL Release 2 is significantly enhanced to deliver high performance. For example, table joins, set operations, and batch read-ahead buffering are improved in Release 2.

***Application Development and Maintenance.*** Application development time and the expense of application maintenance are two key considerations in comparing the costs of developing new ENSCRIBE and NonStop SQL applications. Discounting the startup time and learning curve for those who are unfamiliar with NonStop SQL, NonStop SQL appears to be far superior in both areas. Because most systems cost more to maintain than to develop initially, a tool that aids in maintenance is a significant benefit.

NonStop SQL is easy to learn and use, yet it is a fast and complete language that provides data definition and data manipulation. Application code is simplified because NonStop SQL has only four Data Manipulation Language (DML) statements to perform the retrieve, insert, update, and delete functions. In some cases, a single NonStop SQL statement can replace paragraphs of application code. Using the Conversational Interface (SQLCI), users can test NonStop SQL statements at their terminals before embedding them into a program.

Programmers do not need to explicitly specify the path for accessing data. To optimize database queries, NonStop SQL manages the access path to the data by using statistics for each database.

Programmers can use logical names for NonStop SQL objects in the program code. The logical names are mapped to the true names at compile and execution time. This allows the same program code to be used in multiple environments. For example, the same code can appear in a test system and a production system.

***Changing a Database***. Changing a database is easier in a NonStop SQL than in an ENSCRIBE system. For example, table columns can be added to existing rows without having to change existing code or to unload and reload the database. Indexes can be added or deleted without having to change either the existing code or the database. Automatic data declaration retrieves the latest data declarations from the dictionary into host-language declaration code. At run time, NonStop SQL checks the program for consistency with the database. If the database has changed, the program is automatically recompiled before it is executed. Also, changes can be made using NonStop SQL statements from SQLCI. Some changes, such as new data declarations in the dictionary, are totally transparent; others may require program recompilation.

### Conversion Scenarios

These five scenarios explore possible conversions from an ENSCRIBE system to NonStop SQL. Each scenario presents the reasoning for conversion and examines the effort and cost associated with the scenario, as well as benefits or drawbacks.

***Case 1***. Users may want to convert to learn about NonStop SQL. Converting for this reason is not advisable. Database files should be converted only when the applications that use them benefit by that conversion. Users can gain the same experience and reap greater benefits by developing a new application that uses NonStop SQL.

***Case 2***. Users may want to convert to measure the relative performance of the NonStop SQL and ENSCRIBE systems. One should be careful when converting for this reason. Depending on the environment, application system, and database, certain functions of NonStop SQL may perform poorly when compared to ENSCRIBE; other functions will perform very well.

***Case 3***. Users may decide to convert when two separate applications share common data, and one of these applications must be rewritten. The application to be rewritten has new requirements, but normal operations for the other data-sharing application must be preserved. The user must either convert both applications or maintain two copies of the database, one in ENSCRIBE and one in NonStop SQL. The second alternative is not an attractive prospect.

***Case 4***. Installations that have third-party software systems are good candidates for conversion if one wants to market these systems to SQL users. The magnitude of effort must be determined on a case-by-case basis.

***Case 5***. Users may want to convert for the sake of staying on the leading edge of technology. This is not recommended. Conversion is costly, and a straight conversion will probably not improve the original system. If an ENSCRIBE application is no longer adequate, one should develop a new application rather than perform a straight conversion.

### Redesigning and Rewriting the Application

When rewriting an application, users should consider converting to NonStop SQL. NonStop SQL can add functionality, accommodate user-requested changes, and make it easier to combine applications. Also, the costs of conversion are partly absorbed when resources have already been committed to redesigning the application.

When users plan to convert a database to NonStop SQL, it is strongly recommended that the application be redesigned and rewritten. An application that is converted without being rewritten fails to take advantage of the power of NonStop SQL and may not perform as well as it did in an ENSCRIBE system. For example, poorly written SELECTs can result in full file scans and temporary tables with sorting. An application that uses the ENSCRIBE START statement and reads a predetermined number of records is likely to perform poorly.

Users who are still deciding whether or not to convert should take into account the added effort and the benefits of rewriting the application. Even in a simple conversion that replaces ENSCRIBE I/O statements with SQL statements, some areas of the application should be redesigned. Applications are affected by the different ways ENSCRIBE and NonStop SQL handle data requirements; replicated fields; normalized relational databases; updates, either with or without prior reads; multiple reads; table joins; and error handling.

***Change of Data Requirements.*** Data requirements may have changed since the database was designed. During conversion, the database should be redesigned to meet the new requirements. Redesigning also allows small enhancements to be made with little effort.

***Removal of Replicated Fields.*** Because ENSCRIBE does not support noncontiguous or descending keys, some files contain replicated fields used to construct alternate keys. NonStop SQL does support noncontiguous and descending keys. Therefore, replicated fields should be removed from files being converted to NonStop SQL.

***Normalized Relational Databases.*** NonStop SQL supports normalized relational databases. The ENSCRIBE clause variations, OCCURS and OCCURS DEPENDING ON, are not normalized relations and are not supported in NonStop SQL. A file containing these attributes must be changed. The SQL table definition must contain a column name for each individual column possible. In this sense, NonStop SQL tables are fixed-length tables. They can be variable-length tables only by specifying variable-length columns.

Assume, for example, that the ENSCRIBE Data Definition Language (DDL) specifies a group item named GROUP that contains FIELDA, FIELDB, and an OCCURS 3 TIMES or OCCURS 1 TO 3 TIMES DEPENDING ON *n* clause. When the item is described in a NonStop SQL table definition, provision must be made for three occurrences each of FIELDA and FIELDB. Each occurrence must have a unique name. Space will be allocated for all six columns. The suggested alternative is to normalize the data.

**Figure 1**

```
EXEC SQL UPDATE TABLEx                  Specifies which table.
     SET COL1 = :COL1,                  Changes the values of
         COL2 = :COL2                       two columns.
     WHERE KEY  = :KEY                  Specifies the key of the row.
     AND COL1   = :OLDCOL1              Only if the current value is
     AND COL2   = :OLD COL2                 equal to what was originally
END-EXEC.                                   read.
```

***UPDATE without Prior Reads.*** NonStop SQL supports an UPDATE statement without prior reads. This saves one logical I/O per update. Because ENSCRIBE requires an extra read before any update, the application should be changed to take advantage of this performance gain.

For example, it can be necessary to prevent two users from updating the same record at the same time without locking the record across screen displays. The server code reads the record and sends the information to the requester for screen display. The user enters changes, and the requester sends the complete original record along with the changed record to the server for updating. The server then reads the record, compares it to the original, and updates it only if the original exactly matches the current record. If two people had viewed the record and both made changes, only the first update to arrive would be applied. The second user would have to read the record again.

The read and compare after the initial read and before updating is not necessary with NonStop SQL. The compare can be made during the update by specifying the original data in the WHERE clause using a statement similar to the one shown in Figure 1.

**Figure 1.**
*UPDATE statement without prior read.*

In ENSCRIBE, a read returns the whole record, regardless of whether all the fields are necessary. In NonStop SQL, the user can specify only those columns that are of interest; because less data is transferred, performance is improved. When multiple records are read, NonStop SQL uses virtual sequential block buffering to fill a block with the needed data, and the block is transferred.

NonStop SQL sends fewer messages to the disk process requesting data than does ENSCRIBE. In the UPDATE example in Figure 1, there is no reason to compare the whole record and possibly risk rejection of an update when this program is not sensitive to a column that was changed by a second program. One can also change the interprocess messages to contain only the data columns to which the program is sensitive. As of version C30 of the GUARDIAN 90 operating system, if the index of a key-sequenced file contains the column information necessary to evaluate a NonStop SQL instruction, only the index will be accessed. No access will occur to the base table.

***ENSCRIBE Multiple Read versus SQL Join.*** Typically, ENSCRIBE programs read more than one file. Often data carried in one file functions as the foreign key of another file. This requires the code to read one file, obtain the foreign key, and then read the second file. The program logic that reads the second file may include a loop that locates a specified item or range of items.

These operations can be performed in one NonStop SQL statement using JOIN. To change from reading multiple files to using a single JOIN statement may require major modifications in the program logic. For example, when an ENSCRIBE program has a main program and all I/O is performed in subprograms, each affected subprogram must be changed.

***Error Handling.*** ENSCRIBE programs usually contain simple error-handling routines. The number of return codes in ENSCRIBE is limited, and the read-one-file-at-a-time approach makes it easy to correlate an error with the related file.

NonStop SQL provides more comprehensive error reporting. First, NonStop SQL includes hundreds of possible return codes. Second, it can return both errors and warnings following each DML statement. Third, up to seven return-code conditions are possible for each statement.

With features such as joins and subqueries, it is difficult to correlate any one return code with any one table. For example, when a SELECT statement is issued for a joined table, a not-found condition that follows does not identify which table is missing the data. In a case like this, the table name is stored in a location difficult to find in the PARAMS-BUFFER area of the SQLCA (SQL communications area) block.

Programmers should develop error routines that are more sophisticated in NonStop SQL than those in ENSCRIBE. Whole sections of exception-handling code must be redesigned and rewritten. Because the ENSCRIBE error messages are likely to be replaced and expanded upon, end users may also have to be reeducated.

## Considerations for Successful Conversion

Once the decision to convert has been made, it is useful to compare the two environments in certain key areas to make the conversion as smooth as possible. Specifically, users should compare staffing requirements, dictionaries, programming features, and data definition procedures in the ENSCRIBE and NonStop SQL systems.

### Staffing

A shortage of expertise in SQL may pose a barrier to conversion from ENSCRIBE to NonStop SQL. However, SQL programmers are more plentiful than ENSCRIBE programmers; this is a particular advantage for organizations that employ contract programmers.

After conversion, more resources in database administration may be required. Many functions performed by programmers in an ENSCRIBE environment are shifted to the database administrator (DBA) in a NonStop SQL environment. Performance analysis, tuning, and modifications to databases, such as creating or dropping indexes, are centralized.

In an ENSCRIBE environment, users typically think in terms of files and programs that support the files. In many cases, applications are small and isolated from one another, and programmers are responsible for the applications.

In a NonStop SQL environment, users will begin to think in terms of large volumes of data logically related in some fashion and used by multiple groups; that is, databases. This creates a need for a single point of control; thus, database administration becomes more important and requires more resources.

Shifting the responsibility for the database from the programmers to a single database-administration staff increases productivity for both groups. The DBA staff can become more proficient at doing backup, restore, tuning, and performance analysis, which frees the programmers from performing these tasks and allows them to concentrate on developing application code.

**Two Dictionaries**

The NonStop SQL dictionary is the set of all the catalogs on a network together with the disk file labels for all the objects described in these catalogs. All NonStop SQL object definitions are stored in catalog tables. For example, the definitions of the SQL tables are stored in a catalog table and may be brought into the program by using the INVOKE statement.

The definitions of ENSCRIBE files are described in the DDL dictionary and in copy libraries. Also, definitions of the interprocess messages are typically stored in copy libraries.

The user must maintain the ENSCRIBE and NonStop SQL dictionaries as well as the copy libraries. Database changes are not reflected in the interprocess messages stored in the copy libraries. No tools or utilities are available to ensure consistency between the two dictionaries; however, the ENSCRIBE system does not provide tools to maintain consistency within the one dictionary.

**Programming Issues**

NonStop SQL programs must be written in COBOL85, C, or Pascal. NonStop SQL does not currently support TAL™ (Transaction Application Language). Also, ENFORM reports must be entirely rewritten by using the Report Writer feature of NonStop SQL.

NonStop SQL supports only normalized data. ENSCRIBE supports data definition features such as levels, groups, and redefines.

NonStop SQL is very powerful. Indiscriminate use of ORDER BY or GROUP BY on table columns without an index may result in a full table scan followed by a sort.

NonStop SQL automatically maintains indexes; users cannot directly access indexes. ENSCRIBE programs that perform manual maintenance of alternate-key files cannot be converted and may not be needed with NonStop SQL.

Error return codes and warning return codes in NonStop SQL are different and more extensive than those in ENSCRIBE. A single NonStop SQL statement can return up to seven error or warning return codes.

NonStop SQL locking is considerably different from ENSCRIBE security procedures. ENSCRIBE protects a program from reading uncommitted updates even when locks are not used in the reading program. NonStop SQL offers the same protection only through locks. NonStop SQL locks can be obtained on audited tables only within a TMF™ (Transaction Monitoring Facility) transaction. Therefore, programs being converted must be changed to include BEGIN and END TRANSACTION statements. These statements are usually issued within requesters, so the requesters must be changed.

More locks can be acquired in NonStop SQL than in ENSCRIBE, which can cause programs to be enqueued or to receive Error 73 (timeout while waiting for a lock) in unanticipated situations. In ENSCRIBE, locks are not required with nonaudited files to insert, update, or delete records. Locks are required with nonaudited tables in NonStop SQL, though TMF transactions are not.

**Table 1.**
ENSCRIBE operations not supported in NonStop SQL.

| | | | |
|---|---|---|---|
| **ALTER operations not supported in NonStop SQL** | | | |
| 1 | alter file code | 2* | alter audited attribute |
| 3 | alter refresh attribute | 4 | alter odd-unstructured attribute |
| 5 | alter alternate key | 6 | alter partition |
| 7* | alter broken flag | 8* | alter no-purge-until |
| **CONTROL operations not supported in NonStop SQL** | | | |
| 2 | write EOF | 20 | purge data |
| 21 | allocate/deallocate extents | | |
| **SETMODE [NOWAIT] operations not supported in NonStop SQL** | | | |
| 1 | set security string, progid, and clearonpurge | | |
| 2 | set owner | 3 | set verify writes |
| 4 | set alternate lock mode | 57 | set serial writes |
| 90 | set buffered option | 91 | set accesstype open |
| 92 | set maximum extents | 93 | set unstructured buffer length |
| 94 | set audit-compress | 95 | flush dirty cache buffers |

*With Release C30, NonStop SQL will support these ALTER operations.

### Data Definition Procedures

The ENSCRIBE GUARDIAN 90 system call procedures include these seven operations:

- CREATE defines a file.
- PURGE deletes a file.
- RENAME renames a file.
- ALTER alters file attributes.
- CONTROL performs device-dependent I/O operations.
- CONTROLBUF performs device-dependent I/O operations requiring a buffer.
- SETMODE [NOWAIT] sets device-dependent functions.

These procedures cannot be used to create or alter NonStop SQL objects. Table 1 lists the ALTER, CONTROL, and SETMODE [NOWAIT] operations not supported in NonStop SQL. Also, NonStop SQL does not currently support the RENAME operation; however, it is supported in Release 2.

NonStop SQL provides commands that perform the equivalent functions, where appropriate, for SQL objects. For example, the NonStop SQL ALTER command sets the security string, owner, audit attribute, and file attributes. The PURGE and PURGEDATA utilities or the DROP command delete data or tables. In a NonStop SQL environment, the database administrator usually becomes responsible for these functions; this shift in responsibility changes operational procedures for the DBA staff, programmers, and operators.

## Conversion Methods

A NonStop SQL program can reference both ENSCRIBE files and SQL tables. This allows users to take one of three basic approaches to converting from an ENSCRIBE to a NonStop SQL system. Users can convert individual files, groups of files, or all the files associated with the application.

### One File at a Time

The least desirable method converts one file at a time. An application using the ENFORM product to generate reports from multiple files cannot convert one file at a time. Neither ENFORM nor the NonStop SQL Report Writer supports access to both ENSCRIBE and NonStop SQL.

Users could convert a file and change all programs using that file, but this should be done carefully. If files are shared across applications, programs would have to be modified in multiple applications. The chance of corrupting any one application increases with each additional application that must be changed.

If one takes the file-at-a-time approach, programs must reference both ENSCRIBE and NonStop SQL until all the files are converted. This is acceptable, but it could result in a failure to use NonStop SQL properly.

For example, assume that a program cannot join two or more tables because one is an ENSCRIBE file and the other is a NonStop SQL table. To provide a temporary solution, code is written to read each file separately. When the second file is converted, the user must change whole sections of code to perform the JOIN. The user is more likely to simply convert the access to the second table and lose any chance to improve performance provided by the JOIN.

Programs that use ENSCRIBE files and NonStop SQL tables will contain two different error routines for handling file-error conditions. The method for presenting errors to end users may have to be changed. For example, many ENSCRIBE programs return the FILE STATUS or GUARDIAN error; NonStop SQL has no exact equivalents for these error codes.

### Groups of Files

If the entire application cannot be converted at one time, one can convert groups of related files. This method allows programmers to take advantage of NonStop SQL functions such as joins and subqueries. A special transaction matrix determines which files constitute a group. (See Figure 2.) The matrix lists transactions, the files they use, and the type of reference made to the file.

From the transaction matrix shown in Figure 2, it can be determined that files F1 and F2 constitute a group because at least one transaction references both files. Files F3 and F4 constitute a group for the same reason. The matrix also indicates the amount of effort involved in converting a group. If F1 and F2 are selected as a group, the code that manages three transactions must be converted. If F3 and F4 are selected, two transactions must be converted.

The matrix also can be applied to such tasks as performing tuning functions and estimating system load. If the number of logical I/Os is included with the access type, the matrix can help to determine file placement and predict the load when transaction volumes change.

### Entire Application

If the application is small enough and has a limited number of files, all programs and files of the entire application can be converted at one time. This method is the best one because it allows users to complete the entire redesign at once. It also requires only one testing period, allows programmers to take full advantage of SQL functions such as table joins and subqueries, and disrupts the application for the shortest length of time. A good candidate for this approach is an application that can be converted in three months or less, at a reasonable cost.

