TDS

Concepts

DPS7000/XTA
NOVASCALE 7000

Transaction Processing: General

REFERENCE
47 A2 26UT 00

DPS7000/XTA NOVASCALE 7000

TDS

Concepts

Transaction Processing: General

March 1993

B.P.20845

49008 ANGERS CEDEX 01

FRANCE

REFERENCE
47 A2 26UT 00

The following copyright notice protects this book under Copyright laws which prohibit such actions as, but not limited to, copying, distributing, modifying, and making derivative works.

Copyright © Bull SAS 1993

Printed in France

Suggestions and criticisms concerning the form, content, and presentation of this book are invited. A form is provided at the end of this book for this purpose.

To order additional copies of this book or other Bull Technical Publications, you are invited to use the Ordering Form also provided at the end of this book.

Trademarks and Acknowledgements

We acknowledge the right of proprietors of trademarks mentioned in this book.

Intel® and Itanium® are registered trademarks of Intel Corporation.

Windows® and Microsoft® software are registered trademarks of Microsoft Corporation.

UNIX® is a registered trademark in the United States of America and other countries licensed exclusively through the Open Group.

Linux® is a registered trademark of Linus Torvalds.

The information in this document is subject to change without notice. Bull will not be liable for errors contained herein, or for incidental or consequential damages in connection with the use of this material.

Preface

INTENDED AUDIENCE

This document is intended for anyone interested in learning about Transactional Driven Subsystems (TDS). This can be users, programmers, operators, or system administrators who work with or need to be familiar with TDS applications.

SCOPE AND OBJECTIVES

This document explains the basic concepts of TDS and its place within GCOS 7. More detailed information for developing, using and managing a TDS is found in these TDS documents:

TDS Administrator's Guide.....*47 A2 20UT*
TDS C Language Programmer's Manual.....*47 A2 07UT*
TDS COBOL Programmer's Manual.....*47 A2 21UT*
TDS Quick Reference Handbook.....*47 A2 24UT*
Transactional Intercommunication Using XCP1 Protocol User's Guide.....*47 A2 11UT*
CPI-C/XCP2 User's Guide.....*47 A2 14UT*
High Availability Concepts.....*47 A2 22UT*
High Availability Administrator's Guide.....*47 A2 23UT*

STRUCTURE OF THIS DOCUMENT

Chapter 1	presents an overview of TDS and transaction processing. This section explains how TDS is different from other types of processing, and how TDS interacts with GCOS 7 and other products. Also included is a summary of the types of users of a TDS application.
Chapter 2	defines the basic concepts of TDS and is relevant for all levels of users: TDS Administrators, Programmers, Master Operators, and End-users. This chapter includes explanations of how data and resources are handled, how data is protected and recovered, and how data is organized.
Section 3	explains how a TDS application is created and then run. The preparation and generation of the application is presented. This chapter includes descriptions of the roles of each type of user.
Section 4	describes how the TDS application is designed and optimized. This chapter presents the more complex concepts that must be considered by the TDS Administrator and the TDS Programmer.

A glossary of TDS terms is provided at the end of this document.

RELATED DOCUMENTS

The following documents provide detailed information about topics that affect TDS.

For generating the DPS 7 network:

Network Overview and Generation *47 A2 71UC*
Terminal Management *39 A2 24DN*

Installing, optimizing, and managing the system:

System Installation Configuration and Updating Guide *47 A2 17US*
System Administrator's Manual.....*47 A2 10US*
System Behavior Reporter User's Guide *47 A2 03US*
TILS User's Guide.....*47 A2 04US*

Creating and managing FORMS using the MAINTAIN_FORM utility:

IOF Programmer's Manual..... *47 A2 05UJ*

Generating the GTWriter Network:

Generalized Terminal Writer User's Guide *47 A2 05UU*

COBOL syntax and use:

COBOL 85 Reference Manual *47 A2 05UL*
COBOL 85 User's Guide.....*47 A2 06UL*

File access and data management:

IDS/II Administrator's Guide..... *47 A2 13UD*
UFAS-EXTENDED User's Guide.....*47 A2 04UF*
Data Security Facilities User's Guide.....*47 A2 09UF*

Main Operator tasks:

GCOS 7-V6 Network Operations Reference Manual..... *47 A2 72UC*
DOF 7-PO User's Guide *47 A2 80UC*
System Operator's Guide..... *47 A2 60UU*

Concurrency control:

General Access Control (GAC-EXTENDED) User's Guide *47 A2 12UF*

For GCL commands:

IOF Terminal User's Reference Manual

Part 1 Introduction to IOF 47 A2 31UJ
Part 2 GCL Commands 47 A2 32UJ
Part 3 Processors' Commands 47 A2 33UJ
Part 4 Appendices 47 A2 34UJ

Using IQS under TDS:

IQS-V4/TDS User's Guide 47 A2 81UR

For using ORACLE under TDS:

ORACLE-V6/TDS User's Guide 47 A2 05UR

For Affinity:

Installation & Configuration Guide 40 A2 07WA

For OTM:

Administrator's Guide 86 A2 24SK

For CTP:

CPI-C Programmer's Guide 86 A2 45SL

For IMAGEWorks:

TDS -IMAGEWorks Link User's Guide 47 A2 25UT

Table of Contents

Illustrations

Figures

1. Overview of TDS

This chapter explains what TDS is and includes:

- examples of TDS applications
- a description of how TDS is different from other modes of processing
- an explanation of the advantages of TDS
- an overview of how TDS interacts with GCOS and other products
- a summary of the typical users of a TDS application
- a diagram of the related documentation.

The Transaction Driven Subsystem, or TDS, is a GCOS 7 environment based on transactions or real-time processing. That is, a TDS application accepts a request from an end-user, processes the request, and then sends a response back to the user's terminal. This type of system is called transaction driven because processing is determined by the transaction specified by a user. (The term "**transaction**" as it is used here, indicates an interaction between the end-user and the data stored in the computer system.)

TDS applications allow many users to access files or databases simultaneously while maintaining the integrity of the data. Several users can even submit the same transaction at the same time. TDS applications provide short response times, reduce the time applications are unavailable after an incident, and ensure security and confidentiality of data.

The names of the commands given in this manual are these applicable to GCOS 7 V6.

1.1 WHAT IS A TDS APPLICATION?

A TDS application is an application designed to execute transactions. Bull provides the TDS software which both acts as a monitor of TDS applications and is used to generate the customer's TDS applications. The TDS Administrator and TDS Programmer create the customer's TDS applications using the Bull supplied TDS software. This is the referred to as creating a TDS.

TDS applications are those where many users need to access information simultaneously or process continuous business activities. There are many different kinds of customized TDS applications. Transactional applications are suitable for order processing, inventory control, manufacturing, scheduling, billing, retail banking, and accounting where many different operators or end-users use the same programs at the same time on different terminals.

Examples of TDS Applications:

For example, travel agents are linked via terminals to a computer. Each travel agent can access flight records stored in a central location and make reservations for particular flights.

Another example of a transaction is a bank customer's interaction with an automated teller machine. When the customer withdraws money from an automatic teller machine, several things happen during the processing of the transaction requested (withdrawal of money from an account): the customer's account is read to determine whether there is enough money to cover the withdrawal, the account is updated in the bank's database, the amount of money requested is delivered, and a receipt is printed.

1.1.1 How is TDS Different from Other Modes of Processing?

TDS, like IOF and batch processing, is an environment in GCOS 7. As with IOF or batch processing, several TDS applications can exist in the TDS environment. Figure 1-1 shows this relationship.

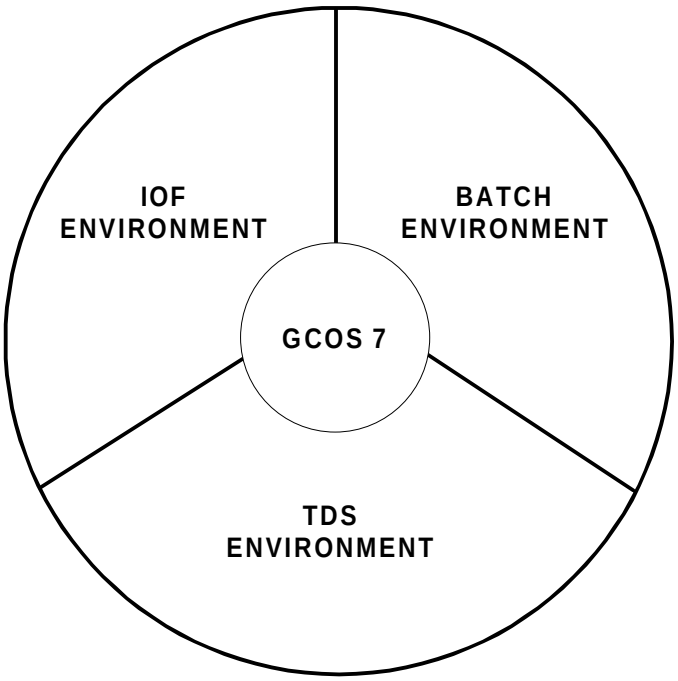


Figure 1-1. The TDS Environment in GCOS 7

TDS satisfies customers information needs rapidly. The amount of time it takes for the TDS application to respond to a request for service made by an "end-user" is the measure of its effectiveness. This interval is called "**response time**". When using a TDS application, a user can expect a response to a request immediately (1 or 2 seconds).

In batch or IOF processing, the data needed to update a master file is collected before the file is updated. The time needed to prepare data before final processing can be long. Furthermore, a master file can be used by only one program at a time. Other programs have to wait for the file to be released.

When using a TDS application, users can enter data for immediate processing and can access updated data any time. An airline reservation system is an example of this real-time processing. When a passenger purchases a ticket, the reservation can be made immediately, and the system updates information about available seats automatically. In addition, TDS applications are designed to respond to multiple information processing requests.

TDS is an inquiry/response type of application. Inquiry means that you can only retrieve information from files, but you cannot modify the contents of these files. Response means that a TDS application returns the appropriate information to a request.

The following scenario illustrates the usefulness of TDS processing:

If a user wants to check the status of the accounts for the business' customers and the company uses batch mode, a job stream must be set up, jobs must be scheduled, and results printed and distributed; a process which can take several hours. In the TDS environment, the user logs on to the system, enters a transaction and sees a menu of functions displayed on the screen. By selecting a function and entering the account codes, the information is available in seconds.

In a TDS application, the transaction is executed as it occurs. In a batch/IOF environment, jobs are scheduled over an interval of time and then executed.

Within GCOS 7, resources (the TDS environment) which contain information to be shared by TDS applications are available. The software that provides centralized control and orders the processing of transactions (requested by users at the terminals) is known as the TDS Monitor. Figure 1-2 shows the relationship of the TDS Monitor to TDS applications.

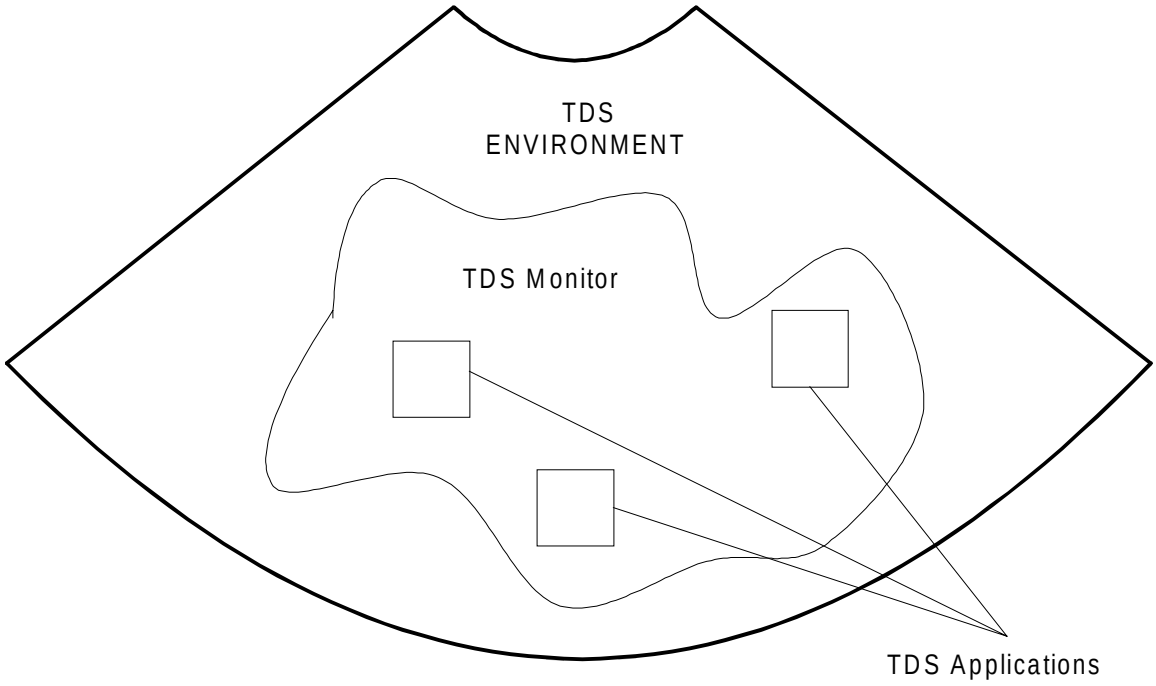


Figure 1-2. The TDS Monitor within the TDS Environment

1.1.2 What are the Advantages of TDS

Transaction Driven Subsystems provide applications with security, integrity, and flexibility using its own mechanisms, and standard GCOS 7 mechanisms.

In a TDS application, the confidentiality of data is assured by TDS authority codes, GCOS 7 access rights, and can be protected by SECUR'ACCESS (the Bull software security product). Users can be granted or refused the right to execute a given transaction.

TDS allows for simultaneous processing. A TDS can run up to 124 processes per application. The file integrity and coherence is maintained by TDS non-concurrency facilities, GAC-EXTENDED, and the protected read mechanisms. The TDS recovery mechanism and GCOS 7 journals provide the means of saving data and restarting after abort.

TDS offers flexible programming. Transaction Processing Routines (the basic units of transactions) can be written in COBOL, C language, or GPL, and can integrate IQS procedures (IQS is the Bull Relational Information System).

TDS recovery mechanism, along with GCOS 7 system journals ensure that application are **available**. Warm start recovery in case of an incident, restarts of interrupted transactions and dynamic recovery of errors are offered by TDS. Furthermore, using TDS with these GCOS 7 High Availability products improves protection of applications and reduces the time that the application cannot be used:

- **TDS-HA** improves the availability of applications (local system backup)
- **Mirror Disks** improves the availability of data
- **RDDF7** improves the availability of databases (remote system backup).

1.2 WHERE IS TDS SITUATED IN GCOS 7?

TDS and Databases:

Figure 1-3 shows TDS in relation to the ORACLE and IDS/II products.

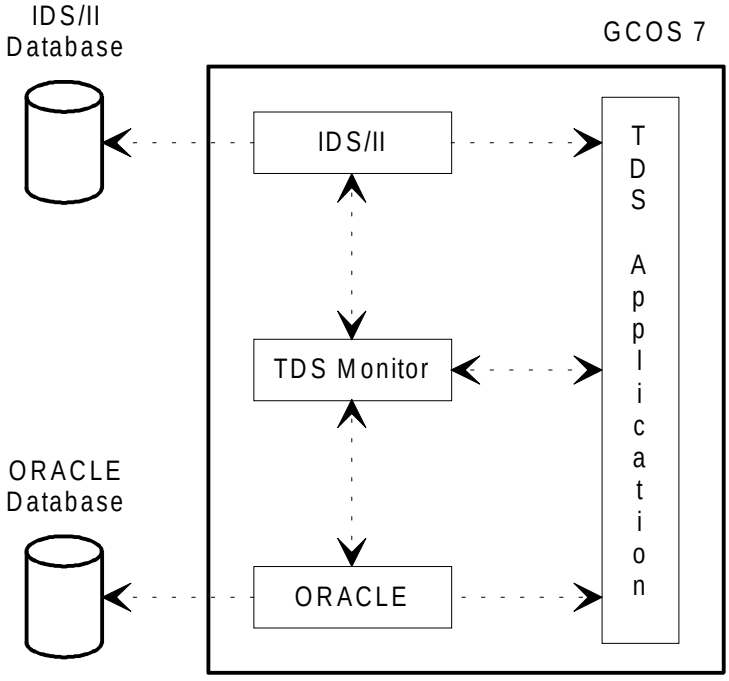


Figure 1-3. TDS, ORACLE, and IDS/II

TDS and FEPS:

Figure 1-4 shows TDS in relation to the FEPS (and VCAM).

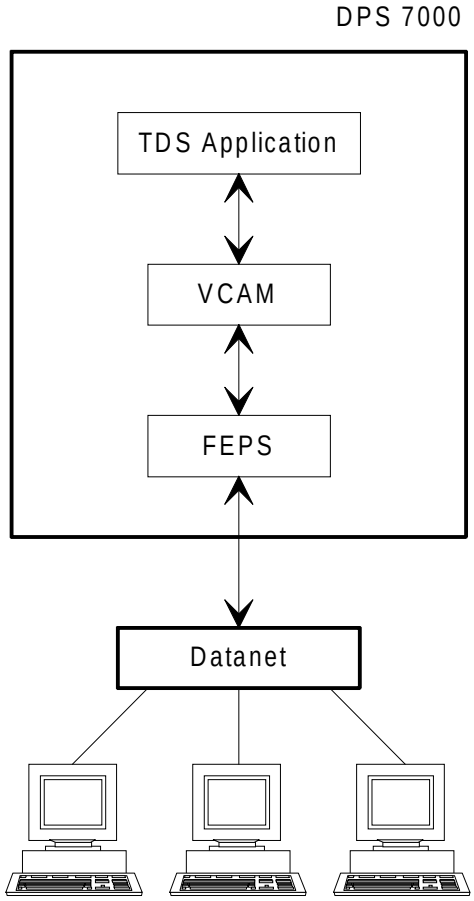


Figure 1-4. TDS and FEPS

TDS and CNP 7:

Figure 1-5 shows TDS in relation to CNP 7 (and VCAM).

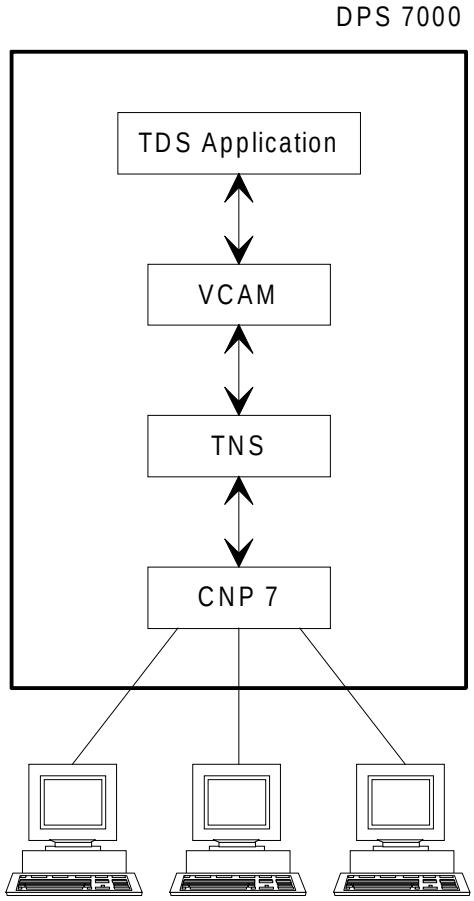


Figure 1-5. TDS and CNP 7

1.2.1 Who Uses TDS?

There are four types of users who work on TDS applications: TDS Administrator, TDS Programmer, master operator, and end-user.

- The TDS Administrator is responsible for determining the requirements of the application and customizing it to meet these needs. The system administrator prepares and generates the application, and then manages the sizing and optimizing of a running application.

- The TDS Programmer writes the transactions (or TPRs) of the application following the outline determined by the system administrator.

- The master operator is responsible for supervising a TDS session. The operator starts and terminates the application, opens and closes files, and restarts the application when an incident occurs.

- The end-user is the person at a terminal connected to a TDS application who runs transactions.

1.2.2 How to Use the TDS Documentation

There are several different documents which explain various aspects of TDS. The type of document you consult depends on what your role is in TDS. This list presents an overview of the documents that should be consulted by the four different type of users of TDS. The documents you consult depend on the specific products on your site. The documents in brackets are optional.

End-User	*TDS Concepts*
Main Operator	*TDS Concepts* *TDS Administrator's Guide (subset "Master Operator Commands")*
TDS Programmer	*TDS Concepts* *TDS COBOL* **or** *C Language Programmer's Guide* *Transactional Intercommunication Using XCP1 Protocol User's Guide* **or** *CPI-C/XCP2 User's Guide* *TDS Quick Reference Handbook* - *[IQS-V4/TDS User's Guide]* - *[ORACLE-V6/TDS User's Guide]*
TDS Administrator	*TDS Concepts* *TDS Quick Reference Handbook* *TDS Administrator's Guide* - *[High Availability Administrator's Guide]*

The TDS Programmer may also read the TDS Administrator's Guide or at least some chapters of it. Similarly, the TDS Administrator should be familiar with some of the contents of the manuals destined for the TDS Programmer.

For a more thorough list of related documents and the document numbers of those listed here, refer to the Preface of this book.

2. Basic Concepts and Terms

To understand TDS, there are several concepts that need to be defined. These concepts are explained in this chapter:

- Transactions and the elements that make up a transaction:

  TPRs
  Exchanges
  Conversations.

- The handling of data and resources by using:

  Commitment Units
  Commitment Points.

- Journalization and recovery techniques:

  Deferred Updates
  Rollbacks
  Rollforwards.

- The files in a TDS application:

  TDS-controlled
  Non-controlled files.

2.1 WHAT IS A TRANSACTION?

In TDS, a transaction is a sequence of activities designed to accomplish a user-defined task. There can be up to 2000 transactions in a TDS application. A transaction, or Tx, is named by a specific message identifier code, or message-id. This code is provided by the TDS Administrator or displayed in a menu.

