TDS COBOL

Programmer's Guide

DPS7000/XTA
NOVASCALE 7000

Transaction Processing: General

REFERENCE
47 A2 33UT 08

DPS7000/XTA NOVASCALE 7000

TDS COBOL

Programmer's Guide

Transaction Processing: General

February 2005

BULL CEDOC

357 AVENUE PATTON

B.P.20845

49008 ANGERS CEDEX 01

FRANCE

REFERENCE

47 A2 33UT 08

Trademarks and Acknowledgements

Preface

Scope and Objectives	This manual describes how to program transactional applications running under TDS.
Intended Readers	The information is intended for the systems analyst or programmer for developing and implementing a TDS application. It is assumed that TDS programmers have a good knowledge of COBOL programming and are familiar with the basic concepts of transactional processing.

Structure

Chapter 1	describes transaction processing, how data is managed and passed to the transaction, and the system interfaces that allow a TDS application to function.
Chapter 2	describes how to program a transaction in COBOL.
Chapter 3	is a guide to the TDS procedures and verbs used for dealing with session management.
Chapter 4	is a guide to the TDS procedures and verbs used for dealing with TPR control.
Chapter 5	is a guide to the TDS procedures and verbs used for dealing with spawn handling.
Chapter 6	is a guide to the TDS procedures and verbs used for dealing with correspondent pools.
Chapter 7	is a guide to the TDS procedures used for dealing with the terminal adapter.
Chapter 8	is a guide to the TDS procedures for handling COMMON-STORAGE.
Chapter 9	is a guide to the TDS procedures and verbs used for file access, including concurrency, commitment, and rollback handling.
Chapter 10	is a guide to the TDS procedures for handling FORMS.

Chapter 11	is a guide to the TDS procedures for dealing with GTWriter.
Chapter 12	describes the special-purpose transactions, and the transaction initialization routine.
Chapter 13	describes how a TDS application is designed, implemented, optimized, and debugged.
Chapter 14	deals with the terminal commands available to users of the TDS application.

The section on IMAGEWorks has been deleted. For information about IMAGEWorks, see the *TDS-IMAGEWorks Link User's Guide*.

The appendices give details and references to all topics mentioned in the manual.

Bibliography

The following publications give more information about specific topics in the set of TDS manuals.

This manual is one of a set of TDS manuals. The other manuals in this set are:

TDS Concepts	*47 A2 01UT*
TDS C Programmer's Guide	*47 A2 07UT*
High Availability Concepts	*47 A2 22UT*
High Availability Administrator's Guide	*47 A2 23UT*
TDS-IMAGEWorks Link User's Guide	*47 A2 25UT*
TDS-IMAGEWorks DPX20 User's Guide	*47 A2 31UT*
TDS Administrator's Guide	*47 A2 32UT*

Prerequisite material is contained in the *TDS Concepts Manual*; related material is contained in the *TDS Administrator's Guide*.

For using TDS with ORACLE-V6:

ORACLE-V6/TDS User's Guide	*47 A2 05UR*
ORACLE-V6/TDS-HA User's Guide	*47 A2 07UR*

For using TDS with ORACLE7:

ORACLE7/TDS User's Guide	*47 A2 14UR*
ORACLE7/TDS-HA User's Guide	*47 A2 16UR*

For generating the DSA network:

DNS V4 System Generation	*39 A2 22DN*
DNS V4 NGL Reference Manual	*39 A2 23DN*
CNS7 A2 NOI Operator Guide	*39 A2 34DN*
CNS 7 A1 NOI Operator Guide	*39 A2 41DM*

For cataloging users, including correspondents, and TDS authority codes using the MAINTAIN_CATALOG utility:

Catalog Management User's Guide *47 A2 35UF*
GCOS 7 System Administrator's Manual *47 A2 54US*

For creating and managing FORMS using the MAINTAIN_FORM utility:

Forms User's Guide...*47 A2 15UJ*

For producing reports:

Generalized Terminal Writer User's Guide *47 A2 55UU*

For COBOL syntax and use:

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

For defining XCP2 correspondents:

CPI-C/XCP2 User's Guide... *47 A2 14UT*

For defining XCP1 correspondents:

Transactional Intercommunication using the XCP1
Protocol User's Guide *47 A2 11UT*

For File Access and Data Management:

Full IDS/II Reference Manual 1 .... *47 A2 05UD*
Full IDS/II Reference Manual 2 .... *47 A2 06UD*
Full IDS/II User's Guide... *47 A2 07UD*
IDS/II Reference Manual...*47 A2 11UD*
IDS/II Administrator's Guide... *47 A2 13UD*
Database Reorganization (DBREORG) User's Guide *47 A2 15UD*
UFAS-EXTENDED User's Guide *47 A2 04UF*
Data Security Facilities User's Guide...*47 A2 09UF*

For main console operator commands and DPS 7000 Network Generation:

GCOS 7 Network Overview and Concepts..*47 A2 92UC*
GCOS 7 Network Generation..*47 A2 93UC*
GCOS 7 Network User Guide...*47 A2 94UC*
GCOS 7 System Operator's Guide.. *47 A2 53US*

For information on installing and optimizing the system:

System Behavior Reporter User's Guide *47 A2 03US*
TILS User's Guide *47 A2 04US*
GCOS 7 – V8 System Installation Configuration and Updating Guide .. *47 A2 19US*
GCOS 7 – V9 System Installation Configuration and Updating Guide .. *47 A2 23US*
GCOS 7 System Administrator's Manual *47 A2 54US*

For concurrency control:

GAC-EXTENDED User's Guide *47 A2 12UF*

For file recovery procedures and journal usage:

File Recovery Facilities User's Guide.. *47 A2 37UF*

For GCL commands:

IOF Terminal User's Reference Manual:
Part 1.. *47 A2 38UJ*
Part 2.. *47 A2 39UJ*
Part 3.. *47 A2 40UJ*

For JCL statements:

JCL Reference Manual *47 A2 11UJ*
JCL User's Guide... *47 A2 12UJ*

For status values:

Messages and Return Codes Directory *47 A2 10UJ*

For using IQS under TDS:

IQS/TDS User's Guide.. *47 A2 81UR*

For information on PCF commands:

GCOS 7 Program Checkout Facility User's Guide *47 A2 15UP*
For Migrating between Releases:

GCOS 7 Evolution Guide *47 A2 20UG*

For Using DOF 7

DOF7-PO User's Guide *47 A2 80UC*
Structured Records (OMH Format) Part 1 - Commands......................... *47 A2 81UC*
Structured Records (OMH Format) Part 2 - Messages........................... *47 A2 82UC*
Structured Records (DSAC Format) Part 1 - Commands and Messages *47 A2 83UC*
DOF7-SM User's Guide *47 A2 84UC*
Structured Records (DSAC Format) Part 2 - Unsolicited Messages....... *47 A2 85UC*

Syntax Notation

UPPERCASE WORDS	Keywords whose presence in a format indicates they may be used in that context. Any such words underlined must be used whenever the relevant option is used. Those which are not underlined may be omitted.
lowercase words	These represent variable items to be supplied by the programmer.
[ ]	square brackets indicate an optional clause
{item}	
{item}	braces indicate a choice
{item}	

Table of Contents

2. Programming the Transaction

Table of Graphics

1. Transaction Processing

1.1 Overview of TDS

This chapter presents a brief overview of the GCOS 7 Transaction Driven Subsystem (TDS). A clear description of the concepts underlying TDS is provided in the *TDS Concepts Manual*. The concepts presented are essential reading. The purpose of this chapter is to show the programmer how useful these concepts are in preparation for programming transactional applications. A system is said to be transaction driven when its operation depends primarily on the transactions that are available for processing. User application programs designed to process transactions under TDS are called Transaction Processing Routines (TPRs). These are coded in a high-level language by the programmer.

The System Administrator describes the environment in which transactions execute at TDS generation. The TDS Generation process is called TDSGEN. This process describes a TDS for a particular application; the TPRs written by the programmer operate in this environment provided by the generated TDS. In general the programmer does not have to be concerned about TDSGEN. It is enough to know that the generation program is generated using COBOL-like statements and entries defining the type and number of transactions and terminals. For the syntax and implementation of TDSGEN, see the *TDS Administrator's Guide*.

The subject of this manual is how to program TPRs.

1.1.1 Initiating a TDS Transaction

After the TDS Administrator has set up the programming environment by performing the steps in TDSGEN and started the TDS application TDSGEN produces, the ordinary users can log onto TDS. To start a transaction, the terminal operator keys in the transaction identifier on the terminal screen. The transaction identifier (maximum eight characters long) is declared at TDSGEN. As each transaction is completed, the user may begin another.

1.1.2 How Do TPRs Relate to TDS?

TDS supervises the execution of transactions as seen in Figure 1-1.

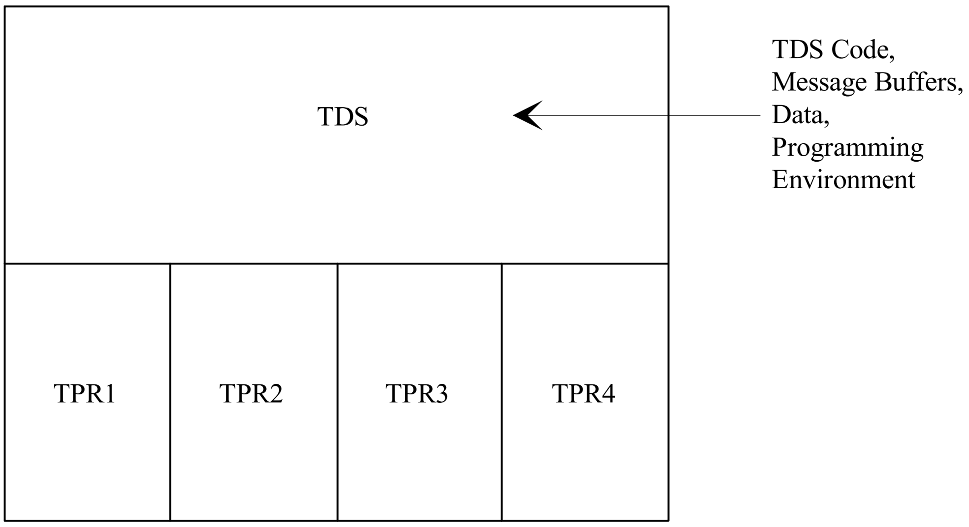


Figure 1-1. Application Programs (TPRs) Managed by TDS

Each transaction is processed by a set of TPRs. Any TPR can be shared by several transactions. A transaction may consist of only a single TPR.

TDS, as seen by the programmer, is a collection of commands in a TPR.

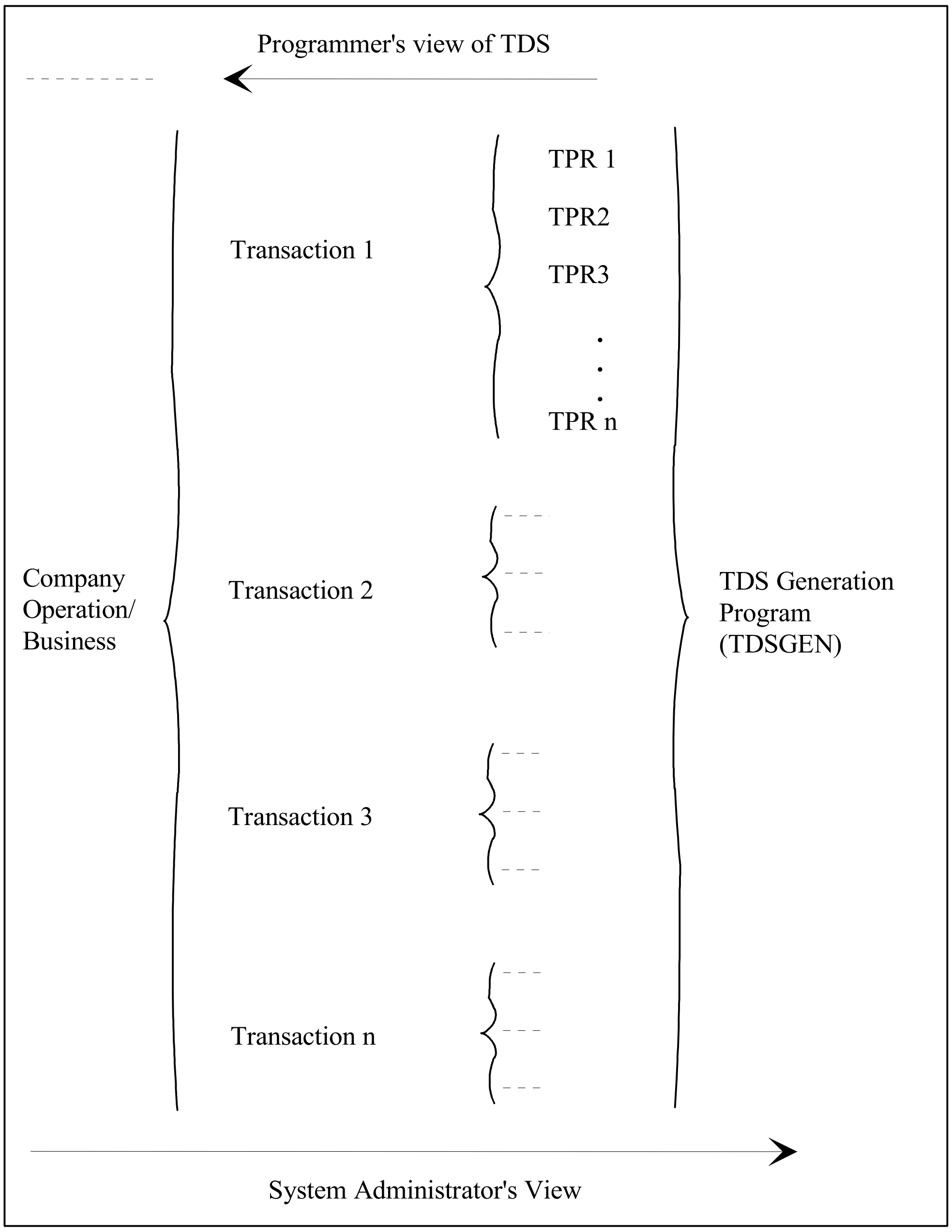


Figure 1-2. The Programmer's View of TDS

The TDS Administrator views TDS from the viewpoint of TDSGEN, whereas the TDS Programmer views TDS from the viewpoint of the terminal operator.

1.1.3 How Do TPRs Function?

TDS uses a message from a terminal, which consists of a transaction identifier and user parameters, to start TPR1. TPR1 is the first TPR in the sequence whose name is given in TDSGEN. It is shown in Figure 1-3, below. When TPR1 terminates, TDS passes control to TPR2 and so on until the final TPR terminates, at which point the transaction is complete. A TPR can be recursive, which means that can pass control to itself.

It is not up to the TPR to call the next one. It simply names the TPR that is to follow. TDS starts it as soon as the necessary conditions are satisfied and resources become available.

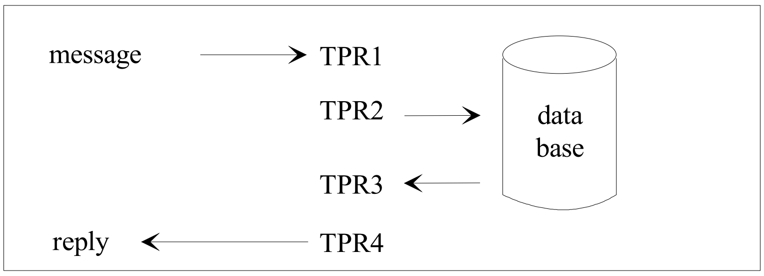


Figure 1-3. Transaction Processed by a Sequence of TPRs

1.1.4 TPRs and Other Programs

A TPR can call subroutines. A transaction might, for example, be processed by a sequence of TPRs as shown in Figure 1-4.

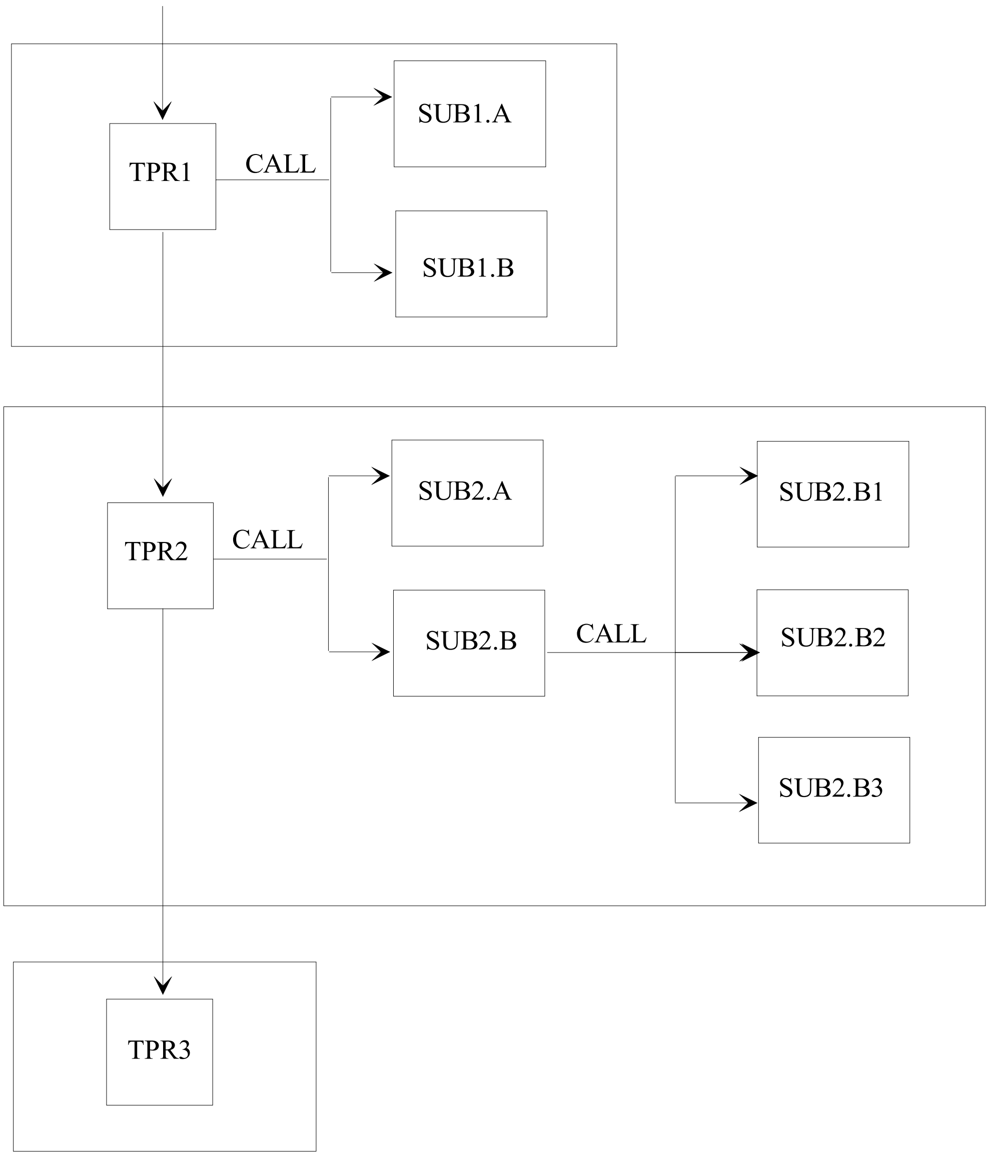


Figure 1-4. TPRs Composed of Calling and Called Subroutines

TPR1 is started by TDS. It performs its processing with the aid of two called subroutines, SUB1-A and SUB1-B and terminates, giving TPR2 as the next TPR. TDS then activates TPR2, which in turn calls other subroutines.

1.1.5 TPRs Executing Simultaneously

DS can handle more than one TPR at the same time. For information on the maximum number of the simultaneous TPRs (the so-called "simultaneity level"), see the *TDS Administrator's Guide*.

Figure 1-5 shows a TDS with a simultaneity level of 3. There are five TPRs, but only three are ever executing simultaneously. The execution of several simultaneous TPRs is called concurrency.

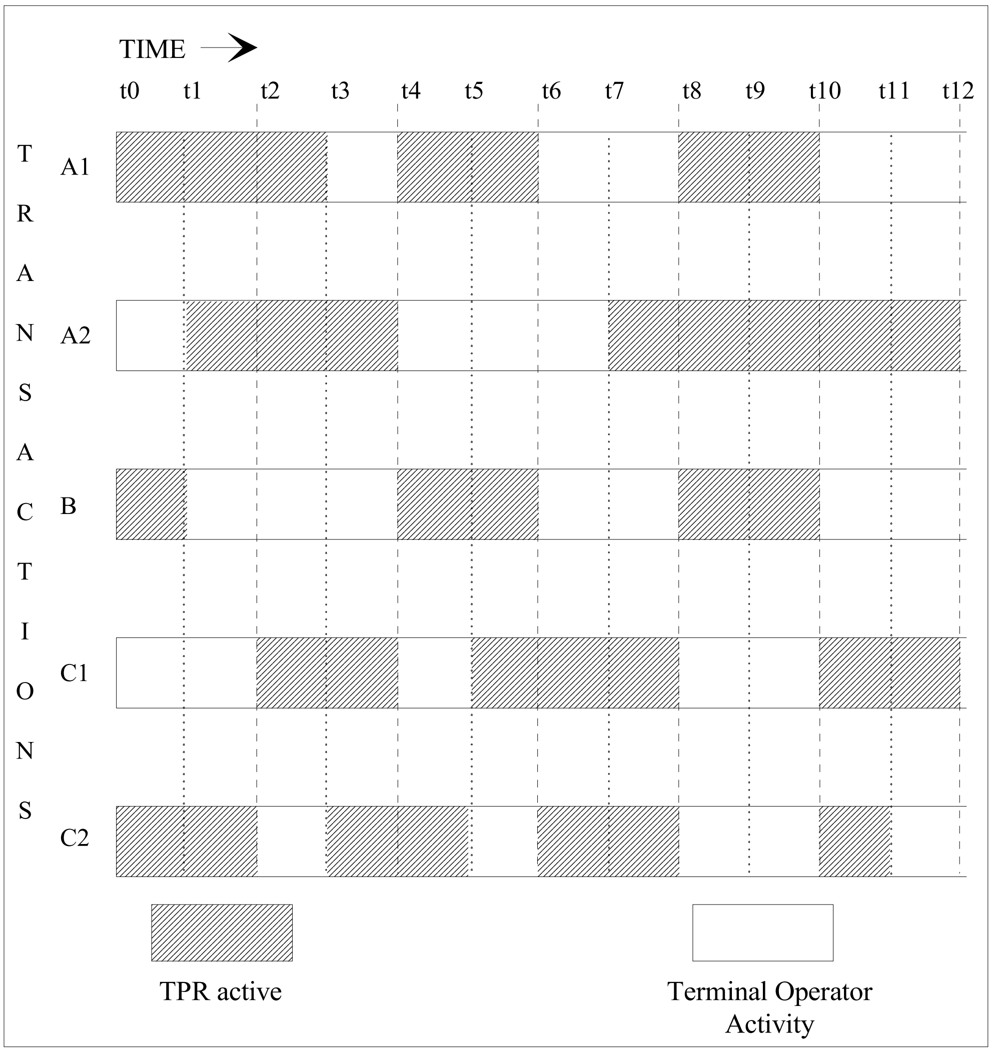


Figure 1-5. Simultaneous TPR Processing

In this example 3 levels of simultaneity are shown, that is, at any time, a maximum of 3 TPRs can simultaneously (concurrently) execute. At time (t0) the 3 transactions A1, B, and C2 execute in simultaneity. B stops executing at time (t1) at which point A2 starts executing. A1 and A2 are 2 separate occurrences of the same transaction A. C1 and C2 are 2 separate occurrences of the same transaction C. A1, A2, C1 and C2 execute as if they are all separate transactions.

The same transaction can be activated several times. Even if more than one transaction requires the same TPR at the same time, only one copy of this TPR is present in memory.

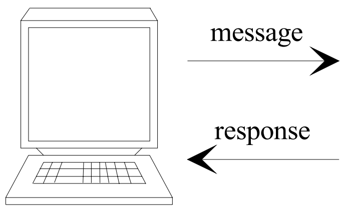
From this example, it follows that some kind of concurrency control mechanism is needed so that transactions do not interfere with each other. This mechanism is described later in this chapter in the section on Files.

1.1.6 Transaction Types

Three types of transactions are defined at TDSGEN:

1. "Update" transactions, containing updates to files/databases
2. FOR INQUIRY
3. FOR DEBUG

1. An "update" transaction is one in which the TPR will modify the database. The modification may be to replace or supplement existing information, or it may require the addition or deletion of whole elements.
2. A FOR INQUIRY transaction only causes information to be retrieved on the terminal.

   A FOR INQUIRY transaction does not modify the database and does not have conversations. There is only 1 exchange with the terminal operator.
3. The FOR DEBUG transactions are the transactions being tested for which file modifications are simulated. The FOR DEBUG option allows transactions to modify files temporarily. However, once the commitment unit terminates, the files will be rolled back to their original unmodified state. This type of transaction is described more fully in the section Debugging in TDSGEN in Chapter 13.

1.1.7 Elements of a Transaction

As explained in the *TDS Concepts Manual*, an exchange is the sequence of input request, transactional processing, and output response.

A transaction comprises:

EXCHANGES	- of messages
COMMITMENT UNITS	- for resource management

In general it is recommended that a COMMITMENT UNIT correspond to an EXCHANGE. This is the default value (IMPLICIT COMMITMENT in TDSGEN). However, commitment units and exchanges can interact. For example,

```
                  { Exchange-1
                  {      .
Commitment Unit   {      .
                  {      .
                  { Exchange-n
```

or

```
                  { Commitment Unit-1
                  {      .
Exchange          {      .
                  {      .
                  { Commitment Unit-n
```

Messages can be displayed in two formats:

message format:	"line mode", or "format mode"
conversation level:	"lines" or "forms".

In line mode, the terminal operator is prompted to enter a piece of information on a line. Once the terminal operator has filled up the line, he/she fills up the next line, and so on. Normally a piece of information corresponds to a line, but you might enter two or more pieces of information on a line as shown in Example 1 below (details about line mode are given in the section on Screen Displays later in this chapter).

In format mode, the terminal operator is prompted to enter pieces of information on certain parts of the screen and the TDS application displays information through a formatted menu (details about format mode are also given in the section on Screen Displays later in this chapter).

In line mode, the terminal operator works on a line by line basis, whereas in format mode the TDS application has complete control over the appearance of a form and can guide the terminal operator to enter the required input.

In line mode, the "think times" are interleaved, which is to say they are prompted. In format mode, the "think times" are accumulated until the terminal operator has filled in the entire form, with all the information using only one prompt.

Remember that a conversation is the phase between two exchanges of the same transaction, that is, the time between the transmission of the response from a TPR and the reception of the next input message from the terminal operator.
The "think-time" is the time it takes the user to enter a new message.

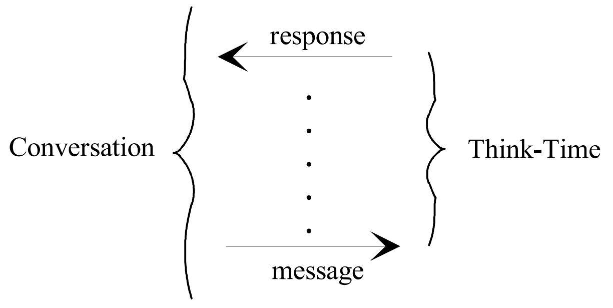


Figure 1-6. Conversations and Think Times

Line Mode: 1 exchange per element.

EXAMPLE 1:

In each exchange, one line is treated.

This is what appears before the terminal operator enters a piece of information.

```
ITEM  No.
```

This is what appears after the terminal operator enters pieces of information.

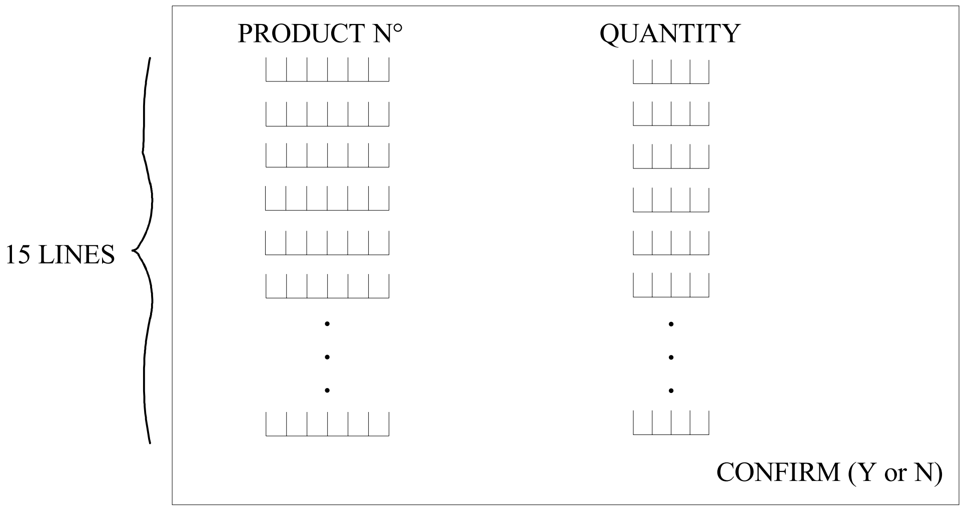
```
ITEM  No.      528
QUANTITY        10


ITEM  No.      529
QUANTITY        30


ITEM  No.      530
QUANTITY         7
```

Forms Mode: 1 exchange for several elements,

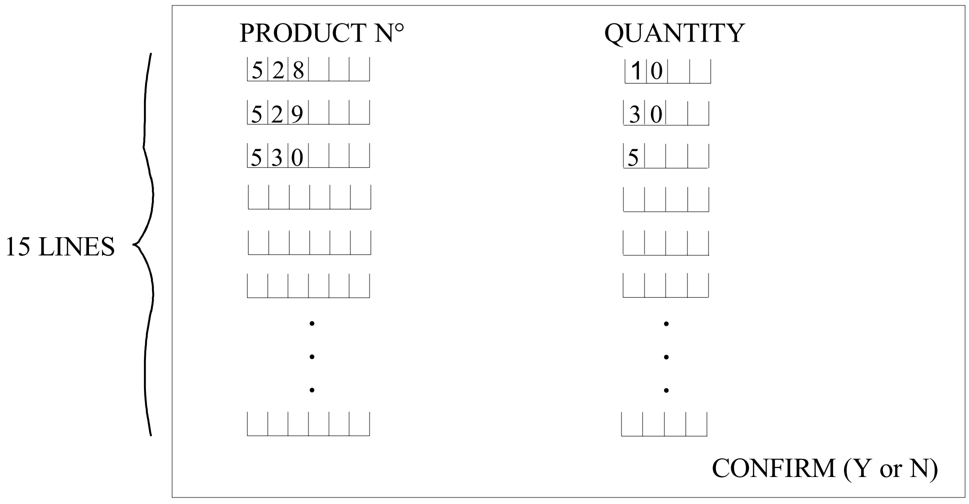
EXAMPLE 2:

In a single exchange the following form is treated.

This is how the empty form appears before the terminal operator enters a piece of information.



This is how the filled form appears after the terminal operator has entered the required information.



PRODUCT N°	QUANTITY
528	10
529	30
530	5

❑

1.1.8 TDS Data Exchange Modes

A TDS application can exchange messages with a partner, terminal operator, or another program in the following modes:

- In line mode through the COBOL statements: RECEIVE and SEND. These are described in *Message Handling Without FORMS* in Chapter 2, and the actual procedures are described in Chapter 3. Note that line mode is not supported with the 327x group of terminals.
- In format mode through the Standard Device Programmatic Interface (SDPI) procedures. FORMS procedures use the SDPI interface. FORMS procedures are described later in *Message Handling With FORMS* in Chapter 2 and the actual procedures are described in Chapter 10.
- In XCP1 mode through the XCP1 primitives. These primitives allow a TDS application to communicate with another application that supports the XCP1 protocol. For more information, see the *Transactional Intercommunication Using XCP1 Protocol User's Guide*.
- In XCP2 mode (refer to the *CPI-C/XCP2 User's Guide*).
- To connect to another application at the same site, or on a different site, without having to log off from TDS, use the `PT (PASS THROUGH)` command that is described in Chapter 14.

1.1.9 Screen Displays

Messages appear on the screen either in line mode, or through the FORMS utility. These two methods of entering messages are explained in the following two sections.

1.1.9.1 Line Mode

In line mode, the transaction is activated by a message from the terminal operator. The TPRs then perform a sequence of SEND and RECEIVE statements to obtain a single data item at a time.

The programmer has available a range of local control functions in CONSTANT-STORAGE such as "clear screen" and "move cursor".

The operator can enter a sequence of data in reply to a prompt from a TPR. However, large messages cause problems of data separation and extraction.

1.1.9.2 Format Mode

In format mode, the TPR names a form to be displayed on the screen. A form is a set of headings with spaces for data entry by the operator. The spaces are known as 'variable fields'. (Appendix C shows a sample form). The form is generated from the data description that has fields for:

- fixed headers,
- strings of control codes,
- data to be keyed in.

The control codes define:

- the screen layout,
- field protection,
- data-range checks,
- cross-field checks.

These attributes and many others are specified when the programmer defines the form.

The terminal is initially in line mode. Any TPR can set the terminal to format mode with the cursor positioned at the start of the first variable field.

The operator then keys in the data. As each variable field is completed, the cursor moves to the next field in sequence. If the operator enters numeric characters instead of alphabetic characters, the entry is rejected. When all fields have been entered, the response is transmitted to TDS and TDS activates the next TPR of the transaction. The TPR receives the message containing only the variable fields separated by tabulation codes.

The TPR can then unstring the message into its component fields to process the data. The advantages of processing in format mode are as follows:

- Screen layout resembles the source document. This makes it easier for the terminal operator to enter data.
- Many data items can be easily entered and transmitted as a single message.
- Many errors are detected and corrected automatically.

1.1.10 Files

The characteristics of the TDS-controlled and non-controlled files are given in the *TDS Concepts Manual*.

The files accessed by a TPR may be TDS-controlled or non-controlled files according to how they were defined in the INPUT-OUTPUT SECTION at TDSGEN.

If the files are non-controlled, they must be declared in the FILE-DEFINITION clause at TDSGEN.

If the files are TDS-controlled, they are declared in the TDS-FILE-DEFINITION or IDS-DEFINITION clauses at TDSGEN and are protected by the General Access Control (GAC) mechanism.

GAC-EXTENDED allows a TDS-controlled file to be shared concurrently, but GAC-EXTENDED cannot be used for non-controlled files. For a complete explanation of GAC-EXTENDED, see the GAC-EXTENDED User's Guide.

1.1.10.1 TDS-controlled Files

TDS-controlled files can be UFAS indexed sequential, or relative files, but IDS/II areas are always considered as TDS-controlled files. They are opened by TDS at the start of the TDS session. TDS-controlled files are processed in INPUT or INPUT-OUTPUT according to the clause specified in PROCESSING-CONTROL at TDSGEN.

During the TDS session, a UFAS-EXTENDED file can be dynamically closed, (for example, the TDS Administrator issues the [M] CLOSE_TDS_FILE command), or can be dynamically re-opened, (for example, the TDS Administrator issues the [M] OPEN_TDS_FILE command).

Each time a TDS-controlled file is used, TDS allocates a control structure that contains a pointer to the current record of the file. See Figure 1-7.

If the transaction was declared with the FOR DEBUG clause at TDSGEN (unless it has been validated by the master command `[ M ] MODIFY_TX VALIDATE = 1`), the verbs that modify or update files are ineffective: the modifications or updates are performed, but they are undone at commitment time.

IDS/II and UFAS file pointers are kept by TDS throughout the commitment unit and are made available to each consecutive TPR of this commitment unit. Outside the commitment, these file pointers are initially null. For UFAS-EXTENDED files, the "KEEP-CURRENCIES" procedure can be called to retain the file pointers, which would otherwise be set to zero. This option is not available for IDS/II. For more information see the section on KEEP-CURRENCIES in Chapter 9.

TDS-controlled files are automatically protected by CI locking.

GAC-EXTENDED/GAC locks CIs to protect against concurrent accesses, whether read/write. If the SHARED READ clause is specified in the TRANSACTION SECTION at TDSGEN, several readers may access the same CI concurrently.

If a file is declared as INPUT-OUTPUT, the control mechanism can be inhibited by specifying in the TRANSACTION SECTION at TDSGEN

```
SUPPRESS CONCURRENT ACCESS CONTROL.
```

This clause may be used when a transaction reads records for statistical purposes while other transactions update these records.

Control is exercised:

- at the control interval (CI) for UFAS-EXTENDED or UFAS files,
- or at the page level for IDS/II areas.

The CI or page containing the record accessed by a transaction is locked for the duration of the current commitment unit.

Concurrent access control causes conflicts when one transaction requests a CI or page that is already locked. The second transaction waits until the resource is freed. If the conflict causes a deadlock (that is, the resource will never be freed), one transaction is aborted by TDS and restarted from the last commitment when the other transaction releases the required CI or page. For a complete explanation of deadlock and long-wait, see the *GAC-EXTENDED User's Guide*.

TDS-controlled files may be protected by both the Before and After Journals. Journals are specified at TDSGEN and through run-time JCL entries. They are therefore not visible to the programmer.

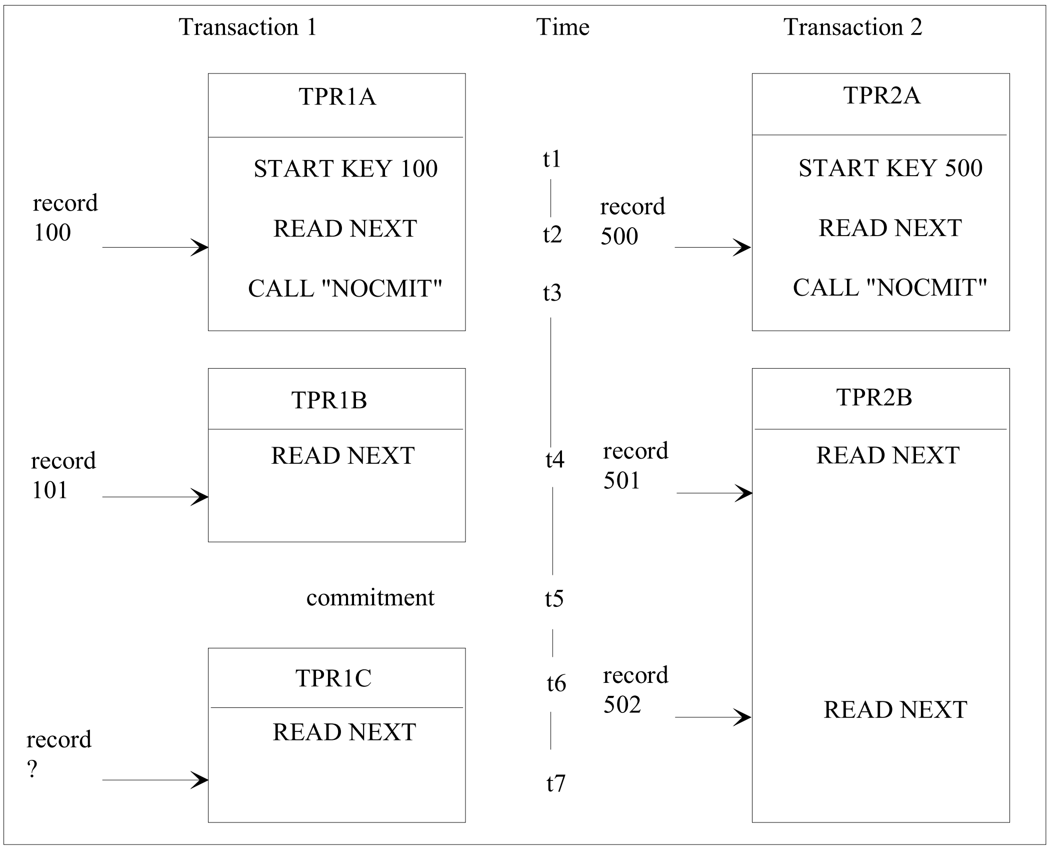


Figure 1-7. Current Record Pointer

Transaction 1 and Transaction 2 access the same file simultaneously. TDS maintains separate file pointers for each transaction and the file is processed correctly by both transactions.

At the end of TPR1A (t3) a call is made to "NOCMIT" (explained in Chapter 3). This means that no commitment is taken. Therefore the READ ... NEXT statement in TPR1B is quite in order. At the end of TPR1B (t5) a commitment point is taken and the file pointer reset so that the READ ... NEXT in TPR1C will not produce the results that were expected: TDS aborts the TPR with the code NOCURREC (no current record defined). The pointer must be repositioned by the TPR unless the KEEP-CURRENCIES procedure is used.

1.1.10.2 Non-controlled Files

Every file except IDS/II areas may be declared a TDS non-controlled file. For these files, the following functions are not supported:

Multiple current record pointers:	non-controlled files cannot be accessed simultaneously because TDS maintains only one set of pointers or currencies per file for the whole TDS application.
FOR DEBUG:	this function indicates which transactions are to be debugged and is described later in Chapter 13.
GAC-EXTENDED:	see the previous subsection.
Deferred updates:	this function means that all updates from a transaction are held in limbo until the transaction is completed. The advantage is that the transaction aborts can easily be handled by discarding these updates. For more information, see the *TDS Administrator's Guide*.

Because there is no concurrent access control and only one current record pointer per file, these files must not be accessed simultaneously.

Non-controlled files may be protected by the Before Journal, but they cannot be protected by the After Journal.

1.1.11 Updates

Records are accessed in packets known as CIs. A CI is the unit of data that is transferred in each physical I/O operation.

When a record is being updated, the whole CI (not the file) is locked. This means that access to the record by another transaction is not permitted until the commitment is taken. Locking at CI level is necessary to avoid concurrent update problems. The updates to the files are not seen by other transactions until a commitment point is taken.

EXAMPLE

The quantity to be ordered will be entered only after the stock level has been checked.

UPDAT 325: The initial message is only to read record 325.

TPR1

```
RECEIVE
  .
  .
    READ record "325"

.
  .
SEND stock
```

The transaction waits for the operator to key in the quantity, say, 12. Meanwhile, record 325 remains locked.

TPR2

```
RECEIVE
  .
  .
calculate

stock - quantity
  .
  .
  .
    REWRITE record "325"
  .
  .
SEND

COMMITMENT
```

❑

Record 325 is locked for the duration of the transaction because a commitment point is not taken at the end of TPR1. Any transaction (say TPR3) trying to access the same control interval (that is, the CI containing record 325) must first wait for TPR2 to terminate, or else will be aborted by TDS. If the commitment unit in TPR3 is aborted, it is automatically restarted at the start of the commitment point when the record is unlocked.

Take the same example, but in this case the quantity ordered is 12 and is fixed at the beginning.

UPDAT 325,12:	The initial message identifies both the record 325 to be updated and the quantity, 12. Processing takes place without further operator intervention.

```
RECEIVE


READ record "325"
    .
    .
stock > 12----------NO
        |           |
        |  YES      |
        |           |
stock - quantity    SEND message
    .               "stock insufficient"
    .
REWRITE record "325"


COMMITMENT
    .
    .
SEND
```

COMMITMENT

STATISTICAL READ

When a TPR tries to access a record in a CI or page already locked by another TPR or by a batch program, the TPR is forced to wait until the CI or page is released, unless "statistical read" is specified. "Statistical read" means that a transaction can access data that is being updated by other commitment units. For further details, see the *TDS Administrator's Guide*.

COMMITMENT

Depending on the type of commitment (IMPLICIT or EXPLICIT option) chosen by the user for each transaction at TDSGEN, the programmer performs the following actions:

IMPLICIT COMMITMENT

When the IMPLICIT COMMITMENT option is specified for a transaction, TDS performs a commitment:

- for every conversation,
- at the end of every transaction,
- at the end of a TPR when a WAIT-TIME value has been specified in the TDS-STORAGE.

The programmer can override the implicit commitment by using the following statements:

• CALL "NOCMIT"	which will cancel a commitment point that would otherwise be processed. This statement is explained in Chapter 3.
• CALL "DFCMIT"	which will force a commitment point to be processed when otherwise no action would be taken. This statement is explained in Chapter 3.

EXPLICIT COMMITMENT

Commitment processing is entirely under the control of the programmer, who must code CALL "DFCMIT" statements for a commitment to be processed.

1.1.12 Restarting after a Failure

Until now we have assumed that each transaction runs happily to completion. In practice, several things might happen to prevent a transaction from completing. The system (GCOS 7) could fail from a variety of hardware and software causes. TDS could abort or maybe only a single transaction aborts.

Restarting a transaction means restoring the status of the transaction to some previous point in time.

After a system crash, or a TDS abort, TDS can be restarted in two ways:

- Warm restart,
- Cold restart.

A TDS warm restart brings the application back to its previous status. This is the normal restart after a crash or a TDS abort. TDS rolls back (reverses) the files and restarts the transactions from their last commitment, that is, from their last safe state.

A cold restart erases information left over from the previous session. For example, events such as the sessions that are abnormally disconnected, or the transactions that are not terminated, are lost.

Note that file integrity is not done by the type of TDS restart. If the previous TDS step aborts, then file integrity functions are executed at the abnormal end of the step. If the system crashes, then file integrity functions are executed at the warm restart of the system.

If a cold, or clean restart is executed for the system, file integrity functions cannot be done at restart time; static rollforward must be performed before a TDS application is restarted. The TDS application is forced to restart in Cold mode.

Less serious than a system crash, or a TDS abort is the failure of a single transaction. To recover from such a failure, TDS maintains the Before and After Journals on which details of all update operations are recorded. Thus, if it becomes necessary to undo some particular update, TDS can use the Before-Journal entry to restore the updated item to its previous value, that is, the commitment unit will be undone.

NOTE:

A FOR INQUIRY transaction is not restarted if TDS aborts or the system crashes.

For more information on the protection of user files, see the *File Recovery Facilities User's Guide*.

1.1.13 Batch Interface

A batch program can exchange data with TDS as long as it simulates a terminal. The batch program logs onto TDS as any terminal does.

The batch program is written in COBOL. It uses `COPY` and `CALL` statements to four special procedures that are explained in the section Debugging Using Batch Interface Procedures in Chapter 13. Messages between the batch program and TDS pass through a supplied data structure that is accessed from a program by a `COPY` statement in the WORKING-STORAGE SECTION.

A batch program can simulate only one terminal.

The maximum number of such pseudo-terminals that may be active at any one time is defined in the TDSGEN.

1.1.14 GTWriter

TDS applications usually require one or more carefully formatted reports to be produced (the term report means a set of formatted output). The GTWriter facility tries to ease this burden by allowing the programmer to specify the physical layout of a report. This facility is described in Chapter 2, and the procedures are given in Chapter 11.

1.1.15 Special Services

TDS executes various functions at the start of, during and at the end of a TDS session. Each function is called a special-purpose transaction and is stored in the sharable module library. These are explained in Chapter 12.

1.1.16 Storages

There are a number of standard data-areas from which the TPR derives information that it needs to perform its processing, and into which it places data for the control of the communications network and for the TDS routines and other TPRs or transactions.

The following data areas are used:

`TDS-STORAGE -------------->`	used as a shared storage between TPRs and TDS.
`CONSTANT-STORAGE --------->`	contains communications control-characters.
`PRIVATE-STORAGE ---------->`	owned by a single TDS user for a whole TDS session, that is, from the user's first log-on to the user's normal log-out, or to the user's cancellation by the master.
`TRANSACTION-STORAGE ------>`	shared by every TPR in one transaction.
`COMMON-STORAGE -----------|`	
`CONTROLLED COMMON-STORAGE |->`	used as storage areas shared by all transactions of one TDS.
`SHARED-STORAGE -----------|`	
`INPUT CD Area ------------|`	
`|->`	These are data areas for data transmission parameters. The data structures generated in the input and output Communication Descriptions (CDs) allow the TPR to handle messages through SEND and RECEIVE statements.
`OUTPUT CD Area -----------|`	

All these storage areas are explained in Chapter 2.

1.1.17 An Example of a COBOL TPR

Because COBOL has been the preferred programming language among TDS users, this is the language used in this manual to illustrate TPR programming. Before getting involved in detail it is important to have an overall view of a TPR. A TPR written in COBOL comprises the same four divisions as a COBOL program: IDENTIFICATION, ENVIRONMENT, DATA, and PROCEDURE.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TPRECHO.
AUTHOR. SMITH.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. DPS7.
OBJECT-COMPUTER. DPS7.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 OUTPUT-MESSAGE          PIC X(50).
01 INPUT-MESSAGE.
   02 DEB-INPUT-MESSAGE    PIC X(3).
   02 IN-MESSAGE           PIC X(97).

LINKAGE SECTION.
COPY TDS-STORAGE.
COPY CONSTANT-STORAGE.
01 TRANSACTION-STORAGE.
   RFU                     PIC X(1000).

COMMUNICATION SECTION.
CD CD-IN FOR INPUT
   SYMBOLIC QUEUE IS       SQI
   MESSAGE DATE IS         MDI
   MESSAGE TIME IS         MTI
   SYMBOLIC SOURCE IS      SSI
   TEXT LENGTH IS          TLI
   END KEY IS              EKI
   STATUS KEY IS           SKI
   MESSAGE COUNT IS        MCI.
```

```
01 CDI PIC X(87).
CD CD-OUT FOR OUTPUT
   DESTINATION COUNT IS    DCO
   TEXT LENGTH IS          TLO
   STATUS KEY IS           SKO
   ERROR KEY IS            EKO
   SYMBOLIC DESTINATION IS SDO.

01 CDO PIC X(23).


PROCEDURE DIVISION USING TDS-STORAGE, CONSTANT-STORAGE
                              TRANSACTION-STORAGE.
START.

1)         MOVE SYMBOLIC-QUEUE TO SQI.
2)         MOVE 100 TO TLI.
3)         MOVE 1 TO DCO.
4)         MOVE 50 TO TLO.
5)         RECEIVE CD-IN MESSAGE INTO INPUT-MESSAGE.
6)         DISPLAY "MESSAGE RECEIVED: "INPUT-MESSAGE UPON ALTERNATE CONSOLE.
7)         IF SKI NOT = "00" GO TO RECEIVE-ERROR.
8)         MOVE SSI TO SDO.
9)         IF DEB-INPUT-MESSAGE = "END" GO TO FIN.
10)        MOVE "READY FOR NEXT MESSAGE (OR END TO TERMINATE)" TO OUTPUT-MESSAGE.
11)        SEND CD-OUT FROM OUTPUT-MESSAGE WITH EGI.
12)        IF SKO NOT = "00" GO TO SEND-ERROR.
13)        MOVE "TPRECHO" TO NEXT-TPR.
           GO TO FIN.

RECEIVE-ERROR.
           DISPLAY "RECEIVE ERROR     " SKI UPON TERMINAL.
           GO TO FIN.
SEND-ERROR.
           DISPLAY "TRANSMISSION ERROR" SKO UPON TERMINAL.
FIN.
           EXIT PROGRAM.
```

Comments:

1. The TPR named TPRECHO loads the input CD SYMBOLIC-QUEUE with the contents of the TDS-STORAGE field SYMBOLIC-QUEUE.
2. Indicates that there are 100 positions available for the data to be received.
3. Sets DCO to 1.
4. Indicates that there are 50 positions available for the data to be sent.
5. The message is made available to TPRECHO.
6. Displays that the message is received on a terminal other than the submitting terminal.
7. The status of the message received is tested.
8. The output CD is to contain the identifier of the activating terminal.
9. Tests for the end of the message to terminate the transaction (see Step 13).
10. Sends a prompt to the terminal operator.
11. The SEND verb transfers the message to the terminal.
12. The status of the message sent is tested.
13. TPRECHO indicates to TDS that TPRECHO is the next TPR. This means that TPRECHO repeats itself until the end of the transaction (see step 9). The absence of a value in the NEXT-TPR field indicates the end of the transaction.

The TPR is covered in full in the next chapter. An example of a TDS Generation Program (TDSGEN) can be found in the *TDS Administrator's Guide*.

2. Programming the Transaction

This chapter describes the DIVISIONs that declare information specific to TDS. The language for programming the TPRs is COBOL.

The TPR is programmed like any communications program in which input and output Communication Descriptions (CDs) are used with SEND and RECEIVE statements. However, the TPR uses storages specific to the TDS environment which are declared at TDSGEN and then referenced by the TPR.

2.1 Identification Division

This division of a TPR is similar to that of a standard COBOL program. The identifier specified in the PROGRAM-ID statement is the name of the TPR specified in TDSGEN (MESSAGE statement) and in the CURRENT-TPR, NEXT-TPR, ON-ABORT-TPR, and PRIOR-TPR fields of TDS-STORAGE. These fields are explained the section LINKAGE SECTION later in this chapter. The name of a TPR must be different from all other names known in the generated TDS (file names, shared data names, etc.).

In the TPR:	In the Generation Program:
IDENTIFICATION DIVISION. PROGRAM-ID. TPRECHO. AUTHOR. SMITH.	TRANSACTION SECTION. MESSAGE "ECHO" ASSIGN TO TPRECHO.

TPRECHO is the name of the TPR. The subsequent TPR is indicated in the NEXT-TPR field of TDS-STORAGE. If there is no subsequent TPR, this means that this TPR is the last TPR in the transaction.

"ECHO" is the message-id as described in the *TDS Administrator's Guide*.

2.2 Environment Division

The ENVIRONMENT DIVISION is used to supply information about the configuration, special hardware characteristics and input/output control. This division is composed of two sections: the CONFIGURATION SECTION and the INPUT-OUTPUT SECTION.

2.2.1 Configuration Section

Defines the functional characteristics of the source and object computer. This section is identical to that used in a batch COBOL program and includes two clauses:

```
SOURCE COMPUTER and OBJECT COMPUTER.
```

2.2.1.1 OBJECT-COMPUTER

Segment sizes are by default 4 Kbytes. In order to override these defaults, the user specifies values for:

- `MAXIMUM PROCEDURE SEGMENT SIZE psegmax`
- `MAXIMUM DATA SEGMENT SIZE dsegmax`

However, at compilation time, the user can again override these sizes by specifying values for the PSEGMAX and DSEGMAX parameters in the COBOL statement. For an explanation of PSEGMAX and DSEGMAX, see the *COBOL 85 User's Guide*.

EXAMPLE

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. DPS7.
OBJECT-COMPUTER. DPS7.
```
❑

2.2.2 INPUT-OUPUT-SECTION

Associates the files to be used by a TPR with those described at TDSGEN.

2.2.2.1 FILE-CONTROL

The FILE-CONTROL paragraph is used by the programmer to identify all files to be used by COBOL TPRs. Every file except the user journal file must be described at TDSGEN which stores file-control entries (SELECT clause) in the tdsname.COBOL file. A TPR that needs a particular file must issue a COPY statement in the ENVIRONMENT DIVISION of a COBOL TPR to retrieve the file-control entries from the library.

If you are using COBOL, you must declare TDS non-controlled files at TDSGEN. If you are using other languages, you can assign and open non-controlled files within a TPR.

Illustrations of FILE-CONTROL entries and the rules regarding their use can be found in the description of the ENVIRONMENT DIVISION in the *COBOL 85 Reference Manual*.

Syntax

```
FILE-CONTROL.

      COPY SELECT-filename. [...]
      [COPY SELECT-USERJRNL.]
```

The filename is up to 24 alphanumeric characters in length and must not start or end with a hyphen.

NOTES:

1. The EXTERNAL phrase in the SELECT clause of TDSGEN, which is optional in standard COBOL, is required by TDS to indicate that the associated internal file name can be declared by more than one TPR.
2. For TDS-controlled files, access mode can be dynamic or random and organization must be UFAS.
3. Each file control entry must be terminated by its own *END statement, in addition to the period that terminates the standard COBOL format.
4. The internal-file-name USERJRNL is reserved for the user journal and must not be specified in any FILE-CONTROL entry. When the TPR needs to write to the user journal file, you must specify the COPY SELECT-USERJRNL clause. The corresponding description is produced by the TDSGEN as a result of the USER-JOURNAL statement; see the example in the description of the FILE SECTION later in this chapter.

Syntax in the TPR:	Syntax in the TDSGEN:
`FILE-CONTROL.`	`FILE-CONTROL.`
	`SELECT EXTERNAL FT1`
`  COPY SELECT-FT1.`	`       ASSIGN TO T1`
	`       ORGANIZATION UFF INDEXED`
	`       ACCESS MODE IS DYNAMIC`
	`       RECORD KEY IS IKEY1`
	`       FILE STATUS IS FSTATUS`
	`       WITH FLR.`
	`*END.`

2.3 Data Division

2.3.1 Purpose and Use

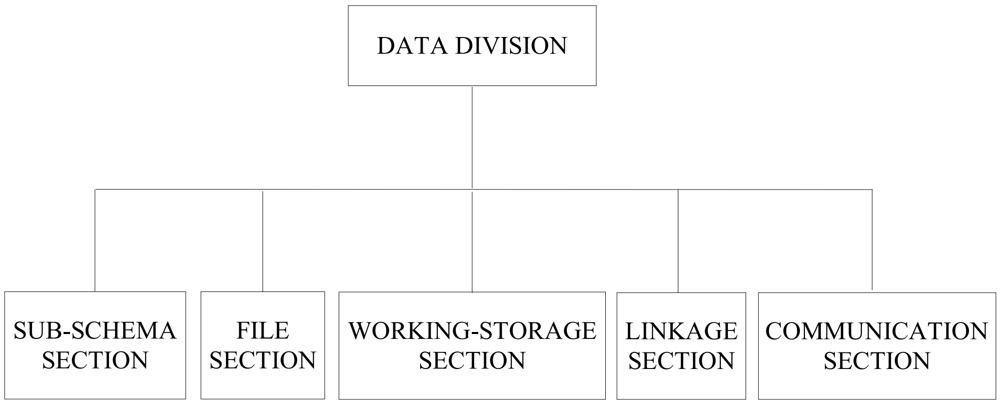
The Data Division defines the nature and characteristics of the data that the TPR is to:

- accept as input,
- manipulate,
- create,
- produce as output.

Data to be processed can be contained in files, placed into intermediate or working storage, or formatted specifically for output purposes. The programmer also uses this division to describe the communication area for storing input and output messages.

2.3.2 Structure

The Data Division of a TPR has the same structure as that used in a batch COBOL program with the exception that the LINKAGE SECTION is required. The following paragraphs describe how each section of the Data Division is used in a TPR.



If you are using IDS/II, the SUB-SCHEMA SECTION must be the first SECTION of the Data Division.

2.3.3 SUB-SCHEMA SECTION

The following IDS/II statements have a special meaning for TDS. For all other IDS/II statements, see the *IDS/II Reference Manual* and the *Full IDS/II Reference Manual*.

The SUB-SCHEMA SECTION is mandatory if IDS/II is used.

2.3.3.1 SUB-SCHEMA SECTION for IDS/II

The SUB-SCHEMA SECTION contains a unique DB schema entry and specifies the database schema that the TPR accesses. The database schema must be described in the IDS-DEFINITION clause at TDSGEN. A TDS application can support up to 32 schemas. It is important to remember that only one DB schema can be used by a TPR.

Syntax

```
DATA DIVISION.

SUB-SCHEMA SECTION.

DB schema-name.

     [ DB-DESCRIPTIONS IN { WORKING-STORAGE } SECTION. ]
                          { LINKAGE         }

     [{ RECORDS ARE     { ALL                     } ]
     [                  { [ NOT ] record-name ... } ]
     [                                              ]
     [{ REALMS ARE      { ALL                     } ]
     [                  { [ NOT ] record-name ... } ]

         .
         .
         .
   WORKING-STORAGE SECTION.
         .
         .
         .
   77 identifier USAGE IS DB-KEY.
         .
         .
         .
```

Usage

The schema referenced by schema-name must be the schema declared in the INPUT-OUTPUT SECTION of TDSGEN.

If the program is the main (or only) program of a TPR (that is, directly called by TDS), the DB-DESCRIPTIONS clause must specify WORKING-STORAGE. If the entire clause is omitted, this is the default.

A program which is called from the main program of a TPR, when the calling program contains a SUB-SCHEMA SECTION (that is, uses IDS/II), must contain the DB-DESCRIPTIONS IN LINKAGE clause.

A program that is called from the main program of a TPR, when the calling program does not use IDS/II, must contain the DB-DESCRIPTIONS IN WORKING-STORAGE clause.

A program with37

the DB-DESCRIPTIONS IN LINKAGE clause must not contain any READY or FINISH statements.

If a program with the DB-DESCRIPTIONS IN WORKING-STORAGE clause contains any READY or FINISH statements, these statements are ignored at execution time.

When there are called programs in a TPR, each call to a called program must contain, in the USING phrase of the CALL statement, arguments concerning the database identifiers that will be used in the execution of the called program. In the called program, the USING phrase of the PROCEDURE statement must contain similar arguments. The user must provide the argument list. The argument list must be defined as follows:

...USING ua ... DB-REGISTERS DB-CXT DB-PARAMETERS rn ...

where:

- ua . . . , are the user data-arguments that are being passed between programs.
- DB-REGISTERS, DB-CXT, and DB-PARAMETERS pass the necessary control structures to permit the execution of all DML functions (except READY and FINISH). DB-PARAMETERS must not be specified if the schema does not contain any database parameter.
- rn..., is a list of the record names referenced in the called program. The record names must be the same as those recorded in the schema. If a record type has no field, it must not be specified in the list.

The RECORDS clause allows the user to declare that the program references only records of a specified series of types. Only information concerning these record types will be placed in the program during DML preprocessing.

If the RECORDS ARE ALL clause is coded, then the entire database as defined by the schema may be referenced. However, only the operations on the areas specified in the TDSGEN will be permitted at execution time.

As an alternative to the RECORDS ARE clause, the user may code the REALMS ARE clause which limits the scope of program reference to the record types defined in the selected areas. These areas must have been specified in TDSGEN. The REALMS clause is only a more convenient way of writing a lengthy list of records.

If the word `NOT` prefixes the list of record, or realm names, then record types defined in the entire database, or in all the areas, except those declared, are available.

The effect of the `REALMS ARE ALL` clause is the same as that of the RECORDS ARE ALL clause.

The use of the `RECORDS/REALMS` clause reduces the size of the UWA (or Linkage area description) which the program will require. In addition, it ensures that only specified records (or realms) will be referenced by the program.

2.3.3.2 SUB-SCHEMA SECTION for Full IDS/II

The SUB-SCHEMA SECTION can contain up to 32 DB subschema entries and specifies the database schemas that the TPR accesses. The database schema must be described in the IDS-DEFINITION clause at TDSGEN.
The USE-IDS-SUBSCHEMA clause must also be specified at TDSGEN.
The database schema is accessed via a subschema and it is important to remember that one database schema cannot be simultaneously accessed via two different subschemas.

Syntax

```
DATA DIVISION.

SUB-SCHEMA SECTION.

{ DB db-ifn-name USING sub-schema-name WITHIN schema-name.

     [ DB-DESCRIPTIONS IN { WORKING-STORAGE } SECTION. ]   }
                          { LINKAGE         }
       .
       .
       .
   WORKING-STORAGE SECTION.
       .
       .
       .
```

Usage

The db-ifn-name is only a qualifier that eliminates ambiguity for a DML verb. The use of a db-ifn-name is mandatory even if there is only one DB clause in the COBOL program.

The sub-schema referenced by sub-schema-name must be defined for the schema referenced by schema-name.

The schema referenced by schema-name must be a schema declared in the INPUT-OUTPUT SECTION of TDSGEN.

The clauses "RECORDS ARE ..." and "REALMS ARE ..." no longer apply. All records described in the sub-schema can be referenced.
The other rules described for IDS/II in the previous section still apply (main and secondary program).

A secondary program can be defined without restriction as a "USE PROCEDURE".

2.3.4 FILE SECTION

Syntax

```
FILE SECTION.

      COPY FD-filename....
```

Description

The FILE SECTION is present if files are referred to in the PROCEDURE DIVISION. FD descriptions in the FILE SECTION of a TPR must be described at TDSGEN. The TPR copies the required FD description. Note that you must declare TDS non-controlled files at TDSGEN.

Usage

The filename appended to `FD-` (note that the hyphen must follow the letters FD) can be up to 24 alphanumeric characters. The name must contain at least one letter and cannot start or end with a hyphen.

The filename must correspond to the internal-file-name specified in the SELECT clause of the FILE-CONTROL paragraph of the ENVIRONMENT DIVISION. See the section FILE-CONTROL earlier in this chapter.

An FD entry for the User Journal is automatically provided at TDSGEN. You must specify a `COPY FD-USERJRNL` clause in the FILE SECTION of the TPR that needs to write records to the User Journal file.

The user journal contains user records and input/output messages, depending on the option specified at TDSGEN.

Example 1

Syntax in the TPR:	Syntax in the TDSGEN:

```
DATA DIVISION.

     FILE SECTION.
```
	```
	 TDS-FILE-DEFINITION.
	 FD FT1
	    BLOCK CONTAINS 10 RECORDS
	    LABEL RECORD IS STANDARD
	    RECORD CONTAINS 80
	CHARACTERS.
	 01 REC1.
	    2 IKEY1 PIC X(5).
	    2 MESS1 PIC X(75).
	 *END.
	```

Example 2: The User Journal

Syntax in the TPR:	Syntax in the TDSGEN:

```
FILE-CONTROL.
COPY SELECT-USERJRNL.

DATA DIVISION.

    FILE SECTION.

         COPY FD-USERJRNL.
```
	```
	TDS SECTION.

	USER-JOURNAL.
	```

2.3.5 WORKING-STORAGE SECTION

The WORKING-STORAGE SECTION is optional and is used in the same manner as in a standard COBOL program. The format and rules for its use can be found in the *COBOL 85 Reference Manual*.

2.3.5.1 WORKING-STORAGE SECTION and SHARED-STORAGE

There are two ways of creating WORKING-STORAGE and only one way of creating SHARED-STORAGE. If used, the SHARED-STORAGE clause must be declared at TDSGEN. WORKING-STORAGE can be defined in the TPR as shown in Format 1 or defined at TDSGEN and copied into the WORKING-STORAGE SECTION of a TPR as shown in Format 2.

```
Format 1

     WORKING-STORAGE SECTION.  }
     77                        }
     .                         }
     .                         }
     .                         }  In Format 1, WORKING-STORAGE
     01                        }  is not defined at TDSGEN.
     02                        }
     .                         }
     .                         }
     .                         }

Format 2

     COPY data-name.           }  In Format2, the WORKING-
STORAGE
                               }  is defined at TDSGEN.
```

SHARED-STORAGE is used to describe user storage areas that are to be shared between two or more transactions. It is defined as part of WORKING-STORAGE in TDSGEN in the SHARED-STORAGE clause. Up to 63 SHARED-STORAGE areas can be used. The maximum size of 1 SHARED STORAGE is 64 Kbytes.

Several TPRs can simultaneously access and update the information in SHARED-STORAGE. Unlike COMMON-STORAGE, it is accessed directly by using MOVE statements. There is no protection against simultaneous access by several TPRs and if this is required it must be established through the non-concurrency clause in TDSGEN, or even better through the `CALL "LOCK"` procedure (described in Chapter 9). Without this protection, inconsistent data may be produced in the following two cases:

- MOVE statements processing more than 255 characters,
- TPRs with different priorities in simultaneous access.

The WORKING-STORAGE Section of a TPR is refreshed automatically by TDS: the original contents of the WORKING-STORAGE are therefore provided each time the TPR is called. However, the SHARED STORAGE area in WORKING-STORAGE has the latest updated contents.

2.3.5.2 COMMON-STORAGE

COMMON-STORAGE is a storage area shared among all transactions of a TDS. A TPR can pass information to another TPR within the same or another transaction through COMMON-STORAGE. Any transaction can read, modify, and save this storage area by calling the appropriate TDS function.

The size and structure of COMMON-STORAGE is described at TDSGEN (in the TDS SECTION and the WORKING-STORAGE clause of the INPUT-OUTPUT SECTION respectively), and COPYed into the WORKING-STORAGE SECTION(s) of the TPR(s). Alternatively, the structure of COMMON-STORAGE can also be described in the WORKING-STORAGE of the TPR itself.

The characteristics of COMMON-STORAGE are as follows:

- maximum size is 65504 bytes (64 Kbytes minus 2 bytes),
- not a controlled area (access control is not handled by GAC-EXTENDED or GAC),
- not subject to rollback.
- The TPR accesses COMMON-STORAGE by means of specific calls that allow the user to control any access conflicts.

The individual TPR within a transaction can access and modify COMMON-STORAGE. At a cold start of the TDS session, COMMON-STORAGE is set to zero. At a warm restart, COMMON-STORAGE is in the state in which it terminated at the end of the previous TDS session.

Any transaction can operate on this area by issuing the following, where data-name-1 is the name of an area declared in the WORKING-STORAGE and data-name-2 is the length of COMMON-STORAGE.

```
              { "READ-COMMON" }
              { "TAKE-COMMON" }
       CALL   { "FREE-COMMON" } USING data-name-1 [,data-name-
2].
              { "SAVE-COMMON" }
              { "KEEP-COMMON" }

       CALL "LENGTH-COMMON" USING data-name-1.
```

Figure 2-1 provides an overview of these CALL statements. For an explanation of each CALL COMMON statement, see Chapter 8.

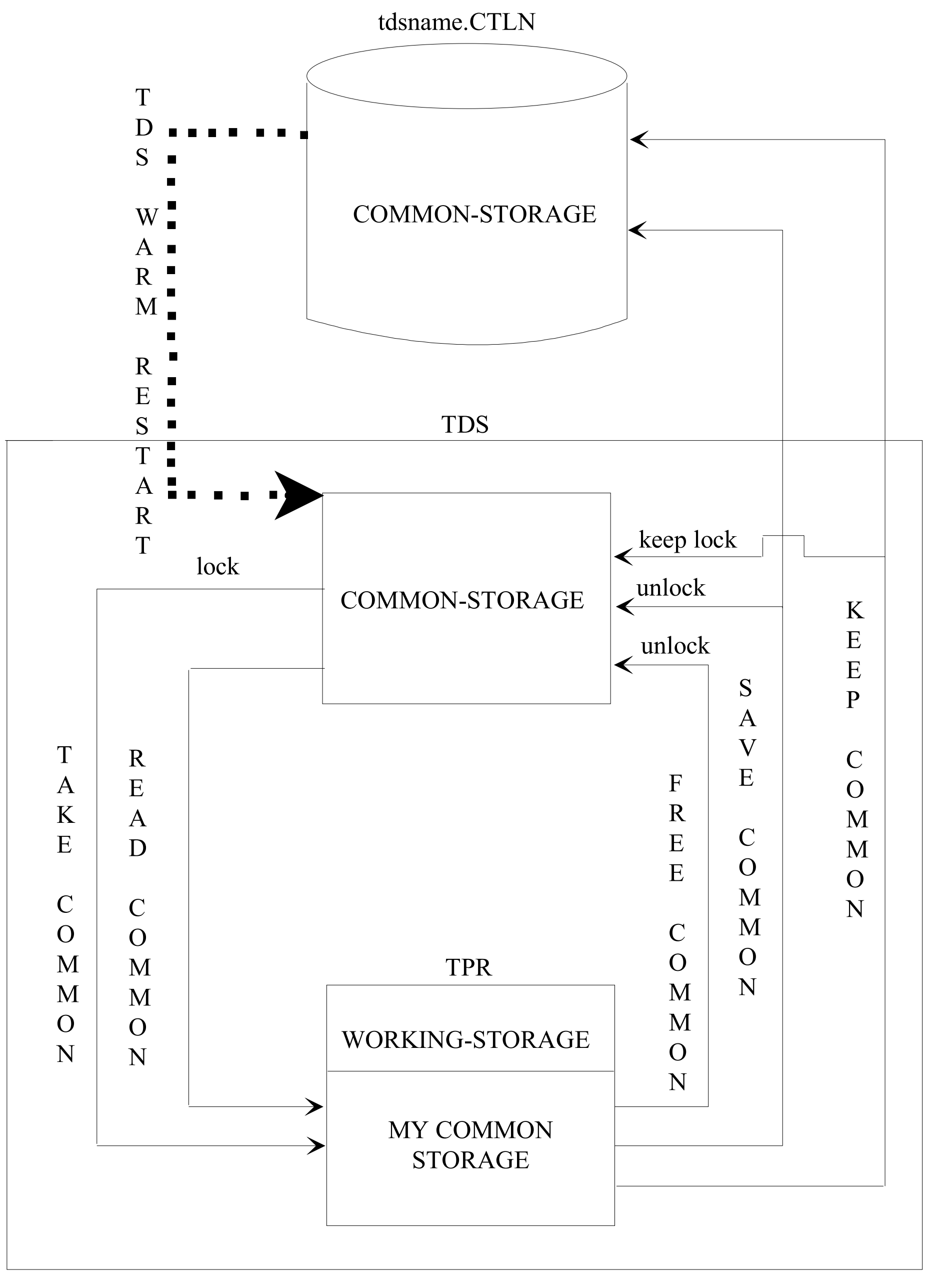


Figure 2-1. Handling COMMON-STORAGE by Using CALL Statements

NOTE:

At TDS warm restart, COMMON-STORAGE is initialized from the tdsname.CTLN file.