**Figure 2**

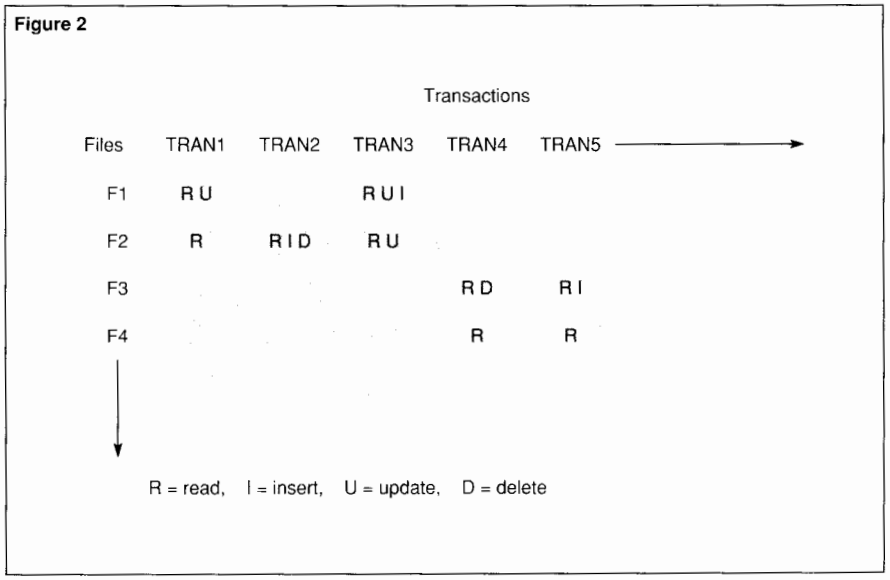



**Figure 2.**
*File groups by transaction.*

## Conversion Tools

Tandem supplies the CONVERT and the LOAD utilities to assist in the conversion process. The CONVERT utility generates the statements required to define and create NonStop SQL tables, and the LOAD utility loads the tables with the data contained in ENSCRIBE files. These utilities are easy to use and have contributed to a successful conversion from ENSCRIBE files to NonStop SQL tables.

### CONVERT Utility

The CONVERT utility converts an ENSCRIBE file described in a DDL dictionary to an SQL table described in a catalog. It produces:

- A table definition containing rows that correspond to the records of the ENSCRIBE file and columns that correspond to the fields in each record.
- Indexes on the table that correspond to the alternate-key files associated with the ENSCRIBE file.

**Figure 3**

**Conditions tested**

```
EXEC SQL
  WHENEVER NOT FOUND      PERFORM : 9999-NOT-FOUND
END-EXEC

EXEC SQL
  WHENEVER SQLERROR       PERFORM : 9999-SQL-ERROR
END-EXEC

EXEC SQL
  WHENEVER SQLWARNING     PERFORM : 9999-SQL-WARN
END EXEC
```

**Program code generated**

```
IF (SQLCODE OF SQLCA=100)
  PERFORM 9999-NOT-FOUND.
IF (SQLCODE OF SQLCA < 0)
  PERFORM 9999-SQL-ERROR.
IF (SQLCODE OF SQLCA > 0) AND
  (SQLCODE OF SQLCA NOT EQUAL 100)
  PERFORM 9999-SQL-WARN.
```

**Figure 3.**
*NonStop SQL program directives.*

CONVERT generates the commands needed for CREATE TABLE, CREATE INDEX, and the LOAD utility and stores them in an edit file. Users can edit this file before executing it.

The DDL record definition is reduced to two levels by eliminating DDL group names, REDEFINE statements, and OCCURS clauses. These repeating fields are defined as individual columns and are assigned system-generated names. These names may not give meaning to the data content or they may be confusing. Most users edit these field names to be more descriptive.

If the data type of a DDL field is a NonStop SQL data type, the column is assigned the same data type. If the data type is not valid in NonStop SQL, the column is assigned a compatible data type. When performing edit functions, be cautious about changing the data type. Column data is typically moved around within a program, and, because the destination fields have been defined previously (for example, as working storage or interprocess messages), the data must be converted when it is moved if the data types do not match.

### LOAD Utility

The LOAD utility loads data from an ENSCRIBE file into a NonStop SQL table. This operation automatically loads all indexes defined for the table. Users are allowed to effect changes, such as reordering or dropping fields, to a NonStop SQL table through a mapping function in the LOAD utility.

## Program Coding Conversion

When ENSCRIBE files are converted to NonStop SQL tables, every program referencing those files must be modified. It is beneficial to standardize certain tasks that appear in many programs. Some tasks can be coded only once and copied into each program. This helps minimize program development and testing time, and it assures adherence to any established organization standards.

### New Error Routines

Because NonStop SQL errors have different values, are more extensive, and are reported differently than ENSCRIBE errors, the program routines that handle these errors must be changed. NonStop SQL provides WHENEVER directives that generate the host-language code needed to test the result of each executable SQL statement and the action to take when a particular condition occurs. The conditions that are tested for include NonStop SQL errors, NonStop SQL warnings, and a not-found condition. The action to be taken can be CONTINUE, GO TO paragraph, or PERFORM paragraph. Figure 3 shows an example of each condition tested and the action to be taken.

### Standard Error Routine

It is recommended that a standard error routine be developed and copied into each program. This code would be performed as a result of the testing described in the previous paragraph.

The programmer can set a switch, defined in working storage, to reflect the result of a NonStop SQL statement. For example:

- 00 = successful operation.
- 01 = not-found.
- 02 = duplicate (INSERT operation).
- 03 = SQL error.
- 04 = SQL warning.

The switch can be tested following each NonStop SQL statement. A decision can be made to continue to the next statement or return with a message.

Create messages in a common area to be returned to the screen. For example:

- REQUESTED RECORD NOT FOUND.
- ADD FAILED — RECORD ALREADY EXISTS.
- DATABASE ERROR — CONTACT SUPPORT.
- DATABASE ERROR = *nnnn* — FILE SYSTEM CODE = *nn*.

These messages should be easy to understand. Most end users cannot determine or resolve a problem when only a return code and file code are presented. Most errors that cannot be identified with a clear, simple message require Support or Operations assistance.

### Routines Supplied by NonStop SQL

The NonStop SQL system supplies three TAL routines, named SQLCAFSCODE, SQLCADISPLAY, and SQLCATOBUFFER, that can help programmers create a standard error routine. Programmers can use any combination of these routines.

***SQLCAFSCODE***. This routine returns a file system code if one is associated with the condition. This code can be displayed in a message as shown in the previous list of sample messages.

***SQLCADISPLAY***. This routine parses the SQLCA control block and sends a message to an output device, the default being the home terminal (HOMETERM). The message:

- Indicates an error or warning and contains the return code.
- Identifies the program and line number of the SQL statement.
- Displays the message text as shown in the *NonStop SQL Messages Manual*.

When multiple errors or warnings occur for a statement, the user can display only one message or all the messages. SQLCI uses this error display routine.

***SQLCATOBUFFER***. This routine builds messages in the same format as the SQLCADISPLAY routine. The SQLCATOBUFFER then returns the messages to a program-defined buffer.

### Using the Supplied Routines

The standard error routine can send messages to HOMETERM by using SQLCADISPLAY. The routine should be selective when writing to HOMETERM, depending on whether the user wants messages for conditions such as not-found.

The standard error routine can use SQLCATOBUFFER to obtain the formatted messages and store them in a NonStop SQL table. The table can have a column for the message text as well as columns for the following data:

- Date.
- Time.
- User ID (if available).
- Error and warning codes.
- Any file system code.

The errors table allows the user to maintain a permanent record of errors and provides columns for later inquiries. By using SQLCI or user-written transactions, users can query the table to resolve problems. Queries can be made by the DBA, a support group, or a resource center in response to end-user requests for assistance.

An errors table can have interesting uses beyond resolving user problems. For example, it could generate a report that lists the number of errors returned in a day, tells how frequently a particular user group attempts to insert duplicates, and shows how often another user group's updates fail because constraints are violated.

A sample errors table and a common error routine are included with the NonStop SQL sample application.[1] Users can adapt them to meet their application requirements. The sample application also supplies a requester-server environment that allows selective displays of errors from the table.

## Locks

The NonStop SQL lock protocol provides more options than the protocol provided by the ENSCRIBE system. The additional versatility of the NonStop SQL lock protocol includes more table-locking options and the ability to read locked records.

[1]The NonStop SQL sample application is provided on the site installation tape and provides assistance for installation of the NonStop SQL product.

***File and Table Locks.*** ENSCRIBE allows a file lock; NonStop SQL allows a table lock. The ENSCRIBE file lock prevents other programs from accessing any records in the file. The NonStopSQL table lock provides two options: share mode or exclusive mode. Share mode allows other programs to read but not update data. Exclusive mode prevents other programs from reading or updating data.

***Record Locks.*** In the ENSCRIBE system, a program doing a READ operation without specifying a lock is protected from reading uncommitted updates (assuming there are no read-through locks). A READ with a lock prevents other programs from reading the same record. Any locked record remains locked until it is released by the program.

In the NonStop SQL system, the user can read records, whether they are locked or unlocked, by using the browse mode. Because the browse mode reads records that are locked or unlocked, it is subject to reading uncommitted updates.

Alternatively, the user can choose to read only those records that have no uncommitted updates by using the stable or repeatable mode; stable is the default. This choice places a shared lock on the row while the NonStop SQL statement is executing in the stable mode or until the end of the transaction in the repeatable mode. While the lock is in effect, no other program can update the row. Other programs can read the affected rows by using the shared lock protocol.

Finally, the user can choose to read only those records that have no locks; that is, records that do not have shared locks or uncommitted updates from other programs. The user specifies exclusive mode as well as the stable or repeatable mode. This choice is similar to the ENSCRIBE statement READ WITH LOCK. Once locked, the row is denied to any other program until it is released at the end of the transaction.

Note that for audited tables, locks are released at the end of the transaction. For nonaudited tables, locks are released by the program using a FREE RESOURCES or CLOSE CURSOR statement.

***Lock Considerations***. In some situations, NonStop SQL programs using row locks can lock an entire table. For example, the NonStop SQL optimizer[2] estimates the number of locks that will be obtained. If that number is very high, a table lock is obtained.

Locks on individual rows can exceed the internal lock limit of about 5000 locks per partition per audited transaction or per non-audited process. When this happens, the entire table is locked.

When a program specifies the repeatable mode, no other program can make inserts anywhere within the range of rows read by this program. To ensure that this is done, NonStop SQL locks not only the rows read, but also the row preceding and the row following the rows read. This situation can produce more locks than anticipated.

***Generic Locks***. In NonStop SQL, a generic lock is a lock held by a process on a subset of the rows within a key-sequenced table. When a table is created or altered by the ALTER command, the user can specify the LOCKLENGTH attribute, which specifies the prefix of the primary key used for locking. The prefix consists of one or more leading bytes of the primary key; the integer specified with LOCKLENGTH determines the length of the prefix. The default length is the entire primary key.

A generic lock is created when the prefix is smaller than the entire key and a lock is obtained. A generic lock works by locking all rows containing the identical key values in the specified number of bytes. To demonstrate generic locking, Figure 4 uses these assumptions: a lock length specification of 3, a table containing the keys shown, and a SELECT on key 123040 with REPEATABLE. Under these conditions, Figure 4 shows how all keys with a prefix of 123 are locked by a generic lock.

[2]The optimizer, a component of the NonStop SQL compiler, automatically selects the most efficient access plan for retrieving data from the database.

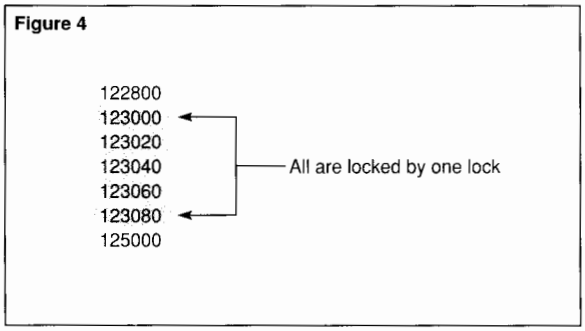



**Figure 4.**
*Rows locked by a single generic lock.*

In some environments, there could be thousands of individual row locks. When generic locks are used, the net effect is that fewer locks are obtained on the database. This can be useful in preventing table lock escalation as a result of obtaining too many locks. Sometimes the keys contain a prefix that denotes a group, such as location, category, or type, and transactions perform operations only within that group. A single generic lock can lock that entire group at once, avoiding the use of many individual row locks.

The danger in using generic locks is that they are defined in the table specification and apply to all transactions referencing the table. A transaction that references only a single row or a few rows could lock thousands of rows and affect the performance of other transactions by causing them to wait for the lock to be released.

### Existing Program Code

Typically, existing program code references I/O buffer areas for records, fields, and groups of fields using names assigned with ENSCRIBE RENAME, OCCURS, and LEVEL 88 functions. Because NonStop SQL tables do not support these functions, the names are changed or dropped during the CREATE TABLE step. Then when a NonStop SQL INVOKE statement creates an I/O buffer area, it generates names that differ from the original ENSCRIBE names.

The programmer could examine every line of program code and make all field names consistent with the new NonStop SQL names, but this is a long task, prone to mistakes. Instead, the programmer can adapt an existing COBOL85 program to the NonStop SQL system without having to change the field names.

ENSCRIBE COBOL85 programs to be converted contain a file description (FD) statement followed by a record description, usually copied from a library, that defines an I/O buffer in storage and describes the format of the file record. This definition usually contains group levels and RENAMEs. Leave this definition in the program, but move it to working storage.

Use the SQL INVOKE statement to define the equivalent I/O buffer and a description of the SQL table columns, called host variables. Convert ENSCRIBE READ statements to equivalent NonStop SQL SELECT statements. Following these statements, move the NonStop SQL table-column data to the working storage area described by the ENSCRIBE system.

When all columns are selected and the data types match, the move can be made with one MOVE statement specifying the 01 level names. If the original DDL definition contained FILLER fields, the definition must be changed, because these fields will not exist in the SQL table definition. The original COBOL85 code will now work correctly without having to modify the field names.

The same technique works when updates and inserts are made to a NonStop SQL table. First, be sure the code uses the working storage area described by ENSCRIBE. Then move that area to the NonStop SQL area before executing the particular NonStop SQL statement. This technique greatly reduces the amount of code modification required.

### Programming Changes

This section describes the required programming changes for each section of a COBOL85 program. Understanding these changes can help users to determine the scope of effort required to convert each program. The information must be extrapolated for other programming languages.

***Input/Output Section.*** Remove any SELECT statements for the ENSCRIBE files that are replaced by NonStop SQL tables. Retain the SELECT statements for the interprocess message files.

***Data Division: File Section.*** Remove any FD statements for the ENSCRIBE files that are replaced by NonStop SQL tables. Retain and move the DDL definitions of the ENSCRIBE files to working storage. Retain all the FD statements for the interprocess messages.

***Working Storage Section.*** Define the SQL DECLARE section using BEGIN DECLARE and END DECLARE SECTION statements. Within this area, include the host variables for the NonStop SQL tables by using INVOKE statements. Next, include an area for testing SQL return codes by using an INCLUDE SQLCA statement. Also, if desired, include an area for NonStop SQL statistics by using an INCLUDE SQLSA.

***Procedure Division: Declaratives.*** Remove all file error statements for the ENSCRIBE files replaced by NonStop SQL tables. Retain these statements for the interprocess message files.

For program consistency, users might want to place the NonStop SQL WHENEVER directives here. These directives control the code logic to be executed for the various conditions returned following the execution of NonStop SQL statements. They can be placed anywhere and changed anywhere in the program. They are compiler directives and have no impact on the run-time environment.

***Procedure Division: Program Logic.*** Remove all OPEN and CLOSE statements for the ENSCRIBE files replaced by NonStop SQL tables. Retain these statements for the interprocess message files.

Convert the ENSCRIBE READ, WRITE, UPDATE, and DELETE statements to their NonStop SQL equivalents. An ENSCRIBE operation that reads multiple records sequentially requires a cursor operation in NonStop SQL. A cursor must be DECLAREd, OPENed, the rows retrieved with a FETCH, and CLOSEd. Note that simply changing I/O statements does not allow the application to take advantage of the power and sophistication of NonStop SQL.

If the ENSCRIBE description is retained in working storage, after the NonStop SQL SELECT statement, move the host variables to the DDL area. Before an insert or update, move the DDL area to the NonStop SQL host variables. Use COBOL85 COPY or SQL SOURCE to move in the common error routine.

## Converting ENFORM Reports to NonStop SQL Report Writer

ENFORM reports must be rewritten using the NonStop SQL Report Writer product. Tables 2, 3, 4, 5, and 6 compare the features of ENFORM and the Report Writer.