A transaction is executed or processed when a user starts a transaction by issuing a specific message-id. The transaction is terminated when all of the task is performed. When the TDS Monitor receives a message-id from a terminal, it loads the first TPR of the transaction. A transaction can be made up of one or several TPRs. If all of the work of a transaction is not completed by the first TPR, other TPRs are started.

Only one message-id can be entered at one time, at a given terminal. When the transaction is successfully executed, the user can enter the next message-id.

2.1.1 Types of Transactions in TDS

There are three main types of transactions in TDS.

- The FOR INQUIRY transaction is a request for information. This transaction answers the user's question by "asking" the database about the current status of data. This transaction does not modify any files.

- The UPDATE transaction is a modification of files or databases. This transaction performs the user's request by adding or removing the requested data. This transaction does updates files.

- The FOR DEBUG can be considered a temporary transaction. This type of transaction is used by the programmer trying to correct a problem or optimize a TDS application.

In addition there are several types of special transactions available to the TDS programmer. For more information on these special transactions, see "TDS Programmer Duties" in the chapter "The Tasks Involved in Setting Up a TDS Application".

2.1.2 What is a Transaction Composed of?

A transaction is composed of:

- Exchanges
- TPRs
- Conversations.

Figure 2-1 shows the relationship of these elements.

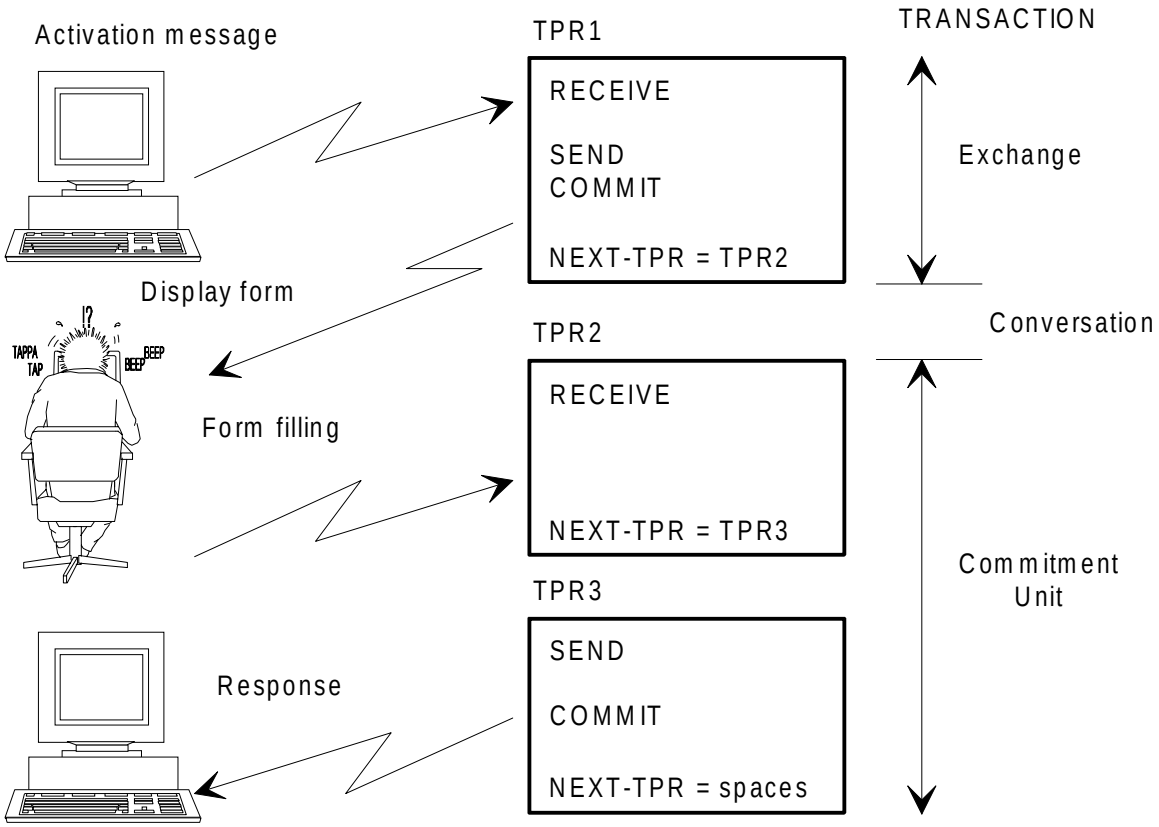


Figure 2-1. The Components of a Transaction

A transaction is a sequence of exchanges and conversations. A transaction consists of at least one **exchange**, each of which include one or more TPRs. However, a transaction may or may not include a conversation. The processing of a transaction is performed by TPRs during an exchange, and the operator reply takes place during the conversation.

2.1.2.1 What is an Exchange?

Each exchange is comprised of an input message, processing, and an output message. The processing that occurs during the exchange is performed by one or more **TPRs** (Transaction Processing Routines).

An exchange starts at the transmission of a message from the terminal operator and ends at the reply sent back to this operator. An exchange is the sequence of:

- a request keyed in by the user (INPUT)
- processing performed by the transaction
- response sent to the user's terminal (OUTPUT).

An exchange contains at least one TPR. However, one exchange can include several TPRs chained to one another. Figure 2-2 shows these possibilities.

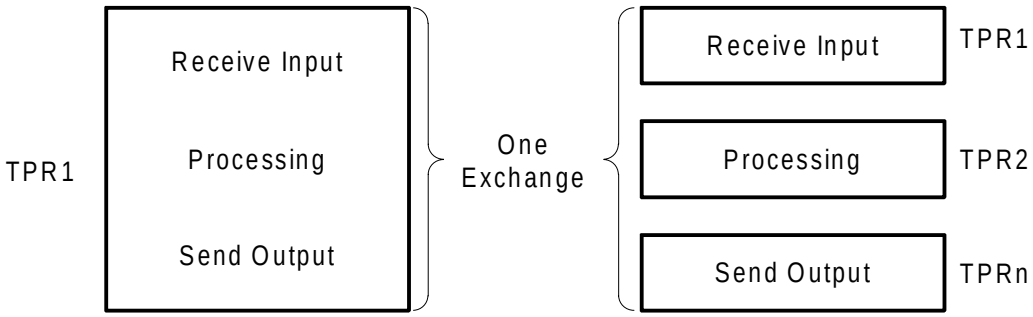


Figure 2-2. Two Types of Exchanges

When a transaction is started, the TDS Monitor loads the first TPR. The processing of the TPR, that is the input and output, is an exchange. If the transaction is interactive, that is, if an end-user must respond to an on-screen message, a conversation is started. When the reply from the terminal operator arrives, the next TPR is loaded and executed.

When an exchange is completed, control is returned to the terminal. The end of the exchange is signalled either by the response to operator input, or by the default reply, "READY" when the transaction ends without a write to the screen. After you enter information, you must wait for this response from the transaction before entering a new message.

2.1.2.2 What is a TPR?

A TPR is a Transaction Processing Routine written in either COBOL, C, or GPL programming language. TPRs are the programs that perform the individual steps of a transaction. A transaction consists of one or several related TPRs. TPRs are used to accept input, to process it, and then to issue the reply.

A TPR usually chains to another TPR. When a TPR does not chain to another TPR, it means that it is the last TPR of the transaction.

The TDS Programmer writes and compiles the TPRs, then stores them in sharable module libraries. TPRs are brought into main memory to be executed when required. Several transactions can share the same TPR, in any sequence (that is, order of execution) with other TPRs. In any TDS application, several different TPRs can run simultaneously.

A system can have up to 40,000 TPRs. Each TDS application can have as many as 25,200 TPRs.

2.1.2.3 What is a Conversation?

In TDS, a conversation is defined as the period of time between two exchanges of the same transaction. A conversation is the time that the user takes between receiving an output message and sending an input message. That is, the period starting when the response is sent by a transaction to the terminal, the time the user spends thinking (or thinktime), and when the next piece of information entered by the user is received.

Figure 2-3 illustrates a conversation.

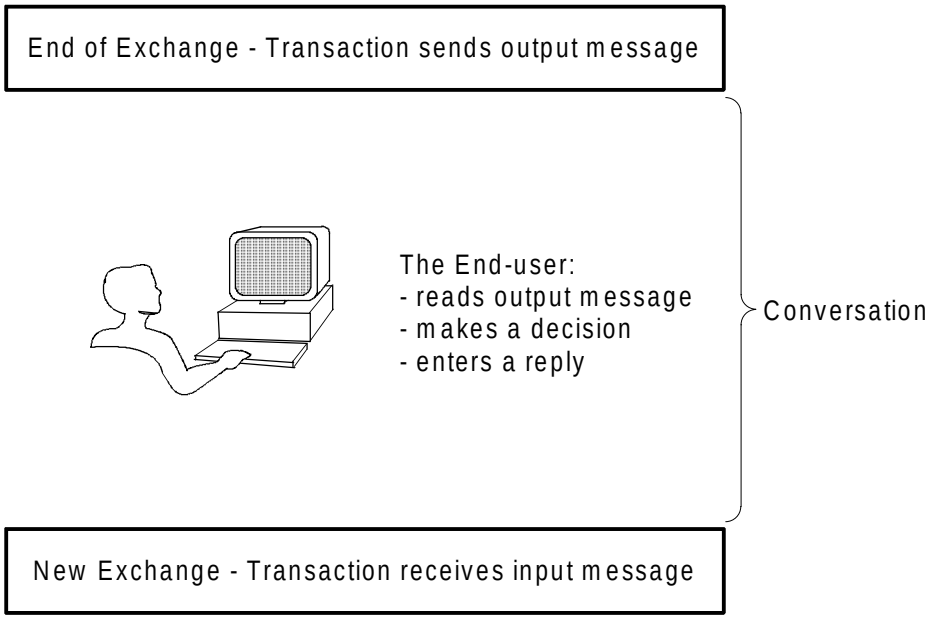


Figure 2-3. A Conversation

A conversation relates to an exchange as follows:

- A transaction does not necessarily include a conversation.
- A conversation includes the "thinktime" between the output message and the response. This period can be long, particularly if the user must do something, or if the transaction involves two people (for example a sales clerk and a customer).

A conversation requires more time than the processing of an exchange as it requires human intervention. Other functions can be performed by a TDS application while the user is reacting, a factor which is important for the TDS Programmer to consider when designing the TPRs.

2.2 HOW TDS HANDLES DATA AND RESOURCES

A transaction does not have unique access to the data that its TPRs access. If a transaction holds all of the data (or resources) that it accesses, other transactions cannot access the same data until the first transaction terminates.

At a given point in processing, a transaction must "save" the modifications made to the data. This save is called "taking a commitment". The place in the transaction where the commitment is taken is called a commitment point.

2.2.1 Commitment Unit

A commitment unit, or CU, is a unit of processing in a transaction. It encompasses all the processing from the last commitment point (or from the beginning of the transaction if it is the first commitment unit) to the current commitment point (or to the end of the transaction if it is the last commitment unit). All writes, deletes, reads, or updates to files are performed according to this unit of processing. When a commitment is taken, the modifications made during the commitment unit are permanent. A commitment allows other transactions (started by other users) to access the same data. Commitments reduce long queues of transactions waiting for resources to become available, and reduce response times. Well placed commitments allow data to be processed faster, more efficiently, and in a more secure way.

The TDS programmer decides where a commitment should be taken within a transaction. As each commitment unit ends successfully, modifications to files or databases are committed. Once the data is committed, the resources are available to be used by other transactions.

2.2.2 Commitment Point

A commitment point is the point, or place, in the execution of a transaction where a commitment is taken. A transaction is always between two commitment points; the previous and the current. At a commitment point:

- All previous processing is completed and cannot be cancelled. Processing includes operations on files, messages, or requests to submit a job.
- Data and files are in a consistent state from the user's point of view.
- Transactions can be restarted if a software or hardware failure occurs. This means that when an incident occurs, all data before the last commitment point is valid.

In addition, the beginning of a transaction is, by default, a commitment point.

At a commitment point, TDS releases resources allocated to the transaction. (During the processing of a transaction, data is transferred from disks to main memory. This data is stored in objects called buffers and pages.)

Commitment points resolve conflicts between TPRs that request the same resources. When a transaction cannot continue processing because of a conflict for resources, the transaction is aborted, then restarted at the previous commitment point. In the meantime, the TPR that had the resource will have released it at its commitment point.

2.2.3 Types of Commitment

There are two types of commitments: explicit commitments controlled by the TDS programmer, and implicit commitment (controlled by TDS). In the generation of the TDS application, the TDS Administrator declares which type of commitment is to be used for each transaction.

Implicit Commitment:

When the administrator specifies an implicit commitment for a transaction, TDS automatically takes a commitment:

- at every conversation
- at the end of the transaction
- at the end of a TPR when the TDS programmer specifies a maximum number of seconds that can elapse between the completion of the TPR and the start of a new TPR (this is known as the wait-time value).

There are commands that allow the programmer to override all implicit commitments, except for those at the end of the transaction.

Explicit Commitment:

Explicit commitments are controlled by the programmer. For every commitment in a transaction, the programmer must add a statement in the TPR program to call a commitment. The commitment will take place at the end of the current TPR.

The four columns in Figure 2-4 compare implicit and explicit commitments, and show how the programmer can change the automatic (implicit or explicit) commitment.

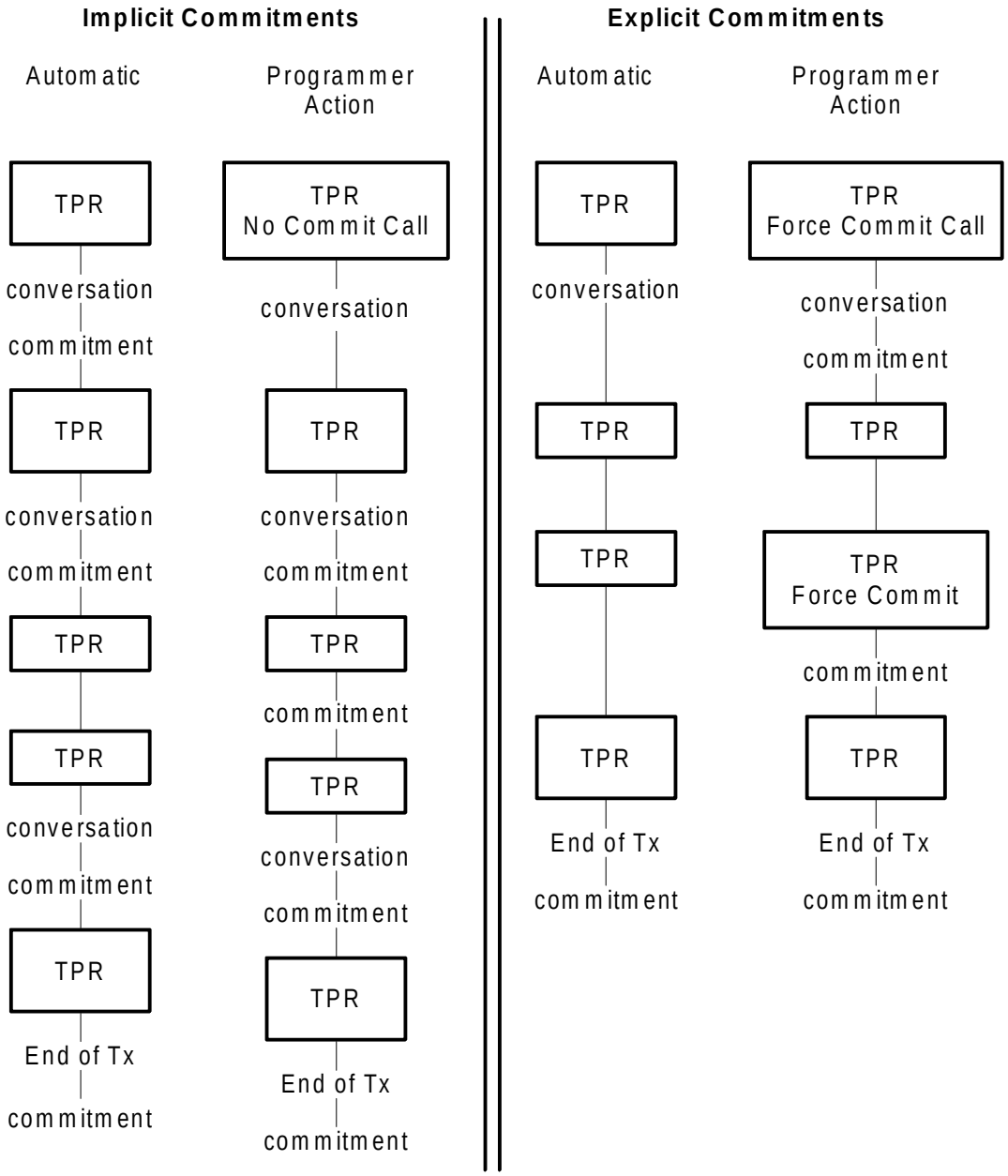


Figure 2-4. Comparison of Explicit and Implicit Commitments

Modifications to files or databases by a TPR are effective only when the TPRs commitment unit is processed.

2.3 PROTECTION OF DATA AND RECOVERY FROM INCIDENTS

TDS provides several mechanisms to protect data from the effects of hardware or software incidents. The Journalization mechanisms and the associated recovery mechanisms maintain the integrity of data in the event of a transaction abort, TDS abort, or GCOS 7 system crash. The integrity procedures and recovery techniques vary according to the type of failure and the level of protection required.

For example, if a system crashes during the execution of transaction that withdraws money from one account and puts it in another account, the "updated" data is inconsistent and possibly incomplete. When an incident occurs, the reliability of updated data depends on whether or not a commitment is taken. The TDS journalization and recovery mechanisms are used to save commitment units and thus handle this type of situation.

2.3.1 Use of Journalization to Protect Data

Journalization is the use of the Journals to maintain data integrity. There are two GCOS 7 Journals that protect data, and another TDS Journal file that can be customized and used for statistics or debugging. The three journals are:

- Before Journal.
- After Journal.
- User Journal.

The Before and After Journals are not specific to TDS, they are GCOS 7 facilities. The User Journal is a TDS facility.

In Journalization, two copies are made of the data being processed. The copies are referred to as "images". The first image of the data is made before the data is updated. This is known as the **before image**. The second image of the data, or **after image**, is made after the data is updated. These images are saved, or journalized, in the TDS Journals; in the Before Journal and the After Journal, respectively.

The Before Journal allows files to be recovered when updates are incomplete due to software failure. The After Journal protects files against any type of destruction, including physical destruction of the file media.

The TDS Administrator determines the level of protection for each application. The journal used depends on whether the file is protected against hardware failure, software failure, or both. Journal protection for a file can be specified either when the file is processed, or in the catalog description cataloged file.

Figure 2-5 shows where the Before and After Journals intervene in a transaction with two exchanges and a file update.

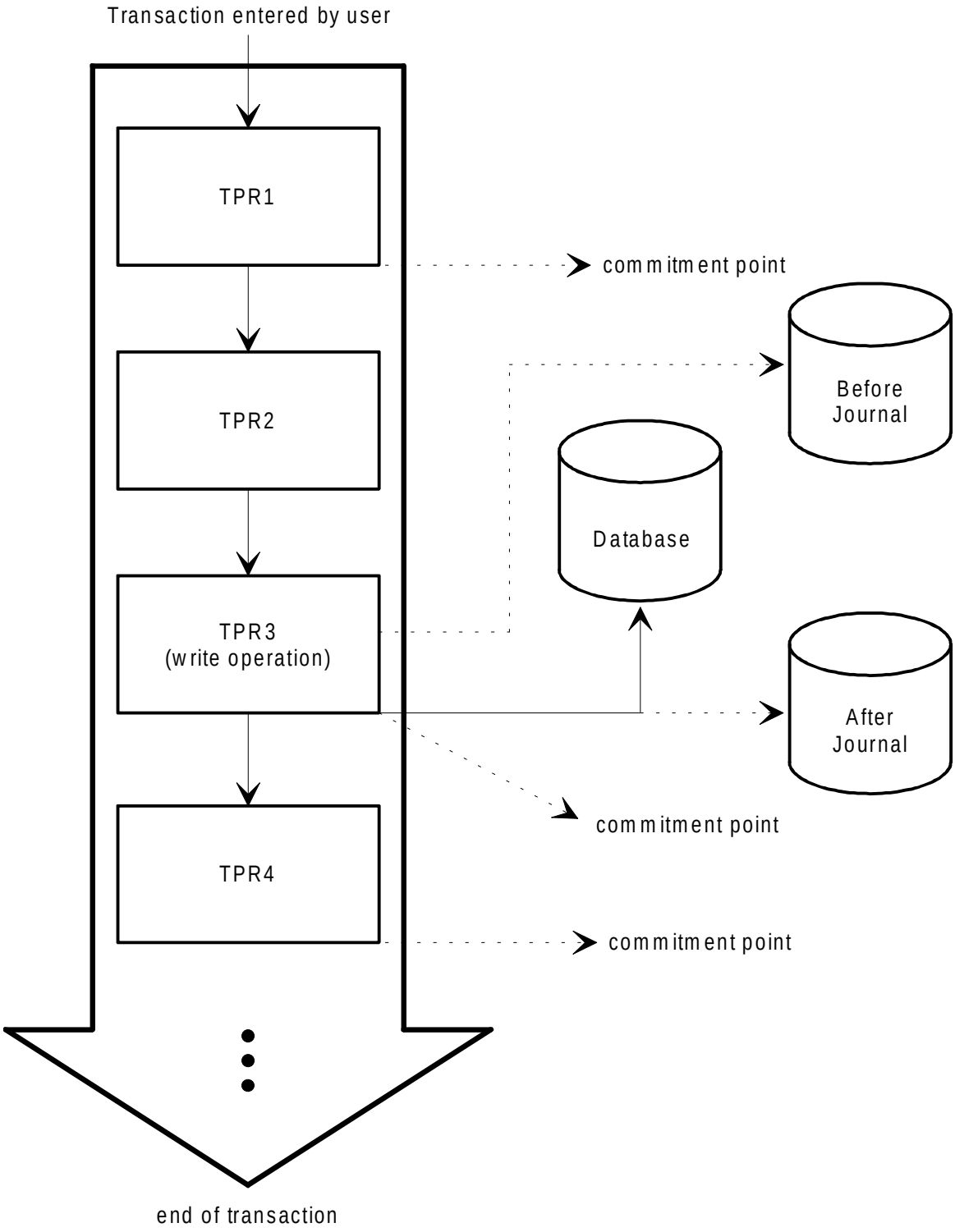


Figure 2-5. TDS Journalization

2.3.1.1 Before Journal

The Before Journal protects files from software incidents. This journal does not protect files against physical destruction due to hardware failure. The Before Journal is used to recover files when updates in progress are aborted.