COMMON-STORAGE can be updated only after TAKE-COMMON is used.

2.3.5.3 CONTROLLED COMMON-STORAGE

CONTROLLED COMMON-STORAGE is a communication area shared among all transactions. Access to these areas is controlled by GAC-EXTENDED (or GAC). Each area is protected by the Before-Journal mechanism and rolled back if a commitment aborts. The main difference between CONTROLLED COMMON-STORAGE and COMMON-STORAGE is recovery; a CONTROLLED COMMON-STORAGE area can be rolled back whereas a COMMON-STORAGE area cannot be rolled back.

CONTROLLED COMMON-STORAGE areas have the following characteristics:

- Maximum size of each area is 64 Kbytes,
- Each area has a unique name,
- Up to 64 CONTROLLED COMMON-STORAGEs can be declared.

The CONTROLLED COMMON-STORAGE clause is defined in the INPUT-OUTPUT SECTION of TDSGEN. The data structure of each CONTROLLED COMMON-STORAGE is retrieved by specifying the following:

```
DATA DIVISION.

WORKING-STORAGE SECTION.

COPY data-name-1.
.
.
.
COPY data-name-n.
```

The TPR can access CONTROLLED COMMON-STORAGE by calling:

CREAD-COMMON	allows the TPR to get a CONTROLLED COMMON-STORAGE from the tdsname. CTLM system file.
CWRITE-COMMON	allows the TPR to update a CONTROLLED COMMON-STORAGE in the tdsname. CTLM system file.
CLENGTH-COMMON	returns the exact length in bytes of a CONTROLLED COMMON-STORAGE.

Any transaction can operate on these areas by issuing the following:

```
CALL {"CREAD-COMMON" } USING data-name-1,
     {"CWRITE-COMMON"}       data-name-2,
                             data-name-3,
                             data-name-4.

CALL "CLENGTH-COMMON" USING data-name-1,
                            data-name-2,
                            data-name-3.
```

For a more detailed explanation of these three CALL statements, see Chapter 8.

2.3.5.4 FORMS

This section explains what elements a TPR must have in its WORKING-STORAGE SECTION if it is to use the FORMS facility.

As a follow up to what was mentioned in Chapter 1 on this topic, it is a good idea to refer to the *IOF Programmer's Manual* for an introduction to the FORMS facility. A description of how FORMS handles messages is given later in this chapter.

Three special data structures (form-nameI, form-nameV and form-nameR) are stored in the COBOL source library (H_CBLIB) and copied into the WORKING-STORAGE SECTION of the TPR to provide the interface with the FORMS utility.

```
WORKING-STORAGE SECTION.

[77 data-name PIC X(6)]       Activate Mechanism
[77 data-name PIC X(4)]       Modify Attribute

01 data-name-1.
   COPY form-nameI.           Copy form
identification
01 data-name2.
   COPY form-nameV.           Copy selection vector
01 data-name-3.
   COPY form-nameR.           Copy data record
```

Usage

form-nameR can be COPYed in the LINKAGE SECTION, as part of TRANSACTION-STORAGE.
COPY form-nameI is useful only in the TPR that activates the form using a CALL "CDGET" statement. The other COPY statements form-nameV and form-nameR are necessary for receiving or sending formatted messages and for modifying attributes.

The form can be activated locally by the terminal before the transaction begins to execute. In this case, calling CDGET to activate the form is not necessary.

SEND with ESI statements do not transmit control codes. These codes are transmitted only in a SEND with EMI or EGI statement.

The form may be active during more than one TPR in the transaction and remains active until a TPR calls the CDRELS procedure to release the form.

The form may be active during more than one transaction (the NO IMPLICIT RELEASE clause is defined in TDSGEN).

For example, if a form is called CUST and the user data-names chosen are TDATA1, TDATA2, TDATA3, a TPR activating the form and communicating through it with the terminal must have:

```
01 TDATA1.
  COPY CUSTI.
.
01 TDATA2.
  COPY CUSTV.
.
01 TDATA3.
  COPY CUSTR.
```

Form Identification

The structure form-nameI is used to identify a form, to verify its compilation date and to specify its mode of activation.

The structure of form-nameI is as follows:

```
01 data-name.
   COPY form-nameI.
   02 FILLER          PIC X     VALUE "3".
   02 FILLER          PIC X(8)  VALUE "formname".
   02 form-name-NO    PIC 9(3)  VALUE ZERO.
   02 form-name-MD    PIC X     VALUE "A" or "O" or "W" or "E".
   02 form-name-OF    PIC X(8)  VALUE SPACES.
   02 form-name-OO    PIC 9(3)  VALUE ZERO.
   02 form-name-LL    PIC 9(3)  VALUE ZERO.
   02 FILLER          PIC X(5)  VALUE "date-of-compilation".
   02 FILLER          COMP-1    VALUE "time-of-compilation".
   02 form-name-AF    PIC 9     VALUE any-flag.
   02 form-name-SL    PIC 9(3)  VALUE 1.
   02 form-name-SC    PIC 9(3)  VALUE 1.
```

form-name-NO	specifies the form occurrence number.
form-name-MD	specifies the mode of activation of the form (A=APPEND, O=OVERLAY, W=WINDOW, E=ERASE).
form-name-OF	specifies the old form. The new form can overlay, or be appended to, or replace the old form.
form-name-OO	specifies the occurrence number of the old form.
form-name-LL	is used only for the APPEND and OVERLAY modes. If form-name-LL is zero, the number of lines allocated is equal to the number of lines in the form. If form-name-LL is greater than or equal to the number of lines in the form, the form is completed with the appropriate number of blank lines. If form-name-LL is less than the number of lines in the form (but greater than zero), the status FUNCNAV is returned.
form-name-AF	is the any-flag, which is initialized according to the TERM option selected at form loading time (1 if ANY, otherwise 0).
form-name-SL	is the form starting line for W and E modes.
form-name-SC	is the form starting column for W and E modes.

Selection Vector

The structure form-nameV is used to select individual fields of a form for sending or receiving their contents, to modify their attributes or to return the status of a field.

The structure of form-nameV is as follows:

```
01 data-name.
   COPY form-nameV.

  02 form-nameV.
    03  FILLER             PIC X       VALUE "2".
    03  FILLER             COMP-1      VALUE "number-of-
fields".
    03  FILLER             PIC X(8)    VALUE "formname".
    03  form-name-VO       PIC 9(3)    VALUE ZERO.
    03  form-name-V.
      04  form-name-FC-V   PIC X.
      04  data-name-01-V   PIC X.
      04  data-name-02-V   PIC X.
          .
          .
      04  array-name-AV.
        05  array-name-V  OCCURS dimension.
          06  data-name-i-V    PIC X.
```

Form-name-V	comprises a single character (PIC X) for each named field. The fields are in order of NFIELDs or in screen image order if NFIELDs are not used.
Form-name-VO	specifies the form occurrence number.
Form-name-FC-V	specifies the selection vector entry corresponding to form-name-FC in the data record (described below).

For example, for a form called CUST with two data fields, NUMBER and NAME (in that order) the selection vector would be as follows:

```
02 CUSTV.
   03  FILLER          PIC X          VALUE "2".
   03  FILLER          COMP-1         VALUE 2.
   03  FILLER          PIC X(8)       VALUE "CUST    ".
   03  CUST-VO         PIC 9(3)       VALUE ZERO.
   03  CUST-V.
     04  CUST-FC-V      PIC X.
     04  NUMBER-V    PIC X.
     04  NAME-V      PIC X.
```

Data Record

The structure form-nameR contains the COBOL description of the named fields (NF) and is used to send and receive fields of the form.

The structure of form-nameR is as follows:

```
01 data-name.
 COPY form-nameR.

   04  form-name-FC              PIC 99.
   04  data-name-01              PIC ...
   04  data-name-02              PIC ..
      .
      .
     04  array-name-A.
       05 array-name OCCURS dimension.
         06  data-name-i                 PIC...
```

The first field (form-name-FC) of the data record is generated by default, except when NO IMPLICIT FUNCTION CODE or FUNCTION-CODE is specified in the Source Form Definition Language. This is described in the *IOF Programmer's Manual*. This field receives the control code from the terminal.

The example shows the form called CUST for which NUMBER and NAME are 6 and 20 alphanumeric characters, respectively:

```
04 CUST-FC                    PIC 99.
04 NUMBER                     PIC X(6).
04 NAME                       PIC X(20).
```

If NUMBER had been defined in SPIC as "`+99.9`", the structure would be:

```
04 CUST-FC                    PIC 99.
04 NUMBER                     PIC S99V9.
04 NAME                       PIC X(20).
```

The structure of an array CUST-ARRAY for 5 occurrences of NUMBER and NAME is:

```
04 CUST-ARRAY-A.
   05 CUST-ARRAY OCCURS 5.
      06 NUMBER               PIC S99V99.
      06 NAME                 PIC X(20).
```

2.3.6 LINKAGE SECTION

The LINKAGE SECTION is the only mandatory section of the Data Division; it describes data referred to by both the calling program (that is, TDS itself), and the called program (TPR). Thus, the LINKAGE SECTION is required in the main program of the TPR. Other programs (sub-programs) of the TPR can use the LINKAGE SECTION as in a COBOL batch program.

The three storage areas of the LINKAGE SECTION are:

- TDS-STORAGE.
- CONSTANT-STORAGE.
- TRANSACTION-STORAGE.

The three storage areas must be described in the following order:

```
[TDS-STORAGE [, CONSTANT-STORAGE [, TRANSACTION-STORAGE ]]]
```

2.3.6.1 TDS-STORAGE

Through the TDS-STORAGE area, control information is passed between TDS and the TPRs. A TDS-STORAGE is kept for each active transaction. It is used by a TPR to store the next TPR. It also contains statistical information about aborts and restarts. The TDS-STORAGE area must be declared in the main (or only) program of the TPR.

The area has the following standard description:

```
01  TDS-STORAGE.
    02  SYMBOLIC-QUEUE            PIC  X(12).
    02  PRIOR-TPR                 PIC  X(12).
    02  CURRENT-TPR               PIC  X(12).
    02  NEXT-TPR                  PIC  X(12).
    02  ON-ABORT-TPR              PIC  X(12).
    02  ABORT-CODE                COMP-1.
    02  USER-ID                   PIC  X(8).
    02  TX-MODE                   PIC  9.
    02  RESTART-STATUS            PIC  9.
    02  TRANSACTION-SERIAL-NUMBER COMP2.
    02  TPR-SERIAL-NUMBER         COMP-1.
    02  WAIT-TIME                 COMP-1.
    02  ABORT-ICC                 PIC X(8).
    02  RESTART-CODE              PIC X.
    02  USER-FULLNAME             PIC X(12).
```

```
02  NO-RESTART                 PIC X(1).
02  REST-CVSTAT                PIC X(1).
02  RESTART-INFO               PIC X(1).
02  LAST-TPRNAME               PIC X(12).
02  EXCP-CLASS-TYPE            PIC X(4).
02  LNODENAME                  PIC X(4).
02  WATCH-TIME                 COMP-1
```

Usage

The TPR must specify:

```
PROCEDURE DIVISION USING TDS-STORAGE ...
```

and must contain as well the LINKAGE SECTION entry to copy the appropriate data description:

```
COPY TDS-STORAGE.
```

Private data declarations used by the TPR must not be inserted after the statement COPY TDS-STORAGE (for example 02 MY-DATA PIC X.). This wrong declaration would lead to a TDS abort.

The SYMBOLIC-QUEUE field contains the tds-name (`PROGRAM-ID` in TDSGEN). It must be moved by the TPR into the input CD SYMBOLIC QUEUE (described later in the section on the COMMUNICATION SECTION) before the TPR issues a RECEIVE statement.

```
MOVE SYMBOLIC-QUEUE TO QUIN
MOVE SPACES TO BUFREC
RECEIVE CDREC MESSAGE INTO BUFREC
```

The input CD is declared as follows:

```
CD CDREC FOR INPUT
SYMBOLIC QUEUE QUIN
```

PRIOR-TPR contains the name of the preceding TPR that activated the current TPR.

The CURRENT-TPR contains the name of the currently executing TPR.

NEXT-TPR - Before a TPR ends, this field must contain either the identifier of the next TPR in the transaction or spaces if it is the last one (it contains spaces at the beginning of the TPR). NEXT-TPR permits multiple branches from a TPR. For example, processing is to be varied on the basis of a question answered by the user:

```
IF ANSWER = "YES" MOVE "TD2-UPDATE" TO NEXT-TPR.

IF ANSWER = "NO" MOVE "TD2-CNCLL" TO NEXT-TPR
   ELSE MOVE "OPERR" TO NEXT-TPR.
```

ON-ABORT-TPR and ABORT-CODE - To give the programmer more control over the handling of exception conditions, several error-processing TPRs may be written. The ON-ABORT-TPR field may be initialized in the current TPR with the name of the appropriate error-processing TPR so that if an abnormal condition arises (for example, TPR exception, time-limit), TDS will set the ABORT-CODE and activate the specified TPR. A TPR may also set ON-ABORT-TPR and then abort itself by means of the statement CALL "ABORT" if necessary.

TRANSACTION-STORAGE and PRIVATE-STORAGE are not rolled back before the ON-ABORT-TPR TPR is executed (as for an abort of a transaction). This makes diagnosing the cause of the error easier.

The ON-ABORT-TPR field is not modified by TDS. The ON-ABORT-TPR field is rolled back when the commitment is aborted to be restarted, for example, in the case of GAC-EXTENDED/GAC rollback for deadlock. However, the ON-ABORT-TPR field is not rolled back when the commitment is aborted and will not be restarted, for example, in the case of CANCELTX which is set in the NEXT-TPR field of an interrupting transaction.

Additional information could be supplied via TRANSACTION-STORAGE, or COMMON-STORAGE. See also the DISCNCT and BREAK transactions described in Chapter 12.

NOTE:

A TPR started at abort time should begin with:

```
RECEIVE .... NO DATA GO TO CONTINUE.
   .
   .
CONTINUE.
```

USER-ID contains the identifier specified at log-on. For an XCP2 session, USER-ID contains the XCP2 correspondent name (name of the partner). If the identifier is more than 8 bytes long, the name is truncated.

TX-MODE - The field indicates whether or not the transaction is executing in TRACE mode. TX-MODE is set by TDS. It is non-zero to indicate TRACE mode. Otherwise the field contains zero. For complete details of TRACE, see Chapter 13.

RESTART-STATUS - Usually any restart is invisible to the user and does not require special attention on the part of the programmer. The TDS files, data areas, and indicators are automatically reset; the one exception is COMMON-STORAGE which is left unchanged. The only sign of a restart is a possible repetition of a message. Sometimes it is necessary to know whether the TPR was activated normally or as a result of a restart.

For example, in some cases, like non-controlled file updating without file protection, if the TPR is restarted, a record insert or delete operation will fail when the record has already been inserted or deleted. The use of RESTART-STATUS will avoid these problems.

Possible values are as follows:

0: the first time that the TPR is activated,

2: a restart after a commitment unit failure, or after a CALL "INVCMIT". For example, a deadlock or LONGWAIT causes the commitment unit of the waiting TPR to abort. Restart also results when CALL "ROLL-BACK" is called.

   After a warm restart of a TDS application, this field is always set to 2 if the transaction is restarted.

The field is reset to zero only when the first TPR of the commitment unit has terminated.

The TRANSACTION-SERIAL-NUMBER field contains a number assigned by TDS. This number identifies the transaction within the TDS session. A TPR must not modify this number.

The TPR-SERIAL-NUMBER field contains a number assigned by TDS. This number establishes the sequence of the TPR within the transaction. A TPR must not modify this field.

The WAIT-TIME field indicates a time delay in seconds that elapses before the next TPR is activated.

- When the TPR terminates with a SEND EGI statement and a specified wait time, the next TPR is activated after the receipt of a new message, or the elapse of the wait time, whichever occurs first. One reason for the timeout may be that the message could not be displayed on the terminal because of long delays in network response times.
- When the TPR terminates with a SEND EMI statement or no SEND statement at all and a specified wait time, the next TPR is activated after the wait time has elapsed and when it is possible for the transaction to send another message.

  For example, if the master terminal requires a report at fifteen-minute intervals, a special transaction can be written for this purpose and activated once at the beginning of the session. The last TPR of the transaction sets the NEXT-TPR field to the identifier of the first TPR of the transaction and fills the WAIT-TIME field with the value 900. This causes the transaction to be reactivated every fifteen minutes for the duration of the session.

  Note that, in this example, the master terminal cannot activate any further transactions. To regain control, the operator must enter a "break" and force the special transaction to terminate. It would therefore be better for the first transaction to spawn the reporting transaction on a DUMMY. The master terminal is then freed for other use. Spawning is described later in this chapter.

ABORT-ICC is set by TDS when ON-ABORT-TPR is activated. It contains the address in the TPR of the statement that caused the abort.

The RESTART-CODE field is set to "1" when a TPR is restarted (`RESTART-STATUS=2`) because a CALL "ROLL-BACK" statement was executed. Otherwise the value of the RESTART-CODE field is set to another value.

The USER-FULLNAME field is the complete name of the user (12 characters in length) for whom the transaction is started. In fact, you can consider it an extended USER-ID field. For an XCP2 session, USER-FULLNAME is the name of the user on whose behalf the transaction is running (XCP2 correspondent name if the transaction has been started with the security option 'NONE').

The NO-RESTART field is for XCP2 users only. It specifies whether the commitment-unit can be restarted.

- A value of 0 means that restarts are supported.
- A value of 1 means that restarts are not supported: in the case of conflicts the transaction is aborted.

The REST-CVSTAT field is for XCP2 users only. It is significant when RESTART-STATUS (in TDS-STORAGE also) is 2. It gives the programmer information about the conversation states. The possible values are:

0: "FIRST-PROCESSING", i.e. this is the first time the TPR is processed.

1: "CONV-RESTORED". This means that, for the local transaction, the state of its conversations have been restored to the state they were in at the beginning of the commitment-unit. This happens when the commitment unit is restarted as a consequence of one or more of the following:

   a request to rollback coming from the local transaction or the partner transaction

   an external event (break, and so on)

   a GAC request to rollback.

   This does not ensure that the partner applications have restored their conversations as well (although they should have). If there is at least one partner application which has not restored the state of a conversation, although it was required, the local transaction will probably get an abnormal status-code on the next XCP2 conversation verb which relates to a conversation that is in an unexpected state. This should happen only in the case of heterogeneous applications (for example an IBM application co-operating with a TDS application).

2: "ALL-CONV-ABORTED". This values is set when:

   the processing is restarted as a result of a warm start or a reinitialization
   the transaction is aborted. This value is always transmitted to the ON-ABORT-TPR.

NOTE:

For a transaction whose main session is an XCP2 session, the TDS-STORAGE is as follows:

USERID = the correspondent name

FULL_USERNAME = the name of the user for which the transaction is running (or blanks if the transaction is invoked with the option 'no user').

The RESTART-INFO field indicates when a commitment unit is restarted after a TDS abort (that can be restarted), or after a warm restart of TDS. This means that the value of the RESTART-STATUS field is 2. Possible values are:

R	The commitment unit is restarted after a TDS restartable abort.
W	The commitment unit is restarted after a warm restart of TDS.
O	Other

The LAST-TPRNAME field contains the name of the last TPR committed (if this TPR was not the last TPR of the transaction). This field is only available during the first TPR of the disconnect transaction. If no transaction was running at disconnection time, this field is filled with spaces.

If the disconnection is related to an XCP1 session, the LAST-TPRNAME field may not be significant (in particular for a session allocated by a TM or DUMMY correspondent). The programmer can use a call "GETTPRPAR" verb to get information about the last TPR committed (if any) and the last TM or DUMMY correspondent using the XCP1 session before the disconnection occurs. Refer to the description of the call GETTPRPAR in chapter 4.

When using stacked contexts, the LAST-TPRNAME field contains the name of the most-recently-committed TPR, no matter what the context rank is. For example, this is the case after a BREAK or ENTERTX.

The EXCP-CLASS-TYPE field contains the class and type of exception that leads to the execution of ON-ABORT tpr. This field is meaningful only in ON-ABORT tpr; otherwise its value is "FFFF".

The LNODENAME field contains the name of the local system.

The WATCH-TIME field indicates that the delay of the answer of the network is to be surveyed at end of TPR with SEND (EMI or EGI).

Recall

- When a TPR ends with a SEND EMI, TDS must wait for the agreement of the network before being able to do another send and then cannot start the next TPR until this agreement is received.
- When a TPR ends with a SEND EGI with a WAIT_TIME and when the terminal user does not answer, TDS at the expiration of the WAIT_TIME asks the network for permission to send another message. TDS must wait for the agreement of the network before starting the next TPR.

Working principles

- The purpose of WATCH-TIME is to ensure a transaction doesn't remain blocked between two TPRs if the network doesn't give its agreement in the two cases described above.
- This functionality is implemented in TDS by means of a new field, WATCH-TIME, in the TDS-STORAGE.
- The WATCH-TIME field indicates a time delay in seconds.
- It is set to the value zero at the beginning of each TPR.
- It is activated at the end of the TPR.
- If the agreement of the network is not received before the delay of the WATCH-TIME is exhausted, the terminal is disconnected.
- The master terminal then receives the message TX88.
- The message TX88 can be suppressed by re-defining the TDS service message 44 with a blank character in the STDS subfile.

Working Rules

1. WATCH-TIME and SEND EMI:

   The WATCH-TIME is not effective in the following cases:

   – Session is neither a terminal nor a TCP-IP session
   – Send is a synchronous one
   – Send is an explicit one.
   – Session is pass-thru mode.

2. The WATCH-TIME and SEND EGI:

   The WATCH-TIME is not effective in the following cases:

   – Session is not a terminal session.
   – Send is the last of the transactions and the terminal is active.
   – Session is in pass-thru mode.

   When a WAIT-TIME has been set, the WATCH-TIME has a meaning only if the value of the WAIT-TIME is less than the WATCH-TIME one because they are both activated at the end of the TPR. If this rule is not respected and if the terminal user does not answer before the delay defined by the WATCH-TIME is expired, the terminal will be disconnected.

2.3.6.2 CONSTANT-STORAGE

The CONSTANT-STORAGE contains communications control characters available to TPRs, usually for formatting purposes on terminals. Eight such characters are pre-defined in this area as part of the software. Examples of such characters are line-feed and carriage-return. Additional characters up to a maximum of 256 can be included in this area through the SPECIAL-CHAR entry in the TDS SECTION at TDSGEN. For example, the code GS (group separator) can be re-defined with a value that the terminal recognizes as GS,

```
SPECIAL-CHAR GS IS "E1".
```

The default value is "1D" (hexadecimal code). This code will be added to the list of codes in the CONSTANT-STORAGE area. Table 2-2 gives the control codes for DKU7007.

Fields in CONSTANT-STORAGE can be 'read' by the TPR, for example,

```
MOVE LF TO A-MESS-17.
```

but cannot be modified. The characters are used to control layout on the terminal when other means are not available or not applicable; they may also be used to delimit input fields.

The CONSTANT-STORAGE area is included in the TPR by using the following statement in its LINKAGE SECTION:

```
COPY CONSTANT-STORAGE.
```

and the TPR using it must contain the following entry in the PROCEDURE DIVISION:

```
PROCEDURE DIVISION USING TDS-STORAGE CONSTANT-STORAGE ...
```

2.3.6.3 PRIVATE-STORAGE and TRANSACTION-STORAGE

PRIVATE-STORAGE contains data that is private to a single TDS user (which corresponds to a user session). PRIVATE-STORAGE is set to zeros when the first transaction, that is, the LOGON transaction, for the user is started and then kept until the user logs off. This is not true for FOR INQUIRY transactions. The user can store statistics and control information in PRIVATE-STORAGE. A PRIVATE-STORAGE is passed from one transaction to the next.

NOTE:

For a remote XCP2 transaction, the time life of the session is the same as that of the remote transaction working for the conversation. Since TS6150 the private-storage of the remote XCP2 transaction is initiated with zeros at the beginning.

A TRANSACTION-STORAGE contains data that is private to a transaction. The programmer can use this storage area to make TPRs belonging to the same transaction communicate with one another: data generated by the previous TPR can be passed to a subsequent TPR within the transaction. Therefore the size of the TRANSACTION-STORAGE may vary according to the transaction. At the beginning of a transaction, all data items in the TRANSACTION-STORAGE area are set to zero up to the size declared at TDSGEN. One exception is the set of special-purpose transactions described in Chapter 3.

Data values set up by the first TPR are passed unchanged by TDS to the next TPR and so on regardless of the commitment unit. The user defines the use of TRANSACTION-STORAGE.

TRANSACTION-STORAGE is useful when several transactions are being processed in parallel, which is the normal case for TDS applications. A storage area is needed that is unique to each transaction. All details related to the transaction are kept in this area.

It is possible to access PRIVATE and TRANSACTION STORAGE separately, using the PROCEDURE DIVISION clause in the TPR, as follows:

```
PROCEDURE DIVISION USING TDS-STORAGE CONSTANT STORAGE
                   PRIVATE-STORAGE TRANSACTION-STORAGE
```

There are two ways of declaring PRIVATE and TRANSACTION STORAGE:

- in the LINKAGE SECTION of the TPR itself:

```
01 PRIVATE-STORAGE.
      02 private-storage-item.
            .
            .
01 TRANSACTION-STORAGE.
      02 transaction-storage-item.
            .
            .
```

  Note that in this case the order of the declarations is not important, because separate pointers in the PROCEDURE DIVISION clause access the structures.

- in the STDS:

  in the TRANSACTION-SECTION of the STDS:

```
01 PRIVATE-STORAGE.
       02 private-storage-item.
             .
             .
01 DEFAULT TRANSACTION STORAGE-SIZE
01 TRANSACTION-STORAGE.
       02 transaction-storage-item.
             .
             .
 *END
```

  Note that in this case the order of the declarations must be as shown above.

- In the TDS-SECTION of the STDS:

  Use the PRIVATE-STORAGE clause in STDS as follows:

```
PRIVATE-STORAGE SIZE IS pssz.

01 PRIVATE-STORAGE.
    02 private-storage-item.
          .
          .
*END
```

Use the DEFAULT TRANSACTION-STORAGE SIZE clause in STDS as follows:

```
DEFAULT TRANSACTION-STORAGE SIZE IS deftxstorage.
```

Use the TRANSACTION-STORAGE clause in STDS as follows:

```
TRANSACTION-STORAGE SIZE IS txxz.

01 TRANSACTION-STORAGE.
      02 transaction-storage-item.
            .
            .
```

NOTE:

Even though PRIVATE STORAGE is not declared in the TRANSACTION SECTION, the size specified in the TRANSACTION-STORAGE ... SIZE IS ... clause must include the PRIVATE-STORAGE size. For example:

```
PRIVATE-STORAGE SIZE IS 24.
01 PRIVATE-STORAGE.
     02 ADTERM PIC X(12).
     02 USERNM PIC X (12).
*END
```

and in the TRANSACTION SECTION:

```
TRANSACTION-STORAGE tx-storage SIZE IS 34. (= 10 + 24)
01 TRANSACTION-STORAGE.
     02 TX-ITEM PIC X(10).
*END
```

Figure 2-2 shows the relationships between the TPR on the one hand, and TDS-STORAGE, COMMON-STORAGE, TRANSACTION-STORAGE, and SHARED-STORAGE on the other. For more information on separate access to PRIVATE-STORAGE and TRANSACTION-STORAGE, refer to the *TDS Administrator's Guide*.

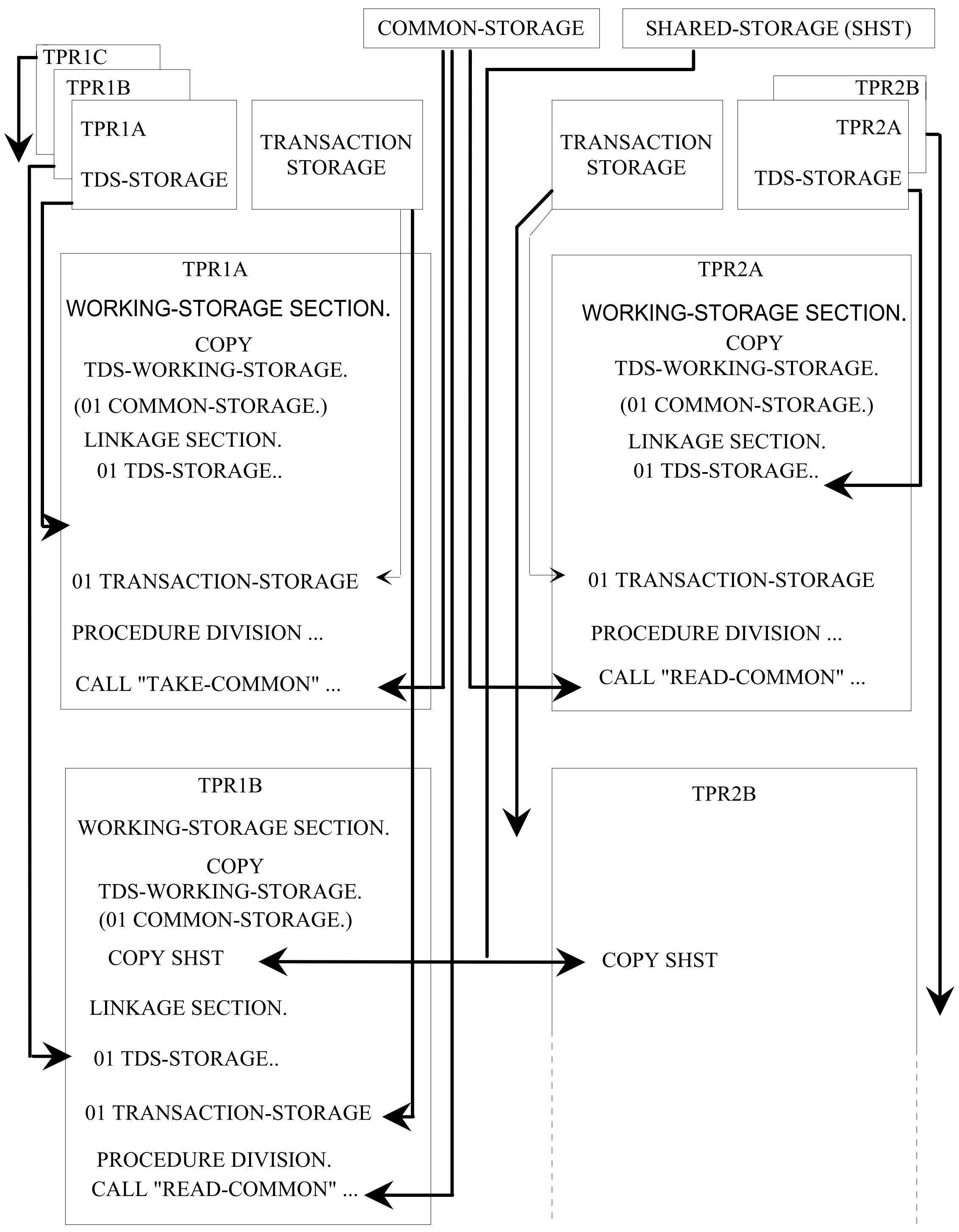


Figure 2-2. TDS-STORAGE, COMMON-STORAGE, TRANSACTION-STORAGE and SHARED-STORAGE

2.3.7 COMMUNICATION SECTION

A TPR that is to communicate with a correspondent must contain a COMMUNICATION SECTION.

The COMMUNICATION SECTION must include an input Communication Description (CD) if the TPR is to receive a message, and an output Communication Description (CD) if the TPR is to send a message.

This section contains CD entries for the input and output messages. The CD... FOR INPUT entry is used by RECEIVE statements, the `CD...FOR OUTPUT` entry by the SEND statement. The programmer must set up some of the fields within the entries before these statements are executed, some are set up by TDS during statement execution.

The CDs are also used for screen displays through FORMS. For FORMS, input and output CDs are used in their "alias" forms, that is, they have a new name.

For communication between TDS applications, refer to the *CPI-C/XCP2 User's Guide, or the Transactional Intercommunication Using XCP1 Protocol Manual.*

To connect to another application at the same site or on a different site, without having to log off from TDS, use the PT (pass through) command that is described in Chapter 14.

Syntax

```
COMMUNICATION SECTION.

  CD cd-name-1 FOR INPUT

  01 data-name-1                Re-defines input CD
  01 input-cd-alias             defines input-cd-alias for
                                FORMS

  CD cd-name2 FOR OUTPUT
  01 data-name2                 re-defines output CD
  01 output-cd-alias            defines output-cd-alias for
                                FORMS
```

2.3.7.1 INPUT CD

An input CD supplies information about the message to be received by the TPR. The format (below) is an adaptation of the standard COBOL format.

Syntax

```
CD cd-name FOR INPUT
     [SYMBOLIC QUEUE IS data-name-1]
     [MESSAGE DATE IS data-name2]
     [MESSAGE TIME IS data-name-3]
     [SYMBOLIC SOURCE IS data-name-4]
     [TEXT LENGTH IS data-name-5]
     [END KEY IS data-name-6]
     [STATUS KEY IS data-name-7]
     [MESSAGE COUNT IS data-name-8].
```

Data-name-1 is an input parameter. All the other parameters are output parameters.

A message can be input by more than one RECEIVE statement. END-KEY=3 means that the entire message has been received.

The RECEIVE statement can be issued only as the first action of the transaction and thereafter in a TPR subsequent to a SEND with EGI (End-of-Group Indicator). This verb is explained in Chapter 3.

Usage

The optional clauses can be written in any order.

If none of the optional clauses of the CD is specified, a level 01 data description entry must follow the CD description entry.

For each input CD, a record area of 87 contiguous characters is allocated. The record area is defined to TDS as follows:

- The SYMBOLIC QUEUE clause defines data-name-1 as the name of the data item whose implicit description is that of an elementary alphanumeric data item of 12 characters occupying positions 1-12 in the record. data-name-1 must contain the TDS name retrieved from TDS-STORAGE.
- A filler occupying positions 13-48 in the record.

- The MESSAGE DATE clause defines data-name2 as the name of a data item of 6 digits without an operational sign, occupying positions 49-54 in the record. At the time a RECEIVE statement is performed, data-name-2 contains the date in the form defined at TDSGEN.
- The MESSAGE TIME clause defines data-name-3 as the name of a data item whose implicit description is that of an integer of 8 digits without an operational sign, occupying positions 55-62 in the record. At the time a RECEIVE statement is performed, data-name-3 contains the time of day.
- The SYMBOLIC SOURCE clause defines data-name-4 as the name of an elementary alphanumeric data item of 12 characters occupying positions 63-74 in the record (the rightmost characters are padded to spaces if the length of the terminal address is less than 12 characters). It contains the terminal address.
- The TEXT LENGTH clause defines data-name-5 as the name of an elementary data item whose implicit description is that of an integer of 4 digits without an operational sign, occupying positions 75-78 in the record.
- The END KEY clause defines data-name-6 as the name of an elementary alphanumeric data item of 1 character occupying position 79 in the record.
- The STATUS KEY clause defines data-name-7 as the name of an elementary alphanumeric data item of 2 characters occupying positions 80-81 in the record.
- The MESSAGE COUNT clause defines data-name-8 as the name of an elementary data item whose implicit description is that of a 6-digit integer without an operational sign occupying positions 82-87 in the record.

Use of the above clauses results in a record whose implicit description is equivalent to the following:

```
01  inaccessible-data-name.
    02  data-name-1 PIC X(12).        (Symbolic Queue)
    02  FILLER      PIC X(36).
    02  data-name2  PIC 9(06).        (Message Date)
    02  data-name-3 PIC 9(08).        (Message Time)
    02  data-name-4 PIC X(12).        (Symbolic Source)
    02  data-name-5 PIC 9(04).        (Text Length)
    02  data-name-6 PIC X.            (End Key)
    02  data-name-7 PIC XX.           (Status Key)
    02  data-name-8 PIC 9(06).        (Message Count)
```

Record description-entries after an input CD implicitly re-define this record and must describe a record of exactly 87 characters. Multiple re-definitions of this record are permitted; however, only the first re-definition can contain VALUE clauses. TDS always references the record according to the data descriptions defined in number 3 above.

All data-names must be unique within the CD; any data-name can be replaced by the reserved word FILLER.

NOTES:

The data item referenced by data-name-1 must be set equal to the data item SYMBOLIC-QUEUE of TDS-STORAGE prior to execution of any RECEIVE or CDRECV statement. It is an input parameter.

The data item referenced by data-name-2, which is assigned a value by TDS as part of the execution of a RECEIVE or CDRECV statement, represents the date of completion of the message in the form defined at TDSGEN. The default format is YYMMDD, where YY is the year, MM is the month and DD is the day. It is an output parameter.

The data item referenced by data-name-3, which is assigned a value by TDS as part of the execution of a RECEIVE or CDRECV statement, represents the time of completion of the message. The format is hhmmsstt, where hh is hours, mm is minutes, ss is seconds and tt is hundredths of a second. It is an output parameter.

During execution of the RECEIVE or CDRECV statement, TDS sets the data item referenced by data-name-4 equal to the symbolic name of the terminal or the correspondent-name of the correspondent that is the source of the message.

When the source is a pseudo-terminal (that is, during batch processing), the data item referenced by data-name-4 is set equal to the name supplied by the pseudo-terminal at connection time (refer to Chapter 13 for details of programming batch interface programs). It is an output parameter.

The data item referenced by data-name-5 is set by TDS equal to the number of positions filled as a result of the execution of the RECEIVE or CDRECV statement. It is an output parameter.

If TDS detects the end of message, it sets the data item referenced by data-name-6 to 3; otherwise, it is set to 0. It is an output parameter.

The data item referenced by data-name-7 indicates the status of the executed RECEIVE, ACCEPT MESSAGE COUNT, or CDRECV for compatibility purposes with communications systems ENABLE INPUT or DISABLE INPUT statements. Status key values appear in Table 2-5. It is an output parameter.

The data item referenced by data-name-8 indicates whether a message is waiting for the current transaction. A value of 1 is affirmative and a value of 0 is negative. This data item is updated by TDS only as part of the execution of an ACCEPT statement with the COUNT option. It is an output parameter. ACCEPT is explained later in Chapter 4.

2.3.7.2 INPUT-CD-ALIAS for FORMS

Input and output CDs cannot be CALLed directly when using the FORMS utility. You must re-define the input and output CDs in the COMMUNICATION SECTION by a 01 structure. Such a re-definition is called an alias.

Syntax

```
CD cd-name-1 FOR INPUT.
01 cd-alias-1.
  02 symbolic-queue              PIC X(12).
  02 FILLER                      PIC X(50).
  02 symbolic-source             PIC X(12).
  02 FILLER                      PIC X(4).
  02 end-key                     PIC X.
  02 status-key                  PIC XX.
  02 FILLER                      PIC X(6).
```

The following fields of the input-cd-alias are used:

- "`symbolic-queue`" to contain the name of the queue.
- "`symbolic-source`" to contain the name of the correspondent.
- "`end-key`" to contain the enclosure level returned after the CDRECV procedure has executed as follows:

  0: Partial Message,
  1: Partial Message, FORMS in append mode,
  3: End of Message.

- "`status-key`" to contain the status returned on execution of various FORMS functions.

The input CD data structure is modified by the `REDEFINES` clause to obtain the input-cd-alias. An example of such an entry is as follows:

```
COMMUNICATION SECTION.

         CD cd-name FOR INPUT.
               SYMBOLIC QUEUE IS SYMQUEUE
               MESSAGE DATE IS MDATE
               MESSAGE TIME IS MTIME
               SYMBOLIC SOURCE IS SYMSOUR
               TEXT LENGTH IS TLENGTH
               END KEY IS ENDKEY
               STATUS KEY IS STATUS
               MESSAGE COUNT IS MCOUNT

         01 input-CD-alias
           02  RSYMQUEUE    PIC X (12).
           02  FILLER       PIC X(50).   to mask "mdate" and
                                                "mtime"
           02  RSYMSOUR     PIC X(12).
           02  FILLER       PIC X(4).    to mask "tlength"
           02  RENDKEY      PIC X.
           02  RSTATUS      PIC XX.
           02  FILLER       PIC X(6).    to mask "mcount"
```

The re-defining clause `(01 input-CD-alias)` causes `input-CD-alias` to share the same storage as `cd-name`.

2.3.7.3 OUTPUT CD

An output CD supplies information about a message to be sent by the TPR. The format (below) closely adheres to the standard COBOL format. No indexing or subscripting is currently possible, so there is no reference to a DESTINATION TABLE OCCURS clause.

Syntax

```
CD cd-name FOR OUTPUT.
     [DESTINATION COUNT IS data-name-1]
     [TEXT LENGTH IS data-name2]
     [STATUS KEY IS data-name-3]
     [ERROR KEY IS data-name-4]
     [SYMBOLIC DESTINATION IS data-name-5].
```

Input Parameters	Output Parameters
data-name-1 data-name-2 data-name-5	data-name-3 data-name-4

Usage

You can write the optional clauses in any order.

If none of the optional clauses is specified, a level 01 data description entry must follow the CD description entry.

For each output CD a record area of 23 contiguous characters is allocated. This record area is defined to TDS as follows:

- The DESTINATION COUNT clause defines data-name-1 as the name of the data item whose implicit description is that of an integer of 4 digits without an operational sign, occupying positions 1-4 in the record.
- The TEXT LENGTH clause defines data-name-2 as the name of an elementary data item whose implicit description is that of an integer of 4 digits without an operational sign, occupying positions 5-8 in the record.
- The STATUS KEY clause defines data-name-3 as an elementary alphanumeric data item of 2 characters occupying positions 9-10 in the record.
- The ERROR KEY clause defines data-name-4 as the name of an elementary alphanumeric data item of 1 character occupying position 11 in the record.

- The SYMBOLIC DESTINATION clause defines data-name-5 as the name of an elementary alphanumeric data item of 2 characters occupying positions `12-23` in the record.

Use of the above clauses results in a record whose implicit description is equivalent to the following:

```
01 inaccessible-data-name.
   02  data-name-1  PIC 9(04).    (Destination Count)
   02  data-name-2  PIC 9(04).    (Text Length)
   02  data-name-3  PIC XX.       (Status Key)
   02  data-name-4  PIC X.        (Error Key)
   02  data-name-5  PIC X(12).    (Symbolic Destination)
```

Record descriptions after an output CD implicitly re-define this record. Multiple re-definitions of this record are permitted; however, only the first re-definition can contain VALUE clauses. TDS always references the record according to the data descriptions defined in number 3 above.

All data-names must be unique within the CD; any data-name can be replaced by the reserved word FILLER.

NOTES:

During the execution of a SEND or CDSEND (or for compatibility with communications systems, an ENABLE OUTPUT or DISABLE OUTPUT) statement, the data item referenced by data-name-1 indicates to TDS the number of symbolic destinations that are to be used from the area referenced by data-name-5. Data-name-1 must equal 1. It is an input parameter.

During execution of a SEND statement, TDS interprets the data item referenced by data-name-2 as the user's indication of the number of leftmost positions (length) of the data to be transferred. It is an input parameter.

The data item referenced by data-name-3 indicates the status of the previously executed SEND, CDSEND, ENABLE OUTPUT, or DISABLE OUTPUT statement. Status key values appear in Table 2-5. It is an output parameter.

The data item referenced by data-name-4 is set by TDS to 0 after execution of a SEND, CDSEND, ENABLE OUTPUT or DISABLE OUTPUT statement except when the contents of the data item referenced by data-name-5 are invalid, in which case the data item referenced by data-name-4 is set to 1. It is an output parameter.

The data item referenced by data-name-5 represents the destination of the message. This "destination" may be the correspondent having initiated the transaction or any other correspondent. In order to send the message to the terminal or correspondent that initiated the transaction, the data item identified as SYMBOLIC SOURCE of the input CD must be moved to data-name-5 prior to execution of the SEND, CDSEND, or CDGET statement. It is an input parameter.

The rightmost four characters of data-name-5 are used to select an output device other than the default output device associated with the symbolic source. Possible values are as follows:

`PRTn` denoting printer n,
`CRTn` denoting display (cathode ray tube) n,
`DSKn` denoting disk(ette) n.

If there is only one device (of the specified type) attached to the identified destination, `"n"` is left blank. The default output device is identified by spaces occupying the four rightmost characters of data-name-5.

The programmer must ensure that the values of the data items referenced by data-name-1, data-name2, and data-name-5 are valid at the time of execution of the SEND statement.

The data items referenced by data-name-3 and data-name-4 are updated by TDS.

2.3.7.4 OUTPUT-CD-ALIAS for FORM

Like the input CD, the output CD cannot be called directly by FORMS. This means that an output CD must be re-defined.

```
CD cd-name2 FOR OUTPUT.
01 cd-alias2.
   02 FILLER                    PIC X(8).
   02 status-key                PIC XX.
   02 FILLER                    PIC X.
   02 symbolic-destination      PIC X(12).
```

The following fields of the output-cd-alias are used:

- "`status-key`" contains the status returned on execution of various FORMS functions.
- "`symbolic-destination`" contains the name of the correspondent.

Entries in the output CD data structure can be modified by implicit re-definitions (at level 01) to obtain the output-cd-alias. Only the first of multiple re-definitions can contain VALUES. An example of such an entry is as follows:

```
COMMUNICATION SECTION.

CD   CD-OUT FOR OUTPUT
     DESTINATION COUNT IS DCOUNT
     TEXT LENGTH IS TLENGTH
     STATUS KEY IS STATUS
     ERROR KEY IS ERROR
     SYMBOLIC DESTINATION IS SYMDEST.

01   output-cd-alias REDEFINES CD-OUT.
     02 RDCOUNT          PIC 9(4)    VALUE 1.
     02 FILLER           PIC X(4).           to mask "tlength"
     02 RSTATUS          PIC XX
     02 FILLER           PIC X               to mask "error"
     02 RSYMDEST.
        03 RTERMINAL     PIC X(8).
        03 RTYPE         PIC X(3).
        03 RNUMBER       PIC X.
```

In this example, RDCOUNT is set to 1, which is the only possible value under TDS.

The SYMBOLIC DESTINATION is split so that the TPR can select for output a slave terminal device. For instance:

```
MOVE SYMSOURCE TO RTERMINAL.
MOVE "PRT" TO RTYPE.
MOVE SPACE TO RNUMBER.
```

will select for output the (only) printer attached to the terminal that initiated the transaction.

2.4 Procedure Division

2.4.1 Purpose and Use

The Procedure Division is the last required division of a TPR. This division contains instructions that specify the data processing steps to be performed by the TPR.

The Procedure Division of a TPR is used in the same manner and has the same format as in a COBOL batch program. For most PROCEDURE DIVISION statements, the programmer should refer to the COBOL 85 Reference Manual. The following paragraphs are concerned with the exceptions that arise in coding a TPR Procedure Division.

2.4.2 Structure

The Procedure Division of the main (or only) program of a TPR must begin as follows:

```
PROCEDURE DIVISION USING TDS-STORAGE [ CONSTANT-STORAGE
                   [ PRIVATE-STORAGE ] [ TRANSACTION-STORAGE ]
].
```

If the TPR is composed of several programs, these storage areas can be passed from the main program to the others, as required.

2.4.3 Syntax Rules

Syntax rules apply as described in the *COBOL 85 Reference Manual*.

2.4.4 Exceptions to Normal Use

Because there is no message queuing in transaction processing, the communications statements ENABLE and DISABLE are irrelevant under TDS. To be compatible, however, the STATUS KEY field of the input CD is updated if the ENABLE statement or the DISABLE statement is specified with the INPUT phrase. The STATUS and ERROR KEY fields are updated if the ENABLE or DISABLE statements are specified with the OUTPUT phrase.

Similarly, the use of other procedure statements in a TPR would be at cross-purposes with the concept of transaction processing. For example, the use of the SORT statement, considering that the execution of a TPR must be completed in milliseconds, would be inappropriate. In such cases, the programmer must decide what constitutes a contradiction in the application of a given statement, because the compiler accepts all PROCEDURE DIVISION statements available in COBOL.

The actual procedures are explained in the following chapter.

2.4.5 Spawning a Transaction

Spawning a transaction means starting a transaction from within a transaction. The transaction that is started is called the spawned transaction. The spawned transaction is started when the commitment unit in which the spawning is performed ends normally. If the commitment unit calling a spawned transaction aborts, it is rolled back and the request for spawning is ignored. The spawning facility directs the spawned transaction to a correspondent (terminal) which can be the same correspondent (that is, the correspondent initiating the transaction that calls the spawn facility) or another. The recipient and the requester must have the necessary authority codes for using the spawned transaction.

The main use of spawning is for report printing on receive-only terminals, writing to diskettes, and reading from devices with no keyboard.

Processing performed by the spawned transaction is desynchronized from the transaction calling the spawn. In other words, the calling transaction can continue to do its own work and does not have to wait for the spawned transaction to start executing.

Spawning can take effect immediately (SPAWN, or SPAWNTX) or after a specified delay (DSPAWN, or SPAWNTX) or at a specified time of day (TSPAWN, or SPAWNTX).

The following CALL statements are used to spawn transactions (see Chapter 5 for more details):

```
CALL "DSPAWN" USING ....
CALL "SPAWN" USING .....
CALL "TSPAWN" USING ....
CALL "SPAWNTX" USING ....
```

The `CALL "SPAWNTX"` statement groups the functions of the `CALL "DSPAWN"`, the `CALL "SPAWN"`, and the `CALL "TSPAWN"`. Using this statement, you can:

- specify a correspondent name with a length up to 12 characters, instead of 8 in previous releases.
- specify the transaction message with a length of up to 130 characters, instead of 46 in previous releases.
- provide more detailed status reporting after the execution of the CALL "SPAWNTX".

The following CALL statement cancels spawned transactions:

```
CALL "DELSPAWN"
```

The CALL "DELSPAWN" statement deletes the pending spawned transactions attached to the current session.

Transaction spawning is

- either "Immediate":

  The spawned transaction is taken into account, that is, it becomes eligible for execution as soon as the spawning commitment unit terminates.

- or "Deferred":

  The spawned transactions are eligible for execution after a specified delay, or when a specified time of day has been reached.

A transaction eligible for execution may not execute if the TDS application is inactive, or if the correspondent is not available, for example, when TDS cannot connect it. In this case, the transaction executes as soon as the TDS application, or the terminal is reactivated.

When several eligible transactions have been spawned onto an active correspondent, TDS takes the turn. This means that as each spawned transaction is completed, the next one queued for this user is started. TDS does not give the turn back to the user until the queue of spawned transactions is empty.

2.4.5.1 Spawning Priorities

If you or other users direct several transactions to the same correspondent, the transactions are arranged in a queue that determines the order of spawning. When you spawn a transaction, you can specify a priority. The four priorities are as follows:

1. Express
2. High
3. Medium (default value)
4. Low

These priorities are used to manage the spawning queues. For each correspondent on which spawning has been requested, there are three associated queues:

HIGH
MEDIUM
LOW.

Table 2-1 shows how these queues are managed.

Table 2-1. Spawning Priorities

QUEUE / PRIORITY	High	Medium	Low
1 Express	LIFO		
2 High	FIFO		
3 Medium		FIFO	
4 Low			FIFO

LIFO (last in first out) means that the last transaction to be placed in the spawning queue is the first to be sent to the correspondent.

FIFO (first in first out) means that the first transaction to be placed in the spawning queue is the first to be sent to the correspondent.

As soon as the correspondent that is designated as a recipient of spawned transactions is idle, the three queues are scanned from HIGH to LOW to identify the first transaction to be started. For example, a spawned transaction that is placed in the "HIGH" queue with an express priority will be started before any other spawned transactions.

For spawned transactions that are deferred, the MEDIUM queue is chosen when the delay has expired or the specified time of day has been reached.

2.4.5.2 Spawning Towards Active and Passive Terminals

The correspondent to which the spawned transaction is directed may be active, or passive. Both these terms are defined in the *TDS Concepts Manual*.

On an active terminal, the spawned transaction is started when the terminal is in command mode, that is, no transaction is running. During the execution of the spawned transaction, the terminal receiving the spawned transaction can hold a conversation if it owns a keyboard, whether active or passive.

When TDS successfully connects a new correspondent, it is set to the passive state; otherwise the state of a correspondent remains unchanged.

Spawning does not affect a correspondent's state. If the correspondent is active, it is still active at the end of the spawned transaction and if it is passive, it remains passive.

In all cases, the spawned transaction is eligible for execution at the normal end of the commitment unit that requires the spawning.

2.4.5.3 Searching for a Correspondent

This section explains how TDS searches for a correspondent that has been specified as the recipient of the spawned transaction.

TDS searches for the name of the correspondent in the following order. Steps 2, 3, and 4 are performed only if the correspondent's name does not contain a star (*).

1. The TDS Correspondent Table, which contains the names of all correspondents known to TDS, including logged or abnormally disconnected correspondents.
2. If the correspondent is not found in step 1, the search for the correspondent's name continues in the TDS Mailbox Table, which contains the DSA addresses (session control and mailbox) of all correspondents known to TDS, including logged or abnormally disconnected correspondents.

   If the correspondent's name specified as a parameter of the SPAWN statement is an address instead of a name, the correspondent's name will be found in the second step.

   For the `CALL "SPAWNTX"` statement, the correspondent's name is taken as a DSA address with the following format:

   ```
   |X X X X|X X X X X X X X|
   |_______|_______________|
       A           B
   ```

   where:

   A represents the session-control name,
   B represents the mailbox name.

   For the `CALL "SPAWN", CALL "DSPAWN"`, and `CALL "TSPAWN"` statements, the correspondent's name is taken as a DSA address with the following format:

   ```
   |X X X X|X X X X bbbb|
   |_______|____________|
       A         B
   ```

where:

A represents the session-control name,
B represents the mailbox name, left aligned with four spaces (represented by b)

3. If the correspondent is not found in Step 1 and 2, the search continues in the list of correspondents defined at network generation.

   If the correspondent is found, it will be known to TDS.

4. If the correspondent's name is not found, the 8 characters (or 12 for CALL "SPAWNTX") given as the correspondent's name are taken as a DSA address with the format described in step 2 above.

   Then TDS tries to connect the correspondent at this address.

2.4.5.4 Using the Star (*) Convention

Spawning can be used to direct a transaction to a specific correspondent or to a correspondent belonging to a group of correspondents. A group is formed by use of the star convention in the correspondent's name. When a transaction is directed to a group of correspondents, the first available correspondent of the group is chosen for executing the spawned transaction. Spawning to a group of correspondents can give extra flexibility, for example, a report can be directed to any printer in the group.

The first parameter (data-name-1) of a spawn statement (SPAWN, DSPAWN, TSPAWN, SPAWNTX) can contain the name of a "group of correspondents" instead of the name a single correspondent. A group of correspondents have their names identical in one or more of their leftmost contiguous characters. The field specified by data-name-1 must contain the common part of the userid with a star ("*") to represent the characters of the userids which are not identical.

EXAMPLE

The group of correspondents WORKERA, WORKERB, WORKERC could be indicated by WORKER*. Neither AWORKER nor WORKA would be included in this group. The spawned transaction would be started for the first free correspondent to become available in the group. All correspondents in the group must have an authority code matching the transaction's authority code; otherwise a status value of 2 (if SPAWN, DSPAWN, or TSPAWN are used), or a status value of 43 (if SPAWNTX is used) may be returned in the status field defined in the spawning TPR.

When the star convention is used, the spawning mechanism cannot be a log-on procedure for a correspondent (see the previous section).

❑

2.4.5.5 DUMMY Correspondent

The user's name for spawning may be set to DUMMY. A dummy correspondent is (as its name implies) not a real correspondent, that is, it does not represent anything. Transactions spawned to such a correspondent cannot dialog with their correspondent.

2.4.5.6 Length of a Correspondent's Name

This section does not apply to the CALL "SPAWNTX" statement.

The first parameter of the CALL statement for spawning is an 8-character field that must contain the correspondent's name.

Some users may be connected to a TDS application with a name of more than 8 characters long (up to 12 characters).

Therefore it is not possible using these CALL statements to spawn using such a name because the parameter is limited to 8 characters.

All correspondents in the group must have an authority code matching the transaction's authority code; otherwise a status "2" (if SPAWN, DSPWAN, or TSPAWN) or "43" (if you use SPAWNTX) may be returned in the status field defined within the spawning TPR.

However, it is possible to do the following:

1. It is recommended that you use the CALL "SPAWNTX" statement that allows you to specify a correspondent's name of up to 12 characters in length.
2. The first 8 characters of the name may be specified as the spawn parameter. Then the correspondent's name is searched for in the TDS Tables as described previously, the name containing 8 characters padded with 4 blanks is searched for, if this name is not found, TDS adds a star as the ninth character in the case of the star convention and the search is repeated. The user's name containing more than 8 characters should be found this time.

EXAMPLE

The user whose name is STEPHENSON (10 characters long) may be used for spawning. The parameter to be specified in a CALL statement must be STEPHENS. If STEPHENS does not exist, TDS will find STEPHENSON.

It is important to know the first correspondent whose name begins with "STEPHENS" is chosen by TDS.

❑

2.4.5.7 Limits

The maximum number of spawned transactions in the same commitment unit is limited to 76. When this limit is reached, an abnormal status is returned ("11" for the CALL "SPAWNTX" statement, "1" for other statements).

The global number of spawned transactions waiting to be validated at any one time is 1,000. When this limit is reached:

- on the first attempt and if the transaction is executed for the first time, the transaction is rolled back and is restarted after 5 seconds.
- otherwise, an abnormal status is returned, that is, status "10" for SPAWNTX, and "1" for other statements.

2.4.5.8 Using the Transaction Initialization Routine for Spawning

The Transactional Initialization Routine is an optional user-written subprogram. This section describes how this routine is used for spawning. For further information on the Transaction Initialization Routine, see Chapter 12.

The transaction initialization routine (if any) is called on a spawn statement to analyze the spawning message. This routine is activated twice:

- first, with the identification of the requester of spawning as an input parameter to control the access rights of the requester to the spawned transaction.
- second, with the identification of the recipient to control the access rights of the recipient to the spawned transaction. This second call to the initialization routine is performed later at the start of the spawned transaction.

2.4.6 Chaining TPRs

TDS loads the first TPR of the transaction in response to the first message entered at the terminal.

Each TPR is called by TDS when the preceding TPR terminates. A TPR specifies its successor by setting the PROGRAM-ID of the successor in NEXT-TPR of TDS-STORAGE.

On normal termination of the current TPR, the next TPR is loaded and executed. The currently executing TPR may specify itself as the NEXT-TPR.

If the NEXT-TPR contains spaces, the transaction terminates at the end of the current TPR. Each executing TPR "knows" its position in the chain by PRIOR-TPR and NEXT-TPR of TDS-STORAGE.

If the NEXT-TPR contains "BYE", the terminal is logged off immediately.

If the NEXT-TPR contains "BYEWEAK", the terminal is logged off:

- immediately only if there are no outstanding transactions eligible for immediate spawning to that terminal,
- as soon as the terminal returns to command mode if an interrupt context (LOGON, BREAK) exists for the user concerned.

Note that "BYEWEAK" is valid only for the duration of the current VCAM session and is no longer in effect after a `CANCEL_JOB` command is issued, or after a system crash has occurred.

("BYEWEAK" is useful for sharing a printer among different applications).

The TPR can specify the PROGRAM-ID of an "abort" TPR in the ON-ABORT-TPR field in TDS-STORAGE.

When a TPR aborts, ABORT-CODE is set giving the reason and the "abort" TPR is executed. The "abort" TPR can chain to other TPRs.

When a transaction aborts, there are two cases to consider:

- if the ON-ABORT-TPR is not specified, the TDS-supplied ON-ABORT-TPR is performed; this abnormally terminates the transaction and sends a message to the initiator of the transaction; if the initiator of the transaction is a printer or a XCP1 application, no message is sent,
- if the ON-ABORT-TPR is specified, the ON-ABORT-TPR is executed while the ON-ABORT-TPR field in the TDS-STORAGE is reset to blanks; this process may be executed 3 times, thus allowing 3 aborts to occur.

2.4.7 Message Handling Without FORMS

The Procedure Division statements used for data communications in COBOL are RECEIVE for input and SEND for output. The RECEIVE verb makes an incoming message available to a TPR. Details are given in Chapter 3. The SEND verb releases a message to the destination specified in the SYMBOLIC DESTINATION of the output CD. The SEND verb is explained in Chapter 3.

To input a message from a terminal, the programmer must provide the CD SYMBOLIC QUEUE with data from the TDS-STORAGE location SYMBOLIC-QUEUE. The RECEIVE statement can then be executed with the input message being placed in a designated location, for example:

```
  RECEIVE CD-INPUT MESSAGE INTO MESSAGE-AREA
```

where CD-INPUT is the cd-name defined in the COMMUNICATION SECTION and MESSAGE-AREA a data area defined in the WORKING-STORAGE SECTION of the TPR. In this input operation TDS will load fields in the CD entry giving such information as the number of characters in the message and the time the RECEIVE statement is processed.

After a RECEIVE statement, 'END KEY' can be tested to ensure that the complete message has been received.

- 0 (it is not the end of the message),
- 3 (it is the end of the received message).

To send a message to a terminal, the programmer must set up parameters in the output CD giving:

- 1 to DESTINATION COUNT (if not initialized in the description),
- the identifier of the destination,
- the number of characters in the message,

and then issue a SEND statement.

This statement has several formats and thus several effects:

Format 1	SEND cd-name FROM identifier-1.
Format 2	SEND cd-name [FROM identifier-1] WITH ESI.
Format 3	SEND cd-name [FROM identifier-1] WITH EMI.
Format 4	SEND cd-name [FROM identifier-1] WITH EGI.

Format 1

SEND cd-name FROM identifier	in this form the statement releases a portion of a message to a TDS buffer without physical transmission to the terminal. The message portion is held in 'quarantine' and is referred to as part of a 'quarantine unit' (see Figure 2-3).

Format 2

SEND cd-name FROM identifier WITH ESI	this has the same effect as the previous statement except that it allows in addition the use of the ADVANCING option. (ESI stands for 'End of Segment Indicator'.)

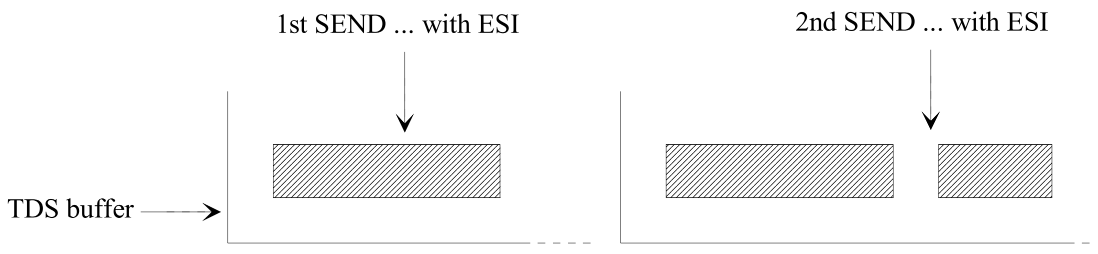


Figure 2-3. Message Buffering with ESI

In neither case will the message be physically transmitted until a SEND ... WITH EMI or EGI is executed; these indicators on the SEND statement validate the quarantine unit prior to its transmission. Any quarantine units not validated upon normal completion will not be transmitted when the TPR terminates.

Format 3

SEND cd-name [FROM identifier] WITH EMI	execution of this statement (EMI means End of Message Indicator) transfers the contents of the message area to the TDS message buffer.

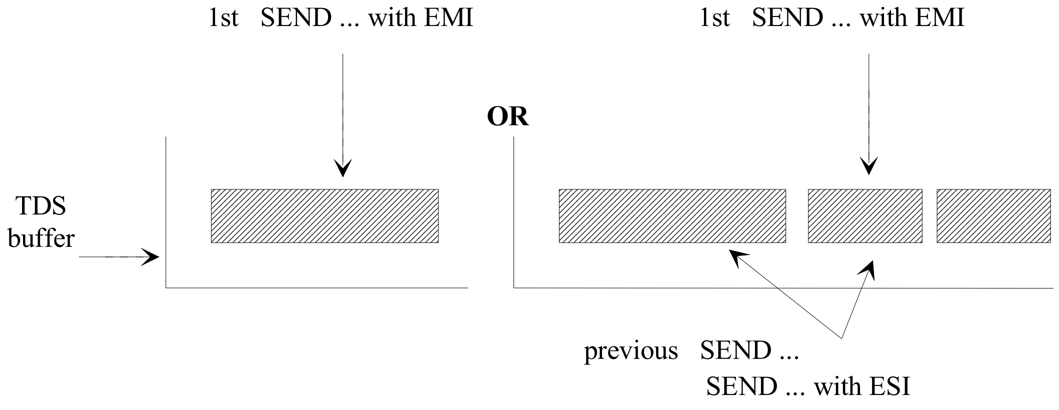


Figure 2-4. TDS Message Buffering with EMI

In Format 1 and Format 2, the physical transmission of the quarantined data takes place at the normal completion of the TPR.

Format 4

SEND cd-name [FROM identifier] WITH EGI	this SEND statement (EGI means End of Group Indicator) causes the message to be transmitted to the terminal and indicates that a response is required from the terminal. In this case the next data communication statement must be RECEIVE which will be issued in the next TPR. It is also used as the last SEND statement of a transaction in which case no response is expected but the operator may enter a new transaction. The statement may refer only to the terminal that activated the transaction.

NOTES on the 4 formats

In the same TPR, the next SEND statement (any format) will transmit validated message segments already in the buffer to the terminal. The buffer is then filled with the new message.

A SEND statement after a SEND WITH EMI statement will degrade performance because the executing task must wait for the SEND WITH EMI statement to be completed.

After a SEND statement, it is recommended that STATUS KEY be tested.

Output Message

If there is only one message to be transmitted by a TPR, it is done asynchronously. If a TPR issues a SEND statement and can continue to do its own work, this is called asynchronous transmission.

Synchronous means that the TPR stops executing from the time a SEND statement is issued until the transmission has been completed. In other words the TPR is synchronized to the completion of the SEND statement. If several messages are sent by a TPR using the SEND with EMI (end-of-message indicator) statement,

- each message is physically and synchronously transmitted by the next SEND except the last,
- the last message is transmitted asynchronously.

EXAMPLE

```
    TPR1
       WORKING-STORAGE SECTION.
       01 OUT-MESSAGE-AREA.
          03 CCI          PIC X.
          03 C2           PIC X.
          03 TITLE1       PIC X(9)     VALUE "TOWN-NAME".

       COMMUNICATION SECTION.

          CD CD-OUT FOR OUTPUT
             DESTINATION COUNT IS DCOUNT

             TEXT LENGTH IS TLENGTH
             STATUS KEY IS SKEY
             ERROR KEY IS EKEY
             SYMBOLIC DESTINATION IS SYMDEST.

          CD CD-IN FOR INPUT
             SYMBOLIC QUEUE IS SYMQUEUE
             MESSAGE DATE IS MESDATE
             MESSAGE TIME IS MESTIME
             SYMBOLIC SOURCE IS SYMSOURCE
             TEXT LENGTH IS TEXTL
             END KEY IS EIKEY
             STATUS KEY IS SIKEY
             MESSAGE COUNT IS METCOUNT.

       LINKAGE SECTION.

          COPY CONSTANT-STORAGE.

       PROCEDURE DIVISION USING TDS-STORAGE, CONSTANT-STORAGE
...
          RECEIVE ...
   1.     MOVE 1 to DCOUNT
   2.     MOVE FF TO CC1.
          MOVE BLK TO C2.
   3.     MOVE SYMSOURCE TO SYMDEST.
   4.     MOVE 11 TO TLENGTH.
   5.     SEND CD-OUT FROM OUT-MESSAGE-AREA WITH EGI
               AFTER ADVANCING 0 LINES.
   6.     IF SKEY NOT EQUAL "00"...
   7.     MOVE "TPR2" TO NEXT-TPR.
          EXIT PROGRAM
```

```
     TPR2
        COMMUNICATION SECTION.

           CD CD-IN FOR INPUT
              SYMBOLIC QUEUE IS SYMQUEUE
              MESSAGE DATE IS MESDATE

              MESSAGE TIME IS MESTIME
              SYMBOLIC SOURCE IS SYMSOURCE
              TEXT LENGTH IS TEXTL
              END KEY IS EIKEY
              STATUS KEY IS SIKEY
              MESSAGE COUNT IS METCOUNT.

       WORKING-STORAGE SECTION.
       01 IN-MESSAGE-AREA.
          03 TOWN-NAME PIC X(25).

       PROCEDURE DIVISION USING TDS-STORAGE, CONSTANT-STORAGE...
    8.     MOVE SYMBOLIC-QUEUE TO SYMQUEUE
           MOVE SPACES TO IN-MESSAGE-AREA.
           RECEIVE CD-IN INTO IN-MESSAGE-AREA.
    9.     IF EIKEY NOT EQUAL "3" ...
```

TPR1 displays "TOWN-NAME" as a prompt to the user to key in "town-name". TP2 is activated as soon as "town-name" is entered.

1. Sets DCOUNT to 1.
2. The control codes FF and BLK clear the screen and cause the field TITLE1 to blink. See Table 2-2.
3. The output CD is to contain the identifier of the activating terminal. In this example, SYMDEST is set to the value of the SYMSOURCE field which has been initialized by the previous RECEIVE statement.
4. The message length is set to 11.
5. SEND transfers the message to the terminal.
6. The status of SEND is tested.
7. TPR1 indicates to TDS that TPR2 is to follow.
8. TPR2 loads the input CD SYMBOLIC-QUEUE with the contents of the TDS-STORAGE SYMBOLIC-QUEUE.
9. The 'END KEY' is tested to determine if the entire message is received.

❑

Table 2-2. DKU7007 Control Codes

EBCDIC value	Control Code	Meaning
OC	FF	Line mode - clear screen Format mode - clear variable fields
OD	CR	Return to beginning of current line
25	LF	Move cursor 1 line down
OD25	NL	Return to beginning of next line
5F	BLK	Characters after 5F blink until next space, end of line, or start of variable field
13	DC3	Position cursor at address given by the next two characters (line and column code - see note below)
27D4	SCM	Change to format mode
27D5	SCN	Change to line mode
1C	FS	Start of fixed field (terminal must be in line mode when this character is received)
1D	GS	Start of variable field (VIP must be in line mode when this character is received). Field is terminated by the FS character.
3C	DC4	Reset cursor to start of first variable field (Terminal must be in format mode)
05	HT	Line mode - set cursor to next tabulation Format mode - set cursor to start of next variable field

Table 2-3. DKU7007 Line and Column Codes

Line/Col number	Graphic Symbol	EBCDIC Code	COBOL collating	Line/Col number	Graphic Code	EBCDIC value	COBOL collating
1		40	65	41	H	C8	201
2	!	4F	80	42	I	C9	202
3	"	7F	128	43	J	D1	210
4	#	7B	124	44	K	D2	211
5	$	5B	92	45	L	D3	212
6	%	6C	109	46	M	D4	213
7	&	50	81	47	N	D5	214
8	'	7D	126	48	O	D6	215
9	(	4D	78	49	P	D7	216
10	)	5D	94	50	Q	D8	217
11	*	5C	93	51	R	D9	218
12	+	4E	79	52	S	E2	227
13	,	6B	108	53	T	E3	228
14	-	60	97	54	U	E4	229
15	.	4B	76	55	V	E5	230
16	/	61	98	56	W	E6	231
17	0	F0	241	57	X	E7	232
18	1	F1	242	58	Y	E8	233
19	2	F2	243	59	Z	E9	234
20	3	F3	244	60	[	4A	75
21	4	F4	245	61		E0	225
22	5	F5	246	62	]	5A	91
23	6	F6	247	63	^	5F	96
24	7	F7	248	64	_	6D	110
25	8	F8	249	65	è	79	122
26	9	F9	250	66	a	81	130
27	:	7A	123	67	b	82	131
28	;	5E	95	68	c	83	132
29	<	4C	77	69	d	84	133
30	=	7E	127	70	e	85	134
31	>	6E	111	71	f	86	135
32	?	6F	112	72	g	87	136
33	@	7C	125	73	h	88	137
34	A	C1	194	74	i	89	138
35	B	C2	195	75	j	91	146
36	C	C3	196	76	k	92	147
37	D	C4	197	77	l	93	148
38	E	C5	198	78	m	94	149
39	F	C6	199	79	n	95	150
40	G	C7	200	80	o	96	151

The control code is specified in the VALUE clause by its COBOL collating sequence. The COBOL collating sequence is the decimal conversion of the EBCDIC value+1.

The GS control code is followed by a second character defining the nature of the variable field: whether it is to contain numeric data, whether it can be transmitted, whether it can be printed locally. Table 2-4 gives the settings of this control character. For example, if GS is followed by F4, this means that the variable field:

- is numeric,
- can be transmitted,
- can be printed locally.

Table 2-4. GS Field Protection Codes (DKU 7007)

Numerics only	Transmission allowed	Printing allowed	Hexadecimal value	COBOL collating sequence
N	Y	Y	F0	241
N	Y	N	F1	242
N	N	Y	F2	243
N	N	N	F3	244
Y	Y	Y	F4	245
Y	Y	N	F5	246
Y	N	Y	F6	247
Y	N	N	F7	248

Y = YES
N = NO

2.4.8 Message Handling with FORMS

This section shows how FORMS handles messages (see also earlier in this chapter). For an introduction to FORMS, see the *IOF Programmer's Manual.*

Access to forms is given through the Standard Device Programmatic Interface (SDPI) statements. These statements provide a basic way to dialog with an endpoint, that is, to a logical terminal as seen by the program.