**Table 2.**
ENFORM statements and NonStop SQL Report Writer.

| ENFORM statements | SQLCI with the Report Writer |
|---|---|
| AT END | REPORT FOOTING and TOTAL commands (only current report) |
| AT START | REPORT TITLE command (only current report) |
| CLOSE | RESET PARAM command, maybe CANCEL (nothing else comparable) |
| DECLARE | No user variables, user aggregates, or user tables; can use aggregates in SELECT for grouped rows only |
| DELINK | Not applicable |
| DICTIONARY | VOLUME command (not really comparable) |
| EXIT | EXIT command |
| FIND | SELECT and LIST commands (not really comparable) |
| FOOTING | PAGE FOOTING command (only current report) |
| LINK | SELECT...FROM TABLE1, TABLE2...(no LINK OPTIONAL—outer join) |
| LIST | SELECT, DETAIL, and LIST commands |
| OPEN | Not applicable |
| PARAM | SET PARAM command (to set in SQLCI, Command Interpreter parameters passed automatically) |
| SET | SET LAYOUT \| STYLE \| SESSION \| PARAM (no user variables or user tables) |
| SUBFOOTING | Not applicable |
| SUBTITLE | Not applicable |
| TITLE | PAGE TITLE command (only current report) |

**Table 3.**
ENFORM clauses and NonStop SQL Report Writer.

| ENFORM clauses | SQLCI with the Report Writer |
|---|---|
| AFTER CHANGE | BREAK TITLE command |
| AS, AS DATE, AS TIME | Same. Use in *print-item* |
| ASCD and DESC | SELECT...ORDER BY... clause |
| AT END PRINT | REPORT FOOTING command (no default) |
| AT START PRINT | REPORT TITLE command (no default) |
| BEFORE CHANGE | BREAK FOOTING command |
| BY and BY DESC | SELECT, GROUP BY, ORDER BY, and BREAK ON clause |
| CENTER | Only for column headings, titles, footings, and whole report body |
| CUM and CUM over | Not applicable |
| FOOTING | PAGE FOOTING command |
| FORM | PAGE and NEED clauses of *print-item* |
| HEADING | HEADING clause in DETAIL line |
| INTERNAL JULIAN-DATE | Not applicable |
| NOHEAD | NOHEAD clause in DETAIL line |
| NOPRINT | Use IF/THEN/ELSE or AS clause, or omit from DETAIL but include in ORDER BY |
| PCT PCT OVER | Use multiple-step queries |
| SKIP | SKIP clause of *print-item* |
| SPACE | SPACE clause of *print-item* |
| SUBFOOTING | Not applicable |
| SUBTITLE | Not applicable |
| SUBTOTAL | SUBTOTAL command (based on BREAK) |
| SUPPRESS | IF/THEN/ELSE of DETAIL command |
| System variable | Functions COMPUTE_TIMESTAMP, CURRENT_TIMESTAMP, LINE_NUMBER, and PAGE_NUMBER |
| TAB | TAB clause of *print-item* |
| TIMESTAMP-DATE<br>TIMESTAMP-TIME | AS DATE/TIME clause |
| TITLE | PAGE TITLE command (no default) |
| TOTAL | TOTAL command |
| WHERE | SELECT...WHERE clause |

**Table 4.**
ENFORM option variables and NonStop SQL Report Writer.

| | SQLCI with the Report Writer | |
|---|---|---|
| **ENFORM option variables** | **Command(s)** | **Option** |
| BLANK-WHEN-ZERO | — | — |
| BREAK-KEY | SET [SESSION] BREAK_KEY | Command |
| CENTER-PAGE | CENTER_REPORT | Layout |
| COPIES | OUT_REPORT... | SPOOL3 |
| COST-TOLERANCE | — | — |
| DATE-FORMAT | DATE_FORMAT | Style |
| DECIMAL | DECIMAL_POINT | Style |
| DISPLAY-COUNT | LIST_COUNT | Session |
| HEADING | HEADINGS | Style |
| LINES | PAGE_LENGTH | Layout |
| MARGIN | LEFT_MARGIN | Layout |
| NEWLINE | NEWLINE_CHAR | Style |
| NONPRINT-REPLACE | — | — |
| OVERFLOW | OVERFLOW_CHAR and TRUNCATE | Style |
| PAGES | PAGE_COUNT | Layout |
| PRIMARY-EXTENT-SIZE | — | — |
| READS | — | — |
| SECONDARY-EXTENT-SIZE | — | — |
| SPACE | SPACE | Layout |
| STATS | DISPLAY STATISTICS and STATISTICS options somewhat comparable | — |
| SUBTOTAL-LABEL | SUBTOTAL_LABEL | Style |
| SUMMARY-ONLY | Only by SELECT... GROUP BY with aggregate functions in *select-list* | — |
| TARGET-RECORDS | — | — |
| TIME-FORMAT | TIME_FORMAT | Style |
| UNDERLINE | UNDERLINE_CHAR | Style |
| VSPACE | LINE_SPACING | Layout |
| WARN | WARNINGS | Session |
| WIDTH | RIGHT_MARGIN | Layout |

## Conclusion

Every ENSCRIBE application is a candidate for conversion to the NonStop SQL system. The decision to convert must be determined on an application-by-application basis.

Conversion is expensive. It requires time, effort, and the commitment of staff and equipment. The costs should be carefully estimated and weighed against the benefits, which also must be carefully measured.

Before deciding on the actual method of conversion (file at a time, groups of files, or entire applications), users must first choose between two fundamental approaches to converting from an ENSCRIBE to a NonStop SQL system. In one approach, the user migrates to a new system as quickly as possible. In this approach, the user creates and loads NonStop SQL tables that match the an ENSCRIBE files and replaces the program I/O statements one-for-one with NonStop SQL equivalents. Although this approach keeps initial costs down, it yields the smallest return. In most cases, the converted application will be no better than the original.

The second approach is slower and more expensive but yields a far greater return. In this approach, the user redesigns and rewrites the application. The user can enhance the application, normalize the databases, and take advantage of the full complement of NonStop SQL functions. The converted application can serve as the basis of future expansion and enhancement or be used as a model for developing future applications.

**References**

Anon, et al. 1985. A Measure of Transaction Processing Power. *Datamation*. Vol. 31, No. 7.

*NonStop SQL Messages Manual (Release 1)*. Tandem Computers Incorporated. Part no. 82328.

Tandem Performance Group. 1988. Tandem's NonStop SQL Benchmark. *Tandem Systems Review*. Vol. 4, No. 1. Part no. 11078.

**Wayne Weikel** is a specialist in NonStop SQL. He has worked in Large Systems Marketing Support at Tandem for over three years.

**Table 5.**
NonStop SQL features not available in ENFORM.

| | | |
|---|---|---|
| NAME | Command | Assigns name for use in other commands. |
| CONCAT | Clause | Concatenates print items. |
| NEED | Clause | Prints lines conditionally, depending on space remaining on current page. |
| VARCHAR_WIDTH | Style option | Maximum number of VARCHAR characters to be printed in a field. |
| LOGICAL_FOLDING | Layout option | Items that do not fit on a line are split or moved as a whole unit to the next line. |
| WINDOW | Layout option | Defines window to view a vertical segment of a report. |
| WRAP | Session option | Lines not within device width are wrapped or truncated. |

In the first release of NonStop SQL, SQLCI and the Report Writer did not provide directly the capability of outer JOINS (LINK OPTIONAL), multiple-level group aggregates, or conditional aggregates. However, these operations can be performed using multiple-step queries.

**Table 6.**
ENFORM features not available in NonStop SQL.

| | |
|---|---|
| ENFORM has a PROCESS interface | A process can masquerade as a disk file to ENFORM. There is a well-defined interface to ENFORM. |
| ENFORM supports "compiled" queries | This functionally allows canned queries to be written and executed (with OBEY command) without having the source query available. |
| ENFORM allows programmatic manipulation of file names | Programs using the programmatic interface to ENFORM can manipulate the file names before calling ENFORMSTART.<br><br>Applications frequently contain configuration files that essentially are logical-to-physical name maps which they manipulate before running a query or opening a file. With NonStop SQL, since the user has no control over OPENs, name mapping is impossible.<br><br>It would be impossible to have a NonStop SQL table of define names to physical table name mapping, to SELECT from the table, then alter the process's defines, then query from a table based upon the new defines. |

# Concurrency Control Aspects of Transaction Design

One challenge for application programmers is to implement transaction designs for online transaction processing (OLTP) applications that have concurrent access to a shared database. When more than one transaction operates on the same data, the correct usage of locking mechanisms is critical to maintaining transaction isolation and acceptable user response times.

The NonStop™ SQL distributed relational database management system and the Transaction Monitoring Facility (TMF™) product provide locking options that give programmers flexibility when regulating contention among concurrent transactions. By using unique combinations of lock mode, granularity, and duration, transaction concurrency can be optimized for each portion of the database. In addition, NonStop SQL and TMF allow programmers to select from transaction design alternatives, such as splitting large transactions into several smaller ones, that can also enhance concurrency.

This article describes how NonStop SQL facilities can be used to regulate concurrency and thus optimize transaction isolation and performance. The article also describes techniques to estimate transaction lock wait times and to determine the chances of locks occurring. Based on these estimates, application designers can anticipate contention and accordingly alter the transaction design. This article assumes a NonStop SQL, TMF, and PATHWAY transaction processing system environment.

## Transactions and the ACID Properties

A transaction mechanism gives the programmer a simple way to handle exceptions and concurrency. Using a transaction mechanism, the programmer brackets a set of database operations with a pair of BEGINTRANSACTION and ENDTRANSACTION statements and declares that all the intervening operations are a group which should have the four properties of atomicity, consistency, isolation, and durability (ACID).

- *Atomicity* requires that either all of the transaction's operations are performed or, in case of an error, none of the operations are performed. For example, completed operations are undone or backed out in the case of a failure of any one operation.
- *Consistency* requires that the group of operations, taken as a whole, are a correct, thus consistent, transformation of the system and database state.
- *Isolation* means that transactions are not influenced by changes performed by concurrent transactions.
- *Durability* indicates that once the transaction completes successfully, its effects on the database will survive system and media failures.

At any time prior to the ENDTRANSACTION, the application programmer can cancel all the effects of the transaction by calling an ABORTTRANSACTION statement.

These four properties and their order combine into the acronym ACID. To address the topic of concurrency control, this article turns its attention to the third ACID property, transaction isolation, and the mechanisms to regulate it in NonStop SQL.

## Isolation of Concurrent Transactions

Isolation of transactions ensures accurate, successful operations on the database. Locking is the standard technique for providing transaction isolation. A lock is a gatekeeper on a record, table, or other object. A request accessing the object first acquires the lock. If the lock is busy, then the request waits until the lock is free.

There are three basic violations of the isolation property:

- A *lost update* arises when one transaction updates a data item, then a second transaction, unaware of the first transaction's update, updates the same data item based on that item's original value.
- A *dirty read* occurs when one transaction reads data which is not yet committed by another transaction. The second transaction subsequently further changes the data or aborts. The data read by the first transaction never was committed.
- A *fuzzy read* occurs when a transaction reads an item and then at a later time rereads it and gets a different value because some other transaction has updated it in the interim.

Concurrency refers to the number of transactions that are active at any time. The more transactions that are active, the higher the concurrency. The basic theorem of concurrency control is: If a transaction locks each object prior to accessing it and if it keeps the locks until the end of the transaction, the transaction is isolated from the effects of other transactions. Although there are some complex details to this theorem, it is used as the basis for all of today's database products. Simple locking may unacceptably restrict concurrency. To eliminate this problem, Tandem™ systems have automatic and manual ways to allow greater concurrency.

### Managing Data Locking

In Tandem systems, all locking is managed by the disk process. Each time an item is accessed by the disk process, the related locks are consulted and updated. For example, a transaction that accesses a table which is distributed on several nodes of the network will have locks on each node.

Locking is generally automatic and invisible to the application programmer. Most SQL programs have no explicit locking statements. At the end of each transaction, all locks are automatically released. The explicit addition of locking options and locking statements to an application or database design is necessary only when there is contention among concurrent transactions.

### Lock Modes

When two transactions merely read an item, they can access it concurrently without violating the ACID properties. The simplest optimization for achieving transaction concurrency is for Tandem systems to distinguish between exclusive mode and shared mode access to a lock, rather than simply locking an item when a transaction accesses it.

An exclusive lock on a single item can be acquired by one transaction only. This transaction can update the item without affecting the other transactions. Shared locks on a single item can be acquired simultaneously by more than one transaction.

The idea of having two lock modes can be expanded to include many more than these two modes; in fact, the Tandem Disk Process 2 (DP2) product internally uses over 20 different lock modes. However, for the application programmer, only the two options of locking a table in shared or exclusive mode are visible, and NonStop SQL appears to the programmer to have only shared and exclusive locks on records.

Shared locks cannot be used if data is updated. A shared lock on data selected by a transaction is converted to an exclusive lock if the data is subsequently updated or deleted by the transaction. When multiple transactions hold a shared lock on the data, none of them can update the data before all other transactions have released their locks. Further complications arise when two transactions try to update data on which they both hold a shared lock. The transactions wait for each other and cause a condition called deadlock.

Cursors, designed to retrieve multiple rows into a host language program, support both shared and exclusive modes. If one wants to retrieve data with the intent to update some or all rows, the FOR UPDATE OF clause must be specified.[1] In this case, all locks are exclusive.

The compatibility matrix for the two visible NonStop SQL lock modes is illustrated in Figure 1. This matrix shows which lock requests are compatible with existing requests and which ones must wait. Lock requests are compatible when new ones can be granted at the same time as existing requests.

Figure 1

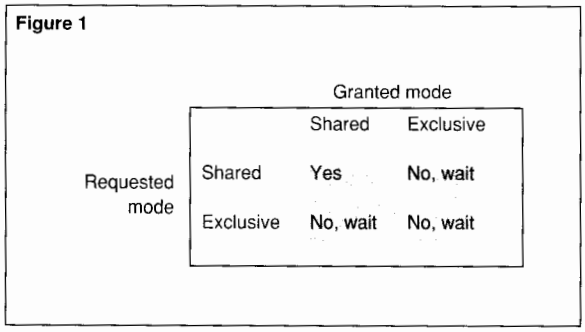



**Figure 1.**
*Compatibility matrix for NonStop SQL lock modes.*

## Lock Granularity

Lock granularity refers to the scope of a lock, from a single record to an entire table. It is defined as fine, intermediate, and coarse granularity for record, generic, and table locking, respectively. For example, interactive transactions accessing only a few records want to lock individual records and allow others to access other records. Small batch transactions and the NonStop SQL sequential block buffering logic want to lock a page or group of records. On the other extreme, batch transactions needing to scan an entire file or table want to lock the whole file without having to explicitly lock each individual record. Figure 2 illustrates how locking is affected by the granularity of the lock.

Figure 2

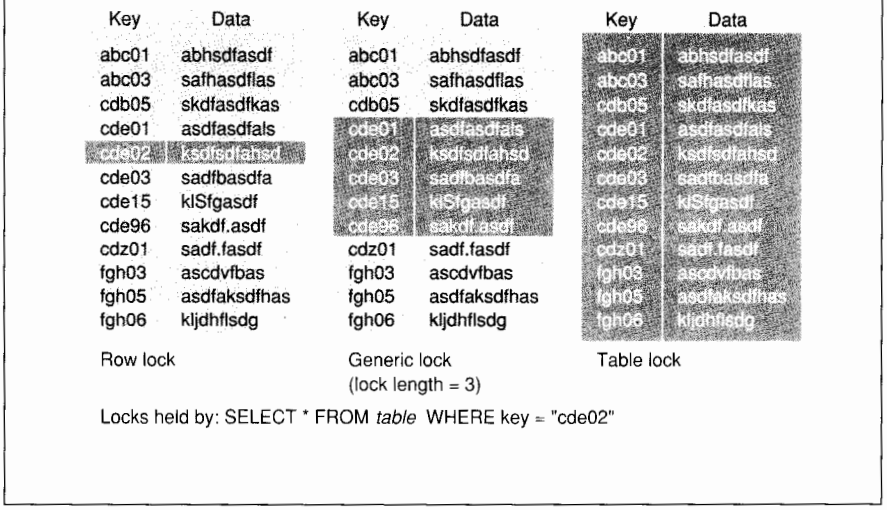



**Figure 2.**
*The number of rows locked by a single lock vary with the three locking granularities.*

Records are the finest granularity for locking in NonStop SQL. Record locking optimizes concurrency. It is the default choice for interactive transactions.

Generic locks, an intermediate granularity, is available on key-sequenced files. A generic lock covers a group of records that have the same key prefix. The database designer can specify that a table has generic locks by specifying the length of this prefix. If generic locking is specified, it is used as the finest unit of locking for all accesses to the table. The shorter the prefix, the coarser the generic lock. The length of the prefix can be altered to change the lock granularity by altering the table definition.

Partitions are the coarsest lock granularity. A partition lock covers all records in that partition of the table or index. Table locks are actually implemented by partition locks. Locking a table means that a corresponding lock is acquired on each partition of the table and on each partition of each index of the table.

## Choosing Lock Modes and Granularity

The NonStop SQL compiler[2] estimates an appropriate lock mode and granularity for each statement. For SELECT statements it uses shared mode, and for INSERT, DELETE, and UPDATE statements it needs exclusive mode. For SELECT commands that access a small subset of the data, the NonStop SQL compiler uses fine-granularity locking; if the entire table is to be accessed, it uses coarse-granularity locking.

[1]The FOR UPDATE OF clause also prevents a problem called the "Halloween" problem. This problem could occur if a column that is part of the access path to the data (even if this is not specified explicitly by an ORDER BY clause) is updated. The row may then be updated multiple times. NonStop SQL solves this problem by not using the columns that were specified in the FOR UPDATE OF clause in the access path.

[2]The NonStop SQL compiler reads the object program and transforms each SQL statement into an access plan.

If the NonStop SQL compiler estimates incorrectly, the disk process adapts. In particular, if the transaction acquires more than 512 locks (the limit for the C10 and the C30 releases of NonStop SQL) on a single partition, the fine-granularity locks are automatically escalated by the disk process to a coarse-granularity partition lock on that table.