The Before Journal stores an image of the file to be updated. The images stored in the Before Journal are used to return files to the most recent valid state; that is, to the beginning of the last commitment unit. The images in the Before Journal are made before the updating begins, while the commitment unit is in process. When the commitment unit is terminated, the before images are released.

The Before Journal must be used for files shared between a TDS application and batch programs. This Journal can be used by itself or in conjunction with the After Journal.

2.3.1.2 After Journal

The After Journal protects data from hardware or software incidents. This journal is used to save modifications to a file when an incident occurs after an update is committed, but before it is written to the file.

When a commitment unit is updating a file, the system keeps an image of each record after it is updated, and writes each "after image" to the After Journal. If an incident occurs, it is possible to reconstruct the file by using a previously saved version of the file, and then overwriting each updated record with the after image.

File saving and restoring are not automatic in TDS. The master operator or administrator must perform these tasks using the data management utilities.

2.3.1.3 User Journal

A third journal exists in TDS which allows users to record certain statistics of a TDS application. A user can decide to log input and output messages, as well as information about transactions in the TDS User Journal.

The User Journal is located in the After Journal. The TDS User Journal is not a file protection mechanism, but is explained here because of its location. However, the information in the User Journal can be transferred to a special file when required, and used for debugging.

2.3.1.4 Journal Utilities

There are other utilities associated with the After Journal. These utilities can only be used by the TDS Administrator:

- The ROLLFWD utility uses the After Journal to reset from 1 to 25 user files to the state they were in at a specified date and time.
- The DUMPJRNL utility takes the data in the User Journal and writes it to a sequential output file.
- The After Journal Recovery Utility, JRU, restores the After Journal to a coherent state when it is corrupted because TDS cannot supply the list of aborted commitment units.

2.3.2 Recovery Techniques

The TDS recovery techniques reconstruct the data so that it is in the same state as it was at the time of the incident. The recovery mechanisms must perform these tasks in a rapid and effortless way so that transactions in progress can resume processing.

Three recovery mechanisms are used in TDS:

- deferred updates
- rollbacks
- rollforwards.

These mechanisms are used with the Journals.

2.3.2.1 Deferred Updates

The deferred update mechanism works with the After Journal. If a deferred update is requested for a file, the WRITE or REWRITE statement in the TPR is not physically executed until after the commitment unit is successfully terminated. During the processing of the commitment unit, the output record is placed in a buffer in the After Journal. If the commitment unit does not terminate successfully, the modifications are ignored.

The WRITES to files are made according to these conditions:

- If a commitment unit is aborted, the system erases the contents of the buffer and the after images. No changes are made to the files.

- If a commitment is validated, the contents of the buffer is transferred to the files.

Deferred updates means that updates to files are accepted but are not actually written to disk until the commitment point is reached. If a commitment unit fails, it does not effect the TDS data.

2.3.2.2 Rolling Back a Commitment Unit

When an incident occurs during the processing of a commitment unit, the commitment unit must be restarted. This is accomplished using "rollback" processing. A rollback returns the journalized files to their valid state at the beginning of the last commitment unit.

To perform a rollback, the before images are copied back to the corresponding files, restoring them to their previous state. That is, if a commitment is not taken, data is restored to its original state by applying its before image. The transaction can be restarted, and process same data as if it were accessed for the first time.

A rollback is shown in Figure 2-6. In this example, the first commitment unit terminates successfully. The abort occurs during the second commitment unit (CU2). When a rollback is processed, the files are return to the state they were in at the end of the first commitment unit.

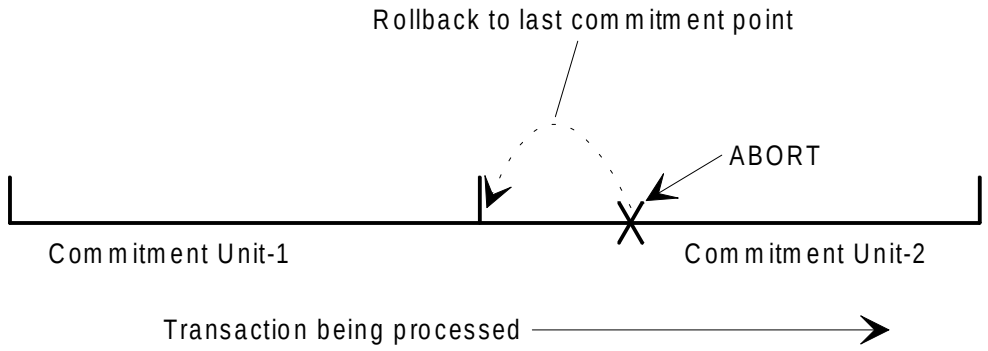


Figure 2-6. Example of a Rollback

2.3.3 Rolling Forward a Commitment Unit

When an incident occurs after a commitment is taken, but before the data is written to the file, the I/O transfer to the file must be restarted. (The operator is notified when the update to the file terminates abnormally). In this type of incident, a rollforward uses the after images to update the file.

If the incident is caused by a hardware problem, the problem must be solved before the after image can be applied to update the file.

A rollforward updates the journalized files to their valid state as determined by the last commitment unit as shown in Figure 2-7. In this example, the first commitment unit terminates successfully but the transfer of these modifications to the file aborts. Because the updates are committed, a rollback can be applied to the file.

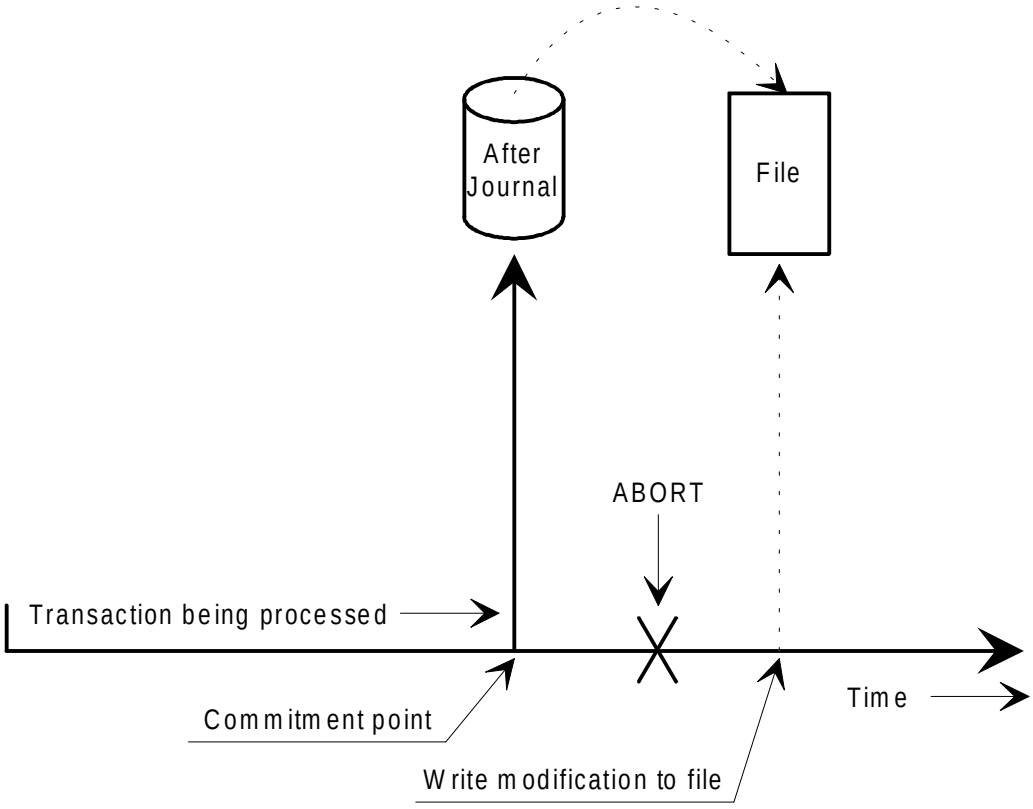


Figure 2-7. Example of a Rollforward

2.4 TYPE OF FILES IN A TDS APPLICATION

In a TDS application, there are two types of files: TDS-controlled and non-controlled files. The basic difference between these files is the way that access conflicts are resolved. Both types of files are UFAS-EXTENDED. However, each type of file has certain restrictions.

When an access conflict occurs for a TDS-controlled file, the conflict is resolved by the TDS Monitor. When a non-controlled file is accessed, it is "locked" (but only during the "access" time) so that other transactions cannot use it.

2.4.1 TDS-Controlled Files

The TDS Administrator can declare a UFAS-EXTENDED file to be TDS-controlled. This file can be either a relative or indexed sequential file. When a file is TDS-controlled, simultaneous access to records in the file is controlled by the TDS Monitor.

When a transaction accesses records in a file, a part of the file is locked for the duration of the commitment unit. Locked records cannot be accessed by other transactions. This is where the TDS-controlled files comes in, any access conflicts are resolved by the TDS Monitor.

For example, two airline ticket agents may try to simultaneously assign the same seat on a flight to two different passengers. If the file containing the seating information is TDS-controlled, the concurrent access to the files is resolved. The identical data requested simultaneously by the two users can be accessed in a controlled way.

The concurrency control mechanism, known as GAC-EXTENDED (General Access Control), allows a TDS-controlled file to be shared concurrently between a transaction and the following:

- another transaction in the same TDS application
- another TDS application
- a batch program
- an IQS application.

The areas of an IDS/II database must be declared TDS-controlled. Under GAC-EXTENDED, TDS-controlled files and IDS/II areas can be assigned exclusively or not to the TDS job. Exclusive access means that access may be given only to the TPR requesting it.

The use of GAC-EXTENDED is specified when the files are assigned. GAC-EXTENDED applies to cataloged or non-cataloged files. In the latter case, the use of GAC-EXTENDED can be defined in the JCL statements that run the TDS job.

Furthermore, TDS-controlled files can be protected by full Journalization facilities.

2.4.2 Non-Controlled File

TDS non-controlled files are also UFAS-EXTENDED files. Non-controlled file can be protected by the Before Journal, but not by the After Journal. This type of file is not protected by GAC-EXTENDED. Therefore access conflicts can occur, for example, between two TPRs of the same TDS application.

Unlike TDS-controlled files, the protection of non-controlled files requires the intervention of the TDS Programmer. The programmer must ensure that when a non-controlled file is accessed, it cannot be accessed at the same time by another transaction in the same or in another TDS application.

There are several ways of protecting a non-controlled files. The programmer can

- Assign the file exclusively to a TDS application.
- Make transactions that are accessing the same file execute non-concurrently.
- Protect non-controlled files by an indirect locking mechanism based on user-defined identification. The form of identification may be letters or numbers. Users can share non-controlled files by associating a file with a given name and then locking the name. This technique must be programmed in the TPRs. Locking and unlocking TPRs allows users to temporarily protect other transactions from concurrently writing to or reading from a file. When the user finishes using the file, the name corresponding to the locked file is unlocked. Because this type of locking mechanism is indirect (that is, the actual files themselves are locked via the use of numbers or letters), no guarantee can be given that the locking/unlocking functions are always effective.

Another way of locking files can be performed by the master operator. The master operator can prevent a specific transaction or class of transactions from being started.

2.4.3 On-Line and Off-Line Files

On-line and off-line files are the files used to generate and operate the TDS application. These files can be considered TDS "system files". They are not user data files that make up the database of an application. A summary of how to use these files is presented in the chapter "The Tasks Involved in Setting Up a TDS Application". For information about individual files, see the *TDS Administrator's Guide*.

Off-line files:

Off-line files are needed to generate, compile and link TDS. These files are not needed at run time. The TDS off-line files are:

<tdsname>.SLLIB	is a source language library that contains the TDS source generation and Linker commands
<tdsname>.COBOL	is a source language library that contains the file descriptions and storage areas defined at the generation.
<tdsname>.SMLIB	is a sharable module library that contains the linked units of TPRs
<tdsname>.LMLIB	is a library that contains the TDS generation load module
<tdsname>.EDITION	is a work file that contains the report produced by the TDS generation program

On-line files:

On-line files are required for processing transactions. Therefore, these files need to be on-line, or accessible, while the application is running. The TDS on-line files are:

<tdsname>.SWAP	is the internal TDS journal file that contains restart information relevant to each terminal logged on to TDS
<tdsname>.CTLM	is a file (controlled by GAC-EXTENDED) that contains control information at generation time and updates this information while the application is running
<tdsname>.CTLN	is a file (not controlled by GAC-EXTENDED) that contains control information at generation time and updates this information while the application is running
<tdsname>.RECOV	is a file that contains information for rollback and dynamic rollforward of TDS-controlled files.
<tdsname>.DEBUG	is a file that contains the results of the trace options.
<tdsname>.PPCLOG	is a file used by the XCP2 service and contains the current state of the pools.
<tdsname>.GMEM	is a file that contains a description of IQS objects.
<tdsname>.H_MMS	is a file that contains information that allows IMAGEWorks to be used.

3. The Tasks Involved in Creating a TDS Application

Creating an operational TDS application involves many different procedures. These responsibilities are handled by the TDS Administrator, the TDS Programmer, the Master Operator, and the End-User.

This chapter explains the tasks that must be carried out, and shows where the responsibilities of each type of user come in during the preparation.

3.1 OVERVIEW OF THE STEPS FOR SETTING UP AN APPLICATION

The TDS environment and generation program is delivered as a standard package. The TDS administrator and programmer must customize the application(s) according to site specifications. Figure 3-1 shows the basic steps that must be performed to customize a TDS application. Next to each step is the type of user responsible for the action.

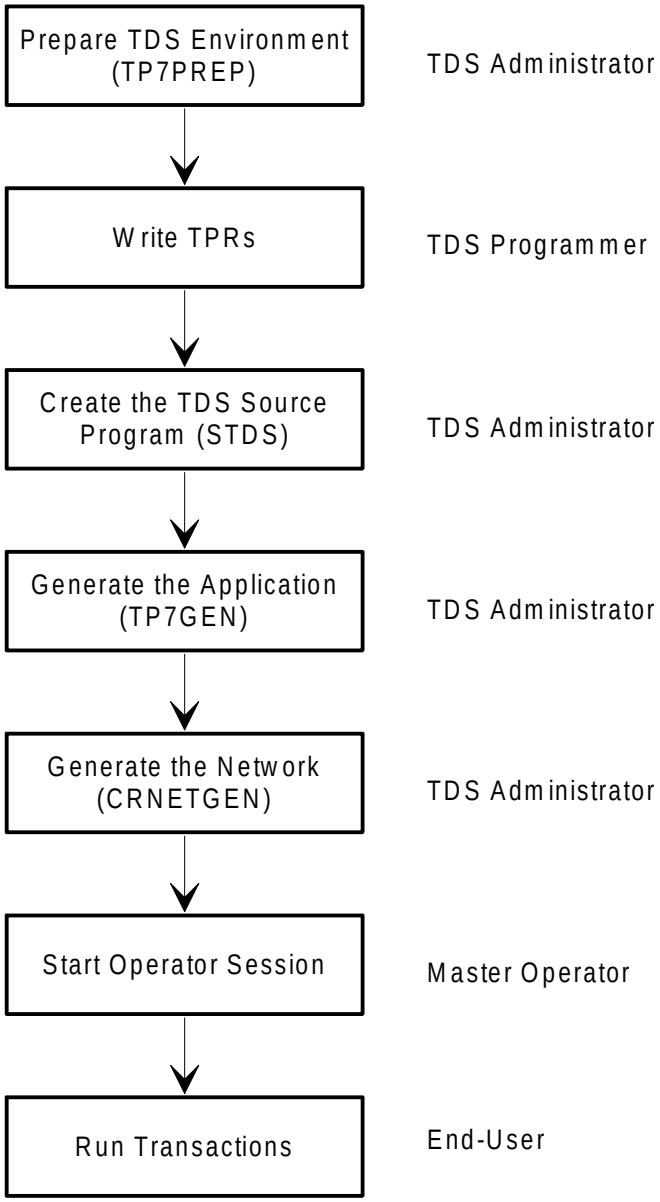


Figure 3-1. The Steps in Building a TDS Application

3.2 DESIGNING AND BUILDING A TDS

The design and creation of a TDS application is handled by the TDS Administrator and the TDS Programmer. After the administrator and programmer set up the application, the Master Operator can start the TDS and End-User can begin running transactions.

3.2.1 TDS Administrator Duties

The TDS Administrator is responsible for planning and defining the application, setting it up, and then overseeing the management of a running application. The administrator creates an application to site specifications by declaring the files, the transactions to be used, the storage structures, and the correspondents that can access the application.

3.2.1.1 Preparing On-Line and Off-Line Files

The on-line and off-line files are the "system files" of a TDS application. They are used to create the TDS environment and to process transactions. These files are not used to handle user data of an application.

The TDS administrator uses the JCL utility, TP7PREP, to create the on-line and off-line files. The TP7PREP utility is a software tool that helps the administrator build the application by providing ready-made files. In this utility, the administrator defines, allocates, and specifies the size of the files as well as the media and device type. If the default file-sizes are not suitable, the administrator can specify other sizes.

3.2.1.2 Writing the TDS Source Program

The TDS Source Program, or STDS, is the program used in the generation of the application. In the STDS, the administrator defines the application. When the program is written, the administrator saves it in the off-line file <tdsname>.SLLIB.

The STDS contains three sections:

- TDS Section.
- INPUT-OUTPUT Section.
- TRANSACTION Section.

In the TDS Section, the administrator specifies the overall constraints of the TDS. In the INPUT-OUTPUT Section, the administrator specifies information about how files are accessed, including file descriptions, file processing modes, file protection, and database schema references. In the TRANSACTION Section, the administrator provides details of all transactions that can be accessed by users, the resources available to transactions, and the restrictions to be observed.

The administrator must arrange the statements of the STDS in the order in which they appear in the TDS Administrator's Guide. For information on the statements used in the STDS, see the *TDS Administrator's Guide.*

3.2.1.3 Generating the TDS Application

After the administrator allocates the required files and stores the source program (STDS) in the off-line file <tdsname>.SLLIB, the TDS Generation program must be run. The TDS Generation program, or TDSGEN, is executed by running the JCL utility TP7GEN.

This utility formats and initializes the TDS "system files", compiles the STDS (Source TDS generation program), and prepares the command files used to link the TPRs. It also creates the members (in <tdsname>.COBOL) that are copied into the TPRs.

3.2.1.4 Cataloging the TDS Authority Codes

The TDS administrator must declare information about users in the Site Catalog. The site catalog contains a list of TDS users associated to projects. Each project is assigned a list of authority codes. The authority codes determine whether or not a project can access a particular set of transactions.

By establishing authority codes, the administrator determines the set of transactions that each operator is allowed to perform, thus eliminating the risk of unauthorized access. For example, if the TDS administrator gives a transaction the authority code of 10 only, to use the transaction the operator must be assigned to a project that has the authority code 10.

3.2.1.5 Generating the Network

In a TDS, terminals often communicate with the application between different physical sites. The communications software necessary for the transmission of data across a network must be described independently of TDS. The purpose of the network description is to reflect the number of users and the types of correspondents declared at TDSGEN.

If the communications network is not already declared, then the administrator must generate it after TDSGEN is executed, and the transactions are compiled and linked. The Network generation, or NETGEN, is processed using the CRNETGEN utility (Create Network Generation). In the NETGEN, the administrator defines the network configuration and declares the number of TDS users and correspondent types.

Figure 3-2 shows a TDS application on a network.

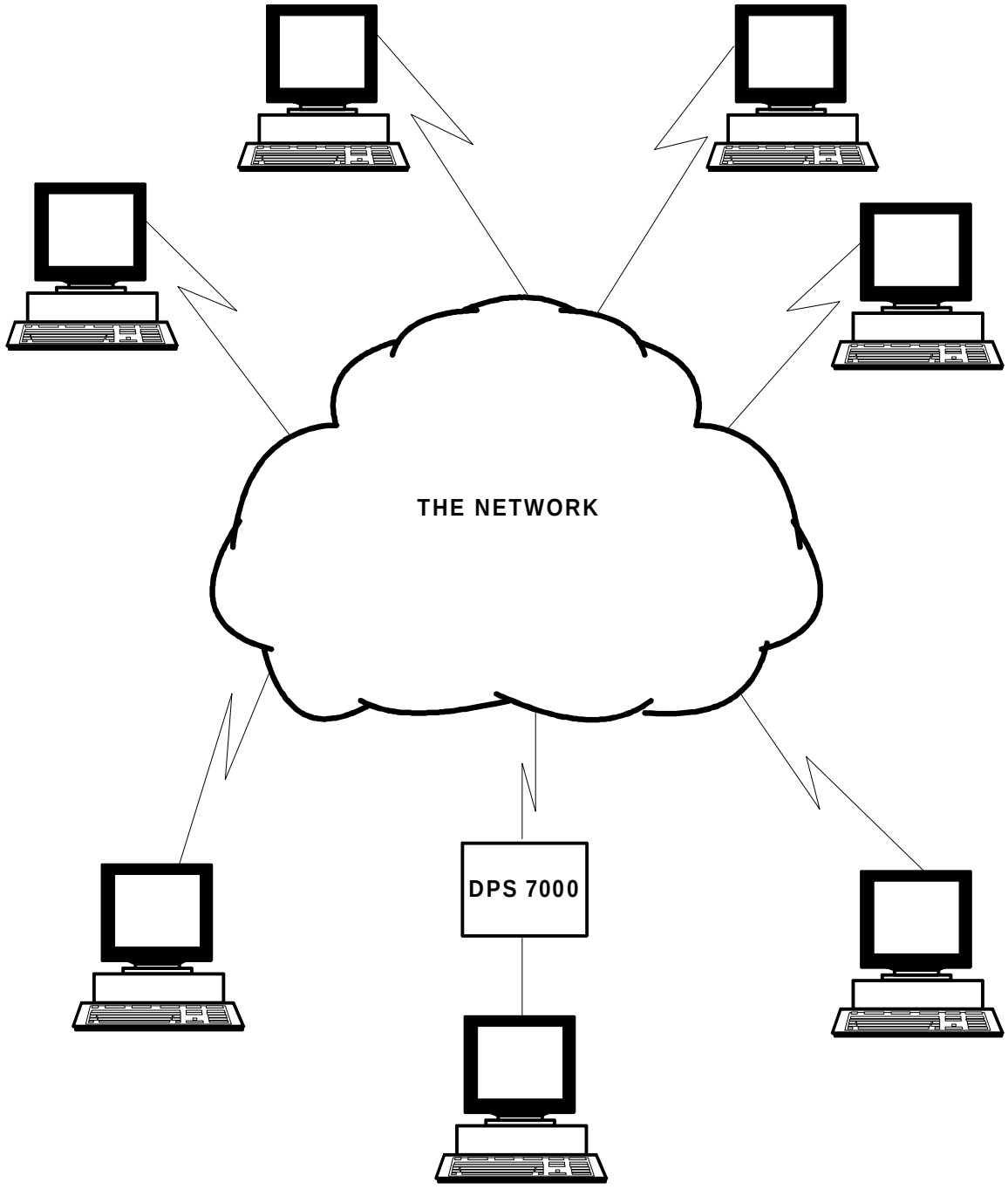


Figure 3-2. TDS Terminals on a Network

Once the network is generated, the communications session and then the TDS session can be started.