Each statement corresponds to an interaction with the terminal, except the CDFIDI statement that is purely informative. This action may result

- either in issuing some kind of message for the terminal (CDGET, CDSEND, CDATTR, CDATTL, CDMECH, CDRELS),
- or in receiving some message (CDRECV, CDPURGE).

For each action of the first kind, the program can specify an enclosure level in the statement which is used to specify when the message is to be delivered to the terminal and, if it is to be delivered immediately, whether the application keeps the turn, (that is, the right to send other messages) or gives it to the terminal.

For a RECEIVE statement, the program gives the name of a queue on which to receive. The statement returns an enclosure level that indicates to the program whether it has more to receive to get the whole message and where the turn is. For each statement, the status key indicates whether the statement is successfully completed.

SDPI procedures are called in the TPR

- to activate the form,
- to send and receive data.

2.4.8.1 Activating a Form

Activating a form consists of displaying the fixed fields and the initial values and attributes (for example, a protected field, highlighted field) of the variable fields on the screen.

To activate a form, the TPR must call a CDGET procedure referencing a structure, called form-nameI. This structure identifies a form, checks its compilation date and specifies its mode of activation. A form can be activated in one of four modes:

- APPEND,
- OVERLAY,
- WINDOW,

- ERASE.

APPEND Mode

The user may specify where to mount the form on the screen. The place that the form occupies on the screen is in relation to the forms already mounted. When a new form is activated, all the forms below the form to which the new form is to be appended are implicitly released. The area occupied by these forms on the screen is cleared.

When a request is made to mount a form at the top of the screen, the whole screen is cleared. Several occurrences of the same form may be simultaneously mounted on the screen. Each occurrence is identified by an occurrence number specified in the form-nameI and form-nameV structures. Recall that a form-nameV is used:

- to select individual fields of a form for sending or receiving their contents,
- to modify their attributes,
- or to return the status of a field.

OVERLAY Mode

All the preceding forms are logically released but the screen is not cleared. The new form overwrites what was displayed.

WINDOW Mode

The WINDOW mode is a mode where each form defines a window on the screen. The window consists of the smallest rectangle that may contain the form beginning in line 1 column 1 up to the maximum line and the maximum column.

The program specifies in the form identification structure the location of the window by giving the coordinates of its top left corner. All the forms previously on the screen are frozen, as in OVERLAY mode. Then the contents of the window that will contain the form are erased. The form is placed in the window and it becomes the new active form.

When the CDGET procedure references a form that is already displayed in a window, this form becomes the active form again with its fields restored to their contents when the form was the active form. In this case, the form is always displayed at the same location on the screen and the window coordinates that may be specified are ignored. This means that you can view forms as a stack of paper sheets that partially overlay each other and where the user may remove a sheet in the middle of stack to put it on the top of the stack.

You may use the POPUP mechanism to release only the currently active form. In this case, the form window is reset to its underlying contents and the next form in the stack becomes the new active form.

ERASE Mode

The ERASE mode is similar to the WINDOW mode except that ERASE mode first releases all forms and clears the screen. A form may be activated in the WINDOW mode only if the previous form was activated in one of the following modes: ERASE or WINDOW. A form cannot be appended to a form activated in WINDOW or ERASE mode.

If you wish to activate a form in either the WINDOW or the ERASE mode, the TERM parameter must be set to ANY in the `MAINTAIN_FORM` command.

You can mount the object form:

- either from the binary libraries or the UFAS-EXTENDED file assigned to the TDS step,
- or from the terminal diskette.

FORMS either sends the object form or requests that the terminal mount the form from the diskette, depending on the options specified when the form was generated. When you activate the MAINTAIN_FORM utility, the terminal mounts the form when a MOVE to the diskette has been executed for the form in question. In this case, the clause "FORM IS" must be declared at TDSGEN. For more information, see the *IOF Programmer's Manual*.

If the form is not present on the diskette and the terminal is a DKU7007, DKU7107 or DKU211, the terminal notifies FORMS that the form cannot be activated. FORMS then sends the object form from its library or UFAS file to the terminal.

For VIP7760 terminals, an unsuccessful activation of the form from a diskette can cause the terminal to disconnect.

2.4.8.2 Sending Data

The following procedures cause a message to be output to a terminal:

CDSEND	sends some variable fields of a form. A selection vector is passed to identify the form and to specify the fields that must be modified.
CDATTR	sets special effects (for example, highlights fields) on some fields of a form. The form and the fields within the form are identified by a selection vector. If the form has been created with the SUBSTITUTE ATTRIBUTES option, a replacement attribute is selected whenever an attribute is specified that does not exist on the terminal, for example, if Reverse Video is requested for a VIP7760 terminal, Blink is selected.
CDATTL	is similar to CDATTR except that it sets a list of attributes instead of only one attribute.
CDMECH	initiates an "one-shot" mechanism that affects the whole device, such as clearing all unprotected fields or setting the alarm.
CDRELS	releases all the active forms and sets the terminal to normal mode. CDRELS does not clear the screen. If the screen is to be cleared, CDRELS must be followed by a CDMECH with the RESET option.

For a complete description of each procedure, refer to Chapter 10.

2.4.8.3 Receiving Data

When the terminal user enters data on the screen, the TPR must issue as many CDRECV statements as there are forms for which data has been entered. In such a case, it is recommended that a CDFIDI statement precede each CDRECV statement. This procedure returns the identification of the next form that is used to receive data.

The selection vector that the TPR gives when it calls the CDRECV statement identifies the form and selects the fields required in this form.

If a function key field has been declared in the `MAINTAIN_FORM` utility, pressing the function key will fill the field with the rank of the function key. Mapping function keys onto ranks is described in the *IOF Programmer's Manual*.

If the message contains some more information for another form, the TPR is notified through an appropriate enclosure level. If the TPR does not want to receive the pending data, it must call CDPURGE.

For a complete description of the CDRECV, CDFIDI, and CDPURGE procedures, see Chapter 10. An example of a COBOL transaction using FORMS is given in Appendix C.

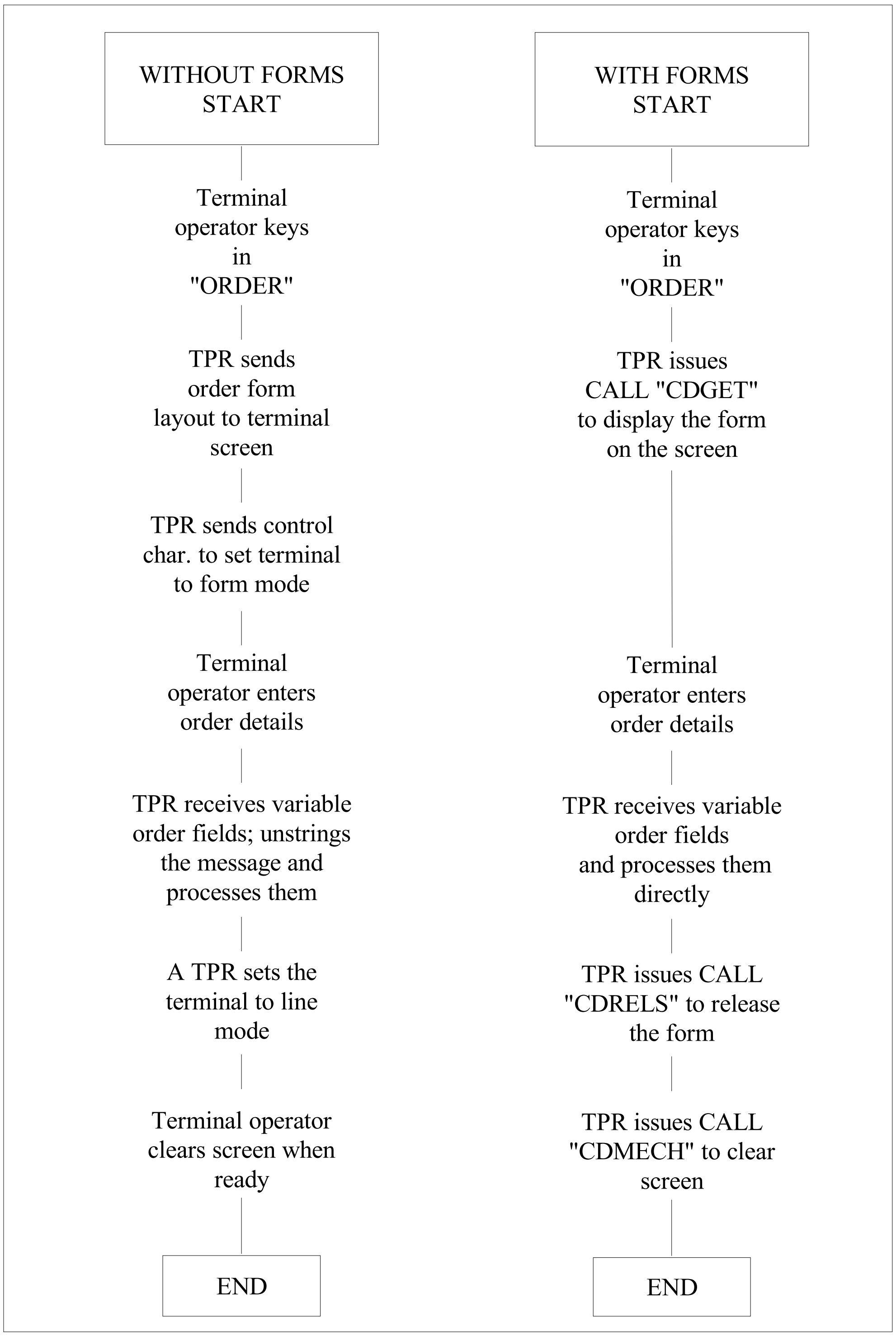

WITHOUT FORMS START
Terminal operator keys in "ORDER"
TPR sends order form layout to terminal screen
TPR sends control char. to set terminal to form mode
Terminal operator enters order details
TPR receives variable order fields; unstrings the message and processes them
A TPR sets the terminal to line mode
Terminal operator clears screen when ready
END
WITH FORMS START
Terminal operator keys in "ORDER"
TPR issues CALL "CDGET" to display the form on the screen
Terminal operator enters order details
TPR receives variable order fields and processes them directly
TPR issues CALL "CDRELS" to release the form
TPR issues CALL "CDMECH" to clear screen
END

Figure 2-5. Order Processing - Sequence of Operations

2.4.9 Character Sets

The character sets used with FORMS are the Pluri Lingual West (PLW) and the C101 character sets. There are fully described in the *IOF Programmer's Manual*.

The PLW character set is a set of graphic characters for writing texts using the Latin alphabet. There are two representations of the PLW character set:

- indirect code:

  The interchange code which is in keeping with the international standard ISO 6937 and is available on terminals such as DKU7107 and DKU211. Some characters, such as the accented letters, are represented by more than one byte.

- direct code:

  A proper superset of international EBCDIC code (the international EBCDIC code being also known as C101 code). This code is supported for FORMS by GCOS 7.

A TPR using FORMS can send and receive all PLW characters of the direct EBCDIC PLW code. See Figure 2-6.

If the terminal supports PLW, it is assumed to operate in this mode and FORMS translates from/to the interchange code to/from the direct code. Whenever possible, FORMS forces the terminal to operate in this mode.

If the terminal does not support PLW, FORMS converts from the PLW code sent by the program into the C101 code where accented letters are replaced by the same non-accented letters. The main purpose of the conversion is to avoid sending invalid characters to the terminal and to provide at least an understandable text.

When a form is generated or reloaded, the `MAINTAIN_FORM` utility, checks whether it contains PLW characters that do not belong to the C101 common subset. If so, the object form member is flagged as being of PLW type.

When a form of PLW type is activated through the CDGET procedure, the object form is scanned in order to translate the PLW characters into:

- either interchange code (indirect) if the terminal supports interchange code,
- or C101 characters if the terminal does not support interchange code.

When a form of PLW type is moved to a diskette by the `MAINTAIN_FORM` utility, the form is always translated into the interchange code.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0					SP	&	-	∅	∅	°	µ		{	}	\	0
1						é	/	É	a	j	~	£	A	J		1
2					â	ê	Â	Ê	b	k	s		B	K	S	2
3					ä	ë	Ä	Ë	c	l	t		C	L	T	3
4					à	è	À	È	d	m	u		D	M	U	4
5					á	í	Á	Í	e	n	v	§	E	N	V	5
6					ã	î	Ã̂A	Î	f	o	w	q	F	O	W	6
7					å	ï	Å	Ï	g	p	x		G	P	X	7
8					ç	ì	ç	Ì	h	q	y		H	Q	Y	8
9					ñ	ß	Ñ	`	i	r	z		I	R	Z	9
A					[	]	│	:		a̲	¡					
B					.	$	,	#		o̲	¿		ô	û	Ô	Û
C					<	*	%	@		œ			ö	ü	Ö	Ü
D					(	)	___	'				¨	ò	ù	Ò	Ù
E					+	;	>	=		Æ		'	ó	ú	Ó	Ú
F					!	^	?	"	±	¤			õ		Õ	

Figure 2-6. EBCDIC/PLW CODE

2.4.10 Printing

Before issuing a SEND statement, the TPR moves the name of the printer to SYMBOLIC DESTINATION in the output CD.

The printer has its set of control codes. However, the ADVANCING phrase specified in the SEND statement performs the same function as Line Feeds (LFs).

You are advised to use spawning.

EXAMPLE 1

Transaction X outputs to a printer with a keyboard. For as long as the printer continues to print, the operator cannot initiate a new transaction on it.

```
TRANSACTION X

       RECEIVE CD-IN MESSAGE INTO IN-BUFFER

       produce required data

           SEND CD-OUT FROM OUT-BUFFER WITH EGI

           EXIT PROGRAM.
```

EXAMPLE 2

Transaction Y generates an output. However instead of outputting to terminal A, transaction Y "spawns" transaction Z which outputs to a printer configured on terminal A. Terminal A can therefore continue initiating new transactions since it does not receive the print output.

```
TRANSACTION Y
initiated at
terminal A

RECEIVE CD- IN MESSAGE INTO IN- BUFFER

produce required data

WRITE records to user file

MOVE "NETWPRT3" TO Z1
                                      USERID must be in catalog:
         ---------------------------> USERID = site name + terminal name
                                                NETW        PRT3

MOVE "Z" TO Z2
                                      name of transaction to be activated
         ---------------------------> (as in a normal message, additional
                                      parameters are possible)

CALL "SPAWN" USING Z1, Z2, Z3

                                 ---> status indicator

IF Z3 NOT = ZERO ...---------------> if spawning is not successful, enter
                                      recovery procedure
continue processing

TRANSACTION Z
spawned on
printer                               message buffer contains transaction
                                      name "Z" (+ optional parameters)

RECEIVE CD-IN MESSAGE INTO
IN-BUFFER NO DATA GO NEXT

                                      (further activations do not have an
NEXT.                                 input message)

      READ records from user file
      SEND CD-OUT FROM OUT-BUFFER WITH EGI
      IF NOT "END-OF-FILE" MOVE
      CURRENT-TPR TO
      NEXT-TPR                         loop back on same TPR

      CALL "DFCMIT"

      EXIT PROGRAM.
```

❑

If a printer is receive-only (no keyboard), a special attribute is declared at network generation.

Spawning Transactions on a Receive Printer via the Datanet

To ensure that all messages are received, it is recommended that only one message per TPR be sent through use of the SEND EGI statement. To execute all the SEND statements in the transaction, include two TPRs with:

- a SEND EGI statement in the first TPR,
- a RECEIVE NO DATA at the beginning of the second TPR.

The SEND EGI statement hands over control to the Datanet that signals to TDS that the message has been printed. Then, TDS activates the second TPR with the RECEIVE NO DATA statement.

(See transaction Z in Example 1).

2.4.11 Report Handling Using GTWriter

The Generalized Terminal Writer (GTW) allows any hardcopy terminal in a communications network to be used as a printer remote from the central site but local to the user, that is any TDS user can request the printing of outputs or files at any terminal known to GTWriter. This section gives an overview: for more information, see Chapter 11 and the *Generalized Terminal Writer User's Guide*.

GTWriter creates and handles reports.

The writer report is generated by first calling `H_TW_USTARTE` to open a subfile in SYS.TW.OUT.

The control record is also created.

The report will be scheduled at the next commitment.

Commitment units in which a report is being created can run concurrently with each other.

Figure 2-7 shows 3 transactions running concurrently. In transaction X, 2 reports are created, whereas in transactions Y and Z one report is created in each transaction.

For this example, the following are assumed:

- a simultaneity of 4 is in effect,
- the SYS.TW.OUT file is declared a non-controlled file at TDSGEN,
- the assignment statement (`ASSIGN SYS.TW.OUT SHARE=DIR`) is included in the JCL for starting the particular TDS application.

When an abort occurs during a commitment unit, the report is not queued, but the affected commitment unit is re-executed and the report is rescheduled.

Note : A other file than SYS .TW.OUT may be used by suffixing the EFN by one character chosen in set (0..9,A..Z).

Examples: SYS.TW.OUT4 or SYS.TW.OUTB .
(see parameter MULTI_SYS_TW_OUT in GTWRITER User's Guide 47 A2 55 UU).

If the file SYS.TW.OUT(i) is closed or has changed between the start of the report and the commitment, the transaction is aborted either with the return code ″NOTOPENS″on WRITE primitive or with the return code ″EFNERR″ (if closed and deassigned) or "NOTOPEN″ (if closed and not deassigned) at commitment time. If the file has been closed then reopened between the last writing TPR and the committing one, the transaction is aborted by TDS with the return code "NOTOPENS".

Transaction X	Transaction Y	Transaction Z
TPR1	TPRa	TPRi
.	CALL « H_TW_USTARTE »	CALL « H_TW_USTARTE »
.	WRITE	End of TPRi
.	.	TPRm
CALL « H_TW_USTARTE »	.	WRITE
WRITE	.	.
WRITE	.	.
.	.	.
.	.	WRITE
Commitpoint is taken	End of TPRa	End of TPRm
TPRn	TPRc	TPRI
CALL « H_TW_UGETR »	.	.
.	.	.
.	.	.
CALL « H_TW_USTARTE »	WRITE	WRITE
WRITE	End of TPRc	End of TPRI
Commitpoint is taken	Commitpoint is taken	Commitpoint is taken
END of Transaction X	END of Transaction Y	END of Transaction Z

Figure 2-7. Creating GTWriter Reports

2.4.12 Terminal Adapter

The Terminal Adapter adapts messages to terminals supported by a TDS application. With the Terminal Adapter services (an option to be specified in TDSGEN), you can:

- adapt messages to your terminal in input and output (including the IBM 327x terminals),
- support foreign character sets,
- modify a user profile that contains variables for adapting screen presentation to the transactions running for the user (the procedure is described in Chapter 7).

You can use the Terminal Adapter for TM correspondents only.

"Old" TPRs using no specific terminal devices remain unchanged.

The Terminal Adapter cannot adapt:

- messages for slave terminals,
- messages sent from a transaction set to FOR INQUIRY,
- messages directed to the Batch Interface.

In these three cases, the messages are sent in the same way as previous releases without Terminal Adapter.

Using the Terminal Adapter in a TDS application does not support:

- the continuation character at the end of a line (-),
- the synchronization characters at the end of a page (+++),
- the function keys.

2.4.12.1 External Messages

External messages (in contrast to messages sent by a user's currently executing transaction) mean those delivered to a user by:

- the Master operator using the commands `[ M ] SEND_TDS_USER` or `[ M ] MODIFY_TDS_MOT`,
- the TDS monitor as service messages,
- a transaction running for another correspondent (who sends a message to an explicit destination).

2.4.12.2 Line Mode/Format Mode

When a user first logs on, the terminal is operating in line or format mode.

The mode is selected by a TPR as follows:

- The terminal is initially placed in line mode when the user logs onto the TDS application.
- To change to format mode, the TPR must call a CDGET statement and a form is activated.
- To return to line mode, the TPR must call a CDRELS statement.
- Either of these modes can also be selected via the options specified at TDSGEN.

A TPR can receive or send formatted messages only to a terminal operating in format mode.

2.4.12.3 Terminal Adapter and Free Presentation

When you specify the USE TERMINAL ADAPTER clause in TDSGEN, no messages are adapted to a user's terminal until the program modifies the individual user's profile (by setting the `TA` variable to 1 in the `CALL "MDPROF"` statement). Until this happens, the user is said to be in "free presentation" mode.

You can use the Terminal Adapter to send external messages to an active form, or to a terminal operating in line mode. External messages sent via the Terminal Adapter appear on the terminal status line of the screen. If the terminal has no status line, the bottom line of the screen is used instead.

2.4.12.4 Switching Between Presentation Modes

The following table shows the effects of switching between modes.

Switching from:	To:	Effect
Format Mode	Line Mode	Screen is cleared only if the user has a Terminal Adapter presentation.
Line Mode	Format Mode	Screen is cleared by the `CDGET` statement.
Terminal Adapter	Free Presentation	Screen is not cleared.
Free Presentation	Terminal Adapter	Screen is cleared only if the user is in line mode.

Note that in the case of `IMPLICIT RELEASE` (without using the `CALL "CDRELS"` statement), the screen is not cleared.

2.4.12.5 Displaying Messages on an User's Terminal

If the terminal operates in free presentation:

- Messages appear on the screen after the cursor position according to the advancing options, or control characters specified.
- If the message does not fit on the screen, the terminal may be disconnected.
- If the message is greater than the screen length, the beginning of the message is displayed; then the screen is cleared and the whole message is displayed. However, the screen display depends on the type of Datanet being used.

If the terminal uses the Terminal Adapter, you must consider whether the terminal operates in roll mode or not.

Terminals in Roll Mode

A message appears on the screen after the cursor position according to the advancing options specified.

Each new line at the bottom of the screen pushes up the remainder of the screen by one line, thus erasing the topmost one.

In the case of printers, messages are continuously displayed according to the advancing options specified. Printers always operate in roll mode.

Terminals Not in Roll Mode

Messages are displayed in one of two ways depending on whether they are external or not, as shown in the following illustration:

	FORMAT MODE	LINE MODE
SEND from User's transaction	Message is sent in Free Presentation (Normally a message is displayed in format mode)	Display A
SEND from another transaction	Display B	Display B
External Message	Display B	Display B
A rollback from Terminal Adapter Presentation	Form is re-displayed	Screen is cleared from the cursor position at the commitment point
A rollback from Free Presentation	Form is re-displayed	Screen is cleared

Figure 2-8. Displaying Messages with Terminal Adapter for a User Not in Roll Mode

Display A

- A message appears on the screen according to the advancing options specified. When a message uses the `AFTER ADVANCING` clause, the message is displayed independently of the cursor position.
- If the message does not fit on the screen, the screen is cleared and the message is then displayed.
- If the message is greater than the length of the page (a profile variable), the status key PAGEOV is sent to the TPR and the message is canceled.

Display B

Messages appear on the terminal status line that must not be used by applications. These messages are:

- either overlaid by a new message,
- or cleared by TDS when a RECEIVE statement is performed.

Messages longer than 62 characters may be truncated, and the status key MSG-TRUNC is sent to the TPR.

Messages are displayed in one of these ways only if the Terminal Adapter is supported by the addressed terminal; otherwise, the message is canceled and the status key TAFAIL is sent.

The Terminal Adapter and the BREAK Transaction

When a BREAK message is received, the screen is cleared from the cursor position, which was current at the most recent commitment point, and the next message appears from this cursor position.

When the BREAK processing has completed, the screen is cleared from the most recent commitment point preceding the BREAK and the next message is sent from this cursor position.

2.4.13 Developing Administrative Transactions

The TDS Administrator/Master terminal operator is offered two different ways of managing correspondents and pools:

- master commands,
- administrative transactions.

Administrative transactions are first developed by the TDS programmer and then used by the TDS Administrator/Master terminal operator as alternatives to the corresponding master commands (listed in parentheses) for doing the following tasks:

Master-terminal Operator Tasks	Call Statement Used by Programmer (see Chapter 3)
Allow/prevent new connections. ([M] ALLOW_NEW_TDS_COR ...)	CALL "MD-NEWCONNECT"
Display telecommunication limits of a TDS application ([M] DISPLAY_TDS ...)	CALL "DISP-SESLIMIT"
Open a session pool ([M] OPEN_COR_POOL ...)	CALL "OPEN-POOL"
Close a session pool ([M] CLOSE_COR_POOL ...)	CALL "CLOSE-POOL"
Modify characteristics of a session pool. ([M] MODIFY_COR_POOL ...)	CALL "MODIFY-POOL"
Display characteristics of a session pool. ([M] LIST_COR_POOL ...)	CALL "DISP-POOL"
List pools ([M] LIST_COR_POOL ...)	CALL "LIST-POOL"
Display characteristics of a correspondent ([M] LIST_TDS_COR ...)	CALL "DISP-COR"
List correspondents ([M] LIST_TDS_COR ...)	CALL "LIST-COR"

You can develop these administrative transactions to:

- manage the pool of dummy correspondents,
- modify the pool of sessions when either the XCP1 or XCP2 protocol is being used,
- display the dynamic, or static state of correspondents and pool objects,
- manage the telecommunication characteristics of a TDS application such as the number of sessions,
  Information concerning the correspondents can be defined statically at NETGEN, or dynamically during a TDS session as follows.
  - At NETGEN, any modification is definitive and overrides the previous static value.
  - When you make a modification through a control-operator call statement, the modification is temporary; that is, the modification is not preserved after the object is deactivated normally.

These control-operator call statements are executed immediately, that is, they do not require a commitment to be taken and they are not canceled if a rollback occurs later in the commitment unit. However, when you modify the NETGEN while a control-operator procedure is executing, the control-operator procedure aborts and an abnormal status is returned to the caller.

2.5 Status Setting

The status settings described below are either the STATUS-KEY in CD structures or the STATUS output parameter in verbs that do not make use of CDs. Some of these status codes appear only for XCP1 users.

Table 2-5. CD Status Keys (1/5)

Status Key	Return Code	Meaning
00	DONE	successful completion of the verb.
04	RCV-LV1	It is not possible to receive data without the turn. SEND is not allowed until you have the turn. Issue another RECEIVE in the next TPR.
1.	UNSUCCESSFUL	CP-ALLOCATE unsuccessful
10	UNS-SESPOOL	CP-ALLOCATE unsuccessful; no more sessions in pool
11	UNS-SESTX	CP-ALLOCATE unsuccessful; no more sessions for transaction
20	DEST-UNKNOWN	destination of SEND unknown
30	DEST-COUNT-INVALID	contents of DESTINATION COUNT invalid
40	WAIT	data not immediately available for RECEIVE or CDRVX. The user must terminate the current TPR and issue a new RECEIVE or CDRVX in next TPR
50	LENGTH-ERROR	character count greater than length of field to be sent
60	PARAM-ERROR	partial segment with 0 character count or no sending area specified on SEND
7.	TXINIT-ERROR	error notification received from a remote correspondent after transaction message has been sent
70	TXI-UNDEFINED	reason not identified
71	TXI-UNKNOWN	unknown transaction
72	TXI-SECLV-NOSUP	security level not supported by remote correspondent
73	TXI-CONVTY-NOSUP	conversation type not supported by remote application
74	TXI-NAV-NRTRY	transaction not available, e.g. authority code on mailbox XCP1
75	TXI-TERMREQ	dialog termination has been requested

Table 2-5. CD Status Keys (2/5)

Status Key	Return Code	Meaning
80	ABEND	dialog abnormally terminated by the remote application
94	TOO-LONG	message not completely transferred as a maximum message size (defined at generation time) has been exceeded
9A	BREAK	break has been received; message not completely transferred
A0	MSG-REJECT	last message sent rejected by remote application
A1	MSG-REFUSED	External message has been sent to an user with Terminal Adapter presentation and having the user-profile variable MAIL set to off (0)
A2	MSG-TRUNC	External message having more than 62 characters has been sent to an user with Terminal Adapter presentation.
A3	MSG-PAGEOV	A message longer than a logical page was sent to an user with Terminal Adapter presentation. The message is lost.
B0	REM-PROG-ERROR	remote transaction has aborted
C0	RESFAIL-NRTRY	dialog with remote application has terminated abnormally due to session failure which is not temporary (such as protocol violation)
D0	RESFAIL-RTRY	dialog with remote application has terminated abnormally due to session failure which is temporary (such as disconnect when security-level=0)

Table 2-5. CD Status Keys (3/5)

Status Key	Return Code	Meaning
E.	VERB-NOT-SUPPORTED	
E0	VNS-SNRVAUX	SEND or RECEIVE on auxiliary session
E1	VNS-RVXFORMATTED	CDSNX/CDRVX used with formatted mode.
E2	VNS-SNNOTTERM	SEND explicit destination to a correspondent which is neither the initiator nor a terminal (not applicable to XCP1 verbs)
E3	VNS-SNXTERM	CDNSNX to a correspondent not defined in the principal or an auxiliary session of the transaction
E4	VNS-RVEMI	RECEIVE with EMI
E5	VNS-SNXESI	CDSNX with ESI
E6	VNS-TAFAIL	The message cannot be adapted to the terminal and is lost.
F.	STATE-CHECK	verb not allowed in this state
F0	STCK-VERB	verb not allowed in this state
F1	STCK-SNSNXALREADY	SEND/CDSNX and a recoverable message already sent or SEND/CDSNX and last message already sent
F2	STCK-SECLEVEL	SYNC-UNIT does not match security level
F3	STCK-LAST	SYNC-UNIT does not match LAST MESSAGE.

NOTE:

E., F. are not status code. They are included simply to show the class of status code (1st character).

Table 2-5. CD Status Keys (4/5)

Status Key	Return Code	Meaning
G.	PARAM-ERROR	parameter error
G0	PARER-UNKSYBQ	unknown symbolic queue
G1	PARER-SNDESTEGI	Send explicit destination with EGI
G2	PARER-NOTALLOC	session not allocated when calling CDSNX or CDRVX
G3	PARER-SNESINEW	Send with ESI to new destination
G4	PARER-RETCTL	return control error on CP-ALLOCATE
G5	PARER-SECLEVEL	security-level error
G6	PARER-TXNAMELG	error in transaction name length
H.	INVALID-PARAM	invalid parameter
H0	INVPAR-CDDCE	invalid CD or DCE
H1	INVPAR-OTHER	invalid parameter on SEND/CDSNX or RECEIVE/CDRVX
H2	INVPAR-RETCTL	invalid return control on CP-ALLOCATE
H3	INVPAR-UNKCOR	correspondent specified is unknown
H4	INVPAR-IDENT	session identification invalid
H5	INVPAR-TXNAMELG	invalid transaction-name length
H6	INVPAR-SECLEVEL	invalid security level
H7	INVPAR-CORNAME	correspondent specified in DFRECOV is invalid

NOTE:
G., H. are not status codes. They are included simply to show the class of the status code (1st character).

Table 2-5. CD Status Keys (5/5)

Status Key	Return Code	Meaning
I0	TIMEOUT	Timeout
J0	DISCONNECT	Dialog with remote application has been interrupted due to a session failure whichj is temporary ; dialog can be restarted with message recovery
K.	FUNCTION-NAV	function is not available
K0	FNAV-SLAVEUNAC	slave inaccessible
K1	FNAV-SNDESTUNAC	SEND explicit destination inaccessible
K2	FNAV-BUFFERFULL	message buffer full
L0	BREAKX	break has been received when calling CDSNX/CDRVX

NOTE:
K. is not a status code. It is included simply to show the class of status code (1st character).

Table 2-6. Communications Verbs and XCP1 Procedures (CD Status Keys) (1/2)

	Verbs/Procedures									
Status Key Return Code	SEND	REC EIVE	CD SNX	CD RVX	CP-ALLO CATE	CP-FREE	INIT WORK	EXTR WORK	ABT-WORK	DF RECOV
00 DONE	X	X	X	X	X	X	X	X	X	X
10 UNS-SESPOOL					X					
11 UNS-SESTX					X					
12 UNS-TFAIL	X									
20 DEST-UNKNOWN	X									
30 DEST-COUNT-INVALID	X									
40 WAIT				X						
50 LENGTH-ERROR	X									
60 PARAM-ERROR	X									
70 TXI-UNDEFINED	X	X	X	X		X	X			
71 TXI-UNKNOWN	X	X	X	X		X	X			
72 TXI-SECLV-NOSUP	X	X	X	X		X	X			
73 TXI-CONVTYP-NOSUP	X	X	X	X		X	X			
74 TXI-NAV-NRTRY	X	X	X	X		X	X			
75 TXI-TERMREQ	X	X	X	X		X	X			
80 ABEND	X	X	X	X		X		X		
94 TOO-LONG	X									
9A BREAK	X									
A0 MSG-REJECT	X	X	X	X		X	X			
B0 REM-PROG-ERROR	X	X	X	X		X	X			
C0 RESFAIL-NRTRY	X	X	X	X		X	X	X		
D0 RESFAIL-RTRY	X	X	X	X		X	X	X		
E0 VNS-SNRVAUX	X	X								
E1 VNS-RVXFORMATTED			X	X						
E2 VNS-SNNOTTERM	X									
E3 VNS-SNXTERM			X							
E4 VNS-RVEMI		X								
E5 VNS-SNXESI										
F0 STCK-VERB	X	X	X	X	X	X	X	X	X	X
F1 STCK-SNSNXALREADY	X		X							
F2 STCK-SECLEVEL			X							
F3 STCK-LAST			X							

Table 2-6. Communications Verbs and XCP1 Procedures (CD Status Keys) (2/2)

	Verbs/Procedures									
Status Key Return Code	SEND	REC EIVE	CD SNX	CD RVX	CP- ALLO CATE	CP- FREE	INIT WORK	EXTR WORK	ABT- WORK	DF RECOV
G0 PARER-UNKSYMBQ		X		X						
G1 PARER-SNDESTEGI	X									
G2 PARER-NOTALLOC			X	X						
G3 PARER-SNESINEW	X									
G4 PARER-RETCTL					X					
G5 PARER-SECLEVEL							X			
G6 PARER-TXNAMELG							X	X		
H0 INVPAR-CDDCE	X		X							
H1 INVPAR-OTHER	X	X	X	X						
H2 INVPAR-RETCTL					X					
H3 INVPAR-UNKCOR					X	X	X		X	
H4 INVPAR-IDENT						X	X		X	X
H5 INVPAR-TXNAMELG							X	X		
H6 INVPAR-SECLEVEL							X			
H7 INVPAR-CORNAME										X
I0 TIMEOUT		X		X						
J0 DISCONNECT	X	X	X	X		X		X		
K0 FNAV-SLAVEUNAC			X							
K1 FNAV-SNDESTUNAC			X							
K2 FNAV-BUFFERFULL			X							
L0 BREAKX			X							

Table 2-7 gives the values and meanings specific to the use of FORMS, together with the relevant return codes.

Table 2-7. Status Key Values for FORMS (1/2)

Value	Return Code	Meaning
NORMAL		
0	DONE	Normal Execution.
A8	SKIPPED	Some received fields were not selected; they are lost.
AB	ALMOST	An error was found in at least one field. Check selection vector contents to find erroneous fields.
AF	DONEIDE	Invalid selection vector contents.
9I	DONEIDC	A function key has been received but the function code field was not selected.
ABNORMAL		
92	ENTRYOV	Maximum number of active or frozen forms exceeded.
92	SPACEOV	No more space available for control structures.
92	MSGOV	Maximum message area declared in TDS generation is too small.
97	OPTERR	Invalid enclosure level.
9A	BREAK	A Break occurred.
9F	CALLVIOL	Environment is neither TDS nor IOF.
9H	DNSPEC	Invalid endpoint name. The contents of the SYMBOLIC DESTINATION or SYMBOLIC QUEUE is illegal.
9J	ALREADY	Form is already active (last form is activated in Window mode).
A0	SNDVIOL	Turn error (SEND)
A1	SEQERR	Endpoint not in forms mode.
A3	RCVVIOL	Turn error (RECEIVE). This status is returned on CDRECV if there are no messages to be received.

Table 2-10. Status Key Values for FORMS (2/2)

Value	Return Code	Meaning
ABNORMAL		
A4	RECNFD	The specified form was not found.
A6	OBJUNKN	Unknown attribute or mechanism.
A7	RECARERR	The selection vector does not match an active form. (RECEIVE)
A7	SNDARERR	The selection vector does not match an active form. (SEND)
A9	DVIDFBID	Device not supported.
A9	CONFLICT	The attribute conflicts with the form (e.g. because the selected field starts in column 1).
AC	ARGERR	Unexpected parameter or incompatible version numbers between data structures and object form.
AE	FUNCNAV	Activation mode not available.
AG	NOMATCH	The creation date of the form does not match the creation date of program structures.
AI	IGNORE	Combination of error 9I and some other erroneous condition.
S1	DAMAGED	System error.

3. Session Management Procedures

3.1 Overview

This chapter deals with the following procedures:

DISP-SESLIMIT	displays the telecommunication limits of a TDS application:
GET-SYNCSTATE	obtains the synchronization of committed data in XA Resource Managers.
MD-NEWCONNECT	prevents, or allows new connections from being established, but does not affect the current connections.
RECONNECT-OPTION	specifies the transactions that will execute at the time of reconnection or disconnection.
SET-ACTIVE	sets a terminal from the passive to the active state, and allows the terminal operator to initiate a new transaction when the current transaction terminates.
SET-PASSIVE	sets a terminal from the active to the passive state,
TERMID	gets the characteristics of the principal correspondent (terminal).

and the following verbs:

RECEIVE	makes an incoming message available to the TPR.
SEND	causes a message to be released to the destination that has been specified in the SYMBOLIC DESTINATION field in the output CD.

3.2 The CALL "DISP-SESLIMIT" Procedure

Syntax

```
CALL "DISP-SESLIMIT" USING option,
                           limit-values,
                           status.
```

Description

Displays the telecommunication limits of a TDS application:

- either as defined in NETGEN (static state),
- or the current telecommunication limits (dynamic state).

The equivalent GCL master command is `[ M ] DISPLAY_TDS...` (described in the *TDS Administrator's Guide*).

Option is an input parameter consisting of a 1-character alphabetic field for selecting the static or dynamic characteristics as follows.

`S`	indicates Static parameters.
`D`	indicates Dynamic parameters.

Limit-values refers to an output data structure which you can copy from the <tdsname>. COBOL file by using the COBOL statement `COPY H-DC-TP-TDSLIMIT`.

```
05 coc-tds-limit.
   06 coc-tds-limit-version  PIC X VALUE 1. input=1
   06 coc-mxnb-of-tmses      COMP-1.        max number of TM
                                            sessions
   06 coc-mxnb-of-xcp1ses    COMP-1.        max number of XCP1
                                            sessions
   06 coc-mxnb-of-xcp2ses    COMP-1.        max number of XCP2
                                            sessions
   06 coc-cnb-of-tmses       COMP-1.        current number of TM
                                            sessions
   06 coc-cnb-of-xcp1ses     COMP-1.        current number of XCP1
                                            sessions
   06 coc-cnb-of-xcp2ses     COMP-1.        current number of XCP2
                                            sessions
```

For the D option, the maximum number fields returned contains 0 if new connections are prevented from being established through the `CALL "MD-NEWCONNECT"` procedure, or through the corresponding master command `[ M ] PREVENT_NEW_TDS_COR`.

The current number fields `(coc-cnb-of-tmses, coc-cnb-of-xcp1ses, coc-cnb-of-xcp2ses)` contain 0 when one of the following conditions is met:

- option is set to S,
- option is set to D and the field refers to a XCP2 number even though the XCP2 service is not allowed for this TDS application.

Status refers to the following data structure that you can copy from the <tdsname>. COBOL file by using the COBOL statement `COPY H-DC-TP-STAT`.

```
05 coc-status.
06 coc-status-version    PIC X  VALUE 1.  input=1
06 coc-external-status   PIC X.
06 coc-system-status.
   07 coc-issuer         PIC X.
   07 coc-code           COMP-1.          status code
   07 coc-subcode        COMP-1.          status subcode
   07 coc-last-rc        COMP-2.
```

Only the `coc-external-status` field can be checked by the caller. The values it can take can be retrieved through the COBOL statement `COPY H-DC-TP-STAT`.
The `coc-external-status` field can contain the following values.

coc-argerr (2)	The field option is not reachable. The field coc-tds-limit is not reachable. coc-status-version does not equal 1. coc-tds-limit-version does not equal 1. option is neither S nor D.
coc-arviol (3)	The transaction issuing the DISP-SESLIMIT was not submitted by the master.
coc-busy (4)	Option equals S: NETGEN is being updated. Retry later.
coc-done (0)	Request is performed.
coc-notdone (9)	PPC incident (returned only if option equals D). NETGEN incident (returned only if option equals S). Contact the Service Center.

coc-objunkn (B)	TDS workstation was not declared in NETGEN (returned only if option equals S). XCP2 workstation was not declared in NETGEN (returned only if option equals S).
coc-ogenunkn (C)	NETGEN was incrementally generated while TDS was processing the command. Retry later (returned only if option equals S).
coc-resnav (D)	The XCP2 service was not started (returned only if option equals D).
coc-wrongpar (H)	Error occurred on the TDS/PPC interface. Contact the Service Center (returned only if option equals D).

3.3 The CALL "MD-NEWCONNECT" Procedure

Syntax

```
CALL "MD-NEWCONNECT" USING param-structure, status.
```

Description

Prevents new connections from being established or allows new connections to be established. It does not affect the current connections.

The equivalent GCL master command is `[ M ] ALLOW_NEW_TDS_COR...` `and/or  [ M ] PREVENT_NEW_TDS_COR` (described in the *TDS Administrator's Guide*).

Input parameters

`param-structure` refers to a structure which you can copy from the <tdsname>. COBOL file by using the COBOL statement `COPY H-DC-TP-NWCNCT`.

```
05 coc-new-connect.
   06 coc-new-connect-version PIC X VALUE 1. input=1
   06 coc-allow-flags.
      07 coc-xcp2-fg          PIC X.  Y -> allows and N -> rejects
                                      U -> unchanged
      07 coc-tm-fg            PIC X.  Y -> allows and N -> rejects
                                      U -> unchanged
      07 coc-xcp1-fg          PIC X.  Y -> allows
                                      N -> rejects
                                      U -> unchanged
```

where:

coc-xcp2-fg	refers to XCP2 connections.
coc-tm-fg	refers to TM connections.
coc-xcp1-fg	refers to XCP1 connections.

To allow new connections to be established, specify Y

To prevent new connections from being established, specify N

To leave connections unchanged, specify U.

`coc-xcp2-fg` set to Y has no effect on future requests for allocating a conversation, but any future request:

- to open a pool for the TDS application will be rejected,
- to increase the number of sessions will be rejected.

For TS 7560, TS 8650, and TS 9662, TDS-TCP/IP correspondents may be selected, but the H-DC-TP-NWCNCT structure (copied from the <tdsname>.COBOL library) does not contain this correspondent type.

If a TDS-TCP/IP correspondent is to be selected, the programmer must add the following declaration after the COBOL statement COPY H-DC-TP-NWCNCT:

```
      07    coc-tcpip-fg    PIC X.
```

and move “2” to the `coc-new-connect-version` field.

The `coc-tcpip-fg` field must be loaded with Y (allows), N (rejects), or U (unchanged).

The programmer may also use a private param-structure including the `coc-tcpip-fg` field; the `coc-new-connect-version` field loaded with “2” enables TDS-TCP/IP correspondent selection.

After TS 9662, the H-DC-TP-NWCNCT structure (copied from the <tdsname>.COBOL library) contains:

- the `coc-new-connect-version` field initialized with the value “2”, and
- the fields: `coc-xcp2-fg`, `coc-tm-fg`, `coc-xcp1-fg`, and `coc-tcpip-fg` initialized with the value “U” (unchanged).

Output parameters

allow-flags:

coc-xcp2-fg = "1"	the request for XCP2 sessions is performed.
coc-xcp2-fg = "0"	the request for XCP2 sessions has failed.
coc-tm-fg = "1"	the request for TM sessions is performed.
coc-tm-fg = "0"	the request for TM sessions has failed.
coc-xcp1-fg = "1"	the request for XCP1 sessions is performed.
coc-xcp1-fg = "0"	the request for XCP1 sessions has failed.

`coc-tm-fg, coc-xcp1-fg` and `coc-xcp2-fg` are meaningful only if the field `"coc-external-status"` does not contain COC-ARGERR, or COC-ARVIOL.

Status refers to the following data structure that you can copy from the <tdsname>. COBOL file by using the COBOL statement `COPY H-DC-TP-STAT`.

```
05 coc-status.
06 coc-status-version    PIC X  VALUE 1.  input=1
06 coc-external-status   PIC X.
06 coc-system-status.
   07 coc-issuer         PIC X.
   07 coc-code           COMP-1.          status code
   07 coc-subcode        COMP-1.         status subcode
   07 coc-last-rc        COMP-2.
```

`The coc-external-status` field can contain the following values:

coc-argerr (2)	The coc-new-connect field is not found. coc-status-version does not equal 1. coc-new-connect-version does not equal 1. coc-xcp2-fg is neither Y nor N. coc-tm-fg is neither Y nor N. coc-xcp1-fg is neither Y nor N.
coc-arviol (3)	The transaction issuing the MD-NEWCONNECT was not submitted by the master.
coc-done (0)	Request is performed.
coc-notdone (9)	A TDS or VCAM incident occurred. Contact the Service Center.

NOTE:
The request is performed in the following order:

```
XCP2
TM
XCP1
```

When a request fails, the first flag (starting from coc-xcp2-fg) set to 0 indicates which request has failed.

3.4 The CALL "RECONNECT-OPTION" Procedure

NOTE:
This call is supported only with the TDS-HA (High Availability) product.

Syntax

```
CALL "RECONNECT-OPTION" USING data-name-1, data-name-2.
```

Description

Specifies the special transactions to execute at time of the disconnection, reconnection, or both. The data-name-1 structure selects these options; so it contains the input parameters. Data-name-2 contains the results of the call. This call does the following:

- reduces the reconnection time by preventing the execution of one or more of the special transactions.
- "hides" the disconnection-reconnection from the end-user by preventing the execution of all special transactions.
- specifies whether or not to restart the end-user's transaction that was in progress at the time of the disconnection.
- specifies a maximum time between disconnection and reconnection. Users not reconnected within this specified time limit are not automatically reconnected.

Usage

- The data-name-1 structure contains input parameters to be passed during the call, as follows:

```
02  MAX-RECONNECT-TIME          COMP-1.
02  DISCONNECT                  PIC X(1).
02  LOGON                       PIC X(1).
02  RESTART                     PIC X(1).
02  RFU                         PIC X(5).
```

- The data-name-2 structure is the output, as follows:

```
01  DATA-NAME-2                 PIC 9.
```

For more information about the TDS-HA product, see the *High Availability Concepts* manual and *High Availability Administrator's Guide*.

Input Parameters

MAX-RECONNECT-TIME	is the maximum time-limit delay, in seconds, taken into account at take-over. If this parameter is not specified, the default is 10 minutes. The delay is the time elapsed, in seconds, from the time of the user's last activity (that is, the last commitment of the last transaction) and the time of reconnection. A user not reconnected during this time limit is then not reconnected automatically. A value of zero means that no automatic reconnection takes place. A negative value gives an error message (see data-name-2, below).
DISCONNECT	specifies whether to execute the special transaction DISCNCT at accidental disconnection or automatic reconnection (after a take-over). A value of 0 means that the DISCNCT transaction does not execute if an accidental disconnection affects any users (TDS abort or member failure). A value of 0 overrides the effect of the TDS master command MODIFY_TDS_RESTART_OPTION with EXEC_DISCONNECT_TX = 1 executed on the TDS (before the take-over). A value of 7 causes the transaction to execute when a user is accidentally disconnected. DISCNCT can also be executes at TDS shutdown or member failure (if a TDS master command MODIFY_TDS_RESTART_OPTION with EXEC_DISCONNECT_TX = 1 is executed on the TDS.
LOGON	specifies if the special transaction LOGON is to be executed at every further reconnection (whether automatic reconnection after a take-over or after any accidental reconnection). A value of 0 means that the LOGON transaction does not execute. A value of 7 means that the LOGON transaction does execute. **NOTE:** By setting the LOGON value to 0, you disable the reconnect-option call. The call must be done in the LOGON transaction.

RESTART	specifies if the special RESTART transaction executes at every further reconnection (whether automatic reconnection after a take-over or any accidental disconnection). A value of 0 means that the RESTART transaction does not execute. A value of 7 means that the RESTART transaction does execute. There must be a context and a user transaction in progress at the take-over.
RFU	five blank characters reserved for future use.

Output Parameters

As a result of the call, TDS places an output value in Data-Name-2. These values and their meanings are as follows:

0	Successful execution
1	Negative value given in the MAX-RECONNECT-TIME parameter.
2	Invalid value specified in DISCONNECT, LOGON, or RESTART.
3	Wrong session type. The principal session of the transaction making the call is not TM.
4	Wrong transaction is making the call. The call can be made only from within a LOGON transaction.
5	Call not supported. The call can be made only from an HA TDS.

NOTES:

1. This procedure can be called in only the LOGON transaction.
2. TDS takes into account the options specified in this call after the commitment-point, following the call.
3. The options remain valid until either the correspondent is no longer known to TDS (after a normal disconnection or M CANCEL_TDS_COR), or a new call "RECONNECT-OPTION" is performed (in the LOGON transaction), which overrides the preceding "RECONNECT-OPTION" call.
4. This call statement also affects the automatic reconnection processing performed on the M ALLOW_NEW_TDS_COR master command.
5. This statement is available only with an HA TDS (which is declared WATCHED BY CMSC). For a take-over of an HA TDS, this call is taken fully into account, and the commands MDTRSO and TTDS have no effect. For a TDS warm restart, the commands M MDTRSO and M TTDS are taken into account first, and the maximum reconnection time is taken into account only for those sessions that must be reconnected. This is as specified in the master command using this statement. Some special transactions can be passed over, no matter what the master command specifies.
6. For an active correspondent, avoiding these special transactions can reduce the reconnection time. When the transaction resumes, the last message is sent again.
7. For a passive correspondent, such as a printer, it may be necessary to indicate the disconnection (with these special transactions) in order to manage the duplicate messages.

3.5 The CALL "SET-ACTIVE" Procedure

Syntax

```
CALL "SET-ACTIVE".
```

Description

Applicable for terminals with a keyboard, with the following results:

- sets the terminal from the passive to the active state,
- Allows the terminal operator to initiate a new transaction when the current transaction terminates.

NOTES:

1. If the user at TDSGEN has supplied no break routine, TDS will switch the terminal to the active state upon receipt of a break.
2. SET-ACTIVE is ignored if the terminal has no input capabilities (receive-only terminals) or if it is already active.

For more information, see Figure 3-1, which shows how to set a terminal between active and passive.

3.6 The CALL "SET-PASSIVE" Procedure

Syntax

```
CALL "SET-PASSIVE".
```

Description

Applicable for terminals with a keyboard, with the following results:

- sets the terminal from the active to the passive state,
- the READY message no longer appears,
- Prevents the operator from initiating a new transaction when the current transaction has terminated.

Usage

The terminal may be switched to the active state by the "SET-ACTIVE" clause.

The terminal may also be switched to the active state by the BREAK key if the user supplies no BREAK transaction. In this case, the current transaction is aborted.

See the following illustration.

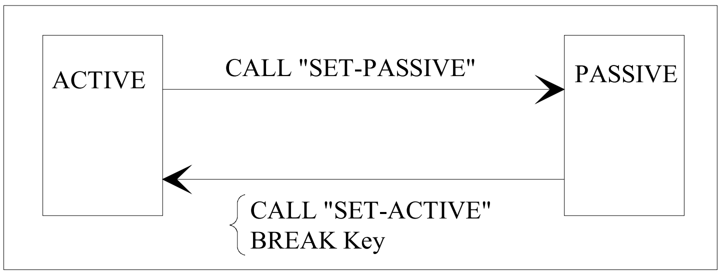


Figure 3-1. Switching a Terminal between Passive and Active States

3.7 The CALL "TERMID" Procedure

Syntax

```
CALL "TERMID" USING ADDRESS OF data-name-1.
```

Description

The TERMID statement gets the characteristics of the principal correspondent (terminal).

Usage

Data-name-1 is an output parameter. It is the name of a structure that must be declared as follows:

```
01 data-name-1.
   02 CORRESPID            PIC X(12).
   02 PROJECT              PIC X(12).
   02 BILLING              PIC X(12).
   02 LOCALDVC.
      03 MODEL             COMP-1.
      03 DVTYPE
         04 DISPLAY        PIC X.
         04 KEYBRD         PIC X.
         04 PRINTER        PIC X.
      03 PAGEL             COMP-1.
      03 LINEL             COMP-1.
      03 DVFEATR.
         04 ROLLUP         PIC X.
         04 WRAPAR         PIC X.
         04 LINFEED        PIC X.
         04 LINFOLD        PIC X.
         04 HTAB           PIC X.
         04 VTAB           PIC X.
   02 FILLER               PIC X(12).
```

All these characteristics are returned by the CALL "TERMID" statement. No status is returned. Before the fields of the data structure are filled, the length of the structure is checked. If the structure is too small, the transaction is aborted with the ARGERR return code.

Parameters

CORRESPID	is a 12-character alphanumeric field. It identifies the principal correspondent of the transaction.
PROJECT	is a 12-character alphanumeric field. It identifies the project of the correspondent.
BILLING	is a 12-character alphanumeric field. It identifies the billing of the correspondent.
MODEL	is COMP-1 field. It gives the device-model identification of the primary correspondent (terminal).
DISPLAY	is a 1-character alphanumeric field. 0 = not a screen. 1 = a screen.
KEYBRD	is a 1-character alphanumeric field. 0 = not a keyboard. 1 = a keyboard.
PRINTER	is a 1-character alphanumeric field. 0 = not a printer. 1 = a printer.
PAGEL	is a COMP-1 field that gives the length of the page.
LINEL	is a COMP-1 field, which gives the length of the line.
ROLLUP	is a 1-character alphanumeric field. 0 = no automatic roll-up. 1 = automatic roll-up.
WRAPAR	is a 1-character alphanumeric field. 0 = no automatic wrap around. 1 = automatic wrap around.
LINFEED	is a 1-character alphanumeric field. 0 = no automatic line feed. 1 = automatic line feed.
LINFOLD	is a 1-character alphanumeric field. 0 = no automatic line folding 1 = automatic line folding.

HTAB	is a 1-character alphanumeric field. 0 = no horizontal tabulation. 1 = horizontal tabulation.
VTAB	is a 1-character alphanumeric field. 0 = no vertical tabulation. 1 = vertical tabulation.

3.8 The RECEIVE Verb

Syntax

```
RECEIVE cd-name MESSAGE INTO identifier-1
                             [ NO DATA imperative-statement ].
```

Description

Makes an incoming message available to the TPR. When no data is available, the RECEIVE statement allows a specified imperative statement. When the terminal adapter is being used and the incoming message has several lines, only the first line is received by the TPR.

Usage

cd-name must reference an input communication description (CD) in the COMMUNICATION SECTION of the Data Division.

identifier-1 is the name of the area in memory where the message is to be placed.

NOTE:

TDS updates the data items defined in the input CD with each execution of a RECEIVE statement.

TDS makes data available to the TPR in identifier-1 when a RECEIVE statement is issued, either at the beginning of the first TPR of a transaction, or at the beginning of a subsequent TPR that has been activated (the previous TPR having issued a SEND WITH EGI statement).

Additional RECEIVE statements may be required to transfer the remainder of the message. These additional RECEIVE statements may be split over several TPRs of the transaction.

If no data is available in identifier-1 during execution of a RECEIVE statement, one of the following occurs:

- If the NO DATA phrase is specified, the RECEIVE operation is terminated with a STATUS KEY indicating that the action is complete. The imperative statement is executed.
- If the NO DATA phrase is omitted, the status key of the input CD is set to F0 (see Table 2-5).

The following apply to data transfer:

- If the size of the message is the same as the size of the area referenced by identifier-1, the message is stored in identifier-1.
- If the size of the message is smaller than the area referenced by identifier-1, the message is aligned to the leftmost position of identifier-1, without space fill.
- If the size of the message is greater than the area referenced by identifier-1, the message fills identifier-1 left to right. The remainder of the message is transferred to identifier-1 by subsequent RECEIVE statements issued by the same TPR, or another TPR prior to any other communication request. As many RECEIVE statements can be issued as are required to transfer to identifier-1 a complete message. If the transaction is terminated before the complete message is transferred, the part not transferred is lost. The whole message has to be transferred before any SEND statement can be performed. Infringement of this rule causes the TPR to obtain the status key STCK-VERB in CD (see Table 2-5).

Data items referenced by data-names in the input CD are updated with each execution of a RECEIVE statement.

3.9 The SEND Verb

Syntax 1

```
SEND cd-name FROM identifier-1.
```

Syntax 2

```
                                          { ESI }
SEND cd-name [ FROM    identifier-1 ] WITH { EMI }
                                          { EGI }

{ identifier-2 }

  [ { BEFORE }           { identifier-3                 } ]
  [ { AFTER  } ADVANCING { integer      [ { LINES } ] } ].
  [                      { PAGE         [ { LINE  } ] } ]
```

Description

The SEND statement causes a message to be released to the destination that has been specified in the SYMBOLIC DESTINATION field in the output CD. If the destination is the terminal or correspondent that activated the current transaction, the SEND operation can also indicate that a response is required from the terminal operator or correspondent.

Usage

cd-name: must reference an output communication description entry (CD) in the COMMUNICATION SECTION of the DATA DIVISION.

identifier-1: names the area in memory from which the message is to be sent.

identifier-2: must reference a one-character integer without an operational sign.

"1"	for ESI (End of Segment Indicator)
"2"	for EMI (End of Message Indicator)
"3"	for EGI (End of Group Indicator).

identifier-3: when used in the ADVANCING phrase, it must reference a decimal value. Identifier-3 can be zero.

NOTES:

1. TEXT-LENGTH of the output CD specifies the number of bytes to be sent from memory beginning from the leftmost position of identifier-1. If the Terminal Adapter is used, ensure that TEXT-LENGTH is less than 2048 bytes.
2. If TEXT-LENGTH is negative or greater than memory, the STATUS-KEY of the output CD is set to 50 (LENGTH-ERROR).

Syntax 1 is equivalent to the SEND with ESI statement, that is, a Syntax 1 SEND releases only a portion of a message or message segment to TDS. TDS transmits only the complete message. Therefore, these portions are held by TDS in the output buffer and they are then said to be "quarantined". All successive portions are concatenated. The message is completed with a SEND EMI or EGI (Syntax 2) statement. At the end of a TPR, any portion or segment of a message not terminated by an end of message indicator (EMI or EGI) of a Syntax 2 SEND statement is discarded from the system. No portion or segment of the message is sent.

A single execution of a SEND statement in Syntax 2 never releases to TDS more than a single message or message segment based on the specified indicator.

ESI	means End of Segment Indicator. ESI is used with the first or intermediate message segments. Since the message is not yet terminated, the next communication request must be another SEND statement. Segments are held in quarantine until a SEND with EMI or EGI statement is made.
EMI	means End of Message Indicator. EMI is used with the last (or only) message segment of a complete message. The message is no longer held in quarantine and may be released to the correspondent. The turn is kept by the TPR. A new message may now be built for the same or another destination.
EGI	means End of Group Indicator. EGI is used to indicate the end of a complete message when a response is required. After the output operation is completed, the turn is given to the terminal or correspondent, that is, the correspondent is switched to input mode.

Implicitly, the next communication verb must be a RECEIVE statement. The EGI indicator should accompany the last SEND statement of a transaction. If this is not done, TDS assumes it.

If the indicator conflicts with the previous communication verb, the STATUS KEY of the output CD is set to F0 (STCK-VERB) (see Table 2-5).

As long as a MESSAGE is not completed by an EMI or EGI indicator, all requests must reference the same destination. Infringement of this rule causes the TPR to receive the status PARAM-ERROR.

When the transaction was spawned for a passive terminal, that terminal may be referenced by a SEND with EGI statement.

ADVANCING (Syntax 2) allows control of the vertical positioning of each message or message segment on a terminal where vertical positioning is applicable. If vertical positioning is not applicable to the terminal, TDS ignores the vertical positioning specified or implied.

On a terminal where vertical positioning is applicable and the ADVANCING phrase is not specified, automatic advancing occurs as if the AFTER ADVANCING 1 LINE phrase were specified.

Note that if the ADVANCING phrase is implicitly or explicitly specified and vertical positioning is applicable:

- If identifier-3 or integer is specified, the transmission to the terminal is positioned downward the number of lines indicated by the value of the data item referenced by identifier-3 or integer.
- If the BEFORE phrase is used, the message or message segment is represented on the terminal before vertical positioning takes place.
- If the AFTER phrase is used, the message or message segment is represented on the terminal after vertical positioning takes place.
- If PAGE is specified, transmissions are represented on the terminal before or after (depending on the option specified) the terminal is positioned to the next page.
- If PAGE is specified but does not apply to the particular terminal, advancing occurs as if BEFORE or AFTER (depending on which was specified) ADVANCING 1 LINE were specified.

The use of the ADVANCING phrase adds to the size of the message or message segment three characters for a page advancement and two additional characters for each line positioned. This should be taken into account when defining the size of the largest message in TDS (MESSAGE-LENGTH clause in the TDS SECTION at TDSGEN).

Special characters from CONSTANT STORAGE can be inserted as part of the contents of the data item referenced by identifier-1. These characters are transmitted to the terminal without modification.

Be careful

If the SYMBOLIC DESTINATION of the CD-output is loaded from TDS storage (for example, PRIVATE or TRANSACTION STORAGE), it is necessary to refresh this storage field in case of user reconnection. In fact, the user may use another terminal at reconnection time. If this storage field is not refreshed, unpredictable results may occur (for example, next TPR launched without message transmission, looping, etc.).

This field must be refreshed from the SYMBOLIC SOURCE field of CD-input once the RECEIVE verb has been executed or from the TERMINAL-ID field of the LOGON TRANSACTION STORAGE.

For more details, refer to CD-output (in particular the explanation of data-name-5).

4. TPR Control Procedures

4.1 Overview

This chapter deals with the following procedures:

ABORT	aborts the transaction.
CANCELCTX	the TPR calling CANCELCTX continues, but the previous context is deleted.
DISPLAY-MENU	displays the TDS menu with the list of available transactions on the terminal from which the request was actually issued.
EXITS	forces the current TPR to terminate.
GETTPRPAR	returns the length and addresses of structures, and the number of stacked contexts.
RESTORE	the TRANSACTION-STORAGE is restored to the state it had in a previous context which has been saved at the interrupt time.
SIMBRK	simulates a break for a correspondent in the same way as if it was issued at the terminal.
SUBJOB	submits a request for asynchronous job execution.
XSIMBRK	simulates a break for a correspondent in the same way as if it was issued at the terminal.

and the following verbs:

ACCEPT	causes the information requested to be transferred to a data item specified by an identifier.
DISPLAY	outputs data to a terminal or SYSOUT.
EXIT	exits the program.
STOP	aborts the transaction.
WRITE in User Journal	outputs user-defined records to the user journal.

NOTE:

The procedures and verbs are described in the order shown above, not in alphabetic sequence.

4.2 The CALL "ABORT" Procedure

Syntax

```
CALL "ABORT" [ USING abort-code ].
```

Description

Aborts the transaction.

Usage

abort-code is a data-name defined as COMP-1. It is stored by TDS in the ABORT-CODE field of TDS-STORAGE. The ABORT-CODE field, which originally contains zero, is updated with the abort code given in CALL "ABORT". The default abort-code is USERREQ (user request).

TRANSACTION-STORAGE and PRIVATE-STORAGE are not rolled back (whether or not the ON-ABORT-TPR mechanism is used).

4.3 The CALL "CANCELCTX" Procedure

Syntax

```
CALL "CANCELCTX" USING RANK.
```

Description

The TPR calling CANCELCTX continues, but the previous context is deleted. RANK is a COMP-1 (FIXED BIN-15) output parameter that contains the number of the context that has been canceled. Zero means that no context has been canceled; one means that the context number 1 (the main context) has been canceled, and so on). Each CANCELCTX reduces the number in RANK by 1.

If there is no previous context, CANCELCTX has no effect.

By testing RANK, the user knows how many previous contexts remain and can decide what action next to take.

The CALL "CANCELCTX" procedure is useful only in the following transactions:

```
BREAK
DISCNCT
LOGON
RESTART
```

If the `CALL` "`CANCELCTX`" procedure is used in a user transaction, RANK will be 0.

If the `CALL` "`CANCELCTX`" procedure is used in system transactions, for example, STARTUP, SHUTDOWN or LOGOUT, then RANK will be (-1).

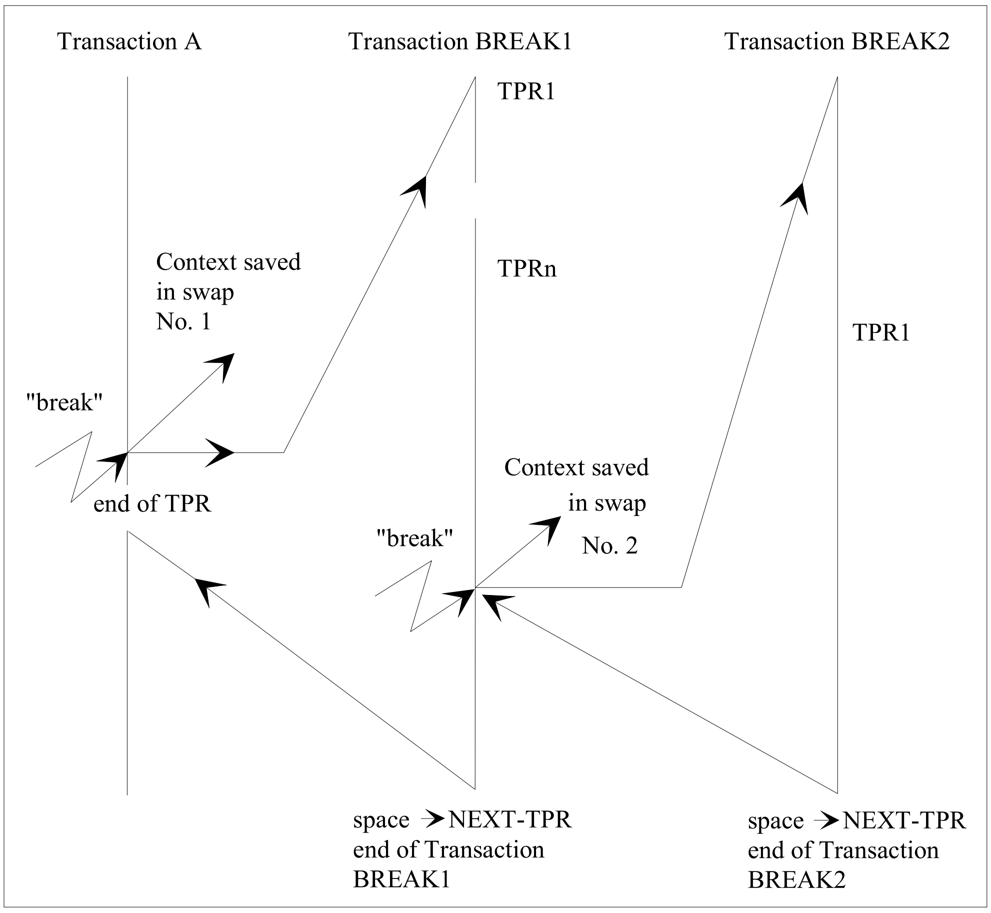
If the previous context refers to an XCP2 transaction, the RANK will be (-2). In this case, the function CANCELCTX is not available. The XCP2 transaction must be terminated in another way (for example, by moving "CANCELTX" to the next TPR).

On deletion of the main context (RANK=1) by the CALL "CANCELCTX" procedure

- The interrupting context replaces the main context, particularly for the READY message, and for FORMS with the NO IMPLICIT RELEASE option in use. The READY message is displayed at the terminal if both no form is active and no message has been sent in the last TPR of the interrupting transaction. Note that the FORMS procedure such as `CALL "CDMECH"` or `CALL "CDRELS"` send messages.
- No context remains when the DISCNCT transaction is executed. This means that the next connection will be as for the first log-on.

Note that in the case of a connection after an abnormal disconnection, the LOGON transaction can use CALL "CANCELCTX" to delete the previous context, that is, the context that existed before disconnection.

EXAMPLE:


Transaction A
Transaction BREAK1
Transaction BREAK2
TPR1
Context saved
in swap
No. 1
TPRn
TPR1
"break"
end of TPR
Context saved
in swap
No. 2
"break"
space → NEXT-TPR
end of Transaction
BREAK1
space → NEXT-TPR
end of Transaction
BREAK2

Transaction A has no previous context.

Transaction BREAK1 has 1 previous context.

Transaction BREAK2 has 2 previous contexts.

1. If the CALL "CANCELCTX" is inside transaction A:
   - no action takes place (CALL "CANCELCTX" is ignored),
   - RANK = 0
2. If the CALL "CANCELCTX" is inside transaction BREAK1, CALL "CANCELCTX" cancels the previous context:
   - transaction A is aborted (ON-ABORT-TPR, if any, is not activated),
   - RANK = 1
   - transaction BREAK1 continues until the end.
3. If the CALL "CANCELCTX" is inside transaction BREAK2, CALL "CANCELCTX" cancels the previous context:
   - transaction BREAK1 is aborted (ON-ABORT-TPR, if any, is not activated),
   - RANK = 2
   - transaction BREAK2 continues until the end.

❑

4.4 The CALL "DISPLAY-MENU" Procedure

Syntax

```
CALL "DISPLAY-MENU".
```

Description

Displays the TDS menu with the list of available transactions on the terminal from which the request was actually issued. A side effect of using this CALL statement is that if several transactions are being spawned to this terminal, note that this CALL statement is automatically spawned with the "high" priority.

4.5 The CALL "EXITS" Procedure

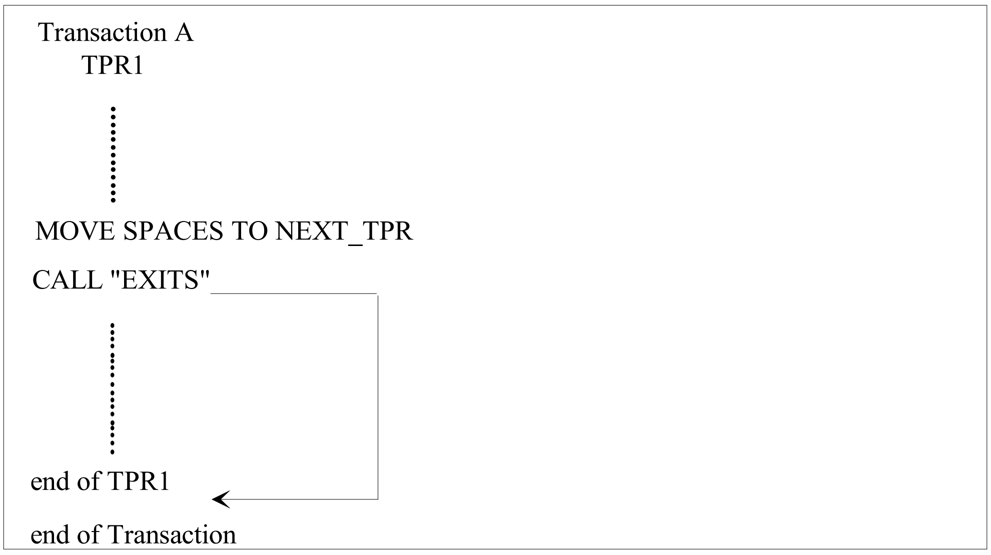
Syntax

```
CALL "EXITS".
```

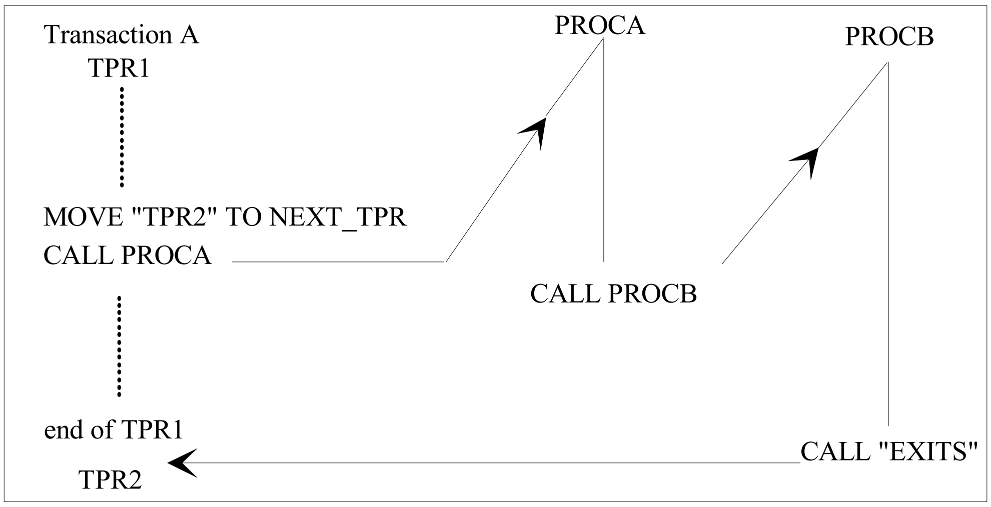
Usage

Forces the current TPR to terminate.