The application programmer can influence these locking decisions with one of two SQL commands. The table may be explicitly locked with the command:

LOCK TABLE *name* IN [SHARE | EXCLUSIVE ] MODE

The programmer can alternately use this command to direct the NonStop SQL optimizer[3] to pick a certain lock mode:

CONTROL TABLE *name* TABLELOCK [ ON | OFF | ENABLE ]

Incidentally, if the program locks the table and does not tell the NonStop SQL compiler by way of a CONTROL statement, the NonStop SQL system uses the default locking protocol for other SQL statements on that table.

[3]The optimizer, a component of the NonStop SQL compiler, automatically selects the most efficient access plan for retrieving data from the database.

**Levels of Isolation**

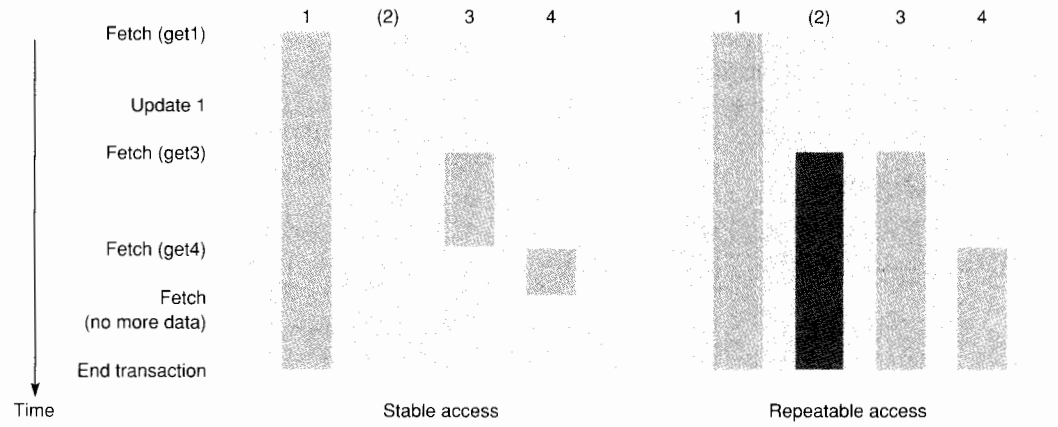
Operations such as INSERT, UPDATE, and DELETE that update the database have only one lock option, which is selecting the lock granularity. Beyond that option, the DP2 product enforces a lock protocol that automatically acquires exclusive mode locks for record, generic, or partition granularities and holds these locks until the completion of the transaction.

By contrast, SELECT statements can read the database at three levels of isolation: stable, repeatable, or browse access. These three options allow the application programmer considerable flexibility in trading off isolation and consistency by controlling the duration of locks acquired by a SELECT statement.

In NonStop SQL, a transaction can select different levels of isolation on a statement-by-statement basis. When considering isolation levels, determine what each level allows other transactions to do to the data being read by a particular SELECT statement.

***Stable Access.*** This access mode isolates a transaction from selecting dirty data (uncommitted updates or objects locked in exclusive mode). If the SELECT statement is submitted via a cursor, the record currently addressed by the cursor is kept locked until the cursor moves. This gives *cursor stability* (the terminology of Date and DB2 [Date, 1989]); the name "stable access" is thus derived from that term. Stable access is the default access mode to select data. Reducing the length of time an item is locked may increase concurrency, especially if the item is a "hotspot," an area where numerous transactions concentrate their access to the database.

Figure 3




| Empnum | Code | Salary |
|---|---|---|
| 1 | Y | 100 |
| 3 | N | 150 |
| 4 | N | 200 |

```
Declare abc cursor for selecting code and
salary from saltab for update of salary

Open abc cursor...

While anything to do
    Fetch abc into :code, :salary

    If code = "Y" then
        Update saltab set salary = salary * 0.5
        where current of abc cursor

Close abc cursor
```

**Figure 3.**
*Lock duration with stable and repeatable access.*

***Repeatable Access.*** Repeatable and stable access differ in the duration of the locks and in the ability to lock holes (positions in the database where records could be locked) in a range of rows. For repeatable access, all locks are held until the end of the transaction, whether or not rows are updated. Importantly, not only are the rows themselves locked, but also the holes between the rows. This prevents other transactions from inserting rows (called phantoms) into a range of rows already accessed by another transaction. This guarantees that repeated access to the data during the same transaction shows the data in exactly the same state as before, thus the term "repeatable access."

Figure 3 shows the difference in lock duration between stable access and repeatable access. Note that row 2 represents the hole between rows 1 and 3.

***Browse Access.*** This access mode does no locking at all. Data that is accessed is not locked, and data that is locked by other transactions can always be accessed. Browse access does not offer any form of data isolation. A particular transaction can read data that was updated but not committed by other transactions; possibly, the data will never be committed. If the tables are accessed for update by any concurrent transactions, browse access should not be used for making critical business decisions.

### Waiting for Locked Data

If a lock request from a transaction cannot be granted immediately, there are two options: to wait for the data to become available (wait option) or to return immediately with an indication that the requested data is locked (return option). The syntax controlling these options is:

```
CONTROL TABLE name

  {|{RETURN | WAIT} IF LOCKED|}
  {|TIMEOUT {value | DEFAULT} [SECONDS]|}
```

where

*value* = 0.01 through 21474836.47 or –1; the value –1 specifies an infinitely long wait time

The decision to select one option over the other depends on the length of the wait and the amount of time the transaction (or rather the end user) is prepared to wait.

The wait option is the default and should be used if the data is rarely locked for long durations. On encountering a lock, the transaction is suspended. This delay may be noticeable to the user; however, the transaction usually completes normally. If the wait is really a deadlock, the timeout passes and the operation returns an error indicator to the application. On encountering a timeout, the transaction is not able to complete normally. The restart mechanism in PATHWAY[4] can be used to hide this from end users, although the user will notice bad response time. In no case should the timeout value be set to infinite. If this happens, deadlocks remain unresolved; users must wait forever; and the durations of the locks become unpredictably long, causing unpredictable response times.

[4]The PATHWAY restart mechanism allows for restarting transactions based on the transaction context at the beginning of the transaction. If no terminal I/Os occur during the transaction, the restart is invisible to the user.

In general, the return option is only selected if there are locks, possibly caused by different types of transactions, that work on the same tables. Of these transactions, some may be batch jobs or may include operator think-times, which cause long-duration locks. The return option then results in predictable and short response times. However, if locked data is encountered, the transactions do not succeed.

In all cases, whether the wait option or the return option is used, the application must be prepared to handle the temporary unavailability of data. The wait option is a mechanism to deal with temporary data unavailability.

### Deadlock and Livelock

If one transaction waits for a second transaction and the second waits for the first, neither can proceed. Such situations are called deadlocks. The most common form of deadlock is two programs reading an item and having a shared lock on it and then each transaction wanting to update that record. This happens with REPEATABLE SELECT statements and with cursor-based STABLE SELECT statements.

NonStop SQL detects deadlocks by way of timeouts. If a wait lasts a long time, it is probably a deadlock. "Long" is hard to define in this case. If no user activity is required during the transaction, 10 seconds is long; if user activity is required, a minute can be short. The premise is that deadlocks are rare. Improper programming techniques, however, can increase the frequency of deadlocks.

Clearly, one wants to avoid deadlocks. The standard way to avoid them is to acquire resources in a fixed order. Because SQL is a nonprocedural language, most of the locking is automatic, and the programmer has relatively little control over it. Deadlocks can be avoided completely if table granularity locks are acquired explicitly, but this limits concurrency. In addition, one can also avoid deadlock by selecting data by way of cursors with the FOR UPDATE OF clause (described previously in the "Lock Modes" discussion).

An application must also be able to handle situations when it receives an error, either a lock timeout or a simple "is locked" return message, from a lock request. If the transaction resubmits the request and waits again, it runs the risk of livelock: two transactions that repeatedly bump into one another. This situation is worse than deadlock because it consumes a significant amount of computing resources and is not easily detected. Generally, the transaction should not immediately retry the operation. Instead, the transaction should abort and choose one of two paths: restart from the beginning after the abort completes or return to the user with an error indication and let the user decide whether to restart or not.

## An Analytic Model of Lock Wait Times

The different locking options can affect concurrency in different ways, and they eventually affect user response times. For example, the effect of one simple option, exclusive long-duration locks, on transaction concurrency and lock wait times is easily estimated. However, the impact of locking granularity and then isolation modes added to a simple locking scenario compounds the complexity of the calculation.

### Exclusive Long-Duration Locks

Exclusive locks have long duration due to their nature of holding the lock until the end of a transaction. If transactions encounter a row, a range of rows, or a table locked by another transaction, they have to wait for the release of the lock. The effect for the end user is an increased response time. It is then important to estimate the frequency of encountering a lock and to estimate the increase in response time when a lock, especially an exclusive lock, is encountered.

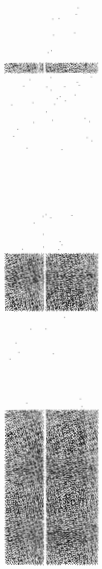
If queueing theory is applied to locking, the following formula can be used to estimate $W$, the average wait time in seconds that a transaction has to wait for the release of a lock:

$$W = ( O / (1 - O) ) \times LD$$

where

$LD$ = average duration of a lock, in seconds.
$O$ = occupancy, which is the percentage of time that the object is locked divided by 100.

**Figure 4**



Row locks: Occupancy $O$ = 0.002

Average wait-for-lock $W$ = 0.002 sec

Generic locks on client-num: Occupancy $O$ = 0.1

Average wait-for-lock $W$ = 0.111 sec

Table locks: Occupancy $O$ = 2

Throughput limited to 1 tps

**Assume:** Rows are accessed at random.
Each lock lasts 1 second.
Primary key is client-num / order-num.
One table of 1000 rows, with
20 clients, each having 50 orders.
Two transactions per second.

This formula assumes a random arrival rate for the lock requests and an exponentially distributed lock duration. For fixed lock durations, the average wait time is shorter. The following formula takes the more pessimistic approach.

Occupancy can be calculated as:

$$O = ( LT \times LD \times T ) / R$$

where

$LT$ = locks per transaction.
$T$ = number of transactions per second.
$R$ = number of rows in the table.

**Figure 4.** *The effect of lock granularity on concurrency when a table is accessed randomly.*

As an example for estimating an average wait time, assume that exclusive row locking is used on a table of 1000 rows, all rows are accessed with the same frequency, and each lock lasts 30 seconds ($LD = 30$). In that case, an arrival rate of 1 lock per second ($LT = 1, T = 1$) yields an occupancy ($O$) of 0.03, meaning that 30 out of 1000 rows are locked. The average wait time is thus about 0.93 second. If the arrival rate becomes 10 per second, the utilization is 0.3, and the average wait time is about 13 seconds. The values in this example are extreme, but they emphasize the effects of the variables.

For low utilizations, most transactions do not have to wait for the release of the locks. The percentage of the transactions that have to wait is determined by the occupancy value. The average wait time for these transactions is about 30 seconds for exponentially distributed lock durations. For fixed-lock durations, the wait time is half of the lock duration; for the example above, it is 15 seconds. In all cases, the end users will notice unpredictable response times if the transaction has to wait for a lock, even though average response times may be acceptable.

**The Impact of Lock Granularity**

The different degrees of lock granularities can affect lock wait times. As a general rule, the coarser the granularity, the greater the chance for lock contention. This is because the occupancy of the data increases with the coarser granularity. If contention is a problem, use the finest granularity available. Figure 4 illustrates the effect of lock granularity on the number of records that are affected by a single lock.

Generic locking is especially useful if access is pseudo-sequential; that is, if most transactions are accessing a set of records with the same prefix. Generic locking then helps in minimizing the number of locks held by the disk process because all records that are accessed have the same prefix. Also, it helps to prevent deadlocks if access in the generic subset is not in a predefined order for all transactions, because the first transaction acquires a lock on the complete generic subset.

The calculations for concurrency on generic or table locks are basically the same as those for row locking, but they are applied instead to the generic sets of records that can be locked. If generic locking is done on groups of 100 rows, then the occupancy for the group is 100 times as high as it would be for each individual row.

While many applications access data randomly across the range of items in the tables, some applications may concentrate access in particular key ranges, causing hotspots. The granularity chosen for access to the table should be fine enough to allow for sufficient concurrency at these hotspots.

NonStop SQL tries to convert row locks or generic locks to table locks if a transaction acquires many (currently 512) locks. This is typically done for batch transactions. Once a table lock is acquired, that lock stops all concurrent transactions on that table. For this reason, NonStop SQL should not use table locks for batch processes that are running during OLTP processing. The SQL statement that sets this parameter is CONTROL TABLE *name* TABLELOCK OFF.

### The Impact of Isolation Modes

The case of the simple, but most pessimistic, approach of using exclusive access to the data was presented initially. When the impact of adding both locking granularity and different locking modes is considered on lock wait times, the model for estimating wait time becomes much more complicated. If shared access is used, there are fewer waits. However, the most important consideration is that the chances of waiting increase with the square of the degree of multiprogramming and with the square of the transaction size (Gray, 1981; Tay, 1987). So, to avoid contention, one wants to control the number of concurrent transactions as well as the transaction size (the duration and number of locked resources).

Within this scope, a common application is one that is dominated by read-only transactions that are not allowed to read dirty data. Therefore, the default of stable access and shared locks is applicable. A small percentage of the transactions, however, update the database; these transactions have a longer duration.

A possible implementation of the application uses the default access modes of stable access and shared locks for the read-only transactions, and it uses exclusive locks with repeatable access for the update transactions. The read-only transactions will encounter contention with the update transactions. The amount of contention can be determined by the average lock duration of the exclusive locks and the occupancy of these locks on the data. (See Figure 5.)

Figure 5

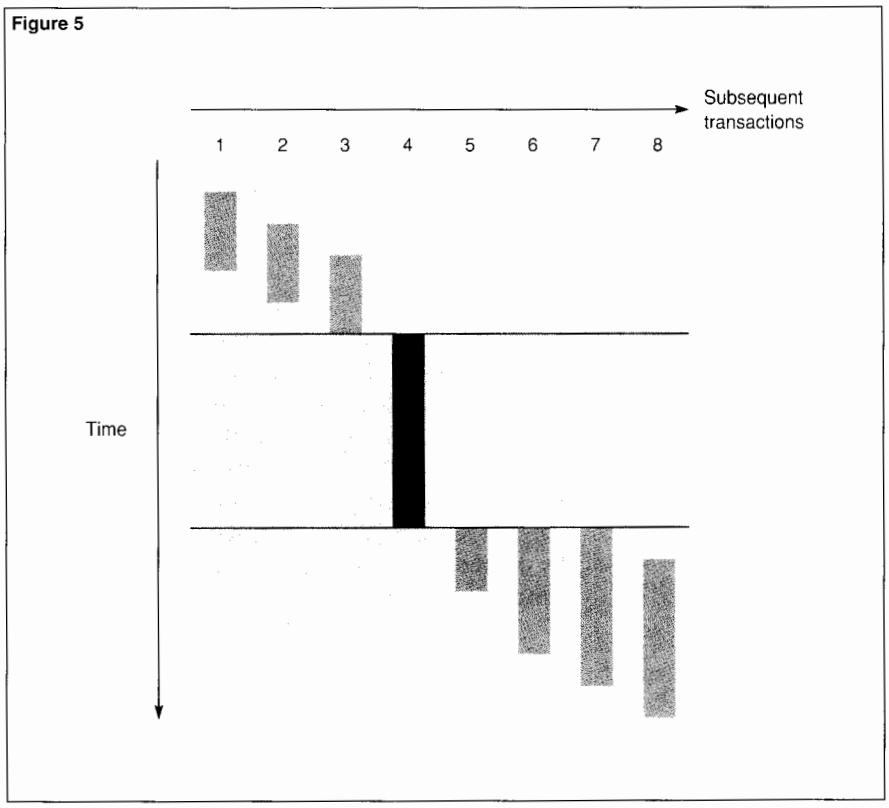



**Figure 5.**
*Waiting for exclusive locks. The read-only transactions (shaded dark grey) have to wait for the update transaction (shaded black). Note that the wait time (in white) depends only on the length of time which the update transaction holds its exclusive locks.*

## Transaction Design

The occupancy of the objects being locked is determined by the frequency of locks on the object and on the lock duration. The duration can vary with different transaction designs. For batch processing, there are two choices: running one large transaction or breaking a large transaction into many small batch transactions. For interactive processing, a designer can pick one of three transaction designs:

- *One-shot transactions* consist of a single context-free message in, some data processing, and a single response message out.
- *Pseudo-conversational transactions* involve multiple context-sensitive interactions with the terminal or client. Each interaction is structured as a single TMF transaction.
- *Conversational transactions* consist of multiple interactions with the client or terminal. A single TMF transaction covers the entire sequence of message and database operations.