3.2.1.6 Optimizing a TDS Application

The TDS Administrator can use these two facilities for measuring performance and optimizing system resources:

- TILS
- SBR.

The System Behavior Reporter (SBR) is a tool used to analyze the workload and usage of the hardware and software resources of a DPS 7000 installation. It can be used to measure the efficiency of TPRs, as well as the behavior of a TDS and GAC-EXTENDED. For more information, see the *SBR User's Guide*.

The Transaction and Interactive Load Simulator (TILS) is designed to simulate a TDS workload. TILS can be used on the same system as the application, or on a remote system. TILS is useful for testing applications under modification or in development. For more information, see the *TILS User's Guide*.

3.2.2 TDS Programmer Duties

The TDS programmer is responsible for:

- writing the Transaction Processing Routines
- writing the special-purpose transactions
- declaring TDS storage areas
- determining how information is displayed on the screen
- handling reports
- testing and debugging TPRs and applications.

3.2.2.1 Writing TPRs

The TDS programmer can start writing the source code for TPRs before the TDS administrator generates the application. Transaction Processing Routines, or TPRs, are the user-defined programs written for an application. TPRs can be written in COBOL, C Language, or GPL. TPRs perform these tasks:

- read files
- process data
- update files
- dialog with the end-user who started the transaction.

For more information on TPRs, see the section "What is a TPR?" in the chapter "Basic Operating Concepts".

Figure 3-3 shows the relationship of TPRs to a transaction. TPR1 through TPR4 are functionally dependent on transaction 2 (Tx2). In this example these TPRs cannot be used by either transactions 1 or 3 (Tx1 and Tx3). However, in an actual application, TPRs are shared by many different transactions.

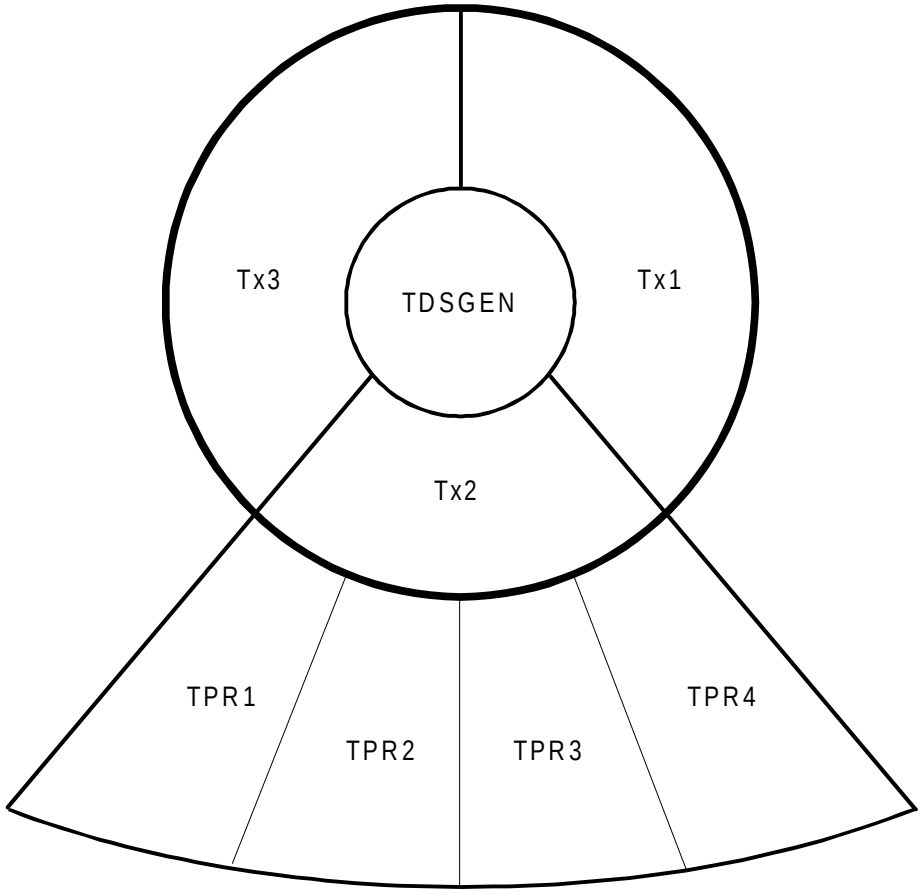


Figure 3-3. The Relationship Between TPRs and Transactions

The TDS programmer must decide which transactions are needed in an application, and then write the TPRs for the transactions. The task of writing TPRs includes coding, editing, compiling, linking, and testing of the TPRs.

The TDS programmer must design and write TPRs to optimize response times. Resources should not be held while waiting for a reply from the terminal operator. Efficient TPRs use and then release resources. Functions must be allocated among different TPRs. If a TPR handles too many tasks, response times are increased, resources are held, and conflicts occur.

The most common TDS design mistake is using one TPR to do everything that needs to be done in a transaction. For example, a TPR can be used to update several records in a restricted set of files. However, a TPR that reads or updates a few dozen records is not efficient. TPRs designed this way conflict with other transactions that need to access these records, or occupy the TDS for a long period, causing delays for other users.

The TDS Monitor provides procedures for managing TPRs, telecommunications, and data.

The TDS programmer can describe files, transactions storages, in the TDS generation program and have the TPRs copy the descriptions by using the COPY function in COBOL. This feature helps to create a consistent structure to a TDS application; it also reduces the time spent coding TPRs, and prevents errors.

Hints for Writing TPRs:

- Design transactions to correspond to a basic unit of processing.
- Write small and compact TPRs.
- Decide which operations are indivisible.
- Make each indivisible operation into a single commitment unit.
- Free resources at the start of a conversation. A conversation requires more time than the time needed to process an exchange.
- Use an outline to write TPRs. Writing individual TPRs can lead to errors and inconsistencies.
- Create special TPRs to handle the processing of "exceptional" items; such as error-handling TPRs.

3.2.2.2 Placing TPRs in a Sharable Module Library

The TDS programmer compiles all TPRs separately, and then links them by running the LINKER utility. This utility creates references between the compile units and sets up the links to TDS procedures. When the TPRs are linked, they are stored in a sharable module library (SMLIB).

When the Source TPRs are compiled, compile units are produced containing the TPR code and data segments as shown in Figures 3-4 and 3-5.

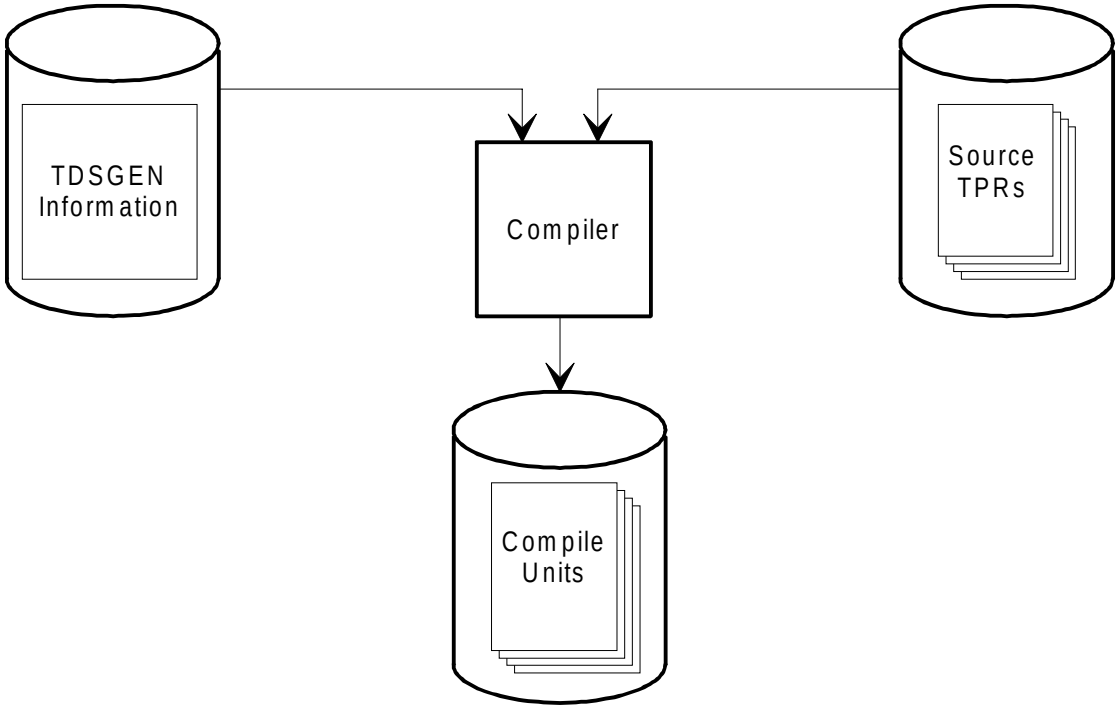


Figure 3-4. Compiling TPRs

Each TPR is composed of code and data segments as shown in Figure 3-5.

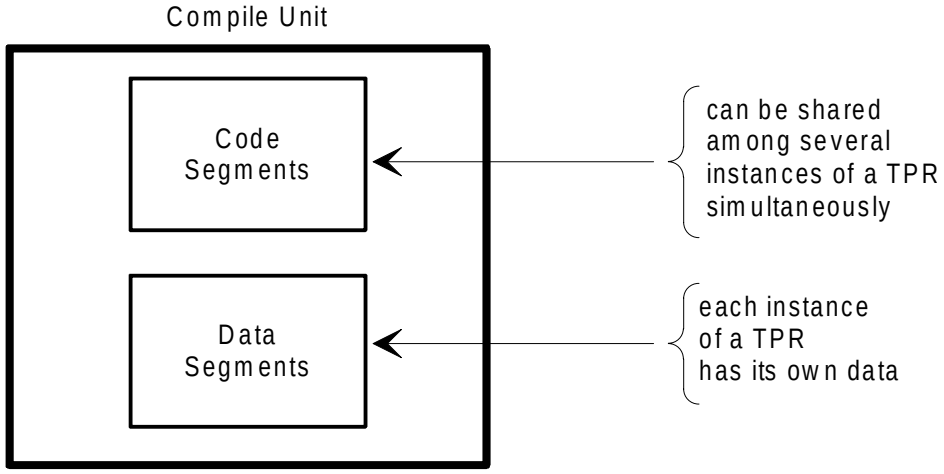


Figure 3-5. Code and Data Segments of a TPR

When the LINKER utility is run, each compiled TPR is stored in a sharable module as shown in Figure 3-6.

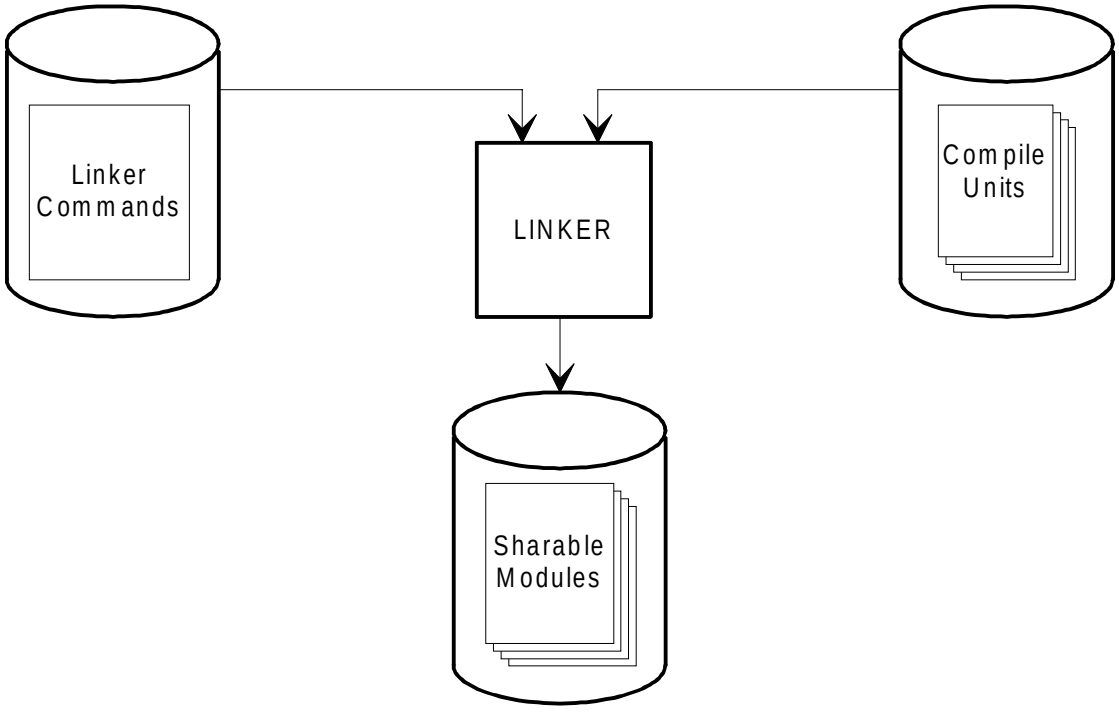


Figure 3-6. Linking TPRs

TPRs are sharable entities. This means that several transactions can use the same TPR at the same time. If a new transaction is activated that requires a TPR currently executing, TDS creates data segments of the TPR in their original state for this new transaction. The code segments need not be duplicated because they are sharable.

3.2.2.3 Testing and Debugging TPRs

There are several ways to test and debug TPRs and TDS applications:

- Use the TRACE command to debug TPRs. This command accepts a subset of the Program Checkout Facility (PCF) commands.
- Specify the DEBUG mode in the TDS generation program. When transactions are executed in DEBUG mode, file updates are cancelled. This means that TPRs can be run and tested without permanently modifying any data.
- Simulate an Operational TDS Application. A special program, known as the TDS Batch Interface, can simulate a fully operational TDS environment without having to use terminals. TDS Batch Interface programs can be written by the programmer, in COBOL, and used to simulate terminal operator functions.
- The batch program can be logged on as a terminal to send and receive messages. The TDS application thinks that a terminal is connected, but does not have the real-time constraints of a terminal.
- Use the Real Time Statistics to display detailed information and general statistics about correspondents and files of a running TDS. This procedure allows the programmer to identify problem areas (blocked users, whether files are open or closed, etc.) and to make the necessary corrections.

3.2.2.4 Defining Special-Purpose Transactions

There are seven default special-purpose transactions used to control a TDS application:

- STARTUP
- LOGON
- RESTART
- BREAK
- DISCNCT (for DISCONNECT)
- LOGOUT
- SHUTDOWN.

Figure 3-7 shows where these special-transactions are in the context of an application.

In a TDS session, there is only one startup and shutdown.

However, during a user session, there may be several logons. For example, if a user is disconnected and subsequently logs on again.

The user can decide to cause a BREAK (for example, by pressing a BREAK key).This interrupts the transaction in progress and starts the BREAK transaction.

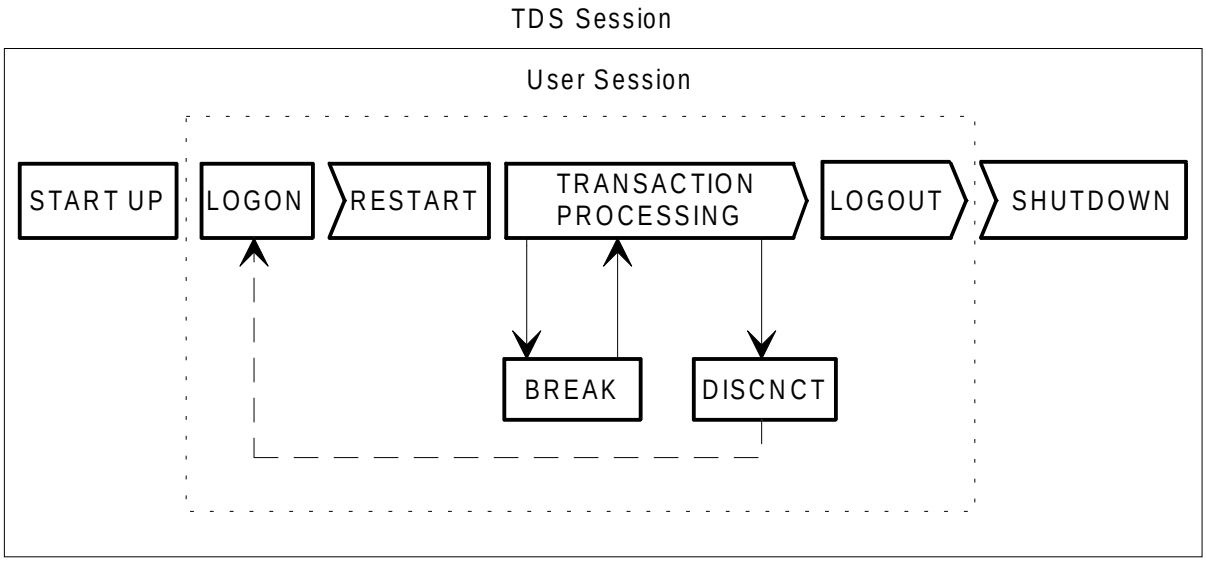


Figure 3-7. Special-Purpose Transactions

The TDS programmer can substitute customized special-purpose transactions for the default transactions. Writing customized transactions is important if the default transactions are inappropriate for a given site or application. For more information on these transactions, see the *TDS Programmer's Manual*.

3.2.2.5 Creating the Application Storage Areas

There are several storage areas in a TDS which allow TPRs to communicate with other TPRs or with TDS. These storage areas allow transactions to access and communicate data:

TDS-STORAGE	A communication area used to pass information between TDS and TPRs.
TRANSACTION-STORAGE	The area that allows one TPRs to pass information to the following TPR within the same transaction.
COMMON-STORAGE	An area shared among all transactions of a TDS, that allows one TPRs to pass information to the following TPR within the same or different transactions.
CONTROLLED COMMON-STORAGE	A communication area, controlled by the GAC-EXTENDED mechanism, that can be rolled back when an incident occurs.
SHARED-STORAGE	User storage areas shared between two or more transactions.
CONSTANT-STORAGE	An area that contains communications control characters.
PRIVATE-STORAGE	A part of the TRANSACTION-STORAGE section that is allocated to one user from logon to logout.
WORKING-STORAGE	An optional storage section used by a TPR in the same manner as in a standard COBOL program.

3.2.2.6 Displaying Information on a Terminal

The TDS programmer can decide to display data on the screen in one of four ways:

- In Normal, or line mode where commands are entered after the READY prompt or after messages sent by a transaction.
- In Format mode, where data is entered on screen-based forms or menus.
- In windowing mode using FORMS.
- Using IMAGEWorks (multimedia product) on a PC (Personal Computer).

In addition, the Terminal Adapter facility allows the TDS Programmer to modify a user profile. When the Terminal Adapter is used, external messages can be sent to an active form, or to the status line (the bottom line of the screen). To use this facility, the programmer must add the TA procedures to TPRs, and the administrator must modify the TDS Generation Program. For more information on the Terminal Adapter, see the *TDS Administrator's Guide* and the *TDS COBOL Programmer's Guide*.

Regardless of the mode selected, an end-user can issue commands or start transactions from a terminal.

When issuing commands, the user enters each command after the "READY" prompt. Commands available to an end-user include displaying the list of message identifiers, re-displaying the last message, or connecting to another TDS application. The characters used to perform these actions must be defined by the TDS administrator.

When transactions are running, the user can only communicate with the transaction. When the transaction is completed, the user can issue another transaction, or a command. The user terminal is notified when a transaction ends normally when either the last message sent, or the service message "READY" is displayed.

Normal mode:

The TDS programmer prepares Normal mode by writing SEND and RECEIVE verbs in TPRs. In Normal mode, end-users can enter commands after the "READY" (or a site specific prompt), or after messages sent by the transaction being executed. (The programmer can modify the "READY" prompt according to site specifications). In this mode, data is entered one line at a time.

Format mode:

In format mode, a form or menu appears on the screen. A form is a set of text blocks with blank spaces, or input fields, where the end-user enter information at the screen. The TDS programmer uses the FORMS facility to create and add forms to an application.

A TDS application accepts data via forms, processes the forms, accesses and updates the database, and output reports in the same manner as data entered in Normal mode. A transaction activates a form, processes it, and then returns an updated form to the terminal. The output form may be either an update to the input form, or it may be an entirely new form.

When forms are used in a application, the accuracy of data entered can be checked to determine whether the data is within the allowed parameters. This is usually the minimum and maximum values allowed for the given transaction. Forms can also supply help and error messages (for example, indicating that an incorrect value is entered) for each input field.