EXAMPLE 1



EXAMPLE 2


Transaction A
TPR1
MOVE "TPR2" TO NEXT_TPR
CALL PROCA
end of TPR1
TPR2
PROCA
CALL PROCB
PROCB
CALL "EXITS"

❑

4.6 The CALL "GETSP-U-CNTXT" Procedure

Syntax

```
CALL "GETSP-U-CNTXT" USING i- reqid,
                           i- area,
                           io-size,
                           o- address,
                           o- status.
```

Description

Gets space in user context.

Space can be allocated in PRIVATE-STORAGE or in TRANSACTION-STORAGE. The allocated subsets are identified by their REQID. For a given REQID, space is allocated upon the first call to GETSP-U-CNTXT. Further calls to GETSP-U-CNTXT with the same REQID may be issued in order to retrieve the address of the subset.

The MAXIMUM PRIVATE-STORAGE and/or MAXIMUM TRANSACTION-STORAGE clauses must be specified in the STDS.

Parameters

i-reqid	is a string of 8 alphanumeric characters identifying the user context. It is an input parameter. The name "-STATIC-" is reserved for the space subsets corresponding to the PRIVATE-STORAGE SIZE and the TRANSACTION-STORAGE size as defined in the STDS. The other subsets are the DYNAMIC subsets and their reqid is named by the user.
i-area	is a 1-character alphabetic field. Its value can be "P" for PRIVATE-STORAGE and "T" for TRANSACTION-STORAGE. It is an input parameter.

io-size	is a COMP-1 field that can be an input or output parameter. It must be specified at least the first time a part of the context is used. If io-size is equal to 0, the procedure puts the right size of the user context subset in io-size, if this one exists. Otherwise, o-status is equal to 6. If io-size is not equal to 0, the value must be the right size of the user context subset. If it is not the case, o-address is returned equal to NULL ptr and o-status to 2.
o-address	(pointer type) is the address of the beginning of the subset of privat-storage or transaction-storage (as it is specified in the i-area parameter) identified by i-reqid.
o-status	is a one-character output field. It gives the status of the operation.

0	the allocation is done
1	parameter error
2	the size parameter is specified and wrong
3	lack of space for the allocation
4	too many user contexts
5	no MAXIMUM TRANSACTION-STORAGE clause or no MAXIMUM PRIVATE-STORAGE clause is present.
6	the specified reqid does not exist

4.7 The CALL "GETTPRPAR" Procedure

Syntax

```
CALL "GETTPRPAR" USING data-name-1,
                       data-name-2,
                       data-name-3.
```

Description

Returns lengths and addresses of structures and the number of stacked contexts before the current transaction, depending on what is specified in the data-name-2 parameter.

The lengths and addresses are in pointer variables, and are for these TDS storage structures: TDS, CONSTANT, PRIVATE, and TRANSACTION. The transaction stacking numbers apply to these transactions: BREAK, LOGON, DISCNT, RESTART, and CONTEXT RANK.

Returns information at disconnection time of an XCP1 session.

Usage

`Data-name-1` has three formats.

Format 1:

The first format is a structure that contains the lengths and addresses of the TDS structures after this call. The COBOL statement COPY TDS-USER-AREA to obtain this structure, as follows:

```
01 TDS-USER-AREA.
    02 TDS-STORAGE-PTR  POINTER.
    02 TDS-STORAGE-LGTH COMP-2.
    02 CST-STORAGE-PTR  POINTER.
    02 CST-STORAGE-LGTH COMP-2.
    02 PRV-STORAGE-PTR  POINTER.
    02 PRV-STORAGE-LGTH COMP-2.
    02 TX-STORAGE-PTR   POINTER.
    02 TX-STORAGE-LGTH  COMP-2.
```

The parameters of this structure have the following meanings:

PTR	Pointer to storage
LGTH	Length of storage
TDS	TDS storage
CST	CONSTANT storage
PRV	PRIVATE storage
TX	TRANSACTION storage

The TX storage length is that defined in the STDS, less the length of the PRIVATE storage.

The TX storage length and the PRIVATE storage length include the space allocated by GETSP-U-CNTXT if any.

Format 2:

The second format is a COMP-1 field that returns the number of existing CONTEXT RANK contexts. This is shown in the following text:

```
     77 TDS-CONTEXT-RANK  COMP-1.
```

Format 3:

The third format is an output structure with the following description:

```
  01 RESULT.
      02 XCP1-TYPE          PIC X(2).
      02 XCP1-LAST-TPRNAME  PIC X(12).
      02 XCP1-LAST-USERID   PIC X(12).
```

This format is used when Data-name-2 is filled with "XCP1-01" (see Data-name-2 below).

XCP1-TYPE can be LP, LA or RP.

LA	means Local Auxiliary, an auxiliary XCP1 session allocated by the CP-ALLOCATE verb on the local side (where the OCPOOL command was entered).
LP	means Local Principal, an XCP1 session on the local side (where the OCPOOL command was entered).
RP	means Remote Principal, an XCP1 session on the remote side.

XCP1-LAST-TPRNAME is the name of the last TPR committed, if any, executed either for an auxiliary session or a principal session.

XCP1-LAST-USERID is the name:

- of the user working with the XCP1 session at disconnection time, if the XCP1-TYPE is LA,
- of the XCP1 correspondent for the other types.

This information is only available if the call "GETTPRPAR" is executed for an XCP1 correspondent in the first TPR of the DISCONNECT transaction. It is initialized with spaces at connection or reconnection time and during an "INIT-WORK" verb.

`Data-name-2` is an input parameter of 13 alphanumeric characters right padded with spaces. Valid values and their meanings are as follows:

TDS-USER-AREA	Indicates the structure to return in format 1.
NBCTXT	Indicates the structure to return in format 2.
XCP-01	Indicates the structure to return in format 3.

`Data-name-3` is a single character (PIC X) in which the status of the call is returned. Valid values and their meanings are as follows:

0	Successful call
1	Unknown option specified for DATANAME-2

If Data-name-2 is XCP1-01, the status 1 is returned if the call is not done in the disconnect transaction, or if the correspondent is not an XCP1 correspondent, or if the data-name-1 structure cannot be accessed.

4.8 The CALL "NOCANCELCTX" Procedure

Syntax

```
CALL "NOCANCELCTX" USING  data-name-1
```

Description

This procedure allows resumption explicitly of the user transaction interrupted by a disconnection or a system failure at this user reconnection time when the "CANCELCTX AT RECONNECTION" clause is present in the TDS generation file.

This procedure may only be called in the logon transaction and the last context of the user will be resumed only after the execution of the logon transaction.

If the main transaction was interrupted by one or several break transactions when the disconnection occurs, the whole stack of interrupted contexts of the user is restored after the execution of the logon transaction.

Usage

data-name-1 is a COMP-1 field containing the output status of the call as follows:

0	successful completion
1	procedure not called in the logon transaction

The procedure has no effect if it is called in the first logon transaction: the status returned will be 0.

The call to "NOCANCELCTX" AND THE "CANCELCTX" procedures are independent and may be called successively. The table below shows the result of one of these calls according to what has happened previously:

	current call			
	with the clause		without the clause	
previous call	"cancelctx"	"nocancelctx"	"cancelctx"	"nocancelctx"
nothing	1	2	1	4
"cancelctx"	1	3	1	4
"nocancelctx"	1	4	1	4

The references 1, 2, 3, 4 have the following meaning:

1	The action of the CANCELCTX procedure remains the same.
2	The transaction or the stack of interrupted transactions will be resumed at the end of the logon transaction.
3	The last stacked transaction has been unstacked by the CANCELCTX procedure. All the stacked transactions remaining will be resumed at the end of the logon transaction.
4	The call has no effect.

4.9 The CALL "RESTORE" Procedure

Syntax

```
CALL "RESTORE" USING RANK.
```

Description

Applicable when at least one context has been stacked for the BREAK, LOGON, RESTART, and DISCNCT transactions (these transactions are described in Chapter 12). On execution of RESTORE, the TRANSACTION-STORAGE is restored to the state it had in the previous context, which has been saved at the interrupt time. PRIVATE-STORAGE that is part of TRANSACTION-STORAGE is also restored.

RANK is a COMP-1 (or FIXED BIN-15) output parameter that contains the number of the context that has been restored. Zero means that no context has been restored; one means that the context number 1 (the main context) has been restored, and so on.

If there is no previous context, the CALL "RESTORE" is ineffective.

NOTE:
Do not use the CALL RESTORE in conjunction with the GETSP-U-CNTXT.

Figure 4-1 illustrates the use of CALL "RESTORE".

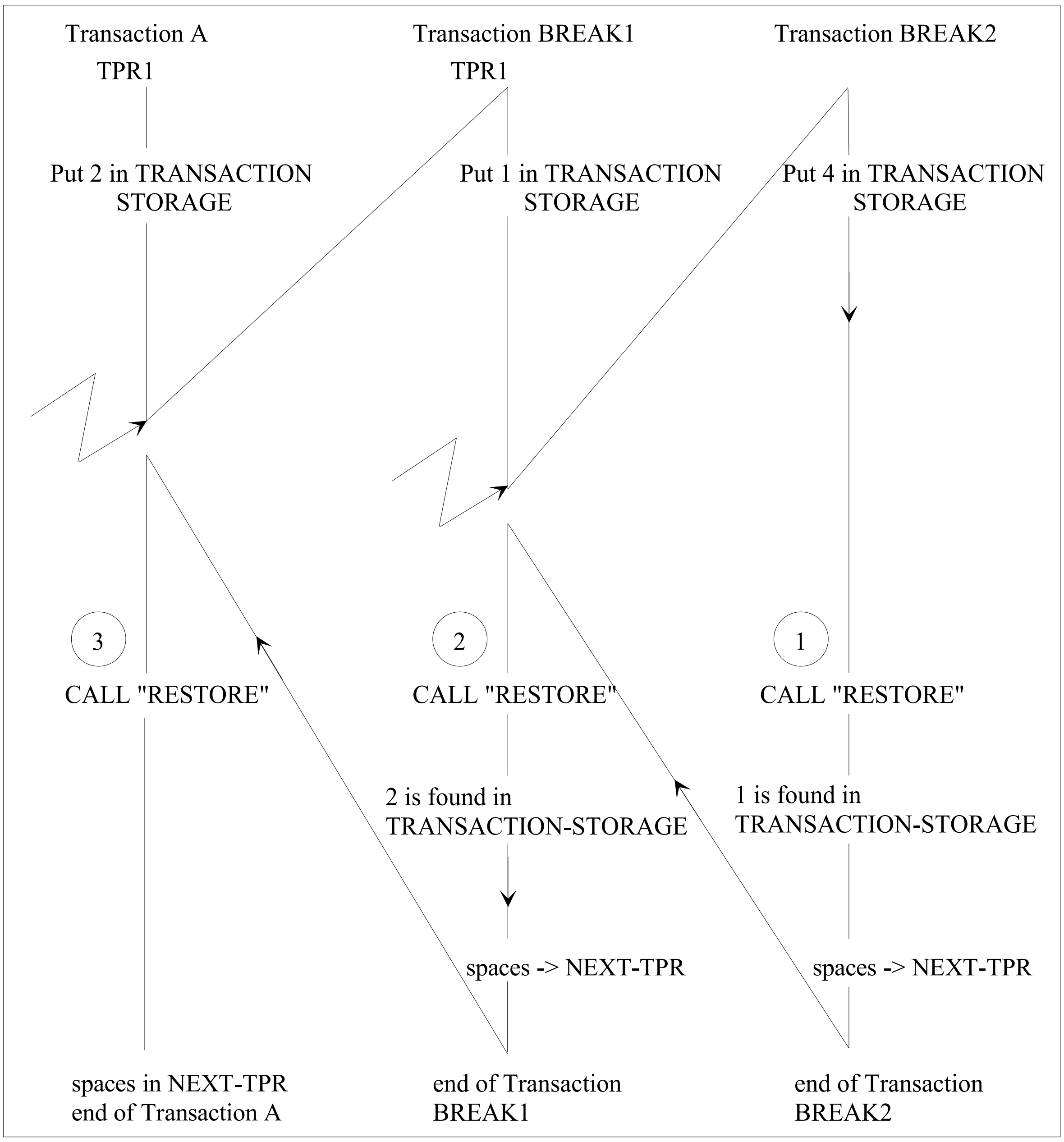

Transaction A
TPR1
Put 2 in TRANSACTION STORAGE
3
CALL "RESTORE"
spaces in NEXT-TPR
end of Transaction A
Transaction BREAK1
TPR1
Put 1 in TRANSACTION STORAGE
2
CALL "RESTORE"
2 is found in TRANSACTION-STORAGE
spaces -> NEXT-TPR
end of Transaction BREAK1
Transaction BREAK2
Put 4 in TRANSACTION STORAGE
1
CALL "RESTORE"
1 is found in TRANSACTION-STORAGE
spaces -> NEXT-TPR
end of Transaction BREAK2

Figure 4-1. Using CALL RESTORE

Transaction BREAK1 interrupts TPR1 of transaction A that is currently executing. Transaction BREAK2 interrupts transaction BREAK1. TDS stores any working data that it needs to continue BREAK1 and BREAK2 at a later time.

In transaction BREAK2, two previous contexts exist:

①   CALL "RESTORE" USING RANK.

After the execution of the CALL "RESTORE",

- RANK = 2
- TRANSACTION-STORAGE is restored with that of transaction BREAK1 (value = 1).

②   CALL "RESTORE" USING RANK

After the execution of the CALL "RESTORE",

- RANK = 1
- TRANSACTION-STORAGE is restored with that of transaction A (value = 2).

③   CALL "RESTORE" USING RANK.

This command is ineffective,

- After the execution of the CALL "RESTORE"
- RANK = 0.

4.10 The CALL "SIMBRK" Procedure

Syntax

```
CALL "SIMBRK" USING data-name-1, data-name-2 [data-name-3].
```

Description

Simulates a break for a correspondent. The BREAK transaction, if defined, will be activated.

The commitment option (wait commitment or rollback commitment), used when "SIMBRK" is called, is the one specified in the STDS for the transaction calling "SIMBRK". If this commitment option has not been specified, the one specified for the BREAK transaction will be used, but the one given for the transaction data-name-2 will never be used. See Chapter 12 for a description of BREAK transaction processing.

Usage

Data-name-1 is a userid of up to 8 alphanumeric characters identifying the correspondent. A userid of less than 8 is right padded with spaces. It is an input parameter.

Data-name-2 is an output parameter. It is a single numeric character that contains the status of the CALL at completion:

0	successful completion
1	unknown correspondent
2	correspondent found, but it has been logged off, or correspondent name is invalid (contains invalid characters such as '<').
3	wrong value for data-name-3 (see data-name-3 below).
5	correspondent is in pass-through mode; this verb is not allowed in this context.

Data-name-3 is an optional input parameter. It is a 7-character string. Three cases are to be considered:

- *data-name-3 is specified* with a value of "REALBRK" to indicate that the CALL "SIMBRK" treatment is exactly the same as if it was issued at the terminal. In this case, the BREAK transaction will be executed according to the specified COMMITMENT option as follows:
  - if WAIT COMMITMENT, at the next commitment point,
  - if ROLL-BACK COMMITMENT, at the end of the current TPR, after its rollback has been done. In this case, the restart status will be set to 2.
- *data-name-3 is specified* with a value other than "REALBRK", CALL "SIMBRK" will not be executed, and data-name-2 will be set to 3.
- *data-name-3 is not specified*, the CALL "SIMBRK" behavior is not exactly the same as if was issued at the terminal; the BREAK transaction will only be taken into account at the next commitment point (the current CU will not be rolled back even if ROLL-BACK COMMITMENT has been specified in the COMMITMENT option).

4.11 The CALL "SUBJOB" Procedure

Syntax

```
CALL "SUBJOB" USING job-description, status, file-description,
                    [dest].
```

Description

Submits a request for asynchronous job execution. A submitted job is stored in the Stream Reader queue and processed by the Stream Reader service asynchronously. The term asynchronous means that the TPR submitting the request can continue to do its own work while the submitted job becomes eligible for execution.

A request is validated only if the submitting commitment unit ends normally. In the case of rollback, the job submitted by the current commitment unit or TPR is canceled.

A job submission and its execution are asynchronous. The console and report messages are not directed to the submitting terminal (the user specified in the $JOB statement), but to the IOF mailbox of the user submitting the TPR which contains the CALL "SUBJOB" procedure.

The user can submit a job to a remote host and direct its output to a different destination.

For an example of how CALL "SUBJOB" is used, see Appendix D.

Note the following points:

$JOB/$ENDJOB present in the JCL of the submitted job:

You must specify the PROJECT and BILLING parameters in the $JOB statement of the submitted job if there are no corresponding default values in the site catalog; otherwise the submitted job aborts.

No $JOB/$ENJOB present in the JCL of the submitted job:

The project and billing of the user submitting the TPR which contains the CALL "SUBJOB" statement is used.

For more information, see the *JCL Reference Manual*.

Usage

Job-description is a data structure that contains the set of parameters applicable to the submitted jobs. The job description must be structured as follows:

```
01 JOB-DESCRIPTION.
 02 JOBDESC-STRUCT-LN          COMP-1.            ]
 02 JOB-CLASS                  PIC X.             ]
 02 JOB-PRIORITY               PIC 9.             ]
 02 JOB-SWITCHES               PIC X   OCCURS 32. ]
 02 JOB-DELETE                 PIC X.             ] 61 bytes
 02 JOB-HOST                   PIC X(4).          ]
 02 JOB-CLASS2                  PIC X(2).         ]
 02 JOB-SKIP-BLANK             PIC X(1).          ]
 02 JOB-NOMESSIOF              PIC X(1).          ]
 02 JOB-OUTDEST.                                  ]
    03  PRIMARY-DEST           PIC X(8).          ]
    03  SECONDARY-DEST         PIC X(8).          ]
 02 JOB-VALUES.
    03 VALUES-STRUCT-LN        COMP-1.
    03 VALUES-STRUCT.
       04 VALUES-STRUCT-HEADER.
          05 NB-OF-POSITIONAL  COMP-1.
          05 NB-OF-KEYWORD     COMP-1.
       04 VALUES-PARAMETERS    PIC X(m).
```

All the fields in JOB-DESCRIPTION are input parameters.

JOBDESC-STRUCT-LN	must contain the exact size in bytes of the JOB-DESCRIPTION structure. This value is typically 61 when values are not used.
JOB-CLASS	If specified, must be one of the sixteen classes from A to P under which the job will be submitted, scheduled and executed. If JOB-CLASS is not specified (space), you can use 2 characters for indicating the class of the submitted job (see JOB-CLASS2 below).
JOB-PRIORITY	represents the job scheduling priority such as can be specified in the $JOB statement. It must be in the range 0 to 7, where 0 is the highest and 7 is the lowest priority.

JOB-SWITCHES	represent the initial job switch values. You must initialize each of them to 0 or 1. They are numbered from 1 to 32 whereas the switches specified in the SWITCHES parameter of the ENTER_JOB_REQUEST directive are numbered from 0 to 31.
JOB-DELETE	must be set to Y (Yes) or N (No). If yes, the subfile which contains the job to be submitted will be deleted once the job has successfully executed. This parameter corresponds to the DELETE option of the $JOB statement.
JOB-HOST	is a DSA node name up to 4 characters long. It specifies the remote host name where the job must be executed. This parameter corresponds to the HOST option of the $JOB statement. If equal to spaces, the job is executed locally. The host name must be cataloged.
JOB-CLASS2	allows you to specify 2 characters for the class of the job to be submitted. If JOB-CLASS contains a value other than a space, then JOB-CLASS2 must be filled with spaces. If JOB-CLASS2 contains a value other than spaces, then JOB-CLASS must contain a space. Both JOB-CLASS and JOB-CLASS2 can be filled with spaces in which case the class of the submitted job is the default batch class for the project. If JOB-CLASS2 contains a class which is not available to the recipient of the submitted job, the class of the submitted job will be P.
JOB-SKIP-BLANK	must be set to Y (yes) or N (no). If yes, spaces to the right of the passed valued are suppressed. If no or any other value, spaces are kept. (Note that this was the default value in V5.)

JOB-NOMESSIOF	determines the destination of the messages IN, STARTED, COMPLETED, and OUTPUT COMPLETED. If JOB-NOMESSIOF=Y or y, messages are sent only to the main operator and not to the lof mailbox of the submitting user. Other values such as N, n, " ", or everything else lead messages to be sent to the main operator and to the lof mailbox of the submitting user. It is the way to avoid accumulating of messages in the lof mailbox of the submitting user (or in the lof mailbox of the TDS master if the parameter DEST is set to MASTER).
JOB-OUTDEST	the two fields of this substructure specify the output destination station. These parameters correspond to the DEST option of the JCL statements SYSOUT, WRITER, or OUTVAL. If both are blank, the job output is directed to the local main station. If the primary destination name is specified (not blank) and the secondary destination name is not initialized, then the job output is directed to the primary RBF station. If the secondary destination name is also initialized, then the output is directed according to the Distributed Job Processing algorithm.

The JOB-VALUES substructure contains the initial job values.

VALUES-STRUCT-LN	defines the size in bytes of the VALUES-STRUCT data structure. If 0, there is no value to be transmitted to the job.
NB-OF-POSITIONAL	defines the number of positional values that are described in the values parameter structure.
NB-OF-KEYWORD	defines the number of keyword values that are described in the values parameter structure. When the submitted job is a GCL procedure NB-OF-KEYWORD is equal to zero.

VALUES-PARAMETERS defines all the elements (positional and keyword values). This structure is the concatenation of all the elements. Each element must have one of the following descriptions. If the element is a positional value, then the description is as follows:

```
05 POSITIONAL.
06 POS-LENGTH COMP-1.
06 POS-VALUE PIC X(POS-LENGTH).
```

where POS-LENGTH is the size in bytes of the positional value.

If the element is a keyword value, then the declaration is as follows:

```
05 KEYWORD.
06 KW-LENGTH COMP-1.
06 KW-NAME   PIC X(8).
06 KW-VALUE  PIC X(KW-LENGTH).
```

where KW-LENGTH is the size in bytes of the keyword values and KW-NAME is the keyword name left justified and padded with spaces.

All the positional values must be declared before the keyword values.

If the default job value is chosen, you must specify the following for:

- a positional value:

```
06  POS-LENGTH COMP-1 VALUE 0.
(do not use POS-VALUE)
```

- a keyword value:

```
06  KW-LENGTH COMP-1 VALUE 0.
06  KW-NAME PIC X(8).
(do not use KW-VALUE)
```

If the positive value or keyword value is not completely filled (that is they contain spaces on the right), the spaces are passed to the reader and can lead to JCL translation errors. This can be seen in this example:

```
05 pos1
     06 POS1 LGTH COMP-1 VALUE 6
     06 POS  VL   PIC X(6) VALUE "ABC  "
```

In this example above, if POS1 VL is used in:

```
MVL A=&posvl_XYZ;
```

then the value of A is ABCbbb_XYZ and not ABC_XYZ.
When the submitted job is a GCL procedure the first POS-VALUE must contain:

```
MWINLIB BIN binary-library-name
            [:media:device-class];
```

library name is the name of the library where the GCL procedure is stored,

The second POS-VALUE must contain:
```
procedure name b
```

where:

`procedure name` is the name of the GCL procedure b represents the blank character suffix that you must add on to the procedure name.

The third and possibly the fourth POS-VALUE fields must contain the parameters of the GCL procedure Each POS-VALUE length is limited to 128 characters.

- Status is a data structure which defines the status of the CALL "SUBJOB" statement. It is an output parameter and must have the following data structure:

```
01 STATUS.
   02  RESULT       PIC 9.
   02  JOBID        COMP-2.

01 STATUSB REDEFINES STATUS
   02  RESULTB      PIC 9.
   02  ERROR-TYPE   COMP-1.
   02  ERROR-NUMBER COMP-1.
```

- `RESULT` can take three possible values. According to these values, either the JOBID field or the ERROR-TYPE and ERROR-NUMBER fields are significant.

RESULT = 0	Successful completion. A request to submit the job is made. The JOBID field of the status structure is filled with the Job Identifier and can be used to get information on the submitted job (see the CALL "JOBINFO" PROCEDURE paragraph). When the commitment unit successfully completes, the job is submitted asynchronously. However, the presence of a zero in RESULT does not necessarily indicate that the submitted job is successfully completed; an error can subsequently occur during execution, for example a JCL error is detected.
RESULT = 1	Abnormal completion. One of the input parameters is erroneous and the job is not submitted. In this case ERROR-TYPE takes the value 1 and ERROR-NUMBER takes one of the following values: 2 - wrong priority (JOB-PRIORITY). 4 - wrong class (JOB-CLASS). 13 - error in the value (JOB-VALUES). 26 - wrong switch (es) (JOB-SWITCHES). 27 - wrong subfile suppression value (JOB-DELETE). 29 - wrong syntax (OCL or GCL) in the file description (FILE-DESCRIPTION). 30 - correct syntax for the parameters but failure of job submission (for example, JOB-HOST correct syntax but non-existent). 33 -- error in the site name (for example, incompatibility between the local system and the remote host where the files resides).
RESULT = 2	Wrong file. The file (or the subfile) cannot be accessed and the job is not submitted. In this case, ERROR-NUMBER takes the value 0 and ERROR-TYPE takes one of the following values: 1 - the file description is wrong. 2 - the file cannot be assigned. 3 - the file cannot be opened. 4 - the subfile (if any) cannot be opened. 5 - problem when creating file management structure (H_FD).

- `File description` is a data structure which identifies the "file literal" description of the file which contains the job to be submitted.

It must have the following format:

```
01 FILE-DESCRIPTION.
   02  FILE-LITERAL-LENGTH  COMP-1.
   02  FILE-LITERAL         PIC X(FILE-LITERAL-LENGTH).
```

where `FILE-LITERAL-LENGTH` is the size in bytes of the FILE-LITERAL item.

`FILE-LITERAL` has the following format:

```
 [subfile-name:] external-filename [:media:device-class]
```

in Operator Control Language (OCL)

or:

```
 external-filename [..subfile-name] [:media:device-class]
```

in GCOS Command Language (GCL)

When the submitted job is a GCL procedure the format is as follows:

```
 SYS.HSLLIB..ABSENTEE
```

When the file is cataloged, it must be cataloged in the site catalog or a private auto-attachable catalog.

Media and device class identify respectively the volume and device on which the external filename resides. They must not be specified when the file is cataloged.

The file may be either a sequential file or a library member.

- `Dest` is an optional input parameter 6 characters long which allows you to direct Console and Report messages to the master terminal instead of to the submitting terminal. To do this, you specify "MASTER" in the dest parameter; any other value causes messages to be sent to the IOF mailbox of the user submitting the TPR which contains the CALL "SUBJOB" procedure. You must also specify "MASTER" in the dest parameter, if the job (submitted via CALL "SUBJOB") is to issue TDS Master Commands through the JCL statement EXDIR.

4.12 The CALL “JOBINFO” Procedure

Syntax

```
CALL "JOBINFO" USING data-name-1
```

Description

Allows you to know the state of a job submitted by the call "SUBJOB" procedure. This procedure does not work when the submitted job is executed on a remote host.

The job outputs must be held (using the HOLDOUT keyword in JCL) in order to be still known by the system after the job termination.

This procedure takes as input the Job Identifier of the submitted JOB (JOBID field of the STATUS structure when the call "SUBJOB" statement is successful), and returns in output the state of the job.

The procedure cannot be called in the same COMMITMENT as the call "SUBJOB" one because the job is really submitted at the normal end of the commit.

A way to get information about the submitted job is:

- to put the JOBID returned by the "SUBJOB" procedure in the Transaction storage
- to set a wait-time in the WAIT-TIME field of the TDS-STORAGE
- to commit the TPR and to call the "JOBINFO" procedure in a further commitment.

Usage

Data-name-1 is a structure having the following format:

```
01 JOB-STRUCT.
   02 JOBID          COMP-2.
   02 RESULT         COMP-1.
   02 RC             COMP-2.
   02 RON            COMP-1.
   02 JOBSTATE       COMP-1.
   02 SUSJOBSTATE    COMP-1.
```

JOBID is an input parameter that contains the Job Identifier.

RESULT gives the result of the "JOBINFO" procedure call:

- 0 - successful completion of the procedure.
- 1 - unsuccessful completion of the procedure.

RC is the RETURN CODE of the called JOB Management primitive (can be useful for debugging when RESULT = 1). RC can be edited in the following way:

```
01 EDITRC    PIC X (30).
   CALL "H_STD_UEDTG4" USING EDITRC ADDRESS OF RC.
   DISPLAY "RETURN CODE = " EDITRC.
```

When the JOBID refers to a job executed on a remote host, the returned RESULT is 1 and the RC is SCHDL24,NOMATCH.

RON is the Binary Ron of the submitted Job identified by JOBID (when RESULT = 0, otherwise RON = 0).
JOBSTATE is the state of the JOB (when RESULT = 0, otherwise JOBSTATE = 0).

JOBSTATE can take one of the following values:

- 1 - IN INTRODUCTION
- 2 - READ
- 3 - IN TRANSLATION
- 4 - TRANSLATED WITH ERROR
- 5 - HELD
- 6 - SCHEDULABLE
- 7 - IN EXECUTION
- 8 - SUSPENDED
- 9 - TERMINATED
- 10 - IDLE
- 11 - INTRODUCED WAITING INPUT

SUBJOBSTATE is a precision on the JOB termination when JOBSTATE has the value "TERMINATED" (otherwise SUBJOBSTATE = 0).

SUBJOBSTATE can take one of the following values:

- 0 - JOB COMPLETED
- 1 - USAGE OF $DATA
- 2 - NOT EXECUTED
- 3 - ABORTED
- 4 - JOB has been killed with CJ strong
- 5 - JOB has been killed with CJ weak

A JOB is completed when JOBSTATE = 9 and SUBJOBSTATE = 0.

4.13 The CALL "XSIMBRK" Procedure

Syntax

```
CALL "XSIMBRK" USING data-name-1
                     data-name-2
                     data-name-3
                     [data-name-4].
                     dataname-3.
```

Description

Simulates a 'break' for a correspondent. The transaction activated is the one specified in data-name-2.

The commitment option (wait commitment or rollback commitment) used when "XSIMBRK" is called is the one specified in the STDS for the transaction calling "XSIMBRK". If this commitment option has not been specified, the one specified for the BREAK transaction will be used, and not the option specified for the transaction given in data-name-2.

The same BREAK transaction processing (see Chapter 12) applies to the transaction specified in data-name-2.

Usage

Data-name-1 is a userid of 12 alphanumeric characters identifying the correspondent. A userid of less than 8 is right padded with spaces. It is an input parameter.

Data-name-2 is an input parameter of 8 characters containing the name of the transaction to be activated.

Data-name-3 is an output parameter. It is a single numeric character that contains the status of the call, as follows:

0 =	Successful
1 =	Unknown correspondent
2 =	Correspondent found, but frozen or logged off
3 =	Unknown transaction name in data-name-2
4 =	Unknown transaction (authority code checking failed)
5 =	correspondent is in pass-through mode; this verb is not allowed in this context.
6 =	wrong value for data-name-4 (see below).

Data-name-4 is an optional input parameter. It is a 7-character string. Three cases are to be considered:

- *data-name-4 is specified* with a value of "REALBRK" to indicate that the CALL "SIMBRK" treatment will be exactly the same as if it was issued at the terminal. In this case, the BREAK transaction specified in data-name-2 will be executed according to the specified COMMITMENT option as follows:
  - if WAIT COMMITMENT, at the next commitment point,
  - if ROLL-BACK COMMITMENT, at the end of the current TPR, after its rollback has been done. In this case, the restart status will be set to 2.
- *data-name-4 is specified* with a value other than "REALBRK", CALL "XSIMBRK" will not be executed, and data-name-3 will be set to 6.
- *data-name-4 is not specified*, the CALL "XSIMBRK" behavior is not exactly the same as if was issued at the terminal; the BREAK transaction will only be taken into account at the next commitment point (the current CU will not be rolled back even if ROLL-BACK COMMITMENT has been specified in the COMMITMENT option).

4.14 The ACCEPT Verb

Syntax 1

```
                         { DATE }
ACCEPT identifier FROM { DAY  } .
                         { TIME }
```

Syntax 2

```
ACCEPT cd-name MESSAGE COUNT.
```

Usage

Syntax 1

Syntax 1 applies as in a COBOL batch program.
The ACCEPT statement causes the information requested to be transferred to the data item specified by the identifier, according to the rules of the MOVE statement.
DATE, DAY and TIME are conceptual data items and, therefore, are not described in the COBOL program.
DATE is composed of the data elements year, month, and day. For example, July 1, 1990 would be expressed as 900701.
DAY is composed of the data elements year and day. For example, July 1, 1990 would be expressed as 90182.
TIME is composed of the data elements hours, minutes, seconds and hundredths of a second. For example, 2:41 P.M. would be expressed as 14410000.

Syntax 2

The ACCEPT statement causes the MESSAGE COUNT field specified for cd-name to be updated to indicate the number of messages that exist in a queue.

On execution of the ACCEPT statement, the contents of the area specified by a Communication Description entry (See Chapter 2) must contain at least the name of the symbolic queue to be tested. Testing the condition causes the contents of the data items referenced by data-name-7 (STATUS KEY) and data-name-8 (MESSAGE COUNT) of the area associated with the Communication Description entry to be updated.

For more information on ACCEPT, see the *COBOL 85 Reference Manual.*

4.15 The DISPLAY Verb

Syntax

```
DISPLAY  { identifier-1 }  {,identifier-2 } ...
         { literal-1    }  {,literal-2    }

              [{ SYSOUT                 }]
         UPON [{ [ ALTERNATE ] CONSOLE }].
              [{ device-name            }]
```

Usage

UPON determines the device on which data is output.

- SYSOUT is the default value. It is an output file. Normally any DISPLAY verbs in a TPR will be ignored at Run-Time. If the TRACE option is selected (see the DISP trace option in Chapter 13), any DISPLAY verbs will become operative and the information designated will be output together with the other TRACE information.
- If CONSOLE, ALTERNATE CONSOLE or device-name is specified, the data is output respectively to:

  the submitting terminal,

  an alternate terminal,

  the designated terminal.

Where the DISPLAY verb has more than one operand, the values displayed are in the same sequence as the operands entered.

For example, if the identifier TOTO contains 492 and the identifier TITI contains 641, then the following statement

```
DISPLAY TOTO, TITI
```

will produce the values 492, 641.

The statement DISPLAY "TOTO", "TITI" produces the values TOTO, TITI.

If the USE DISPLAY_IN_JOR clause is present in the TDS generation, then you can write data in the JOR using:

- DISPLAY "xx" or,
- DISPLAY "xx" UPON SYSOUT.

4.16 The EXIT Verb

Syntax

```
EXIT PROGRAM.
```

Description

PROGRAM must be specified in the EXIT statement of the main (or only) body of the TPR. One of the following occurs when the EXIT PROGRAM statement is executed:

- If the NEXT-TPR field of TDS-STORAGE contains a valid TPR name, TDS will load and activate the named TPR together with a copy of the TRANSACTION-STORAGE area. The TPR named could be the TPR being terminated; hence, a TPR can cause itself to be terminated and then loaded.
- If the NEXT-TPR field is blank, the TPR and the transaction are both terminated. (This field is reset to blanks by TDS before execution of each TPR).
- If the NEXT-TPR field contains an invalid TPR name, the transaction is aborted.
- The TPR identified as NEXT-TPR will become eligible for activation immediately if no WAIT-TIME has been specified and no message has been sent in the previous TPR, or after a delay delimited by:
  - the arrival of a response if the last TPR terminated with a SEND EGI statement,
  - the transmission of another message to the terminal when allowed (the last TPR terminated with a SEND EMI statement to the originating terminal),
  - the expiration of WAIT-TIME in TDS-STORAGE.

  In any of the above cases when the next TPR is the first of a commitment unit, the TPR will be activated only if the transaction to which it belongs is non-concurrent with the running transactions.

4.17 The STOP Verb

Syntax

```
STOP.
```

Description

STOP causes the transaction to abort.

4.18 The WRITE in User Journal verb

Syntax

```
WRITE data-name [FROM identifier-1].
```

Description

Outputs user-defined records to the user journal.

Usage

Data-name is the name of the record described in the user journal FD in the File Section.

Identifier-1 defines the area in memory containing the record to be written.

If no user journal is used for this application, WRITE is ignored.

5. Spawn Handling Procedures

5.1 Overview

This chapter deals with the following procedures:

DELSPAWN	cancels all the spawned transactions pending for the correspondent that initiates the transaction in which the call is submitted.
DSPAWN	directs a transaction towards a correspondent after a specified delay.
NBSPAWN	returns the number of valid spawned transactions on a correspondent, except those spawned with timer.
SPAWN	directs a transaction towards a correspondent (active or passive).
SPAWNTX	directs a transaction towards any active or passive correspondent.
TSPAWN	directs a transaction towards a correspondent at a specified time of day.

5.2 The CALL "DELSPAWN" Procedure

Syntax

```
CALL "DELSPAWN".
```

Description

Cancels all the spawned transactions pending for the correspondent that initiates the transaction in which the call is submitted.

The CALL "DELSPAWN" request takes effect when the commitment unit in which the call is performed ends normally.

The spawn requests taken into account are those which were validated before the CALL "DELSPAWN". This means that all the SPAWN, DSPAWN, TSPAWN, and SPAWNTX calls requested in a commitment unit containing the CALL "DELSPAWN" take effect at the end of the commitment unit. This is true even if these spawning requests are sent to the correspondent that started the transaction containing this commitment unit.

5.3 The CALL "DSPAWN" Procedure

Syntax

```
CALL "DSPAWN" USING  data-name-1,
                     data-name-2,
                     data-name-3,
                     data-name-4.
```

Description

Directs a transaction towards a correspondent after a specified delay. The spawned transaction becomes eligible for execution after the specified delay. If the specified correspondent is not connected, it is automatically connected before the transaction is activated.

Usage

Data-name-1 is an alphanumeric field of up to 8 characters. This field must contain the name of the correspondent to which the transaction is spawned. It is an input parameter.

Data-name-2 is an alphanumeric field of up to 46 characters. This field must contain a message identifying the transaction and its parameters in the same format as keyed in from a terminal. Data-name-2 is an input parameter.

Data-name-3 is a 1-character numeric field that defines the status after the execution of DSPAWN as follows:

0	successful completion
1	number of spawns by commitment unit has exceeded the maximum allowed. For an explanation of this limit, see the section Spawning a Transaction in Chapter 2.
2	unknown transaction (This may happen because the authority codes of the spawn conflict.)
3	unknown correspondent
4	no longer used.

Data-name-4 is a COMP-1 field specifying the delay, in seconds, to elapse before the transaction becomes eligible for execution.

If the delay is zero or negative, the spawn is immediate with MEDIUM priority. No error message or status is sent.

- If a MDTIME operator command is issued between the moment when the **call dspawn** is issued and the moment when the spawning is validated: delay is corrupted when:
  - local time is shifted back (for example from 10:20 to 10:10) the 10 mn difference are added to the delay.
  - local time is shifted forward (for example from 10:20 to 10:35) the 15 mn difference are subtracted to the delay. If the delay becomes negative, an immediate spawning is issued.
- When the delay has elapsed, the transaction becomes eligible for execution with the MEDIUM priority.

The Transaction Initialization routine (if any) is called twice:

- during the CALL "DSPAWN", with the requester of the spawn in the USERID parameter of the Transaction Initialization Routine.
- at the start of the spawned transaction, with the recipient of the spawn in the USERID parameter of the Transaction Initialization Routine. During this second call, if an invalid access right or unknown transaction name is detected, a message is sent to the master terminal.

  The recipient and the requester must have authority codes that allow the recipient and the requester to use the spawned transaction. Authority codes define the list of authorized TDS users at TDSGEN.

  For an explanation of the Transaction Initialization routine, see Chapters 2 and 12.

5.4 The CALL "NBSPAWN" Procedure

Syntax

```
CALL "NBSPAWN" USING data-name-1,data-name-2.
```

Description

CALL "NBSPAWN" returns the number of valid spawned transactions on a correspondent, except those spawned with timer.

Usage

Data-name-1 is an input field twelve characters long (PIC X(12)), which contains the correspondent name for which the information is required.

Data-name-2 is an output COMP-1 field (FIXED BIN(15)), in which the number of spawned transactions is returned.

5.5 The CALL "SPAWN" Procedure

Syntax

```
CALL "SPAWN" USING data-name-1,
                   data-name-2,
                   data-name-3,
                   [,data-name-4].
```

Description

Directs a transaction towards a correspondent (active or passive). The spawned transaction becomes eligible for execution at the end of the commitment unit that performs the CALL "SPAWN" statement.

Usage

Data-name-1 is a userid of up to 8 alphanumeric characters identifying the correspondent to which the transaction is spawned. It is an input parameter.

Data-name-2 is an alphanumeric field of up to 46 characters. It must contain a message identifying the transaction and its parameters in the same format as keyed in from a terminal. Data-name-2 is an input parameter.

Data-name-3 is a 1-character numeric field. It is an output parameter defining the status after the execution of the CALL "SPAWN" statement, as follows:

0	successful completion,
1	number of spawns by commitment unit has exceeded the limit. For an explanation of the limit, see the section on Spawning a Transaction in Chapter 2.
2	unknown transaction (This may happen because the authority codes of the spawn conflict),
3	unknown correspondent, or connection rejected for the correspondent. Since TS 9764, on a TCP/IP correspondent: - with the ATMI.DLL version < 3.0.6, data-name-1 contains neither a correspondent name; nor the string "H-TCPIP-CLI". - with the ATMI.DLL version >= 3.0.6, data-name-1 contains neither a correspondent name, nor the TERMID (of the spawner) provided at tpconnect time, nor the string "H-TCPIP-CLI" if the TERMID was not provided at tpconnect time.
4	wrong type; spawn towards this type of correspondent not allowed,
5	wrong priority.

Data-name-4 is an input parameter. It is a 1 decimal digit specifying the priority of the spawn:

1	express
2	high
3	medium (default)
4	low

The Transaction Initialization Routine (if any) is called twice when the CALL "SPAWN" statement is performed:

- first, with the requester of the spawn in the USERID parameter of the Transaction Initialization Routine.
- second, with the recipient of the spawn in the USERID parameter of the Transaction Initialization Routine.

  For an explanation of the Transaction Initialization routine, see Chapters 2 and 12.

  The recipient and the requester must have authority codes that allow the recipient and the requester to use the spawned transaction. Authority codes define the list of authorized TDS users at TDSGEN.

5.6 The CALL "SPAWNTX" Procedure

Syntax

```
CALL "SPAWNTX" USING data-name-1,
                     data-name-2,
                     data-name-3,
                     data-name-4,
                     data-name-5,
                     data-name-6.
```

Description

Directs a transaction towards any active or passive correspondent. The spawned transaction becomes eligible for execution at one of the following times:

- at the end of the commitment unit that performs the CALL "SPAWNTX" statement (when DELAY, and TIME-OF-DAY are set to zero in data-name-3),
- when the specified delay has elapsed (when DELAY is not set to 0),
- at a specified time of day (when TIME-OF-DAY is not set to 0).

Usage

Data-name-1 identifies the correspondent to which the transaction is spawned. The star convention can be used as described in Chapter 2. Data-name-1 is an input parameter containing 12 alphanumeric characters, left justified, and right padded with blanks.

Data-name-2 is an input parameter with the following data structure:

```
01 DATA-NAME-2.
  02 MSG-LENGTH COMP-1.
  02 MSG        PIC X(y).
```

where

- MSG-LENGTH is the length of the second parameter. The maximum value allowed is 130.
- "PIC X(y)" means that "y" must be replaced by a value supplied by the programmer. This value must be less than or equal to 130.
- The value specified in MSG-LENGTH determines the actual length of MSG.
- MSG contains a message-id identifying the transaction and its parameters in the same format as keyed in from a terminal. The parameters are not checked.
- The entire message is passed to the Transaction Initialization Routine, if any, in order to get the message-id.
- The message-id, if there is no Transaction Initialization Routine, is extracted from the beginning of the input string. Its maximum length is 8. When this length is less than 8, it must be separated from the remainder by a blank. The Transaction Initialization Routine is described in Chapters 2 and 12.

For example, if you specify 50 in MSG-LENGTH and assuming that you specified 90 in MSG, then only the first 50 characters in MSG are used.

Data-name-3 is an input parameter with the following data structure:

```
01 DATA-NAME-3.
   02 VERSION      COMP-1.
   02 DELAY        COMP-1.
   02 TIME-OF-DAY.
      03 HOUR      COMP-1.
      03 MINUTE    COMP-1.
      03 SECOND    COMP-1.
```

where

- VERSION must be set to zero.
- DELAY specifies the delay in seconds, to elapse before the transaction becomes eligible for execution. DELAY must be set to zero when it is not used.
- If a MDTIME operator command is issued between the moment when the **call dspawn** is issued and the moment when the spawning is validated: delay is corrupted when:
  - local time is shifted back (for example from 10:20 to 10:10), the 10 mn difference is added to the delay.
  - local time is shifted forward (for example from 10:20 to 10:35), the 15 mn difference is subtracted from the delay. If the delay becomes negative, an immediate spawning is issued.

- TIME-OF-DAY defines when the spawned transaction will be eligible for execution on the day:
  - HOUR must range from 0 to (22 X 24), that is, the limit is 528 or about 3 weeks from the current date.
  - MINUTE must range from 0 to 59.
  - SECOND must range from 0 to 59.
- Only one of the fields DELAY, or TIME-OF-DAY can be non-zero, otherwise an error occurs. The field that is filled with zeros determines the type of spawn.
- If both DELAY and TIME-OF-DAY are filled with zeros, then the spawn is immediate.

Data-name-4 is a 1-character numeric field. This input parameter specifies the priority of the spawn:

1	express
2	high
3	medium
4	low

It is significant only for an immediate spawn. When DELAY, or TIME-OF-DAY is not zero, the priority must be set to MEDIUM.

For an explanation of the priorities, see Chapter 2.

Data-name-5 is a 2-character numeric field. This output parameter defines the status after the execution of the CALL "SPAWNTX" statement as follows:

00	successful completion.
10	spawn table overflow, that is, the total number of spawns waiting for validation is exceeded (see Spawning a Transaction in Chapter 2).
11	the maximum number of spawns exceeds the number allowed per commitment unit (see Chapter 2 for more information on limits).
20	DELAY and TIME-OF-DAY mixed.
21	TIME-OF-DAY already reached.
22	TIME-OF-DAY is not valid (negative, or out of range).
23	DELAY is not valid (negative).
30	wrong priority. A priority can be 1, 2, 3, or 4.
31	priority for deferred spawns must be medium (equal to 3).

40	transaction name is not valid (blank).
41	transaction name is not found.
42	authority code of the requester of the spawn does not match that of the spawned transaction.
43	authority code of the recipient of the spawn does not match that of the spawned transaction.
44	in a TDS-HA, do not spawn a CMA transaction.
50	correspondent name is not valid (blank, or with illegal characters).
51	connection has been requested, but was not successful.
52	reconnection has been requested, but was not successful.
53	the correspondent (with star convention) is unknown.
54	the state connection of the corespondent is unstable, retry later.
55	wrong type: spawn on a primary active correspondent is forbidden if it is a XCP1 correspondent.
56	wrong type: spawn towards a correspondent which is not passive, whereas only a passive correspondent was specified at TDSGEN (via the clause USE PASSIVE-SPAWN-CHECK).
57	wrong type: spawn towards a XCP2 correspondent.
58	wrong type of correspondent.
59	the spawned transaction tried to use the XCP2 service, but the XCP2 service is not available.
60	Since TS 9764, on a TCP/IP correspondent: - with the ATMI.DLL version < 3.0.6, data-name-1 contains neither a correspondent name; nor the string "H-TCPIP-CLI". - with the ATMI.DLL version >= 3.0.6, data-name-1 contains neither a correspondent name, nor the TERMID (of the spawner) provided at tpconnect time, nor the string "H-TCPIP-CLI" if the TERMID was not provided at tpconnect time.

`Data-name-6` is an output parameter declared as PIC X(2). `Data-name-6` is used by the Service Center for debugging purpose (when `data-name-5` contains 51 or 52, `data-name-6` gives the reason why the connection was unsuccessful). It is not to be tested by the application program.

5.7 The CALL "TSPAWN" Procedure

Syntax

```
CALL "TSPAWN" USING data-name-1,
                    data-name-2,
                    data-name-3,
                    data-name-4.
```

Description

Directs a transaction towards a correspondent at a specified time of day. The spawned transaction becomes eligible for execution at a specified time of day. If the specified correspondent is not connected, it is automatically connected before the transaction is started.

Usage

Data-name-1 is a userid of up to 8 alphanumeric characters identifying the correspondent to which the transaction is spawned. It is an input parameter.

Data-name-2 is an alphanumeric field of up to 46 characters. It must contain a message identifying the transaction and its parameters in the same format as keyed in from a terminal. Data-name-2 is an input parameter.

Data-name-3 is a 1-character numeric field which defines the status after the execution of the CALL "TSPAWN" statement as follows:

0	successful completion
1	number of spawns by commitment unit has exceeded the maximum allowed. For an explanation of this limit, see Chapter 2.
2	unknown transaction (This may happen because the authority codes of the spawn conflict).
3	unknown correspondent, or connection rejected for the correspondent.
4	no longer used.
5	time-of-day already reached
6	data-name-4 is not a valid time-of-day.

Data-name-4 is an 01 level data-name that defines when the spawned transaction will be eligible for execution on the day. The format of this data item is as follows:

```
01  data-name-4.
    02  hour     COMP-1.
    02  minute   COMP-1.
    02  second   COMP-1
```

where:

- `hour` must range from 0 to (22 x 24), that is, the limit is 528 or about three weeks from the current date,
- `minute` must range from 0 to 59,
- `second` must range from 0 to 59.

NOTES:

The Transaction Initialization routine (if any) is called twice:

1. During the CALL "TSPAWN", with the requester of the spawn in the USERID parameter of the Transaction Initialization Routine.
2. At the start of the spawned transaction, with the recipient of the spawn in the USERID parameter of the Transaction Initialization Routine. If an invalid access right or unknown transaction name is detected, a message is sent to the master terminal.

   For an explanation of the *Transaction Initialization routine*, see Chapters 2 and 12.

   The recipient and the requester must have authority codes that allow the recipient and the requester to use the spawned transaction. Authority codes define the list of authorized TDS users at TDSGEN.

❑

6. Correspondent Pool Handling Procedures

6.1 Overview

This chapter deals with the following procedures:

CLOSE-POOL	closes one or more pools.
DISP-COR	displays static (as declared in the NETGEN), or dynamic characteristics of the specified correspondent.
DISP-POOL	displays static (as declared in NETGEN), or dynamic (current) characteristics of the specified pool.
GET-TDS-STAT	gives detailed statistics of correspondents and files of a TDS that is running.
LIST-COR	lists the correspondents declared in NETGEN (static list), or connected to the TDS application (dynamic list).
LIST-POOL	lists the XCP2 pools declared in NETGEN (static list), or running for a correspondent (dynamic list).
MODIFY-POOL	modifies the characteristics of a session pool.
OPEN-POOL	opens one or all session pools between a local TDS application and a partner application.

6.2 The CALL "CLOSE-POOL" Procedure

Syntax

```
CALL "CLOSE-POOL" USING corresp-name, all-option
                                      pool-name,
                                      close-option,
                                      drain-sc,
                                      drain-tg,
                                      status.
```

Description

The equivalent GCL master command is [M] CLOSE_COR_POOL... (described in the *TDS Administrator's Guide*).

XCP2:
Closes one or all session pools between a local TDS application and a partner application. A pool is identified by the local name of the partner correspondent, and a pool name.

XCP1:
The pool is the object named "ATTRIBUTE" in the XCP1 protocol definition and in the master command [M] CLOSE_COR_POOL ...

You must already have opened the pool(s) by the CALL "OPEN-POOL" procedure, or by the master command [M] OPEN_COR_POOL ...

The pool(s) can be closed in 2 ways:

- normally, it is closed after all conversations have been deallocated (XCP2), or sessions became unused (XCP1).
- abnormally, conversations are aborted and the transactions using these conversations receive the status "dealloc-abend" (XCP2). For XCP1, transactions using the allocated sessions are aborted.

Input parameters

corresp-name is a string of 12 alphanumeric characters defining the partner application. It must be a name of a XCP2COR object, or a XCP1COR object previously connected to the TDS application. "DUMMY" characters are not accepted.

all-option is a 1-character alphabetic field as follows:

Y	(the command applies to all opened pools).
N	(the command applies to the pool specified in pool-name).

pool-name is specified only if the all-option field contains N. For XCP2, this is a string of 8 alphanumeric characters. For XCP1, the first 2 bytes must contain the pool attribute and the remaining characters must be blank. If all the characters are blank, TDS takes as the pool attribute the first 2 characters of the correspondent name.

close-option is a 1-character alphabetic field for determining how the pool(s) must be closed:

N indicates Normal termination.

For XCP2, sessions of the specified pool(s) are closed immediately if no conversation uses them; otherwise they are closed after the conversations using them end.

For XCP1, sessions are freed when the transactions (if any) using them ends; when all sessions of the pool are released, the session pool is deleted.

S indicates Strong (or abnormal termination).

For XCP2, conversations using sessions of the specified pool(s) are abnormally terminated and session pool(s) are closed after transactions using these aborted conversations have received the abnormal status and have deallocated these conversations.

For XCP1, all sessions are immediately disconnected (any related transactions abort) and the session pool is deleted.

Note that N has same effect on the pool as the [M] CANCEL_TDS_COR STRONG=0 command has, whereas S has the same effect as the [M] CANCEL_TDS_COR STRONG=1 command.

drain-sc/drain-tg are 1-character alphabetic fields for determining what is to be done with the enqueued conversations before the XCP2 pools are closed.

drain-sc and/or drain-tg apply only to XCP2 pools.

- you must specify drain-sc and/or drain-tg when the all-option parameter is set to Y.
- drain-sc and/or drain-tg are not taken into account if the close-option is set to S (they are forced to 0 by TDS), or if all-option is set to N.
- drain-sc must be set to one of the following values:

  Y indicates YES, that is, the local application processes (drains) the enqueued conversation-requests.

  N indicates NO, that is, the local application rejects (does not drain) the enqueued conversation-requests.
- drain-tg must be set to one of the following values:

  Y indicates YES, that is, the remote application processes (drains) the enqueued conversation-requests.

  N indicates NO, that is, the remote application rejects (does not drain) the enqueued conversation-requests.

Output parameters

These are returned only for XCP2 pools.

drain-sc and drain-tg contain values returned by the PPC even if the all-option is set to N, or if the close-option is set to S.

status refers to the following data structure that you can copy from the <tdsname>.COBOL file by using the COBOL statement `COPY H-DC-TP-STAT`.

```
05 coc-status.
06 coc-status-version    PIC X  VALUE 1.  input=1
06 coc-external-status   PIC X.
06 coc-system-status.
   07 coc-issuer         PIC X.
   07 coc-code           COMP-1.          status code
   07 coc-subcode        COMP-1.          status subcode
   07 coc-last-rc        COMP-2.
```

The field coc-external-status can have one of the following values:

coc-argerr (2)	The corresp-name field is not found. The all-option field is not found. The pool-name field is not found. The close-option field is not found. The drain-sc field is not found. The drain-tg field is not found. coc-status-version does not equal to 1. corresp-name refers to a XCP1 correspondent and the last 6 characters of pool-name are not blank. all-option is neither Y nor N. drain-sc is neither Y nor N. drain-tg is neither Y nor N. close-option is neither N nor S.
coc-arviol (3)	The transaction issuing the CLOSE-POOL has not been submitted by the master.
coc-done (0)	Request is performed.
coc-notdone (9)	PPC or TDS incident occurred. Contact the Service Center.
coc-notop (A)	The pool is not open (returned only for a XCP2 context).
coc-objunkn (B)	corresp-name unknown to TDS and PPC. pool-name is unknown to the PPC (returned only for a XCP2 context).
coc-resnav (D)	corresp-name refers to a XCP2 correspondent and the XCP2 service is not started.
coc-typeerr (G)	corresp-name equals "DUMMY". corresp-name refers to a wrong correspondent name. The specified correspondent refers to a TM or dummy correspondent connected to the TDS application.
coc-wrongpar (H)	An error occurred on the TDS/PPC interface. Contact the Service Center.

6.3 The CALL "DISP-COR" Procedure

Syntax

```
CALL "DISP-COR" USING corresp-name, option,
                                    cor-attrib,
                                    status.
```

Description

Displays static (as declared in the NETGEN), or dynamic characteristics of the specified correspondent.

The dynamic characteristics can be supplied only if the correspondent is currently connected to the TDS application.

The static characteristics are supplied only if the correspondent can be reached by the TDS application submitting the request, that is,

- for an XCP2 correspondent declared as "parallel" in NETGEN, at least one pool must have been declared in NETGEN and this pool must link this correspondent to the TDS submitting the request.
- for an XCP1 correspondent, the correspondent must be primary, that is, an "open-pool" request is performed toward this correspondent.
- for a terminal (TM correspondent), there is no restriction, that is, the correspondent is always displayed.

The equivalent GCL master command is [M] LIST_TDS_COR... (described in the *TDS Administrator's Guide*).

Usage

corresp-name is a string of 12 alphanumeric characters defining the local correspondent name of the partner application. If the option parameter is set to D, corresp-name can be the name of a virtual correspondent such as DUMMY.

option is a 1-character alphabetic field indicating which characteristics are to be displayed. The option input parameter must contain one of the following values:

S	indicates Static parameters.
D	indicates Dynamic parameters.

Output parameters

cor-attrib refers to a structure that you can obtain by using the COBOL statement `COPY H-DC-TP-DISPCOR`:

```
05 coc-cor-attrib.
   06 coc-cor-attrib-version  PIC X VALUE 1. input=1
   06 coc-cor-type            PIC X.         COC-TM, COC-XCP1,
                                             COC-XCP2 or COC-DUMMY
   06 coc-cor-address.
      07 coc-sess-control     PIC X(4).
      07 coc-mailbox          PIC X(8).
      07 coc-extension        PIC X(4).      blank for tm and xcp2
   06 coc-cor-attrib-static.
      07 coc-cor-backup       PIC X(12).
      07 coc-cor-profile      PIC X(6).
   06 coc-cor-attrib-dynamic.
      07 coc-cnb-of-pool      COMP-1.        current nb of pools
      07 coc-mxalcses         COMP-2.        max nb of allocated
                                             sessions
      07 coc-cumalcses        COMP-2.        cumulated nb of
                                             allocated sessions
      07 coc-failalcses       COMP-2.        nb of allocated
                                             sessions which failed
      07 coc-txacc            COMP-2.        transaction account
      07 coc-tpracc           COMP-2.        TPR account
      07 coc-state            PIC X.         C -> Connected
                                             D -> Disconnected
```

Static Characteristics

The output fields returned are: `coc-cor-type`, `coc-cor-address`, and `coc-cor-attrib-static`.

The contents of the `coc-cor-profile` parameter depends on the correspondent type:

- a TM correspondent's profile can be mapped on to the following structure:

```
06 profile.
   07 *           PIC X(4).
   07 cross       PIC X(1).      Y or N
   07 first       PIC X(1).      Y or N if cross=Y,
                                    U otherwise
```

- an XCP1 correspondent's profile can be mapped on to the following structure:

```
06 profile.
   07 primary        PIC X(1).    always Y
   07 active         PIC X(1).    Y or N
   07 init-work      PIC X(1).    Y or N
   07 debug          PIC X(1).    Y or N
   07 cross          PIC X(1).    Y or N
   07 first          PIC X(1).    Y or N if cross=Y,
                                  otherwise U
```

- an XCP2 correspondent's profile can be mapped on to the following structure:

```
06 profile.
   07 primary                PIC X(1).  Y or N
   07 parallel               PIC X(1).  Y or N
   07 winner                 PIC X(1).  Y or N
   07 max-sync-lvl           PIC X(1).  C -> Confirm
                                        S -> Syncpoint
   07 cross                  PIC X(1).  Y or N
   07 first                  PIC X(1).  Y or N if cross=Y,
                                        U otherwise
```

Dynamic Characteristics

The output fields returned are: coc-cor-type, coc-cor-address and coc-cor-attrib-dynamic. For a TM correspondent connected to a TDS application via IOF (for example, the TDS submitter), then the coc-cor-address field is filled with blanks.

The coc-extension field is significant only for a XCP1 secondary passive correspondent; otherwise, it is filled with blanks.

coc-cnb-of-pool is significant only for XCP2 and XCP1 primary correspondents; otherwise, 0 is returned.

coc-mxalcses, coc-cumalcses, coc-failalcses are returned only for XCP1 primary correspondents; otherwise these fields are set to 0.

coc-txacc, coc-tpracc are returned only for active secondary XCP1 correspondents, DUMMY correspondents, or TM correspondents; otherwise these fields are set to 0.

status refers to the following data structure that you can copy from the <tdsname>.COBOL file by using the COBOL statement `COPY H-DC-TP-STAT`.

```
05 coc-status.
06 coc-status-version    PIC X  VALUE 1.  input=1
06 coc-external-status   PIC X.
06 coc-system-status.
   07 coc-issuer         PIC X.
   07 coc-code           COMP-1.          status code
   07 coc-subcode        COMP-1.          status subcode
   07 coc-last-rc        COMP-2.
```

The field coc-external-status can have one of the following values:

coc-argerr (2)	The corresp-name field is not found. The option field is not found. The coc-cor-attrib field is not found. coc-status-version does not equal 1. coc-cor-attrib-version does not equal 1. corresp-name equals DUMMY or is blank. option is neither S nor D.
coc-arviol (3)	The transaction issuing the CALL "DISP-COR" was not submitted by the master.
coc-busy (4)	NETGEN is being updated. Retry later.
coc-done (0)	Request is performed.
coc-dupname (8)	corresp-name has a multiple definition (TM, XCP1, XCP2) in NETGEN.
coc-notdone (9)	TDS incident (returned only if option equals D). PPC incident (returned only for a XCP2 context and if option equals D). NETGEN incident. Contact the Service Center.

coc-objunkn (B)	corresp-name is unknown (returned only for a XCP2 context and if option equals D). corresp-name is unknown (returned only if option equals S). corresp-name is unknown (returned only if it is not a XCP2 context and if option equals D). corresp-name refers in NETGEN to a XCP1 correspondent which is not primary (returned only if option equals S). corresp-name refers in NETGEN to a XCP2 correspondent which is not reachable by TDS (because it is parallel and no pool is declared from the current TDS towards the specified correspondent) (returned only if option equals S).
coc-ogenunkn (C)	NETGEN was incrementally generated while TDS was processing the command. Retry later.
coc-resnav (D)	corresp-name refers to an XCP2 correspondent and the XCP2 service is not started, or the TPR executing the call belongs to a transaction that is not allowed to use XCP2 (the TDS generation clause XCP2 SERVICE USED does not exist for this transaction).
coc-typeerr (G)	corresp-name refers to a wrong correspondent name. The specified correspondent refers to a correspondent connected to the TDS application that is not XCP1, TM, or dummy (returned only if option equals D).
coc-wrongpar (H)	Error occurred on the TDS/PPC Interface. Contact the Service Center (returned only for a XCP2 context and if option equals D).

6.4 The CALL "DISP-POOL" Procedure

Syntax

```
CALL "DISP-POOL" USING corresp-name, pool-name,
                                     option,
                                     pool-attrib,
                                     status.
```

Displays static (as declared in NETGEN), or dynamic (current) characteristics of the specified pool. Dynamic characteristics can be displayed only if the pool is opened.

Static characteristics can be obtained for XCP2 pools only.

Static characteristics can be displayed only if the pool has been declared for the TDS submitting the request, that is, in NETGEN, the pool is defined with the same XCP2WKS as that corresponding to the TDS application.

The equivalent GCL master command is [M] LIST_COR_POOL... (described in the *TDS Administrator's Guide*).

Input parameters

corresp-name is a string of 12 alphanumeric characters for defining the local correspondent name of the partner application.

pool-name:

XCP2 correspondent	pool-name is a string of 8 alphanumeric characters for defining the pool.
XCP1 correspondent	The first 2 characters must contain the pool attribute and the last 6 characters must be blank characters. If all characters are blank, the TDS application will take as the pool attribute the first 2 characters of the correspondent name.
DUMMY correspondent	pool-name must be filled with blank characters

option is a 1-character alphabetic field for selecting the characteristics to be displayed. It must contain one of the following values:

S	indicates Static parameters.
D	indicates Dynamic parameters.

Only the D value is accepted for a XCP1, or a DUMMY correspondent.

Output parameters

pool-attrib refers to a structure that you can copy from the <tdsname>.COBOL file by using the COBOL statement COPY H-DC-TP-DISPPOOL:

```
05 coc-disp-pool.
   06 coc-disp-pool-version  PIC X VALUE 1. input=1
   06 coc-dispp-pool-x2.
      07 coc-max-ses-nb      COMP-1.  max no. of sessions
      07 coc-min-win-source  COMP-1.  min no. of winner
                                      sessions for source
      07 coc-min-win-target  COMP-1.  min no. of winner
                                      sessions for target
      07 coc-win-auto-activ  COMP-1.  no. of winner auto-active
                                      sessions
      07 coc-drain-source    PIC X.   default value for drain
                                      option for local
                                      application
      07 coc-drain-target    PIC X.   default value for drain
                                      option for partner
                                      application
      07 coc-max-synclvl     PIC X.   max synchronization
                                      level:
                                      C -> confirm,
                                      or S -> syncpoint
      07 coc-cur-ses-count   COMP-1.  current no. of sessions
      07 coc-cur-win-source  COMP-1.  current no. of winner
                                      sessions for source
      07 coc-cur-win-target  COMP-1.  current no. of winner
                                      sessions for target
      07 coc-trans-count     COMP-1.  no. of sessions in transient
                                      state
   06 coc-dispp-pool-dux1.
      07 coc-connect-count   COMP-1.  current no. of connected
                                      sessions
      07 coc-trans-count     COMP-1.  current no. of transient
                                      sessions
      07 coc-frozen-count    COMP-1.  current no. of frozen
                                      sessions
```

Since TS 7356, the H_SX-TP7-DISPOOL subfile has been modified in the <tdsname>.COBOL file by adding the following declarations after'07 coc-frozen-count COMP-1.'

```
06 coc-dispp-pool-dux1ses.
   07 coc-entry-count     COMP-1.
   07 coc-entry           occurs 100.
   08 coc-name            pic x(4).
   08 coc-state           COMP-1.
   08 coc-resstate        COMP-1.
   08 coc-correspname     pic x (12)
```

- COC-ENTRY-COUNT is the number of returned entries in the COC-ENTRY substructure.
- COC-NAME contains the session name (ATTRIBUTE concatenated with two digits identifying the session: AT00, AT01... AT99).
- COC-STATE values are:

  ```
  1 = LOGON, 2 = LOGGED, 3 = DISC, 4 = FROZEN, 5 = UNLOGGED.
  ```

- COC-RESSTATE values are:

  ```
  1 = PERMANENT, 2 = FREE, 3 = ALLOCPRINC, 4 = ALLOCAUX,
  12 = FREE/NALC
  ```

- COC-CORRESPNAME contains the name of the user whom the session has been allocated to when the COC-RENAME value is 4 (ALLOCAUX).

To obtain the additional information, the field COC-DISP-POOL-VERSION must be set to 2.

See the master command LST_COR_POOL in the *TDS Administrator's Guide* for more details.

XCP2:	The three fields coc-cur-ses-count, coc-cur-win-source, and coc-cur-win-target in coc-dispp-pool-x2 are filled only when you select the dynamic option; otherwise, they are meaningless.
	The content of coc-drain-source (coc-drain-target) depends on the value you specify in the option field:
	Static option: contains the values declared in NETGEN, that is, Y (Yes), or N (No).
	Dynamic option: is set to Y if draining is requested either locally for coc-drain-source, or remotely for coc-drain-target when the CALL "CLOSE-POOL" statement, or the [M] CLOSE_COR_POOL is executed. Otherwise, it is set to N.
XCP1 or DUMMY:	The three fields of coc-dispp-pool-dux1 are filled.

status refers to the following data structure that you can copy from the <tdsname>.COBOL file by using the COBOL statement `COPY H-DC-TP-STAT`.

```
05 coc-status.
06 coc-status-version    PIC X  VALUE 1.  input=1
06 coc-external-status   PIC X.
06 coc-system-status.
   07 coc-issuer         PIC X.
   07 coc-code           COMP-1.          status code
   07 coc-subcode        COMP-1.         status subcode
   07 coc-last-rc        COMP-2.
```

The field coc-external-status can have one of the following values:

coc-argerr (2)	The corresp-name field is not found. The pool-name field is not found. The option field is not found. The coc-disp-pool field is not found. coc-status-version does not equal 1. coc-disp-pool-version does not equal 1. corresp-name equals "DUMMY" or is blank. corresp-name refers to a TM correspondent (returned only if option equals "D"). corresp-name refers to a correspondent type other than XCP2 (returned only if option equals "S"). corresp-name refers to a XCP1 correspondent and the last 6 characters of pool-name are not blanks. corresp-name equals "DUMMY" and pool-name is not blank. pool-name is blank (returned only if option equals "S"). option is neither S nor D.
coc-arviol (3)	The transaction issuing the DISP-POOL was not submitted by the master.
coc-busy (4)	NETGEN is being updated. Retry later.
coc-done (0)	Request is performed.
coc-dupname (8)	Corresp-name has a multiple definition (TM, XCP1, XCP2) in NETGEN.
coc-notdone (9)	Internal error (returned only if option equals D). Internal error (returned only for a XCP2 context and if option equals D). NETGEN incident. Contact the Service Center.
coc-notop (A)	The pool is not open, i.e. there are no active sessions for that pool (returned only for a XCP1 or dummy context and if option equals D).

coc-objunkn (B)	corresp-name or pool-name unknown to ppc (returned only if it is a XCP2 context and if option equals D). corresp-name was not declared in NETGEN. pool-name was not declared in NETGEN.
coc-ogenunkn (C)	NETGEN was incrementally generated while TDS was processing the command. Retry later.
coc-resnav (D)	corresp-name refers to an XCP2 correspondent and the XCP2 service is not started, or the TPR executing the call belongs to a transaction that is not allowed to use XCP2 (the TDS generation clause XCP2 SERVICE USED does not exist for this transaction).
coc-typeerr (G)	Corresp-name refers to a wrong correspondent name. The specified correspondent refers to a correspondent connected to the TDS application that is not XCP1, TM, or DUMMY (returned only if option equals D). Corresp-name refers to a secondary XCP1 correspondent (the command is only allowed for a primary correspondent) (returned only if option equals D).
coc-wrongpar (H)	Error occurred on the TDS/PPC interface. Contact the Service Center (returned only for a XCP2 context and if option equals D).

6.5 The CALL "GET-TDS-STAT" Procedure

Syntax

```
CALL "GET-TDS-STAT" USING data-name1, ,...data-nameN.
```

Description

Gives detailed information and general statistics about the correspondents and files of a running TDS. Each parameter requests a different type of information about the running TDS. There can be up to 15 parameters (N). The parameter is at the "01" group level name of the structures that receive the collected information.

The parameters specified in the TRS-CONTROL structure determine the information that this procedure passes to GET-TDS-STAT. The only obligatory parameter is TRS-CONTROL. The other parameters are not obligatory and can be specified more than once, in the following order:

TRS-CONTROL	First parameter
TRS-GENERALINFO	Optional, always after TRS-CONTROL.
TRS-USERINFO	Optional, always after the TRS-CONTROL parameter and any TRS-GENERALINFO parameters (if they exist).
TRS-FILEINFO	Optional, always after the TRS-CONTROL parameter and any TRS-GENERALINFO or TRS-USERINFO parameters (if they exist).

The copy members are available in the TDSNAME.COBOL after a TDS generation.

This procedure has the following limits:

- The TRS procedure can return information for a maximum of 4096 users. A maximum of 14 TRS-USERINFO structures can be passed to GET-TDS-STAT.
- The TRS procedure can return information for a maximum of 500 files in any one call. A maximum of 10 TRS-FILEINFO structures can be passed to GET-TDS-STAT.
- A maximum of 15 data-name parameters can be passed to GET-TDS-STAT.
- The release level of the TDS can influence what information that the TRS procedure can provide.

The following are examples of what the GET-TDS-STAT call does:

- It can display the current TDS status on a terminal and update it.
- It can determine if any given file is open or closed before a TPR issues I/O operations against it.
- It can determine which users are logged onto the TDS at any given time.
- It can identify all blocked users and those users blocking them. (A user is blocked when a transaction is not available and the user must wait for it.)