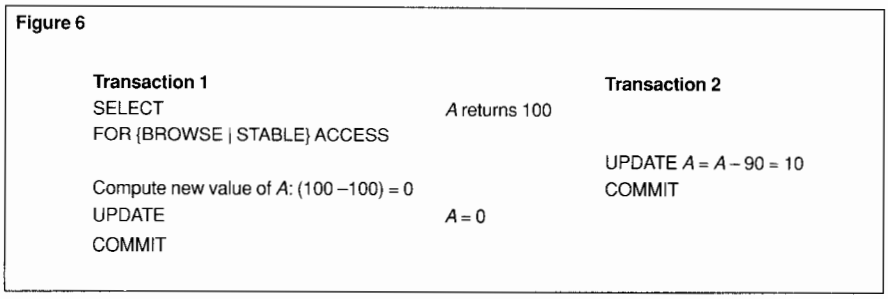



**Figure 6.**
*An example of an occurrence of the lost update problem.*

The most frequent designs are one-shot or pseudo-conversational transactions. These designs minimize lock durations and consequently maximize concurrency. Although the conversational transaction design is the most general and logical to program and use, very few OLTP applications use it because it implies lock durations that allow for operator think-time, which is measured in minutes or hours.

## The One-Shot Transaction

One-shot transactions are the typical case in OLTP systems. In general, these transactions involve a single requester (a PATHWAY terminal control process), a single server (application process), and one or more disk processes supporting the SQL tables. The programs typically have no special locking statements in them. They accept the default locking of NonStop SQL, which is cursor stability, automatic granularity, and a 60-second timeout for lock waits (the number of times a request had to wait for the release of a lock).

These default lock modes are selected to provide a trade-off between isolation and concurrency. Because repeatable access is not the default, the application runs some risk of experiencing lost updates and fuzzy reads. If data chosen by a SELECT statement via a short-duration lock is updated, the lost update problem may occur; that is, the data value may have been changed by another transaction since the SELECT operation. The update by the first transaction may overwrite the other update.

Figure 6 presents an example of this situation. Transaction 1 reads the value of *A* as 100, but by the time the update is applied, the value is really 10. There are many ways to correct the lost update problem. The simplest approach is to always use repeatable access. If this is unacceptable, one can use cursor-based SELECT statements with stable access. A third technique involves only using update expressions, rather than literals, in the update. This approach would also solve the problem above. Transaction 1 can code the update as UPDATE ... SET $A = A - 100$, rather than UPDATE ... SET $A = 0$.

A related technique compares the current value or version of the record with its value at the time of the original SELECT operation. If the value or version has changed, an update of the database will be denied by NonStop SQL. The transaction may then reread the data and retry the update.

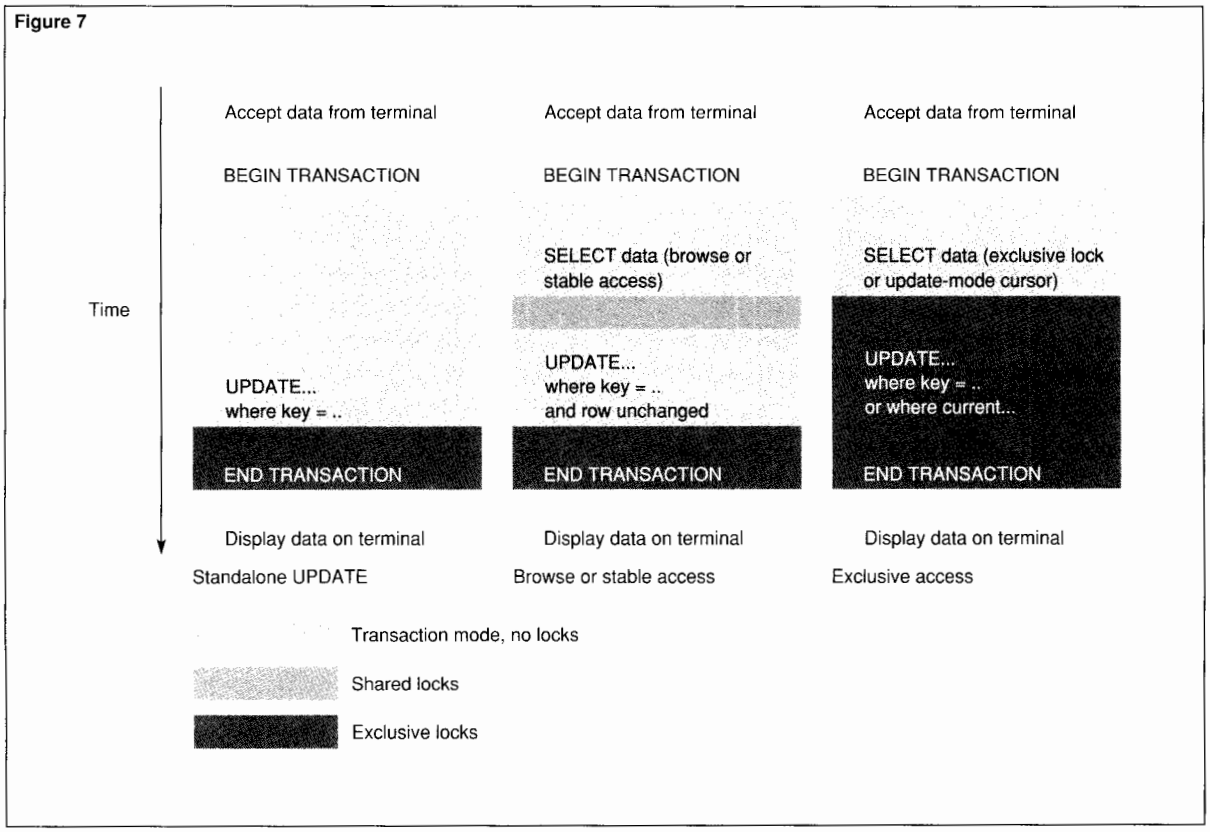



**Figure 7.**
*Three different scenarios for implementating one-shot transactions.*

Given that the program logic is correct and that isolation is not violated, it is possible to minimize lock durations by writing the same application in many different ways. The simplest way is to defer all updates to the end of the transaction, using only browse mode locks in the early steps of the transaction. Of course, such designs must consider the problems of lost updates and dirty data. Such programs can be a maintenance problem, because later changes to the application may violate assumptions that a particular program makes about the behavior of others.

To improve concurrency, it is generally best to place exclusive locks as late as possible in the transaction and minimize the number of shared mode locks. Figure 7 shows the three strategies for updating a record: (1) package the selection and updating operations as a single operation, (2) select the data with a shared lock, or (3) acquire an update mode or exclusive mode lock on the item when it is selected.

***Scenario 1.*** In the first scenario of Figure 7, a standalone UPDATE statement updates the data. This scenario is suitable for simple transactions such as withdrawing money from an account. The UPDATE statement can explicitly check if the transaction is allowed (for example, checking the account balance) or if the database administrator has defined constraints.

***Scenario 2.*** In the second scenario of Figure 7, a standalone SELECT (no cursor) operation reads the data. This way the transaction can programmatically access the data and can also determine if an update will be done. The SELECT operation uses browse or stable access. No locks are held on the data after the SELECT operation. In this case, checks that determine the item has not changed can be included in the WHERE clause of the UPDATE statement. This standalone SELECT operation is useful if all SELECT statements do not lead to an update and if the concurrency requirements are relatively high.

***Scenario 3.*** In the third case, the data is selected in exclusive mode to the end of the transaction and possibly updated later during the transaction. This option is useful if concurrency requirements are not extreme. If a cursor is used and the record is not a candidate for an update, it is unlocked when the cursor moves; this allows for more concurrency.

In these scenarios transactions are not likely to last long. Concurrency problems may arise only if both the number of database accesses requiring exclusive access and the concurrency requirements on database hotspots are very high.

## Simple Pseudo-Conversational Transaction Design

A simple transaction can illustrate the different approaches to designing a pseudo-conversational transaction. In this transaction, the operator requests information (a record) from the database; once this information is displayed, the operator can enter an amount to be subtracted from the balance field of the record that was read in the first step of the transaction. The server then applies this change to the database. Even for such a simple transaction, the isolation aspects may vary greatly between various implementations. It is important to note that the end user notices concurrency aspects in different ways; most noticeably, in response times.

If the options for specifying the lock modes and the exclusion modes are omitted, the three SQL statements for this sample transaction consist of a SELECT statement and one of two UPDATE statements:

```
SELECT CLIENTNAME, BALANCE
FROM ACCOUNT
WHERE ACCNR = :ACCNR;
```

and

```
UPDATE ACCOUNT
   SET BALANCE = BALANCE – :AMOUNT
WHERE ACCNR = :ACCNR;
```

or

```
NEW-BALANCE = BALANCE – :AMOUNT
UPDATE ACCOUNT
   SET BALANCE = :NEW-BALANCE
WHERE ACCNR = :ACCNR;
```

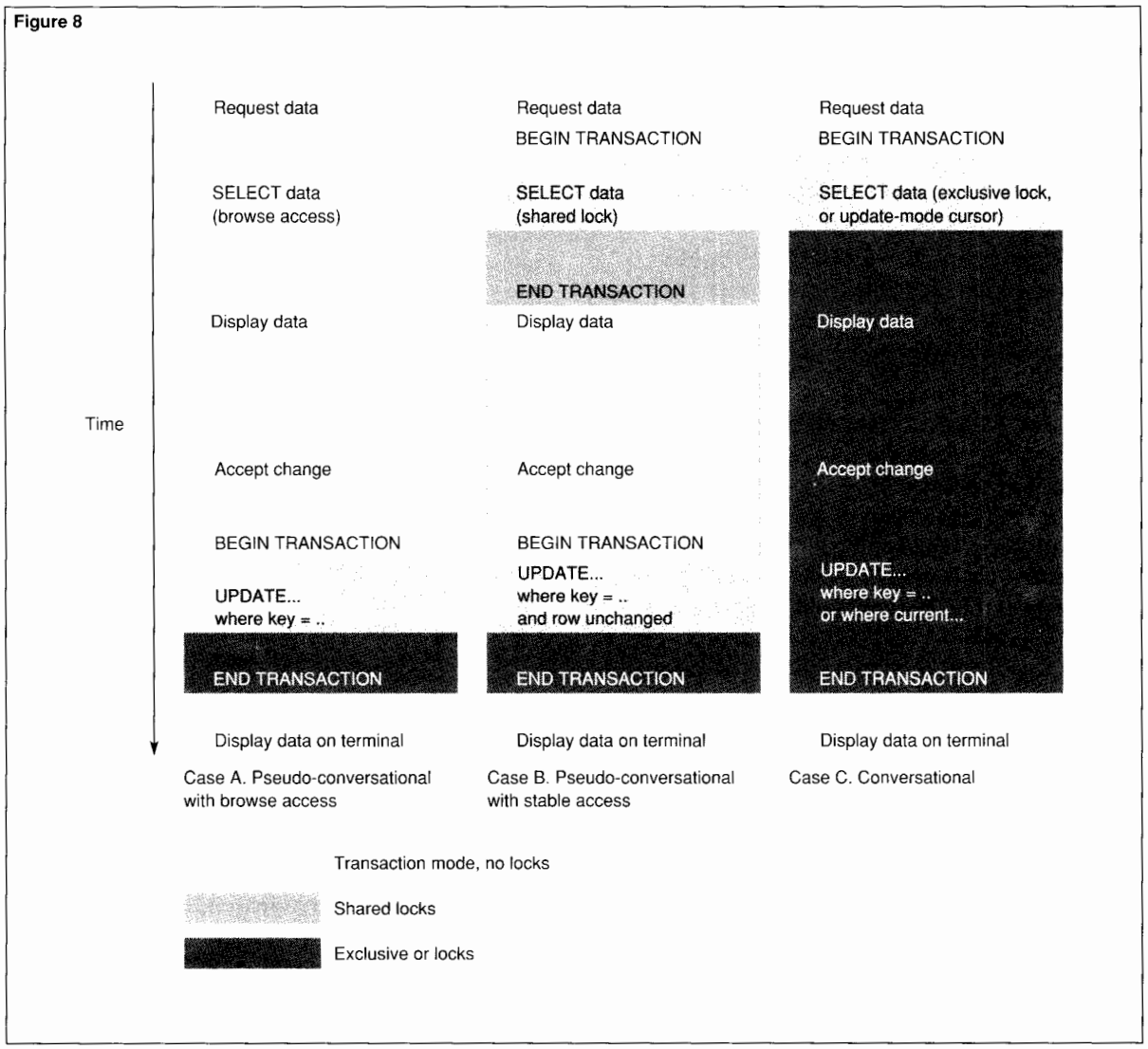



**Figure 8.**
*Pseudo-conventional and conversational transaction design.*

***Using Browse Access.*** Using this method allows for maximum concurrency. Case A in Figure 8 gives an overview of browse access. The data that is retrieved by one or more SELECT statements may not yet be committed by another transaction, which is also updating the same record with an INSERT, DELETE, or UPDATE statement. Thus the application and user may see data that is incorrect. Even though the data may not have been correct as it was selected, the UPDATE operation can check if the data that was selected is the same as the actual data in the database when the UPDATE is done. This requires some extra tests in the WHERE clause of the UPDATE statement:

```
UPDATE ACCOUNT
   SET BALANCE = BALANCE + :AMOUNT
WHERE ACCNR = :ACCNR AND BALANCE
   = :OLD-BALANCE;
```

In this case, the end user never has to wait while the data is selected. The UPDATE operation, however, can either complete normally, perhaps after a wait, or it can be empty because the row has been changed or deleted by another transaction. The application needs special logic to handle this denial.

Several alternative designs are possible:

- If insufficient funds is the the only reason for denying a withdrawal from an account, the test BALANCE > 0 could be declared as a constraint on the account table. In this case, NonStop SQL will deny an attempt to withdraw an amount that would create a negative balance.
- If having a negative balance would be less cumbersome than denying a transaction that at a first glance (using browse access) looked positive, checking for a change in the balance could be omitted.

In all cases, the actual business requirements with respect to handling of the transaction determine which option to choose.

During the second part of the transaction, the program checks whether the database is still in the same state it was when the program read the data. The program must remember the original value. The section titled "Pseudo-Conversational Transactions" discusses this more general subject of keeping context.

***Using Shared Locks and Stable Access.*** Using this method allows for medium concurrency. Case B in Figure 8 gives an overview of this technique. The data retrieved by the SELECT statement is guaranteed not to be dirty. Thus, the user does not see incorrect data. With this method, the data may still be changed by another transaction between a SELECT and an UPDATE operation. If a decision based on outdated information is not acceptable, the UPDATE operation has to check that the data has not been changed since the SELECT statement.

A TMF transaction is needed to acquire a shared lock. This transaction is ended prior to replying to the terminal so that no locks are held for the duration accommodating an operator's think-time. When the reply is read from the operator, a new transaction is begun and the data revalidated by a program check just as in the paragraph above. This costs an extra TMF transaction, which translates into a higher demand on the CPU and higher audit trail activity.

**Simple Conversational Transaction Design**

This design method does not optimize concurrency. However, users can be sure that once the data is shown on the screen, they can update it without any risk of another concurrent update. The only place where this transaction may have to wait for a lock to be released is at the time the data is selected. Case C in Figure 8 gives an overview of this technique. The disadvantage of this option is its minimal concurrency caused by using repeatable and exclusive access SELECT operations or update mode cursors involving long-duration locks. However, the advantages, with respect to ease of programming, are very significant:

- The data is guaranteed to be current and available at the time of the UPDATE operation.
- There is no need to check for such things as versions during the UPDATE operation.
- If updating is done on numeric fields, both absolute settings and relative settings can be used; for example, BALANCE = :NEW-BALANCE can be used as BALANCE = BALANCE – :WITHDRAWAL.

As was mentioned before, conversational transactions are the most logical to use. If concurrency is not an issue (a heavy restriction), using conversational transactions may minimize development efforts.

To prevent extremely long transactions, all ACCEPT statements to the user terminals must use a timeout. Extremely long transactions decrease concurrency and may even cause TMF to abort the transaction automatically if the MINFILES[5] limit is reached. Note that there may also be other reasons to handle timeouts.

### Conversational and Pseudo-Conversational Transaction Design

If the database cannot be brought from one consistent state to another within one terminal interaction, the conversational transaction size increases. Typically these transactions select and update multiple rows within multiple tables. This leads to longer locks and to potentially less concurrency.

To improve concurrency, one can implement the conversation as a pseudo-conversational transaction. Lock durations are much shorter for pseudo-conversational transactions than for a pure conversational transaction design because no locks are held during operator think-times.

Several techniques are available to implement conversations. Figure 9 illustrates a conversational transaction, a pseudo-conversational transaction, and an approach to conversational transactions called collected updates. Keep in mind that these techniques are all subject to the lost update problem, dirty reads, and fuzzy reads. They require careful design and careful program maintenance to avoid serious business errors.

[5]MINFILES is a TMF configuration parameter. TMF writes before and after images and commits records to a rotating set of audittrails. Transactions cannot span more than MINFILES audittrails.