Figure 3-8 shows a simple customer form.

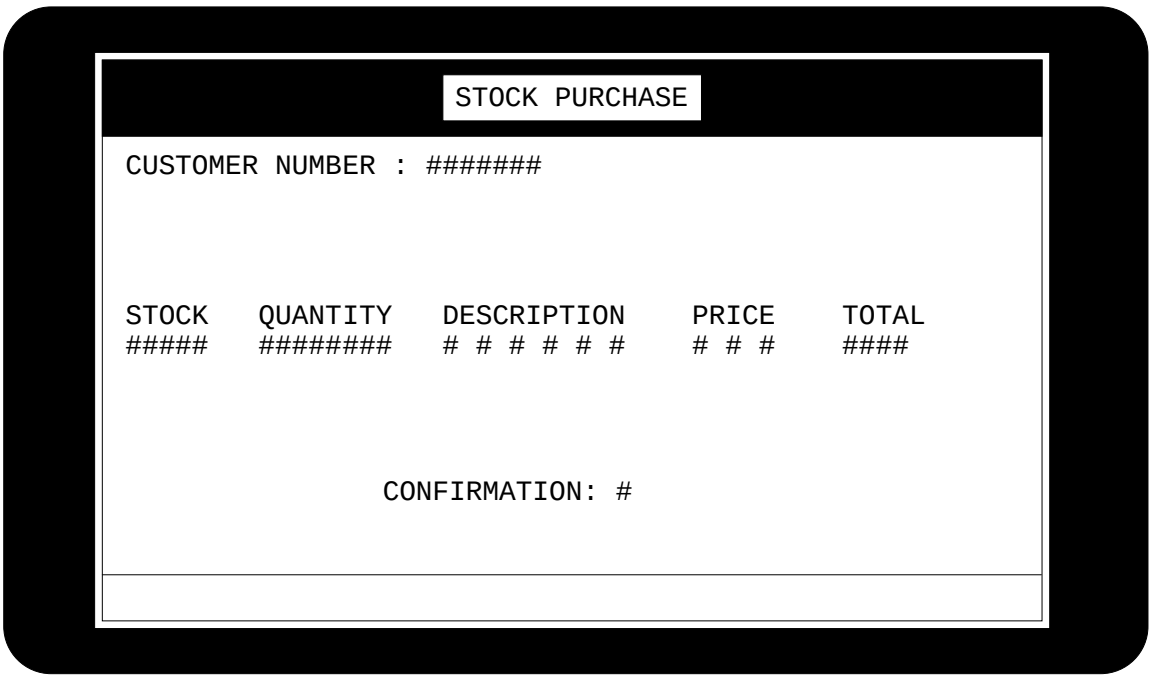


Figure 3-8. An Example of a Form

IMAGEWorks Windowing mode:

On a system with IMAGEWorks, an end-user can access multimedia services from within the TDS application. IMAGEWorks emulates a TDS terminal in a window on PC. A TPR can open other windows through IMAGEWorks, to display lists or documents.

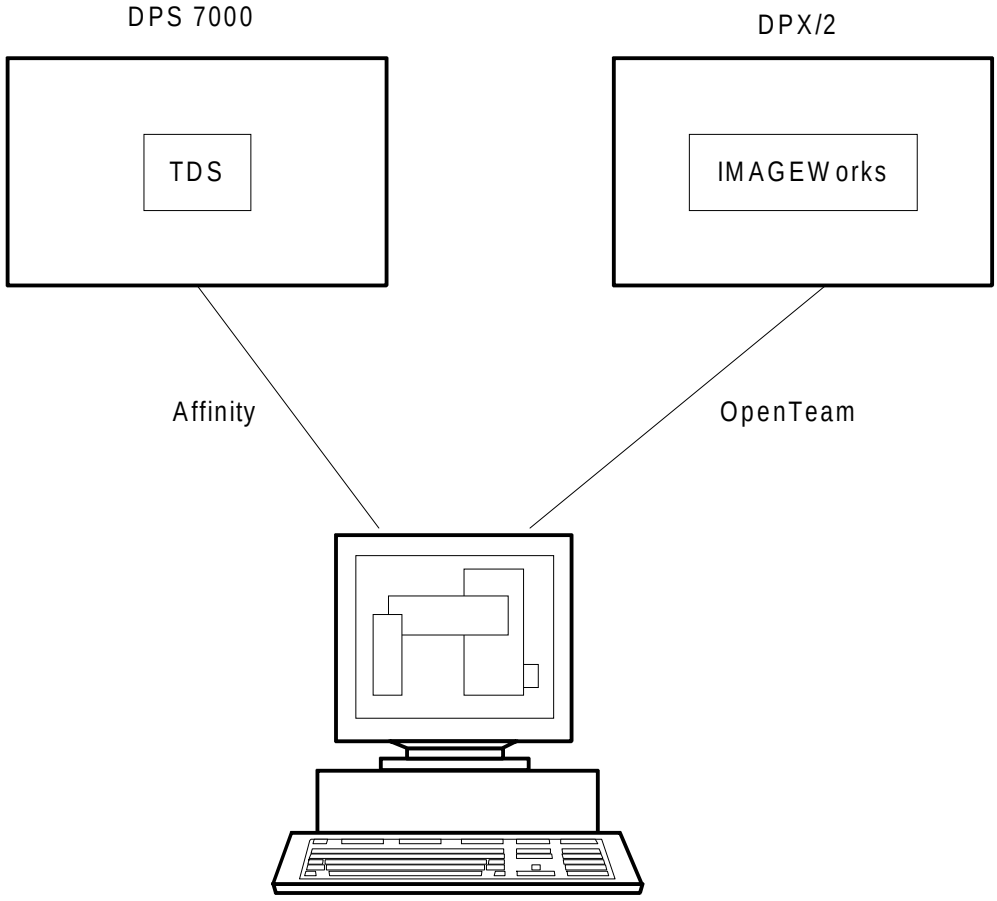


Figure 3-9. Using TDS with IMAGEWorks

When IMAGEWorks is used, the application can be displayed in either Normal or Format mode. IMAGEWorks allows a TDS application to interact with a DPX so that an end-user can get information from either system.

3.2.2.7 Handling Reports

The Generalized Terminal Writer (GTWriter) allows any hardcopy terminal in a communications network to be used as a printer. Any TDS user can request that outputs or files be printed at any terminal known to GTWriter.

To use this utility, the TDS Programmer must add the GTWriter procedures to TPRs. For more information on these procedures, see the *TDS COBOL Programmer's Guide*, or the *Generalized Terminal Writer User's Guide*.

3.2.3 Fitting the Pieces of the Application Together

The TDS Administrator puts the different parts of the TDS together by first generating the network, running the TP7PREP utility to produce the TDS files, then running the TP7GEN utility to create the TDS load module.

The TDS Programmer compiles the source TPR code segments (producing the compile units), and then links them (see Figure 3-10).

Using the SYSMAINT utility, the TDS administrator loads all sharable modules into backing store.

When the TDS is running and transactions are activated, code segments are swapped into main memory from these sharable modules when required. In other words TDS finds the desired TPR in backing store and brings it into main memory for execution.

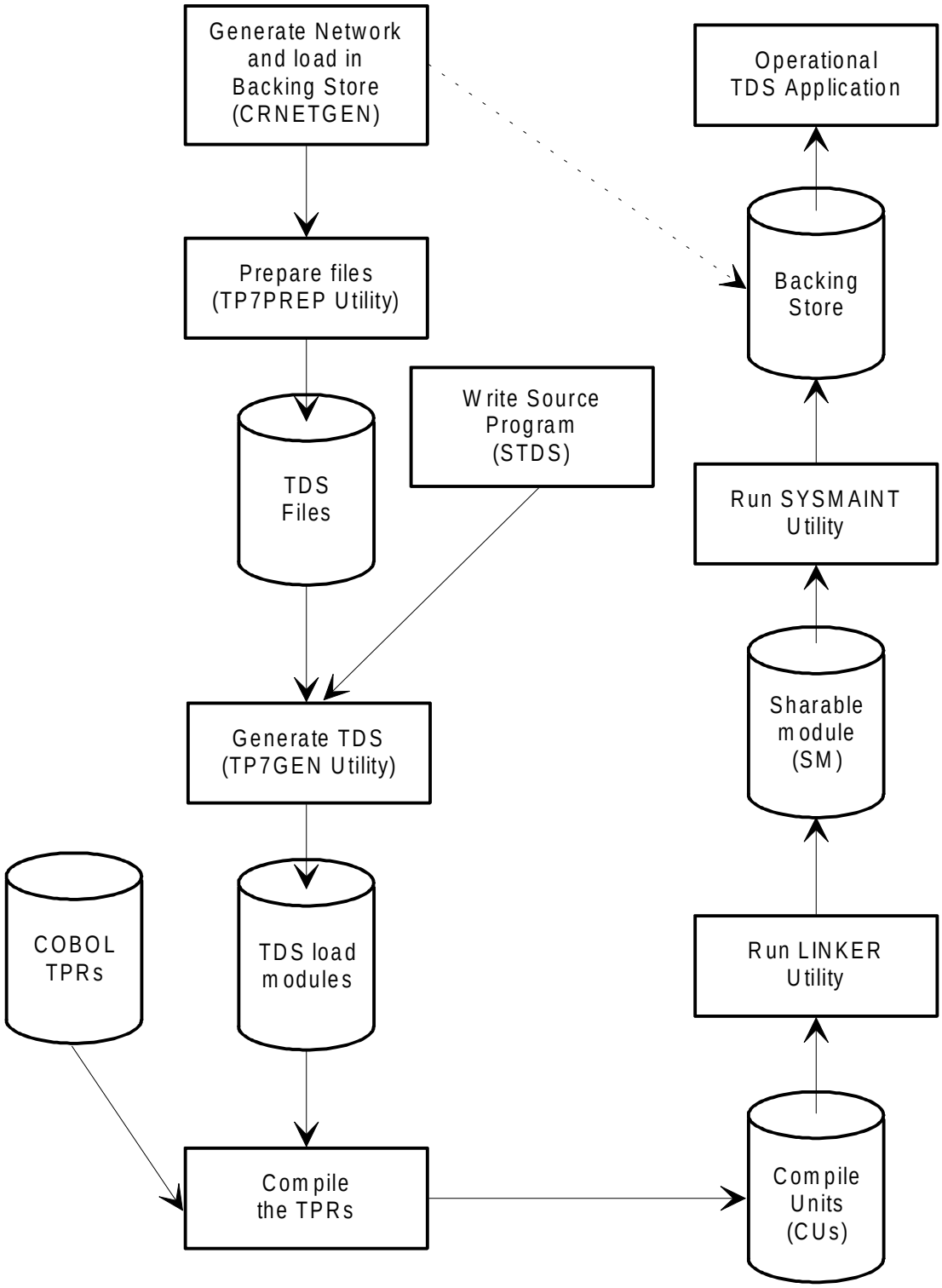


Figure 3-10. Assembly TDS

Figure 3-10 shows how the TDS must be assembled. The steps to be followed are given in the order, though steps 4 and 5 can be done simultaneously.

1. The TDS Administrator or System Administrator generates the network using the CRNETGEN utility.

2. The TDS Administrator allocates the TDS files by running the TP7PREP utility.

3. The TDS Administrator writes the source program (STDS).

4. The TDS Administrator executes the TDS Generation Program by running the TP7GEN utility. This utility produces: LINKER command files for linking TPRs, and the TDS load module.

5. The TDS Programmer writes TPRs and stores them in a source library.

6. The TDS Programmer compiles the source TPRs. The input library (<tdsname>.COBOL) contains the file definitions and storage areas described in the Generation Program.

7. The TDS Programmer stores the compile units in the CULIB library.

8. The LINKER, using the command file produced at TDS generation, links the TPRs.

9. The LINKER stores the resulting linked units in a TPR sharable module (SM) library. There can be several TPR SMs in the same library.

10. The TDS Administrator uses the SYSMAINT utility to load the TPR sharable modules into backing store.

The TDS application is ready to be used.

3.2.4 Later Modifications to a TDS Application

Once the application is running, the administrator or the programmer may need to make modifications or updates to the TDS due to errors, changing requirements, or new developements.

A TDS application that is properly implemented will need fewer modifications than one that is poorly implemented. Depending on the nature of the revisions, the programmer or administrator can make changes at two levels:

- Clauses in the TDS generation program can be changed at any time. The only requirement is to that the TP7GEN utility must be re-run for the clauses to become permanent. Be aware that changing or inserting a clause in the TDS generation program can require that other changes be made. Any TPRs affected by new clauses must be recompiled, relinked, and reloaded into backing store.

- TPRs can be modified and then recompiled, relinked, and reloaded. The TDS Generation Program (the TP7GEN utility) does not need to be re-executed when TPRs are modified.

3.3 RUNNING A TDS APPLICATION

After the Administrator and Programmer create the application, the Master Operator can start the TDS, and then End-Users can log on to the application.

3.3.1 Master Operator Duties

The master operator is the user that is responsible for controlling a running TDS application. Once the master terminal operator logs on to the TDS application, end-users can connect to the application. Furthermore, the master operator is responsible for these tasks:

- starting the TDS application
- opening and closing files
- performing restarts after failures
- stopping the TDS application.

In addition, the master operator can use a special set of commands known as the master commands which allow this operator to control a running TDS application. Master commands can be used to modify or display information about the application. For more information on these commands, see the *TDS Administrator's Guide*.

3.3.1.1 Starting a TDS Application

A session is the duration of an activity. A GCOS 7 session starts when the system is loaded (the GCOS READY message appears). The communications session starts when the Main Console operator enters the required command. Finally, a TDS session can be started by the master operator from the system console or any terminal. There can be up to 5000 connected (end-user) sessions on any TDS. These different sessions are all required in running a transactional application.

Figure 3-11 shows the relationship of the different types of sessions.

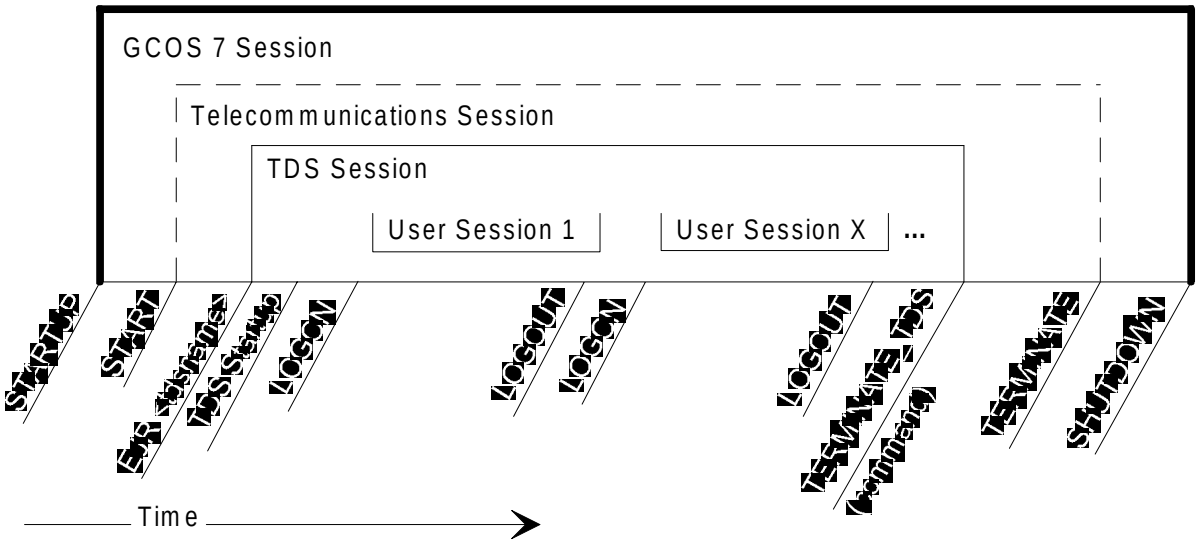


Figure 3-11. Different Types of Sessions

To start a TDS application, the master operator by enters the GCL command ENTER_JOB_REQUEST (EJR) specifying the name of the TDS application. The JCL for activating a TDS job is described in the *TDS Administrator's Guide*.

Once a TDS session is started, TDS operations are under control of the master operator, who can issue privileged (master) commands. Any terminal in the network can act as the master as long as the master operator is connected to it. All other terminals in the network are known as user terminals. User terminals must log on in order to start transactions.

Associated with the concept of master operator is the idea of a mailbox. In TDS, a mailbox is a data structure reserved for the user who controls the TDS session.

- If the master mailbox is defined by the TDS administrator when the TDS is generated, then **the first terminal with the correct access rights in the network to connect** to this master mailbox becomes the master operator. No other user can log on to that TDS application until the master operator is logged on.

- If a master mailbox is not defined by the TDS administrator, **the user who submits the TDS job** automatically becomes master.

For example, when the TDS job is submitted from any user terminal and a master mailbox is not defined by the administrator, the first terminal that logs on becomes the master operator as shown in Figure 3-12.

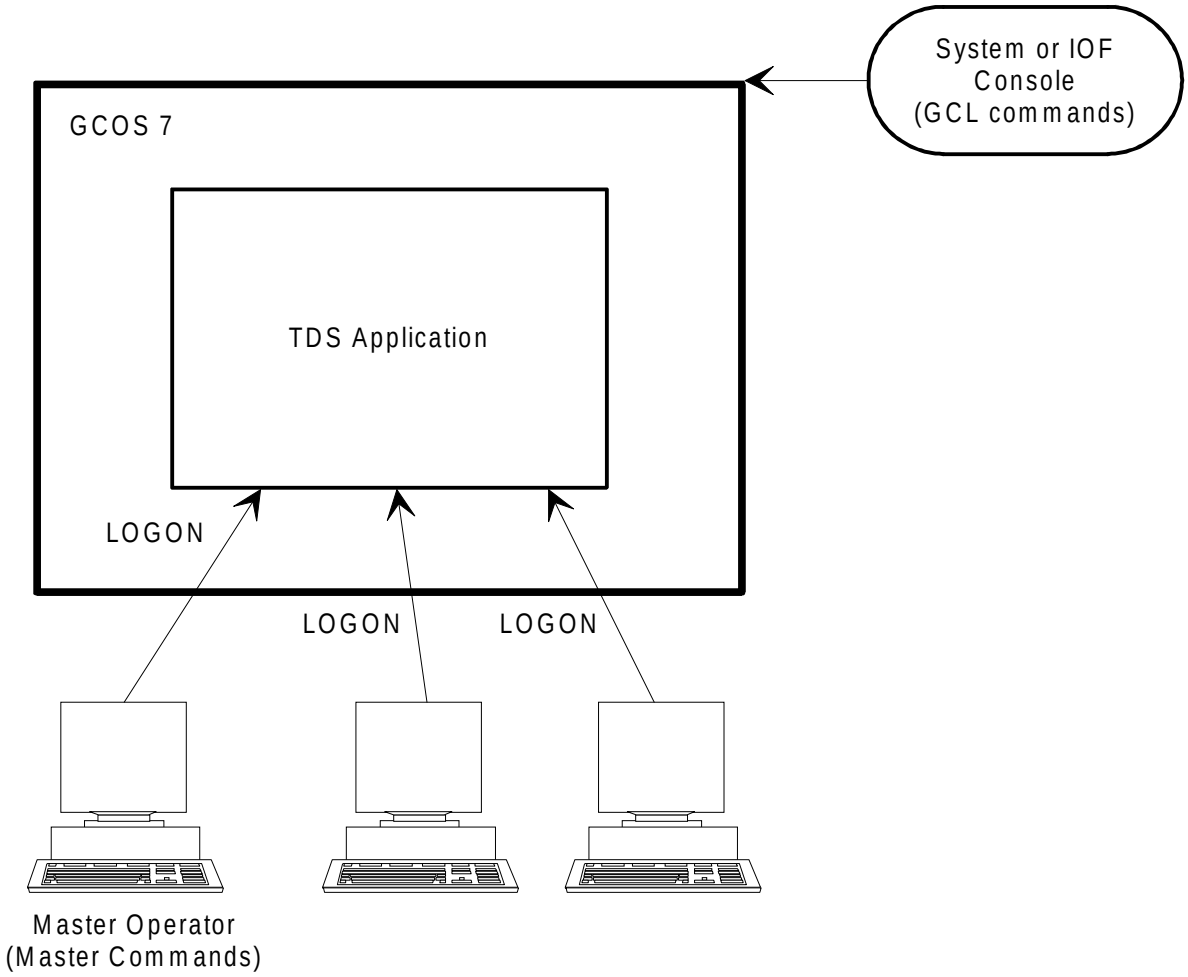


Figure 3-12. End-User Terminal as Master Operator

However, if the TDS job is submitted from the System Console or an IOF terminal, the console or terminal becomes the master mailbox as shown in Figure 3-13.