Usage

The TRS-CONTROL structure is as follows:

```
01  TRS-CONTROL
    02  TRS-STATUS      COMP-1 VALUE 0.
    02  TRS-SUB-STATUS  COMP-1 VALUE 0.
    02  NB-GBLK         COMP-1 VALUE 1.
    02  NB-UBLK         COMP-1 VALUE 0.
    02  NB-UENT         COMP-1 VALUE 0.
    02  SELECT-USERS    PIC X(12) VALUE SPACES.
    02  MAX-USERS       COMP-1 VALUE 200.
    02  NB-USERS        COMP-1 VALUE 0.
    02  NB-FBLK         COMP-1 VALUE 0.
    02  NB-FENT         COMP-1 VALUE 0.
    02  SELECT-FILES    PIC X(8) VALUE SPACES.
    02  MAX-FILES       COMP-1 VALUE 500.
    02  NB-FILES        COMP-1 VALUE 0.
```

Parameters

TRS-STATUS
This field informs the calling TPR of any errors with the following status values:

0	Successful call to TRS. This is the only status that returns requested TDS information.
1	One or more of these fields contain a negative value: NB-GBLK, NB-UBLK, NB-FBLK, MAX-USERS, or MAX-FILES.
2	The value is not 0 or 1 in NB-GBLK in TRS-CONTROL.
3	Either too many parameters are being passed to TRS, or at least one of these fields exceeds the TRS limits: NB-GBLK, NB-UBLK, or NB-FBLK.
4	Too few parameters are being passed to TRS. The valid number of parameters is one more that the sum of these fields: NB-GBLK, NB-UBLK, and NB-FBLK.
5	Either user information is requested when NB-UENT is less than zero, or the value of the NB-UBLK field multiplied by that of the NB-UENT field is less than the value of MAX-FILES.
6	Either user information is requested when NB-FENT is less than zero, or the value of the NB-FBLK field multiplied by that of the NB-FENT field is less than the value of MAX-FILES.
-2	One of the parameter structures passed to TRS is too small to receive all the requested information. This can be because the "occurs" values for USERINFO, FILEINFO, or both are different than the values of MAX-USERS and MAX-FILES.

TRS-SUB-STATUS	This field indicates which structure contains an error. Valid values for this field are: 1 -The error is in the general control structure. 2 -The error is in the user control structure. 3 -The error is in the file control structure. This field is required when the TRS-STATUS field (described above) returns a value of 1, 3, or -2.
NB-GBLK	This field indicates when a TRS-GENERINFO structure is being passed to TRS. Valid values are: 0 -The TRS-GENERALINFO structure is being passed.

(NB-UBLK, NB-UENT, MAX-USERS, NB-USERS)

These four fields combine to specify what user information TRS should report. Different combinations of these fields produce different reports. If no user information is required, then the NB-UBLK, NB-UENT, and MAX-USERS fields should all be set to zero. The value of the NB-UBLK field multiplied by the value of the NB-UENT field must be greater than or equal to the value of MAX-USERS. All TRS-USERINFO structures must have the same number of entries. The user fields are as follows:

NB-UBLK	This field indicates how many TRS-USERINFO structures are being passed to TRS.
NB-UENT	This field indicates how many entries are in each TRS-USERINFO structure.
MAX-USERS	This field indicates that the calling TPR specifies the maximum number of users for whom detailed information is required. TRS-USERINFO returns the details.
NB-USERS	This field indicates how many users about whom to retrieve detailed information. TRS clears the other user fields.

(NB-FBLK + NB-FENT + MAX-FILES + NB-FILES)

These four fields combine to specify what file information TRS should report. Different combinations of these fields produce different reports. If no file information is required, then the NB-FBLK, NB-FENT, and MAX-FILES fields should all be set to zero. The value of the NB-FBLK field multiplied by the value of the NB-FENT field must be greater than or equal to the value of MAX-FILES. All TRS-FILEINFO structures must have the same number of entries. The file fields are as follows:

NB-FBLK	This field indicates how many TRS-FILEINFO structures are being passed to TRS.
NB-FENT	This field indicates how many entries are in each TRS-FILEINFO structure.

MAX-FILES	This field indicates that the calling TPR specifies the maximum number of files about which detailed information is required. TRS-FILEINFO returns the details.
NB-FILES	This field indicates how many files about which to retrieve detailed information. TRS clears the other file fields
SELECT-USERS, SELECT-FILES	These fields restrict the users or files about which TRS reports. These fields can both use a simplified star convention in that they support only a single trailing star. For example, these fields support both A* and ABC*, but not A*B or *B. These fields can select users and files in three ways. TRS returns information on all users and files if these fields contain spaces, or a single star (*). TRS returns information only on the specified user or file if they contain a full user-id or file name (without the star convention). TRS returns information for the user or files whose names match the supplied star convention. If the value in the MAX-USERS or MAX-FILES field is less than the number of users or files matching the star convention, TRS uses the value in the MAX field.

The following two examples show how to call a TRS.

EXAMPLE

In this example, the TRS-CONTROL structure contains the following parameters:

```
NB-GBLK = 1
NB-UBLK = 2
NB-FBLK = 1
```

With the above parameters, the structure contains a status field that tells the TPR if the call to GET-TDS-STAT was successful. The procedure should be as follows:

```
CALL "GET-TDS-STAT" USING TRS-CONTROL,
                          TRS-GENERALINFO,
                          TRS-USERINFO1,
                          TRS-USERINFO2,
                          TRS-FILEINFO.
```

❑

EXAMPLE

In this example, the TRS-CONTROL structure contains the following parameters:

```
NB-GBLK = 0
NB-UBLK = 0
NB-FBLK = 1
```

With the above parameters, the procedure should be as follows:

```
CALL "GET-TDS-STAT" USING TRS-CONTROL, TRS-FILEINFO.
```

❑

6.5.1 TRS-GENERALINFO

Description

If specified in the TRS-CONTROL structure, this parameter summarizes information about the whole TDS operation. The NB-GBLK field in the TRS-CONTROL structure indicates that this information is required.

Usage

```
01  TRS-GENERALINFO.
    02  TDS-NAME        PIX X(4).
    02  TPR             COMP-2.
    02  TPR-ABORT       COMP-2.
    02  TX              COMP-2.
    02  TX-ABORT        COMP-2.
    02  EXCHANGE        COMP-2.
    02  COMMIT          COMP-2.
    02  CPU-MIN         PIC 9999V999.
    02  CPU-MILSEC      COMP-2.
    02  ELAPSED-MIN     PIC 9999V999.
    02  ELAPSED-MILSEC  COMP-2.
    02  USERS           COMP-1.
    02  FROZEN          COMP-1.
    02  TOTAL FILES     COMP-1.
    02  CUR-MEM-AREAS   COMP-1.
    02  MAX-MEM-AREAS   COMP-1.
    02  BUFOV           COMP-2.
    02  LONGWAIT        COMP-2.
    02  TABOV           COMP-1.
    02  DIRTY-READ      COMP-1.
    02  WDNAV           COMP-1.
    02  DEADLOCK        COMP-1.
    02  SERIAL          COMP-1.
    02  NON-CONC        COMP-1.
    02  TIMELIMIT       COMP-2.
    02  IDLE-TIME       COMP-2.
    02  MAX-TERMS       COMP-1.
    02  CUR-SIMU        COMP-2.
    02  FRZ-SIMU        COMP-1.
    02  COMMON-SIZE     COMP-2.
    02  PRIVATE-SIZE    COMP-2.
    02  MAX-TX-SIZE     COMP-2.
    02  HA-TDS          PIC X.
    02  BROADCAST-SIZE  COMP-1.
    02  BROADCAST-MESS  PIC X(46).
    02  SMLIBS OCCURS D.
    03  SMLIB_NM        PIC X(44).
    02  FILLER          PIC X(100).
```

Parameters

TDS-NAME	the name of the TDS application. This entry is useful because a TPR can run under several TDSs.
TPR	the number of TPRs that have been executed.
TPR-ABORT	the number of TPR aborts that have occurred.
TX	the number of transactions that have been executed.
TX-ABORT	the number of transaction aborts that have occurred.
EXCHANGE	the number of exchanges (message dialogs) that has taken place.
COMMIT	the number of commitments completed.
CPU-MIN	the total CPU time, in minutes, used by all TPRs within the TDS.
CPU-MILSEC	the total CPU time, in milliseconds, used by all TPRs within the TDS.
ELAPSED-MIN	the total elapsed time, in minutes, of all TPRs up to this point.
ELAPSED-MILSEC	the total elapsed time, in milliseconds, of all TPRs up to this point.
USERS	the total number of users, connected or frozen, currently known to TDS.
FROZEN	the number of users out of the total given in "USERS", that are frozen on the application (that is, abnormally disconnected from TDS) but where a context is maintained to allow these users to resume at logon.
TOTAL-FILES	the total number of files opened or closed in the application.
CUR-MEM-AREAS	the current number of TDS memory-areas, manual or automatic.
MAX-MEM-AREAS	the maximum number of TDS memory-areas that may be configured.
BUFOV	the number of aborts caused by buffer-overflows.
LONGWAIT	the number of aborts caused by long waits.
TABOV	the number of aborts caused by lock table-overflows.

DIRTY-READ	the number of aborts caused by IDS "dirty-reads".
WDNAV	the number of aborts cause by deferred writes not being available.
DEADLOCK	the number of TPR aborts due to the return code DEADLOCK received from GAC.
SERIAL	the number of times that the TDS was serialized. Certain master commands (such as [M] CLOSE) cause TDS serializations.
NON-CONC	the number of times any user waited due to a transaction that was non-concurrent with those transactions already running.
TIMELIMIT	the current TPR CPU time-limit, at which the TPR aborts.
IDLE-TIME	the TDS idle-time at which a user can remain at the "command level" without entering anything before automatically logged-off from TDS.
MAX-TERMS	the maximum number of terminals that can be connected to a running TDS.
CUR-SIMU	the current TDS simultaneity level.
FRZ-SIMU	the number of simultaneities not used by the TDS; those suspended by the master command MODIFY_TDS SIMUL=N.
COMMON-SIZE	the size of the non-controlled common-storage area available in TDS.
PRIVATE-SIZE	the size of the private-storage area within transaction-storage. It is the maximum private-storage size when specified.
MAX-TX-SIZE	the size of the largest transaction-storage area. This value is used to calculate the size of the swap file. It is the maximum transaction-storage size when specified.
HA-TDS	"H" specifies that High Availability TDS is in used and is monitored by the COMPLEX MANAGEMENT SERVICE. " " specifies that the application is not HA TDS.

BROADCAST-SIZE	the length of the current broadcast message. If the value of this field is zero, no current broadcast message exists.
BROADCAST-MESS	the broadcast message, as sent by the master command MODIFY_TDS_MOT.
SMLIB_NM	the current sharable module libraries assigned to the TDS in the order in which they are searched.

6.5.2 TRS-USERINFO

Description

If specified in the TRS-CONTROL structure, this parameter provides detailed information about the TDS users. The MAX-USERS field in the TRS-CONTROL structure indicates that this information is required. The SELECT-USERS field in the TRS-CONTROL structure specifies the user selection. The NB-UBLK parameter in the TRS-CONTROL structure indicates the number of USERINFO structures. The default array size and the name of this structure can be modified, as follows:

```
COPY TRS-USERINFO REPLACING 200 BY 400
                  TRAILING "USERINFO" BY "USERINFO2".
```

Usage

```
01  TRS-USERINFO.
    02  USERINFO    OCCURS 200.
        03  USER-TYPE           PIC X(2).
        03  USERID              PIC X(12).
        03  STATE               PIC 9.
        03  SEND-LEVEL          PIC 9.
        03  WAITTIME            PIC X(1).
        03  SERIALIZE           PIC X(1).
        03  BLOCKING            PIC X(1).
        03  MULTI-TPR-COMMIT    PIC X(1).
        03  PASS-THROUGH        PIC X(1).
        03  TERM-TYPE           PIC X(8).
        03  TERM-NAME           PIC X(8).
        03  TX-NAME             PIC X(8).
        03  TX-STORE-SIZE       COMP-2.
        03  TX-NAME-BLOCKED     PIC X(8).
        03  FILLER              PIC X(20).
```

Parameters

USER-TYPE	the type of user as seen by TDS: "space" = normal user "B = batch-interface user "I = user connected via IOF (master operator only) "D" = dummy correspondent "X1" = XCP1 correspondent "X2" = XCP2 correspondent.
USERID	the user identification.

STATE	the current state of the user as follows: 1 = Processing - executing a TPR 2 = End TPR - between TPRs (next TPR has not started) 3 = Command - not running a transaction (READY or IDLE) 4 = Frozen - abnormal disconnection (including $*$DIS) 5 = Blocked - blocked due to non-concurrence 6 = GAC Wait - waiting on GAC after LONGWAIT, DEADLOCK, TABLV or ENQUE abort 7 = Transient - in a state of flux or between states 8 = Unlogged - user unlogged after master command M CANCEL 9 = CMG Wait - waiting commitment unit restart after an abort with WCNAV, BUFNBOV, CMWSOV, ITMNAV or ENQUE. The return code ENQUE received at TPR start if serialization is requested by another commitment unit.
SEND-LEVEL	indicates level of last physical send from TDS to the terminal (EMI or EGI), or more accurately, the type of event that TDS is waiting for (for EMI, an acknowledgement of the sent message, for EGI, the next input from the terminal). Possible values are: 1 = No physical send - no send or only send ESI. 2 = Send EMI and no acknowledgement received 3 = Send EGI and the next terminal input awaited.
WAITTIME	indicates whether the previous TPR set the wait-time in TDS-STORAGE before completing: "space" = wait-time not set "W" = wait-time set (determines start of NEXT-TPR).

SERIALIZE	indicates whether a user executes alone (serialization) or waits for all active commit units to finish before executing alone. Possible values are: "space" = not serializing "S" = serializing or waiting to serialize.
BLOCKING	indicates whether specified user is blocking other users by running a transaction non-concurrent with itself or with other transactions. Possible values are: "space" = not blocking other users "B" = blocking other users via non-concurrence mechanism. Not that the blocked users should have a STATE of BLOCKED.
MULTI-TPR-COMMIT	indicates whether user is executing a multi-TPR commitment unit. That is, a TPR executed a SEND WITH EGI without a commitment. "space" = last TPR ended with a commitment "M" = last TPR ended without a commitment.
PASSTHROUGH	indicates whether user is using TDS Pass-Through: "space" = user is not using TDS Pass-Through "P" = user is using TDS Pass-Through
TERM-TYPE	the type of terminal that the user is connected to.
TERM-NAME	the name of the user's terminal. This value is use as symbolic source or symbolic destination.
TX-NAME	the name of the transaction (if any) that the user is running.
TX-STORE-SIZE	the size of the TRANSACTION-STORAGE associated with the transaction. It includes the space allocated by GETSP-U-CNTXT if any.
TX-NAMED-BLOCKED	the name of the transaction that the user is blocked on if the STATE is BLOCKED.

6.5.3 TRS-FILEINFO

Description

If specified in the TRS-CONTROL structure, this parameter provides detailed information about TDS files. The MAX-FILES field in the TRS-CONTROL structure indicates that this information is required. The SELECT-FILES field in the TRS-CONTROL structure specifies the file selection. The NB-FBLK parameter in the TRS-CONTROL structure indicates the number of FILEINFO structures. The default array size and name of this structure can be modified, as follows:

```
COPY TRS-FILEINFO REPLACING 200 BY 40
                  TRAILING "FILEINFO" BY "FILEINFO2".
```

Usage

```
01   TRS-FILEINFO.
     02  FILEINFO      OCCURS 200.
         03  FILE-TYPE             PIC X(1).
         03  FILE-NAME             PIC X(8).
         03  FILE-STATE            PIC X(1).
         03  AFTER-JOURNAL         PIC X(1).
         03  BEFORE-JOURNAL        PIC X(1).
         03  SHARE-LEVEL           PIC X(1).
         03  DEFERRED-UPDATE       PIC X(1).
         03  SHARE-MODE            PIC X(1).
         03  ACCESS-MODE           PIC X(1).
         03  INIT-PMD              PIC X(2).
         03  CUR-PMD               PIC X(2).
         03  FILE-INTEGRITY        PIC X(1).
         03  EFN                   PIC X(44).
         03  FILLER                PIC X(20).
```

Parameters

FILE-TYPE	the type of the file: "C" = TDS controlled file "N" = TDS non-controlled file "D" = IDS-II database area
FILE-NAME	the internal file name as specified in TDSGEN and used by TPRs.
FILE-STATE	indicates whether the file is open or not: "O" = file is open "C" = file is closed or being closed.
AFTER-JOURNAL	indicates whether the file is protected by the After Journal: "space" = no after Journal on file "A" = file is protected by After Journal.
BEFORE-JOURNAL	indicates whether the file is protected by the Before Journal: "space" = no Before Journal on file "B" = file is protected by Before Journal.
SHARE-LEVEL	shows the current sharability of the file. If the file is closed and re-opened without specifying the option "SHARED" on the M OPEN command, then it is exclusive. "S" = file is opened in shared-mode "E" = file is exclusive to TDS.
DEFERRED-UPDATE	indicates whether Deferred Updates is used for the file: "space" = Deferred Updates is not used "D" = Deferred Updates is used
SHARE	file sharing mode: "N" = normal "M" = monitor "O" = onewrite "D" = directory "F" = free.

ACCESS	file access mode: "R" = read "W" = write "S" = spwrite "P" = spread.
INIT-PMD+CUR-PMD	displays the initial processing-mode for the file, the value specified in TDSGEN and retrieved at the first TDS startup or the last cold restart (whichever was last). If the file is closed and re-opened, the current processing-mode may be different. This current processing-mode is retained for the next session if a TDS warm restart is used. The processing-modes are: "IN" = input "UP" = input-output/update "OU" = output "AP" = extend/append.
FILE-INTEGRITY	the file integrity as specified in the TDSGEN. This is the minimum level of protection that TDS provides for the file: "H" = high "M" = medium "N" = none.
EFN	the external file name.

6.6 The CALL "LIST-COR" Procedure

Syntax

```
CALL "LIST-COR" USING type,
                      option,
                      list-cor,
                      status.
```

Description

Lists the correspondents

- either declared in NETGEN (static list),
- or connected to the TDS application (dynamic list).

The list is not sorted.

The equivalent GCL master command is `[ M ] LIST_TDS_COR...` (described in the *TDS Administrator's Guide*).

Input parameters

type is a 1-character field for selecting the correspondent type. It must contain one of the following values:

1. for Terminal Manager correspondents,
2. for XCP1 correspondents,
3. for XCP2 correspondents,
4. for DUMMY

A static list of correspondents varies according to the type of correspondent requested:

- if the type requested is XCP2, all the XCP2 correspondents that can be reached from the submitting TDS application are listed. This means that if the correspondent is declared as parallel in NETGEN, a pool exists in NETGEN that associates it with the TDS application. If the correspondent is not parallel, it is always listed.
- if the type requested is XCP1, all the XCP1 correspondents whose role is primary are listed.
- if the type requested is TM, all the TM correspondents defined in NETGEN are listed.

option is a 1-character field that specifies the type of characteristics to be displayed and must contain one of the following values:

S	indicates that Static parameters are to be displayed (not allowed if type is set to 4.
D	indicates that Dynamic parameters are to be displayed.

list-cor is a data structure that returns the list of correspondents. You can copy this data structure from the <tdsname>.COBOL file through the following COBOL statement:

```
  "COPY H-DC-TP-LISTCOR REPLACING MAX-COR-NB BY
                                   <user-supplied maximum value>":

05 coc-list-cor.
   06 coc-list-cor-version   PIC X VALUE 1.  input=1
   06 coc-c-index            COMP-1.         index to be returned
                                             upon next call
   06 coc-c-list-max-size    COMP-1.         no. of coc-cor-name
                                             fields in
                                             coc-cor-list
   06 coc-c-actual-list-size COMP-1.         number of
                                             correspondents
                                             in the list
   06 coc-c-total-list-size  COMP-1.         number of
                                             correspondents
                                             (total)
   06 coc-cor-list occurs 1 to max-cor-nb times
                         depending on
                         coc-c-list-max-size.
      07 coc-cor-name         PIC X(12).
      07 coc-cor-state       PIC X(1).      C indicates Connected,
                                            D indicates
                                              Disconnected
                                            U indicates Undefined
```

The following items are **input** parameters:

- coc-list-cor-version corresponds to the version of the data structure and must contain 1.
- coc-c-list-max-size is the number of coc-cor-name field in the coc-cor-list sub-structure. It must be less than or equal to 5000.
- coc-c-index is used when residual list size is greater than the value specified in the coc-c-list-max-size field, that is, when several calls are necessary to get the full list. The coc-c-index field must contain 0 for the first call. If the coc-external-status is set to COC-TRUNC, the content of coc-c-index is modified by TDS and must be supplied upon the next call statement.

Output parameters

In the list-cor structure:

- `coc-c-total-list-size` returns the current number of correspondents only when the option parameter is set to D and type is XCP2. Note that `coc-c-total-list-size` contains the same value after the first call and after the following ones (for the same list); you can use it to adapt the coc-c-list-max-size parameter.
- `coc-c-actual-list-size` is the number of correspondents returned.
- `coc-cor-name` is the local correspondent name.
- `coc-cor-state` returns U (Undefined) only for TM and XCP1 correspondents and a blank when option is static.
- `coc-c-index` is significant when the coc-external-status field in the status structure equals COC-TRUNC. Its value must be supplied as an input parameter upon the next CALL statement in order to get the next correspondents in the list. The list cannot be obtained with several calls when type contains 3 and option contains D.

status refers to the following data structure that you can copy from the <tdsname>.COBOL file by using the COBOL statement `COPY H-DC-TP-STAT`.

```
05 coc-status.
06 coc-status-version     PIC X  VALUE 1.  input=1
06 coc-external-status    PIC X.
06 coc-system-status.
   07 coc-issuer          PIC X.
   07 coc-code            COMP-1.          status code
   07 coc-subcode         COMP-1.          status subcode
   07 coc-last-rc         COMP-2.
```

coc-trunc is returned in coc-external-status if the coc-cor-list structure is not long enough.

When the coc-external-status equals coc-trunc, the next CALL "LIST-POOL" must be performed in the same TPR, before any other procedure; otherwise coc-argerr is returned.

Here is the list of status values that the coc-external-status field can contain.

coc-argerr (2)	The type field is not found. The option field is not found. The coc-list-cor field is not found. coc-status-version does not equal 1. coc-list-cor-version does not equal 1. coc-c-list-max-size is less than 0 or greater than 5000. coc-c-index is not equal to 0 and it is the first time a "CALL LIST-COR" is performed in the current TPR. coc-c-index was changed between the previous and the current call. type was changed between the previous and the current call. option was changed between the previous and the current call. previous call was a CALL "LIST-POOL". type is not 1 (TM), or 2 (XCP1), or 3 (XCP2), or 4 (dummy). type equals 4 (dummy) and option equals S. option is neither S nor D.
coc-arviol (3)	The transaction issuing the CALL "LIST-COR" was not submitted by the master.
coc-busy (4)	NETGEN is being updated. Retry later.
coc-done (0)	The requested information is listed.
coc-notall (7)	The list of correspondents increased while TDS was processing the command. The list may be incomplete (returned only if option equals D).
coc-notdone (9)	PPC incident. TDS incident. NETGEN incident. Contact the Service Center.
coc-ogenunkn (C)	NETGEN was incrementally generated while TDS was processing the command. Retry later.

coc-resnav (D)	Type equals 3 (XCP2) and the XCP2 service is not started, or the TPR executing the call belongs to a transaction that is not allowed to use XCP2 (the TDS generation clause XCP2 SERVICE USED does not exist for this transaction). This is returned only on the first call and if option equals D.
	corresp-name refers to a XCP2 correspondent and the XCP2 service is not started, or the TPR executing the call belongs to a transaction that is not allowed to use XCP2 (the TDS generation clause XCP2 SERVICE USED does not exist for this transaction).
coc-trunc (F)	A new call is necessary to list the remainder of the requested information because c-list-max-size was too small.
coc-wrongpar (H)	Error occurred on the TDS/PPC interface. Contact the Service Center.

6.7 The CALL "LIST-POOL" Procedure

Syntax

```
CALL "LIST-POOL" USING corresp-name,
                       option,
                       list-pool,
                       status.
```

Description

Lists the XCP2 pools:

- either declared in NETGEN (static list),
- or running for a correspondent (dynamic list).

In both cases (static or dynamic), only the pools attached to the TDS application are listed. The list is not sorted.

The equivalent GCL master command is `[ M ] LIST_COR_POOL...` (described in the *TDS Administrator's Guide*).

Input parameters

`corresp-name` is a string of 12 alphanumeric characters for defining the local correspondent name of an XCP2, or XCP1 partner application. If the option is set to 'D', the corresp-name must contain the name of a correspondent connected to the TDS application.

`option` is a 1-character field for indicating which characteristics are to be displayed and must contain one of the following values:

S	indicates that Static parameters are to be displayed (for a XCP2 correspondent only).
D	indicates that Dynamic parameters are to be displayed.

list-pool refers to a structure that returns the list of pools. You can copy this structure from the <tdsname>.COBOL file using the following COBOL statement:

```
"COPY H-DC-TP-LISTPOOL REPLACING MAX-POOL-NB BY
                               <user-supplied maximum value>"
05 coc-list-pool.
   06 coc-list-pool-version  PIC X  VALUE 1.      input=1
   06 coc-p-index            COMP-1.              index to be returned upon
                                                  next call
   06 coc-p-list-max-size    COMP-1.              no. of coc-pool-name
                                                  fields in pool list
   06 coc-p-actual-list-size COMP-1.              no. of pools in the list
   06 coc-p-total-list-size  COMP-1.              no. of pools for the
                                                  correspondent
   06 coc-pool-list occurs 1 to max-pool-nb times depending on
                                                  coc-p-list-max-size.
      07 coc-pool-name       PIC X(8).
      07 coc-pool-state      PIC X(1).            O indicates Open,
                                                  C indicates being Closed
```

The following items are input parameters:

- coc-list-pool-version is the version of the declarative structure. It must contain 1.
- coc-p-list-max-size is the number of the coc-pool-name field in the coc-pool-list sub-structure and must be less than, or equal to 5000.
- coc-p-index is used when residual list size is greater than the value specified in the coc-p-list-max-size field, that is, when several calls are required to get the full list. This field must contain 0 for the first call. If the coc-external-status is set to COC-TRUNC, the content of coc-p-index is modified by TDS and must be supplied upon the next call statement.

Output parameters

In the list-pool structure:

- coc-p-total-list-size returns the current number of pools for the correspondent. Returned only when option equals D. Note that it contains the same value after the first call and after the following ones (relevant to the same list). Use coc-p-total-list-size to adapt the coc-p-list-max-size field.
- coc-p-actual-list-size returns the number of pools in the pool-list.

- `coc-pool-name:`

  For a XCP2 correspondent, `coc-pool-name` is the name of the pool.

  For a XCP1 correspondent, `coc-pool-name` consists of the pool attribute (2 characters) padded with 6 blank characters.

- `coc-pool-state` is meaningful only when the option parameter equals D.
- `coc-p-index` is meaningful only when the `coc-external-status` field in the status structure equals `COC-TRUNC`. Its value must be supplied as an input parameter upon the next CALL statement in order to obtain the next pool(s) in the list. The list of pools cannot be obtained using several calls when `option` is set to D.

status refers to the following data structure that you can copy from the <tdsname>.COBOL file by using the COBOL statement `COPY H-DC-TP-STAT`.

```
05 coc-status.
06 coc-status-version    PIC X  VALUE 1.  input=1
06 coc-external-status   PIC X.
06 coc-system-status.
   07 coc-issuer         PIC X.
   07 coc-code           COMP-1.          status code
   07 coc-subcode        COMP-1.          status subcode
   07 coc-last-rc        COMP-2.
```

`coc-trunc` is returned in `coc-external-status` if the pool-list structure is not long enough.

When the `coc-external-status equals coc-trunc, the next CALL "LIST-POOL" must be performed in the same TPR, before any other procedure; otherwise coc-argerr is returned.`

Here is the list of status values that the coc-external-status field can contain.

coc-argerr (2)	The corresp-name field is not found. The option field is not found. The coc-list-pool field is not found. coc-status-version is not equal to 1. coc-list-pool-version does not equal 1. corresp-name equals DUMMY or is blank. coc-p-list-max-size is lower than 0, or greater than 5000. coc-p-index contains a value other than 0 and it is the first time a "CALL LIST" is performed in the current TPR. coc-p-index was changed between the previous and the current call. option was changed between the previous and the current call. previous call was a "CALL "LIST-COR". corresp-name was changed between the previous and the current call. option is neither "S" nor "D".
coc-arviol (3)	The transaction issuing the CALL "LIST-POOL" was not submitted by the master.
coc-busy (4)	NETGEN is being updated. Retry later.
coc-done (0)	Request is performed and all the information to be listed is provided.
coc-dupname (8)	Corresp-name has a multiple definition (TM, XCP1, XCP2) in NETGEN.
coc-notall (7)	The list of pools increased while TDS was processing the command. The list can be incomplete (returned only if option equals D).
coc-notdone (9)	PPC incident TDS incident NETGEN incident. Contact the Service Center
coc-notop (A)	No pool is open, that is, there are no active sessions towards the given correspondent (returned only if for a XCP1 context and if option equals D).
coc-objunkn (B)	Corresp-name is unknown to the PPC.

coc-ogenunkn (C)	NETGEN was incrementally generated while TDS was processing the command. Retry later.
coc-resnav (D)	Corresp-name refers to an XCP2 correspondent and the XCP2 service is not started, or the TPR executing the call belongs to a transaction that is not allowed to use XCP2 (the TDS generation clause XCP2 SERVICE USED does not exist for this transaction). This is returned only on the first call and if option equals "D".
coc-trunc (F)	A new call is necessary to obtain the remainder of the list because p-list-max-size was too small.
coc-typeerr (G)	corresp-name refers to a wrong correspondent name. The specified correspondent refers to a correspondent connected to the TDS application other than a XCP1 correspondent (returned only if option equals D). corresp-name refers to a correspondent other than a XCP2 correspondent. corresp-name refers to a XCP1 correspondent which is secondary (the command applies to primary only) (returned only if option equals S).
coc-wrongpar (H)	Error occurred on the TDS/PPC interface. Contact the Service Center.

6.8 The CALL "MODIFY-POOL"

Syntax

```
CALL "MODIFY-POOL" USING corresp-name,
                         pool-name,
                         mod-values,
                         status.
```

Description

Modifies characteristics of a session pool, such as the maximum number of sessions. It is assumed that the specified pool is already opened.

When a TDS application is cold re-started, or when the specified pool is closed, any modifications made by the master terminal operator are lost. Otherwise, the modifications are preserved until the next CALL "MODIFY-POOL" request is performed, or the master command `[ M ] MODIFY_COR_POOL` is entered.

XCP2:

After negotiating the parameters to be modified with the partner application, the caller is informed of any modifications made to the session pool.

If you use this statement to decrease the number of active sessions, the surplus sessions are deallocated as soon as the conversations using them are deallocated.

XCP1 and DUMMY:

Same as for the [M] MODIFY_COR_POOL command. Here, a pool name is the value specified in the ATTRIBUTE parameter of this command.

If the relative number of sessions is negative, the surplus sessions are closed as soon as they are freed.

If the relative number of sessions is positive, additional sessions are connected.

The equivalent GCL master command is `[ M ] MODIFY_COR_POOL...` (described in the *TDS Administrator's Guide*).

Input parameters

`corresp-name` is an input parameter of 12 alphanumeric characters for defining the partner application. It must contain *dummy* or the name of an XCP2COR object, or XCP1COR object previously connected by the CALL "OPEN_POOL procedure, or the master command [M] OPEN_COR_POOL ...

pool-name is an input parameter of 8 alphanumeric characters defining the pool.

For a **XCP2** correspondent, this parameter is the pool name (the POOL object in NETGEN).

For a **XCP1** correspondent, the first 2 characters must contain the pool attribute, that is, the pool attribute as defined for the `[ M ] MODIFY_COR_POOL` command. The remaining characters must be filled with blank characters. If all characters are blank, TDS will take as the pool attribute the first 2 characters of the correspondent name.

For a **DUMMY** correspondent, this parameter must be filled with blank characters.

`mod-values` refers to a structure that you can copy from the <tdsname>.COBOL file by using the COBOL statement `COPY H-DC-TP-MODPOOL`:

```
05 coc-mod-pool.
   06 coc-mod-pool-version     PIC X VALUE 1. input=1
   06 coc-mod-pool-x2.
      07 coc-mod-flags-x2.
         08 coc-max-ses-nb-fg  PIC X.     Y -> max no. of sessions
                                               changed
                                          N -> unchanged
         08 coc-win-source-fg  PIC X.     Y -> max no. of winner
                                               sessions for source
                                               changed
                                          N -> unchanged
         08 coc-win-target-fg  PIC X.     Y -> max. no. of winner
                                               sessions for
                                               target changed
                                          N -> unchanged
         08 coc-auto-activ-fg  PIC X.     Y -> no. of auto-active
                                               changed
                                          N -> unchanged
         08 coc-filler         PIC X(12). reserved for future use
      07 coc-mod-val-x2.
         08 coc-max-ses-nb     COMP-1.    absolute no. of sessions
         08 coc-min-win-source COMP-1.    min no. of winner
                                          sessions for source
         08 coc-min-win-target COMP-1.    min no. of winner
                                          sessions for target
         08 coc-win-auto-activ COMP-1.    max no. of winner auto
                                          active sessions
         08 coc-filler         PIC X(14). reserved for future use

   06 coc-mod-pool-dux1.
      07 coc-mod-flags-dux1.
         08 coc-ses-nb-fg      PIC X.     must contain Y
         08 coc-filler         PIC X(15). reserved for future use

      07 coc-mod-val-dux1.
         08 coc-ses-nb         COMP-1.    relative no. of sessions
         08 coc-filler         PIC X(20). reserved for future use
```

XCP2 pool

All the fields in `coc-mod-pool-x2 are input parameters`.

If you request the `[ M ] PREVENT_NEW_TDS_COR X2C=1` command, or execute the CALL "MD-NEWCONNECT" with coc-xcp2-fg=Y, increasing the number of sessions is rejected, but decreasing the number of sessions is accepted (unless the partner name on the receptor site is unknown in which case the request is always rejected).

Note that the `coc-max-ses-nb` is an absolute number.

XCP1 pool or a **DUMMY** correspondent:

All fields in coc-mod-pool-dux1 are input parameters. Note that coc-ses-nb is a relative number.

Output parameters

In the mod-values structure:

- For a `XCP2` pool, the structure `coc-mod-val-x2 contains the negotiated values`.
- For a `XCP1` pool, or a `DUMMY` correspondent, no outputs appear in `coc-mod-val-dux1`.

status refers to the following data structure that you can copy from the <tdsname>.COBOL file by using the COBOL statement `COPY H-DC-TP-STAT`.

```
05 coc-status.
06 coc-status-version    PIC X  VALUE 1.  input=1
06 coc-external-status   PIC X.
06 coc-system-status.
   07 coc-issuer         PIC X.
   07 coc-code            COMP-1.         status code
   07 coc-subcode        COMP-1.          status subcode
   07 coc-last-rc        COMP-2.
```

Here is the list of status values that the coc-external-status field can take.

coc-argerr (2)	The corresp-name field is not reachable. The pool-name is field not reachable. The coc-mod-pool field is not reachable. coc-status-version does not equal 1. coc-mod-pool-version does not equal 1. corresp-name refers to a XCP1 correspondent and the last 6 characters of pool-name are not blanks. corresp-name equals DUMMY and pool-name is not blank. coc-ses-nb-fg is neither Y nor N (XCP1 or DUMMY). coc-max-ses-nb-fg is neither Y nor N. coc-win-source-fg is neither Y nor N. coc-win-target-fg is neither Y nor N. coc-auto-activ-fg is neither Y nor N.
coc-arviol (3)	The transaction issuing the MODIFY-POOL has not been submitted by the master.
coc-busy (4)	A previous request to modify the pool is still being processed (returned only if for a XCP1 context and if ses-nb is positive).
coc-done (0)	Request is performed.
coc-nonew (6)	TDS does not allow any new XCP2 connections to be established (as a result of the CALL "MD-NEWCONNECT" verb, or a [M] PREVENT_NEW_TDS_COR master command) and maxsesnb is greater than the actual maximum XCP2 session count (returned only for a XCP2 context and if coc-max-ses-nb-fg equals Y).

coc-notall (7)	The requested number of active sessions is greater than the "XCP1SESS" parameter attached to the TDS workstation (TDSWKS is declared in NETGEN). Only the remaining unused sessions were activated (returned only for a XCP1 context and if coc-ses-nb is positive).
coc-notdone (9)	A PPC or TDS incident occurred. Contact the Service Center.
coc-objunkn (B)	corresp-name unknown to tds and ppc. – pool-name unknown to PPC (XCP2). – corresp-name was not declared in NETGEN (returned only for a XCP1 context and if coc-ses-nb is positive).
coc-resnav (D)	corresp-name refers to a XCP2 correspondent and the XCP2 service is not started.
coc-syserr (E)	Internal error (returned only if it is a XCP1 context and if coc-ses-nb is positive).
coc-typeerr (G)	corresp-name refers to a wrong correspondent name. The specified correspondent refers to a correspondent connected to the TDS application that is neither XCP1 nor DUMMY. In NETGEN, corresp-name refers to a XCP1 correspondent which is not primary (returned only for a XCP1 context and if coc-ses-nb is positive).
coc-wrongpar (H)	Error occurred on the TDS/PPC interface. Contact the Service Center.

6.9 The CALL "OPEN-POOL" Procedure

Syntax

```
CALL "OPEN-POOL" USING corresp-name,
                       all-option,
                       pool-name,
                       param-name,
                       status.
```

Description

XCP2:

Opens one or all session pools between a local TDS application and a partner application. Before the sessions can be established and allocated to conversations, the CALL "OPEN-POOL" must be issued:

- either by the local site,
- or by the partner site.

The partner correspondent can be defined in NETGEN with PARALLEL=1 or PARALLEL=0. The local name of the partner correspondent and a pool name identifies the pool. The characteristics of the pool(s) must have been declared at NETGEN.

A `CALL "OPEN-POOL"` request is rejected:

- if either a [M] PREVENT_NEW_TDS_COR command,
- or a CALL "MD-NEWCONNECT" request was previously performed.

XCP1:

Same effect as the `[ M ] OPEN_COR_POOL` command. Here, a pool name is the value specified in the `ATTRIBUTE` parameter.

The equivalent GCL master command is [M] OPEN_COR_POOL... (described in the TDS Administrator's Guide).

Input parameters

`corresp-name` is an input parameter of 12 alphanumeric characters for defining the local correspondent name of the partner application. It must contain the name of a XCP2COR, or a XCP1COR object previously declared in NETGEN.

`corresp-name` cannot be "DUMMY", blank, or identical to the name of a terminal operator.

`all-option` is a 1-character input parameter in which you must specify one of the following values:

Y	indicates that this call applies to all pools declared in NETGEN that can link the application to the specified correspondent. TDS tries to open all the pools. (If a fatal error is encountered, only some of the pools are opened).
N	indicates that this call applies to the pool specified in pool-name.

For a XCP1 correspondent, the `all-option` input parameter is ignored.

`pool-name`

For **XCP2**, `pool-name` is a string of 8 alphanumeric characters for defining the pool. It must have been described through the NETGEN.

For **XCP1**, the first 2 characters of `pool-name` must contain the pool attribute. The remaining characters must be filled with blank characters. If all characters are blank, TDS will take as the pool attribute the first 2 characters of the correspondent name.

The `pool-name` input parameter is ignored if `all-option` is set to Y.

For a XCP2 pool, the two sites negotiate the pool name according to the pool attributes defined in their respective NETGEN declaration. If on the acceptor site, the pool name is not known, the pool names proposed of the initiator site are accepted; except for the following 4 attributes (refer to the *GCOS 7-V6 Networks: Overview and Generation* manual).

```
-  winner_auto     set to 0            by the acceptor.
-  drain_source    set to 0            by the acceptor.
-  drain_target    set to 0            by the acceptor.
-  sync_level      set to "confirm"    by the acceptor.
```

param-name refers to a structure that you can copy from the <tdsname>.COBOL file by using the COBOL statement COPY H-DC-TP-OPENPOOL:

```
05 coc-open-pool.
   06 coc-open-pool-version      PIC X VALUE 1. input=1
   06 coc-open-pool-x2.
      07 coc-max-ses-nb          COMP-1.    maximum no. of
                                            sessions
      07 coc-min-win-source      COMP-1.    min no. of winner
                                            sessions for source
      07 coc-min-win-target      COMP-1.    min no. of winner
                                            sessions for target
      07 coc-opened-pool-nb      COMP-1.    total no. of pools
                                            opened
   06 coc-open-pool-x1.
      07 coc-activ-ses-nb        COMP-1.    no. of sessions
                                            activated immediately
```

For a XCP2 pool, only coc-open-pool-version is an input parameter.

For a XCP1 pool, coc-open-pool-version and coc-open-pool-x1 are input parameters. coc-activ-ses-nb cannot be 0.

Output Parameters

In the param-name structure:

For a XCP2 pool

Unless the verb is used for several pools, coc-max-ses-nb, coc-min-win-source, and coc-min-win-target contain the attributes of the pool as negotiated with the correspondent. Otherwise, the TDS monitor does not modify these fields.

coc-opened-pool-nb always contains the number of pools that has been opened successfully. If all-option is set to Y, this number is less than or equal to the total XCP2 pools number defined in NETGEN (if it is less, a value other than COC-DONE is returned in the coc-external-status).

For a XCP1 pool, the structure remains unchanged.

- status refers to the following data structure that you can copy from the <tdsname>.COBOL file by using the COBOL statement `COPY H-DC-TP-STAT`.

```
05 coc-status.
06 coc-status-version     PIC X  VALUE 1.  input=1
06 coc-external-status    PIC X.
06 coc-system-status.
   07 coc-issuer          PIC X.
   07 coc-code             COMP-1.          status code
   07 coc-subcode         COMP-1.          status subcode
   07 coc-last-rc         COMP-2.
```

- Only the `coc-external-status` field can be checked by the caller. The values it can take can be retrieved through the COBOL statement `COPY H-DC-TP-STAT`.
- The `coc-system-status field` is reserved for the use of the Service Center.

The `coc-external-status` field can contain the following values.

coc-already (1)	The specified pool is already opened (returned only in a XCP2 context and if all-option equals N). Some sessions with the same attribute as this specified in the pool-name field are already established (returned only in a XCP1 context).
coc-argerr (2)	The corresp-name field is not found. The all-option field is not found. The pool-name field is not found. The coc-open-pool field is not found. corresp-name equals "DUMMY" or is blank. coc-status-version does not equal 1. coc-open-pool-version does not equal 1. corresp-name refers to a XCP1 correspondent and the last 6 characters of the pool-name field are not blanks. corresp-name refers to a XCP2 correspondent, all-option equals N and pool-name is blank. corresp-name refers to a XCP1 correspondent and coc-activ-ses-nb is negative or zero. all-option is neither Y nor N.
coc-arviol (3)	The transaction issuing the OPEN-POOL was not submitted by the master.
coc-busy (4)	NETGEN is being updated. Retry later.

coc-done (0)	The request is performed.
coc-dupname (8)	corresp-name has a multiple definition (TM, XCP1, XCP2) in NETGEN.
coc-empty (5)	A request to open a pool towards an XCP2 correspondent was made, but no pools are defined in NETGEN for this correspondent.
coc-nonew (6)	A request to open pool(s) towards a XCP2 correspondent was made, but TDS rejects new XCP2 connections. (This is caused by the "MD-NEWCONNECT" verb, or a "[M] PREVENT_NEW_TDS COR" master command).
coc-notall (7)	all-option equals Y and some pools cannot be opened (returned only in a XCP2 context).
	coc-activ-ses-nb is greater than the value defined in the XCP1SESS parameter associated with the TDS workstation (TDSWKS is declared in NETGEN). Only XCP1SESS sessions were activated (returned only in a XCP1 context).
coc-notdone (9)	NETGEN incident. An incident occurred for the ppc (returned only if all-option equals N). All attempts failed (returned only if all-option equals Y). Contact the Service Center.
coc-ogenunkn (C)	NETGEN was incrementally generated while TDS was processing the command. Retry later.
coc-objunkn (B)	corresp-name was not declared in NETGEN. pool-name was not declared in NETGEN.
coc-resnav (D)	corresp-name refers to a XCP2 correspondent and the XCP2 service is not started.
coc-syserr (E)	XCP1: Internal error (returned only in a XCP1 context).
coc-typeerr (G)	The corresp-name in NETGEN is neither XCP1 nor XCP2. corresp-name refers to a XCP1 correspondent that is not primary.
coc-wrongpar(H)	Internal error occurred on the TDS/PPC interface.

7. Terminal Adapter Procedures

7.1 Overview

This chapter deals with the following procedures:

MDPROF	modifies the variables in the profile of a user for whom a transaction is running.
RDPROF	reads the variables from the profile of a user for whom the transaction is running.

7.2 The CALL "MDPROF" Procedure

Syntax

```
CALL "MDPROF" USING profile-description,
                    status-description,
                    [,enclosure-level].
```

Description

Modifies the variables in the profile of a user for whom a transaction is running.

CALL "MDPROF" generates control characters that are sent to the terminal. The effect of some modifications may be postponed depending on the profile variable in question.

No preceding quarantined messages must exist when you use the CALL "MDPROF" statement (and follows the same rules as for the SEND verb).

If the USE TERMINAL ADAPTER clause is not specified at TDSGEN, the CALL "MDPROF" statement is ignored.

If MDPROF is called while the terminal is in formatted mode, the following remarks apply:

- The INVCHAR field is taken into account immediately by FORMS.
- The CSET field is taken into account only on output (because the terminal will receive the message indicating the modification only after exit from formatted mode).
- All the other fields are taken into account only after exit from formatted mode (in fact, after execution of the FORMS verb CDRELS).

If MDPROF is called while the terminal is in line mode and the terminal is subsequently in formatted mode, the following remarks apply:

- The AUTOLF, EXPTABS, MAIL, PL, PW, ROLL, TA, and TABS fields are not relevant to FORMS. However, their values effective again after exit from formatted mode.
- The INVCHAR field is taken into account by FORMS.
- The CSET field is taken into account by FORMS.

Usage

`profile-description` is a data structure containing the set of variables of a user profile. Use the COBOL statement COPY PROFILE-DESC to obtain the following data structure.

```
02 PROFILE-DESCRIPTION.
   03 AUTOLF-D      PIC 9.          takes effect from next SEND
   03 CSET-D        PIC 9.          takes effect from next SEND
   03 EXPTABS-D     PIC 9.          takes effect from next
                                          RECEIVE
   03 INVCHAR-D     PIC X.          takes effect from next SEND
   03 MAIL-D        PIC 9.          takes effect immediately
   03 PL-D          PIC 999.        takes effect from next SEND
   03 PW-D          PIC 999.        takes effect from next SEND
   03 ROLL-D        PIC 9.          takes effect from next SEND
   03 TA-D          PIC 9.          takes effect after the next
                                    terminal I/O
   03 TABS-ARRAY-D. PIC 9.          takes effect after the next
      04 TABS-D  PIC 999 OCCURS 16. terminal I/O
```

All the fields in the PROFILE-DESCRIPTION structure are input parameters and are described as follows.

AUTOLF	AUtO Line Feed can be either 0 or 1. If AUTOLF is 1, pressing the return or the transmit key automatically generates a line feed. If AUTOLF is 0, the line feed is generated by the system. The default value is that specified in the network generation.
CSET	Character SET the character set to be used for coding terminal characters. The value can be: 0 for code C101 (EBCDIC), 1 for code C127 (extended EBCDIC). The default value is 1. 3 for code C114 (Arab characters). 4 for code C118 (Greek characters). 5 for code C113 (Cyrillic characters). 6 for code C094 (Chinese characters). 7 for code PLE (PLURI-LINGUAL EAST). For further details on the character set, see Chapter 2.

EXPTABS	EXpand TABulationS. When EXPTABS is 1, all tabulation characters keyed in are replaced by a number of spaces, as defined by the TABS variable. When EXPTABS is 0, the tabulation characters are transmitted as they are. The default value is 1.
INVCHAR	INValid CHARacter representation (PIC X). Represents characters that cannot be displayed on the user's terminal. If value is "00"X (zero), the characters are left unchanged. The default value is "00"X.
MAIL	refers to the reception of messages. The value is either 1, or 0. The default value (MAIL = 1) allows external messages to be displayed. If MAIL = 0, such messages do not appear and the sender receives the return code MSG-REFUSED if the SEND verb is used in a TPR.
PL	indicates the Page Length and can contain up to 3 numeric characters (PIC 999). Specifies how many lines can be entered on a screen. The default value is terminal dependent. PL must be greater than 10, and less than, or equal to the terminal's actual page length, as specified in the network generation.
PW	Printing Width (PIC 999). The maximum number of characters per line. The default value is terminal dependent. PW must be greater than 35 and less than or equal to the physical width of terminal, as specified in the network generation.
ROLL	ROLL mode can be 0 or 1. If ROLL is 1, the terminal operates in rollmode. Each new line at the bottom of the screen pushes up the remainder of the screen by one line, thus erasing the topmost line. The default value is that specified in the network generation.

TA	`TA=0` puts the user out of terminal adapter presentation. This is the default value. `TA=1` puts the user into terminal adapter presentation. Messages can be sent to or received by the user regardless of terminal type.
TABS	TABulation Stops. Redefines the tab stop positions on the terminal. Tab stops are expressed as column numbers starting with 1 for the leftmost character position. The tab stops are sorted in ascending order. type PIC 999 OCCURS 16. The default value is 0. The values must be positive and less than or equal to the physical width of the terminal. You cannot modify tabulation stops for a user working in format mode.

`Status-description` is the data structure containing the status key that informs you if an error occurs for a profile variable.

You can obtain this data structure by using the COBOL statement COPY PROFILE-STATUS.

```
01 PROFILE-STATUS.
   02 STATUS-S       PIC 9.
   02 AUTOLF-S       PIC 9.
   02 CSET-S         PIC 9.
   02 EXPDTABS-S     PIC 9.
   02 INVCHAR-S      PIC 9.
   02 MAIL-S         PIC 9.
   02 PL-S           PIC 9.
   02 PW-S           PIC 9.
   02 ROLL-S         PIC 9.
   02 TA-S           PIC 9.
   02 TABS-S         PIC 9.
```

All the fields in PROFILE-STATUS are output parameters.

`PROFILE-STATUS` takes the following values at the completion of statement:

0	successful completion
1	at least one warning and no error occurred
2	at least one error occurred
3	invalid parameter structure
4	ignored statement
5	misplaced statement for example, a quarantined message already exists, or there is a turn error.

The status items in the PROFILE-STATUS structure can take the following values at the completion of the statement:

0	modification request is taken into account.
1	warning: this user cannot use this value for this item.
2	error: the value is out of range for this item.

An erroneous status item does not prevent a valid modification from taking effect for another status item.

Some of the modifications must be made known to the terminal. Consequently, the `enclosure-level`, which specifies how a message terminates, must be accurately chosen to save these modifications.

`enclosure-level` takes the following values:

1 The message is quarantined, but is not sent to the terminal. This is the default value. This is equivalent to ESI in TDS terms.

2 All quarantined messages are now sent to the terminal. The application still retains the turn. This is equivalent to EMI in TDS terms.

3 All quarantined messages are now sent to the terminal. The turn is given to the terminal. This is equivalent to EGI in TDS terms.

7.3 The CALL "RDPROF" Procedure

Syntax

```
CALL "RDPROF" USING profile-description, status.
```

Description

Reads the variables from the profile of a user for whom the transaction is running.

If the `USE TERMINAL ADAPTER` clause is not specified at TDSGEN, this statement is ignored.

Usage

`profile-description` is a data structure containing the set of variables of a user profile. You can obtain this data structure by using the COBOL statement `COPY PROFILE-DESC`.

```
02 PROFILE-DESCRIPTION.
   03 AUTOLF-D          PIC 9.
   03 CSET-D            PIC 9.
   03 EXPTABS-D         PIC 9.
   03 INVCHAR-D         PIC X.
   03 MAIL-D            PIC 9.
   03 PL-D              PIC 999.
   03 PW-D              PIC 999.
   03 ROLL-D            PIC 9.
   03 TA-D              PIC 9.
   03 TABS-ARRAY-D.
      04 TABS-D         PIC 999 OCCURS 16.
```

All the fields in PROFILE-DESCRIPTION are output parameters. For an explanation of these parameters, see the `CALL "MDPROF"` statement earlier in this chapter.

`status` defines the status of the statement CALL "RDPROF" and is an output parameter. Status takes the following values at the completion of the statement:

0 Successful completion of the statement. Current values of the profile are stored in the structure.
1 Abnormal completion. The parameter structure is not valid.
2 Ignored statement.

❑

8. COMMON-STORAGE Handling Procedures

8.1 Overview

This chapter deals with the following procedures:

CLENGTH-COMMON	returns the actual length in bytes of the specified CONTROLLED COMMON-STORAGE defined at TDSGEN.
CREAD-COMMON	reads the contents of the specified CONTROLLED COMMON-STORAGE from the tdsname.CTLM file and moves it into a specified area.
CWRITE-COMMON	updates the contents of the specified CONTROLLED COMMON-STORAGE that is stored on the tdsname.CTLM file (on disk) with the contents of a user area declared in WORKING-STORAGE.
FREE-COMMON	moves the contents of a specified data area to COMMON-STORAGE and unlocks COMMON-STORAGE.
KEEP-COMMON	moves the contents of a specified data area to the COMMON-STORAGE, and saves, that is, updates the COMMON-STORAGE on a TDS system file, named tdsname.CTLN.
LENGTH-COMMON	returns the length of COMMON-STORAGE as defined at TDSGEN.
READ-COMMON	moves the contents of the COMMON-STORAGE area to a specified data area.
SAVE-COMMON	moves the contents of a specified data area to the COMMON-STORAGE, writes the COMMON-STORAGE onto the TDS system file, named tdsname.CTLN, and unlocks COMMON-STORAGE.
TAKE-COMMON	moves the contents of COMMON-STORAGE to a specified data area and locks COMMON-STORAGE.

8.2 The CALL "CLENGTH-COMMON" Procedure

Syntax

```
CALL "CLENGTH-COMMON" USING data-name-1,
                            data-name-2,
                            data-name-3.
```

Description

Returns the actual length in bytes of the specified CONTROLLED COMMON-STORAGE defined at TDSGEN.

Usage

`data-name-1` is a field of 12 alphanumeric characters identifying CONTROLLED COMMON-STORAGE. It is an input parameter.

`data-name-2` is a COMP-2 field that contains the length (in bytes) of the CONTROLLED COMMON-STORAGE. Data-name-2 is an output parameter.

`data-name-3` is a single numeric character defined as `PIC 9` that contains the status of the CALL. It is an output parameter. The possible status values are:

0 = successful completion.

1 = unknown CONTROLLED COMMON-STORAGE name.

8.3 The CALL "CREAD-COMMON" Procedure

Syntax

```
 CALL "CREAD-COMMON" USING data-name-1,
                           data-name-2,
                           data-name-3,
                           data-name-4.
```

Description

Reads the contents of the specified CONTROLLED COMMON-STORAGE from the tdsname.CTLM file and moves it into an area named data-name-2. The CONTROLLED COMMON-STORAGE is locked until the next commitment.

Usage

`data-name-1` is a field of 12 alphanumeric characters identifying CONTROLLED COMMON-STORAGE defined at TDSGEN. It is an input parameter.

`data-name-2` names an area declared in the WORKING-STORAGE SECTION of the TPR. Data-name-2 is an output parameter.

`data-name-3` is a COMP-2 field that must contain the length of data to be read. Data-name3 is an input parameter.

If the specified length is lower than the real length of the CONTROLLED COMMON-STORAGE, only the specified length is read from the beginning of the CONTROLLED COMMON-STORAGE.

The value of data-name-3 must not be negative, zero, or greater than the size of the CONTROLLED COMMON-STORAGE as given at TDSGEN. If one of these conditions occurs, the size given at TDSGEN is retained and no error status is returned. However, the real size used to perform the move to the user area (data-name-2) is adjusted to the end of the user segment in order to avoid the exception (out of bounds). Note that the value of data-name3 is not modified.

`data-name-4` is a single numeric character defined as PIC 9. This output parameter contains the status of the CALL. At completion, this field contains one of the following values:

0 = successful completion
1 = unknown CONTROLLED COMMON-STORAGE name.

8.4 The CALL "CWRITE-COMMON" Procedure

Syntax

```
CALL "CWRITE-COMMON" USING data-name-1,
                           data-name-2,
                           data-name-3,
                           data-name-4.
```

Description

Updates the contents of the specified CONTROLLED COMMON-STORAGE that is stored on the tdsname.CTLM file (on disk) with the contents of the user area (data-name-2) declared in WORKING-STORAGE.

The specified CONTROLLED COMMON-STORAGE is locked until the next commitment point.

Usage

`data-name-1` is a field of 12 alphanumeric characters identifying the CONTROLLED COMMON-STORAGE. It is an input parameter.

`data-name-2` names an area declared in the WORKING-STORAGE of the TPR. Data-name-2 is an input parameter.

`data-name-3` is a COMP-2 field containing the length of data to be updated. It is an input parameter.

If the specified length is lower than the real length of the CONTROLLED COMMON-STORAGE, only the specified length starting at the beginning of the CONTROLLED COMMON-STORAGE is written onto the secondary storage.

The value must not be negative, zero, or greater than the size of the CONTROLLED COMMON-STORAGE as given at TDSGEN. If one of these conditions occurs, the size given at TDSGEN is retained and no error status is returned. However, the size used to perform the move from the user area (data-name-2) is adjusted to the end of the user segment in order to avoid the exception (out of bounds).

`data-name-4` is a single numeric character (defined as PIC 9) containing the output status of the CALL as follows:

0 = successful completion
1 = unknown CONTROLLED COMMON-STORAGE name.

8.5 The CALL "FREE-COMMON" Procedure

Syntax

```
CALL "FREE-COMMON" USING data-name-1 [,data-name-2].
```

Description

Moves the contents of an area named data-name-1 to COMMON-STORAGE and unlocks COMMON-STORAGE. See Figure 2-1.

TAKE-COMMON should have been called beforehand.

Usage

`data-name-1` is the name of an area declared in the WORKING-STORAGE of the TPR that contains the data to be moved. It is an input parameter.

`data-name-2` indicates the length of the data to be moved. It is a COMP-2 field specifying the number of bytes from the start of the data-name-1 area to be moved into COMMON-STORAGE. It is an input parameter.

If data-name-2 is not specified, the length used for the move is the length of COMMON-STORAGE as declared at TDSGEN.

If the length given as the parameter has a wrong value (either negative, or zero, or greater than the size of the COMMON-STORAGE), then the real size will be retained.

However, the size used to perform the move from the data-name-1 area will be adjusted to the end of the user segment (which contains data-name-1) in order to avoid the exception (out of bounds).

8.6 The CALL "KEEP-COMMON" Procedure

Syntax

```
CALL "KEEP-COMMON" USING data-name-1 [,data-name-2].
```

Description

Moves the contents of an area named data-name-1 to the COMMON-STORAGE, and saves, that is, updates the COMMON-STORAGE on a TDS system file, named tdsname.CTLN. The CALL "KEEP-COMMON" statement does not unlock the COMMON-STORAGE. See Figure 2-1.

TAKE-COMMON must precede KEEP-COMMON to lock COMMON-STORAGE. If the TPR does not unlock COMMON-STORAGE by calling FREE-COMMON before terminating, the transaction will be aborted with RESVIOL.

Usage

`data-name-1` is the name of an area declared in the WORKING-STORAGE of the TPR. data-name-1 contains the data to be moved into COMMON-STORAGE. It is an input parameter.

`data-name-2` indicates the length of the data to be moved. It is a COMP-2 field specifying the number of bytes from the start of the data-name-1 area to be moved into COMMON-STORAGE. It is an input parameter.

If data-name-2 is not specified, the length used for the move is the length of COMMON-STORAGE as declared at TDSGEN.

If the length given as the parameter has a wrong value (either negative, or zero, or greater than the real size of the COMMON-STORAGE), then the real size will be retained.

However, the size used to perform the move from the data-name-1 area will be adjusted to the end of the user segment (which contains data-name-1) in order to avoid the exception (out of bounds).

8.7 The CALL "LENGTH-COMMON" Procedure

Syntax

```
CALL "LENGTH-COMMON" USING data-name-1.
```

Description

Returns the length of COMMON-STORAGE as defined at TDSGEN.

Usage

`data-name-1` indicates the length of COMMON-STORAGE. It is a COMP-2 output parameter that contains the number of bytes of COMMON-STORAGE as defined at TDSGEN.

8.8 The CALL "READ-COMMON" Procedure

Syntax

```
CALL "READ-COMMON" USING data-name-1 [,data-name-2 ].
```

Description

Moves the contents of the COMMON-STORAGE area to an area named `data-name-1`. See Figure 2-1.

Usage

`data-name-1` is the name of an area declared in the WORKING-STORAGE of the TPR. data-name-1 contains the data moved from COMMON-STORAGE. It is an output parameter.

`data-name-2` indicates the length of the data to be moved. It is a COMP-2 field specifying the number of bytes to be moved from the start of COMMON-STORAGE. It is an input parameter.

If `data-name-2` is not specified, the length used for the move is the length of COMMON-STORAGE as declared at TDSGEN.

If the length given as the parameter has a wrong value (either negative, or zero, or greater than the real size of the COMMON-STORAGE), then the real size will be retained.

However, the size used to perform the move from the COMMON-STORAGE area will be adjusted to the end of the user segment (which contains data-name-1) in order to avoid the exception (out of bounds).

8.9 The CALL "SAVE-COMMON" Procedure

Syntax

```
CALL "SAVE-COMMON" USING data-name-1 [,data-name-2].
```

Description

- moves the contents of an area named data-name-1 to the COMMON-STORAGE,
- saves, that is, writes the COMMON-STORAGE onto the TDS system file, named tdsname.CTLN,
- unlocks COMMON-STORAGE.

COMMON-STORAGE should have previously been locked by TAKE-COMMON. See Figure 2-1.

After a TDS or system failure, TDS at warm restart initializes COMMON-STORAGE with the last "saved" contents from the TDS file.

Usage

`data-name-1` is the name of an area declared in the WORKING-STORAGE of the TPR, that contains the data to be moved into COMMON-STORAGE. It is an input parameter.

`data-name-2` indicates the length of the data to be moved. It is a COMP-2 field specifying the number of bytes from the start of the data-name-1 area to be moved into COMMON-STORAGE. It is an input parameter.

If data-name-2 is not specified, the length used for the move is the length of COMMON-STORAGE as declared at TDSGEN.

If the length given as the parameter has a wrong value (either negative, or zero, or greater than the real size of the COMMON-STORAGE), then the real size will be retained.

However, the size used to perform the move from the data-name-1 area will be adjusted to the end of the user segment (which contains data-name-1) in order to avoid the exception (out of bounds).

8.10 The CALL "TAKE-COMMON" Procedure

Syntax

```
CALL "TAKE-COMMON" USING data-name-1 [,data-name-2].
```

Description

Moves the contents of COMMON-STORAGE to an area named data-name-1 and locks COMMON-STORAGE. See Figure 2-1.

If the calling TPR does not release COMMON-STORAGE before terminating, the transaction will be aborted with RESVIOL.

Usage

`data-name-1` is the name of an area declared in the WORKING-STORAGE of the TPR. data-name-1 contains the data moved from COMMON-STORAGE. It is an output parameter.

`data-name-2` indicates the length of the data to be moved. It is a COMP-2 field specifying the number of bytes to be moved from the start of COMMON-STORAGE. It is an input parameter.

If data-name-2 is not specified, the length used for the move is the length of COMMON-STORAGE as declared at TDSGEN.

If the length given as the parameter has a wrong value (either negative, or zero, or greater than the real size of the COMMON-STORAGE), then the real size will be retained.

However, the size used to perform the move to the data-name-1 area will be adjusted to the end of the user segment (which contains data-name-1) in order to avoid the exception (out of bounds).

9. File Access Concurrency and Commitment Procedures

9.1 Overview

This chapter deals with the following procedures:

CMIT-U-CNTXT	commits user storage either immediately or after specified events.
DFCMIT	requests a commitment to be taken at the end of the TPR.
INVCMIT	invalidates the current commitment unit, but retains the context of the TDS-STORAGE.
KEEP-CURRENCIES	keeps the currencies of a TPR, that is, the current points, active after commitment. The next commitment unit does not need to reposition the currencies of the file.
LOCK	prevents concurrent access to private or shared resources. Examples of a resource are shared storage or program coding.
NOCMIT	cancels the commitment action at the end of the current TPR, unless the TPR is the last TPR of the transaction.
RESET-NON-CONCURRENT	disables the non-concurrency mechanism for the next commitment unit. The transaction must be declared MANUALLY NON-CONCURRENT or NON-CONCURRENT at TDSGEN time.
ROLL-BACK	rolls back the transaction to the first TPR of the current commitment unit.

SET-NON-CONCURRENT	activates the non-concurrency mechanism for the next commitment unit, whether the transaction is declared MANUALLY NON-CONCURRENT or NON-CONCURRENT at TDSGEN time.
UNLOCK	unlocks a resource that has been previously locked. By using the unlock facility, you can free a resource before the next commitment point.

This chapter deals with the following verbs:

CLOSE	allows a TPR to close any file. The file may be a TDS-controlled file or a TDS non-controlled file declared or not declared at TDSGEN.
OPEN	allows a TPR to open a file. This file may be one of the following: a TDS-controlled file (such a file may be opened only in INPUT or UPDATE mode), a file that is not controlled by TDS but which is declared in TDSGEN, a non-declared file (only under COBOL 85 which allows files to be generated dynamically).

Note that the descriptions are in the order shown in this list, and not in alphabetic sequence.

9.2 The CALL "CMIT-U-CNTXT" Procedure

Syntax

```
CALL "CMIT-U-CNTXT" USING data-name-1,
                          data-name-2,
                          data-name-3,
                          data-name-4.
```

Description

Commits user storage either immediately or at specified events. With this call, you can commit storage at a specific time, rather than only at the commit point of a TPR that is terminating normally. The storage can be PRIVATE storage, part of the TRANSACTION storage, or both.

When this call commits immediately, if the TPR aborts later, the current contents of the defined areas are made available to the session, the transaction, or both. This is instead of the contents found at the last commit point. Immediate commitment occurs no matter how the TPR terminates, including user error, user requests, GAC conflicts, and breaks.

When this call commits at specified events, the request for committing storage is stored, and then later executed when the specified event occurs. (It is executed first, before any other action.) The specified events are ROLLBACK and BACKOUT. If more than one call to the procedure is made during a commit, this procedure takes into account only the last one. If the commit terminates before the specified events occur, the request is abandoned.

For example, the committed area could be the TRANSACTION storage and the specified event could be ROLLBACK. In this case, if a GAC conflict causes a commit-rollback, the TRANSACTION storage is saved as at the event occurrence time. It is then available to the transaction at the restart of the current commit.

NOTE:

Do not use the CMIT-U-CNTXT procedure in conjunction with the GETSP-U-CNTXT procedure.

Usage

Data-name-1 is a four-character input field that defines when the commit action occurs. Valid values for this field and their meanings are as follows:

I	The action is immediate.
B	The action occurs at a BACKOUT event. This causes file ROLLBACK, but with a continuation of the commit. This event is possible when an XCP2 environment uses the verb "BACKOUT without conversation restore". (The "USE XCP2" clause declares this transaction.)
R	The action occurs at a ROLLBACK event. This is an event that either rolls back and then restarts the commit, or aborts the transaction. This event includes call ROLLBACK, break, disconnection, GAC conflict, and system crash.
BR	The action occurs at both BACKOUT and ROLLBACK.

Other values are user specification errors.

`Data-name-2` is a four-character input field that specifies the areas to commit. Valid values for this field and their meanings are as follows:

P	The area is the entire PRIVATE storage.
T	The area is the TRANSACTION storage.
PT	The area is both PRIVATE and TRANSACTION storages.

Other values are user specification errors.

`Data-name-3` is an input comp-2 field that specifies the amount (the length) of the transaction storage to commit. The committed amount starts at displacement 0 and ends at a displacement of the value in this field less 1 (included).

The value in `data-name-3` is significant only when committing TRANSACTION storage, in which case it must always be greater than 0. When the value is greater than the TRANSACTION storage, TDS takes the entire TRANSACTION storage. A negative error is a user specification error.

`Data-name-4` is a one-character output field. Values for this field are as follows:

0	The procedure executed successfully.
1	User error

Any other value indicates a system error.

9.3 The CALL "DFCMIT" Procedure

Syntax

```
CALL "DFCMIT".
```

Description

Requests a commitment to be taken at the end of the TPR.

Usage

The commitment may be requested anywhere in the TPR; however, the commitment becomes effective only when the TPR terminates normally.

NOTES:

1. The commitment clears all IDS/II database currencies, cancels the current record pointer in non-database files, and releases any resource attached to the TPR such as non-concurrency or locked CIs.
2. All IDS/II database and file modifications, performed before the commitment, become fixed at the commitment point. Thus any future rollback decided after this commitment point will not affect the updates made before the commitment point.
3. For more information about commitment points, see the section on KEEP-CURRENCIES later in this chapter, and Chapter 1.
4. CALL "DFCMIT" cancels any CALL "NOCMIT" called beforehand, in the TPR.

9.4 The CALL "INVCMIT" Procedure

Syntax

```
CALL "INVCMIT".
```

Description

Invalidates the current commitment unit, but retains the context of the TDS-STORAGE.

Usage

When the commitment point is taken either by a CALL "DFCMIT" statement, or at the end of the TPR for an implicit commitment, the following occurs:

- any modifications which have been made to all files (IDS/II and UFAS-EXTENDED) during the current commitment unit are invalidated,
- the CALL "KEEP-CURRENCIES" statement is invalidated,
- any quarantined messages, portions and message segments are kept,
- the RESTART-STATUS field of TDS-STORAGE is set to 2,
- the TPR indicated in the NEXT-TPR field of the TDS-STORAGE is activated,
- the TRANSACTION-STORAGE and PRIVATE-STORAGE are set to the values they held before the commitment unit which performed the CALL "INVCMIT" statement.

NOTES:

1. The CALL "INVCMIT" statement takes effect only at commitment time, which is determined by the IMPLICIT COMMITMENT clause and the CALL "DFCMIT"/ CALL "NOCMIT" statements.
2. The CALL "INVCMIT" statement produces the same result as a transaction declared FOR DEBUG at TDSGEN for the current commitment unit except for TRANSACTION-STORAGE and PRIVATE-STORAGE.
3. Any transactions spawned during the current commitment unit are invalidated.

9.5 The CALL "KEEP-CURRENCIES" Procedure

Syntax

```
CALL "KEEP-CURRENCIES" USING data-name-1, data-name-2.
```

Description

Keeps the currencies of a TPR, that is, the current points, active after commitment. The next commitment unit does not need to reposition the currencies of the file.

Usage

`data-name-1` specifies the file whose currencies are to be kept.
`Data-name-1` is a PIC X (8) string containing up to 8 alphanumeric characters right padded with blanks.

`data-name-2` is a single numeric character string (PIC X) in which the status is returned.

0 = successful execution

1 = the statement was already issued for the file

2 = `data-name-1` unknown

3 = `data-name-1` file is not open.

4 = KEEP-CURRENCIES forbidden (inquiry transaction or IDS/II).

NOTE:

If the file whose currencies are to be kept, is closed after the end of the current commitment unit, the next commitment unit will not be allowed to start, and the transaction will be aborted with the IFNERR return code. Please refer to Appendix B (of this manual).

9.6 The CALL "LOCK" Procedure

Syntax

```
CALL "LOCK" USING lock-description, data-name-1.
```

Description

Prevents concurrent access to private or shared resources. Examples of a resource are shared storage or program coding.

The application is entirely in charge of identifying resources.

Usage

Lock-description is a data structure that must have the following format:

```
01 LOCK-DESCRIPTION.
   02   RESOURCE-IDENTIFIER  COMP-1.
   02   RESOURCE-TYPE        PIC X.
   02   LOCK-MODE            PIC X.
```

- RESOURCE-IDENTIFIER identifies the resource to be locked. Its value ranges from 0 to 16383 and is agreed upon by all users accessing the same resource. For example, file A could be called 24.

  RESOURCE-IDENTIFIER is an input parameter.
- RESOURCE-TYPE defines the scope of the resource identifier. It must take one of the following two values:

L as local	the resource is private to the TDS application. No other application can access this resource. Two local resources identified by the same resource identifier in different TDS applications are 2 different objects.
G as global	he resource is common to all the applications. Because all applications can access the same resource, locking mechanisms control access to the resource by all TDS applications. Note that if 2 resources have the same resource-identifier value, one as local and the other as global, these 2 resources are different.

RESOURCE-TYPE is an input parameter.

LOCK-MODE defines how the TPR is to access the resource as follows:

E (exclusive)	the resource is requested in exclusive mode by the TPR.
S (shared)	the resource is requested in shared mode by the TPR.

If a resource is requested in exclusive mode by a TPR while locked in shared mode by another TPR, the exclusive request is set to wait until the tenant unlocks the resource.

LOCK-MODE is an input parameter.

`data-name-1` is an output parameter. It is a single numeric character in which the status is returned:

0 = successful execution

1 = at least one wrong argument was detected by TDS

2 = wrong argument detected by the concurrency mechanism.