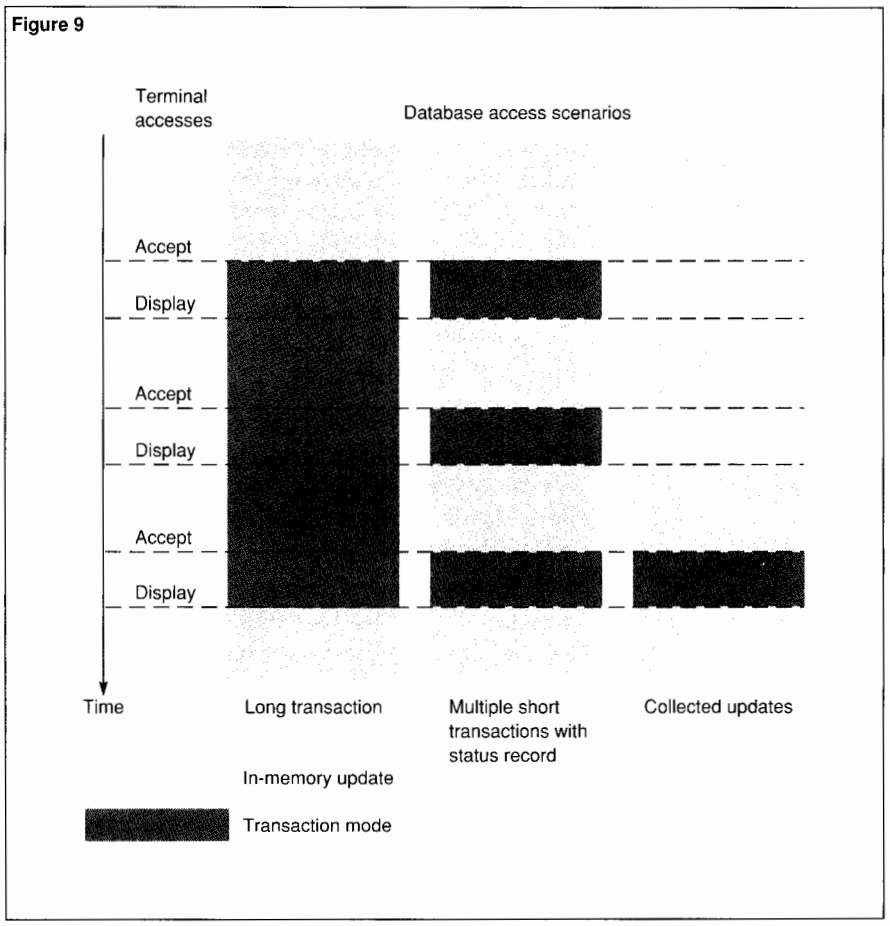



**Figure 9.**
*Implementations of transactions with multiple terminal I/Os.*

***Conversational Transactions.*** An easy way to develop transactions with multiple terminal I/Os and database accesses is to define one TMF transaction that covers all the operations; for example, all operations involved in entering order information. If, instead of a single TMF transaction, the TMF transaction must span entering of data on one screen and entering of order details on others, the transaction may last a significant length of time. This has three negative effects that often prevent conversational transactions from being implemented successfully:

- All locks last for a significant length of time. This may become a problem if locks are held on certain key data tables that are updated when information is entered. Also, deadlocks on key tables are hard to prevent.
- In case of a failure that causes TMF or the program to abort the transaction, all data being entered is lost. While preserving database consistency, it is very unfriendly to the end user. The additional programming effort to hide this occurrence from the end user can be significant.
- A backout of an individual subtransaction cannot be handled by TMF but must be handled programmatically. Only the complete transaction can be backed out with TMF.

The positive aspect of this approach is that database integrity can be easily maintained. Nonetheless, if conversational transactions are used and predictable response times are required, the default method of waiting for locks cannot be used. Instead the return option should be used (as discussed previously).

***Pseudo-Conversational Transactions.*** If a long transaction is split into short subtransactions, database integrity is harder to guarantee in case of failures. One method that can be used to maintain integrity is to maintain a status of the transaction in an audited table. This status information is usually called the pseudo-conversational transaction context table. This table contains an entry for each in-progress pseudo-conversational transaction.

Typically, the transaction context table has a list of database changes by the transaction, current cursor positions, and context information associated with the terminal. If the operator requests cancellation of a previous step or of the entire transaction, the server uses the context information to logically undo all the operations of the transaction.

An advantage of this application-managed context table is that it is possible to back out individual subtransactions with TMF. Larger units can be backed out by the application. The disadvantage is that it requires additional application design, implementation, and testing.

Using a separate context table provides a solution for developing pseudo-conversational transactions. However, many applications allow for a simpler approach. For example, a flag in an orderheader record can indicate that work on an order is in progress. Other transactions should not access the header or detail records for the order if the flag is set. If orderdetail records are added to an order, the detailrecords themselves can be used as the list of database changes. In this case, the context handling is designed into the database. This approach complicates the initial database design and the structure of the database but simplifies the transaction design.

The transaction context table can also be maintained in the memory of the program. In a PATHWAY environment, the terminal control process (TCP) is responsible for maintaining the context table in memory. If the system fails, the context is lost. If the TCP is configured as a NonStop process pair, the context table can survive CPU failures at the cost of extra checkpoint messages.

***Collected Updates.*** A related approach to conversational transactions is to postpone all database updates to the end of the transaction. Until the last step of the transaction, the database selects information (typically through browse or stable access). All update requests from the end user are not applied to the database but are stored in a memory area similar to the transaction context.

During the last step of the transaction, all updates are applied within one short TMF transaction. As with the pseudo-conversational transaction design, collection of the updates must be handled by the application program.

Two considerations must be weighed when evaluating this approach. One very significant obstacle is the limited size of the memory area, however large it may be. If the updates do not fit in the memory area, they are rejected unless additional coding to handle buffer overflows is developed. All complications of writing pseudo-conversational transactions apply if the decision is made to write more code.

A second issue is keeping the updates during the complete transaction. If the context table is kept in memory, it will not survive a downed system. It may not survive a single downed CPU if the context is not checkpointed. So, just like conversational transactions, the user must start from the beginning again.

## Concurrency and Consistency Testing

When conducting concurrency tests, the use of a large, life-size database is very important for obtaining accurate, reliable results. Furthermore, the normal load on the system should be simulated. The number of terminals that are used in testing should be the same size as the number of terminals that are connected to the real system. If the transactions include terminal I/Os, the expected operator think-times should be simulated. To test for database hotspots, a realistic distribution of key values should be used.

If a large database is unavailable, another approach would be to simulate only the hotspots in the database. In that case a smaller database can be used, and only the transactions that access the hotspots need to be simulated.

When evaluating the results of concurrency tests, both the average response times and the distribution of those response times produced during testing are important to access. If a high percentage of the transactions complete in a relatively short time, but the remaining transactions have an average response time that is much higher than this value, this indicates that a relatively small number of transactions must wait for locks that are outstanding for a relatively long time. In many cases, this is still unacceptable. The analytical model presented here can be applied to examine the cause of the problems.

For larger applications, concurrency testing is not trivial. If new transactions are added to the application, two cases must be examined. The new transactions have to be evaluated, and the existing transactions must be reviewed in the changed environment as well. For example, if a new transaction is added to the system, existing transactions may encounter deadlocks. Automated test procedures for both consistency and concurrency testing seem the only adequate way to ensure that the tests can be repeated over again in a changing environment.

Information about concurrency can be obtained from various places in the system. The File Utility Program (FUP) and the operator interface of TMF (TMFCOM) are suitable to gather information about the current state of the system, and the MEASURE™ system performance measurement tool can be used to obtain statistical information. SQLCI, the NonStop SQL conversational interface, provides statistical information about data in the database and about individual SQL statements.

### FUP

The LISTLOCKS command can be used to obtain a list of the locks that are currently granted or waiting. Information can be retrieved per disk volume or per table. The displayed information shows the lock list per object that is being locked: the objects are identified by their primary key.

The LISTLOCKS command is especially useful for investigating locks with a long duration, or even deadlocks. If deadlocks are detected by using the default timeout mechanism, the locks disappear after a relatively short period (within one minute). Thus, in many cases, the deadlocks vanish before an investigation can be done. Because of this, it might be useful to increase timeout values to a longer, possibly infinite, period during testing. This way the operational staff might be able to detect the cause of the deadlocks. Note, however, that a very long or infinite timeout value is totally unacceptable for production environments. Deadlocks would hang the system, making them visible to the user.

### TMFCOM

The STATUS TMF command provides the current transaction rate, and the STATUS TRANSACTION command shows the number of active transactions. By using the following formula, the results of these two commands can be used to determine the average duration of the TMF transactions:

Average transaction duration
= active transactions / transaction rate.

Note that the result of this formula provides only an average. It can be built up from transactions that complete very quickly and from transactions with a very long duration.

### MEASURE

The DISC, DISCOPEN, TMF, and SQLSTMT entities all contain information about concurrency.

DISC and DISCOPEN reports show the number of requests per disk volume and per opener of a file that had to wait for the release of locked data. This information is stored in the REQUESTS-BLOCKED counter. Note that the counter does not reveal any information about the duration of the waits.

TMF reports present statistical information about the transaction rate and the duration of the transactions. Also, the percentage of the transactions that were aborted can be determined.

An SQLSTMT (NonStop SQL statement) report shows information per SQL statement. The information includes:

- Escalations of locks to table locks.
- Lock waits.
- Timeouts. These do not have to be caused by deadlocks. However, in a well-tuned system with reasonable timeout values, it is not likely that timeouts are caused by other reasons.

Also, the average elapsed time for the execution of the statements is presented in the ELAPSED-TIME-BUSY counter.

### SQLCI

SQLCI, a dynamic NonStop SQL application, allows static and dynamic SQL statements to be executed interactively.

The SELECT statement can be used to obtain statistical information about the distribution of values in the primary keys. This information can be valuable in determining the lock granularity.

The DISPLAY STATISTICS command provides information about such things as the estimated costs or the number of records that are accessed per table.

Similar statistical information about the execution of individual embedded SQL statements in a host language can be obtained by examining the NonStop SQL statistics area (SQLSA). The information includes lock waits and escalations that can be examined with MEASURE tool on a statistical basis as well.

## Conclusion

Transaction design for the ENSCRIBE and NonStop SQL systems is similar. Both systems encourage one-shot, pseudo-conversational, and, for batch processing, small batch designs in preference to conversational and single-transaction batch designs. The goals are to have highly concurrent access to shared data and to minimize the granularity and duration of locks. Simple locking mechanisms in ENSCRIBE forced most developers to design short TMF transactions. NonStop SQL and TMF offer flexibility in regulating concurrency primarily through the use of lock modes, lock granularities, and the duration of locks.

Two lock modes can be specified. Shared locks allow for multiple owners of a lock, and exclusive locks allow for only one owner of a lock. Once a row is updated, the lock is always exclusive.

Lock granularity, the scope of a lock, can be regulated by specifying row locking or generic locking on a per table base. Generic locking can often be used on tables without a negative effect on concurrency. If generic locking can be used, it may reduce the total number of outstanding locks and the chance of escalation to table locks. NonStop SQL may escalate fine-granularity locks to a table lock once it has acquired a large number of locks for one transaction on a table. Conversion of row locks or generic locks to table locks may virtually stop other application processing until the table lock is released. NonStop SQL can be directed not to escalate to table locks.

Lock duration can be regulated by specifying an access mode on SELECT statements. These options are stable access, with short locks; repeatable access, with long locks; or browse access, with no locks at all. Stable access may cause lost updates and fuzzy reads if the version of an updated row is not compared with the version at the time the row was selected. The combination of repeatable or stable access and shared locks may cause deadlocks or lost updates. These deadlocks can only be resolved with a timeout mechanism. Deadlocks can often be avoided by accessing the data via cursors with the FOR UPDATE OF clause. This acquires exclusive mode locks on the data.

The chances that a transaction will have to wait for locks to be released can be estimated on a quantitative basis. The duration of the waits can also be estimated. For short transactions, waiting for the release of locks has a predictable effect on response times; the magnitude of the effect depends on both the occupancy of the locked data and the lock duration. Hotspots in a database can present concurrency problems in that some applications experience lock contention. In those cases, there are many techniques to minimize lock duration.

There are a number of transaction designs. The most typical transaction for OLTP applications is the one-shot transaction design. Conversational transactions, although the most natural to use, are generally not recommended, as concurrency can present problems. Pseudo-conversational transaction designs are optimizations for conversational transaction design. This design splits up large transactions into many short ones, and it implies that the context of the conversation is stored safely, preferably in an audited table, or that the database is designed to hold the context itself.

NonStop SQL allows the several new locking options discussed here that give the application designer more design freedom. Designers must note, however, that if isolation is not correctly designed, it may limit concurrency as well as system throughput and response times, or it may lead to database inconsistencies.

**References**

Bernstein, P.A., Hadzilacos, V., and Goodman, N. 1987. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley.

Date, C.J., and White, C.J. 1989. *A Guide to DB2*, 3rd ed. Addison-Wesley.

Gray, J.N., Lorie, R.A., Putzolu, G.R., and Traiger, I.L. 1976. *Granularity of Locks and Degrees of Consistency in a Shared Data Base.* Proceedings IFIP TC-2 Working Conference on Modelling in Data Base Management Systems (ed. G.M. Nijssen). North-Holland.

Gray, J., Homan, P., Obermarck, R., and Korth, H. 1981. *A Strawman Analysis of the Probability of Waiting and Deadlock in Database Systems.* IBM TR RJ 3066.

Tay, Y.C. 1987. *Locking Performance in Centralized Databases.* Academic Press.

**Acknowledgments**

The author thanks all those who have contributed in the process of writing this article, particularly Mark Anderton, Moore Ewing, Bart Grantham, and Mike Noonan, who provided reviews. Special thanks go to Jim Gray for his contributions in presenting the material in this article.

**Wouter Senf** joined Tandem in 1984 after five years of experience in software development for two software houses. He works in the Dutch office as an advisory analyst in the areas of application and database design and performance.

# Tandem Systems Review Index

## March 1990

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 1, Number 2 | 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 this issue, 13 issues of the *Tandem Systems Review* have been published:[1]

| | | |
|---|---|---|
| Volume 1, Number 1 | February1985 | Part no. 83934* |
| Volume 1, Number 2 | June 1985 | Part no. 83935* |
| Volume 2, Number 1 | February1986 | Part no. 83936** |
| Volume 2, Number 2 | June 1986 | Part no. 83937* |
| Volume 2, Number 3 | December1986 | Part no. 83938 |
| Volume 3, Number 1 | March 1987 | Part no. 83939 |
| Volume 3, Number 2 | August 1987 | Part no. 83940 |
| Volume 4, Number 1 | February1988 | Part no. 11078** |
| Volume 4, Number 2 | July 1988 | Part no. 13693** |
| Volume 4, Number 3 | October 1988 | Part no. 15748 |
| Volume 5, Number 1 | April 1989 | Part no. 18662 |
| Volume 5, Number 2 | September 1989 | Part no. 28152 |
| Volume 6, Number 1 | March 1990 | Part no. 32986 |

The articles published in all 17 issues are arranged by subject below. (*Tandem Journal* is abbreviated as TJ and *Tandem Systems Review* as TSR.) A second index, arranged by product, is also provided.

### Index by Subject

| Article title | Author(s) | Publication | Volume, Issue | Season or month and year | Part number |
|---|---|---|---|---|---|
| **Application Development and Languages** | | | | | |
| Ada: Tandem's Newest Compiler and Programming Environment | R. Vnuk | TSR | 3,2 | Aug. 1987 | 83940 |
| A New Design for the PATHWAY TCP* | R. Wong | TJ | 2,2 | Spring 1984 | 83932 |
| An Introduction to Tandem EXTENDED BASIC* | J. Meyerson | TJ | 2,2 | Spring 1984 | 83932 |
| Debugging TACL Code** | L. Palmer | TSR | 4,2 | July 1988 | 13693 |
| New TAL Features | C. Lu, J. Murayama | TSR | 2,2 | June 1986 | 83837 |
| PATHFINDER—An Aid for Application Development* | S. Benett | TJ | 1,1 | Fall 1983 | 83930 |
| PATHWAY IDS: A Message-level Interface to Devices and Processes* | M. Anderton, M. Noonan | TSR | 2,2 | June 1986 | 83937 |
| State-of-the-Art C Compiler* | E. Kit | TSR | 2,2 | June 1986 | 83937 |
| 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 |
| The ENABLE Program Generator for Multifile Applications* | B. Chapman, J. Zimmerman | TSR | 1,1 | Feb. 1985 | 83934 |
| TMF and the Multi-Threaded Requester* | T. Lemberger | TJ | 1,1 | Fall 1983 | 83930 |
| Writing a Command Interpreter* | D. Wong | TSR | 1,2 | June 1985 | 83935 |

[1]Articles and issues indicated by an asterisk (*) are no longer in stock.
Articles and issues indicated by a double asterisk (**) are available in limited quantities.