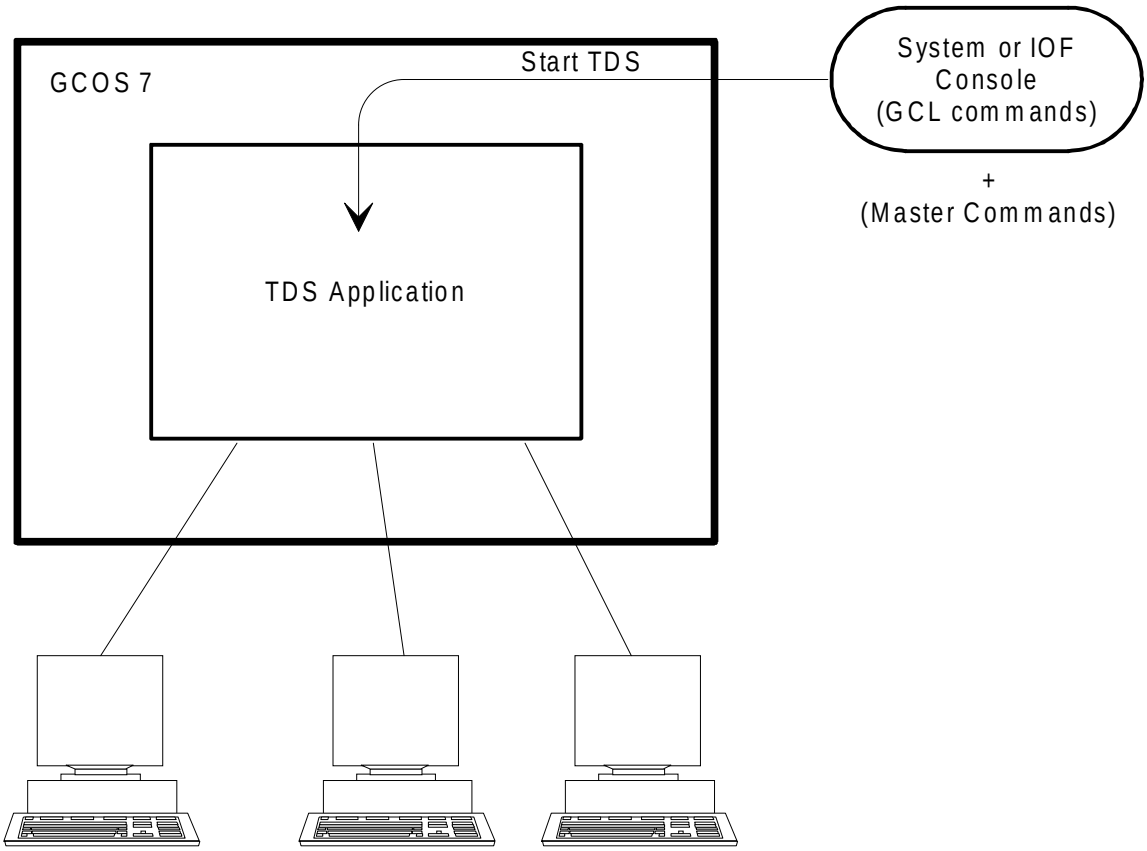


Figure 3-13. System Console or IOF Terminal as Master Mailbox

3.3.1.2 Using the Programmed Operator to Control a TDS

A program can be written using the Distributed Operator Facility (DOF 7-PO) which allows a TDS application to be automatically controlled by a Programmed Operator. A Programmed Operator can control several TDS applications simultaneously.

This facility manages exchanges, responses, or unsolicited messages using services such as data enqueuing and addressee notification. For more information on the Programmed Operator, see the *DOF 7-PO User's Guide*.

3.3.1.3 File Opening and Closing

After a TDS session is started, all files defined in the TDS Generation Program are automatically opened. However, the files can be closed and re-opened dynamically by the master operator using the master commands. At the end of a TDS session, all files are automatically closed.

3.3.1.4 Restarting After Failure

The master operator must intervene when an incident occurs that prevents an end-user from successfully executing transactions.

If a user session is abnormally interrupted without the user having entered BYE (logged off), the user is "frozen" and cannot enter any commands. Interruption can be caused by:

- disconnected terminals
- system crashes
- abort of the TDS (due to a system failure or by master operator command)
- incident on communications network.

The master operator must determine why the incident occurred and then restart the TDS.

3.3.1.5 Stopping a TDS Application

The master operator can log off without affecting end-users who remain logged on to the TDS application. However, the master operator can stop a TDS application, by issuing the TERMINATE_TDS command and specifying whether other users are to be logged off or not.

In normal operations, the master operator allows current users to log off when they have finished the transactions in progress. This message is displayed on the end-user terminals:

```
tdsname SHUTDOWN
```

When the master operator performs an immediate shutdown (the STRONG option is specified in the TERMINATE_TDS command), all user transactions in progress are aborted.

3.3.2 End-User Duties

The end-user session starts when the terminal operator logs on via a terminal to the TDS application. An end-user session is all activities performed by a given user, including the TDS responses, from the time the user logs on until the user logs off a specific TDS application. An end-user can have several sessions during any one TDS session.

3.3.2.1 Starting an End-User Session

To log on to a TDS application, a user enters the following at a terminal:

```
$*$ CN <Name of a TDS application>
```

When a TDS user logs on to TDS, the system verifies whether or not the user is allowed to log on to TDS by checking that the user is listed under a project that can access the application. When the logon is successful, the "READY" prompt (or the application's equivalent) appears on the screen. At the logon, unknown users, duplicate users (that is, users who are already logged on), or users with incorrect passwords are rejected. The TDS administrator can also set up an automatic logon facility that allows a terminal to be automatically logged on to a TDS application as soon as the terminal operator logs on to GCOS.

3.3.2.2 Starting Transactions

An end-user connected to a TDS application can enter a transaction code (message-id) to activate a transaction and thus start application processing. The message-id must be one of those defined by the TDS administrator when the application is prepared. A message-id can be up to eight characters long, can be followed by a space, and can include the parameters expected by the transaction.

If a user enters a message-id that is not valid in the TDS application, "UNKNOWN TRANSACTION" is displayed at the terminal. End-users can activate only those transactions which the TDS administrator has authorized them to use.

In Transaction mode, an end-user can interrupt a transaction that is currently executing by pressing the BREAK key or by entering $*$BRK. The resulting actions depend on how the TDS administrator has set up the TDS application. Usually, sending a BREAK in command mode has no impact other than the sending of a "READY" message. However, if a BREAK command is sent from a passive terminal, the terminal is reset to active. For more information on active and passive terminals, see the section "Correspondents" in the chapter "Designing and Optimizing a TDS Application".

3.3.2.3 Terminating an End-User Session

When a transaction completes execution, it returns a status to the terminal. The user is told when the transaction is completed, but remains logged on to the TDS application. The user can either start another transaction or log off from the terminal.

When all his/her TDS activities are finished, the end-user can terminate the session by issuing the BYE command. The BYE command prevents other end-users from logging on to someone else's end-user session. In addition, the master operator can force an end-user to log off.

Figure 3-14 shows the sequence of events of an end-user session.

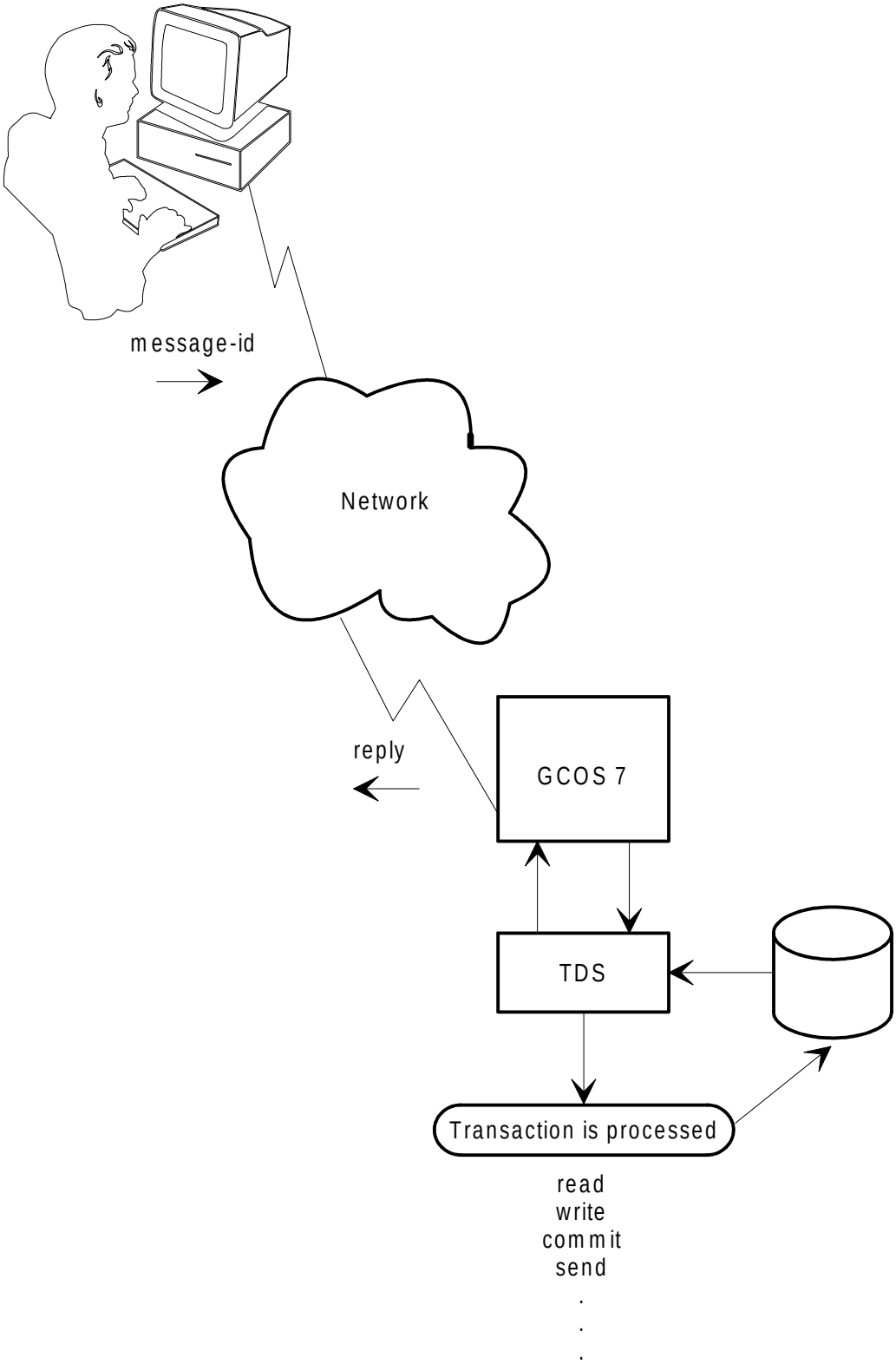


Figure 3-14. Sequence of Events when a Transaction is Submitted

4. Designing and Optimizing a TDS Application

This section focuses on concepts that the TDS Administrator must be familiar with to design an efficient application.

- simultaneity
- non-concurrency
- spawning
- mapping
- correspondents
- communications protocols
- security functions.

4.1 CONTROLLING THE RATE AND AMOUNT OF DATA PROCESSED

In GCOS 7-V6, a TDS application can have up to 2000 transactions. TDS applications, such as those used in banking, airlines, or inventory control, must support a high volume of transactions. In large applications, many transactions run simultaneously.

In applications where many transactions run simultaneously and files are accessible to all users, concurrent access to the files must be controlled. Controlling access to files ensures that data remains logically consistent. The TDS administrator controls access to files and resources by specifying the SIMULTANEITY, NON-CONCURRENT, and UNMAPPING clauses in the generation of the application.

- The simultaneity level determines the maximum number of TPRs that can be executed simultaneously.
- Non-concurrency prevents specified transactions from being run simultaneously. This is used to avoid resource conflicts and prevents concurrent access to non-controlled files.
- Unmapping releases resources held by transactions being processed.

4.1.1 Simultaneous Transactions

The TDS Administrator can control the number of simultaneous transactions by specifying a simultaneity level in the TDS Generation Program. This clause allows the administrator to specify the maximum number of processes available to handle TPRs, and thus defines the maximum number of TPRs that can execute concurrently. In V6, the maximum number of TPRs that can be executed concurrently is 140.

A simultaneity level of 1 does not protect files from concurrent access. Although only one TPR can be processed at a time, several transactions can be active, and their respective TPRs are simultaneous.

Figure 4-1 shows that two TPRs of two transactions are active and try to access the same resource.

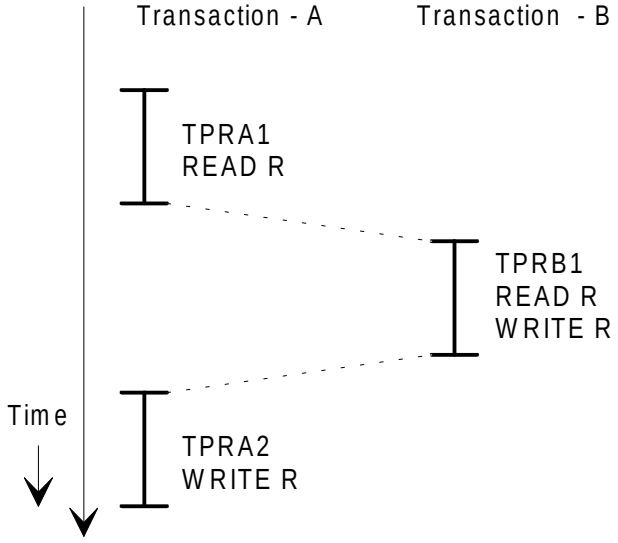


Figure 4-1. Simultaneity Level of One

Even though the level of simultaneity is 1, TPRA1 can read a record, then TPRB1 can read and rewrite the same record before the record is written by TPRA2. The file reflects only the update performed by Transaction A.

To prevent this type of problem, the TDS Administrator can specify that certain transactions be non-concurrent, as described below.

4.1.2 Non-concurrent Transactions

The TDS Administrator can declare transactions as non-concurrent with (a list of) other transactions at the TDS generation. By specifying transactions as non-concurrent, the TDS Administrator:

- avoids resource conflicts which reduce system performance
- prevents concurrent access to (TDS) non-controlled files.

In a TDS application, many tasks are independent and can be performed at the same time (or concurrently) by separate transactions. For example, several users can concurrently read or update data in a TDS application. However, sometimes it is advantageous to complete tasks one step at a time. This is done by declaring transactions to be non-concurrent, or specifying non-concurrency.

Non-concurrency prevents the commitment units of TPRs from being run simultaneously when there is a risk of resource conflicts or even deadlocks. Non-concurrency allows TDS to handle these conflicts automatically, but at the expense of system performance. As far as performance is concerned, it is more efficient **not** to declare file non-concurrent, but to use TDS-controlled files protected by GAC-EXTENDED.

The non-concurrency feature can be controlled automatically or manually depending on the choice made in the TDS Generation Program.

- If the non-concurrency control is automatic, TDS automatically locks out the non-concurrent transactions at the commitment unit level. That is, as soon as a non-concurrent transaction starts to execute, any transactions declared non-concurrent with the initial transaction are not permitted to execute until the end of the commitment unit.

- If the non-concurrency is manual, the TDS Programmer can decide when and where non-concurrent transactions occur by switching the protection on and off.

Running a transaction non-concurrently with one or more other transactions is required to prevent concurrent access to non-controlled files. When transactions are allowed concurrent access (that is, **not** declared non-concurrent), the way a resource conflict is handled depends on whether the file is controlled by TDS or not. On TDS-controlled file, if two TPRs try to update the same record, the commitment unit can be aborted. In the case of non-controlled files, one of the updates can be lost.

Figure 4-2 shows Transaction-1 and Transaction-2 competing for access to Record-R. While Transaction-1 is accessing Record-R, it has complete control over the record. Meanwhile, Transaction-2 is also trying to gain control of the record. Transaction-2 is activated just after Transaction-1, but cannot start executing until the commitment unit in Transaction-1 ends.

When the administrator specifies non-concurrency, each commitment unit of the transaction is executed non-concurrently. That is, Record R-is accessed first by the commitment unit of Transaction1 and then by the commitment unit of Transaction 2.

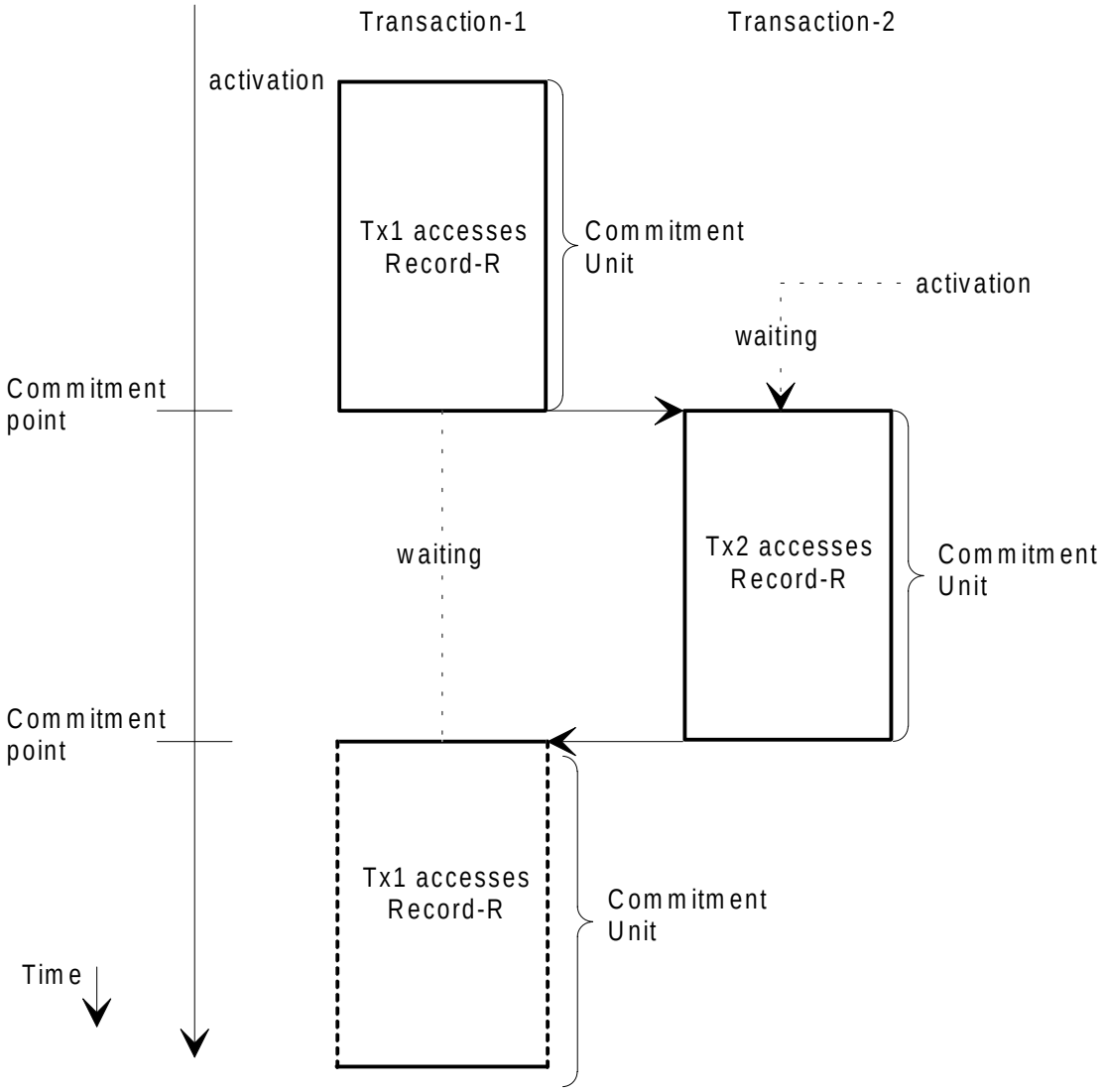


Figure 4-2. Transactions Specified as Non-Concurrent

Even when a commitment unit corresponds to several TPRs, as shown in Figure 4-3, there is no danger of corrupting files because all processing performed by the first transaction is completed before the second transaction can access the record.

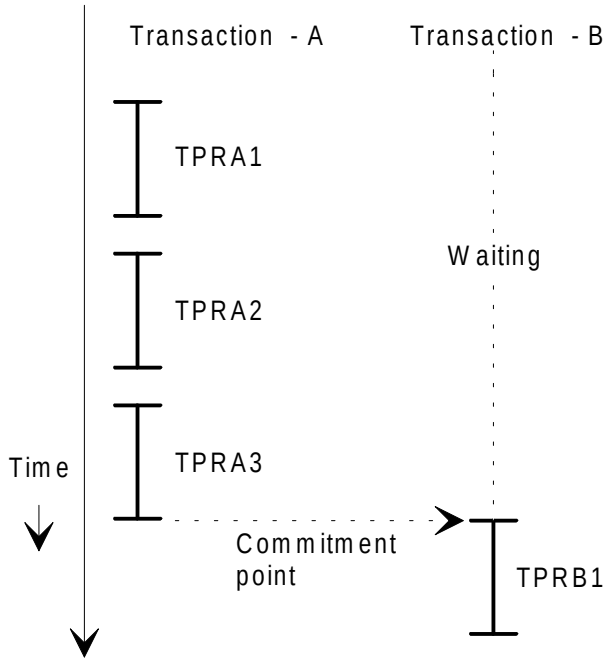


Figure 4-3. Commitment Units of Non-Concurrent Transaction

In this example, TPRA1 reads a non-controlled file and TPRA2 rewrites updated data to it. After this modification is made, TPRB1 can read and rewrite the same file.

4.1.3 Mapping and Unmapping Resources

In a TDS application, there are general-purpose control tasks used simultaneously by several transactions. (A task is a set of system resources necessary for executing a sequence of code). These tasks correspond to physical "processes" in the GCOS7 meaning of the term, and are the processes used to execute TPRs.

To process a transaction, TDS allocates one of these control tasks to a TPR and then executes the TPR. The allocation of a control task to a TPR is called **mapping**. At the end of the TPR, TDS determines whether a commitment is to be taken and whether the control task is released. When the control task is released, the TPR is **unmapped** and the control task can be used by another TPR. Note that mapping is done at the beginning of a TPR and unmapping is done at the end of a TPR.

When a TPR is unmapped, its context (or information about the TPR and its current state of execution) in the session is saved so that it can be restored later to be used by the next TPR. The context of a TPR is saved in a swap file. If the following TPR needs to read the context of the preceding TPR, it accesses and then reads the swap file.

Figure 4-4 shows two transactions. The TPRs of the first transaction are unmapped, in the second transaction the resources are not unmapped, or are declared as having "NO AUTOMATIC UNMAPPING".