The GAC-EXTENDED concurrency mechanism handles resources as follows:

- in the case of conflict, deadlock or longwait, the waiting TPR is aborted, rolled back to the last commitment and restarted when the resource becomes available.
- all locks are released at the end of a commitment even if UNLOCK is not called.

In the following examples, the abbreviations are as follows:

R-I	Resource Identifier
R-T	Resource-Type
L-M	Lock Mode
CU	Commitment Unit
L	Local (Resource-Type)
E	Exclusive Lock Mode
S	Shared Lock Mode

For an explanation of CALL "UNLOCK", see later in this chapter.

EXAMPLE 1: Local Resource

In TDS Application A, 3 represents the resource to be locked, that is, file A

In TDS Application B, 3 represents the resource to be locked, that is, file B.

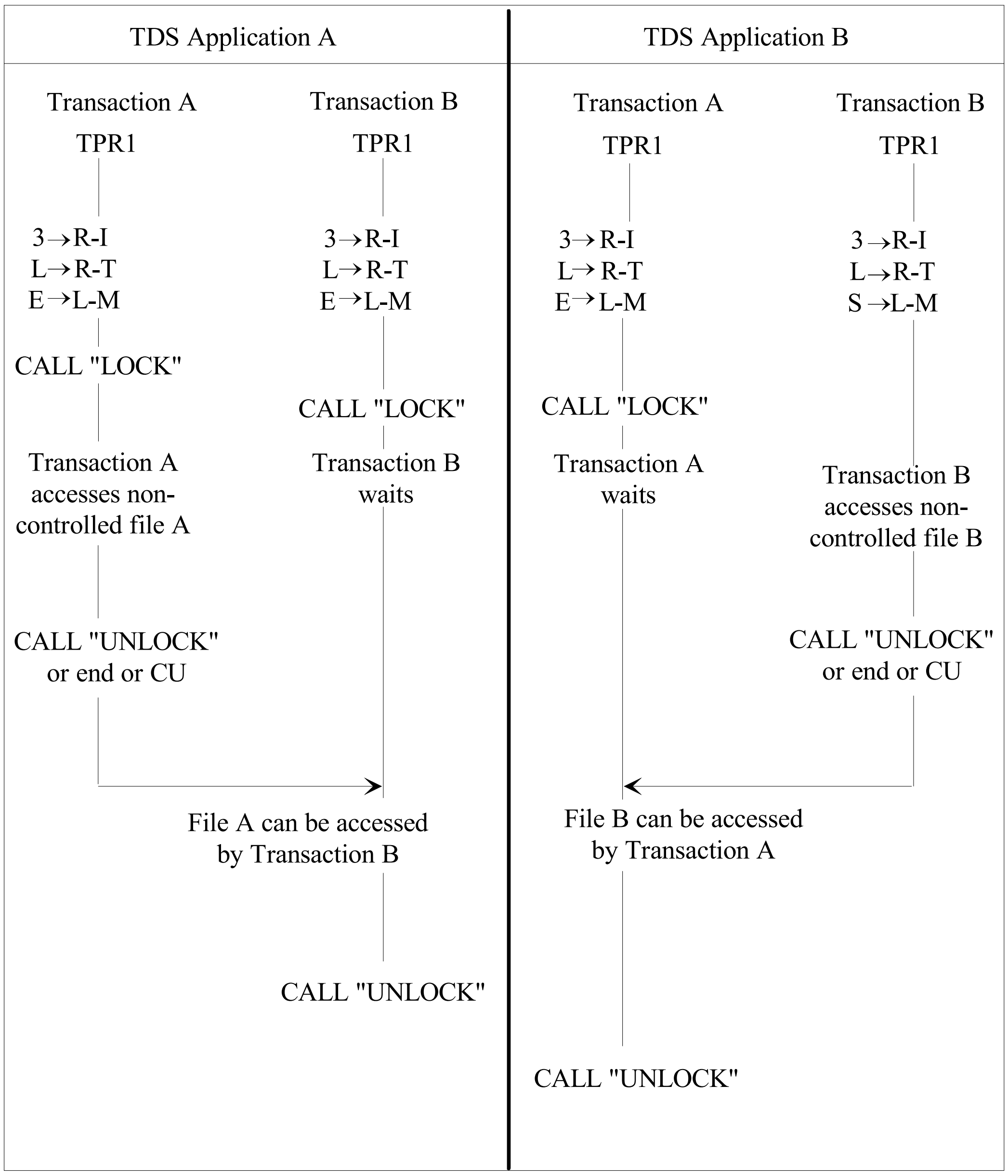

TDS Application A
TDS Application B
Transaction A
Transaction B
TPR1
3→R-I
L→R-T
E→L-M
S→L-M
CALL "LOCK"
Transaction A accesses non-controlled file A
Transaction B waits
Transaction A waits
Transaction B accesses non-controlled file B
CALL "UNLOCK" or end or CU
File A can be accessed by Transaction B
File B can be accessed by Transaction A
CALL "UNLOCK"

Figure 9-1. Using CALL LOCK for a Local Resource

EXAMPLE 2: Global Resource

In TDS application A and TDS application B, 3 represents the resource to be locked, that is, file C

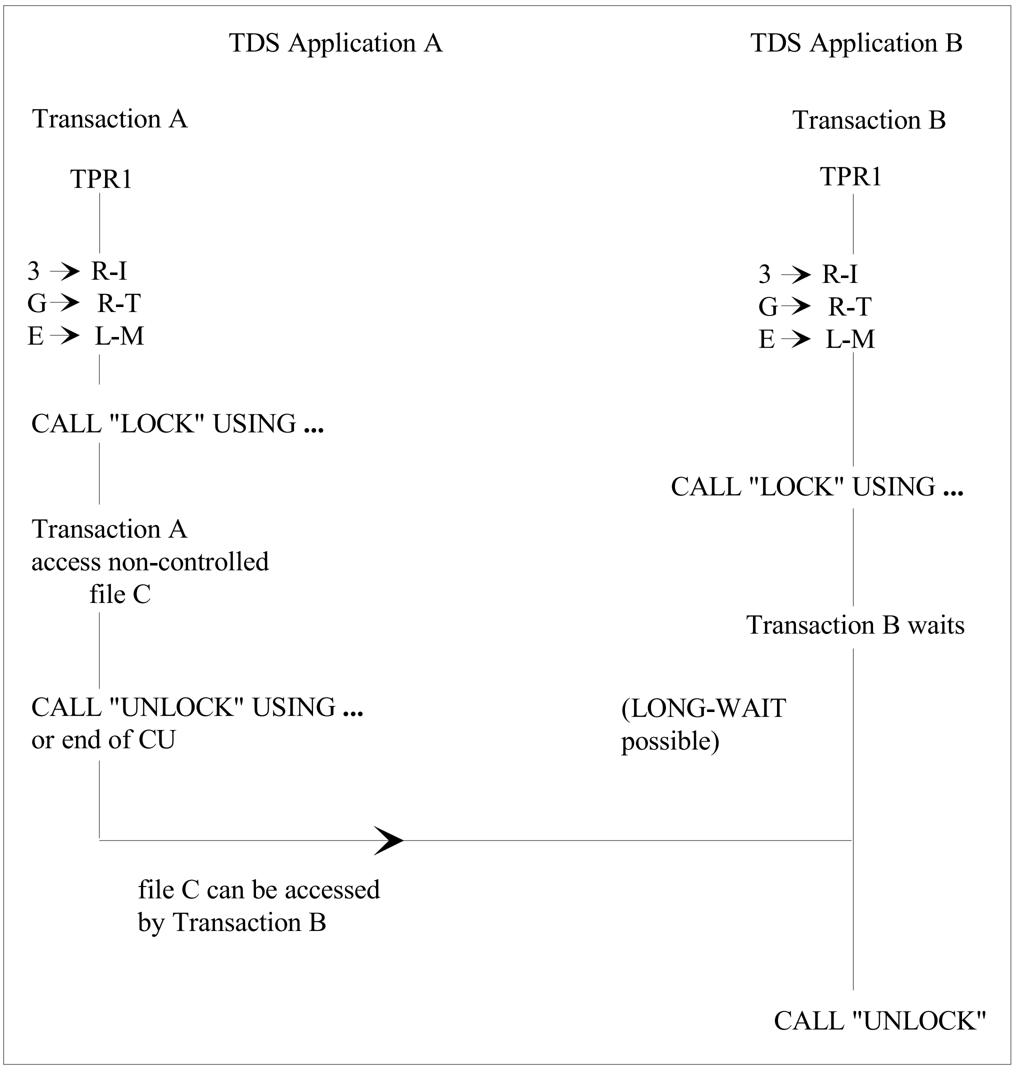


Figure 9-2. Using CALL LOCK for a Global Resource

❑

9.7 The CALL "NOCMIT" Procedure

Syntax

```
CALL "NOCMIT".
```

Description

CALL "NOCMIT" cancels the commitment action at the end of the current TPR, unless the TPR is the last TPR of the transaction.

Useful only when the IMPLICIT COMMITMENT option is used for the transaction.

Usage

CALL "NOCMIT" can be requested during the execution of the TPR, but takes effect at the end of the TPR.

CALL "NOCMIT" cancels any CALL "DFCMIT" called beforehand in the same TPR.

9.8 The CALL "RESET-NON-CONCURRENT" Procedure

Syntax

```
CALL "RESET-NON-CONCURRENT".
```

Description

Disables the non-concurrency mechanism for the next commitment unit. The transaction must be declared MANUALLY NON-CONCURRENT or NON-CONCURRENT at TDSGEN time.

Usage

Action is deferred until the start of the next commitment unit and has effect for the remainder of the transaction or until the CALL "SET-NON-CONCURRENT" statement is performed. See Figure 9-3.

Action is symmetric between transactions. If transaction A, non-concurrent with transaction B, calls RESET-NON-CONCURRENT, transaction A will not wait for transaction B, and transaction B will also not wait for transaction A.

9.9 The CALL "ROLL-BACK" Procedure

Syntax

```
CALL "ROLL-BACK".
```

Description

Rolls back the transaction to the first TPR of the current commitment unit.

Usage

The current TPR is immediately aborted.

Any modifications to the IDS/II database or files are invalidated.

Quarantined messages are purged.

The first TPR of the commitment unit is activated with the TRANSACTION-STORAGE, PRIVATE-STORAGE, and input message as they were when the commitment unit was initiated.

After the rollback, the RESTART-STATUS field of TDS-STORAGE is set to 2.

NOTE:
Data transferred between the transaction and the terminal during the commitment unit may be duplicated (sent another time).

9.10 The CALL "SET-NON-CONCURRENT" Procedure

Syntax

```
CALL "SET-NON-CONCURRENT".
```

Description

Activates the non-concurrency mechanism for the next commitment unit, whether the transaction is declared MANUALLY NON-CONCURRENT or NON-CONCURRENT at TDSGEN time. The CALL "SET-NON-CONCURRENT" procedure does not affect the current commitment unit.

Usage

Action is deferred until the start of the next commitment unit.

The non-concurrency mechanism is effective for the remainder of the transaction and can be cancelled by use of the CALL "RESET-NON-CONCURRENT" statement.

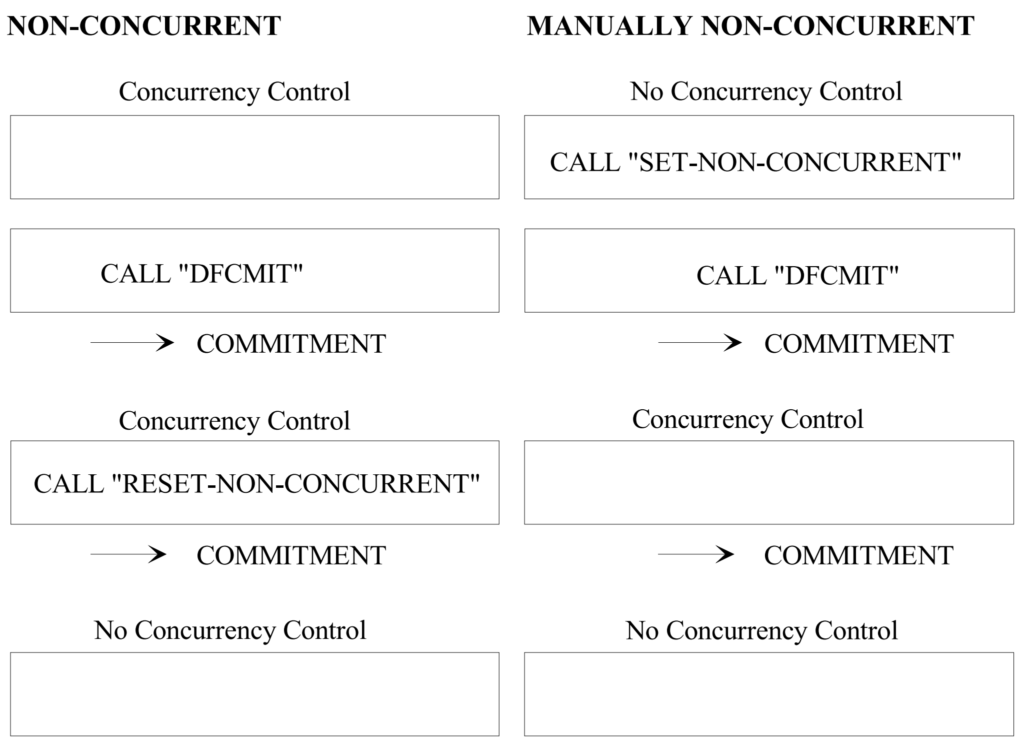


Figure 9-3. Non-concurrency

The left-hand column in Figure 9-3 shows that when the NON-CONCURRENT clause is defined at TDSGEN, non-concurrency is activated from the start or the transaction.

The right-hand column shows that when the MANUALLY NON-CONCURRENT clause is defined at TDSGEN, non-concurrency is activated only on the commitment unit after the SET-NON-CONCURRENT procedure is called.

9.11 The CALL "UNLOCK" Procedure

Syntax

```
CALL "UNLOCK" USING unlock-description, data-name-1.
```

Description

Unlocks a resource that has been previously locked. By using the unlock facility, you can free a resource before the next commitment point. See Figure 9-1 and Figure 9-2.

Usage

Unlock-description is a data structure of the following format:

```
01  UNLOCK-DESCRIPTION.
    02  RESOURCE-IDENTIFIER   COMP-1.
    02  RESOURCE-TYPE         PIC X.
```

- RESOURCE-IDENTIFIER is an integer that ranges from 0 to 16383. It identifies the resource to be released and must correspond to the identifier used to lock the resource.
- RESOURCE-TYPE defines the resource as it was defined for LOCK.

L (local)	the resource is private to the application.
G (global)	the resource is common to all the applications.

`data-name-1` is a single numeric character in which the status is returned:

0 = successful execution

1 = wrong argument

2 = unknown resource identifier

9.12 The CLOSE Verb

Syntax

```
CLOSE ifn.
```

Description

Allows a TPR to close any file. The file may be a TDS-controlled file or a TDS non-controlled file declared or not declared at TDSGEN.

For files declared at TDSGEN, the CLOSE verb is executed asynchronously. So, a further OPEN request issued by a transaction may be rejected.

For files not declared at TDSGEN, the CLOSE verb takes effect immediately.

NOTE:
At the start of the TDS session, all files declared in TDSGEN and assigned to the TDS job are opened by TDS. During the TDS session, the user can close a file. The file remains closed until the user issues an OPEN statement.

Usage

The status must be tested through the primitive H_CBL_UGETG4 in the declaratives clause in COBOL. Refer to the *COBOL 85 User's Guide*.

For the status values, see the *Messages and Return Codes Directory*.

9.13 The OPEN Verb

Syntax

```
      { INPUT        }
      { INPUT-OUTPUT }
 OPEN {              }  , ifn.
      { OUTPUT       }
      { EXTEND       }
```

Description

This statement allows a TPR to open a file. This file may be one of the following:

- a TDS-controlled file (such a file may be opened only in INPUT or UPDATE mode),
- a file that is not controlled by TDS but which is declared in TDSGEN,

Usage

The file must be assigned, otherwise invalid status is returned,

If the file is TDS-controlled, the TPR issuing the OPEN statement is denied access to the file before taking a commitment.

Use of the COBOL verb ASSIGN for dynamic file assignment is forbidden in TDS.

NOTE:
Do not use the OPEN statement for non-declared files: the result can be either exceptions inside COBOL run-time or an abnormal return code (IFNSTRU).

Wrong status may be reported and must be tested for by using the primitive H_CBL_UGETG4 in the declaratives clause in COBOL. Refer to the *COBOL 85 User's Guide*.

9.14 The CALL "GET-SYNCSTATE" Procedure

Syntax

```
CALL "GET-SYNCSTATE" USING sync-state.
```

Description

The function is used to determine the synchronization state of committed data. It refers to the result of the last commitment (or rollback in the event of a transaction abort), for the user session at the time the function was invoked.

It is strongly recommended that each TDS-XA TPR should begin with this function in order to be aware of the state of the XA resources before proceeding with data processing.

Output Parameter

sync_state is represented by a 1-character field with one of the following two values:

0 SYNC: synchronized state

1 DESYNC: desynchronized state

A failure has disrupted the last commitment or rollback and the commitment unit must be resynchronized by TDS.

A DESYNC state warns the user that an incident has occurred. Before continuing, the user should contact the TDS administrator for further information about the incident, the desynchronization-resynchronization process, and the final state of his last commitment unit (committed or rolled back).

If the last commitment performed for this session was not involved with XA protocol, the primitive completes successfully and returns 0.

The GET-SYNCSTATE procedure can be invoked from the ON_ABORT_TPR to determine the result of the transaction abort rollback.

10. FORMS Procedures

This chapter deals with the following procedures:

CDATTL and CDATTR	apply a list of rendition attributes to a given selection of fields.
CDFIDI	returns the name and occurrence number of the form to be received by the transaction.
CDGET	activates a form.
CDMECH	applies a mechanism to determine how fields in a form are to function.
CDPURGE	purges all input messages and gives the turn back to the application.
CDRECV	receives a named data-form.
CDRELS	releases all active forms and puts the terminal in line mode.
CDSEND	sends data from a data record to the terminal.

10.1 The CALL "CDATTL" Procedure

Syntax

```
CALL "CDATTL" USING output-cd-alias, selection-vector,
            attribute-identifier [,enclosure-level].
```

Description

Applies a list of rendition attributes to fields selected in the selection vector.

Usage

`output-cd-alias`: the contents of "symbolic-source" in the input-cd-alias must be moved into "symbolic-destination" of the output-cd-alias.

`selection-vector` specifies the form name and occurrence number of the form to which the command applies.

For the other fields of the selection-vector, see CALL "CDATTR".

`attribute-identifier` is a structure defined as follows:

```
01 attribute-identifier.
02 data-name      PIC 9(3)    VALUE "number of attributes".
02 data-name-1    PIC X(4)    VALUE "attribute".
  .
  .
  .
02 data-name-n    PIC X(4)   VALUE "attribute".
```

where n = "number of attributes".

For the list of attributes, see CALL "CDATTR".

`enclosure-level` specifies how the message terminates as follows:

1 - end-of-record	The message is quarantined but is not sent to the terminal. This is the default value. This is equivalent to ESI in TDS terms.
2 - end-of-quarantine	All quarantined messages are now sent to the terminal. The application still retains the turn. This is equivalent to EMI in TDS terms.
3 - end-of-interaction	All quarantined messages are now sent. The turn is given to the terminal. This is equivalent to EGI in TDS terms.

10.2 The CALL "CDATTR" Procedure

Syntax

```
CALL "CDATTR" USING output-cd-alias, selection-vector,
                    attribute-identifier [,enclosure-level].
```

Description

Applies an attribute to fields selected in the selection vector. If the enclosure-level is 1 and if all enclosure-levels associated with subsequent CALL statements (including CALL "CDRELS" itself) are also 1, then changes to the attribute values are lost after the CALL "CDRELS" statement has been executed.

If a form has been created with THE SUBSTITUTE ATTRIBUTE clause, an attribute that does not exist for a given terminal, is replaced (e.g., if BD is selected for a VIP7760, RV is substituted).

An attribute remains modified until a call to

- either CDATTR/CDATTL (to change an attribute with an explicit value),
- or CDMECH with INITAT (to reset all attributes to their initial values).

Usage

`output-cd-alias`: the contents of symbolic-source in the input-cd-alias must be moved into "symbolic-destination" of the output-cd-alias.

`selection-vector` specifies the form name and occurrence number of the form to which the command applies. Depending on the action required for a particular named field (NF), the other fields of the selection vector are set as follows:

space	Do not affect the attributes of the associated NF.
S	Affect the attributes of the associated NF.
C	Clear the contents of the associated NF.
B	Clear the contents of the associated NF and affect the attributes as specified.

`attribute-identifier` is a four-character data item that may take the following values:

BI	Blink
BD	Bold (that is, high intensity)
Bxxx	Background color, where `xxx` may be:

RED	Red
YEL	Yellow
BLU	Blue
GRE	Green
CYA	Cyan
MAG	Magenta
WHI	White
BLA	Black
DFT	Default color
CN	Conceal
COS	Column separator
CP	Cursor position
DFT	Default rendition (NBI, NHL, NRV, NCOS, NUL, BDFT, FDFT, and normal intensity)
	Faint (decreased intensity)
FT	Foreground color, (for xxx,
Fxxx	see Bxxx)
	Rendition highlighted
HL	specific to terminal
	Initial attributes
INIT	No blink
NBI	No column separator
NCOS	Not highlighted
NHL	Not protected
NPR	Not reverse video
NRV	Not underlined
NUL	Protected
PR	Reverse video
RV	Underlined
UL	

1 - end-of-record	The message is quarantined but is not sent to the terminal. This is the default value. This is equivalent to ESI in TDS terms.
2 - end-of-quarantine	All quarantined messages are now sent to the terminal. The application still retains the turn. This is equivalent to EMI in TDS terms.
3 - end-of-interaction	All quarantined messages are now sent. The turn is given to the terminal. This is equivalent to EGI in TDS terms.

10.3 The CALL "CDFIDI" Procedure

Syntax

```
CALL "CDFIDI" USING input-cd-alias, form-identifier.
```

Description

Returns the name and occurrence number of the form to be received by the transaction. CDFIDI can be used to determine the name of a form loaded by the preceding transaction.

Usage

`input-cd-alias`: the contents of "SYMBOLIC-QUEUE" of TDS-STORAGE must be moved into "symbolic-queue" of the input-cd-alias.

`form-identifier` has the following format:

```
01  form-identification.
  02  form-name           PIC X(8).
  02  occurrence-number   PIC 9(3).
```

10.4 The CALL "CDGET" Procedure

Syntax

```
CALL "CDGET" USING output-cd-alias, form-nameI
                   [,enclosure-level].
```

Description

Activates a form depending on the values specified in form-nameI, according to the following modes:

APPEND	If the APPEND mode is specified, that is, the field form-name-MD is set to A, all forms following the form specified by the old partition name (form-name-OF) and occurrence number (form-name-OO) fields of the structure form-nameI are released and cleared. The new form is appended after the old form. If the old form and occurrence number fields specify a name of spaces and an occurrence of zero (default initialization), the new form will be appended at the top of the screen and all other forms will be released and cleared.
OVERLAY	If the OVERLAY mode is specified, that is, the field form-name-MD is set to O, the old form and occurrence number fields must specify a name of spaces and an occurrence number of zero. The forms that were active are frozen, that is, they remain visible on the screen but all their fields become protected and they are no longer addressable from the program. The new form is activated at the top of the screen.

WINDOW	If the WINDOW mode is specified, that is, the field form-name-MD is set to W, the forms that were active are frozen. If the form to be activated is already displayed on the screen with the same occurrence number, this form is put on top of the other forms and activated again with its previous contents. Otherwise, the form-name-SL and form-name-SC fields determine the line and column numbers of the top left corner of the rectangle where the form is to be placed. The contents of this rectangle are cleared and the form is activated.
ERASE	If the ERASE mode is specified, that is, the field form-name-MD is set to E, all the forms are released and the screen is cleared. The form-name-SL and form-name-SC fields determine the line and column numbers of the top left corner of the rectangle where the form is to be placed.

Figure 10-1 shows examples of forms in OVERLAY and APPEND modes and Figure 10-2 shows examples of forms in the ERASE and WINDOW modes.

The maximum number of active or frozen forms is 6.

When CDGET is executed for a new form, all preceding cursor requests are ignored.

By default, the cursor is positioned either on the first field having the initial Cursor Position (CP) attribute, or on the first unprotected field. If the new form is activated in APPEND Mode and does not contain any field with the CP attribute or any unprotected fields, the cursor is positioned on the first field having the CP attribute or on unprotected field.

Usage

`output-cd-alias`: the contents of "symbolic-source" in the input-cd-alias must be moved into "symbolic-destination" of the output-cd-alias.

`form-nameI`: the fields in form-nameI are set as follows:

- form-name-MD: specifies the mode of activation of the form. (A for Append, O for Overlay, W for WINDOW, E for ERASE),
- form-name-OF and form-name-OO are set according to where the new form is to appear, namely:

  if the new form is to appear at the top of the screen, form-name-OF = spaces and form-name-OO= 0

  if the new form is to be appended after an old form (APPEND Mode only), both parameters contain user-defined values of the old form.

`enclosure-level` specifies how the message terminates as follows:

1 - end-of-record	The message is quarantined but is not sent to the terminal. This is the default value. This is equivalent to ESI in TDS terms.
2 - end-of-quarantine	All quarantined messages are now sent to the terminal. The application still retains the turn. This is equivalent to EMI in TDS terms.
3 - end-of-interaction	All quarantined messages are now sent. The turn is given to the terminal. This is equivalent to EGI in TDS terms.

EXAMPLE OF HOW THE CALL "CDGET" STATEMENT FUNCTIONS

To modify the default cursor position, call CDATTR with the CURSOR-POSITION (CP) argument. CP causes the cursor to be positioned to the leftmost character of the field as follows:

```
CDGET     F1
CDGET     F2   Append to   F1        enclosure-level:1
CDATTR    F1   CP  (field selected)  enclosure-level:2
```

If, however, the following sequence is used, CDATTR is ignored:

```
CDGET     F1
CDATTR    F1   CP  (field selected)
CDGET     F2   Append to   F1        enclosure-level3
```

❑

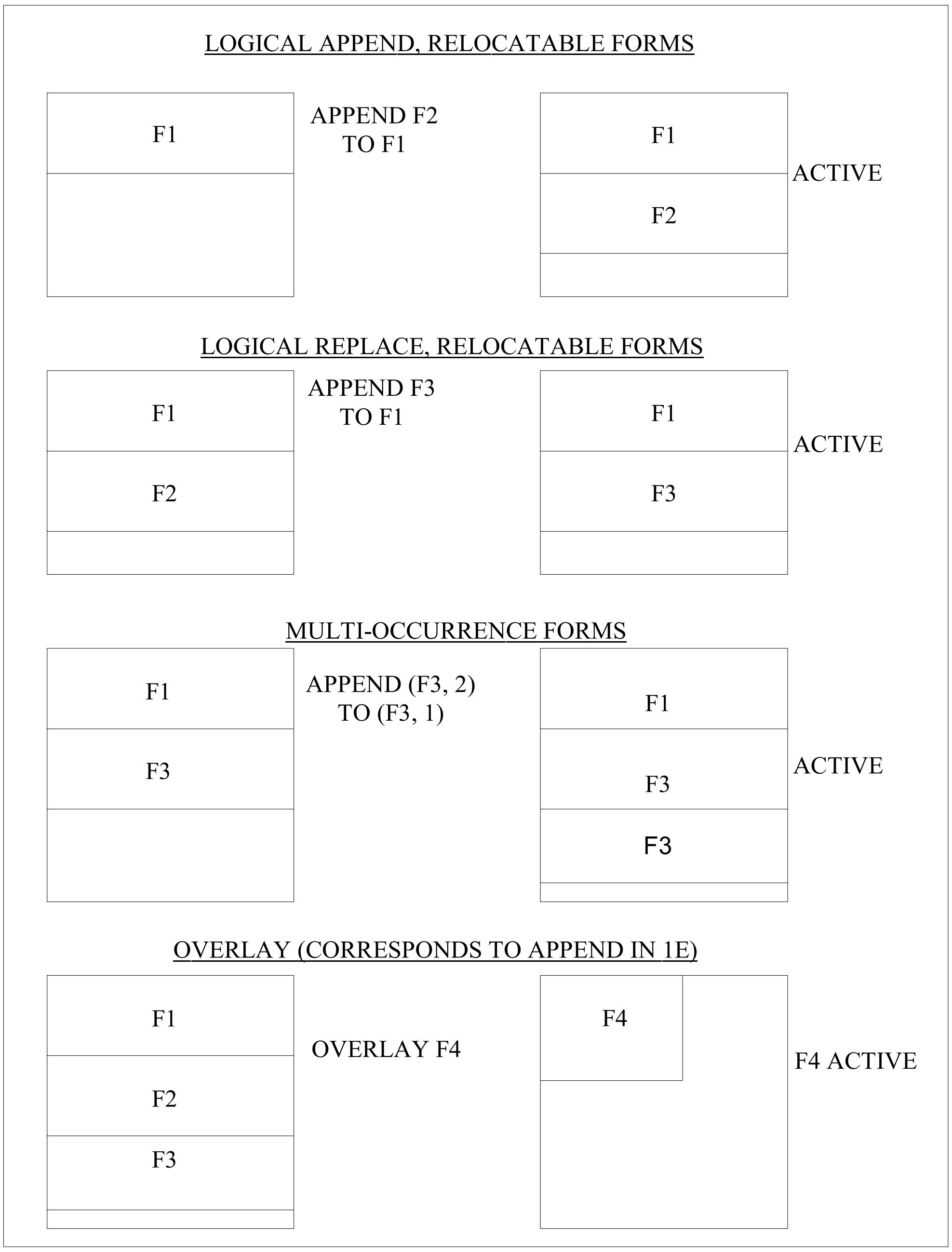

LOGICAL APPEND, RELOCATABLE FORMS
F1
APPEND F2 TO F1
F1
F2
ACTIVE
LOGICAL REPLACE, RELOCATABLE FORMS
F1
F2
APPEND F3 TO F1
F1
F3
ACTIVE
MULTI-OCCURRENCE FORMS
F1
F3
APPEND (F3, 2) TO (F3, 1)
F1
F3
F3
ACTIVE
OVERLAY (CORRESPONDS TO APPEND IN 1E)
F1
F2
F3
OVERLAY F4
F4
F4 ACTIVE

Figure 10-1. Forms in Overlay and Append Mode

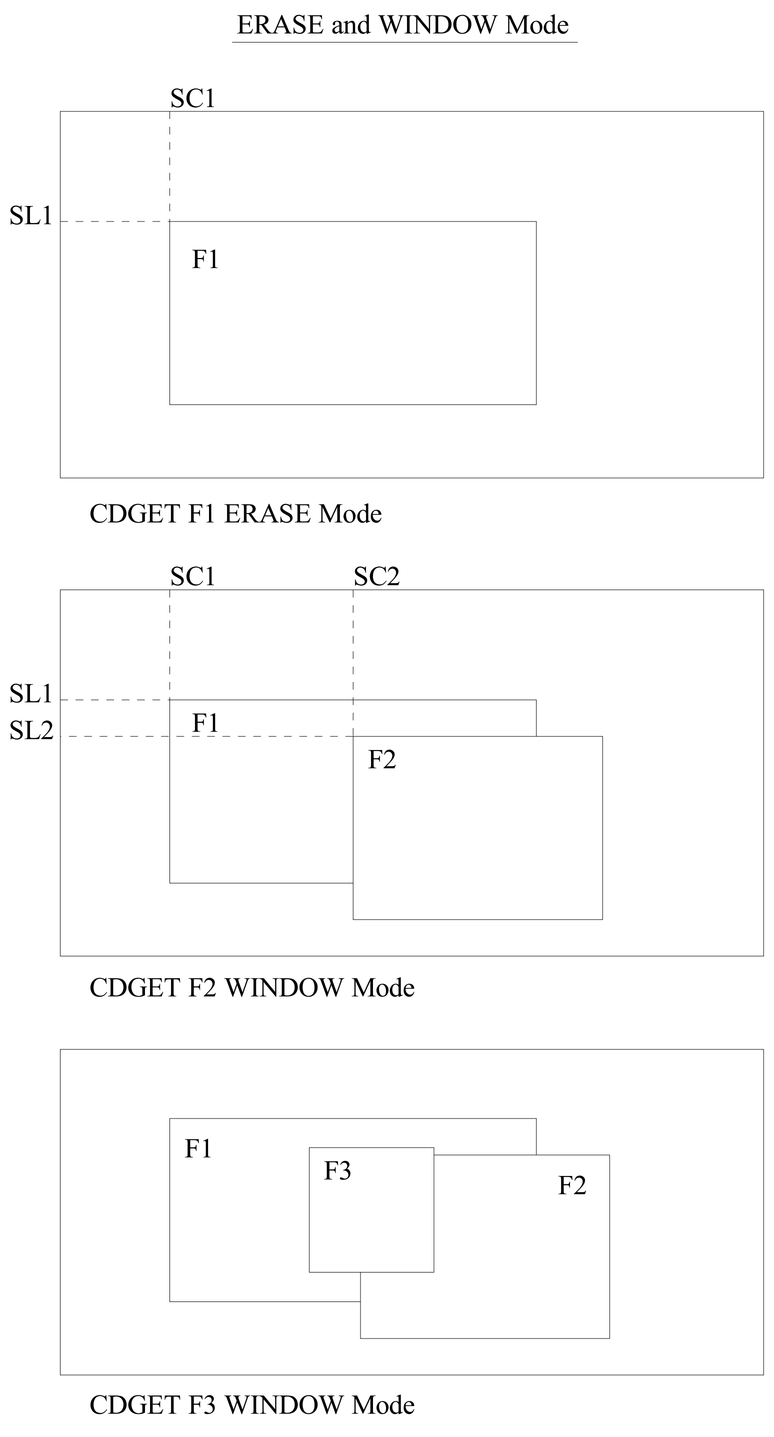
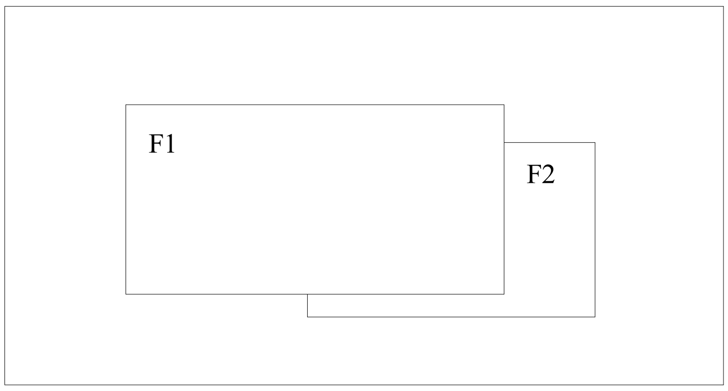
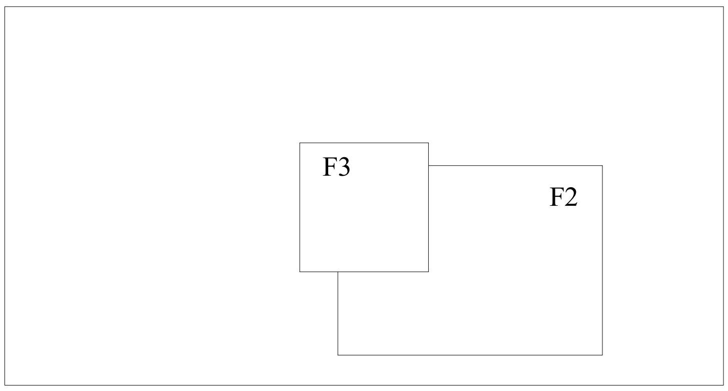


Figure 10-2. Forms in Erase and Window Mode (1/2)



CDGET F1 WINDOW mode



CDMECH POPUP

Figure 10-2. Forms in Erase and Window Mode (2/2)

10.5 The CALL "CDMECH" Procedure

Syntax

```
CALL "CDMECH" USING output-cd-alias, mechanism-identifier
                    [,enclosure-level].
```

Description

Applies a mechanism to determine how fields in a form are to function. If the enclosure-level is 1 and if all enclosure-levels associated with subsequent CALL statements (including CALL "CDRELS" itself) are also 1, then changes to the attribute values are lost after the CALL "CDRELS" statement has been executed.

Usage

`output-cd-alias`: the contents of "symbolic-source" in the input-cd-alias must be moved into "symbolic-destination" of the output-cd-alias.

`mechanism-identifier` is a six-character alphanumeric data item that may take the following values:

ALARM	activates the audio or visual alarm
CLEAR	clears all unprotected fields
PROTCT	protects all NFIELDS of all active forms. This command is effective on the next CDGET.
INITAT (alias RESET)	clears all unprotected fields and resets attributes to their initial values. When the screen is in line mode, (that is, after CDRELS with end-key=1), this command clears the screen.
INIT	resets all forms to their initial state
STPRV	valid only for a printer. All subsequent CDSENDs print only the variable fields.
STPRA	subsequent CDSENDs print all fields. This is the default value.

CPON	sets up a mode where the TPR will be notified of the cursor position in the subsequent CDRECV statements, for terminals supporting this feature (QUESTAR 200, IBM3278/79, MINITEL).
CPOFF	resets the previous CPON.
POPUP	when the active form has been activated in the WINDOW mode, the POPUP mechanism releases the form and activates the previous form in the stack of displayed forms. The window associated with the released form is reset to the underlying contents. When the active form has not been activated in the WINDOW mode, or when the stack of displayed forms is reduced to one form (that is, no previous form exists), the return code FUNCNAV is sent.

`enclosure-level` specifies how the message terminates as follows:

1 -end-of-record	The message is quarantined but is not sent to the terminal. This is the default value. This is equivalent to ESI in TDS terms.
2 - end-of-quarantine	All quarantined messages are now sent to the terminal. The application still retains the turn. This is equivalent to EMI in TDS terms.
3 - end-of-interaction	All quarantined messages are now sent. The turn is given to the terminal. This is equivalent to EGI in TDS terms.

10.6 The CALL "CDPURGE" Procedure

Syntax

```
CALL "CDPURGE" USING input-cd-alias.
```

Description

Purges all pending input messages and gives the turn back to the application.

Usage

`input-cd-alias`: the contents of the "SYMBOLIC-QUEUE" field of TDS-STORAGE must be moved into "symbolic-queue" of the input-cd-alias.

10.7 The CALL "CDRECV" Procedure

Syntax

```
CALL "CDRECV" USING input-cd-alias, form-nameR, wait-indicator,
                    selection-vector.
```

Description

Receives a data record form-nameR, according to the selection-vector form-nameV. For each field in the data record form-nameR that is to receive data, the corresponding field in the selection vector form-nameV must be set to "S".

The form name and occurrence-number of the selection vector form-nameV must match a form and occurrence number for which data is available.

On receiving data, the "end-key" of the input-cd-alias specifies the enclosure level associated with the data. This is a one-character data item that contains one of the following values:

1 - End-of-record	More data in the same message to be received.
3 - End-of-interaction	All data in the message has been received. The turn is given to the application.

Usage

`input-cd-alias`: the contents of "SYMBOLIC-QUEUE" of TDS-STORAGE must be moved into "symbolic-queue" of the input-cd-alias.

`wait-indicator` is set to either 1 or 0 and determines the action to be taken on expiration of WAIT−TIME declared in TDS−STORAGE.

If set to 0:

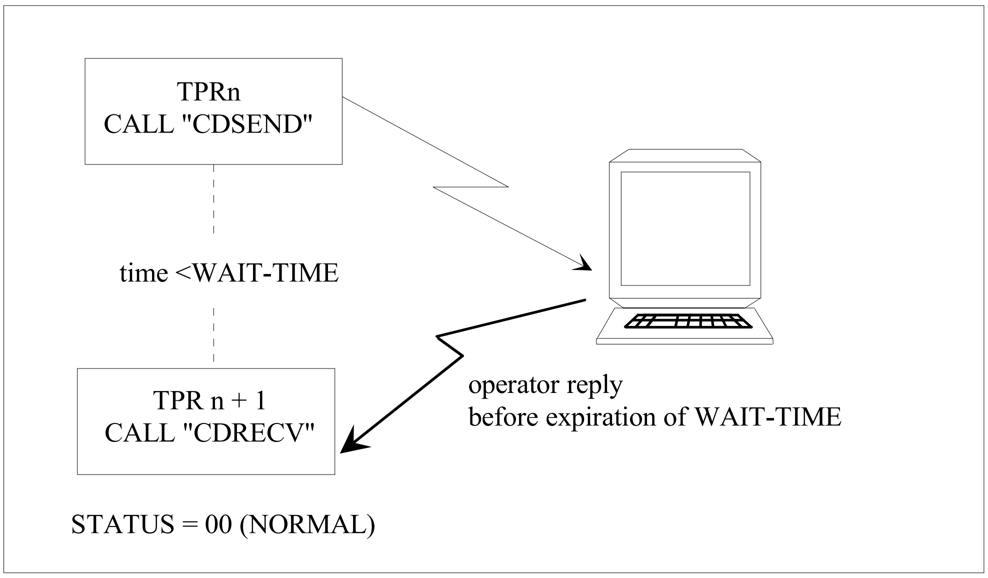
If the operator has entered a response before the expiration of WAIT-TIME, STATUS = 00 (NORMAL): see Example 1.

If the operator has not replied, on expiration of WAIT-TIME, STATUS = 3 (RCVVIOL): see Example 2.

If set to 1:

Whether the operator replies or not, and whether WAIT-TIME has expired or not, STATUS = 00 (NORMAL).

EXAMPLE: WAIT-INDICATOR = 0 OR 1



❑

`selection-vector` is set to one of the following values after execution,

R	Normal reception. Valid data has been moved to the corresponding field.
S	No data is available for the field, i.e. either the field is not transmittable or its contents are set to null or to spaces.
+	A sign was entered in an unsigned field. Data has not been transferred.
T	Data has been transferred, but the least significant digits have been truncated.
O	Data has not been transferred, because the significant digits would have been truncated.
A	Data has not been transferred, because incorrect characters have been input to NFIELD.
C	This field contains the cursor (may also be returned for a non-selected field).
X	This field contains the cursor and valid data has been moved to it (R + C).
D	This field has the attribute DT or IT and data are received in the field.

The values O, T, and + can be returned only for fields with a numeric or numeric edited SCREEN PICTURE or COMP-1 usage.

Before execution, all selection-vector fields, that is, those in form-nameV, must be set to spaces and the fields to be selected must be set to S.

If data is available for a non-selected field, the data is not transferred and the corresponding field is set to L (=lost); if no data is available the selection-vector field is not affected.

EXAMPLE OF SELECTION-VECTOR VALUES

For SPIC="99.99", the values keyed in at the terminal change the selection vector as follows:

Selection Vector Before	Value Keyed in	Value Received in TPR	Selection Vector After
S	12.34	1234	R
S	1.23	0123	R
S	1.2	0120	R
S	1.234	0123	T
S	$1.2	none	A
S	1234	none	O
S	none	none	S
S	+1.23	none	+
Blank	none	none	Blank
Blank	1234	none	L

❑

10.8 The CALL "CDRELS" Procedure

Syntax

```
CALL "CDRELS" USING output-cd-alias [,enclosure-level].
```

Description

Releases all active forms and puts the terminal in line mode. However, the screen is not cleared. To clear the screen, CDRELS must be followed by CDMECH with the INITAT option. When the CALL "CDRELS" statement is executed and the enclosure-level is 1, changes to the attribute values are lost.

Usage

`output-cd-alias`: the contents of "symbolic-source" in the input-cd-alias must be moved into "symbolic-destination" of the output-cd-alias.

`enclosure-level` specifies how the message terminates as follows:

1 - end-of-record	The message is quarantined, but is not sent to the terminal. This is the default value. This is equivalent to ESI in TDS terms.
2 - end-of-quarantine	All quarantined messages are now sent to the terminal. The application still retains the turn. This is equivalent to EMI in TDS terms.
3 - end-of-interaction	All quarantined messages are now sent. The turn is given to the terminal. This is equivalent to EGI in TDS terms.

10.9 The CALL "CDSEND" Procedure

Syntax

```
CALL "CDSEND" USING output-cd-alias, form-nameR, enclosure-level,
                                       selection-vector.
```

Description

Sends data from the data record form-nameR to the terminal according to the criteria specified in the selection-vector.

After the execution of the CALL "CDSEND" statement, the contents of the selection vector fields are unchanged. If the contents of a selected numeric field are invalid, the selection vector field is loaded with an "A".

Before sending data, the transaction must have received all the data; otherwise it aborts with the SNDVIOL abort code.

Usage

`output-cd-alias`: the contents of "symbolic-source" in the input-cd-alias must be moved into "symbolic-destination" of the output-cd-alias.

`enclosure-level` specifies how the message terminates as follows:

1 - end-of-record	The message is quarantined, but is not sent to the terminal. This is the default value. This is equivalent to ESI in TDS terms.
2 - end-of-quarantine	All quarantined messages are now sent to the terminal. The application still retains the turn. This is equivalent to EMI in TDS terms.
3 - end-of-interaction	All quarantined messages are now sent. The turn is given to the terminal. This is equivalent to EGI in TDS terms.

selection-vector	specifies the form name and occurrence number of the form to which the command applies. Depending on the action required for a particular NF, the other fields of the selection vector are set as follows:
"space"	Do not move the associated NF from the data record.
S	Move the associated NF from the data record.
C	Clear the contents of the NF.

11. GTWRITER Procedures

GTWRITER is described in Chapter 2. For a detailed description of how to use GTWriter procedures, refer to the *Generalized Terminal Writer User's Guide* 47 A2 55 UU.

.

H_TW_USTART is an old interface but is still assumed;

It is replaced by H_TW_USTARTE.

Other procedures:

H_TW_UCOMM	sends a GTWriter command to the Command Handler.
H_TW_UDRE	returns the status of a specified driver.
H_TW_UGETR	returns the allocated report number.
H_TW_UMAINE	reads fields in the main GTWriter table.
H_TW_UQNE	reads the next report in the GTWriter queue.
H_TW_UQRE	reads a report description from the GTWriter queue.
H_TW_USAVE	saves a report member in the SITEOUT library.
H_TW_USTARTE	opens a report.
H_TW_UTRE	returns a description of a terminal and its state.

This list is not exhaustive.

See *Generalized Terminal Writer User's Guide for more information.*

Note : An other file than SYS .TW.OUT may be used by suffixing the EFN by one character chosen in set (0..9,A..Z).

Examples: SYS.TW.OUT4 or SYS.TW.OUTB .
(see parameter MULTI_SYS_TW_OUT in *Generalized Terminal Writer* User's Guide).

12. Special-purpose Transactions and the Transaction Initialization Routine

12.1 Overview

This section discusses certain special-purpose transactions, written by the user, which are called by TDS in circumstances such as TDS startup/shutdown or user log-on/log-off. Each transaction must be written according to certain rules that are described below.

The special-purpose transactions are activated when the following events occur:

- The LOGON transaction is activated each time a correspondent connects successfully.
- The LOGOUT transaction is activated each time a correspondent disconnects normally.
- The RESTART transaction is activated when a user logs onto TDS and a context already exists for the user.
- The DISCNCT transaction is activated each time a correspondent disconnects abnormally.
- The STARTUP transaction is activated when TDS has started successfully.
- The SHUTDOWN transaction is activated when TDS terminates normally or abnormally.
- The BREAK transaction is activated after a break signal is received from a correspondent.

Figure 12-1 shows how the special-purpose transactions interact.

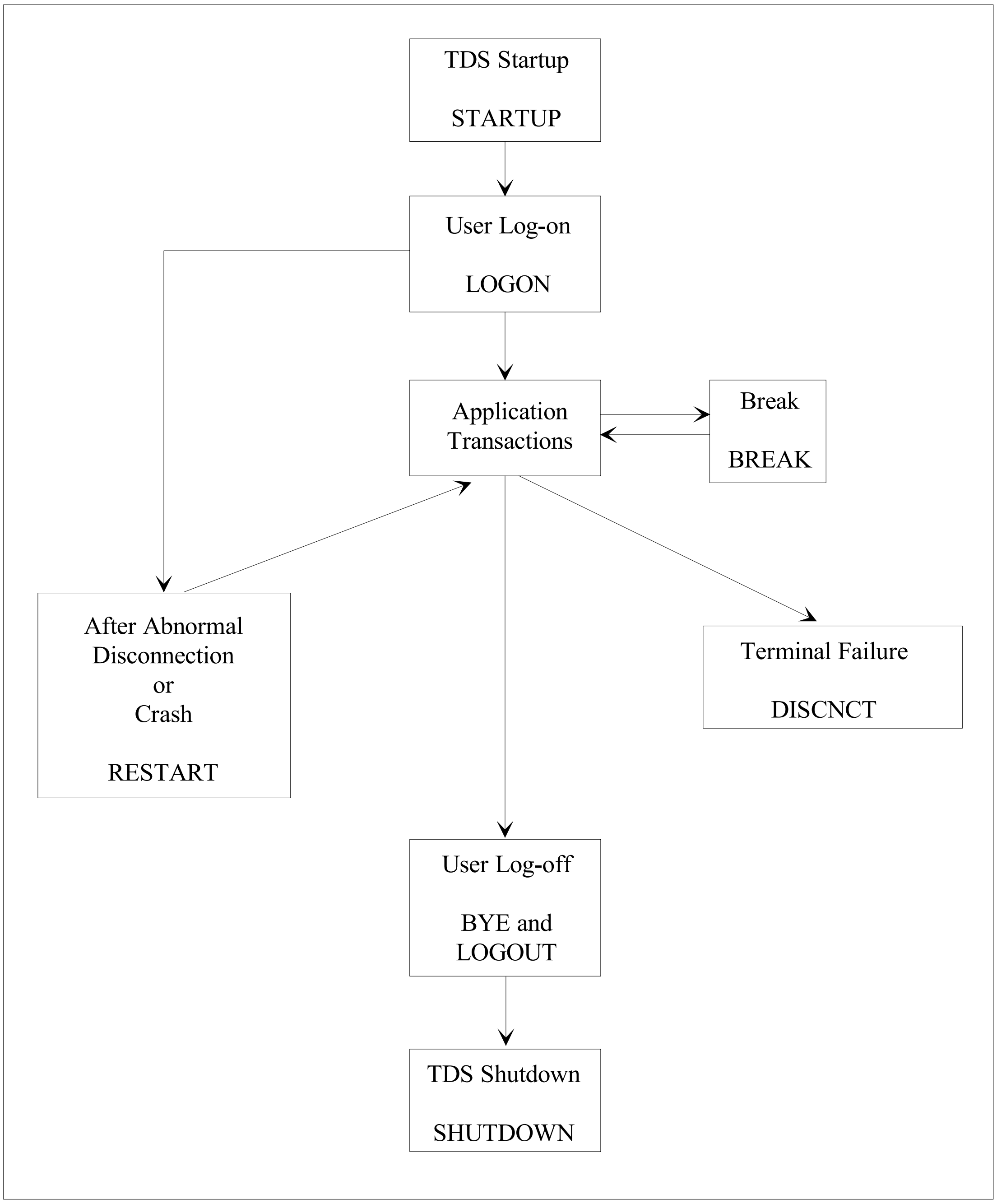


Figure 12-1. Context of Special Services

In all these events, the following rules apply:

- If a message-id has been defined in the TRANSACTION-SECTION of TDSGEN with the same name as the corresponding event (for instance: message-id LOGON for a logon event), the first TPR of this transaction is activated. The TPR may have any identification. If a user wishes to define a special-purpose transaction, the following syntax is used in TDSGEN:

  ```
  TRANSACTION SECTION.

  MESSAGE "LOGON" ASSIGN TO UTPRLG
  .
  .
  .
  MESSAGE "BREAK" ASSIGN TO UTPRBR
  .
  .
  .
  MESSAGE "SHUTDOWN" ASSIGN TO UTPRSH
  ```

  When LOGON is entered by TDS, the user TPR UTPRLG is executed. Similarly, when SHUTDOWN is entered by TDS, the user TPR UTPRSH is executed.

- If no message-id has been defined in the TRANSACTION-SECTION of TDSGEN, TDS gives control to a TPR having the same name as the event (for instance: LOGON TPR) if there is one. One exception is the DISCNCT transaction that requires that the TPR named DISCONNECT be activated.
- If no corresponding TPR is found, TDS default TPRs are then activated.
- An advantage of defining one's own special transactions in TDSGEN is that the user may modify the processing otherwise applied by default by TDS. If the user writes these TPRs, for example, LOGON, BREAK, it is recommended that the user write the corresponding transactions in TDSGEN. Otherwise, the authority codes, sizes, etc are set by the TDS-supplied transactions, which may not be suitable.

- In all cases, the transactions are processed like any interrupt transaction:
  - A new context is allocated to the transaction, the interrupted context being saved.
  - TRANSACTION-STORAGEs are initialized according to each special transaction. The different TRANSACTION-STORAGEs are described in the following subsections. The TRANSACTION-STORAGE size is defined in TDSGEN.
  - If a PRIVATE-STORAGE is defined, the size must be defined in the TRANSACTION-STORAGE of the relevant transaction. The special transaction is passed the PRIVATE-STORAGE of the terminal.
  - When the transaction terminates (with spaces in the NEXT-TPR field), the context is released and the interrupted transaction if any (case of the BREAK transaction) is automatically resumed.
- No restrictions apply to the processing activated by these transactions in terms of database access or data exchange (except DISCNCT that may not send data to its correspondent). Rules for commitment and rollback apply in the same way as for normal transactions.
- All the special-purpose transactions except the BREAK transaction are not restarted at TDS warm restart if they have been terminated abnormally because of a TDS abort or a system crash. Any referenced files remain in the state they were in at the time of the last valid commitment.
- When a BREAK signal is received on a session for which a special-purpose transaction (except BREAK) is running, the BREAK signal is ignored.

12.2 BREAK Transaction

The BREAK transaction can be defined at TDSGEN. Then it replaces the standard BREAK processing. This can be done by assigning a TPR to the message-id BREAK in the MESSAGE statement in the TRANSACTION SECTION at TDSGEN or simply by naming a TPR "BREAK". The BREAK transaction is activated when a Break message (simulated or not) is received from a correspondent.

In previous releases, BREAK event handling was confined to the BREAK transaction, but in release V6, you can handle a BREAK event in any transaction. This means that any transaction processing a BREAK event is executed according to the COMMITMENT options specified for the MESSAGE statement declared in the TRANSACTION SECTION at TDSGEN:

WAIT	The current transaction will be interrupted at the next commitment point and the BREAK transaction is activated.
ROLL-BACK	When the current TPR ends, the current transaction is rolled back to the previous commitment and the BREAK transaction is activated.
No Value Specified	Takes the value specified for the BREAK transaction itself.
No Value Specified for the BREAK transaction	ROLL-BACK applies.

How the BREAK Transaction is Processed

When the current transaction is interrupted, the following occur:

- the context of the interrupted transaction is retained
- the first TPR of the BREAK transaction is activated. The TDS-STORAGE fields are initialized as for the first TPR of any transaction.

This first TPR must issue a RECEIVE NO DATA in order to be allowed to send messages later.

The BREAK transaction is organized and processed like any other transaction. The first TPR may chain to any other TPR. It can be aborted, (CALL "ABORT" using abort-code), in which case, the interrupted transaction is resumed at its last commitment point. The CALL "RESTORE" and CALL "CANCELCTX" statements may be used. These are described in Chapter 4.

The BREAK transaction may itself be interrupted by another BREAK message.

Therefore, the same processing applies and a new BREAK transaction is entered. The context of the interrupted BREAK transaction is stacked. Although the theoretical limit to the maximum number of interrupts may be the capacity of the internal TDS swap file, TDS may limit the maximum number of interrupts to about 16, by ignoring the BREAK signal.

The BREAK transaction terminates according to the options specified for the NEXT-TPR field of TDS-STORAGE.

NEXT-TPR = "spaces".	The interrupted transaction is resumed where the BREAK transaction was activated.
NEXT-TPR = CANCELTX.	The interrupted transaction, if any, is aborted with the abort-code "BREAK". ON-ABORT-TPR, if any, will be activated.
NEXT-TPR = ENTERTX.	The terminal is set to active (command mode). The READY message indicates that the user has the turn and can enter a new transaction while the interrupted transaction is retained. When the new transaction terminates and sets NEXT-TPR to "spaces", the interrupted transaction resumes.
Terminal Re-connection	When an incident occurs during break transaction processing, the terminal will be restarted at the last commitment point of the transaction that was current at the disconnection point. All the contexts are restored to the state existing at that commitment point.

Form Handling

When a transaction is interrupted to process the BREAK transaction, the terminal is set to non-formatted mode.

When the transaction resumes:

all the forms activated during the BREAK processing are released,

all the forms which were active at the break time are re-activated,

all the variable fields are re-transmitted to the terminal in such a way that the user is re-established in the exact situation as at break time.

During BREAK Processing:

when the BREAK transaction, at the end of its processing, starts another transaction through the command NEXT-TPR = ENTERTX, the form mounted by the BREAK transaction is kept if the NO IMPLICIT RELEASE option is specified; the form is released if the NO IMPLICIT RELEASE option is not specified.

No Break Transaction Specified

If no BREAK transaction is specified, TDS performs the following:

if no transaction is currently executing, TDS switches a passive terminal to the active state and sends READY to the terminal,

otherwise, TDS aborts a currently executing transaction with the return code BREAK.

12.3 DISCNCT Transaction

The DISCNCT transaction is activated when a correspondent is accidentally disconnected.

The following actions take place if the disconnection occurs while a TPR is executing for this terminal:

- the TPR is aborted immediately if TDS verbs or procedures are executed (RECEIVE, SEND, CALL ROLL-BACK, CALL RESTORE, CALL KEEP-CURRENCIES, CALL CDRECV, CALL CDSEND, CALL SUBJOB, CALL SET-ACTIVE, etc.).
- the process of disconnection will start at the end (normal or abnormal) of the TPR.

If there is a message, it is not sent to the correspondent. A DISCNCT transaction may be activated at TDS restart if the DISCNCT transaction could not take place because of a TDS shutdown, or system failure.

If the disconnection is related to an XCP1 session, the LAST-TPRNAME field of the TDS-STORAGE may not be significant (in particular for a session allocated by a TM or DUMMY correspondent).

The following programming rules apply to the DISCNCT transaction:

- The commitment unit which is currently active, if any, is rolled back to its initial state. The TRANSACTION-STORAGE and PRIVATE-STORAGE of the disconnected user are passed to the DISCNCT transaction and the DISCNCT transaction is executed.
- To identify the terminal, the first DISCNCT TPR can perform a RECEIVE NO DATA. No message must be sent to the terminal, because it is disconnected.
- The first DISCNCT TPR may update controlled files and chain to subsequent TPRs.
- All the TDS-STORAGE fields except PRIOR-TPR are initialized as for the first TPR of any transaction. The PRIOR-TPR field remains as it was before the DISCNCT transaction was activated.
- The TRANSACTION-STORAGE is copied from the interrupted transaction on a length equal to the TRANSACTION-STORAGE size of the DISCNCT transaction.

  If the user session was idle at disconnection time, the TRANSACTION-STORAGE of DISCNCT transaction is initialized with "0".

  The programmer can use a call "GETTPRPAR" verb to get information about the last TPR committed (if any) and the last TM or DUMMY correspondent using the XCP1 session before the disconnection occurs. Refer to the description of the call GETTPRPAR in chapter 4.

12.4 LOGON Transaction

The LOGON transaction is activated each time a correspondent connects successfully to TDS, whether TDS requests the connection or not. Recall that there are two ways of logging on to a TDS application:

1. The terminal operator enters $*$CN tdsname ...
2. Once the terminal operator powers on the terminal, TDS connects the correspondent (see Chapter 2).

When a terminal (T1) is generated with slaves, this terminal cannot be connected until all the slaves are connected. The LOGON transaction is started for the terminal (T1) when all slaves are connected.

The Transaction-Storage description must have the following format:

```
01  TRANSACTION-STORAGE.
    02 USER-ID          PIC X(12).
    02 PROJECT          PIC X(12).
    02 BILLING          PIC X(12).
    02 TERMINAL-ID      PIC X(24).
    02 TERMINAL-TYPE    PIC X(8).
    02 TERMINAL-MODE    PIC X.
    02 CONTEXT-FLAG     PIC X.
    02 AUTHORITY-CODES  PIC X(32).
    02 FILLER           PIC X(1).
    02 SPAWNCOUNT       COMP-1.
    02 USED-BY-TDS      PIC X(12).
    02 FILLER           PIC X(n).
```

When the first TPR is activated, the above fields in the TRANSACTION-STORAGE area are initialized by TDS as follows:

- The USER-ID, PROJECT, and BILLING fields will contain the user-id, project and billing supplied by the user during logging-on. However, the length of the TDS user name is limited to 8 characters.
- The TERMINAL-ID field will contain the terminal identification supplied by the terminal, CRNETGEN, or terminal controller.
- The TERMINAL-TYPE field will contain the terminal model identification (for example, DKU 7211).
- The TERMINAL-MODE flag contains one of the following values.
  - “A”: When the terminal is active.
  - “P”: When the terminal is passive.

- The CONTEXT-FLAG is used to indicate whether a context already exists for the incoming user. The flag is set to " " (space) when there is no context; if a context exists from the previous session the flag is set to "C" if no transaction was in progress or "F" if a transaction is to be resumed.
  The RESTART transaction is explained later in this chapter.
- AUTHORITY-CODES is the list of authority codes of the user-id just logged. It represents a 32-character string of "0" or "1" digits.
- SPAWNCOUNT is used to indicate whether spawned transactions are waiting for this correspondent (SPAWNCOUNT = 1), or not (SPAWNCOUNT = 0).
- To send data, the first TPR must perform a RECEIVE NO DATA before calling SEND.
- The USED-BY-TDS field is reserved for internal use and not available for the users.

The first LOGON TPR may call other TPRs in order to do the following:

- reject the connection,
- accept the connection and resume the interrupted transaction (if any, this happens only in the case of relog-on),
- accept the connection and reject the interrupted transaction (if any, for a relog-on).

  The connection can be rejected by chaining to the standard TDS "BYE" transaction. Note that:

  – if no previous context exists (first log-on), the LOGOUT transaction will be activated,
  – if a previous context exists (relog-on), the DISCNCT transaction will be activated.

    The connection can be accepted and the interrupted transaction, if any, is resumed by terminating the LOGON TPR (or a TPR called by this TPR) with spaces in the NEXT-TPR field.

The connection can be accepted and the interrupted transaction rejected by chaining to the standard TDS TPR "CANCELTX". The "CANCELTX" TPR will:

- terminate the logon sequence,
- abort the interrupted transaction (all of them if there are several); the ON-ABORT-TPR of the aborted transaction, if any, is activated (see Chapter 2),
- put the terminal in "command" mode.

The RESTART TPR will not be entered.

See Figure 12-2.

NOTE:

The logon process can manage the "symbdest" (symbolic destination) parameter. This allows an interrupted transaction, when it resumes, to issue a send statement with EGI even if the end user has changed terminals. The transaction stores the destination in private storage and reads it when the transaction does not contain an exchange.

Handling of TDS service message header and trailer

Syntax

```
CALL "SETMGPRES" USING function,
                       presentation-length,
                       presentation-value,
                       status.
```

Description

This function is aimed to override either the TDS service message header and trailer, or the TDS transaction message header and trailer. The default presentation of these service messages is defined either by TDSGEN through the SERVICE-MESSAGE and TDSTX-MESSAGE statements, or during the logon sequence by explicitly entering the hexadecimal value. Note that only the service message header can be redefined during the logon sequence. Refer to the TDS ADMINISTRATOR GUIDE for a complete description of this service.

The "SETMGPRES" TDS function is issued to set up dynamically an appropriate presentation for the TDS service and transaction message to match the specific terminal type of the requestor. Typically this function should be used during the LOGON TPR by taking into account the TERMINAL-TYPE which is provided in the transaction storage. The presentation values are retained until the requestor is logged out, or another call is performed.

Parameter Description

- function is a COMP-1 (FIXED BIN(15)) to specify whether the TDS service message or transaction message header or trailer is to be set up. It must take one of the 4 values:

1 =	the TDS service message header is to be set up
2 =	the TDS service message trailer is to be set up
3 =	the TDS transaction message header is to be set up
4 =	the TDS transaction message trailer is to be set up

- presentation-length is a COMP-1 (FIXED BIN(15)) which must contain the length of the header or trailer which are passed as the third parameter. It is the 'presentation-value' size. It must be comprised between 0 and 16, and be an even number.
- presentation-value is the value of the header or trailer. It must contain up to 16 hexadecimal characters the size of which must be provided in the previous parameter.
- code is a 1 numeric character that contains the result of the function. The following values may be returned:

0 =	successful completion
1 =	erroneous function code out of the range [1,4]
2 =	erroneous length, out of the range [0,16], or odd number
3 =	erroneous value which is not an hexadecimal value

EXAMPLE

You can modify your LOGON TPR as follows:

```
working-storage section.
77 p-func comp-1.
77 p-length comp-1.
77 p-value pic x(16).
77 p-code pic x.

procedure division.

if terminal-type = "DKU7211"
   move 1 to p-func
   move 4 to p-length
   move "0d25" to p-value.
   call "setmgpres" using p-func p-length p-value p-code.
   if p-code not = "0" display "setmgpres erreur: " p-code
   upon alternate console.
```
❑

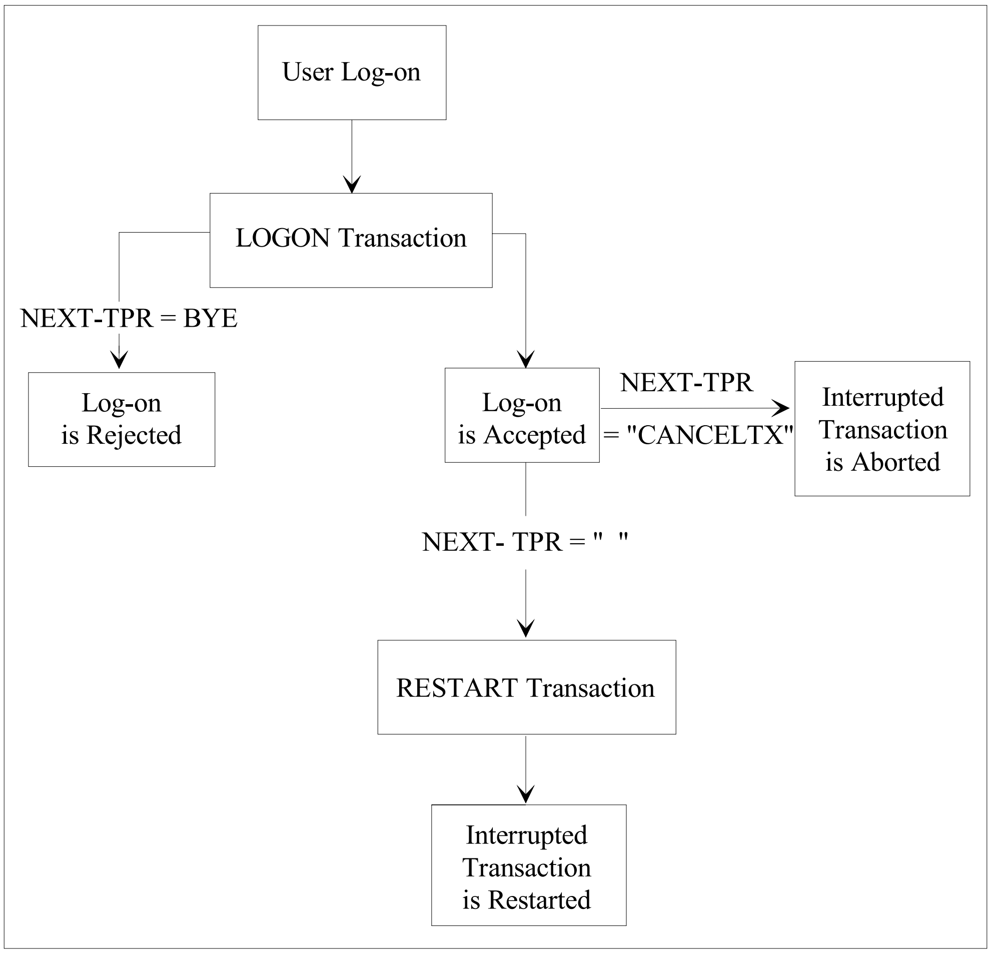


Figure 12-2. LOGON Transaction

Form Handling

Like any transaction, the LOGON Transaction can set the terminal to formatted mode. At the end of the LOGON transaction, the current form is kept if the NO IMPLICIT RELEASE option has been specified at TDSGEN.

Note for XCP2 users

You **must** declare the LOGON transaction in the STDS in order to define USE XCP2 in the corresponding MESSAGE clause.

12.5 LOGOUT Transaction

The LOGOUT transaction is activated after the BYE transaction has executed. The standard accounting message is sent to the terminal before the LOGOUT transaction is called. In order to send a message to the terminal, the LOGOUT transaction must first perform a RECEIVE NO DATA.

For an explanation of the BYE transaction, see Chapter 14.

12.6 RESTART Transaction

The RESTART transaction is activated when a user logs on to TDS and a context already exists for the user and a transaction is to be resumed. In this case, CONTEXT-FLAG is set to "F" (the user has been logged off accidentally or there has been a TDS failure). TDS-STORAGE is initialized as for the first TPR of any transaction, but the PRIOR-TPR and NEXT-TPR fields remain as they were before the RESTART transaction was activated. The transaction-storage is copied from the interrupted transaction. The RESTART transaction can be used to reinitialize some terminal functions not performed by TDS.

The interrupted TPR (if any) is resumed when the RESTART Transaction terminates (with NEXT-TPR set to spaces).

Note that, if the `NEXT-TPR = CANCELTX` in the LOGON transaction, the RESTART transaction will not be executed.

12.7 SHUTDOWN Transaction

The SHUTDOWN transaction is activated when TDS terminates normally or abnormally (through the `[ M ] TERMINATE_TDS STRONG = 1 command)`. The SHUTDOWN transaction is the last transaction to be executed in a TDS session.
When SHUTDOWN has started, all users except the master terminal, are logged off and all user transactions are either completed or interrupted. The SHUTDOWN transaction is defined in TDSGEN and there are no restrictions on its execution, that is, SHUTDOWN can chain to other TPRs and can dialog with the master terminal. Commitments are allowed and files can be modified.

The Transaction-Storage description must have the following format:

```
01  TRANSACTION-STORAGE.
    02  SHUTDOWN-FLAG     PIC X.
    02  FILLER            PIC X(n).
```

'n' depends on the value defined in TDSGEN (See the *TDS Administrator's Guide*, "TRANSACTION-STORAGE").

When the transaction is activated, SHUTDOWN-FLAG indicates whether a `[ M ] TERMINATE_TDS, or [ M ] TERMINATE_TDS STRONG = 1` command was used:

" " (space)	Normal TDS termination (`[ M ] TERMINATE_TDS`), user transactions terminate normally,
"S"	Fast TDS shutdown (`[ M ] TERMINATE_TDS STRONG = 1`); termination of user transactions is forced.

The first SHUTDOWN TPR either may terminate without calling another TPR or may call another TPR that may hold a conversation with the master terminal. In the second case shutdown becomes complete when `NEXT-TPR = space` is specified.

High Availability (HA) Only

When TDS is working with the High Availability (HA) feature, the transaction storage in the SHUTDOWN transaction has the following format:

```
01  TRANSACTION-STORAGE
      02 SHUTDOWN-FLAG           PIC X.
      02 MSTPRESENT-FLAG         PIC X.
      02 HAINITROLE-FLAG         PIC X.
      02 FILLER                  PIC X(N).
```

If the TDS master is connected, TDS sets MSTPRESENT-FLAG to the value one, and the SHUTDOWN transaction can dialog with the master. If the TDS master is not connected, TDS sets MSTPRESENT-FLAG to the value two, and the SHUTDOWN transaction itself cannot dialog with the master. TDS creates a special dummy session to perform the shutdown sequence. If the SHUTDOWN transaction attempts a dialog, it aborts. When the SHUTDOWN transaction is activated, TDS sets the value of HAINITROLE-FLAG to A, B or N (see the explanation of the STARTUP transaction).

NOTE:

A disconnection of the master TDS (i.e. which is not the special dummy session) occurring during the SHUTDOWN transaction prevents TDS termination. If a TAKEOVER is in progress, it cannot be completed. The reconnection of the master TDS is mandatory to resume the SHUTDOWN transaction in order to terminate the TDS session.

To identify such a situation, in order to complete the shutdown processing:

- A cobol display upon [alternate] console can be used in the SHUTDOWN tpr to inform that the SHUTDOWN transaction is running for the master TDS (testing MSTPRESENT-FLAG set to one).
- A cobol display upon [alternate] console can be used in the DISCONNECT tpr to warn that the master's reconnection will be mandatory to complete the SHUTDOWN transaction.

For more information about the HA product, see the *High Availability Concepts* manual and *High Availability Administrator's Guide*.

12.8 STARTUP Transaction

The STARTUP transaction is activated when the TDS session has successfully started (cold startup) or restarted (warm restart). The sequence of events is as follows:

1. A TDS application is restarted (warm or cold).
2. User files are opened.
3. The STARTUP transaction is executed.
4. Users can log on.