| Article title | Author(s) | Publication | Volume, Issue | Season or month and year | Part number |
|---|---|---|---|---|---|
| **Customer Support** | | | | | |
| Customer Information Service | J. Massucco | TSR | 3,1 | March 1987 | 83939 |
| Remote Support Strategy | J. Eddy | TSR | 3,1 | March 1987 | 83939 |
| Tandem's Software Support Plan | R. Baker, D. McEvoy | TSR | 3,1 | March 1987 | 83939 |
| **Data Communications** | | | | | |
| An Overview of SNAX/CDF | M. Turner | TSR | 5,2 | Sept. 1989 | 28152 |
| A SNAX Passthrough Tutorial* | D. Kirk | TJ | 2,2 | Spring 1984 | 83932 |
| Changes in FOX* | N. Donde | TSR | 1,2 | June 1985 | 83935 |
| Introduction to MULTILAN** | A. Coyle | TSR | 4,1 | Feb. 1988 | 11078 |
| Overview of the MULTILAN Server** | A. Rowe | TSR | 4,1 | Feb. 1988 | 11078 |
| SNAX/APC: Tandem's New SNA Software for Distributed Processing | B. Grantham | TSR | 3,1 | March 1987 | 83939 |
| SNAX/HLS: An Overview* | S. Saltwick | TSR | 1,2 | June 1985 | 83935 |
| Using the MULTILAN Application Interfaces** | M. Berg, A. Rowe | TSR | 4,1 | Feb. 1988 | 11078 |
| **Data Management** | | | | | |
| A Comparison of the B00 DP1 and DP2 Disc Processes* | T. Schachter | TSR | 1,2 | June 1985 | 83935 |
| Concurrency Control Aspects of Transaction Design | W. Senf | TSR | 6,1 | March 1990 | 32968 |
| Converting Database Files from ENSCRIBE to NonStop SQL | W. Weikel | TSR | 6,1 | March 1990 | 32986 |
| DP1-DP2 File Conversion: An Overview** | J. Tate | TSR | 2,1 | Feb. 1986 | 83936 |
| Determining FCP Conversion Time** | J. Tate | TSR | 2,1 | Feb. 1986 | 83936 |
| DP2's Efficient Use of Cache* | T. Schachter | TSR | 1,2 | June 1985 | 83935 |
| DP2 Highlights* | K. Carlyle, L. McGowan | TSR | 1,2 | June 1985 | 83935 |
| DP2 Key-sequenced Files* | T. Schachter | TSR | 1,2 | June 1985 | 83935 |
| High-Performance SQL Through Low-Level System Integration** | A. Borr | TSR | 4,2 | July 1988 | 13693 |
| Improvements in TMF* | T. Lemberger | TSR | 1,2 | June 1985 | 83935 |
| Optimizing Batch Performance | T. Keefauver | TSR | 5,2 | Sept. 1989 | 28152 |
| Overview of NonStop SQL** | H. Cohen | TSR | 4,2 | July 1988 | 13693 |
| NetBatch: Managing Batch Processing on Tandem Systems | D. Wakashige | TSR | 5,1 | April 1989 | 18662 |
| NetBatch-Plus: Structuring the Batch Environment | G. Earle, D. Wakashige | TSR | 6,1 | March 1990 | 32986 |
| NonStop SQL: The Single Database Solution | J. Cassidy, T. Kocher | TSR | 5,2 | Sept.1989 | 28152 |
| NonStop SQL Data Dictionary** | R. Holbrook, D. Tsou | TSR | 4,2 | July 1988 | 13693 |
| NonStop SQL Optimizer: Basic Concepts** | M. Pong | TSR | 4,2 | July 1988 | 13693 |
| NonStop SQL Optimizer: Query Optimization and User Influence** | M. Pong | TSR | 4,2 | July 1988 | 13693 |
| NonStop SQL Reliability** | C. Fenner | TSR | 4,2 | July 1988 | 13693 |
| The Relational Data Base Management Solution* | G. Ow | TJ | 2,1 | Winter 1984 | 83931 |
| Tandem's NonStop SQL Benchmark** | Tandem Performance Group | TSR | 4,1 | Feb.1988 | 11078 |
| The TRANSFER Delivery System for Distributed Applications* | S.Van Pelt | TJ | 2,2 | Spring 1984 | 83932 |
| TMF Autorollback: A New Recovery Feature* | M. Pong | TSR | 1,1 | Feb. 1985 | 83934 |
| **Manuals/Courses** | | | | | |
| B00 Software Manuals* | S. Olds | TSR | 1,2 | June 1985 | 83935 |
| C00 Software Manuals** | E. Levi | TSR | 4,1 | Feb. 1988 | 11078 |
| New Software Courses* | M. Janow | TSR | 1,2 | June 1985 | 83935 |
| New Software Courses** | J. Limper | TSR | 4,1 | Feb. 1988 | 11078 |
| Subscription Policy for Software Manuals** | T. McSweeney | TSR | 2,1 | Feb. 1986 | 83936 |
| Tandem's New Products** | C. Robinson | TSR | 2,1 | Feb. 1986 | 83936 |
| Tandem's New Products* | C. Robinson | TSR | 2,2 | June 1986 | 83937 |

| Article title | Author(s) | Publication | Volume, Issue | Season or month and year | Part number |
|---|---|---|---|---|---|
| **Operating Systems** | | | | | |
| Highlights of the B00 Software Release* | K. Coughlin, R. Montevaldo | TSR | 1,2 | June 1985 | 83935 |
| Increased Code Space* | A. Jordan | TSR | 1,2 | June 1985 | 83935 |
| Managing System Time Under GUARDIAN 90** | E. Nellen | TSR | 2,1 | Feb. 1986 | 83936 |
| New GUARDIAN 90 Time-keeping Facilities* | E. Nellen | TSR | 1,2 | June 1985 | 83935 |
| New Process-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 |
| Overview of the C00 Release** | L. Marks | TSR | 4,1 | Feb. 1988 | 11078 |
| Robustness to Crash in a Distributed Data Base: A Nonshared-memory Approach* | A. Borr | TSR | 1,2 | June 1985 | 83935 |
| The GUARDIAN Message System and How to Design for It* | M. Chandra | TSR | 1,1 | Feb. 1985 | 83935 |
| The Tandem Global Update Protocol* | R. Carr | TSR | 1,2 | June 1985 | 83935 |
| **Performance and Capacity Planning** | | | | | |
| A Performance Retrospective | P. Oleinick, P. Shah | TSR | 2,3 | Dec. 1986 | 83938 |
| Buffering for Better Application Performance** | R. Mattran | TSR | 2,1 | Feb. 1986 | 83936 |
| Capacity Planning Concepts | R. Evans | TSR | 2,3 | Dec. 1986 | 83938 |
| C00 TMDS Performance** | J. Mead | TSR | 4,1 | Feb. 1988 | 11078 |
| 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 |
| Estimating Host Response Time in a Tandem System | H. Horwitz | TSR | 4,3 | Oct. 1988 | 15748 |
| FASTSORT: An External Sort Using Parallel Processing | J. Gray, M. Stewart, A. Tsukerman, S. Uren, B. Vaughan | TSR | 2,3 | Dec. 1986 | 83938 |
| Getting Optimum Performance from Tandem Tape Systems | A. Khatri | TSR | 2,3 | Dec. 1986 | 83938 |
| How to Set Up a Performance Data Base with MEASURE and ENFORM | M. King | TSR | 2,3 | Dec. 1986 | 83938 |
| Improved Performance for BACKUP2 and RESTORE2* | A. Khatri, M. McCline | TSR | 1,2 | June 1985 | 83935 |
| MEASURE: Tandem's New Performance Measurement Tool | D. Dennison | TSR | 2,3 | Dec. 1986 | 83938 |
| Message System Performance Enhancements | D. Kinkade | TSR | 2,3 | Dec. 1986 | 83938 |
| Message System Performance Tests | S. Uren | TSR | 2,3 | Dec. 1986 | 83938 |
| Network Design Considerations | J. Evjen | TSR | 5,2 | Sept. 1989 | 28152 |
| NonStop VLX Performance | J. Enright | TSR | 2,3 | Dec. 1986 | 83938 |
| Optimizing Sequential Processing on the Tandem System* | R. Welsh | TJ | 2,3 | Summer 1984 | 83933 |
| Performance Considerations for Application Processes | R. Glasstone | TSR | 2,3 | Dec. 1986 | 83938 |
| Performance Measurements of an ATM Network Application | N. Cabell, D. Mackie | TSR | 2,3 | Dec. 1986 | 83938 |
| Predicting Response Time in On-line Transaction Processing Systems* | A. Khatri | TSR | 2,2 | June 1986 | 83937 |
| The 6600 and TCC6820 Communications Controllers: A Performance Comparison | P. Beadles | TSR | 2,3 | Dec. 1986 | 83938 |
| The ENCORE Stress Test Generator for On-line Transaction Processing Applications* | S. Kosinski | TJ | 2,1 | Winter 1984 | 83931 |
| 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 |
| Sizing Cache for Applications that Use B-series DP1 and TMF* | P. Shah | TSR | 2,2 | June 1986 | 83937 |
| Sizing the Spooler Collector Data File | H. Norman | TSR | 4,1 | Feb. 1988 | 11978 |
| Tandem's 5200 Optical Storage Facility: Performance and Optimization Considerations | S. Coleman | TSR | 5,1 | April 1989 | 18662 |
| Tandem's Approach to Fault Tolerance** | B. Ball, W. Bartlett, S. Thompson | TSR | 4,1 | Feb. 1988 | 11078 |
| Understanding PATHWAY Statistics* | R. Wong | TJ | 2,2 | Spring 1984 | 83932 |

| Article title | Author(s) | Publication | Volume, Issue | Season or month and year | Part number |
|---|---|---|---|---|---|
| **Peripherals** | | | | | |
| 5120 Tape Subsystem Recording Technology | W. Phillips | TSR | 3,2 | Aug. 1987 | 83940 |
| An Introduction to DYNAMITE Workstation Host Integration* | S. Kosinski | TSR | 1,2 | June 1985 | 83935 |
| Data-Encoding Technology Used in the XL8 Storage Facility* | D.S. Ng | TSR | 2,2 | June 1986 | 83937 |
| Data-Window Phase-Margin Analysis* | A. Painter, H. Pham, H. Thomas | TSR | 2,2 | June 1986 | 83937 |
| Introducing the 3207 Tape Controller* | S. Chandran | TSR | 1,2 | June 1985 | 83935 |
| Peripheral Device Interfaces | J. Blakkan | TSR | 3,2 | Aug. 1987 | 83940 |
| Plated Media Technology Used in the XL8 Storage Facility* | D.S. Ng | TSR | 2,2 | June 1986 | 83937 |
| Streaming Tape Drives | J. Blakkan | TSR | 3,2 | Aug. 1987 | 83940 |
| The 5200 Optical Storage Facility: A Hardware Perspective | A. Patel | TSR | 5,1 | April 1989 | 18662 |
| The 6100 Communications Subsystem: A New Architecture* | R. Smith | TJ | 2,1 | Winter 1984 | 83931 |
| The 6600 and TCC6820 Communications Controllers: A Performance Comparison | P. Beadles | TSR | 2,3 | Dec. 1986 | 83938 |
| The DYNAMITE Workstation: An Overview* | G. Smith | TSR | 1,2 | June 1985 | 83935 |
| The Model 6VI Voice Input Option: Its Design and Implementation* | B. Huggett | TJ | 2,3 | Summer 1984 | 83933 |
| The Role of Optical Storage in Information Processing | L. Sabaroff | TSR | 3,2 | Aug. 1987 | 83940 |
| The V8 Disc Storage Facility: Setting a New Standard for On-line Disc Storage* | M. Whiteman | TSR | 1,2 | June 1985 | 83935 |
| **Processors** | | | | | |
| NonStop CLX: Optimized for Distributed On-Line Transaction Processing | D. Lenoski | TSR | 5,1 | April 1989 | 18662 |
| NonStop VLX Hardware Design | M. Brown | TSR | 2,3 | Dec. 1986 | 83938 |
| 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 |
| The VLX: A Design for Serviceability | J. Allen, R. Boyle | TSR | 3,1 | March 1987 | 83939 |
| **Security** | | | | | |
| Distributed Protection with SAFEGUARD* | T. Chou | TSR | 2,2 | June 1986 | 83937 |
| **System Connectivity** | | | | | |
| Building Open Systems Interconnection with OSI/AS and OSI/TS | R. Smith | TSR | 6,1 | March 1990 | 32986 |
| Network Design Considerations | J. Evjen | TSR | 5,2 | Sept. 1989 | 28152 |
| Terminal Connection Alternatives for Tandem Systems | J. Simonds | TSR | 5,1 | April 1989 | 18662 |
| The OSI Model: Overview, Status, and Current Issues | A. Dunn | TSR | 5,1 | April 1989 | 18662 |
| **System Management** | | | | | |
| Configuring Tandem Disk Subsystems | S. Sitler | TSR | 2,3 | Dec. 1986 | 83938 |
| Data Replication in Tandem's Distributed Name Service | T. Eastep | TSR | 4,3 | Oct. 1988 | 15748 |
| Enhancements to TMDS | L. White | TSR | 3,2 | Aug. 1987 | 83940 |
| Event Management Service Design and Implementation | H. Jordan, R. McKee, R. Schuet | TSR | 4,3 | Oct. 1988 | 15748 |
| Introducing TMDS, Tandem's New On-line Diagnostic System* | J. Troisi | TSR | 1,2 | June 1985 | 83935 |
| Overview of DSM | P. Homan, B. Malizia, E. Reisner | TSR | 4,3 | Oct. 1988 | 15748 |
| Network Statistics System | M. Miller | TSR | 4,3 | Oct. 1988 | 15748 |
| SCP and SCF: A General Purpose Implementation of the Subsystem Programmatic Interface | T. Lawson | TSR | 4,3 | Oct. 1988 | 15748 |
| Tandem's Subsystem Programmatic Interface | G. Tom | TSR | 4,3 | Oct. 1988 | 15748 |
| Using FOX to Move a Fault-tolerant Application* | C. Breighner | TSR | 1,1 | Feb. 1985 | 83934 |
| Using the Subsystem Programmatic Interface and Event Management Services | K. Stobie | TSR | 4,3 | Oct. 1988 | 15748 |
| VIEWPOINT Operations Console Facility | R. Hansen, G. Stewart | TSR | 4,3 | Oct. 1988 | 15748 |
| VIEWSYS: An On-line System-resource Monitor* | D. Montgomery | TSR | 1,2 | June 1985 | 83935 |
| **Utilities** | | | | | |
| Enhancements to PS MAIL | R. Funk | TSR | 3,1 | March 1987 | 83939 |

# Index by Product

| Article title | Author(s) | Publication | Volume, Issue | Season or month and year | Part number |
|---|---|---|---|---|---|
| **3207 Tape Controller** | | | | | |
| Introducing the 3207 Tape Controller* | S. Chandran | TSR | 1,2 | June 1985 | 83935 |
| **5120 Tape Subsystem** | | | | | |
| 5120 Tape Subsystem Recording Technology | W. Phillips | TSR | 3,2 | Aug. 1987 | 83940 |
| **5200 Optical Storage** | | | | | |
| Tandem's 5200 Optical Storage Facility: Performance and Optimization Considerations | S. Coleman | TSR | 5,1 | April 1989 | 18662 |
| The 5200 Optical Storage Facility: A Hardware Perspective | A. Patel | TSR | 5,1 | April 1989 | 18662 |
| The Role of Optical Storage in Information Processing** | L. Sabaroff | TSR | 4,1 | Feb. 1988 | 11078 |
| **6100 Communications Subsystem** | | | | | |
| The 6100 Communications Subsystem: A New Architecture* | R. Smith | TJ | 2,1 | Winter 1984 | 83931 |
| **6530 Terminal** | | | | | |
| The Model 6VI Voice Input Option: Its Design and Implementation* | B. Huggett | TJ | 2,3 | Summer 1984 | 83933 |
| **6600 and TCC6820 Communications Controllers** | | | | | |
| The 6600 and TCC6820 Communications Controllers: A Performance Comparison | P. Beadles | TSR | 2,3 | Dec. 1986 | 83938 |
| **Ada** | | | | | |
| Ada: Tandem's Newest Compiler and Programming Environment | R. Vnuk | TSR | 3,2 | Aug. 1987 | 83940 |
| **BASIC** | | | | | |
| An Introduction to Tandem EXTENDED BASIC* | J. Meyerson | TJ | 2,2 | Spring 1984 | 83932 |
| **C** | | | | | |
| State-of-the-art C Compiler* | E. Kit | TSR | 2,2 | June 1986 | 83937 |
| **CIS** | | | | | |
| Customer Information Service | J. Massucco | TSR | 3,1 | March 1987 | 83939 |
| **CLX** | | | | | |
| NonStop CLX: Optimized for Distributed On-Line Transaction Processing | D. Lenoski | TSR | 5,1 | April 1989 | 18662 |
| **COBOL85** | | | | | |
| Tandem's New COBOL85** | D. Nelson | TSR | 2,1 | Feb. 1986 | 83936 |
| **COMINT (CI)** | | | | | |
| Writing a Command Interpreter* | D. Wong | TSR | 1,2 | June 1985 | 83935 |
| **DP1 and DP2** | | | | | |
| A Comparison of the B00 DP1 and DP2 Disc Processes* | T. Schachter | TSR | 1,2 | June 1985 | 83935 |
| Determining FCP Conversion Time** | J. Tate | TSR | 2,1 | Feb. 1986 | 83936 |
| 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 |
| Sizing Cache for Applications that Use B-series DP1 and TMF* | P. Shah | TSR | 2,2 | June 1986 | 83937 |
| **DSM** | | | | | |
| Data Replication in Tandem's Distributed Name Service | T. Eastep | TSR | 4,3 | Oct. 1988 | 15748 |
| Event Management Service Design and Implementation | H. Jordan, R. McKee, R. Schuet | TSR | 4,3 | Oct. 1988 | 15748 |
| Overview of DSM | P. Homan, B. Malizia, E. Reisner | TSR | 4,3 | Oct. 1988 | 15748 |
| Network Statistics System | M. Miller | TSR | 4,3 | Oct. 1988 | 15748 |
| SCP and SCF: A General Purpose Implementation of the Subsystem Programmatic Interface | T. Lawson | TSR | 4,3 | Oct. 1988 | 15748 |
| Tandem's Subsystem Programmatic Interface | G. Tom | TSR | 4,3 | Oct. 1988 | 15748 |
| Using the Subsystem Programmatic Interface and Event Management Services | K. Stobie | TSR | 4,3 | Oct. 1988 | 15748 |
| VIEWPOINT Operations Console Facility | R. Hansen, G. Stewart | TSR | 4,3 | Oct. 1988 | 15748 |