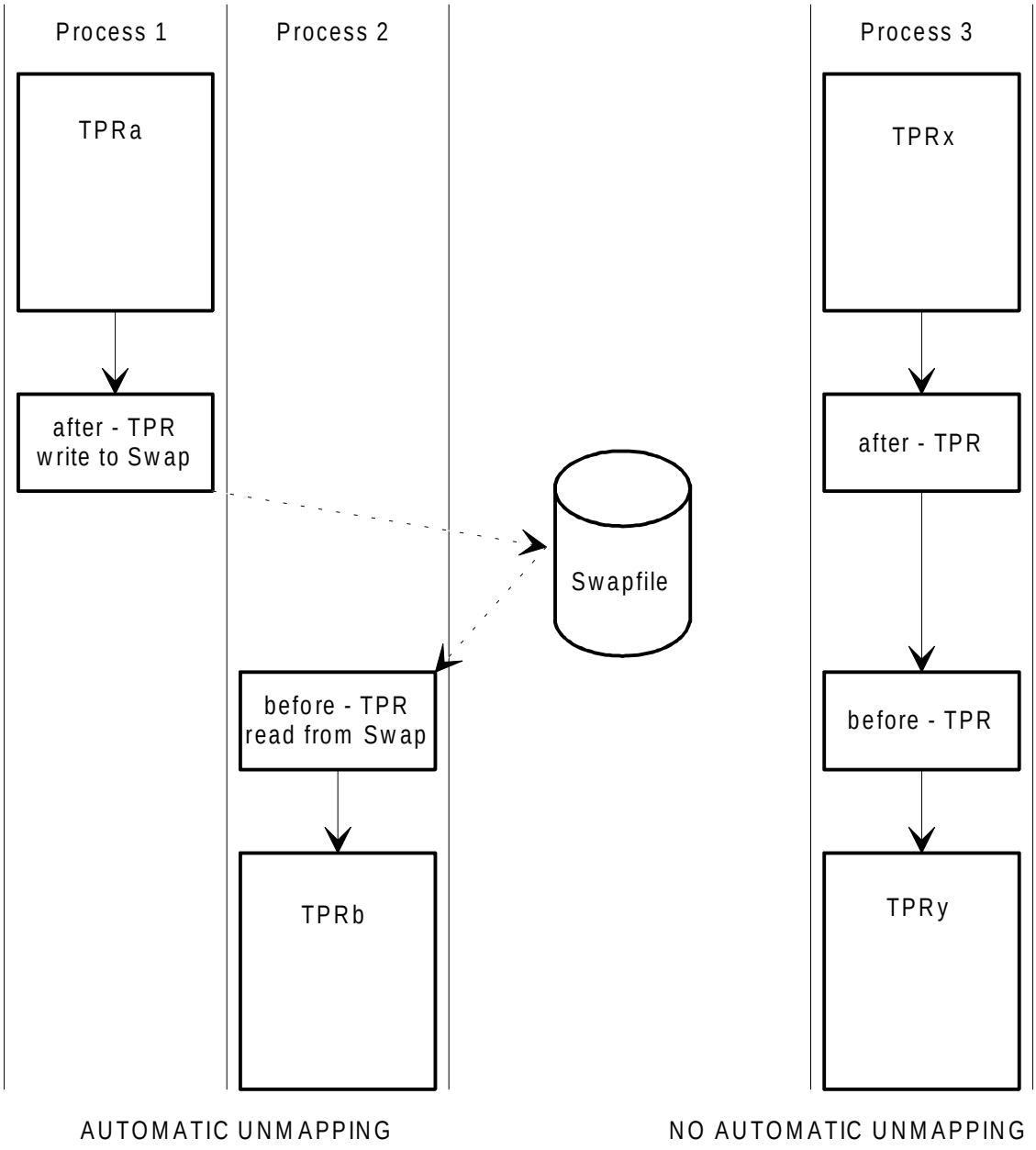


Figure 4-4. Unmapping versus No Automatic Unmapping

The use of the swap file (in unmapping) increases the number of input/output operations. However, when TPRs are not unmapped, response time can be increased due to queues of TPRs waiting for tasks.

Unmapped transactions occupy a control task only when a TPR is executing. The TPR releases the task when a conversation is started. Unmapping allows a few control tasks to serve a large number of transactions, running almost in parallel.

The TDS Administrator can specify that unmapping occur systematically at the end of each TPR, or that automatic unmapping is suppressed. When TPRs are not unmapped, the I/O overhead is reduced because there is no need for saving the transaction context between TPRs. This is particularly significant for transactions composed of many TPRs. Although specifying "NO AUTOMATIC UNMAPPING" improves the performance of certain transactions, this clause can reduce the overall performance of a system.

4.2 COMMUNICATION BETWEEN CORRESPONDENTS, APPLICATIONS, AND OTHER GCOS 7 PRODUCTS

TDS applications can exchange information with applications and users of other TDS applications.

This section explains:

- Types of Correspondents.
- Spawning.
- Communication protocols used for dialogs inside the same TDS and/or between TDS and IBM, or IBM-compatible applications, and the protocols used for transaction processing on UNIX.

4.2.1 Correspondents

Definition of Correspondent:

A **correspondent** is simply an "object" that a TDS application can "talk to". The "object" can be a terminal (or the end-user logged onto it), a "virtual" correspondent, a "dummy" correspondent, or even an application. Correspondents must be defined in the TDS Generation Program so that TDS can establish the necessary connections. However, correspondents that initiates a connection to TDS, as a terminal does, do not need to be declared at TDSGEN.

Terminal Correspondent:

In general, correspondents (terminals) are linked to TDS when the correspondent logs on. However, certain correspondents such as printers can be automatically assigned. Applications are connected to TDS when requested by the master operator.

The master operator is the correspondent in charge of the TDS application. Like any other user, the master operator correspondent can start transactions.

Correspondents which are terminals can operate in either active or passive mode.

- A terminal in **active mode** can initiate transactions and can use a wide range of functions.

- A terminal in **passive mode** cannot be used to start transactions. Messages can be sent to the terminal, or transactions can be activated for it from elsewhere. A passive terminal can take part in conversations but is not set input mode when the transaction terminates.

  A passive terminal is often a terminal without a keyboard, but can also be an active terminal set to passive by the TDS programmer. The programmer can set a terminal with a keyboard to passive mode by sending a specific TPR.

Application Correspondent:

Access to a correspondent which is an application requires the use of a communications protocol (see the paragraph "Protocols for Communication between Correspondents" in this chapter).

4.2.2 Spawning New Tasks from within a Transaction

Starting a new transaction from within a transaction is called **spawning**. In spawning, one correspondent starts a transaction on behalf of another correspondent from an active transaction. The transaction that activates the operation is known as the spawning transaction and the transaction that is started is known as the spawned transaction.

Once a transaction is spawned, its execution is independent of the transaction that activated the spawning. Spawned transactions are placed in queues and activated when the correspondent on which the spawned transaction is directed, becomes idle. A spawned transaction is validated when the commitment unit in which the spawning is performed ends normally.

Spawning can be used when a transaction needs access to facilities (such as printers) which the transaction cannot start. Spawning can also be used to start a transaction at a specified time in the future, or to run a transaction using different priorities than the ones initially requested. A transaction spawned to a dummy correspondent can only retrieve transaction parameters; it cannot send information to the dummy correspondent.

Figure 4-5 shows a simple example of spawning. In this example, Transaction-A spawns Transaction-B to user X. Transaction-B can be executed immediately, after a specified time interval, or at a specified time of day depending on the requirements of Transaction-A.

The CALL "SPAWN" statement is executed at time t2, but is not validated until time t3. Transaction-B remains in the spawning queue between time t3 and time t10. During this period, Transaction-0 (TX0) is executed for user X. At time t10, Transaction-B is executed for user X.

The starting time of Transaction-B is not dependent on the finishing time of Transaction-A, unless the correspondent of Transaction-A is the same as the correspondent associated with Transaction-B.

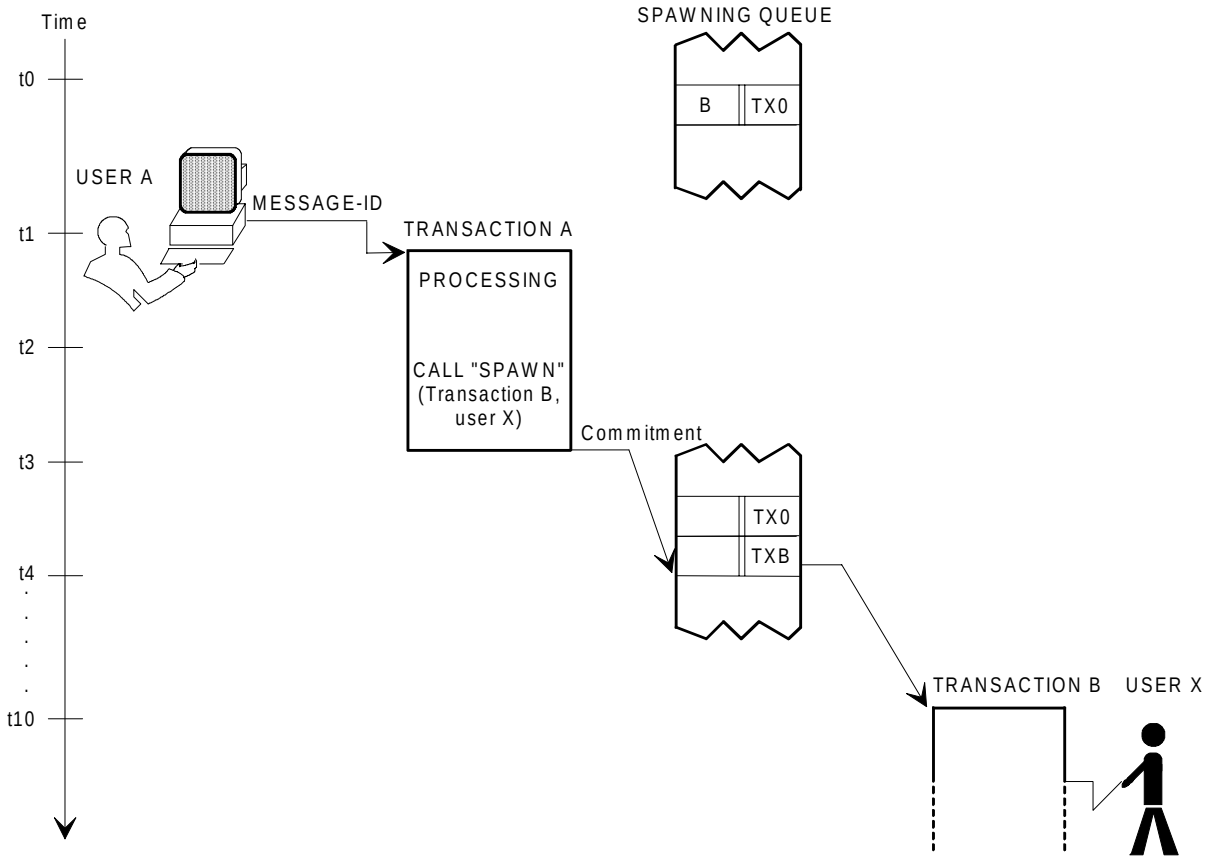


Figure 4-5. Spawning Transactions

4.2.3 Protocols for Communication between Correspondents

Users communicate with TDS or with an application during a **communications session**, which is a structured dialog. Applications can also communicate with each other. A predefined structure is necessary so that these dialogs are standardized. **Protocols** are the set of communication rules between two correspondents of a communication sessions that provide this standardization.

TDS can use these communication protocols:

- the Terminal Manager (TM) Protocol
- the Pass Through (PT) facility
- the Extended Communication Protocol XCP1
- the Extended Communication Protocol XCP2.

4.2.3.1 The TM Protocol

The Terminal Manager (TM) protocol is a facility used to manage the communication between a terminal or group of terminals and a correspondent. (A correspondent can be another terminal or an application with which the session is established.) The TM translates application data to the format defined for the terminal.

There must be a logical connection between the TM and the correspondent. The information necessary for this connection is declared by the administrator during the generation of the communication network.

4.2.3.2 TCAM and TDS Pass-Thru capabilities

The TDS Pass-Thru facility allows a TDS end-user to log on to remote application without requiring the user to log off the active TDS host session. (A **host session** is the initial TDS application to which the user logged on.) The remote application can be another TDS application (in the same room or on another site), or IOF.

The TDS Communications Access Method (TCAM) opens the TDS Pass-Thru session, or logical link, between the user and the application. TDS creates this link at the time of the connection. Figure 4-6 shows a user with two sessions: the principal session and the TDS Pass-Thru session. Only one TDS Pass-Thru session can be open at a time.

After issuing the TDS Pass-Thru command, the user begins works directly under the remote application(s) to which he or she is connected, not under the initial TDS application. To return to the initial TDS application, the user must log off from the remote TDS application.

Figure 4-6 shows a TDS Pass-Thru (PT) session.

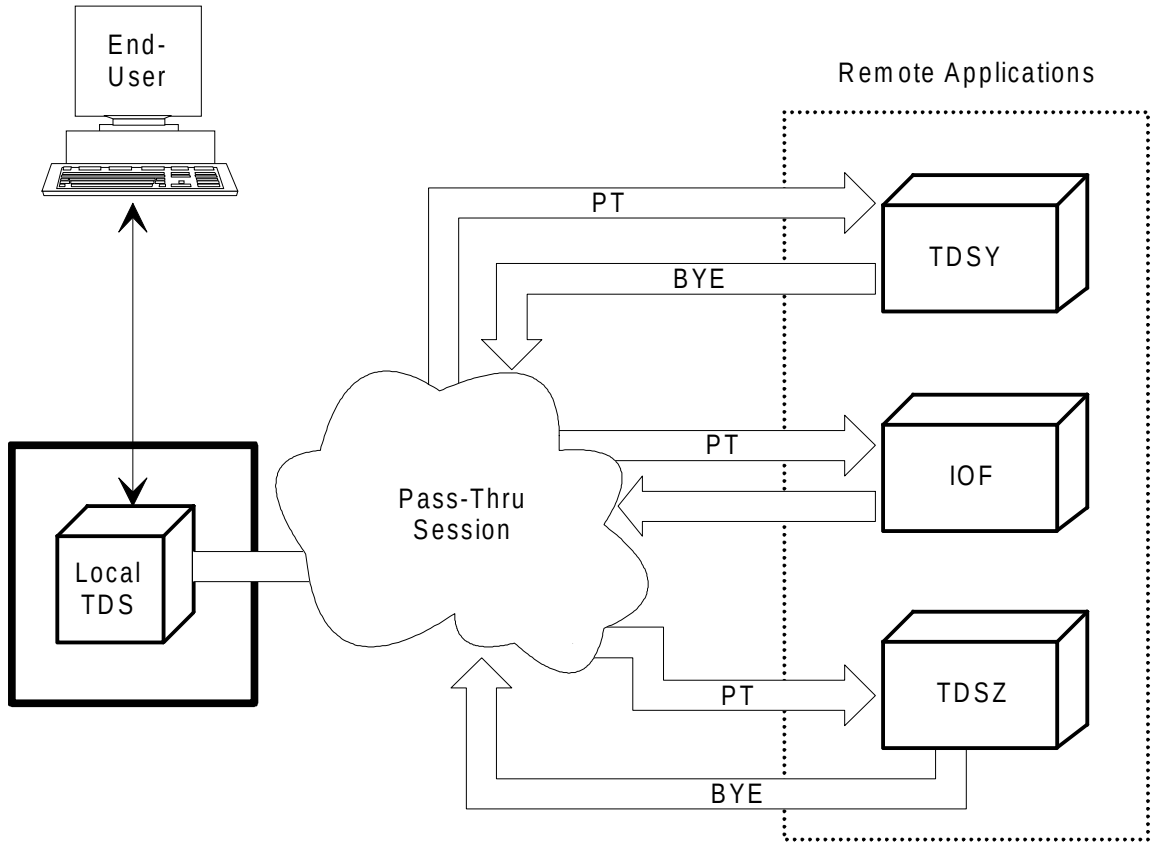


Figure 4-6. Using Pass-Thru to Connect to Remote Applications

4.2.3.3 The XCP1 Protocol

The Extended Communications Protocol (XCP) family of protocols that allow users to link to other applications for inquiry and modification. The applications do not have to run on GCOS 7, but can be on any operating system as long as XCP1 is supported.

XCP1 provides compatibility with IBM's LU0 (CICS) and LUP (IMS) protocol. The XCP1 protocol allows applications to exchange messages with these other applications. TDS implements XCP1 and can therefore communicate with other applications that support this protocol.

XCP1 allows data to travel back and forth between applications running on either local or remote systems. ("Remote" means either in the same room, or at another site). Cooperation takes place at application level and not at correspondent level. In other words, a correspondent can run several applications cooperating through the XCP1 protocol.

When TDS dialogs with an IBM-compatible application, TDS functions as the front-end application, and IBM as the back-end application. That is, a GCOS 7 transaction can start the IBM application, but the IBM application cannot start a GCOS 7 transaction.

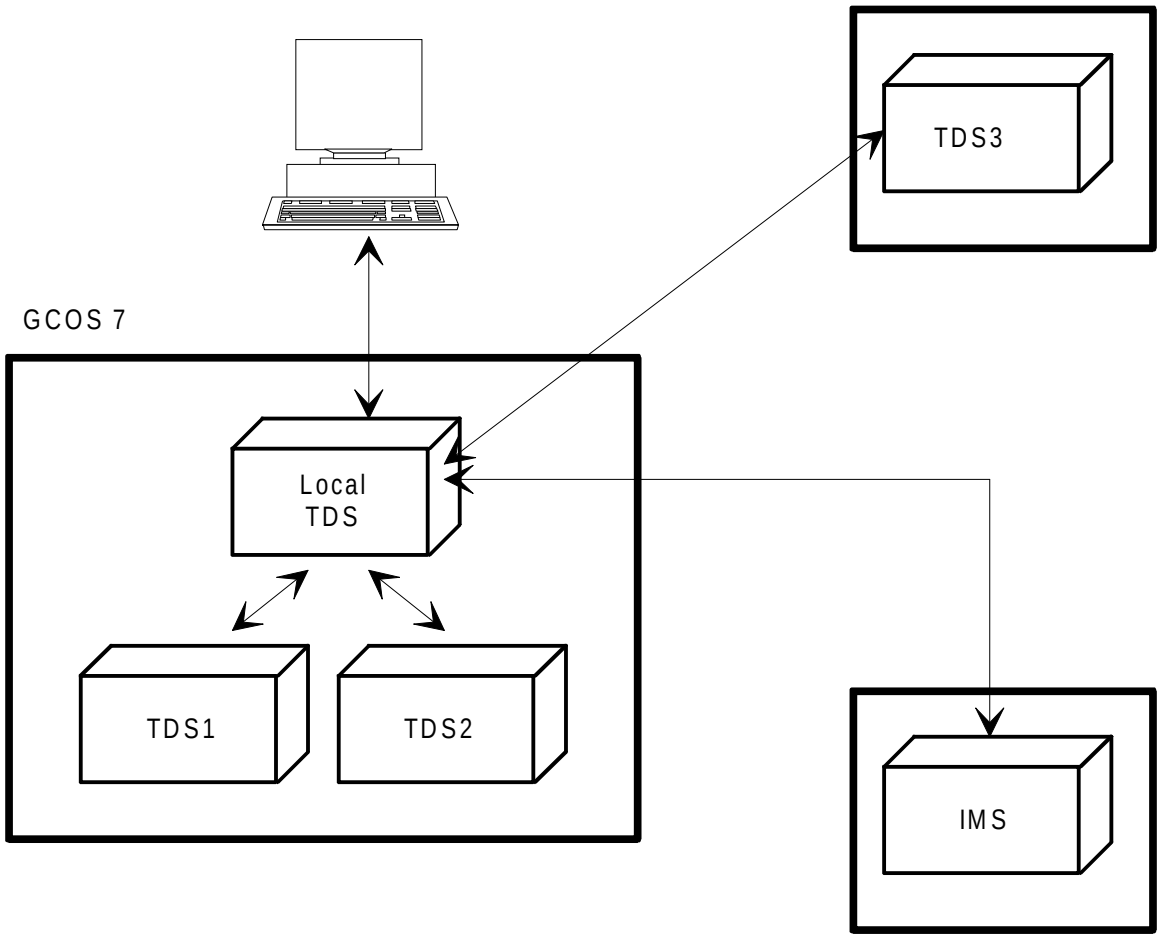


Figure 4-7. An XCP1 Communications Session

Figure 4-7 shows how a TDS application using the XCP1 protocol communicates with another application over the communications session. A transaction started by the end-user and executed in the user's local TDS application works with transactions in four different applications:

- TDS1
- TDS2
- TDS3 (on a remote site)
- an IMS application in an IBM environment.

When two correspondents dialog over an XCP1 session, one is the primary correspondent and the other is the secondary correspondent.

To use the XCP1 protocol in a TDS application, the TDS programmer must design transactions the XCP1 CALL statements. Then the TDS Administrator must modify the TDS environment during the preparation of the application. For more information on TDS and XCP1, see the *Transactional Intercommunication Using XCP1 Protocol User's Guide*.

4.2.3.4 The XCP2 Protocol

The XCP2 protocol, is part of the Extended Communications Protocol (XCP) family of protocols which allow application to application communication.

In the case of XCP2, this communication permits the management of distributed transactions and the exchange of messages.

The support by TDS of the XCP2 protocol enables a TDS application to dialog with the following:

- another TDS on the same system or another system,
- an IBM application using the IBM System Network Architecture (SNA) Logical Unit type 6.2,
- any platform where a "transactional like" subsystem using an XCP2 protocol exists (for example, a Bull DPX system with the CPI/C-OSI product).

Figure 4-8 shows an XCP2 Communications session.