TRANSACTION-STORAGE must have the following format:

```
01   TRANSACTION-STORAGE.
     02  CONTEXT-FLAG       PIC X.
     02  ASSIGN-FLAG        PIC X.
     02  ANEW-FLAG          PIC X.
     02  XCPCONTEXT-FLAG    PIC X.
     02  FILLER             PIC X(m).
```

When the TPR is activated, the above flags in the TRANSACTION-STORAGE are set by TDS as follows:

CONTEXT-FLAG	space	There is no frozen terminal user. This happens in one of the following cases: a "cold" restart ([M] TERMINATE_TDS, or `[ M ] MODIFY_TDS_RESTART_OPTION` executed during the previous session), First TDS execution after a TDSGEN, Previous TDS session normally completed (all users logged off with a BYE command or were cancelled).

	"F"	All known users whether they are connected or frozen when the previous TDS session ended and not reconnected at restart time are in the frozen state. For further details, refer to the description of the CONNECT parameter in: either the `[ M ] TERMINATE_TDS` command if the previous TDS session terminated normally or the `[M] MODIFY_TDS_RESTART_OPTION` command if the last TDS session terminated abnormally as a result of: (`[M] TERMINATE_TDS STRONG=1, CJ,` a TDS abort).

Note that dummy correspondents and XCP correspondents are not taken into account for CONTEXT-FLAG.

ASSIGN-FLAG	S	The files opened are those assigned to TDS by the JCL. This value is set in one of the following cases: a "cold" restart (`[M] TERMINATE_TDS MODE=COLD, [M] MODIFY_TDS_RESTART_OPTION MODE=COLD`), first TDS execution after a TDSGEN.
	P	The files opened are those which were used in the previous TDS session. Any file declared in the JCL (static assignment) which is different from those used in the previous session is reported in the TDS JOR. In this case the file used for the previous session is used instead of that declared by the static assignment. Table 12-1 summarizes which files are assigned to which external file name.
ANEW-FLAG	A	Always set regardless of the TDS STARTUP.

XCPCONTEXT-FLAG	space	There is no frozen XCP correspondent.
	F	There are frozen XCP correspondents. The last TDS session terminated abnormally while the transactions using the XCP protocol were being executed.

The STARTUP transaction can be composed from one or several TPRs.

TPRs of STARTUP transaction can do any of the following :

- do a send operation to the master terminal only.
- hold a conversation with the master terminal.
- spawn a transaction to the master terminal (connections to other terminals are not yet allowed).
- spawn a transaction to a dummy correspondent (the spawned transaction will start at the end of the STARTUP transaction).
- read and write files.
- chain to another TPR.

Table 12-1. File Assignments on ASSIGN-FLAG = P

Previous Session ASSIGN-FLAG = P	JCL Static Assignment	Result on Restart
ifn1, efn1	ASSIGN ifn1, efn1	ifn1, efn1 no mismatch
ifn1, efn2	Assign ifn1, efn1	ifn1, efn2 mismatch reported in TDS JOR

If there is a mismatch in file assignment between the previous session and the job description, the TDS warm restart does the following:

- ignores the assignment in the job description,
- uses the previous session values,
- inserts a message in the TDS JOR giving the files used.

Table 12-2. CONTEXT-FLAG and ASSIGN-FLAG Values

FLAG	COLD	WARM Last Session Normal End	WARM last Session Abrupt or Abnormal End
CONTEXT-FLAG	space	F or space	F
ASSIGN-FLAG	S	P	P

Table 12-2 summarizes the values that CONTEXT-FLAG and ASSIGN-FLAG may have for a COLD or WARM startup.

12.8.1 High Availability (HA) Only

When TDS is working with the High Availability (HA) feature, the transaction storage in the STARTUP transaction has the following format:

```
01  TRANSACTION-STORAGE
      02 CONTEXT-FLAG            PIC X.
      02 ASSIGN-FLAG             PIC X.
      02 ANEW-FLAG               PIC X.
      02 XCPCONTEXT-FLAG         PIC X.
      02 MSTPRESENT-FLAG         PIC X.
      02 HAINITROLE-FLAG         PIC X.
      02 FILLER                  PIC X(m).
```

If the TDS master is connected, TDS sets MSTPRESENT-FLAG to the value one, and the STARTUP transaction can dialog with the master.

If the TDS master is not connected, TDS sets MSTPRESENT-FLAG to the value two, and the STARTUP transaction itself cannot dialog with the master. TDS creates a special dummy session to perform the startup sequence. If the STARTUP transaction attempts a dialog, it aborts.

When the STARTUP transaction is activated, TDS sets the value of HAINITROLE-FLAG as follows:

A	If the TDS was initially started in Active mode,
B	If the TDS was initially started in Backup mode and has become Active after a Takeover,
N	if the TDS is not managed by HA.

For more information about the HA product, see the *High Availability Concepts* manual and *High Availability Administrator's Guide*.

12.8.2 H_REINIT Transaction

After a restartable abort, the H_REINIT transaction is executed. H_REINIT can be seen as a special STARTUP transaction executed as part of a RESTARTABLE ABORT sequence. It is executed after the recovery phase and before Commits are restarted.

For example, you may use H_REINIT to open UFAS files not declared in the TDS.

To avail of this facility, you must declare it in the TDS generation as follows:

```
MESSAGE "H_REINIT" ASSIGN TO user-tpr-name
                   AUTHORITY-CODES 0,1      (for example)
                    TRANSACTION-STORAGE SIZE x.
```

This transaction is executed in a temporary session so no SEND can be performed.
No specific Transaction Storage is provided by TDS.
TPRs can read and/or write files, chain to other TPRs, and spawn on known correspondents (connections are not yet allowed).

If the level of current simultaneity is not at least 2, or if there is a lack of space in the TDS Tables or in the SWAP file, a fatal abort of TDS (with message MV34) occurs.

12.9 H_XAEVT Transaction

This transaction is useful only for a TDS using XA commitment protocol with ORACLE7.

A basic transaction called H_XAEVT is provided by ORACLE7, see *ORACLE7/TDS User's Guide*. It may be customized so that events that could lead to data inconsistency can be handled. Although using this transaction is not mandatory, note that the default action is limited to sending a message to the master to display information as such contained in the message storage described further, the message is also written in the JOR of the TDS, see *TDS Administrator's Guide*.

The H_XAEVT transaction is defined by assigning a TPR to the message-id H_XAEVT in the Transaction Section of the STDS subfile for TP7GEN.

The transaction is started twice by TDS each time a commitment unit resynchronization occurs: as soon as desynchronization is detected and after its resynchronization completion.

The H_XAEVT transaction runs for a temporary correspondent, that is why no message can be sent to him.

The first TPR may have any identification. It receives the message as follows:

```
01 XAEVT-MSG.
   02 TPR-NAME                PIC X (12).
   02 USER-NAME               PIC X (12).
   02 OCCUR-NB                COMP-1.
   02 TDS-XA-STATUS           COMP-1.
      88 ROLLBACK             VALUE    0.
      88 COMMIT               VALUE    1.
   02 XA-GLOBAL-STATUS        COMP-1.
      88 DONE                 VALUE    0.
      88 RETRY                VALUE    4.
      88 HEURISTIC-MIXED      VALUE    5.
      88 HEURISTIC-ROLLBACK   VALUE    6.
      88 HEURISTIC-COMMIT     VALUE    7.
      88 HEURISTIC-HAZARD     VALUE    8.
      88 RMERR                VALUE   -3.
      88 NOTA                 VALUE   -4.
      88 RMFAIL               VALUE   -7.
```

When the transaction is activated, the above information is set by the TDS as follows:

TPR_NAME

Is the current TPR of the commitment unit that has been resynchronized.

USER_NAME

Is the correspondent identifier, for which the TPR was running.

OCCUR_NB

Can have the value 1 or 2. If 1, the transaction was started after detection of desynchronization. If 2, it was started after completion of resynchronization.

TDS_XA_STATUS

Indicates the completion status of non-XA Resource Managers (for example UFAS files, IDS2 databases) on the TDS side. This information may help the database administrator to take a heuristic decision.

XA-GLOBAL-STATUS

Is the status returned by ORACLE7/TDS that involves resynchronization of the commitment unit or last returned one, when the transaction is launched after resynchronization completion. In this latter case, last returned one means on last commit or rollback request.

DONE	the XA Resource Managers (ORACLE databases) successfully completed the same commitment action (EITHER Commit or Rollback, depending on TDS_XA_STATUS).
RETRY	the XA Resource Managers are not able to commit at this time, TDS will re-issue a *commit* request later.
HEURISTIC COMMIT	due to a heuristic decision, the work done was committed.
HEURISTIC ROLLBACK	due to a heuristic decision, the work done was rolled back.
HEURISTIC MIXED	due to a heuristic decision, the work done (on XA and non XA Resource Manager) was partly committed and partly rolled back.
HEURISTIC HAZARD	due to some failure, the work done may have been heuristically completed.
RMERR	when a commit request was issued: an error occurred in committing the work performed and this work has been rolled back. When a roll back request was issued: an error occurred in rolling back the work performed.
RMFAIL	an error has resulted in one or more Resource Managers being unavailable.
NOTA	a commit request was issued and the work done rolled back. The transaction identifier is no more known by concerned Resource Managers.

12.10 Transaction Initialization Routine

Transaction Initialization Routine is a user-written subprogram. This subprogram is called by TDS when a message is received either from a terminal that has no transaction active (in "command" mode) or as a first message whose purpose is to spawn a transaction. The subprogram must return a transaction identifier to TDS. The existence of such a subprogram is indicated by the `USE Procedure name FOR TRANSACTION INITIALIZATION` statement in the TDS SECTION of the TDSGEN.

The following programming rules apply to the Transaction Initialization Routine.

- The routine must not access any file, whether TDS controlled or non controlled, nor call any TDS function (for example, SEND, RECEIVE, or CALL statements).
- The routine may call other subprograms; however, these subprograms must neither access any file nor call any TDS function.
- The Working-Storage area of the routine is not refreshed at each call. If the Working-Storage area must contain constant value data items, these data items should remain unchanged throughout the execution of the routine or should be initialized before the routine exits.
- The PROCEDURE DIVISION statement must take the following form:

```
PROCEDURE DIVISION USING INPUT-TEXT, TEXT-LENGTH, TX-TYPE,
                                     TERMINAL-INFO.
```

- The LINKAGE SECTION must contain the following data items:

```
01  INPUT-TEXT.
    02  INPUT-TEXT-1 PIC X(maximum-message-length).
77  TEXT-LENGTH COMP-1.
77  TX-TYPE PIC X(8).
01  TERMINAL-INFO.
    02  SYMBOLIC-SOURCE  PIC X(12).
    02  TERMINAL-TYPE    PIC X(8).
    02  USERID           PIC X(12).
    02  TERM-MODE        PIC X(1).
    02  DVCHDR-LG        COMP-1.
```

Table 12-3 summarizes these parameters. The x indicates the type of parameter. All these parameters are input parameters except TX-TYPE which is both an input and an output parameter.

Table 12-3. LINKAGE SECTION Parameters for the Transaction Initialization Routine

	Input Parameter	Output Parameter
INPUT-TEXT	x	
TEXT-LENGTH	x	
TX-TYPE	x	x
TERMINAL-INFO	x	
SYMBOLIC-SOURCE	x	
TERMINAL-TYPE	x	
USERID	x	
TERM-MODE	x	
DVCHDR-LG	x	

INPUT-TEXT	contains the message to be decoded. In the case of spawning, the message text of the spawn command is used as input text.
TEXT-LENGTH	contains the length of the received text and includes the device header when the TERM-MODE field is set to "U".
TX-TYPE	contains the transaction identifier (left justified and, if necessary, padded with spaces on the right). The transaction initialization routine should return the transaction identifier in TX-TYPE.
TERMINAL-INFO	contains information about the terminal that may be needed by the transaction initialization routine when decoding the message.
SYMBOLIC-SOURCE	is the name of the correspondent.
TERMINAL-TYPE	is the terminal model, for example, DKU7211.
USERID	contains the name of the user who activated the transaction.

TERM-MODE	contains the type of connection mode. By default, a terminal operates in edited mode, that is, TERM-MODE = E. Otherwise a terminal can operate in unedited mode, that is TERM-MODE = U. In edited mode, control information such as the device header is not actually displayed for a message being prepared for transmission, whereas in unedited mode all such control information is displayed.
DVCHDR-LG	contains the length of the device header and applies only when TERM-MODE is set to U.

The transaction initialization routine must not modify any of the fields except the `TX-TYPE` field. `TX-TYPE` is set to spaces by TDS each time the routine is called. If the value of `TX-TYPE` is not changed during the routine, that is, it remains spaces, the transaction identifier retained by TDS is contained in the field starting at the first character of the message and bounded by:

- either the first space in the message,
- or the first 8 characters of the input text.

Note, if the transaction uses FORMS, the sequence of operations is as follows:

- the Transaction Initialization Routine receives the text as it came from the terminal,
- the Transaction Initialization Routine returns the transaction identifier,
- the first TPR of the transaction receives the message after being edited by FORMS.

Thus the text received by the TPR is that received from the terminal, possibly edited by FORMS. This message may or may not contain the transaction identifier.

❑

13. Implementing the Transaction

13.1 Creating the Application

Before a TDS application can be run, a certain number of tasks must be performed. TPRs must be compiled and linked. This involves the standard JCL set up described in this chapter. The network must be configured and defined to allow terminals and application transaction programs to communicate with one another. This entails running the Network Generation (CRNETGEN) utility.

The user must be declared in the system; see the *Catalog Management User's Guide*. TPR sharable modules must be loaded from the appropriate sharable module library into backing store from where segments can be swapped in and out of main memory and shared by all active transactions. At some stage (either before or after TDS is launched), the operator must initiate telecommunications in order to provide TDS with access to the terminal network.

The following pages describe how to compile and link a TPR. The other topics, which are dealt with by the System Administrator, are treated in the *TDS Administrator's Guide*.

Later sections show you how to improve a transaction's performance, and how to test and debug an application.

13.1.1 Compiling a Program

A TPR is compiled as a standard COBOL program. (See the *COBOL 85 User's Guide*). The user need specify only the input source library tdsname.COBOL.

The TPR is compiled through the following job description:

```
$JOB job-name, USER=userid [,PROJECT=project [,BILLING=billing]];

LIB SL INLIB1 = (user-source-name [,DVC = device-class,
                                   MD = volume-name]),

      [INLIB2 = (tdsname.COBOL [,DVC = device-class,
                                 MD = volume-name]),]

      [INLIB3 = (H_CBLIB [,DVC=device-class, MD=volume-name])];


COBOL SOURCE=tpr-name,

      [CODAPND] [,DSEGMAX = integer-1] [,PSEGMAX = integer-2],...

      CULIB = (user-culibrary-name, DVC = device-class,
                                    MD = volume-name);

$ENDJOB;
```

You can execute this job interactively through IOF.

The description of parameters is as follows:

- tdsname.COBOL allows COPY statements tdsname.COBOL contains:
  - all the control and file description entries defined at TDSGEN, for example, common data definitions declared for WORKING-STORAGE, COMMON-STORAGE and TRANSACTION-STORAGE.
  - storages to be shared, namely, TDS-STORAGE and CONSTANT-STORAGE.
- The TPR issues COPY statements to retrieve these file and storage entries.
- The H_CBLIB file must be specified to allow COPY statements to be used for FORMS.

- The CODAPND parameter is recommended because it appends the code segment to the linkage segment. This reduces the number of entries in the SM (sharable module) and the amount of disk I/Os required.
  TPR performance can be improved by specifying CODAPND and adjusting PSEGMAX and DSEGMAX in the COBOL statement in order to control the creation of segments in the compile unit. The use of CODAPND, PSEGMAX, and DSEGMAX is explained later in this chapter.
- The CULIB parameter specifies the library in which the resulting compile unit is to be stored.
- If a TPR is to be run with PCF commands, it must be compiled with DEBUG.

NOTE:

All files declared in a TPR compiled with COBOL 85 must also have been declared in the STDS (see the *TDS Administrator's Guide*), otherwise the TPR may abort at execution time.

13.1.2 Linking a Program

The sharable module library must have been allocated and initialized before linkage. This may be done through the `TP7PREP` utility that the TDS Administrator uses to allocate TDS system files. One potential source of confusion for new TDS programmers is one group of these system files that are called the off-line files even though they are on-line. They are called off-line because they are used only in the preparation and generation of TDS.

The TPR is linked through the following job description:

```
$JOB job-name, USER = userid [,PROJECT=project [,BILLING=billing]];
LIB CU, INLIB1 = (user-culibrary-name [,DVC = device-class,
                                        MD = volume-name]);

LINKER tpr-name, SM,
  OUTLIB = (tdsname.SMLIB [,DVC = off-line-device-class,
                            MD = off-line-volume-name]),
  COMFILE = (tdsname.SLLIB [,DVC = off-line-device-class,
                             MD = off-line-volume-name],

SUBFILE = TP7LINKTPRi);

$ENDJOB;
```

This job can be performed interactively through IOF as follows:

- The JCL statement LIB CU sets up a search path for the LINKER utility so that it can find the appropriate compile-unit library or libraries.
- `tpr-name` is the PROGRAM-ID name specified in the IDENTIFICATION DIVISION of the TPR.
- `tdsname.SMLIB` is the sharable module library in which the TPR being linked is to be stored. This SM library must have been previously allocated and initialized as part of the group of files that are called the off-line files.
- `tdsname.SLLIB` is the source library containing TP7LINKTPR created at TDSGEN. There are as many TP7LINKTPRs as there are TPR-sharable-modules declared at TDSGEN. The program is linked to the SM named TPR[i].
- off-line and volume name as defined at TP7PREP time.

A TPR can be run only if it has been correctly linked.

13.1.3 Linking a Batch Interface Program

The batch program is linked through the following job description:

```
$JOB job-name, USER = userid [,PROJECT = project [,BILLING = billing]];

LIB CU, INLIB1 = (library-name [,DVC = device-class, MD = volume-name]);

LINKER load-module-name, OUTLIB = (load-module-library
       [,DVC = device-class, MD = volume-name]), COMFILE = *ien;

$INPUT ien;
       ENTRY = entry-point-name,
       LINKTYPE = (DACM),
       STARTASG = (PRIVATE = 17);

$ENDINPUT;

$ENDJOB;
```

The description of parameters is as follows:

- load-module-name is the name of the batch program.
- LINKTYPE and STARTASG commands are specified as shown. Note that from the technical status TS 6152, STARTASG is no longer needed.
- Other linker clauses can be included to meet user requirements.

The batch interface program is executed like any $STEP statement.

EXAMPLE

```
STEP load-module-name, FILE = (load-module-library,
    DVC = device-class, MD = volume-name), DUMP = DATA;
```
❑

Figure 13-1 summarizes what compilation and linking of TPRs involve.

The COBOL compiler compiles the source code, making references where necessary to the COBOL source library (<tdsname>.COBOL) that is produced at TDSGEN. TPRs retrieve information from <tdsname>.COBOL through COPY statements. The object code is then output to the compile unit library (user-culibrary-name). Each TPR is then linked into a sharable module in a predetermined sharable module library (<tdsname>.SMLIB). Now that a fully executable sharable module has been created, it can be fetched and loaded into main store for execution.

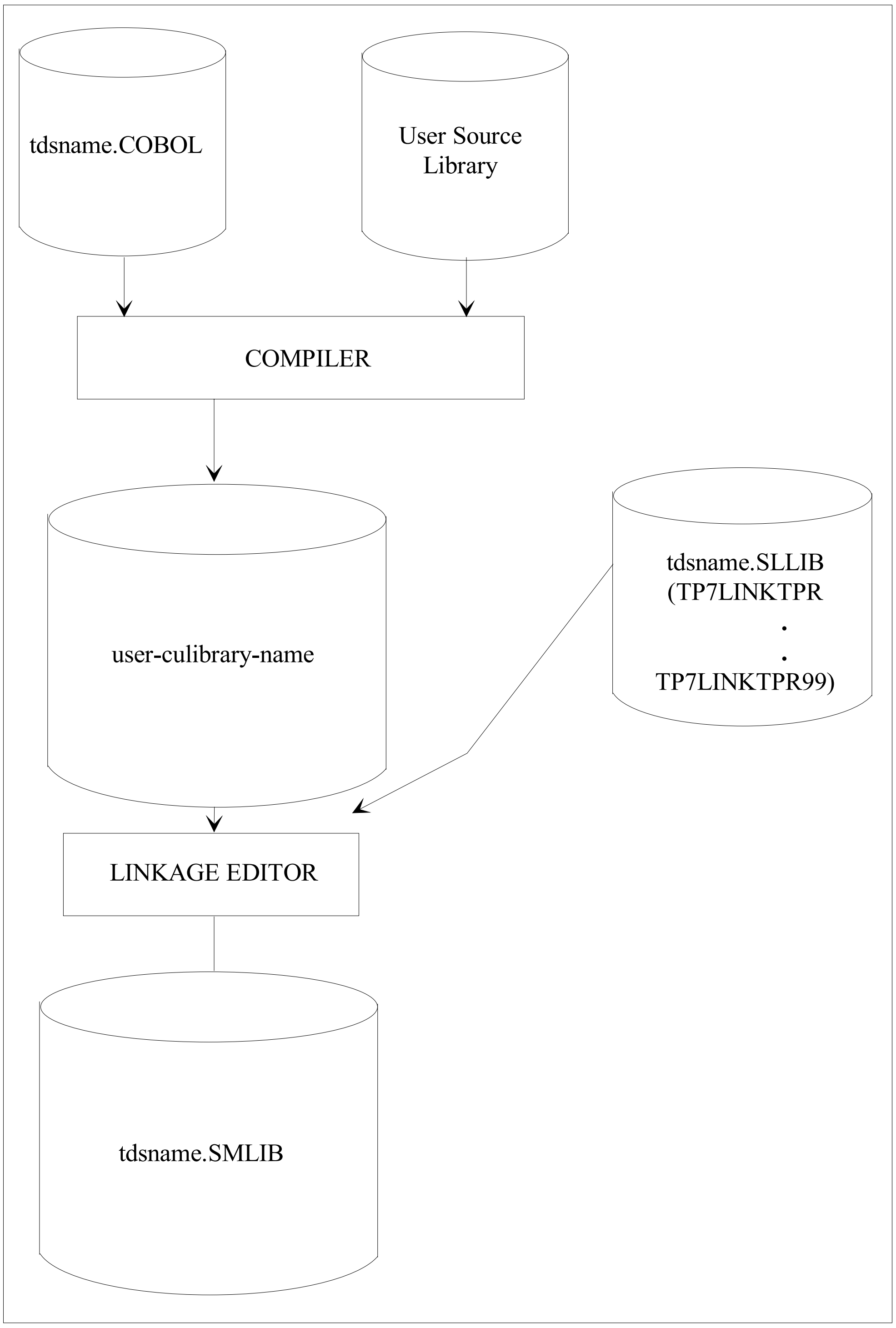


Figure 13-1. Preparing TPRs for Execution

13.1.4 Compiling/Linking Queries

These tasks are described in the *IQS/TDS User's Guide*.

13.1.5 Deleting a TPR

The following job description deletes a TPR from a sharable module:

```
$JOB job-name, USER = userid [,PROJECT = project [,BILLING = billing]];

LIBMAINT SM LIB = (tdsname.SMLIB [,DVC=offline-device-class,
                                            MD=offline-volume-name]),

                 COMFILE= *ien;

$INPUT ien;

   DELETE sm-name, LKU = tpr-name;

$ENDINPUT;

$ENDJOB;
```

The description of parameters is as follows:

- `tpr-name` is the PROGRAM-ID name specified in the IDENTIFICATION DIVISION of the TPR.
- `sm-name` is the name of the TPR-sharable-module containing `tpr-name`.
- Offline device and volume-name are defined at `TP7PREP` time.

13.2 Tuning the Application

TPRs are executed within a unique environment as defined at TDSGEN. Although a TPR is a single entity, many transactions are processed simultaneously. Therefore, TPR design and the programming techniques adopted must be consistent throughout and oriented towards a single efficient system.

For a TPR to operate effectively within its environment, consider the following suggestions:

13.2.1 Eliminating Segment Faults

- Use the CODAPND option. A TPR is most efficient as a single code segment because the time taken to search for and load the program from disks is reduced. Use of the CODAPND parameter will ensure that the generated code segment will be placed in the same segment as the linkage segment.
- In the main code segment, try to group procedures that are frequently used. This reduces the number of I/O operations and saves memory. Therefore place all other procedures in alternative code segments.
- Optimize data storage areas by selective grouping of like data and related records to reduce overall I/O activity. Use of the COBOL REDEFINES clause reduces working area size and therefore saves work space.
- Certain COBOL statements can be very costly in terms of generated code.

  Therefore it is important to optimize generated code size by minimizing the number of SEND, RECEIVE, STRING, UNSTRING, and INITIALIZE statements.

EXAMPLE

Instead of writing two SEND statements, use the PERFORM statement to branch to a single SEND.

Original Coding	**Alternative Coding**
SEND FROM A	PERFORM SEND-A
SEND FROM B	MOVE B TO A
	PERFORM SEND-A
	.
	.
	.
	SEND-A

❑

- Do not specify DEBUG for a transaction that is already debugged because the resultant compilation creates unnecessary data segments.
- Reduce the number of CALL statements. For a function containing only a few lines of code that is used by several TPRs, incorporate the function within each TPR by using the COBOL statement COPY.
- Limit and standardize code segment sizes. The size of code segments should be:
  - the same to even out "search/load" time
  - as small as possible to be conveniently swapped in.

  The TPR should be segmented according to its logical structure. This will help VMM to retain in memory the minimum number of segments, thereby reducing the swapping. The normal rate of missing segments is one per TPR. Details are given in the JOR (PROG MISSING SEGMENTS/PAGES) and also on the TDS execution report (NUMBER OF USED TPRs). A high rate of missing segments is due to either inadequate TDS memory or a programming error.

13.2.2 Using SEND and RECEIVE Statements

The following points are recommended:

- Avoid numerous RECEIVE statements. Input messages are oversliced if the receiving data item in the TPR is of inadequate size.
- Reduce the number of SEND with ESI statements and SEND statements without an indicator. This action reduces processing time and increases memory availability.
- Avoid the use of SEND with EMI statements followed by another SEND statement within the same TPR. TDS waits for the first transmission to complete before processing the second SEND statement. This results in slow throughput and hence increases response time for other terminals. This, however, does not apply to a TDS with a large number of simultaneities declared at TDSGEN.
- Avoid using SEND statements to terminals other than the originating terminal. Spawn the transaction in order to recover the message in the case of terminal failure.
- Use the information already in the COMMUNICATION SECTION. For example, RECEIVE gives DATE and TIME, therefore the ACCEPT (DATE and TIME) can be avoided.

- Avoid complicated message formats when not using FORMS. Well-spaced messages are easier to read. Limit the number of characters transmitted. Tabstops or addressable entry markers, when available on a terminal should be used instead of embedding control codes within the message. Minimize the length of the message to reduce buffer size and transmission time.
- Use FORMS for easy handling of screen displays and for minimizing working-storage area.

13.2.3 USING CALL Statements

Where possible, replace the CALL statement by a PERFORM statement which incorporates a COPY statement. However, the routine must be recompiled each time the TPR is modified.

13.2.3.1 Advantages

A routine to be called can be in a different sharable module, thereby avoiding recompilation each time.

Routines performing special functions can be written separately and merged later into a single sharable module. The USE clause integrates the routine in the generated TDS load module.

13.2.3.2 Disadvantages

A called routine needs a separate code segment. This leads to a greater demand on memory, an increase in execution time and a greater risk of missing segments.

Each code segment uses one entry in the sharable module, thereby reducing the number of entries available to TPRs.

13.2.4 Accessing Files and Databases

In order to obtain the optimum throughput and fast response times, note the following points:

- Reduce the number of file accesses to reduce CPU and elapsed times. The average I/O is approximately 30-50 milliseconds in elapsed time and 5-10 milliseconds of CPU time. The average number of I/Os per TPR can be calculated from the TDS statistics report and the JOR.
- Always try to commit the processing as soon as possible, thereby reducing the number of concurrent access conflicts. Avoid conversations within a commitment unit, as these will lock CIs for long periods.
- Use the SHARED READ or SUPPRESS CONCURRENT clauses if possible.
- Resolve any deadlock and concurrent access conflicts that appear in the TDS statistical report. Resulting aborts increase I/O activity, incur additional processing, increase response time, and involve additional operator intervention.
- Unnecessary protection for file integrity increases I/O overheads.
- Do not dynamically reorganize indexed files. Leave enough free space when creating the file.
- If the commitment unit includes a conversation, access as many required records as possible prior to the conversation to avoid having to repeat the conversation with the terminal operator in the case of deadlock. However, since this procedure locks pages, it should be used only when strictly necessary.
- Some records, such as inventory records and directories, cause frequent access conflicts as they are accessed by many transactions. Try to split these records up into several smaller records to be located in different CIs.

13.2.5 Designing Access to Resources

- Avoid locking COMMON-STORAGE because this will delay the processing of other transactions. Use SHARED-STORAGE wherever possible.
- Non-concurrency should be reduced. The TDS statistics report provides details of the number of conflicts caused by non-concurrency clauses. Certain master commands including [M] OPEN_TDS_FILE and [M] CLOSE_TDS_FILE require non-concurrency with all transactions. Avoid issuing these commands unnecessarily.
- Avoid specifying too many TPRs within the same exchange. Use one TPR per exchange. The overhead involved in starting one TPR is one swap I/O, one VMM I/O (segment loading) and one I/O at the end of each TPR if the TPRs are unmapped systematically.
- Avoid separating into different SMs, TPRs
  - either of the same transaction,
  - or of transactions likely to be concurrent.

13.2.6 Optimizing Program Coding

Structured programming and modular segmentation break the TPR into smaller, more manageable units or modules. These modules are linked by PERFORM statements. As shown in Figure 13-2, a hierarchy of PERFORMs executes modules of successively lower levels. Higher level modules determine the logic of the TPR, and the lower-level modules proceed to process the details.

The ideal TPR consists of a single segment containing the code and link segments. A multi-segment TPR, while having one segment executing, may have to wait for the next segment because in the meantime it has been swapped out or overlaid by another segment. If this is a large segment, considerable VMM activity is required to swap it in. The CODAPND parameter of the COBOL statement merges code and link segments when the size of the TPR is small.

Avoid GO TO verbs. They can occasionally be used for handling exceptions when they simplify the logic to improve overall performance. A well-designed TPR does not need GO TOs.

Paragraph and data names should be self-explanatory. An entry captioned PERFORM XYZ says little whereas PERFORM COMPUTE-RETAIL-PRICE is immediately obvious.

```
PROCEDURE DIVISION USING ...

TPR-START.
    PERFORM RECEIVE-MSG.
    IF NO-ERROR-INPUT
       IF PRIOR-TPR = SPACES
          PERFORM FIRST-TIME
       ELSE
          IF NBR-PASS = NBR-TIME
             PERFORM LAST-TIME
          ELSE
             PERFORM ONE-TIME
       ELSE
          CALL "ABORT".
       .
       .
       .
TPR-EXIT.
    EXIT PROGRAM.
```

Figure 13-2. Example of Structured Programming

13.2.6.1 Large TPRs

When a TPR is large, it should be segmented.

Two types of segmentation are available:

- Logical
- Physical

Logical segmentation should be attempted first as it gives much better performance.

LOGICAL SEGMENTATION

Group in the main code segment those operations occurring most frequently and routines for handling exception conditions and errors.

Segments are logically segmented by specifying segment numbers in the SECTION headers of the PROCEDURE DIVISION, for example:

```
section-name SECTION 22.
```

For each set of SECTIONs with the same segment number, the compiler generates object code in a single segment. One segment is generated for each segment number. Segment numbers 0 to 49 correspond to fixed segments, which are retained because they contain the results of ALTERs and PERFORMs.

Sections of coding which are logically related and which are normally executed together can, thus, be grouped into a single segment even if they are not physically contiguous in the PROCEDURE DIVISION.

PHYSICAL SEGMENTATION

Whether logical or not, all segments are subdivided by the compiler if their size exceeds the preferred size. The preferred size is by default 4 Kbytes and can be overwritten successfully as follows:

- in OBJECT-COMPUTER of the CONFIGURATION SECTION of the transaction's ENVIRONMENT DIVISION by:

  ```
  MAXIMUM PROCEDURE SEGMENT SIZE psegmax.

  MAXIMUM DATA SEGMENT SIZE dsegmax.
  ```

- and later in the COBOL statement at compilation time.
  - PSEGMAX
  - DSEGMAX.

For DSEGMAX, do not exceed 65,504 bytes.

For more information on PSEGMAX and DSEGMAX, see the *COBOL 85 Reference Manual*.

Segmentation can occur in the middle of a frequently used iterative sequence of code. If it does, performance is impaired due to unnecessary swapping and additional memory requirements. In this case, increase the segment size so that the logical code sequence is contained in a single segment.

13.2.6.2 Small TPRs

Several small TPRs constituting a transaction should be grouped in a single TPR.

The advantages are as follows:

- facilitates the Virtual Memory Management (VMM) in its "search/load",
- allows VMM to retain all these small TPRs in memory for a longer time, thus improving performance by reducing the number of swaps.

The only constraint in programming such a TPR is its complexity. For example, the TPR might start with a GO TO DEPENDING ON which then branches to various routines.

13.2.7 Message Handling

13.2.7.1 Form Display

Whenever a form is displayed on the screen, the next prompt should be sent by erasing the variable fields rather than sending the whole form again.

A short message could be sent at the same time to inform the operator if an entry keyed in is incorrect. An average form consists of between 500 and 1000 characters. Erasing the variable fields requires one character.

13.2.7.2 Positioning the Cursor

If more than three blanks appear in a form, it is more efficient to move the cursor to the corresponding position than adding spaces.

13.2.7.3 Data Transmission without Forms

Because more than one SEND with EMI statement gives rise to synchronous processing (the TPR must wait until the next SEND is allowed), avoid several SEND WITH EMI statements in a TPR.

SEND with EMI statements can be replaced by SEND with ESI statements terminated with a SEND with EMI or EGI. Instead of transmitting several short messages during the TPR, TDS will transmit one long message at the end of the TPR. Processing is therefore faster and more efficient. Consequently, the overall response time is much faster.

When a TPR contains several SEND statements, move the output messages to a single area and issue PERFORM on the paragraph containing the SEND statement. This reduces memory occupancy.

The SEND statement in the paragraph would be of the format:

```
SEND......WITH identifier.
```

where identifier is the name of a numeric data field containing:

```
1    for ESI
2    for EMI
3    for EGI
```

Any type of transaction may use the NO AUTOMATIC UNMAPPING option that is specified in the TRANSACTION SECTION at TDSGEN. This option prevents the process executive from being released at the end of a TPR. Therefore the transaction context need not be saved between TPRs. This results in reducing I/O overhead in a transaction composed of many TPRs.

The NO AUTOMATIC UNMAPPING option is not effective when a TPR:

- either terminates with:
  - a commitment, or
  - SEND with EGI statement,
- or specifies WAIT-TIME in TDS-STORAGE.

When the NO AUTOMATIC UNMAPPING option is specified, the TPR which issues a SEND with ESI or EMI remains waiting (idle) until the transfer to the network is completed. If the NO AUTOMATIC UNMAPPING option is omitted, and a single SEND with EMI statement has been issued, the TPR is unmapped on termination, thereby adding to CPU and I/O overhead. When several SEND with EMI statements are issued, they are all synchronous except the last SEND with EMI or EGI statement.

SEND statements toward either slave terminals or terminals other than the current one are also synchronous. Synchronous transmission reduces throughput and should therefore be avoided.

13.2.7.4 Data Transmission with Forms

Data sent to a terminal is handled by calls to FORMS procedures, described in Chapter 10.

FORMS assumes that messages can be recovered.

13.2.8 Memory Occupancy

To minimize the amount of memory that a TPR occupies:

- share code using the PERFORM statement,
- ensure that a COPY statement does not result in unnecessary data structure expansion; if so, remove unnecessary fields,
- avoid unnecessarily large working areas by using REDEFINES clauses.

NOTES:
TRANSACTION-STORAGE is swapped:

1. at each commitment,
2. at each intermediate clean-point on WAIT-TIME and SEND with EMI.

13.3 Testing and Debugging

13.3.1 The TRACE Command

The TRACE command is used to debug TPRs by tracing selected events. It produces output messages to a private SYSOUT file. Although the TRACE command selects the events to be traced, it cannot modify them. Only a subset of Program Checkout Facility (PCF) commands using symbolic addressing is allowed in TDS. This subset of PCF commands can be used if the TPR has been compiled with the DEBUG option.

TRACE must not be used when a form is active between two transactions, that is, when the NO IMPLICIT RELEASE option has been specified in the USE FORMS clause of the TDS SECTION at TDSGEN.

In release V6, you can use the extensions to the TRACE command to debug transactions involving a XCP1, or a XCP2 principal conversation. Now you can request a trace for both a particular transaction and a particular user. In other words, each occurrence of the transaction running for a specified user is traced.

Standard Syntax

```
                                            [DISP]
                                            [FLOW]
      { PRINT}                              [IDS2]
TRACE { SEND } [=terminal-name[ PRT]]       [MESG]
      { OFF  }                              [MREC]
                                            [RREC]
                                            [TRST]
                                            [PCF ]
                                            [XPCF]
                                            [XCP]
                                            [XCPM]
                                            [PT]
                                            [XA]
```

Extended Syntax 1

```
                                                              [DISP]
                                  {name-12}                   [FLOW]
TRACE PRINT [FROM] TX=name-8 USER={       } [TPR=name-12] [IDS2]
                                  {   *   }                   [MESG]
                                                              [MREC]
                                                              [RREC]
                                                              [TRST]
                                                              [PCF ]
                                                              [XCPM]
                                                              [XCP ]
                                                              [XPCF]
                                                              [PT]
                                                              [XA]
```

Extended Syntax 2

```
TRACE SEND [= terminal-name  [ PRT]] TX = name-8

             USER= {name-12}
                   {   *   }

             [ TPR=name-12 ]
             [ options-list as for Extended Syntax 1 ]
```

Extended Syntax 3

```
TRACE OFF { TX = { name-8 } }
          {      {   *    } }
```

Extended Syntax 4

```
TRACE LIST
```

Reserved for the Master Terminal Operator Only.

Extended Syntax 5

```
TRACE OFF TX = { name-8 }  [ OPER = { name-12 }]
               {   *    }  [        { *       }]
```

Extended Syntax 6

```
           [ TX = name-8  OPER = name-12  ]
TRACE LIST [ TX =   *     OPER = name-12  ]
           [ TX = name-8  OPER =    *     ]
           [ TX =   *     OPER =    *     ]
```

13.3.1.1 Description of the Standard TRACE Command

The TRACE PRINT option puts the terminal in trace mode and delivers the trace information to a subfile of the on-line DEBUGFILE file, from where it can be dumped to a printer. The message "TERMINAL IN TRACE MODE OUTPUT ON member-name" is received at the terminal.

When trace mode is terminated, the subfile is printed.

The TRACE SEND command puts the terminal in trace mode and prints trace information at the terminal. The message "TERMINAL IN TRACE MODE OUTPUT ON HARD COPY [terminal-name]" is sent to the terminal. Terminal-name can identify

- either a terminal other than the one used,
- or a pseudo-terminal.

You can specify if the terminal is a VIP printer, by adding the keyword PRT, preceded by a space, after the keyword =termnlname. This allows TDS to transmit a VIP device header with "status print" coding. You can use this keyword only if the device is really a VIP printer, and the destination TRACE SEND must not be already in use by any transaction.

When terminal-name is omitted, the output is sent to the terminal at which the command is keyed in.

NOTES:

You cannot specify the SEND option:

1. at the system console serving as the master terminal,
2. for a pseudo-terminal in a batch-interface program,
3. or, for a terminal having no printer.

TRACE Options

The following trace options are available with the `TRACE` command. When no options are specified, the whole list is taken by default except PCF and XPCF. These two options are mutually exclusive. With this exception you can specify any combination of options as long as you separate them with spaces.

DISP	traces the information supplied as a result of the execution of any COBOL `DISPLAY` verbs in the TPRs. Note that any `DISPLAY` verbs in the TPR (except the UPON CONSOLE version) will not be executed unless the associated terminal is in trace mode and the DISP option has been selected.
FLOW	traces the processing flow of the transaction.
IDS2	traces actions performed by IDS/II as specified in the IDS/II options file.
MESG	traces the input messages to and output messages from a transaction.
MREC	traces modifications to records of TDS-controlled files during the processing of a transaction.
RREC	traces read operations performed on TDS-controlled file records during the processing of a transaction.
TRST	records an image of the Transaction-Storage area on completion of each TPR.

PCF

The Program Checkout Facility allows the user to enter debugging commands that apply to subsequent TPRs.
If PCF is specified, a dialog is initiated between the terminal operator and the transaction. The terminal operator enters PCF commands that will be applied to each TPR once TRACE has been activated. The input is ended by EOD. If the TRACE transaction aborts with return code FUNCNAV, this means that the DEBUG option is present in the TDS STEP JCL statement.

```
 -->      TRACE { SEND  } [=terminal-name] [options-list] PCF
                { PRINT }

 <-- TERMINAL IN TRACE MODE OUTPUT ON xxxxxxxxx

 --> ENTER COMMAND
 --> PCF-command1
     .
     .
     .
 <-- ENTER COMMAND
 --> PCF-command
 <-- ENTER COMMAND
 --> EOD
 <-- READY
```

The set of commands is written to a subfile belonging the on-line DEBUGFILE. The set of commands will apply to all TPRs following the TRACE command. If a new set of commands is required, you must start a new TRACE command.

For details of PCF commands, see the *Program Checkout Facility User Guide*.

For a quick reference to the syntax of PCF commands, see Appendix A. This Appendix also gives a more complete description of each trace option.

XPCF

The Program Checkout Facility allows the user to debug interactively a TPR from an IOF console. For more information, see the *Program Checkout Facility User's Guide*.

With the XPCF option, if the TRACE transaction aborts with return code FUNCNAV, this means that the DEBUG option is present in the TDS STEP JCL statement.

Canceling the Trace

To cancel a trace, enter:

```
TRACE OFF
```

This sets the terminal to line mode and sets the terminal to READY. If the PRINT option was previously specified, the contents of DEBUGFILE are printed immediately.

13.3.1.2 Description of the Extended `TRACE` Command

The TRACE PRINT (or TRACE SEND) command sets the transaction to the 'traced' state. This means that the trace is started each time the specified transaction starts for the specified user (or each user if '*' is specified) starting this transaction and the trace stops when the transaction ends. The type of events traced depends on the options you specify.

TX = name-8	specifies the particular transaction which you wish to be traced. If the `TRANSACTION INITIALIZATION ROUTINE` service is used, it is the message-id returned by the routine. This is useful for tracing transactions involved in XCP1 or XCP2 conversations. Only users with the required authority codes for running the specified transaction can enter the `TRACE` command. The `TRACE PRINT` (or `SEND`) command has no immediate effect, but it is effective on next occurrences of the specified transaction. In other words, this command is ineffective for transactions already running when the command is entered.
USER = name-12	specifies the particular user for which any occurrence of the transaction must be traced. The user involved in the trace need not be connected to the TDS application when you enter the TRACE command.
USER = *	specifies that the transaction must be traced each time it is started, regardless of the user involved.
TPR = name-12	starts tracing from the specified TPR for one commitment unit. However, the `tpr-name` is taken into account only if this is the name of the first TPR in the commitment unit.

FROM	The trace continues after the specified transaction has ended and stops: – either when the user involved in the trace logs off, – or when you enter the `TRACE OFF` command. You can force the trace to continue tracing any further commitment units whose first TPR corresponds to the specified named TPR if you specify both `TPR` and `FROM`. In this release, the parameter `FROM` is taken into account only for transactions started by TM or XCP1 correspondents (not XCP2).
OPER	The user who enters the TRACE command.

Results of the Trace

The results of the trace are stored as follows for the TRACE PRINT option:

The member name created by a terminal in trace mode is as follows:

```
tx name_____user name_______trace instance number
<- 8 char -><- 12 char  ->  <- 1 character + 2 digits ->
```

The user name (12 characters long) and the transaction name (8 characters long) are padded with underscore characters and the trace instance number is preceded by an underscore.

One member is created for each occurrence of a traced transaction.

When 'FROM' is specified (and meaningful), only one member is created for the whole trace session (for that user).

When 'FROM' and 'USER=*' are both specified, one member is created for each user activating the transaction. This member is used for the whole trace session (for that user).

The results of a `TRACE SEND` option are presented in the same way as described for the Standard Syntax above.

Additional TRACE Options

In addition to the TRACE options listed above, there are four new options.

XCPM	traces each message exchanged, including each XCP2, or XCP1 verb.
XCP	traces each XCP1, or XCP2 verb/call procedure, but does not trace the contents of the messages.
PT	traces TCAM verbs used for passthrough communications.
XA	traces ORACLE7-XA routines when an error occurs during their processing, or in all cases when ORA/TDS trace level is 2 (detailed). See the *ORACLE7/TDS User's Guide* for more details. This trace is useful for internal debugging purposes, especially for finding out the exact error status returned by ORACLE7-XA.

Canceling the Trace

To cancel a trace for a transaction, enter:

```
TRACE OFF TX = name-8
```

This sets the transaction to the non-traced state. This means that further occurrences of the transaction will not be traced, whereas the current occurrences are traced until they complete.

Only the master operator, and the user who actually started tracing the specified transaction (`oper-user`) can cancel the trace for a transaction.

Only the master operator can specify the `oper-name` parameter to cancel the traces requested by a specified `user`.

If the `oper-name` parameter is omitted, then only the traces requested by the user who actually enters the `TRACE OFF` command are cancelled.

Displaying Traced Transactions

To display the list of transactions that are being traced, enter:

```
TRACE LIST
```

Only the master operator can specify the `tx-name` and `oper-name` parameters.

13.3.1.3 Report/Output Produced by the Extended TRACE Command

After the `TRACE` command has executed, one of the following messages is displayed on the requester's terminal.

```
PRINT option -> READY
SEND option ->TRANSACTION xxxxxxxx TRACED ON TERMINAL xxxxxxxxxxxx
PCF  option ->: PCF COMMANDS ARE IN xxx..x_INxx
```

13.3.1.4 Status Values Returned by the TRACE Command

When you use the TRACE command, you may receive the following status values:

- INVALID TRANSACTION
- INSUFFICIENT AUTHORITY CODES
- TRACE MANAGED BY ANOTHER USER
- TRACE ALREADY ACTIVE
- DEVICE NOT AVAILABLE FOR TRACES
- ERROR IN TRANSACTION PARAMETERS
- UNKNOWN TRANSACTION

13.3.1.5 Examples of Using the TRACE Command

EXAMPLE 1

SMITH enters: `TRACE PRINT TX=DEBIT USER= DUPONT`	DEBIT is the transaction for which the trace is to be started.
DUPONT logs on and starts transaction DEBIT	starts tracing DEBIT
GRANT starts transaction DEBIT	DEBIT is not traced
SMITH enters: `TRACE LIST`	Displays the DEBIT transaction on Smith's terminal. This is the only transaction for which Smith requested the trace.
DUPONT enters: `TRACE LIST`	No transactions are being traced for Dupont
SMITH enters: `TRACE OFF TX=DEBIT`	Stops the trace for DEBIT.

SMITH enters: `TRACE LIST`	No transactions are being traced for Smith
DUPONT starts transaction DEBIT	DEBIT is not traced

EXAMPLE 2

SMITH enters: `TRACE PRINT FROM TX=DEBIT USER= DUPONT`	Indicates that DEBIT is to be traced
DUPONT logs on starts transaction DEBIT	DEBIT is traced
DUPONT starts transaction CREDIT	CREDIT is traced
SMITH enters: `TRACE OFF TX=DEBIT`	Stops tracing DEBIT
DUPONT starts transaction CREDIT	CREDIT is not traced

EXAMPLE 3

SMITH enters: `TRACE PRINT TX=DEBIT USER= DUPONT`	Indicates that DEBIT is to be traced
Master operator enters: `TRACE LIST TX=* USER=SMITH`	Displays DEBIT
Master operator enters: `TRACE OFF TX=* USER=SMITH`	Cancels the trace for DEBIT

❑

13.3.1.6 Notes on the Extended TRACE Command

1. The TDS monitor does not manage any conflicts:

   When you use the `SEND` option, several simultaneous occurrences of the transaction are displayed on the same terminal.

   Only one oper-user can manage the trace of a transaction at a time. As soon as a user has started the trace, no other users can either start the trace, or cancel it (except the master operator) until the first `oper-user` cancels it.

   Entering the `TRACE OFF` command for a transaction while this transaction is running may cause the current traces to abort, especially if the PCF option was on.

2. Do not confuse the user and correspondent concepts. In some cases, they are different:

   Transactions spawned towards a dummy correspondent run on behalf of a user who also activated the spawning transaction.

   For instance, a transaction TX1 running for SMITH spawns a transaction TX2 towards the DUMMY correspondent. Then TX2 runs also on behalf of the user SMITH. Therefore, to trace the spawned transaction, you should specify SMITH as the user-name.

   LOGERR and SYNCPEVT transactions run for the user <tds-name>_ADM.

   Transactions where the principal conversation is XCP2 run on behalf of the user specified in the `H_PPC_UINVK` verb (described in the *CPI-C/XCP2 User's Guide*). If the parameter PPC_SECURITY_OPT is set to P_SECUR_NONE, the transaction runs for the XCP2 correspondent that invokes the transaction.

3. Effect of the `TRACE TX=name-8` command on the `standard TRACE` command as used in previous releases:

   The `TRACE TX=name-8` command has no effect if and as long as the specified traced-user is itself in traced mode. This means that if the trace is running for a user, the trace goes on in same way even if the user starts a transaction in the traced state (trace options specified for the transaction are ignored). This situation is shown in the following example:

DUPONT logs on and enters: `TRACE PRINT`	starts the trace for DUPONT

SMITH logs on and enters:
`TRACE SEND = terminal-name`
`TX=DEBIT USER=DUPONT`
	Indicates that DEBIT is to be traced

DUPONT starts the transaction DEBIT	Traces DEBIT and stores the results of the trace in a subfile (`PRINT` option).

But, if the specified `oper-user` sets itself to traced mode, while it was already being traced (it started a transaction in traced state with the FROM option), the new options are taken into account. This is shown below:

SMITH logs on and enters: `TRACE SEND = terminal-name FROM TX=DEBIT USER=DUPONT`	Indicates that DEBIT is to be traced
DUPONT starts the transaction DEBIT	Traces DEBIT and stores the results of the trace in a subfile (`SEND` option).
DUPONT starts the transaction CREDIT	Traces CREDIT because you specified `FROM` and sends the results of the trace to the terminal.
DUPONT enters: `TRACE PRINT`	starts the trace for DUPONT
DUPONT starts the transaction DEBIT	Traces DEBIT and stores the results of the trace in a subfile.

4. When a TDS application is Warm restarted, the effects of the Extended `TRACE` command are lost, that is, the transaction is no longer in traced mode; therefore further occurrences of the transaction will not be traced. But if the Standard `TRACE` command was running for users when the TDS application failed, then this trace is reactivated.

13.3.2 Debugging at TDSGEN

By specifying FOR DEBUG in the MESSAGE statement of the TRANSACTION SECTION in TDSGEN, the user can specify which transactions are to be debugged. This allows transactions to modify files. However, as soon as the commitment unit terminates, the files will be rolled back to their original unmodified state. New transactions can therefore be developed and tested on 'live' files or databases under real conditions without risking corrupting data or disturbing operational transactions.

13.3.3 Debugging Using TDS Batch Interface Procedures

TDS Batch Interface procedures provide a debugging tool that allows a batch program to simulate a terminal. The simulated terminal is known as a batch pseudo-terminal. Such a pseudo-terminal can use the protocol associated with a real terminal. The batch interface allows TDS to be debugged, without the constraints that would be imposed by using live terminals.

This allows FORMS to be used through the batch interface, provided that FORMS supports the simulated terminal.

These procedures allow communication to be established with a remote TDS.

Several pseudo-terminals can connect to TDS simultaneously. Each pseudo-terminal can be either conversational or receive-only, according to the CONVERSATION-KEY parameter passed when the batch program simulates log-on.

The interface between the batch program and TDS includes three subroutines that are called by the batch program:

- H_TP7_UBCNCT which logs on the pseudo-terminal.
- H_TP7_UBDIALOG which sends and receives messages.
- H_TP7_UBRESUME that informs TDS that the last message has been processed and that another can be received.

As from TS 7254, Batch Interface subroutine names can be written "H_TP7_xx" instead of "H_MT_xx". The old syntax "H_MT_xx" is still supported.

The data structure retrieved by COPY BATCH-MSG-AREA serves to exchange data between the program and TDS.

```
01  BATCH-MSG-AREA.
    02  MESSAGE-LENGTH                          COMP-1.
    02  CONVERSATION-KEY                        PIC 9.
        88  CONVERSATION                        VALUE 1.
        88  NO-CONVERSATION                     VALUE 0.
        88  END-OF-SESSION                      VALUE 5.
    02  END-KEY                                 PIC 9.
        88  INTERMEDIATE                        VALUE 0.
        88  END-OF-TPR                          VALUE 1.
        88  END-OF-COMMIT                       VALUE 2.
        88  END-OF-TX                           VALUE 3.
    02  ERROR-KEY                               PIC 99.
        88  NO-ERROR                            VALUE 0.
        88  ABORTED                             VALUE 1.
        88  UNKNOWN-TX                          VALUE 2.
        88  LOCAL-RESTART                       VALUE 3.
        88  TDS-RESTART                         VALUE 4.
        88  TIMEOUT                             VALUE 96.
        88  ARGERR                              VALUE 97.
        88  NOSEG                               VALUE 98.
        88  DENIED                              VALUE 99.
    02  MESSAGE-TEXT                            PIC X(1024).
    02  CONNECT-TEXT REDEFINES MESSAGE-TEXT.
        03  TDS-NAME                            PIC X(4).
        03  BATCH-NAME                          PIC X(4).
        03  PASSWORD                            PIC X(8).
        03  USER-NAME                           PIC X(8).
        03  PROJECT-NAME                        PIC X(8).
        03  ACCOUNT-NAME                        PIC X(8).
        03  TERMINAL-TYPE                       PIC X(8).
        03  NODE-NAME                           PIC X(4).
        03  FILLER                              PIC X(n).
```

Usage

- The MESSAGE-LENGTH data item identifies the length of information found in data item MESSAGE-TEXT. It must be initialized with the maximum length of MESSAGE-TEXT before the connect function (CALL "H_TP7_UBCNCT"...) is called, or with the current message length in MESSAGE-TEXT before the dialog function (CALL "H_TP7_UBDIALOG") is called.

  The contents of the data item MESSAGE-LENGTH are set by the standard Batch Interface as a result of a call to the dialog ("H_TP7_UBDIALOG"), or resume function ("H_TP7_UBRESUME"). MESSAGE-LENGTH indicates the current length of the message stored in MESSAGE-TEXT by TDS. MESSAGE-LENGTH can be an input or output parameter.

- The CONVERSATION-KEY data item indicates whether the message generated by TDS gives the turn to the batch program. Before calling the connect function, the batch program must initialize this data item to CONVERSATION if the program is to operate conversationally; alternatively, if the program is to operate in a receive-only mode, CONVERSATION-KEY must be set to NO-CONVERSATION. It is an output parameter.

- The END-KEY data item is set by TDS as a result of the execution of a dialog or resume function, in order to indicate which entity (TPR, commitment unit, or transaction) is closed by the message. It is an output parameter.

- The ERROR-KEY data item is set by TDS as a result of the execution of a connect, dialog or resume function. It indicates if any error has occurred, and the type of error. It is an output parameter.

  Value 96 (TIMEOUT) is returned when an EXT-TIMEOUT value not null has been used in the second parameter structure and this delay is exhausted.

  Value 97 (ARGERR) is returned by H_TP7_UBCNCT when the second parameter is used and EXT-VERSION is out of allowed values or if EXT-TIMEOUT is negative.

  Value 98 (NOSEG), when returned by H_TP7_UBCNCT, means that the working segment cannot be created. When returned by H_TP7_UBDIALOG or H_TP7_RESUME, it means that the working segment has not been created.

- The MESSAGE-TEXT data item contains either an input text provided by the batch program or a message sent by TDS. The input text is either an input message for a transaction or a conversational input for the connect function. This standard description is named CONNECT-TEXT. It can be an input or output parameter.

- The MESSAGE-TEXT data item length supplied by the COPY statement ranges from 52 characters to the value specified in the MESSAGE-LENGTH clause at TDSGEN. The default value is 1024 characters.

- Before the execution of the connect function, the data structure named CONNECT-TEXT is to be set as follows:

  TDS-NAME should contain the name of the TDS subsystem to be accessed.

  BATCH-NAME should be a unique system name identifying the pseudo-terminal as a correspondent of the communication facility. This name does not need to be described in the network generation; it has the same format as a terminal name.

  PASSWORD should contain the password associated with the user name specified in the USER-NAME field.

  USER-NAME should contain the unique name of a user known by the specified TDS subsystem; this user name is used in a similar way as for a terminal user.

  PROJECT-NAME and ACCOUNT-NAME should contain the project and billing under which the user intends to work. When default project and/or billing are to be used, the corresponding data item should contain spaces.

  TERMINAL-TYPE, when requested to simulate a true terminal, may contain the name of a supported terminal type. The following terminals are supported:

  - DKU7007,
  - DKU7107,
  - DKU7211,
  - IBM3270,
  - IBM3278/3279,
  - VIP7804,
  - PC7800,
  - Minitel.

  If the terminal type specified is erroneous, the connect function is denied and an error-key DENIED is returned.

  If this field is completed, the functions and protocols relevant to terminals apply to the simulated terminal:

- The user must specify specific terminal character strings, such as, message headers and device protocol headers (STA, FC1, FC2).

- FORMS is accessible to simulated terminals if the terminal type is supported by the current version of FORMS. The program will receive data in the specific format of the terminal.

  NODE-NAME is an optional field but if present it must contain a valid node name as generated in the Network. This name is the node name under which the TDS application is running and enables a batch application to be connected to a remote TDS.

  FILLER: (n) = 976 or 972 if node name is specified.

13.3.3.1 CONNECT Function

Syntax

```
CALL "H_TP7_UBCNCT" USING BATCH-MSG-AREA [ EXT-AREA ].
```

Description

The connect function must be performed as the first batch-entry function and is similar to a terminal user log-on.

Usage

- The connect function may be performed only once and must be successfully executed before the dialog and resume functions.
- The BATCH-MSG-AREA data structure is used in a specific manner prior to execution of the connect function.
- The MESSAGE-LENGTH field contains the maximum length of the MESSAGE-TEXT data item.
- The CONVERSATION-KEY field specifies the mode (receive-only or conversational) in which the program will operate.
- The MESSAGE-TEXT field contains the data structure identified as CONNECT-TEXT.
- Upon completion of the connect function, the ERROR-KEY data item should be checked for successful completion (NO-ERROR). After an unsuccessful connect no other functions may be performed.
- The CONVERSATION-KEY data item will indicate if the next function should be dialog or resume upon completion of the connect function.

- The EXT-AREA is used to pass a 12-character password. It has the following format:

```
01  EXT-AREA.
    02  EXT-VERSION               COMP-1.
    02  EXT-PASSWORD              PIC X(12).
    02  EXT-TIMEOUT               COMP-1.
    02  EXT-USERNAME              PIC X(12).
    02  EXT-PROJECTNAME           PIC X(12).
    02  EXT-ACCOUNTNAME           PIC X(12).
    02  EXT-STATUS-AREA.
        03 EXT-ERRORID            PIC X(2).
        03 EXT-RETURN-CODE        PIC X(30).
        03 EXT-STAT               PIC X(2).
        03 EXT-REASON             PIC X(4).
        03 EXT-COMP-STAT          PIC X(2).
```

If EXT-AREA is absent, an 8-character password is taken from the BATCH-MSG-AREA.

If EXT-VERSION is 1, only the EXT-PASSWORD field is taken into account (the password is taken from this field).

If EXT-VERSION is 2, both the EXT-PASSWORD and EXT-TIMEOUT fields are used. The connection will be done with a TIMEOUT value (if a positive value is supplied). This timeout value, specified in seconds (to connection time), is supplied in EXT-TIMEOUT. If EXT-TIMEOUT is null (0), no timer detection upon connection is done. The value is limited by the COMP-1 capacity (about 9 hours). The EXT-PASSWORD must be filled with 12 characters. If EXT-VERSION is not 1 or if EXT-TIMEOUT is negative, ERROR-KEY is filled with ARGERR (97). If EXT-TIMEOUT is not null (0) and no response to the connection received passed this delay, ERROR-KEY is filled with TIME-OUT (96).

If EXT-VERSION is 3, the fields EXT-PASSWORD, EXT-TIMEOUT, EXT-USERNAME, EXT-PROJECTNAME, and EXT-ACCOUNTNAME of this extended area are taken into account and must be filled. If no timeout detection is desired, EXT-TIMEOUT must be set to 0 (zero). If default project and/or billing are to be used, then EXT-PROJECTNAME and EXT-ACCOUNTNAME may be set to blanks.

If EXT-VERSION is 4, the fields of EXT-STATUS-AREA are available. EXT-STATUS-AREA in an output area used to help you debug problems when finalizing a Batch Interface program.

If EXT-VERSION is not 1, 2, 3, or 4 or if EXT-TIMEOUT is negative or an exception has been detected on the parameters provided by the caller, the ERROR-KEY is filled with ARGERR (97).

If USERNAME is to be passed with more than 8 characters, the EXT-VERSION field should be set to 3 and the three fields EXT-USERNAME, EXT-PROJECTNAME, and EXT-ACCOUNTNAME must be filled.

EXAMPLE

```
CALL "H_TP7_UBCNCT" USING BATCH-MSG-AREA [ EXT-AREA ].
CALL "HTP7_UBDIALOG" USING BATCH-MSG-AREA
[ DVCHD-AREA EXT-AREA ].
CALL "H_TP7_UBRESUME" USING BATCH-MSG-AREA [ EXT-AREA ].
```

❑

If the EXT-AREA structure is to be used with the "H_TP7_UBDIALOG" subroutine, the DVCHD-AREA structure must be filled (if no device header is to be sent, STRUCT-LGT should be set to 0).

The fields EXT-PASSWORD and EXT-ACCOUNTNAME are used (in input) by the "H_TP7_UBCNCT" routine only. These fields are neither taken into account nor modified by the subroutines "H_TP7_UBDIALOG" and "H_TP7_UBRESUME".

EXT-ERRORID is a unique identifier for each abnormal return from the Batch Interface subroutine.

EXT-RETURN-CODE is the last return code received (in case of anomaly it is shown in edited format). Its value may be DONE.

EXT-STAT is the status of the VCAM verb returning the error state.

EXT-REASON is the reason for the disconnection or the connection rejection.

The content of EXT-COMP-STAT depends on the value of EXT-ERRORID. It is explained below.

EXT-ERRORID Values Returned by "H_TP7_UBCNCT"

"00"	No error. All fields of EXT-STATUS-AREA are loaded with the character "0".
"01"	Problem when activating the workstation. The following status fields are meaningful: EXT-RETURN-CODE (CHECK,ARGERR) When "CHECK", see the EXT-STAT value. When "ARGERR", contact your Service Center.

EXT-STAT Values

"32"X: The workstation is already active (name is already known).
Action: Verify name and/or check program logic.

"33"X: The workstation is de-activating.
Action: Contact your Service Center.

"34"X: System resource overload.
Action: Contact your Service Center.

“02” Problem with model of declared terminal.
The following status fields are meaningful:
EXT-RETURN-CODE (NOMATCH, other RC)
When "NOMATCH", the terminal type is unknown.
Action: Check the terminal type value with the H_TERM subfile of SYS.HSLLIB.
When "other RC", contact your Service Center.

“03” Problem when activating mailbox.
The following status fields are meaningful:
EXT-RETURN-CODE (CHECK, ARGERR)
When "CHECK", see the EXT-STAT value. When "ARGERR", contact your Service Center.

EXT-STAT Values

"32"X: The mailbox is already active (name is already known).
Action: Verify name and/or check program logic.

"33"X: The mailbox is de-activating.
Action: Contact your Service Center.

"34"X: System resource overload.
Action: Contact your Service Center.

“04” Problem when opening message group.

The following status fields are meaningful:
EXT-RETURN-CODE (CHECK, ARGERR)
When "CHECK", see the EXT-STAT value.
When "ARGERR", contact your Service Center.

EXT-STAT Values

"34"X: System resource overload.
Action: Contact your Service Center.

"35"X: Connection reject.
Action: See the reason for the connection rejection in EXT-REASON.

"05" Problem on reception of event during connection phase.
The following status fields are meaningful:
EXT-RETURN-CODE, EXT-STAT
Action: Contact your Service Center.

"06" OPENACK event was expected, but another one was received.
The following status fields are meaningful:
EXT-RETURN-CODE, EXT-COMP-STAT (received event)
Action: Contact your Service Center.

"07" OPENACK REJECT (connection is refused in the second phase).
The following status fields are meaningful:
EXT-RETURN-CODE, EXT-STAT (reject code "35"X), EXT-REASON
Action: See the reason for the rejection in EXT-REASON.

"08" Problem at connection negotiation ($H_INQUIR VCAM verb)
The following status fields are meaningful:
EXT-RETURN-CODE, EXT-STAT
Action: Contact your Service Center.

EXT-ERRORID Values Returned by Other Subroutines

"09" Erroneous subroutine call in this context. The subroutine has probably been called while the Batch Interface was already in error.
No status fields are meaningful.
Action: Check your program.

"10" BATCH-MSG-AREA area is not the same as the one used at the last call to the "H_TP7_UBCNCT" subroutine (control is done on address).
No status fields are meaningful.
Action: Check your program.

"11" H_TP7_UBDIALOG subroutine has been called instead of H_TP7_UBRESUME (conversation key was "NO-CONVERSATION").
No status fields are meaningful.
Action: Check your program.

“12” Input message length is negative.
No status fields are meaningful.
Action: Check your program.

“13” H_TP7_UBRESUME subroutine has been called instead of
H_TP7_UBDIALOG (conversation key was "CONVERSATION").
No status fields are meaningful.
Action: Check your program.

“14” Abnormal status when receiving an interruption during another VCAM primitive.
The following status fields are meaningful:
EXT-RETURN-CODE, EXT-STAT
Action: Contact your Service Center.

“15” Abnormal status (neither done nor interrupt pending) when sending or receiving a message.
The following status fields are meaningful:
EXT-RETURN-CODE, EXT-STAT
Action: Contact your Service Center.

“16” Abnormal status (neither v_done nor v_skip) when receiving an event.
The following status fields are meaningful:
EXT-RETURN-CODE, EXT-STAT
Action: Contact your Service Center.

“17” Disconnection occurred (V_MSGCLOSED has been received).
The following status fields are meaningful:
EXT-RETURN-CODE, EXT-REASON
Action: None, mailbox and workstation have been de-activated.

“18” Abnormal status when receiving an interruption after a V_INTERRUPT event.
The following status fields are meaningful:
EXT-RETURN-CODE, EXT-STAT
Action: Contact your Service Center.

“19” Unexpected event received by the Batch Interface.
The following status field is meaningful:
EXT-COMP-STAT (received event)
Action: Contact your Service Center.

“20” Status neither done nor moredata when receiving a message.
The following status field is meaningful:
EXT-RETURN-CODE, EXT-STAT
Action: Contact your Service Center.

"21" Detected level not supported.
The following status field is meaningful:
EXT-COMP-STAT (detected level)
Action: Contact your Service Center.

EXT-ERRORID Values Returned by Other Subroutines

"01"X Abnormal rejection.
"02"X Destination node not operable.
"03"X Destination node saturated.
"04"X Mailbox unknown.
"05"X Mailbox not operable.
"06"X Mailbox saturated.
"07"X Destination application saturated.
"09"X Dialog rejection (as a result of negotiation).
"0A"X Presentation rejection (as a result of negotiation).
"15"X Timeout.
"18"X Security violation.
"40"X Destination node unknown.
"41"X Path to the destination node is not available.
"42"X Duplicate user identifier.

13.3.3.2 DIALOG Function

Syntax

```
CALL "H_TP7_UBDIALOG" USING BATCH-MSG-AREA.
```

Description

Sends a message to TDS and awaits the corresponding reply.

Usage

- The dialog function may be executed only when the previous statement (Call H_TP7_UBCNCT, or Call H_TP7_UBDIALOG) has obtained a CONVERSATION-KEY set to CONVERSATION.
- Before the function is executed, the message to be sent to TDS must be moved to the MESSAGE-TEXT field and MESSAGE-LENGTH be set to the appropriate value. After the function is executed, the reply from TDS is available in the MESSAGE-TEXT field, the length of which is stored in MESSAGE-LENGTH.

13.3.3.3 DIALOG Function with the Device Header

Syntax

```
CALL "H_TP7_UBDIALOG" USING BATCH-MSG-AREA DVCHD-AREA.
```

Description

Sends a message to TDS with DEVICE HEADER and awaits the corresponding reply.

Usage

In addition to the DIALOG function just described, the following structure must be filled:

```
01 DVCHD-AREA.
  02 STRUCT-LGT COMP-1.
  02 HEADER.
  03 HEADER-LGT COMP-1.
  03 HEADER-VL PIC X (i).
```

- The STRUCT-LGT data item defines the length of the structure and must be equal to HEADER-LGT + 4.
- The HEADER-LGT data item defines the length of the value in HEADER-VL and is always even because two characters give one hexadecimal byte.
- HEADER-VL contains the device header value whose PIC must not exceed 30. The values specified in the picture string must be a hexadecimal value.

EXAMPLE

```
MOVE 6 TO HEADER-LGT.
MOVE "7DD7C7" TO HEADER-VL.
MOVE 10 TO STRUCT-LGT.
```
❑

Remarks

The device header is not tested for terminal type and the terminal is forced to operate in unedited mode. Thus, in line mode, the device header is added before the input text and must be taken into account by TPRs.

13.3.3.4 RESUME Function

Syntax

```
CALL "H_TP7_UBRESUME" USING BATCH-MSG-AREA.
```

Description

Notifies TDS that the message received is processed and that the BATCH-MSG-AREA is available for a new message.

Usage

- The resume function may be executed only if CONVERSATION-KEY is set to NO-CONVERSATION upon the return of H_TP7_UBCNCT, or H_TP7_UBDIALOG, or H_TP7_UBRESUME.
- After the execution of the function, the message from TDS is available in the MESSAGE-TEXT field, the length of which is stored in MESSAGE-LENGTH.

13.3.4 Example of a Batch Interface Program

A batch program operating a pseudo-terminal would have the structure illustrated by Figure 13-3.

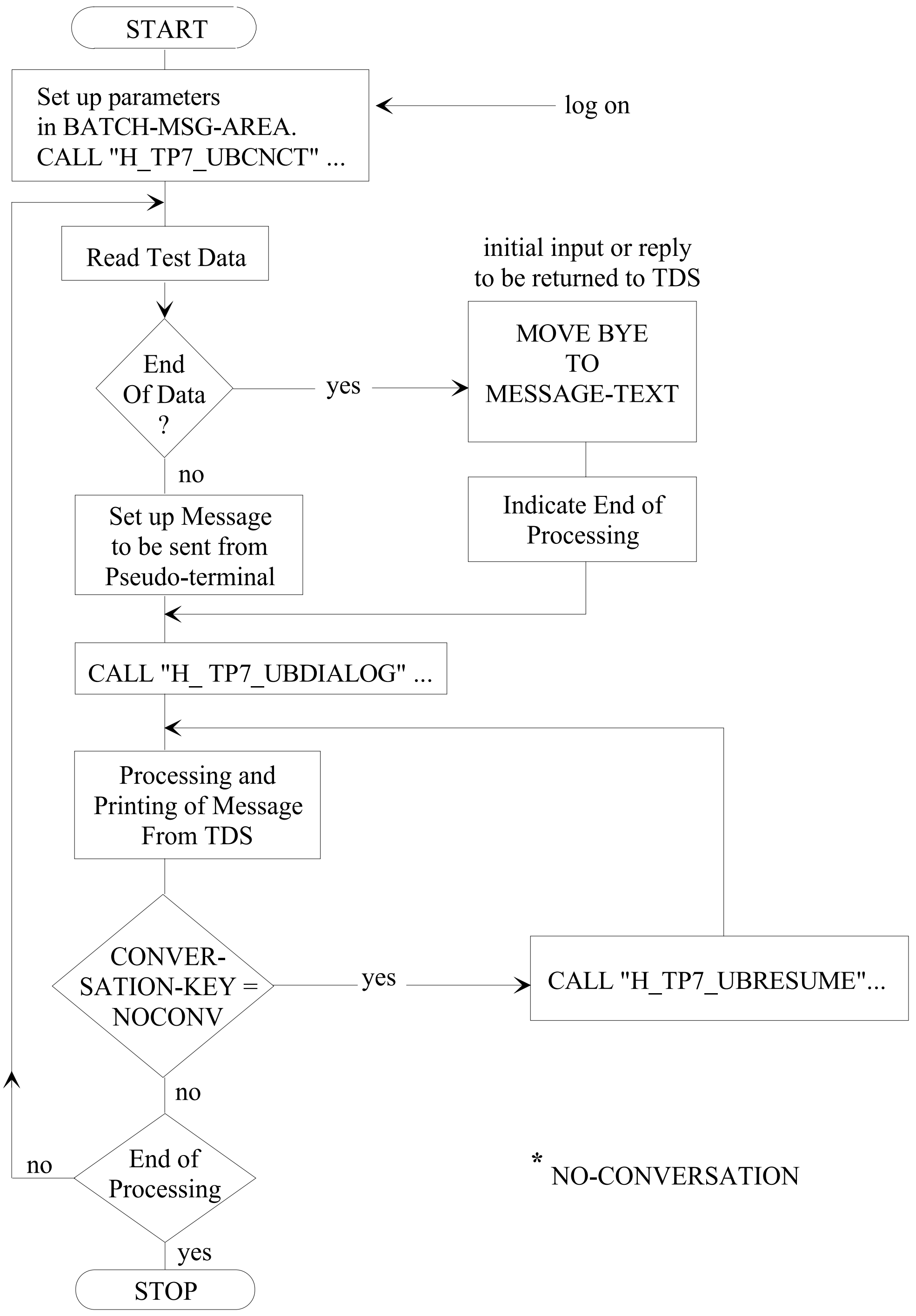

START
Set up parameters in BATCH-MSG-AREA. CALL "H_TP7_UBCNCT" ...
log on
Read Test Data
initial input or reply to be returned to TDS
End Of Data ?
yes
MOVE BYE TO MESSAGE-TEXT
no
Set up Message to be sent from Pseudo-terminal
Indicate End of Processing
CALL "H_ TP7_UBDIALOG" ...
Processing and Printing of Message From TDS
CONVER-SATION-KEY = NOCONV
yes
CALL "H_TP7_UBRESUME"...
no
no
End of Processing
* NO-CONVERSATION
yes
STOP

Figure 13-3. Structure of a TDS Batch Interface

The logic shown in Figure 13-3 is that of a batch program operating as a conversational pseudo-terminal.