| Article title | Author(s) | Publication | Volume, Issue | Season or month and year | Part number |
|---|---|---|---|---|---|
| **DYNAMITE** | | | | | |
| An Introduction to DYNAMITE Workstation Host Integration* | S. Kosinski | TSR | 1,2 | June 1985 | 83935 |
| The DYNAMITE Workstation: An Overview* | G. Smith | TSR | 1,2 | June 1985 | 83935 |
| **ENABLE** | | | | | |
| The ENABLE Program Generator for Multifile Applications* | B. Chapman, J. Zimmerman | TSR | 1,1 | Feb. 1985 | 83934 |
| **ENCOMPASS** | | | | | |
| The Relational Data Base Management Solution* | G. Ow | TJ | 2,1 | Winter 1984 | 83931 |
| **ENCORE** | | | | | |
| The ENCORE Stress Test Generator for On-line Transaction Processing Applications* | S. Kosinski | TJ | 2,1 | Winter 1984 | 83931 |
| **ENSCRIBE** | | | | | |
| Converting Database Files from ENSCRIBE to NonStop SQL | W. Weikel | TSR | 6,1 | March 1990 | 32986 |
| **FASTSORT** | | | | | |
| FASTSORT: An External Sort Using Parallel Processing | J. Gray, M. Stewart, A. Tsukerman, S. Uren, B. Vaughan | TSR | 2,3 | Dec. 1986 | 83938 |
| **FOX** | | | | | |
| Changes in FOX* | N. Donde | TSR | 1,2 | June 1985 | 83935 |
| Using FOX to Move a Fault-tolerant Application* | C. Breighner | TSR | 1,1 | Feb. 1985 | 83934 |
| **GUARDIAN 90** | | | | | |
| B00 Software Manuals* | S. Olds | TSR | 1,2 | June 1985 | 83935 |
| C00 Software Manuals** | E. Levi | TSR | 4,1 | Feb. 1988 | 11078 |
| Highlights of the B00 Software Release* | K. Coughlin, R. Montevaldo | TSR | 1,2 | June 1985 | 83935 |
| 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 |
| Managing System Time Under GUARDIAN 90** | E. Nellen | TSR | 2,1 | Feb. 1986 | 83936 |
| Message System Performance Enhancements | D. Kinkade | TSR | 2,3 | Dec. 1986 | 83938 |
| Message System Performance Tests | S. Uren | TSR | 2,3 | Dec. 1986 | 83938 |
| New GUARDIAN 90 Time-keeping Facilities* | E. Nellen | TSR | 1,2 | June 1985 | 83935 |
| New Process-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 |
| Overview of the C00 Release** | L. Marks | TSR | 4,1 | Feb. 1988 | 11078 |
| Robustness to Crash in a Distributed Data Base: A Nonshared-memory Multiprocessor Approach* | A. Borr | TSR | 1,2 | June 1985 | 83935 |
| Tandem's Approach to Fault Tolerance** | B. Ball, W. Bartlett, S. Thompson | TSR | 4,1 | Feb. 1988 | 11078 |
| The GUARDIAN Message System and How to Design for It* | M. Chandra | TSR | 1,1 | Feb. 1985 | 83935 |
| The Tandem Global Update Protocol* | R. Carr | TSR | 1,2 | June 1985 | 83935 |
| **MEASURE** | | | | | |
| How to Set Up a Performance Data Base with MEASURE and ENFORM | M. King | TSR | 2,3 | Dec. 1986 | 83938 |
| MEASURE: Tandem's New Performance Measurement Tool | D. Dennison | TSR | 2,3 | Dec. 1986 | 83938 |
| **MULTILAN** | | | | | |
| Introduction to MULTILAN** | A. Coyle | TSR | 4,1 | Feb. 1988 | 11078 |
| Overview of the MULTILAN Server** | A. Rowe | TSR | 4,1 | Feb. 1988 | 11078 |
| Using the MULTILAN Application Interfaces** | M. Berg, A. Rowe | TSR | 4,1 | Feb. 1988 | 11078 |
| **NetBatch-Plus** | | | | | |
| NetBatch: Managing Batch Processing on Tandem Systems | D. Wakashige | TSR | 5,1 | April 1989 | 18662 |
| NetBatch-Plus: Structuring the Batch Environment | G. Earle, D. Wakashige | TSR | 6,1 | March 1990 | 32986 |

| Article title | Author(s) | Publication | Volume, Issue | Season or month and year | Part number |
|---|---|---|---|---|---|
| **NonStop SQL** | | | | | |
| Concurrency Control Aspects of Transaction Design | W. Senf | TSR | 6,1 | March 1990 | 32986 |
| Converting Database Files from ENSCRIBE to NonStop SQL | W. Weikel | TSR | 6,1 | March 1990 | 32986 |
| High-Performance SQL Through Low-Level System Integration** | A. Borr | TSR | 4,2 | July 1988 | 13693 |
| NonStop SQL Data Dictionary** | R. Holbrook, D. Tsou | TSR | 4,2 | July 1988 | 13693 |
| NonStop SQL: The Single Database Solution | J. Cassidy, T. Kocher | TSR | 5,2 | Sept. 1989 | 28152 |
| NonStop SQL Optimizer: Basic Concepts** | M. Pong | TSR | 4,2 | July 1988 | 13693 |
| NonStop SQL Optimizer: Query Optimization and User Influence** | M. Pong | TSR | 4,2 | July 1988 | 13693 |
| NonStop SQL Reliability** | C. Fenner | TSR | 4,2 | July 1988 | 13693 |
| Overview of NonStop SQL** | H. Cohen | TSR | 4,2 | July 1988 | 13693 |
| Tandem's Nonstop SQL Benchmark** | Tandem Performance Group | TSR | 4,1 | Feb. 1988 | 11078 |
| **OSI** | | | | | |
| Building Open Systems Interconnection with OSI/AS and OSI/TS | R. Smith | TSR | 6,1 | March 1990 | 32986 |
| The OSI Model: Overview, Status, and Current Issues | A. Dunn | TSR | 5,1 | April 1989 | 18662 |
| **PATHFINDER** | | | | | |
| PATHFINDER—An Aid for Application Development* | S. Benett | TJ | 1,1 | Fall 1983 | 83930 |
| **PATHWAY** | | | | | |
| A New Design for the PATHWAY TCP* | R. Wong | TJ | 2,2 | Spring 1984 | 83932 |
| PATHWAY IDS: A Message-level Interface to Devices and Processes* | M. Anderton, M. Noonan | TSR | 2,2 | June 1986 | 83937 |
| The PATHWAY TCP: Performance and Tuning* | J. Vatz | TSR | 1,1 | Feb. 1985 | 83934 |
| Understanding PATHWAY Statistics* | R. Wong | TJ | 2,2 | Spring 1984 | 83932 |
| **PS MAIL** | | | | | |
| Enhancements to PS MAIL | R. Funk | TSR | 3,1 | March 1987 | 83939 |
| **SAFEGUARD** | | | | | |
| Distributed Protection with SAFEGUARD* | T. Chou | TSR | 2,2 | June 1986 | 83937 |
| **SNAX** | | | | | |
| An Overview of SNAX/CDF | M. Turner | TSR | 5,2 | Sept. 1989 | 28152 |
| A SNAX Passthrough Tutorial* | D. Kirk | TJ | 2,2 | Spring 1984 | 83932 |
| SNAX/APC: Tandem's New SNA Software for Distributed Processing | B. Grantham | TSR | 3,1 | March 1987 | 83939 |
| SNAX/HLS: An Overview* | S. Saltwick | TSR | 1,2 | June 1985 | 83935 |
| **SPOOLER** | | | | | |
| Sizing the Spooler Collector Data File** | H. Norman | TSR | 4,1 | Feb. 1988 | 11078 |
| **TACL** | | | | | |
| Debugging TACL Code** | L. Palmer | TSR | 4,2 | July 1988 | 13693 |
| TACL, Tandem's New Extensible Command Language** | J. Campbell, R. Glascock | TSR | 2,1 | Feb. 1986 | 83936 |
| **TAL** | | | | | |
| New TAL Features | C. Lu, J. Murayama | TSR | 2,2 | June 1986 | 83837 |
| **TMDS** | | | | | |
| C00 TMDS Performance** | J. Mead | TSR | 4,1 | Feb. 1988 | 11078 |
| Enhancements to TMDS | L. White | TSR | 3,2 | Aug. 1987 | 83940 |
| Introducing TMDS, Tandem's New On-line Diagnostic System* | J. Troisi | TSR | 1,2 | June 1985 | 83935 |
| **TMF** | | | | | |
| Improvements in TMF* | T. Lemberger | TSR | 1,2 | June 1985 | 83935 |
| 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 |
| **TRANSFER** | | | | | |
| The TRANSFER Delivery System for Distributed Applications* | S. Van Pelt | TJ | 2,2 | Spring 1984 | 83932 |
| **TXP** | | | | | |
| 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 |

| Article title | Author(s) | Publication | Volume, Issue | Season or month and year | Part number |
|---|---|---|---|---|---|
| **V8** | | | | | |
| The V8 Disc Storage Facility: Setting a New Standard for On-line Disc Storage* | M. Whiteman | TSR | 1,2 | June 1985 | 83935 |
| **VIEWSYS** | | | | | |
| VIEWSYS: An On-line System-resource Monitor* | D. Montgomery | TSR | 1,2 | June 1985 | 83935 |
| **VLX** | | | | | |
| NonStop VLX Hardware Design | M. Brown | TSR | 2,3 | Dec. 1986 | 83938 |
| NonStop VLX Performance | J. Enright | TSR | 2,3 | Dec. 1986 | 83938 |
| The VLX: A Design for Serviceability | J. Allen, R. Boyle | TSR | 3,1 | March 1987 | 83939 |
| **XL8** | | | | | |
| Data-encoding Technology Used in the XL8 Storage Facility* | D.S. Ng | TSR | 2,2 | June 1986 | 83937 |
| Plated Media Technology Used in the XL8 Storage Facility* | D.S. Ng | TSR | 2,2 | June 1986 | 83937 |
| **Miscellaneous**[2] | | | | | |
| A Performance Retrospective | P. Oleinick | TSR | 2,3 | Dec. 1986 | 83938 |
| Buffering for Better Application Performance** | R. Mattran | TSR | 2,1 | Feb. 1986 | 83936 |
| Capacity Planning Concepts | R. Evans | TSR | 2,3 | Dec. 1986 | 83938 |
| Configuring Tandem Disk Subsystems | S. Sitler | TSR | 2,3 | Dec. 1986 | 83938 |
| Credit-authorization Benchmark for High Performance and Linear Growth** | T. Chmiel, T. Houy | TSR | 2,1 | Feb. 1986 | 83936 |
| Data-window Phase-margin Analysis* | A. Painter, H. Pham, H. Thomas | TSR | 2,2 | June 1986 | 83937 |
| Estimating Host Response Time in a Tandem System | H. Horwitz | TSR | 4,3 | Oct.1988 | 15748 |
| Getting Optimum Performance from Tandem Tape Systems | A. Khatri | TSR | 2,3 | Dec. 1986 | 83938 |
| Network Design Considerations | J. Evjen | TSR | 5,2 | Sept. 1989 | 28152 |
| New Software Courses* | M. Janow | TSR | 1,2 | June 1985 | 83935 |
| New Software Courses** | J. Limper | TSR | 4,1 | Feb. 1988 | 11078 |
| Optimizing Batch Performance | T. Keefauver | TSR | 5,2 | Sept. 1989 | 28152 |
| Optimizing Sequential Processing on the Tandem System* | R. Welsh | TJ | 2,3 | Summer 1984 | 83933 |
| Performance Considerations for Application Processes | R. Glasstone | TSR | 2,3 | Dec. 1986 | 83938 |
| Performance Measurements of an ATM Network Application | N. Cabell, D. Mackie | TSR | 2,3 | Dec. 1986 | 83938 |
| Peripheral Device Interfaces | J. Blakkan | TSR | 3,2 | Aug. 1987 | 83940 |
| Predicting Response Time in On-line Transaction Processing Systems* | A. Khatri | TSR | 2,2 | June 1986 | 83937 |
| Remote Support Strategy | J. Eddy | TSR | 3,1 | March 1987 | 83939 |
| Streaming Tape Drives | J. Blakkan | TSR | 3,2 | Aug. 1987 | 83940 |
| Subscription Policy for Software Manuals** | T. McSweeney | TSR | 2,1 | Feb. 1986 | 83936 |
| Tandem's New Products** | C. Robinson | TSR | 2,1 | Feb. 1986 | 83936 |
| Tandem's New Products* | C. Robinson | TSR | 2,2 | June 1986 | 83937 |
| Tandem's Software Support Plan | R. Baker, D. McEvoy | TSR | 3,1 | March 1987 | 83939 |
| Terminal Connection Alternatives for Tandem Systems | J. Simonds | TSR | 5,1 | April 1989 | 18662 |
| The Performance Characteristics of Tandem NonStop Systems* | J. Day | TJ | 1,1 | Fall 1983 | 83930 |
| The Role of Optical Storage in Information Processing | L. Sabaroff | TSR | 3,2 | Aug. 1987 | 83940 |

[2]This category contains *Tandem Systems Review* articles that contain product information, but are not specifically product-related.

# TANDEM SYSTEMS REVIEW CUSTOMER SURVEY

The purpose of this questionnaire is to help the *Tandem Systems Review* staff select topics for publication. Postage is prepaid when mailed in the U.S. Customers outside the U.S. should send their replies to their nearest Tandem sales office.

1. How useful is each article in this issue?

*Building Open Systems Interconnection with OSI/AS and OSI/TS*

01 ☐ Indispensible 02 ☐ Very 03 ☐ Somewhat 04 ☐ Not at all

*NetBatch-Plus: Structuring the Batch Environment*

05 ☐ Indispensible 06 ☐ Very 07 ☐ Somewhat 08 ☐ Not at all

*Converting Database Files from ENSCRIBE to NonStop SQL*

09 ☐ Indispensible 10 ☐ Very 11 ☐ Somewhat 12 ☐ Not at all

*Concurrency Control Aspects of Transaction Design*

13 ☐ Indispensible 14 ☐ Very 15 ☐ Somewhat 16 ☐ Not at all

2. I specifically would like to see more articles on (select one):

17 ☐ Overview discussions of new products and enhancements.
18 ☐ Performance and tuning information.
19 ☐ High-level overviews on Tandem's approach to solutions.
20 ☐ Application design and customer profiles.
21 ☐ Technical discussions of product internals.
22 ☐ Other_____

3. Your title or position:

23 ☐ President, VP, Director 24 ☐ Systems analyst 25 ☐ System operator
26 ☐ MIS manager 27 ☐ Software developer 28 ☐ End user
29 ☐ Other_____

4. Your association with Tandem:

30 ☐ Tandem customer 31 ☐ Tandem employee 32 ☐ Third-party vendor 33 ☐ Consultant
34 ☐ Other_____

5. Comments

_____

_____

_____

_____

_____

NAME

_____

COMPANY NAME

_____

ADDRESS

_____

_____

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 482 CUPERTINO, CA. U.S.A.

POSTAGE WILL BE PAID BY ADDRESSEE

**TANDEM COMPUTERS INCORPORATED**
TANDEM SYSTEMS REVIEW
LOC 216-05
19333 VALLCO PARKWAY
CUPERTINO CA 95014-9990

FOLD

# TANDEM SYSTEMS REVIEW ORDER FORM

Use this form to subscribe, change a subscription, and order back copies. Tandem customers must complete the first portion of this form and submit it to their local Tandem representative for approval. The Tandem representative will complete the second part of this form and submit it for processing.

***To be completed by the Tandem customer.***

Subscription information:

COMPANY

NAME

JOB TITLE

DIVISION

ADDRESS

COUNTRY

TELEPHONE NUMBER (include all codes for U.S. dialing)

Back order requests:

- ☐ Part No. 83938, Vol. 2, No. 3, December 1986
- ☐ Part No. 83939, Vol. 3, No. 1, March 1987
- ☐ Part No. 83940, Vol. 3, No. 2, August 1987
- ☐ Part No. 11078, Vol. 4, No. 1, February 1988
- ☐ Part No. 13693, Vol. 4, No. 2, July 1988
- ☐ Part No. 15748, Vol. 4, No. 3, October 1988
- ☐ Part No. 18622, Vol. 5, No. 1, April 1989
- ☐ Part No. 28152, Vol. 5, No. 2, September 1989

Please allow eight weeks for back orders.

- ☐ New subscription
- ☐ Address change
  Subscription number: _____
  (Your subscription number is in the upper right corner of the mailing label.)

***To be completed by the Tandem representative.***

Please complete this portion of the form to approve the above request.

NAME

DEPARTMENT NUMBER

TITLE

LOC

TELEPHONE NUMBER

CUSTOMER NUMBER

SYSTEM NUMBER

SIGNATURE

1. For subscription processing only, send form to:

   Tandem Computers Incorporated
   *Tandem Systems Review*
   LOC 216-05
   18922 Forge Drive
   Cupertino, CA 95014-0701

2. For requesting back orders for customers, use COURIER. The menu sequence is: Marketing Information, Literature Orders, Tandem Systems Review. The COURIER form allows the literature order to be sent directly to your customer's address.

   ☐ Date of COURIER order submittal: _____

Tandem Computers Incorporated
19333 Vallco Parkway
Cupertino, CA 95014-2599

MARC BRANDIFINO
DEPT 8449
LOC NUM 50-00
NEW YORK NY DISTRICT

Part No. 32986

400108 3/90 Printed in USA