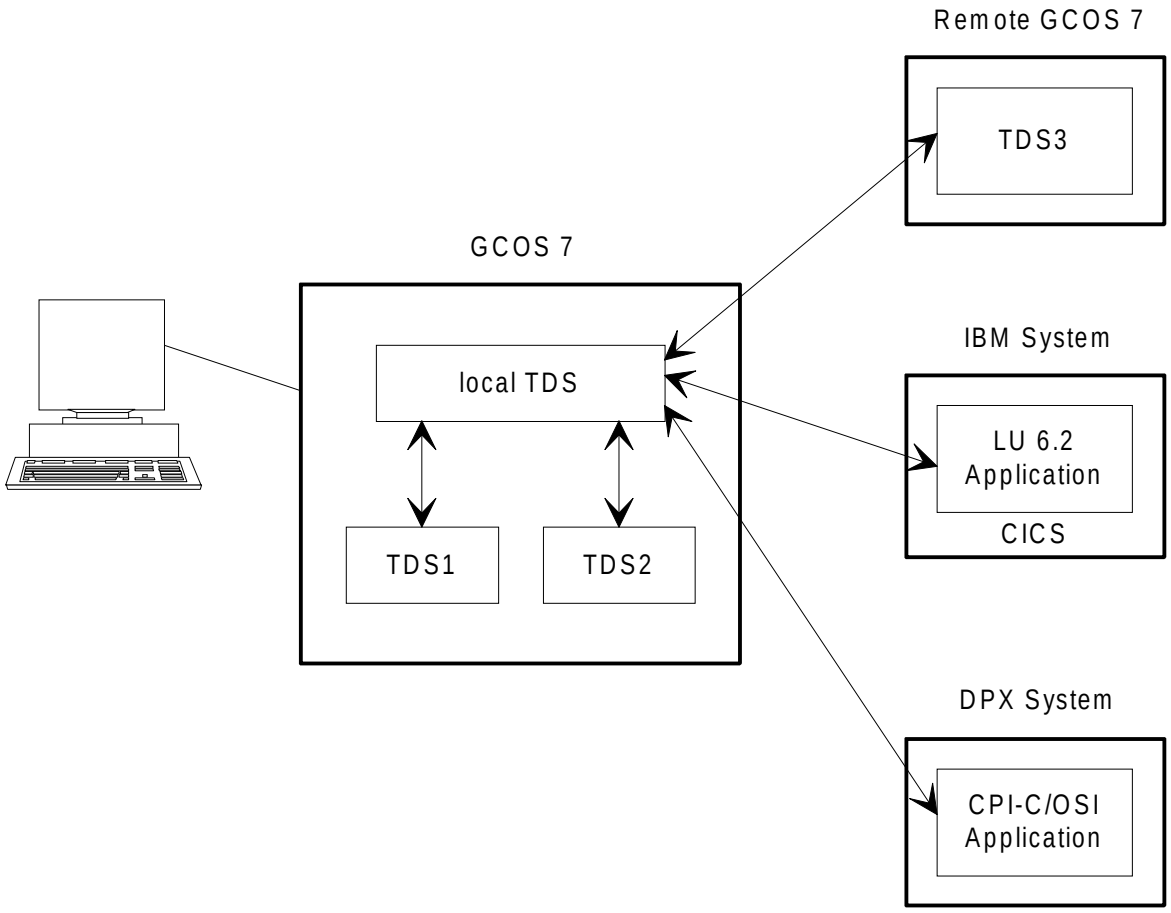


Figure 4-8. An XCP2 Communications Session

In the transactions, the TDS programmer uses either the X/Open CPI-C programming interface, or the Bull PPC-PI programming interface. The TDS Administrator has to set up a specific TDS environment for CPI-C/XCP2. For more information on XCP2, see the *CPI-C/XCP2 User's Guide*.

An interesting feature of XCP2 is that any of the partners (applications) can initiate a remote transaction.

4.3 INTERACTION WITH UNIX WORLD

2LTP (Two Level Transaction Processing):

It is possible to connect to a TDS application from an MS-DOS application or a UNIX workstation. TDS sees the application as a terminal. See the Affinity and OTM documentation.

CTP (Cooperative Transaction Protocol):

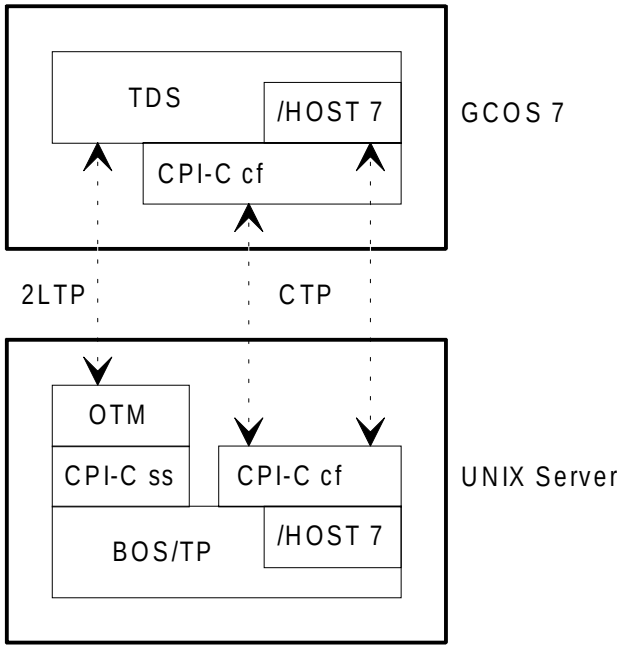
The CPI-C application programmatic interface ("API") supports the XCP2 protocol (at "confirm" level) and allows communication between a TDS application and a BOS/TP application. See the *CPI-C Programmer's Guide*.

HOST 7 is a function based on CPI-C which allows you to invoke a TDS transaction through the normal BOS/TP "API". This feature is available only after Technical Status 6152.

IMAGEWorks:

There is an "API" which enables a TDS transaction to manipulate an IMAGEWorks document. See the *TDS-IMAGEWorks Link User's Guide*.

Figure 4-9 shows TDS in the UNIX world.



CTP = Co-operative Transaction Processing
2LTP = Two Level Transaction Processing
cf = Confirm Set } Application Programmatic Interface
ss = Starter Set
OTM = Open Terminal Manager

Figure 4-9. TDS in the UNIX World

4.4 ACCESS AND ORGANIZATION OF DATA IN TDS

A TDS application can simultaneously access ORACLE data, IDS/II databases, and UFAS files. A TPR can update either an ORACLE database, or IDS/II and UFAS data.

The data used in a TDS application can be organized in any of these databases:

- IDS/II
- IQS
- ORACLE

4.4.1 IDS/II and TDS

Transactions within TDS can access IDS/II databases. The IDS/II areas have to be declared in the source of the TDS Generation. Schemas and sub-schemas can be used. Areas of IDS/II are UFAS controlled files and follow the usual integrity rules. See the paragraph on "TDS Controlled Files" in chapter 2 and the IDS/II documentation.

4.4.2 Using IQS with TDS

IQS queries can be executed in a TDS environment. Such queries can access IDS/II and UFAS-EXTENDED databases for which IQS schemas and/or views have been defined.

IQS queries can use object macros, forms, formats, and reports under TDS, just as under IOF. These can be loaded explicitly by the master operator for use by all users, or implicitly by IQS when used by a query (and thus are private to the user).

After the IQS queries are compiled, they are linked to create an object TPR. A transaction can consist of a mixture of COBOL TPRs and IQS queries. However, no data can be passed between TPRs and queries.

After the queries are written by the TDS Programmer, the TDS Administrator must declare IQS views, schemas, files, and areas in the TDS Generation Program. In addition, the administrator must specify the <tdsname>.GMEM file in the TDS preparation. For more information on using IQS with TDS, see the *IQS-V4/TDS User's Guide*.

4.4.3 ORACLE and TDS

ORACLE databases can be used with TDS. Several TDS applications can share an ORACLE database, and can access the database simultaneously. The database may reside on the same (local), system or on a remote GCOS 7 system

In ORACLE/TDS, the TDS programmer writes COBOL TPRs which contain embedded SQL statements that access ORACLE databases. TPRs written in C Language are not supported.

The ORACLE server runs independently of the TDS application. ORACLE databases have their own log file and rollback segments, and all maintenance is handled by ORACLE tools.

The TDS Administrator must declare the database in the TDS Generation Program. For information on using ORACLE databases with TDS, see *ORACLE-V6/TDS User's Guide*.

4.5 PRODUCTS USED WITH TDS TO PROVIDE ADDED INTEGRITY & SECURITY

Several GCOS 7 products, offered as options, can be used in a TDS environment to improve system security. Using this products minimizes the impacts of hardware incidents, software failures, or unauthorized access attempts. These GCOS 7 options are:

- TDS-HA
- Mirror Disks
- RDDF7
- SECUR'ACCESS.

4.5.1 TDS-HA

The purpose of High Availability, or HA, is to increase the time that applications are available, and reduce the impact of incidents on applications. TDS-HA is the GCOS 7 product that provides TDS users with this continuous service. TDS-HA increases the availability of TDS applications when hardware or software incidents occur. TDS-HA can be implemented on one or more TDS applications on a site (with 2 systems); thus TDS-HA can co-exist with non-HA applications such as TDS or IOF. TDS-HA can use ORACLE and UFAS/IDS/II databases.

The TDS-HA product:

- reduces restart time after a crash
- restarts all transactions from the last commitment point
- provides a backup system in case of incidents
- performs system recovery and reconnection simultaneously
- reconnects all users connected before the incident
- is based on coupled systems
- supports Mirror Disks
- recovers journalization files.

During operation, a TDS application is tracked by a back-up TDS operating on a separate processor. If a failure occurs, the system switches processing from the active to the back-up TDS. The back-up TDS takes over the session and continues normal processing.

For more information on high availability, see the *High Availability Concepts* and the *High Availability Administrator's Guide*.

4.5.2 Mirror Disks

Mirror Disks is a GCOS 7 product that offers high availability of data stored on disks. By increasing the availability of disk based data, Mirror Disks provides continuous service of data to applications.

Mirror Disks can provide this availability by maintaining two copies of data; each copy is stored on a separate disks. Each disk has a disk which is its mirror image, or copy. Every write operation generates two write, one to each copy. If one copy is lost, the other copy is still available. When a failure that effects disks or controllers occur, Mirror Disks allows application to continue normal processing using the other copy.

Mirror Disks can be used on mono or coupled systems, and on systems running TDS-HA. This option can be applied to selected disks, or all disks on a system. Mirror Disks requires additional disks. Every disk declared as a Mirror Disk must have an identical back-up copy.

For more information on data availability, see the *Mirror Disks User's Guide*.

4.5.3 Protecting Databases with RDDF7

The Remote Database Duplication Facility, RDDF7, allows a backup machine to maintain a copy of ORACLE databases on GCOS 7. RDDF7 can be used with TDS, IOF, or batch transactions.

While a TDS application is running and the database is updated, RDDF7 transfers After Journal images to the backup machine. The after images are applied to a duplicate copy of the database. The duplication of updates is deferred on the remote site. The RDDF7 software ensure the consistency of data between the two sites.

If the main machine fails, TDS can be started up on the backup machine using the duplicate database. When the main machine is restarted, the after images can be transferred to the restored system.

For more information on improving the availability of databases, see the *RDDF7/RTO User's Guide*.

4.5.4 SECUR'ACCESS

SECUR'ACCESS is a logical access control software that can be used to protect a transactional application. This software identifies, authenticates, and then authorizes user to connect or re-connect to the application. These controls can also be programmed to verify user access at any moment during a work session.

For more information about using SECUR'ACCESS with TDS, see these documents:

- *SECUR'ACCESS Security Administrator's Guide*
- *SECUR'ACCESS Delegate Administrator's Guide*
- *SECUR'ACCESS User's Guide*
- *SECUR'ACCESS Programming and Implementation Guide.*

Glossary

Any words appearing in bold in the text of a definition are explained in the glossary.

Access rights:

A list of codes declared for each GCOS 7 project which allow (or prevent) users to access disks, applications, environments, or stations. These access rights are declared at the *site level* and should not be confused with TDS authority codes which concern the access of a TDS transaction.

Administrator:

see TDS Administrator.

After image:

The copy of an update to a file, recorded on an After Journal, used to recover files in the case of a **rollfoward**.

After Journal:

A GCOS 7 function that protects against hardware failure. This journal is used to store updates (after images) to be written to a file or a database.

Authority codes:

A list of codes which allow or prevent users from executing a given transaction *within TDS*. As in other GCOS 7 applications, all users belong to one or more projects declared in the site catalog. Authority codes are also declared in the site catalog and allocated to projects. Thus, through the authority codes associated with the projects, users have access to transactions.

Batch interface:

This is an interface which allows you to establish a session and dialog between a batch program and a TDS application.

It can be used as a program debugging facility. A batch program simulates a terminal, allowing the program to be logged on as a terminal to send and receive messages.

Before image:

A copy of the file containing the data in the state prior to an update. Before images are recorded sequentially in the Before Journal, and are used in case of a rollback.

Before Journal:

A GCOS 7 function that protects against software failure. When a record is processed, a copy of the original record is written (as a Before images) to this journal.

Commitment:

The action taken periodically by TDS after the normal end of processing. It is a point within a transaction where:

- TDS saves all processing of the transaction up to that instant: the processing is definite and cannot be lost or cancelled
- the transaction may be restarted after a software or hardware failure
- all resources allocated to the transaction are released
- resource conflicts can be resolved.

Commitment unit:

All of the processing completed since the last commitment (or from the beginning of the transaction if it is the first commitment unit) to the current commitment (or to the end of the transaction if it is the last commitment unit). This is sometimes referred to as a CU.

Conversation:

A conversation is the interval between two **exchanges** of the same **transaction**. That is, the time between the transmission of the transaction's first input message and the reception that transaction's next input message of a given user.

A conversation has a different meaning in an XCP2 context.

Correspondent:

An application, a user logged on at a terminal, or a "virtual" correspondent that interacts with a TDS application. A correspondent uses a protocol to dialog with a TDS. It can be a TM (PT, Batch Interface), XCP1, XCP2 or virtual session.

Coupled system:

Two DPS 7000s using the Coupled System product and sharing one or more disk volumes.

CRNETGEN:

The facility used to generate the network and also to define correspondents.

Deadlock:

A conflict between one or more commitment units for system resources.

Deferred Updates:

An updating method in which the modification is not made to the file until the **commitment unit** is completed. If a transaction aborts before the end of the commitment unit, the update is not made. The principle of deferred updates is used by the **rollforward** utility. This mechanism improves performance but it needs memory resources (UFAS buffers).

Dirty read:

The access and reading of a file while it is being updated. The information from a dirty read is or is not correct depending on whether the read was before or after the commitment point of the TPR updating the file.

Dummy correspondent:

A "virtual" correspondent, that is, a correspondent that is not real but seen only by TDS. It allows parallel processing to be coded through the spawn mechanism.

End-user:

A user who interacts with a TDS application via a terminal. In terms of this document, an end-user is anyone who logs on to TDS to run transactions. An end-user is not concerned with the development or administration of the TDS application.

Exchange:

A sequence which includes the input message, the processing by the application, and the output message. A TPR results in one exchange, but an exchange may be dispersed over any number of TPRs. A transaction may include several exchanges.

Exclusive Read:

A type of access to a file in which the record is locked while being read. Only one TPR can access the record at a time; access to the file is "exclusive".

FORMS:

An on-screen form with blanks, or fields, to be filled in by the user.

GAC-EXTENDED:

A file sharing facility that allows several concurrent users in various processing environments to read and write to the same file.

GCL:

GCOS 7 Command Language.

GTWriter:

A GCOS 7 facility that allows information to be asynchronously spooled and printed on the specified hardcopy terminal.

High Availability:

A GCOS 7 software product designed to increase the availability of applications. This is also referred to as HA.

IDS/II:

A hierarchical networked database fully integrated with TDS. It is based on UFAS files.

IOF:

Interactive Operation Facility. A GCOS 7 facility that allows users to access information (files, databases, applications, programs, or facilities) and submit requests.

JCL:

Job Control Language.

Journalization:

The selection of one or both of the GCOS 7 journals (Before and/or After) to protect files.

Library:

A file that can contain objects of various types known as **members**. A library is functionally divided into subfiles.

Linked unit:

A COBOL TPR compile unit that has been linked. The Static Linker forms linked units and stores them in a sharable module on a SM library.

Mailbox:

A string of 8 alphanumeric characters external address that identifies an endpoint, either a user, correspondent, terminal, or printer.

The Master mailbox identifies the Master Operator.

Mapping:

The process of allocating a process or a task to a transaction.

Master commands:

Commands available to the Master Operator, used to control TDS.

Master file:

A file that contains the user data of the application. This file can be thought of as part of the data base. A master file is frequently used and always kept up-to-date.

Master mailbox:

A mailbox name (a string of 8 alphanumeric characters) cataloged as an application in the site catalog with at least authority code 0. The Master mailbox identifies the Master Operator.

Master operator:

The correspondent in charge of managing the TDS session. The Master Operator must complete the start-up dialog with TDS before other users can log on.

Member:

A subfile of a **library**.

Message-id:

The name used to identify a transaction. When a user types in the MESSAGE or selects it from a menu, the transaction is executed.

MTGEN:

The V3/V5 JCL utility used to generate the TDS application. The corresponding V6 utility is TP7GEN.

MTPREP:

The V3/V5 JCL utility used to prepare the TDS application by allocating file space for TDS files. The corresponding V6 utility is TP7PREP.

NETGEN:

see CRNETGEN.

Non-concurrency:

The prevention of activating simultaneously, tasks belonging to different transactions. A transaction may be declared as non-concurrent with other transactions (or with itself) at TDS generation.

Non-controlled file:

A UFAS-EXTENDED (or UFAS) file which is not affected by the access control provided by TDS.

Off-line files:

Files used to generate, compile and link TDS. These files are not required at run-time.

On-line files:

Files used to process transactions. These files need to be available (or on-line) at run time.

ORACLE:

A Relational Database Management System. ORACLE can be used with TDS, but is managed by independent software.

Pool:

A set of parallel sessions having common characteristics. They may have the same synchronization level or security options. A pool must contain at least one session. A maximum number of sessions is defined for each pool.

Rollback:

The process of restoring a file to an earlier state by discarding all the work done since the last commitment point. The Before image replaces the updated record.

Rollforward:

A process of recovering files in the event of a TDS abort or system crash. The most recent state of the files is achieved by replacing records with the valid modifications saved as After images.

Session:

A logical connection that allows a succession of transmissions between two-endpoints, called **mailboxes**.

Sharable Module library:

A library where compiled and linked TPRs are stored.

Shared read:

A type of access to a file in which the record can be read by multiple TPRs. A shared read is only used to read the file. If the TPR wishes to write to the file, it must wait until it has an **exclusive read**.

Simultaneity:

The level of multitasking permitted in a TDS application. That is the maximum number of tasks that can be executed at any one time. Simultaneity is specified at TDS generation.

Spawning:

The activation of a transaction from within another transaction.

Special-Purpose transactions:

Transactions written by the programmer and called by TDS in the case of a TDS startup or shutdown, user log-on or log-off, when TDS is disconnected or restarted, or when a Break is sent.

Statistical read:

A read of a file in which transactions can access data (for a read, but not a write) that is being updated by other commitments.

STDS:

The Source TDS, or program used to generate the TDS application.

TDS Monitor:

The software that controls the execution of application programs during transaction processing. This is sometimes referred to as the TDS Executive.

TDS-controlled file:

A UFAS or UFAS-EXTENDED relative or indexed sequential file, or an IDS/II database, controlled by TDS during generation. This type of file has concurrent access control.

TDS Administrator:

The person who performs the administrative functions in TDS including the configuration, optimization, and implementation of a TDS application.

TDSGEN:

The TDS Generation Program is used to define the environment in which transaction processing applications are to be run.

Terminal Manager:

A protocol used to exchange data between a TDS application and a terminal.

Terminal Operator:

see End-user.

TPR:

Transaction Processing Routine. A program unit written in either COBOL, C language, or GPL by the TDS programmer, and designed to execute in the TDS environment. One or more of these program units form a transaction. TPRs are stored in sharable modules.

TP7GEN:

The V6 JCL utility used to generate the TDS application.

TP7PREP:

The V6 JCL utility used to prepare the TDS application by allocating file space for TDS files.

Transaction:

A unit of processing defined by the TDS programmer. Each transaction consists of one or more Transaction Program Routines (TPRs). An example of a transaction is the updating of a customer's bank account after a payment is made using a credit card. The transaction includes accessing the customer file, the dialog with the Operator, and the modification to the file.

User:

see End-user.

User Journal:

A private journal that allows a user(or a programmer) to record messages, statistics, or any user information.

Windowing:

The ability to overlay or superimpose one *form* or window on another.

XCP1:

(eXtended Cooperative Protocol Level 1) A standard application-to-application communications protocol.

XCP2:

(eXtended Cooperative Protocol Level 2) A standard application-to-application communications protocol.

Index

F

G

H

I

J

L

M

N

O

P

R

S

T

U

W

X

Technical publication remarks form

Title :	DPS7000/XTA NOVASCALE 7000 TDS Concepts

Reference N° :	47 A2 26UT 00	**Date:**	**March 1993**

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be promptly investigated by qualified technical personnel and action will be taken as required.
If you require a written reply, please include your complete mailing address below.

NAME : _____ Date : _____

COMPANY : _____

ADDRESS : _____

_____

Please give this technical publication remarks form to your BULL representative or mail to:

Bull - Documentation Dept.
1 Rue de Provence
BP 208
38432 ECHIROLLES CEDEX
FRANCE
info@frec.bull.fr

Technical publications ordering form

To order additional publications, please fill in a copy of this form and send it via mail to:

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

Phone: +33 (0) 2 41 73 72 66
FAX: +33 (0) 2 41 73 70 66
E-Mail: srv.Duplicopy@bull.net

CEDOC Reference #	Designation	Qty
__ __ ____ _ [__]		
__ __ ____ _ [__]		
__ __ ____ _ [__]		
__ __ ____ _ [__]		
__ __ ____ _ [__]		
__ __ ____ _ [__]		
__ __ ____ _ [__]		
__ __ ____ _ [__]		
__ __ ____ _ [__]		
__ __ ____ _ [__]		
__ __ ____ _ [__]		
__ __ ____ _ [__]		

[_ _] : The latest revision will be provided if no revision number is given.

NAME: ______________________________ Date: __________

COMPANY: ______________________________

ADDRESS: ______________________________

____

PHONE: ______________________ FAX: ______________

E-MAIL: ______________________________

For Bull Subsidiaries:

Identification: ______________________________

For Bull Affiliated Customers:

Customer Code: ______________________________

For Bull Internal Customers:

Budgetary Section: ______________________________

For Others: Please ask your Bull representative.

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

REFERENCE
47 A2 26UT 00