The input data consists of values to be sent to the TDS subsystem as though they had been typed at a terminal, including values in error to simulate mistakes.

The data has to be carefully checked by the batch program especially where a transaction includes a large number of exchanges; missing test data or misplaced data will disrupt the conversational processes. It may be useful to separate the test data records for a sequence of transactions, so that if data is accidentally lost, the program can recover from the next marker point.

The output from the batch program consists of the data received as messages from the TDS subsystem to the pseudo-terminal; the batch program will edit the data received, not only to make the printed output easier to read but also because the data may contain embedded control characters.

EXAMPLE

Take an order processing system simulating one of the terminals for order entry. Two types of transaction can be started at this terminal. Therefore the input will contain two types of data. The program is as follows:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. BATCHTEST.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
     SELECT TERMINAL-DATA ASSIGN TO ...
     SELECT TDS-REPLIES ASSIGN TO ...
```

The data required for the simulation of the terminal operator's actions appears in the file TERMINAL-DATA. The results obtained from TDS are output to TDS-REPLIES.

In this example, data for each transaction is input as a variable-length record. Output data is printed as it is received from TDS.

```
               DATA DIVISION.
               FILE SECTION.
               FD TERMINAL-DATA ...
               01 ORDER-DATA.
                  02 DTYPE ...
                  02 CUST NUMBER ...
                  02 OCOUNT ...
                  02 ORDER-DETAILS OCCURS 20 DEPENDING ON OCOUNT.
                     03 OITEM-NUMBER ...
                     03 OQUANTITY ...
               01 QUOTE-DATA.
                  02 FILLER ...
                  02 QITEM-NUMBER ...
                  02 QUANTITY ...
FD TDS-REPLIES ...
01 PLINE                              PIC X(60).
```

❑

The test data consists of two types of record:

- ORDER-DATA for the transaction ORDER contains a customer number and up to 20 order items. The record type is indicated by the field DTYPE that is common to both types of record.
- QUOTE-DATA for the transaction QUOTE contains the item number for which a quote is required and the quantity ordered.

The test data consists of two types of record:

- ORDER-DATA for the transaction ORDER contains a customer number and up to 20 order items. The record type is indicated by the field DTYPE that is common to both types of record.
- QUOTE-DATA for the transaction QUOTE contains the item number for which a quote is required and the quantity ordered.

Data is output to TDS-REPLIES. The maximum message length, including control codes, expected from TDS is 60 characters.

```
WORKING-STORAGE SECTION.
77 K                PIC 99.
   COPY BATCH-MSG-AREA.
01 MGROUP.
   02 TRANS-NAME ...
   02 MCUST-NUMBER ...
```

The description of the Batch Message Area can be obtained from the source library by a COPY statement.

The initial message transmitted by the transaction ORDER contains the message identifier and the customer number. These two items have to be set up in MGROUP before being transferred.

```
PROCEDURE DIVISION.
FIRST-PARAGRAPH.
   OPEN INPUT TERMINAL-DATA ...
        OUTPUT TDS-REPLIES ...
   MOVE 1024 TO MESSAGE-LENGTH.
   MOVE 1 TO CONVERSATION-KEY.
   MOVE "TDS1" TO TDS-NAME.
   MOVE "BT01" TO BATCH-NAME.
   MOVE "JOANIE" TO USER-NAME.
   MOVE SPACES TO PASSWORD, PROJECT-NAME, ACCOUNT-NAME.
   MOVE SPACES TO TERMINAL-TYPE, NODE-NAME.
   CALL "H_TP7_UBCNCT" USING BATCH-MSG-AREA.
   IF ERROR-KEY NOT = ZERO STOP RUN.
```

Having opened the necessary files, the batch program logs on to TDS as a pseudo-terminal in conversational mode (CONVERSATION-KEY = CONV) quoting the name of the appropriate version of TDS, giving the 'terminal name', BT01, and the user identifier. If the attempt to log on is unsuccessful, the program ends.

```
LOOP1.
    READ TERMINAL-DATA AT END GO TO LAST-PARA.
    IF DTYPE = 1 GO TO QUOTE-PARA.
    MOVE "ORDER" TO TRANS-NAME.
    MOVE CUST-NUMBER TO MCUST-NUMBER.
    MOVE MGROUP TO MESSAGE-TEXT.
    MOVE 12 TO MESSAGE-LENGTH.
    CALL "H_TP7_UBDIALOG" USING BATCH-MSG-AREA.
    IF ERROR-KEY NOT = ZERO GO TO ERR-PARA
    PERFORM PRINT-TDS-REPLY.
    MOVE "YES" TO MESSAGE-TEXT.
    MOVE 3 TO MESSAGE-LENGTH.
    CALL "H_TP7_UBDIALOG" USING BATCH-MSG-AREA.
    IF ERROR-KEY NOT = ZERO GO TO ERR-PARA.
    PERFORM PRINT-TDS-REPLY.
```

The program then processes the test data file. It either branches to QUOTE-PARA if the data refers to the transaction QUOTE, or continues to process the transaction ORDER.

The message identifier and customer number are set up together in MESSAGE-TEXT. Their combined lengths are placed in the MESSAGE-LENGTH. UBDIALOG is called to submit this transaction to TDS. Any error returned by TDS will cause the program to branch to ERR-PARA. If the return is normal, the message from TDS is printed.

The normal response from TDS is the name of the customer corresponding to the number sent from the pseudo-terminal.

Two conversations are represented.

```
    PERFORM LOOP2 VARYING K FROM 1 BY UNTIL K = OCOUNT.
LOOP2.
    MOVE OITEM-NUMBER (K) TO MESSAGE-TEXT.
    MOVE 6 TO MESSAGE-LENGTH.
    CALL "H_TP7_UBDIALOG" USING BATCH-MSG-AREA.
    IF ERROR-KEY NOT = ZERO GO TO ERR-PARA.
    PERFORM PRINT-TDS-REPLY.
    MOVE "YES" TO MESSAGE-TEXT.
    MOVE 3 TO MESSAGE-LENGTH.
    CALL "H_TP7_UBDIALOG" USING BATCH-MSG-AREA.
    IF ERROR-KEY NOT = ZERO GO TO ERR-PARA.
    PERFORM PRINT-TDS-REPLY.
    MOVE OQUANTITY (K) TO MESSAGE-TEXT.
    MOVE 5 TO MESSAGE-LENGTH.
    CALL "H_TP7_UBDIALOG" USING BATCH-MSG-AREA.
    IF ERROR-KEY NOT = ZERO GO TO ERR-PARA.
    PERFORM PRINT-TDS-REPLY.
END-LOOP2.
    MOVE ZERO TO MESSAGE-TEXT.
    MOVE 6 TO MESSAGE-LENGTH.
    CALL "H_TP7_UBDIALOG" USING BATCH-MSG-AREA.
    IF ERROR-KEY NOT = ZERO GO TO ERR-PARA.
    PERFORM PRINT-TDS-REPLY.
    GO TO LOOP1.
```

LOOP2 is executed as many times as there are order items in the test data record. It has three sets of five statements; each set:

- defines a message,
- indicates its length,
- calls H_TP7_UBDIALOG,
- checks that the message was handled correctly by TDS,
- prints the reply received from TDS.

The first set inputs an item number to TDS and gets back the corresponding item description. The second set confirms the item. The third set inputs the quantity of that item.

END-LOOP2 sets to zero the item number to indicate the end of the current order. The program then branches back to read the next set of test data.

```
QUOTE-PARA
    MOVE "QUOTE" TO MESSAGE-TEXT.
    MOVE 5 TO MESSAGE-LENGTH.
    CALL "H_TP7_UBDIALOG" USING BATCH-MSG-AREA.
    IF ERROR-KEY NOT = ZERO GO TO ERR-PARA.
    PERFORM PRINT-TDS-REPLY.
    MOVE QITEM-NUMBER TO MESSAGE-TEXT.
    MOVE 6 MESSAGE-LENGTH.
    CALL "H_TP7_UBDIALOG" USING BATCH-MSG-AREA.
    IF ERROR-KEY NOT = ZERO GO TO ERR-PARA.
    PERFORM PRINT-TDS-REPLY.
    MOVE QUANTITY TO MESSAGE-TEXT.
    MOVE 5 TO MESSAGE-LENGTH.
    CALL "H_TP7_UBDIALOG" USING BATCH-MSG-AREA.
    IF ERROR-KEY NOT = ZERO GO TO ERR-PARA.
    PERFORM PRINT-TDS-REPLY.
    GO TO LOOP1.
```

QUOTE-PARA processes the transaction QUOTE in a similar manner to the way in which the transaction ORDER was processed earlier.

```
ERR-PARA.
    MOVE "** ERROR **" TO PLINE.
    WRITE PLINE AFTER 3.
    MOVE ERROR-KEY TO PLINE.
    WRITE PLINE AFTER 1.
    PERFORM PRINT-TDS-REPLY.
DUMMY-PARA.
    GO TO LOOP1.
```

ERR-PARA prints a header, the contents of ERROR-KEY and the text of the message, if any, from TDS. DUMMY-PARA is required for the last GO TO if ERR-PARA is entered from within LAST-PARA. DUMMY-PARA ignores incorrect data and branches to LOOP1 to continue with the input of new data.

```
PRINT-TDS-REPLY.
    MOVE MESSAGE-TEXT TO PLINE.
    INSPECT PLINE REPLACING CODE-VAL BY PRINTABLES.
    WRITE PLINE AFTER 1.
```

The INSPECT statement requires that the following data areas be declared in the WORKING-STORAGE SECTION:

```
01 INSPECT-VALUES.
   02 CODE-VAL              PIC X(12)
     VALUE ""06, 13, 14, 20, 21, 29, 30, 38, 40, 208, 219, 281"".
   02 CODE-REP              PIC X(12)
     VALUE "<>*!?+/%@=#$".
```

CODE-VAL contains values of the COBOL collating sequence. These values represent control codes or symbols that are not graphically represented.

CODE-REP contains printable symbols representing 1-for-1 the values appearing in CODE-VAL.

```
LAST-PARA.
    MOVE "BYE" TO MESSAGE-TEXT.
    MOVE 3 TO MESSAGE-LENGTH.
    CALL "H_TP7_UBRESUME" USING BATCH-MSG-AREA.
    IF ERROR-KEY NOT = ZERO PERFORM ERR-PARA
       ELSE PERFORM PRINT-TDS-REPLY.
    CLOSE TERMINAL-DATA TDS-REPLIES.
    STOP RUN.
```

LAST-PARA is optional and logs off the batch program from TDS. The pseudo-terminal is logged off when the batch program has terminated.

13.3.4.1 Comments on Example

The user-written batch program has a simple structure; in practice it is a powerful debugging tool.

The batch program is free to make use of the TDS trace facility as a pseudo-terminal in the same way as a real terminal would be.

The master terminal can also be simulated. The batch program can issue master commands, provided that the user at the pseudo-terminal logs on as the master defined at TDSGEN. TDS can be run with a mix of real terminals and pseudo-terminals, while being developed and also in a fully operational environment.

The batch interface facility can be used by user-written batch programs simply in order to use the processing characteristics of TDS.

13.3.4.2 Compilation, Linkage, and Execution

The preparation and execution of TDS Batch Interface programs follows the standard pattern, described at the beginning of this chapter.

13.3.5 Debugging Using Batch Programs

Instead of writing a program that uses the TDS batch interface procedures, the user can test his TDS by means of the Standard Batch Program. This program acts as a terminal but operates only one pseudo-terminal.

The test data can be supplied from any file. Each message to be received by TDS occupies as many records as required to hold the text. Figure 13-4 shows the layout of data in the record.

The JCL for the standard batch utility is as follows:

```
STEP H_TDSCTP, FILE = SYS.HLMLIB, DUMP = DATA;
ASSIGN H_CR, input-file-specification;
[ASSIGN H_PR, output-file-specification;]
ENDSTEP;
```

This utility operates with up to 14 entries, each entry having a maximum of 80 characters. An entry is one line as shown in Figure 13-4.

The maximum message length declared in TDSGEN is 1024 characters, otherwise the input messages will be truncated.

The format of data input to the program is positional and starts from position 1 in the following field sizes:

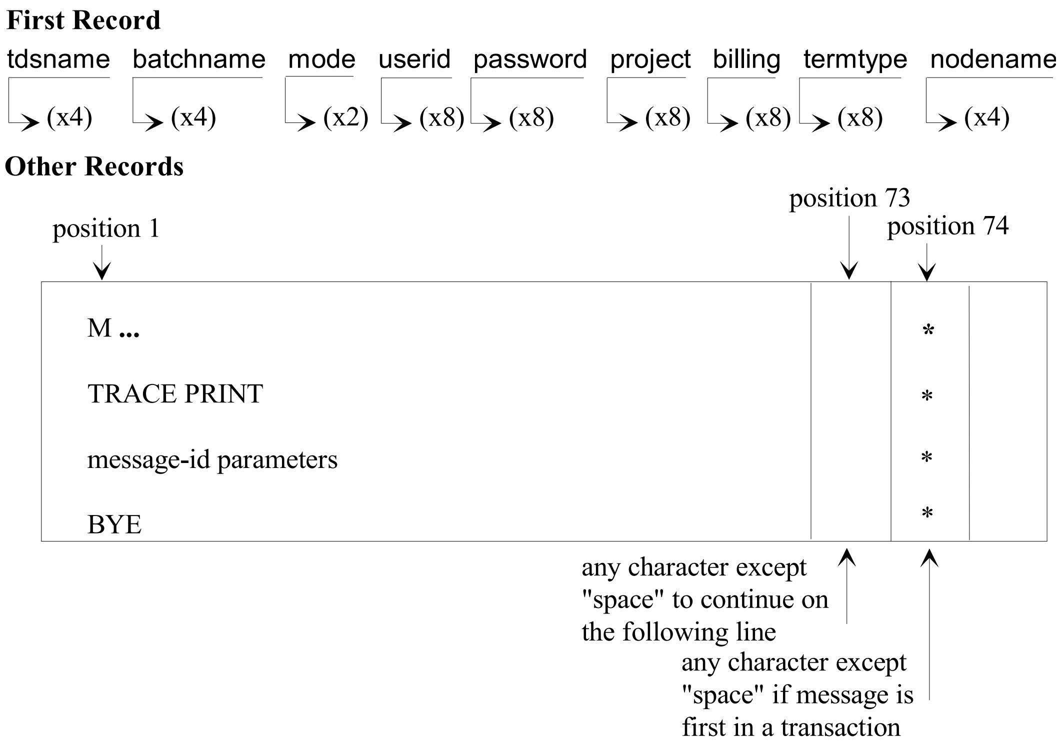


Figure 13-4. Data Format for the Standard Batch Program

The first record holds the log-on information: tdsname, batchname, mode (space if conversational, RO if receive only), userid, password, project, billing, and optionally the terminal type and node name. `Tdsname` is the name of the TDS application. When the `master terminal` is defined at TDSGEN, you can specify the mailbox name.

The following terminals are supported: DKU7007, DKU7107, DKU7211, IBM 3270, IBM 3278/3279, VIP7804, PC7800, and Minitel.

The optional nodename allows the user to connect to a remote TDS application through a network. Local nodename is the default value.

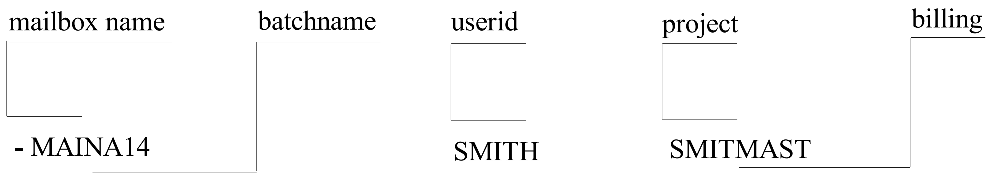
The reason for delimiting transactions by a character in position 74 is that if one transaction aborts, the next in sequence can be correctly located, any intervening messages being omitted.

Messages from TDS are printed via SYSOUT in the form in which they would appear at the terminal. If the TRACE transaction is to be activated, it should specify PRINT or SEND with termnlname.

EXAMPLE OF THE STANDARD BATCH PROGRAM

Here is the JCL for starting H_TDSCTP:

```
$STEP H_TDSCTP FILE=SYS.HLMLIB XPRTY=9;
$ASSIGN H_CR .JCLLIB DVC=&DVC1 MD=&MD1 SUBFILE=&TEST;
$ASSIGN H_PR .PRT DVC=&DVC1 MD=&MD1 SUBFILE=CTP1;
$ENDSTEP;
```



```
_MAIMAI4  SMITH   SMIT   MAST                     POSITION 74
message-id                                               *
MESSAGE "*** This is the name of a user transaction ***"
END OF TRANSACTION
M MODIFY_TX TX = trans1 VALIDATE=0
BYE (user log-off)                                       *
```

❑

14. Terminal Operations

14.1 Introduction

Successful log-on means that the user is connected to GCOS 7 and the appropriate TDS application.

When a physical incident leads to a disconnection, the user can relog on using the same userid. If a transaction was in progress when the disconnection occurred, the transaction is restarted from the last commitment. The last message sent or received is repeated at the terminal.

Terminals ASSIGNed to a TDS application will be logged onto this TDS application, as soon as the terminal operator logs on and the TDS subsystem becomes available.

An unassigned AUTO terminal becomes logged onto a TDS application when a message is sent to it, or when a transaction is spawned to it. At warm restart after a communications failure, these terminals are logged on again by an [M] `ANEW` command.

All terminals can operate in one of two modes:

- command mode, in which the user communicates directly with TDS,
- transaction mode, in which the user communicates with one of the exchanges comprising the transaction.

14.2 Command Mode

The terminal is in the command mode immediately after logon when no transaction is active for it. To exit from the command mode, the user initiates a transaction.

"Break" has no effect other than sending READY, except if a user-defined break transaction is supplied at TDSGEN.

A command can be rejected for one of the following reasons:

- the command is unknown to the TDS application,
- the user is not authorized to request the transaction (mismatch in authority code),
- the command can be overlaid by a message from the master terminal by a [M] `SEND_TDS_USER`, or [M] `MODIFY_TDS_MOT command.`

There are two types of commands that can be issued from the terminal.

- TDS commands which address TDS,
- user commands which address transactions.

14.2.1 User Commands

A user command is entered to initiate a transaction.

Syntax

```
message-id transaction parameters
```

Description

- message-id identifies the transaction by causing TDS to load and activate the first TPR of the transaction. Message-id is up to 8 alphanumeric characters and can be any character string except M, `TRACE`, or `BYE` and must not begin with $*$, question mark (?), the characters defined in the USE LAST MESSAGE or USE MENU clauses (see the *TDS Administrator's Guide*).

  If the message-id is unknown to TDS, UNKNOWN TRANSACTION is displayed.
- transaction-parameters refer to those expected by the transaction.

14.2.2 TDS Commands

These commands for addressing a TDS application, are BYE, TRACE, "menu", PT and the "last message character".

14.2.2.1 BYE

The BYE command is a TDS command used to log off a terminal.

Syntax

```
BYE
```

The terminal receives a log-off message. The terminal is then logged off from TDS and is disconnected from the DPS 7000.

The LOGOUT Transaction, if any, is executed to allow accounting and statistics to be processed.

14.2.2.2 TRACE

Syntax

```
TRACE
{PRINT}
{SEND }[=termnlname[PRT]][TX=name-8][USER=name-12][TPR=name-12][options-list]
{OFF  }
```

For an explanation of the TRACE command, see Chapter 13.

14.2.2.3 MENU

MENU lists available transactions from which the user selects a transaction identifier to start a transaction.

You activate MENU by entering a slash / (default) or any character string defined by in USE MENU FOR TRANSACTION-MENU clause in the TDS SECTION at TDSGEN.

A menu is displayed in the following format:

```
        p/n                                    tdsname

 1    message-id1                      explanatory-text1
 2    message-id2                      explanatory-text2
 .        .                                    .
 .        .                                    .
 .        .                                    .        -->__
 18   message-id18                     explanatory-text18
-:_____________________________________________________________
-:_____________________________________________________________
```

where:

- p is the menu number and n is the total number of menus,
- `tdsname` is the name of the TDS application,
- up to 18 lines describe the available transactions, each of which is numbered and includes information declared in the TRANSACTION SECTION at TDSGEN:
  - a message-id defined in the MESSAGE statement.
  - an explanatory-text defined by the `PROMPT` clause
- a two-character field prefixed by `-->` allows you
  - either to enter a number corresponding to the transaction if the transaction to be selected appears on the screen
  - or, if the transaction is not on the screen, to search for the transaction by manipulating scroll-up and scroll-down as follows:

    > or blank to move up the next page,

    < to move down the preceding page,

    > n or <n to move n pages up or down,

    `=n` to move to page number n (=0 moves to the last page),

    slash (/) or semicolon (;) to quit the menu.
- two blank lines prefixed by `-:` allow you to enter the message-id followed by parameters as in command mode.

14.2.2.4 PT

PT (pass through) allows you to connect to another application at the same site or on a different site, without having to log off from TDS.

PT opens a pass-through session during which the terminal is directly connected to the new application.

The terminal returns to TDS when the user logs off.

NOTE:

If PT is activated with no parameter, the user is connected to IOF on the local site.

This command may be modified by the TDS Administrator if he does not wish you to use the pass-through feature. At TDSGEN, the message-id can be assigned to another transaction by using the `MESSAGE PT ASSIGN TO` clause.

Syntax

```
PT [ destination-node]  [,APPL = { IOF              } ]
                                 { application-name }
   [ USER = userid ]
   [ PASSWORD = user-password ]
   [ PROJECT = project ]
   [ BILLING = billing ]
   [ STATION = station ]
   [ NEW ]
   [ NSTARTUP ]
   [ STR = ? OPTION (length of system header)%string]
   [ NTERMSG ]
```

Parameters

`Destination node` is a 4-character parameter and specifies the name of the remote application node. It is the only positional parameter. This means that it must be the first parameter.

`APPL` is an 8-character parameter specifying the mailbox name of the remote application. The default value is IOF.

`USER` is a 12-character parameter specifying the name of the user who will be connected to the remote application.

`PASSWORD` is a 12-character parameter specifying the password of the user who requests the connection. If USER is specified, then PASSWORD must be specified.

`PROJECT` is a 12-character parameter specifying the project of the user.

`BILLING` is a 12-character parameter specifying the billing of the user. If PROJECT is specified, BILLING must be specified.

`STATION` is an 8-character parameter specifying the name of the station.

`NEW/NSTARTUP` are specific parameters for IOF.

The `STR` parameter overrides service message headers and is used only in TDS. There are two choices for this parameter: a system header option with system header length or a string. The % is a delimiter that distinguishes the user string from the system header option. Here is an example of the system header option with system header length:

```
STR = ?A
```

The string is an alphanumeric field of up to eight characters that the first TPR of a LOGON receives. Here is an example of a string:

```
STR = ?A%string
```

For further information, see *the DNS Terminal Management Reference Guide*.

The NTERMSG parameter, if specified, is used by the PT transaction to prevent the message "SESSION TERMINATED BY REMOTE APPLICATION" being sent.

All the above parameters are subject to catalog limitations and rules.

14.2.2.5 Redisplaying the Last Message

When the terminal is in either command mode, or transaction mode, you can enter the `last message character` to request that TDS re-send the most recently displayed message to the user terminal. Note that only the last message character specified in TDSGEN is permitted and in transaction mode, this character must be the first and only character transmitted. The default is the question mark (?) or any single character defined in the USE lastmsg FOR LAST MESSAGE clause.

To list the transactions available, enter a slash (/) or any character defined in the USE menu FOR TRANSACTION-MENU clause in the TDS SECTION at TDSGEN.

14.3 Transaction Mode

In transaction mode, the user communicates directly with the transaction. All entries are addressed to the transaction and not to TDS.

There are no restrictions on the syntax of messages. However, messages from the terminal must not begin with `$*$`.

The terminal returns to the command mode when the transaction has terminated. The user is notified that the transaction is normally terminated either by the last message sent, or by READY.

If a transaction is interrupted because of a programming error, the following message is displayed:

```
ABORT (x.xx.xxxx.) RC=yyyyyyyy, zzzz (TPR name)
```

where:

(x.xx.xxxx.)	the address in the TPR at which the abort occurred.
yyyyyyyy	system integration unit
zzzz	return code of the relevant abort, see Table B-1.

Note that TRANSACTION-STORAGE and PRIVATE-STORAGE are not rolled back.

To interrupt a transaction in progress, Break can be issued at a terminal:

- by pressing the BREAK key,
- or by entering `$*$BRK`,
- or `esc tdsname` when you are the master terminal logged under IOF.

In addition to redisplaying the last message in command mode, you can also do this in transaction mode. In this mode, the `last message character` must be the first and only character that you enter before you press the TRANSMIT key.

❑

A. Trace Options and TDS-Authorized PCF Commands

This appendix describes the format for each option of the TRACE command.

FLOW

Gives the following information about a transaction:

```
BEGINNING OF TRANSACTION <messageid>;

TSN <transaction-serial-number>

BEGINNING OF TPR <tprname> PRSN : <tpr-serial-number>

RESTART STATUS : <restart-status>

NORMAL END OF A { TPR [WITH COMMITMENT] } [WAIT-TIME:xxx]
                { TRANSACTION           }
```

When TPR is specified, the transaction is still in progress. WAIT-TIME is specified only if the corresponding field in TDS-STORAGE is valid.

```
                  { TPR             }
ABNORMAL END OF A { TRANSACTION     }
                  { COMMITMENT UNIT }

 [AT:xxxxxxxx] G4=yyyyyyyy  System Integration Unit, return code
                            where xxxxxxxx is the address in the
                            program where the abort was detected.
```

If COMMITMENT UNIT is specified, it may be a functional abort (BUSY, BLKBUSY, DEADLOCK), the TPR will be restarted. Alternatively COMMITMENT UNIT may be specified for a user abort (ON-ABORT-TPR) in which case the specified program will be started.)

The following message indicates that a call "H_PPC_UBACK" WITH RESTORE=0 has been done in the TPR:

```
END OF {TPR        } WITH BACKOUT NO RESTART [WAIT-TIME:xxx]
       {TRANSACTION}
```

BREAK REQUEST

A Break signal was issued from the terminal that initiated the transaction.

MESG

Indicates each message sent and received

```
RECEIVE STATEMENT.END key : { EGI  } LENGTH:length  STATUS:status
                            { NONE }
```

followed by a dump of the message, if applicable.

There is no dump. If EGI is specified, the whole message is received, otherwise only a portion is received.

```
                           { NONE }
SEND STATEMENT INDICATOR : { ESI  } DESTINATION : destination
                           { EMI  } LENGTH : length
                           { EGI  } STATUS : status
```

followed by a dump of the message, if applicable.

If NONE, the message portion was sent without an end indicator and there is no dump. When ESI, EMI or EGI are specified, the whole message or segment is dumped, including the control characters generated by the line spacing requests.

RREC

Indicates the execution of each READ or START directed to a TDS-controlled file:

```
{ START }
{ READ  } [NEXT]   IFN : internal-file-name AT : xxxxxxxx
```

xxxxxxxx is the address of the instruction in the program. If NEXT is specified, the file will be accessed sequentially. If READ is specified, RREC is followed by a dump of the record.

MREC

Indicates each execution of a DELETE, WRITE or REWRITE directed to a TDS-controlled file:

```
{ DELETE  }
{ WRITE   } IFN : internal-file-name AT : xxxxxxxx
{ REWRITE }
```

where xxxxxxxx is the address of the instruction in the program. MREC is followed by a dump of the record (not DELETE).

TRST

TRANSACTION STORAGE

Indicates the completion of each TPR of the transaction and gives the context of the TRANSACTION-STORAGE.

DISP	Indicates the object on the execution of each DISPLAY.
IDS2	Activates the IDS/II traces as specified by the run-time options.

PCF

PCF facilities available in COBOL are as follows:

- only symbolic addressing mode is provided. It implies the TPR was compiled with DEBUG to generate the compile unit database,
- DEBUG must not be specified in the job,
- the transaction may have been declared for DEBUG at TDSGEN to invalidate or validate the database updates. This clause is independent of PCF debugging commands,
- as PCF commands apply to TPRs using shared code, serialization mechanisms are implemented for TPRs in debug mode. This allows a TPR to be shared by productive terminals and debugging terminals. These mechanisms apply at commitment level and mean that TPRs in debug mode execute alone at a specific time until the commitment unit terminates. This may lead to reduced throughput,
- if a command is to be added, changed or removed from the command file, all the commands must be reintroduced unless they are cataloged. The PCF EXEC command can be used to make the TDS-PCF interface operate better,
- Once a TPR is validated, it must be recompiled without the DEBUG to prevent unnecessary loading of the database.

Table A-1. TDS-Authorized PCF Commands (1/2)

Command	Explanation
APPLY (A)	A [command-number-list] IN procedure-name [.block]; The specified command applies to the named procedure or block.
CHANGE (C)	[command-number] C assignment [, assignment...] [AT action-point-list] [IF-clause]; Assign value to specified data items when control reaches specified action points.
DUMP (D)	[command-number] D data-expression [, data expression...] [AT action-point-list] [IF-clause]; Display specified data items when control reaches specified action points.
END	[command-number] END [DUMP] [AT action-point-list] [IF-clause]; Abort the execution at specified point.
EXEC (E)	[command-number] E "file-description" [AT action-point--list] [IF-clause]; Execute PCF commands from specified file.
GOTO (G)	[command-number] GO action-point [AT action-point-list] [IF-clause]; Force a transfer of control at specified point.
HELP (H)	H pcf-command-name; Display syntax of specified PCF command.
KILL (K)	[command-number] K [command-number-list] [AT action--point-list] [IF-clause]; Delete specified commands when control reaches specified action points.

Table A-1. TDS-Authorized PCF Commands (2/2)

Command	Explanation
LIST (L)	[command-list] L [command-number-list] [AT action-point-list] [IF-clause]; Display the specified commands when control reaches the specified action points.
OUTPUT (O)	[command-number] O "file-description" [AT action-point-list] [IF-clause]; Directs the PCF execution report to a specified file.
RECOVER	[command-number] RECOVER $\left\{\begin{array}{l}\text{ILLDEC}\\\text{SUBSCRIPT}\end{array}\right\}$ [AT action-point-list] [IF-clause]; Defines a recovery action for certain execution incidents.
RESUME (R) SUSPEND (S)	[command-number] $\left\{\begin{array}{c}\text{R}\\\text{S}\end{array}\right\}$ [command-number-list] [AT action-point-list] [IF-clause]; To activate/deactivate specified PCF commands when control reaches specified action points. Restricts debugging to a desired section of a program.
TRACE (T)	[command-number] T $\left\{\begin{array}{c}\text{NOT}\\\text{MAP}\end{array}\right\}$ [AT action-point-list] [IF-clause]; Trace flow of program through specified action point.
WHERE (W)	[command-number] W [$\left\{\begin{array}{l}\text{TRACK}\\\text{TK}\end{array}\right\}$] [AT action-point-list] [IF-clause] Display the context of specified action points.

- List of PCF commands which are not allowed in TDS are as follows:

  `BEGIN` only aborts the TPR

  `PAUSE` is ineffective.

- List of PCF commands which are not relevant to TDS are:

  ```
  CHECKPOINT
  FSE
  GO
  INLIB1
  INLIB2
  INLIB3
  LIB.
  ```

- For more information on syntax and semantics, refer to the *PCF User's Guide.*
- Example of PCF command file used for debugging a transaction:

  ```
  APPLY IN TPR1;
  DUMP TDS-STORAGE AT BEG-PROC;
  GOTO ABC AT ERROR-SECTION IF COUNT=1;
  RECOVER ILLDEC;
  APPLY IN TPR3;
  CHANGE NEXT-TPR="TPR5" AT EXIT-TPR;
  LIST AT EXIT-TPR;
  APPLY IN TPR5;
  TRACE MAP AT SEARCH-KEY;
  ```

Description of PCF Parameters

```
action-point         ::= { ?[[stn.]ste.]sra                             }
                         {                                              }
                         { label                                        }
                         {                                              }
                         { {LINE ! LN}  line-number                     }
                         {                                              }
                         { ILN number                                   }
                         {                                              }
                         { {POST_MORTEM ! POST-MORTEM ! P_M ! P-M}      }
                         {                                              }
                         { {ALL_LABELS  ! ALL-LABELS ! A_L ! A-L}       }
                         {                                              }
                         { {EACH_REF ! EACH-REF ! E_R ! E-R}  / sym-ref}

action-point-list    ::= action-point [TO action-point]
                         [,action-point [TO action-point]]...
assignment           ::= data-reference=  {data-expression ! constant}
bit-string-constant  ::= "bits" B

block                ::= { {LINE ! LN}  line-number                     }
                         {                                              }
                         { ILN number                                   }
                         {                                              }
                         { procedure-name [.procedure-name...]          }

character-constant   ::= "string" [C]
command-number       ::= number
command-number-list  ::= command-number [TO command-number]
                         [,command-number [TO command-number]]...
complex-constant     ::= (numeric-constant,numeric-constant)
```

Description of PCF Parameters

```
constant            ::= { numeric-constant     }
                        {                      }
                        { character-constant   }
                        {                      }
                        { hexadecimal-constant }
                        {                      }
                        { {.T. ! .F.}          }

data-expression     ::= { (data-expression)        }
                        {                          }
                        { constant                 }
                        {                          }
                        { symbolic-reference       }
                        {                          }
                        { effective-reference      }
                        {                          }
                        { semi-symbolic-re}ference }

                        [operator-data-expression...] [format]

displacement        ::= hexa6

effective-reference ::= { ?[[stn.]ste.]sra                           }
                        {                                            }
                        { ?KSTKi! ?STK(i[ {: ! TO} j])               }
                        {                                            }
                        { ?BRi  ! ?BR (i[ {: ! TO} j])               }
                        {                                            }
                        { ?GRi  ! ?GR (i[ {: ! TO} j])               }
                        {                                            }
                        { ?XRi  ! ?XR (i[ {: ! TO} j])               }
                        {                                            }
                        { ?SRi  ! ?SR (i[ {: ! TO} j])               }
                        {                                            }
                        { {?IC  ! ?TR ! ?ST ! ?SAM ! ?PSA ! ?PTV } }
                        { {                                      } }
                        { {?NBP ! ?LOC ! ?PAR ! ?COM ! ?SAV      } }

                        . displacement
```

Description of PCF Parameters

```
fcode                 ::= {B ! C ! DPS ! DPU ! DUSLO ! DUSLS ! DUSTS !

                          DUSTO ! DUU ! EFB ! FB ! H ! L ! LB ! LFB !

                          P ! RC ! SFB ! X}

file-description      ::= file-literal
                          (see the IOF Terminal User's Reference Manual,
                           Part I Section VI).

format                ::= # fcode [p [.q] ]

hexadecimal-constant ::= "hexa"  {X ! H}

IF-clause             ::= IF {data-expression ! /SEV ! /COUNT}

                        [{NOT ! ? }]

                        {= ! < ! > ! < ! > ! EQ ! NE ! GT ! LT ! GE ! LE}

                        data-expression

                        [ {AND ! & ! OR}                                    ]
                        [                                                   ]
                        [ {data-expression ! /SEV ! /COUNT}                 ]
                        [                                                   ]
                        [ [{NOT ! ^ }]                                      ]
                        [                                                   ]
                        [ {= ! < ! > ! < ! > ! EQ ! NE ! GT ! LT ! GE ! LE} ]
                        [                                                   ]
                        [ data-expression...                                ]

label                 ::= name

library-description  ::= lib-78
                          (see the IOF Terminal User's Reference Manual,
                           Part I Section VII).

line-number           ::= number [.number...] [*rank]

name                  ::= characters

number                ::= digits
```

Description of PCF Parameters

```
numeric-constant     ::= [{+ ! -}] [number] [.number]

                           [{E ! D ! Q} [{+ ! -}] number]

operator           ::= { {+ ! -}                                          }
                       {                                                  }
                       { [{NOT ! ? }]                                     }
                       {                                                  }
                       { {= ! < ! > ! < ! > ! EQ ! NE ! GT ! LT ! GE ! LE} }

pointer-constant   ::= [ / {DIRECT ! DT  !  INDIRECT ! IT} ]
                       [                                   ]
                       [ / {R0 ! R1 ! R2 ! R3}             ]

                       / [ [stn.] ste.] sra

procedure-name     ::= name

rank               ::= number

semi-              ::= { @name [.displacement] }
symbolic-reference     {                       }
                       { :isn [.offset]        }

sra                ::= hexa6
segment relative address

ste                ::= hexa2
segment table entry

stn                ::= hexa1
segment table number

subscript          ::= data-expression [{: ! TO} data-expression ]

sym-ref            ::= { name [.name...]            }
symbolic-reference     {                            }
                       { name [{OF ! IN}  name...]  }

                       [(subscript [,subscript...])]
```

❑

B. Explanation of the Abort Codes

NOTE:
Codes shown in brackets may be returned to the user or may appear in a dump: codes where no USE was specified are returned only in the TPR.

[0804]	ADDROUT	Address out of bounds
1853	ARGERR	COBOL SYMBOLIC-QUEUE within the output CD or device-name separated by "/" (1C Syntax).
1501	ARVIOL	Access rights violation. WRITE requested while in "statistical read".
1E04	BREAK	Break request from a terminal
1206	BUFNBOV	TDS buffer pool overflow (the number of buffers exceeds the number defined at TDSGEN) - switching to the Before Journal is not possible. Increase buffer pool size or add the before Journal.
1878	NOMATCH	The next program name is unknown
1223	ENTRYOV	Segment entry overflow - number of entries exceeds limit. Retry CLOSE DEASSIGN.

[1C0B]	EXHAUST	End-of-file already signaled
[1228]	FILEOV	File overflow due to record insertion
[0A06]	FLNAV	File is not available - must be recovered before it can be opened
0A05	FUNCNAV	Unavailable function for specified file, e.g. READ of user-journal file
1466	LOCKVIOL	The transaction tries to access files opened in statistical read mode, and this mode is not explicitly specified for it.
00D3	CANEVT	Transaction cancelled (see CANCELTX in the LOGON transaction
[1020]	INDUNKN	Index unknown - TDS error
[0A07]	ITMNAV	A program tries to modify a file protected by the Before Journal but the journal was suppressed by the "SUPPRESS BEFORE JOURNAL" clause
[0204]	KEYCHG	Attempt to change record key on REWRITE
[1807]	LNERR	Erroneous length for variable-length record
[1C04]	NOCURREC	A sequential file request preceded a direct request
[1C19]	NODELETE	No delete is allowed

0115	NOTALL	Set by the M CANCEL command when execution was incomplete. Repeat the command.
1C01	NOTOPEN	The referenced file is not open
[0700]	NOWAIT	Simultaneous requests on a file have occurred, in violation of non-concurrency
1730	OPERATOR	Transaction cancelled by the M CANCEL command
1806	OPTERR	Attempt to execute a transaction with the SUPPRESS BEFORE JOURNAL clause while files are SHARED and integrity is set to HIGH or MEDIUM
1463	PMDVIOL	Processing-mode violation
04C5	PROCEXP	Procedure exception within a program
1021	QUNKN	SYMBOLIC-QUEUE within input CD is incorrect
1623	RESVIOL	Result of a resource handling violation, e.g. FREE/SAVE-COMMON function performed without TAKE-COMMON, violation of programming rules in transaction for inquiry, mono-phase transaction, or transaction aborted due to TDS supervisor with ORACLE in synchronous mode
1442	RCVVIOL	Violation when wrong RECEIVE
0A18	SCIDXNAV	Secondary index of UFAS indexed file was not created

1214	SFNBOV	Too many users in TRACE Print Mode - DIRSIZE parameter of the "<tdsname>-DEBUG" file is too small to accommodate all users
1441	SNDVIOL	Violation when using the SEND verb
1219	TABOV	The transaction tried to lock more pages than currently allocated. The transaction will be automatically restarted with a higher number of locked pages allowed
0027	DATALIM	Not enough space for saving file currencies in the swap-file buffer; this lack of space may cause this return code to occur: • when unmapping is performed and no commitment point is performed (see Chapter 13), • on a commitment point if the CALL "KEEP-CURRENCIES" procedure has been called for the files protected by the before Journal. (Sizing the swap-file buffer is discussed in the *TDS Administrator's Guide*).
0029	TIMELIM	TPR time limit exceeded
[1E05]	UNRECIO	Irrecoverable I/O error
0980	USERREQ	Abort requested by user
120F	COUNTOV	The transaction requires more locked pages than declared at TDSGEN (see the MAXIMUM NUMBER OF LOCKED PAGES clause in the TDS SECTION)
186B	UBUGERR	IDS/II user error: see detailed explanation in JOR or TRACE. When a TPR aborts with this code, the message is printed in the JOR, and the trace file when the terminal is in TRACE mode

```
1821      IFNERR
```

Occurs when a CALL "KEEP-CURRENCIES" statement was used in a previous commitment unit for a file which has since been closed

IDS/II TRACE

The format of the trace message when IDS/II is used under TDS is as follows:

```
PGID: <tprname> SLN: <internal-line-no> { ERR | WNG } :
                                        <error-no> <text>
```

where:

<tprname>	is a 30-character string containing the program identification as specified in the PROGRAM-ID clause. If this information is not available or is irrelevant, asterisks replace it.
<internal-line-no>	is a 7-digit decimal number identifying the source internal line number where the statement involved is coded. If this information is not available or is irrelevant, 0 replaces it.
ERR/WNG	indicate a fatal error or a warning respectively.
<error-no>	is a 4-digit decimal number. The meaning of each error number is described below.
<text>	describes the error. Though these texts are made as explicit as possible, further information may be found in the *Messages and Return Codes Directory.*

Error Number	Description
0402	Mismatch between schema name or date in TPR and object file. The TPR was not compiled with the same object schema as the one assigned to the TDS.
0516 0530 0532	Check that the TPR is programmed according to TDS/IDS rules.
0896 to 0904	DML inconsistency. Contact the Service Center.
1792 to 1811	DML/schema inconsistency: check for consistency between TPR and schema. Possibly the TPR was not recompiled after a schema modification or an internal error.

Further information on this subject may be found in the *IDS/II Reference Manual.*

❑

C. COBOL Example Using Forms

This appendix is an example of a transaction with COBOL using FORMS.

The form is called SALES and is used for sales. If data is entered incorrectly, the fields of the corresponding line are "highlighted" and the data can be re-entered. The program calculates the total sales value for each product and a grand total for the sales of all products.

The form image of SALES is as follows:

```
#                                  PRODUCT TYPE     ##########

NAME OF REPRESENTATIVE   ###########################  CODE  ###


PRODUCT           PRICE            QUANTITY          CODE        TOTAL

######     :      ####      :            ###     :    ####   :  #######
######     :      ####      :            ###     :    ####   :  #######
######     :      ####      :            ###     :    ####   :  #######
######     :      ####      :            ###     :    ####   :  #######
######     :      ####      :            ###     :    ####   :  #######
######     :      ####      :            ###     :    ####   :  #######
######     :      ####      :            ###     :    ####   :  #######

                                          GRAND TOTAL    ########
SELECTION FIELD
(IF / = STOP)
```

TDS-name	PRFM is the name of the TDS application
Transaction	TCDE is the name of the transaction assigned to the TPR named TCD1
TCD1 (TPR)	Display form SALES on the screen
TCD2 (TPR)	Enter data, calculate grand total; if erroneous data re-run TCD2
TCD3 (TPR)	Clear all NF (not protected) fields then return to TCD2

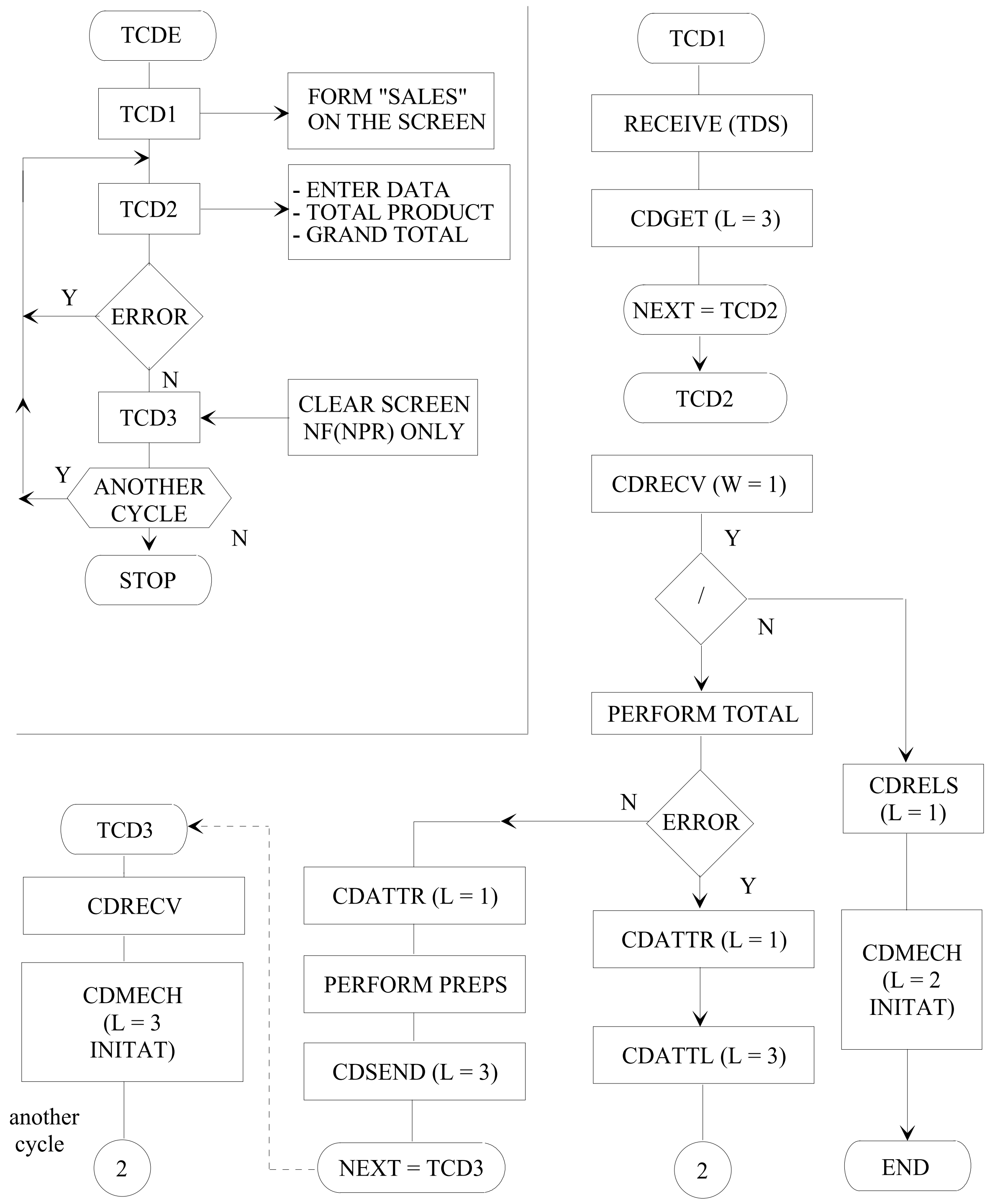


Figure C-1. Flowchart For Transaction TCDE (TPRs TCD1, TCD2, TCD3)

Screen Definition of form SALES

```
      SALES                            FROM : FASL
      CD=01/27/86  CT=17:33    ND=01/27/86 MT=17/33  SL=CBX NN=01

  NF SLASH LINE IS 01 COL IS 02 SCREEN-PIC IS X(1) UL   .
  NL SALE-TITLE LINE IS 01 COL IS 37 SCREEN-PIC IS X(10) UL   .
  NF REP-NAME LINE IS 04 COL COL IS 29 SCREEN-PIC IS X(27) UL   .
  NF REP-CODE LINE IS 4 COL IS 64 SCREEN-PIC IS X(6) UL   DI   .
  ARRAY SALE-LINES OCCURS 7.
  NF PRODUCT LINE IS 07 COL IS 05 SCREEN-PIC IS X(6) UL  BCYA .
  NF PRODUCT-CODE LINE IS 07 COL IS 48 SCREEN-PIC IS X(4) DI   BMAG .
  NF QUANTITY LINE IS 07 COL IS 61 SCREEN-PIC IS 9(3) DI   BYEL .
  NF PRICE LINE IS 7 COL IS 20 SCREEN-PIC IS ZZ9.99 NU   BBLU .
  NF TOTAL LINE IS 07 COL IS 61 SCREEN-PIC IS ZZZ9.99 PR   .
  UF LINE IS 07 COL IS 03 VALUE IS "%" CSPS .
  UF LINE IS 07 COL IS 15 VALUE IS "%" CSPS .
  UF LINE IS 07 COL IS 29 VALUE IS "%" CSPS .
  UF LINE IS 07 COL IS 44 VALUE IS "%" CSPS .
  UF LINE IS 07 COL IS 57 VALUE IS "%" CSPS .
  UF LINE IS 07 COL IS 69 VALUE IS "%" CSPS .
  UF LINE IS 08 COL IS 03 VALUE IS "#***********/*************/" CSPS .
  END-ARRAY.
  NF GRAND-TOTAL LINE IS 24 COL IS 50 SCREEN-PIC IS ******9.99 PR   FRED.
  UF LINE IS 01 COL IS 23 VALUE IS "PRODUCT TYPE" RV   .
  UF LINE IS 04 COL IS 04 VALUE IS "NAME"  .
  UF LINE IS 04 COL IS 09 VALUE IS "OF"  .
  UF LINE IS 04 COL IS 12 VALUE IS "REPRESENTATIVE" .
  UF LINE IS 04 COL IS 58 VALUE IS "CODE" .
  UF LINE IS 05 COL IS 03 VALUE IS
  "#***********.*************.**************.************.***********,"
  CSPS FMAG  .
  UF LINE IS 06 COL IS 03 VALUE IS "%" CSPS .
  UF LINE IS 06 COL IS 05 VALUE IS "PRODUCT" .
  UF LINE IS 06 COL IS 15 VALUE IS "%" CSPS
  UF LINE IS 06 COL IS 19 VALUE IS "PRICE" .
  UF LINE IS 06 COL IS 29 VALUE IS "%" CSPS .
  UF LINE IS 06 COL IS 31 VALUE IS "QUANTITY" .
  UF LINE IS 06 COL IS 44 VALUE IS "%" CSPS .
  UF LINE IS 06 COL IS 49 VALUE IS "CODE" .
  UF LINE IS 06 COL IS 57 VALUE IS "%" CSPS .
  UF LINE IS 06 COL IS 62 VALUE IS "TOTAL" .
  UF LINE IS 06 COL IS 69 VALUE IS "%" CSPS .
  UF LINE IS 24 COL IS 36 VALUE IS "GRAND TOTAL" BGRE.
```

TCDE: Structures SALES (I, V, R) used in the TPRs TCD1, TCD2, TCD3

```
             SALESI                              FROM : FASL
             CD=01/03/86  CT=21:31       MD=01/27/86  MT=17:43
*
02 FILLER PIC X VALUE "3".
02 FILLER PIC X(8) VALUE "SALES".
02 SALES-NO PIC 9(3) VALUE ZERO.
02 SALES-MD PIC X VALUE "A".
02 SALES-OF PIC X(8) VALUE SPACES.
02 SALES-OO PIC 9(3) VALUE ZERO.
02 SALES-LL PIC 9(3) VALUE ZERO.
02 FILLER PIC X(5) VALUE "86003"
02 FILLER COMP-1 VALUE 0252.
02 SALES-AF PIC 9 VALUE 1.
02 SALES-SL PIC 9(3) VALUE 1.
02 SALES-SC PIC 9(3) VALUE 1.
```

```
   11   02          SALESV                              FROM : FASL
              CD=01/03/86 CT=21:31        MD=01/27/86  MT=17:43
02 SALESV.
     03 FILLER PIC X VALUE ""2"".
     03 FILLER COMP-1 VALUE 41.
     03 FILLER PIC X(8) VALUE "SALES".
     03 SALES-NO PIC 9(3) VALUE ZERO.
     03 SALES-V.
          04 SALES-FC-V PIC X.
          04 SLASH-V PIC X.
          04 SALES-TITLE-V PIC X.
          04 REP-NAME-V PIC X.
          04 REP-CODE-V PIC X.
          04 SALE-LINES-AV.
               05 SALE-LINES-V OCCURS 7.
                    06 PRODUCT-V PIC X.
                    06 PRODUCT-CODE-V PIC X.
                    06 QUANTITY-V PIC X.
                    06 PRICE-V PIC X.
                    06 TOTAL-V PIC X.
          04 GRAND-TOTAL-V PIC X.
```

```
             SALES-R                             FROM : FASL
             CD=01/03/86 CT=21.31        MD=01/27/86  MT=17.43

          04 SALES-FC       PIC 99.
          04 SLASH          PIC X(1).
          04 SALE-TITLE     PIC X(10).
          04 REP-NAME       PIC X(27).
          04 REP-CODE       PIC X(6).
          04 SALES-LINES-A .
```

```
            05 SALES-LINES OCCURS 7.
                 06 PRODUCT PIC X(6).
                 06 PRODUCT-CODE PIC X(4).
                 06 QUANTITY PIC 9(3).
                 06 PRICE PIC 999V99.
                 06 TOTAL PIC 9999V99.
       04 GRAND-TOTAL     PIC 9999999V99.
```

TCD1: Display form SALES on the screen

```
                            TCD1                          FROM : FASL
                            CD=01/02/86         ND=01/12/85 MT=13:27

 10    IDENTIFICATION DIVISION.
 20    PROGRAM-ID. TCD1.
 30    AUTHOR. CHEGMA-ROBBE.
 40    ENVIRONMENT DIVISION.
 50    CONFIGURATION SECTION.
 60    SOURCE-COMPUTER. LEVEL-64.
 70    OBJECT-COMPUTER. LEVEL-64.
 80    DATA DIVISION.
 90    WORKING-STORAGE SECTION.
100    01  W-ARGUMENTS.
110      02  LEVEL     PIC X.
120      02  WAIT      PIC 9.
130      02  MECH      PIC X(6).
140    01  ZMSG.
150      02  ZMES      PIC X(20).
160    01  Z-PAR.
170      02  TRAN      PIC X(4).
180      02  WTIME     PIC X(3).
190      02  WIND      PIC X.
200   *
210    01  SALESI.
220        COPY SALESI.
230   *
240    LINKAGE SECTION.
250        COPY TDS-STORAGE.
260        COPY CONSTANT-STORAGE.
270   *
280    01  TRANSACTION-STORAGE.
290      02  TS-PRI-ST.
300        03  TS-WTIME  PIC 999.
310        03  TS-WIND   PIC 9.
320        03  FILLER    PIC X(226).
330      02  TS-REC      PIC X(223).
340   **
```

```
 COMMUNICATION SECTION.
 CD  CDIN  INPUT
     SYMBOLIC QUEUE        ISQ
     MESSAGE DATE          IMD
     MESSAGE TIME          IMT
     SYMBOLIC SOURCE       ISS
     TEXT LENGTH           ITL
     END KEY               IEK
     STATUS KEY            ISK
     MESSAGE COUNT         IMC.
 01  CDI    PIC X(87).
 CD  CDOUT OUTPUT
     DESTINATION COUNT     ODC
     TEXT LENGTH           OTL
     STATUS KEY            OSK
     ERROR KEY             OEK
     SYMBOLIC DESTINATION  OSD.
 01  CDO      PIC X(23).
/*
 PROCEDURE DIVISION USING
     TDS-STORAGE CONSTANT-STORAGE TRANSACTION-STORAGE.
  D10-START.
     MOVE SPACES TO CDI CDO ZMSG.
     MOVE 0 TO TS-WIND WAIT.
     MOVE SYMBOLIC-QUEUE TO ISQ.
     RECEIVE CDIN MESSAGE INTO ZMSG.
     IF ISK NOT = "00" GO DISP-ISK.
     UNSTRING ZMES DELIMITED BY ALL " " OR ","
              INTO TRAN WTIME WIND.
     IF WIND NUMERIC AND WTIME NUMERIC
              MOVE WIND TO TS-WIND  MOVE WTIME TO TS-WTIME.
     IF WIND = 1   MOVE TS-WTIME TO WAIT-TIME.
*
     MOVE ISS TO OSD.
     MOVE 1 TO ODC OTL.
     MOVE 3 TO LEVEL.
     CALL "CDGET" USING CDO SALESI LEVEL.
     IF OSK = "A4" GO DISP-OSK.
     MOVE "TCD2" TO NEXT-TPR.
 F90-FIN.
*
     EXIT PROGRAM.
 DISP-ISK.
     DISPLAY "CDI-ISK=" ISK UPON SYSOUT. GO F90-FIN.
 DISP-OSK.
     DISPLAY "CDO-OSK=" OSK UPON SYSOUT. GO F90-FIN.
```

TCD2: Enter data - total by product - grand total

```
                           TCD2                    FROM : FASL
                           CD=01/03/86 CT=16:43  ND=01:12:86 MT=19.04

 IDENTIFICATION DIVISION.
 PROGRAM-ID. TCD2.
 AUTHOR. CHEGMA-ROBBE.
 INSTALLATION. GAMBETTA CENTER CENTER.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. LEVEL-64.
 OBJECT-COMPUTER. LEVEL-64.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 77  I                PIC  999.
 77  J                PIC  9.
 77  W-I              PIC 9.
 77  ERROR-INDICATOR  PIC 9.
 77  W-NBF            PIC 9.
 77  TOTAL-MAN        PIC 9(4)V99.
 77  TOTAL-NUM        PIC 9(7)V99.
 01  SALES-SW.
     COPY  SALESV REPLACING TRAILING "-V" BY "-W".
 01  W-ARGUMENTS.
   02  LEVEL    PIC X.
   02  WAIT     PIC 9.
   02  ATTR     PIC X(4)   VALUE "INIT".
   02  MECH     PIC X(6).
 01  ATTL.
   02  FILLER   PIC 999.   VALUE 2.
   02  FILLER   PIC X(4)   VALUE "CP".
   02  FILLER   PIC X(4)   VALUE "ML".
 01  ZMSG.
   02  ZMES     PIC X(30).
   02  TRAN     PIC X(4).
*
 01  SALES-SV.
     COPY SALESV.
*
 LINKAGE SECTION.
     COPY TDS-STORAGE.
     COPY CONSTANT-STORAGE.
*
 01  TRANSACTION-STORAGE.
   02  TS-PRI-ST.
     03  TS-WTIME  PIC 999.
     03  TS-WIND   PIC 9.
     03  FILLER    PIC X(226).
```

```
450      02  SALESR
460          COPY SALESR.
470    **
480     COMMUNICATION SECTION.
490     CD  CDIN INPUT.
500         SYMBOLIC QUEUE            ISQ
510         MESSAGE DATE              IMD
520         MESSAGE TIME              IMT
530         SYMBOLIC SOURCE           ISS
540         TEXT LENGTH               ITL
550         END KEY                   IEK
560         STATUS KEY                ISK
570         MESSAGE COUNT             IMC.
580     01  CDI PIC X(87).
590     CD  CDOUT OUTPUT
600         DESTINATION COUNT         ODC
610         TEXT LENGTH               OTL
620         STATUS KEY                OSK
630         ERROR KEY                 OEK
640         SYMBOLIC DESTINATION      OSD.
650     01  CDO      PIC X(23).
660    /*
670     PROCEDURE DIVISION USING
680         TDS-STORAGE CONSTANT-STORAGE TRANSACTION-STORAGE.
690     D10-START.
700         MOVE SYMBOLIC-QUEUE TO ISQ.
710         MOVE 1 TO ODC.
720         MOVE "3" TO LEVEL.
730         MOVE ALL "S" TO SALES-W.
740         MOVE TS-WIND TO WAIT.
750         IF WAIT = 1 MOVE TS-WTIME TO WTIME.
760     D20-LOOP.
770         MOVE SPACES TO SALESR
780         MOVE ALL "S" TO SALES-V.
790         CALL "CDRECV" USING CDI SALESR WAIT SALES-SV.
800         MOVE ISS TO OSD.
810         IF  SLASH-V = "R" AND SLASH = "/"
820             MOVE "INITAT" TO MECH
830             MOVE SPACES TO ZMSG
840             MOVE "1" TO LEVEL  CALL "CDRELS" USING CDO LEVEL
850             MOVE "2" TO LEVEL  CALL "CDMECH" USING CDO MECH LEVEL
860    **       SEND CDOUT FROM ZMSG EMI
870             MOVE SPACES TO NEXT-TPR GO F60-END.
880         MOVE SPACES TO SALE-TITLE-V SLASH-V REP-CODE-V REP-NAME-V.
890         MOVE 0 TO ERROR-INDICATOR.
900         MOVE ZERO TO TOTAL-NUM
910         PERFORM TOTAL-O THRU TOTAL-F
920                 VARYING I FROM 1 BY 1 UNTIL I > 7.
930         IF ERROR-INDICATOR = 0 GO Y40-RIGHT.
940     N30-ERROR.
```

```
     MOVE "1" TO LEVEL
     CALL "CDATTR" USING CDO SALES-SW ATTR LEVEL
     MOVE "3" TO LEVEL
     MOVE SPACES TO GRAND-TOTAL-V
     CALL "CDATTL" USING CDO SALES SV ATTL LEVEL
     MOVE SALES-V TO SALES-W
     MOVE "TCD2" TO NEXT-TPR.
     GO F60-END.
 T40-RIGHT.
     MOVE "1" TO LEVEL
     MOVE ALL "S" TO SALES-W
     CALL "CDATTR" USING CDO SALES-SW ATTR LEVEL
     PERFORM PREP-SEND THRU END-PREP VARYING J
     FROM 1 BY 1 UNTIL J > 7
     MOVE "S" TO GRAND-TOTAL-V
     MOVE TOTAL-NUM TO GRAND-TOTAL
     MOVE "3" TO LEVEL
     CALL "CDSEND" USING CDO SALESR LEVEL SALES-SV.
 F50-LOOP.
     MOVE "TCD3" TO NEXT-TPR.
 F60-END.
     EXIT PROGRAM.
/*
 TOTAL-D.
     IF PRODUCT-V (I) NOT = "R" GO TO TOTAL-NR.
     IF PRODUCT-CODE-V (I) = "R"
     AND QUANTITY-V (I) = "R"
     AND PRICE-V (I) = "R"
     MULTIPLY QUANTITY (I) BY PRICE (I) GIVING TOTAL-MAN
     MOVE TOTAL-MAN TO TOTAL (I)
     ADD TOTAL-MAN TO TOTAL-NUM
     MOVE SPACES TO SALE-LINES-V (I)
     GO TO TOTAL-F.
 TOTAL-NR.
     IF PRODUCT-CODE-V (I) = "S"
     AND QUANTITY-V (I) = "S"
     AND PRICE-V (I) = "S"
     MOVE SPACES TO SALE-LINES-V (I)
     GO TO TOTAL-F.
 TOTAL-ERROR.
     MOVE 1 TO ERROR-INDICATOR
     MOVE ALL "S" TO SALE-LINES-V (I).
 TOTAL-F.
     EXIT.
****
 PREP-SEND.
```

```
    IF SALE-LINES (J) = SPACES
    MOVE SPACES TO SALE-LINES-V (J)
    GO TO END-PREP.
    MOVE ALL "S" TO SALE-LINES-V (J)
END-PREP.
    EXIT.
```

TCD3: Clear all NF (NPR) - if another cycle go to TCD2

```
                              TCD3                          FROM : FASL
                              CD=01/03/86 CT=18:38    MD=01/12/86 MT=18:32

 IDENTIFICATION DIVISION.
 PROGRAM-ID.  TCD3.
 AUTHOR. CHEGNA-ROBBE.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. LEVEL-64.
 OBJECT-COMPUTER. LEVEL-64.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  W-ARGUMENTS.
     02  LEVEL    PIC X.
     02  WAIT     PIC X.
     02  MECH     PIC X(6).
 01  ZMSG.
     02  ZMES   PIC X(30).
     02  TRAN   PIC X(4).
*
 01  SALES-VR.
     COPY SALES-V.
   02 SAES-R.
     COPY SALESR.
*
 LINKAGE SECTION.
     COPY TDS-STORAGE.
     COPY CONSTANT-STORAGE.
*
 01  TRANSACTION-STORAGE.
   02  TS-PRI-RT.
     03   TS-WTIME  PIC 999.
     03   TS-WIND   PIC 9.
     03   FILLER    PIC X(226).
   02  TS-REC       PIC X(223).
**
 COMMUNICATION SECTION.
 CD  CDIN INPUT
     SYMBOLIC QUEUE    ISQ
     MESSAGE DATE      IMD
     MESSAGE TIME      IMT
     SYMBOLIC SOURCE   ISS
     TEXT LENGTH       ITL
     END KEY           IEK
     STATUS KEY        ISK
     MESSAGE COUNT     IMC.
 01  CDI     PIC X(87).
 CD  CDOUT  OUTPUT
     DESTINATION COUNT ODC
     TEXT LENGTH       OTL
     STATUS KEY        OSK
     ERROR KEY         OEK
     SYMBOLIC DESTINATION  OSD.
 01  CDO     PIC X(23).
/*
 PROCEDURE DIVISION USING
     TDS-STORAGE CONSTANT-STORAGE TRANSACTION-STORAGE.
 D10-START.
     MOVE SPACES TO CDI.
     MOVE SYMBOLIC-QUEUE TO ISQ.
     MOVE 1 TO ODC.
     MOVE 3 TO LEVEL.
     CALL "CDRECV" USING CDI SALESR WAIT SALESV.
     MOVE ISS TO OSD.
     MOVE "INITAT" TO MECH.
     CALL "CDMECH" USING CDO MECH LEVEL.
     MOVE "TCD2" TO NEXT-TPR.
 F90-END.
     EXIT PROGRAM.
```

TDSC-12TCD1 - DATA ENTRY

```
                        PRODUCT TYPE FORMS-TDS

NAME OF REPRESENTATIVE  RS                                  CODE 100

PRODUCT         PRICE          QUANTITY        CODE        TOTAL
  IOF           200              10             1

  GCL           400              20             2

  FORMS         500              10             3
```

TCD2- TOTAL BY PRODUCT THEN GRAND TOTAL

```
                        PRODUCT TYPE FORMS-TDS

NAME OF REPRESENTATIVE  RS                                  CODE 100

PRODUCT         PRICE          QUANTITY        CODE        TOTAL
  IOF           200.00           010            1          2000.00

  GCL           400.00           020            2          8000.00

  FORMS         500.00           010            3          5000.00

                               GRAND TOTAL  **15000.00
```

TCD3- CLEAR SCREEN NF (NPR) - IF "/" TOTAL CLEAR -STOP

```
                          PRODUCT TYPE

NAME OF REPRESENTATIVE                                     CODE

PRODUCT         PRICE          QUANTITY        CODE        TOTAL

                                                            2000.00

                                                            8000.00

                                                            5000.00
                                   GRAND TOTAL  **15000.00
```

❑

D. Example of SUBJOB

This Appendix is divided into two parts. The first part shows how a job is submitted through a TPR and the second part shows how a GCL procedure is submitted.

Assume that a user wishes to submit a job by means of a TPR. The job is contained in the sub-file named C. This file is stored in the uncataloged JCSL library that resides on the MS/D500 disk volume BD112.

Contents of the subfile:

```
COBOL SOURCE=&1  INLIB=(&2 &3) &XR  LEVEL=NSTD;

$ENDJOB
```

The TPR is written as follows:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SDP11.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 IND-ESI PIC 9 VALUE 1.
01 IND-EGI PIC 9 VALUE 3.
01 DESCRPT-JOB.

  02 JOB-LONG-STRUCTURES COMP-1 VALUE 61.
  02 JOB-CLASS PIC X VALUE SPACE.
  02 JOB-PRIORITY PIC 9 VALUE 7.
  02 JOB-SWITCHES VALUE "00000000000000000000000000000000".
     03 SW PIC 9 OCCURS 32 INDEXED BY I.
  02 JOB-DELETE PIC X VALUE "N".
  02 JOB-HOST PIC X(24) VALUE SPACE.
  02 PARAMETERS.
   03 LONG-STRUCTURE COMP-1 VALUE 54.
   03 DESCR-PARAMETERS
    04 NB-POSITIONALS COMP-1 VALUE 3.
    04 NB-KEYWORDS COMP-1 VALUE 1.
    04 DEFINITION-PARAMETERS.
     05 POSITIONAL-1 COM-1 VALUE 6.6---------|
     05 VALUE-1 PIC X(6) VALUE "UT4V00".<----|
     05 POSITIONAL-2 COMP-1 VALUE 4.------|
     05 VALUE-2 PIC X(4) VALUE "JBSL".<---|
     05 POSITIONAL-3 COMP-1 VALUE 20.---------------------|
     05 VALUE-3 PIC X(20) VALUE "DVC=MS/D500 MD=BD111".<--|
     05 KEYWORDS.
      06 KW-LONG COMP-1 VALUE 4.
      06 KW-NAME PIC X(8) VALUE "XR".
      06 KW-VALUE PIC X(4) VALUE "XREF".
01 STAT.
  02 RESULT PIC 9.
  02 FILLER COMP-2.
01 JOB-START.
  02 LENGTH-FOLLOWING COMP-1 VALUE 20.--------------------------|
  02 JOB-COORDINATES PIC X(20) VALUE "C:JCSL:BD112:MS/D500".<---|
                                       |--------------------
                                       |
                                       V
                                     sfn:efn:md:dvc (uncataloged)

01 MESS-ERR.
  02 FILLER PIC X(11) VALUE "ERROR SKO=".
  02 SKO-ERR PIC XX.
  02 FILLER PIC X(8) VALUE "LEVEL".
  02 LEV-ERR PIC XX.
  02 FILLER PIC X(10) VALUE "VECTORS=".
  02 VECTOR PIC X(12).
01 ZRECEP PIC X(6).
01 AIG PIC 9 VALUE 0.
01 MEMBERI.
   COPY G-COMCORI.
01 MEMBERV.
   COPY G-COMCORV.
01 MEMBERR.
   COPY G-COMCORR.
LINKAGE SECTION.
COPY TDS-STORAGE.
COPY CONSTANT-STORAGE.
01 TRANSACTION-STORAGE PIC X(256).
COMMUNICATION SECTION.
CD CDIN FOR INPUT
   SYMBOLIC QUEUE              SQI
   MESSAGE DATE                MDI
   MESSAGE TIME                MTI
   SYMBOLIC SOURCE             SSI
   TEXT LENGTH                 TLI
   END KEY                     EKI
   STATUS KEY                  SKI
   MESSAGE COUNT               MCI.
01 CDIN-R PIC X(87).
CD CDOUT FOR OUTPUT
   DESTINATION COUNT           DCO
   TEXT LENGTH                 TLO
   STATUS KEY                  SKO
   ERROR KEY                   EKO
   SYMBOLIC DESTINATION        SDO.
01 CDOUT-R.
   02 FILLER PIC 9(4) VALUE 1.
   02 FILLER PIC 9(4) VALUE 1.
   02 FILLER PIC X(15).

PROCEDURE DIVISION USING TDS-STORAGE CONSTANT-STORAGE
TRANSACTION-STORAGE.

START.
   MOVE SYMBOLIC-QUEUE TO SQI.
   RECEIVE CDIN MESSAGE INTO ZRECEP.
   IF EKI = ZERO MOVE 1 TO AIG GO TO START.
   MOVE SSI TO SDO.
   IF AIG = 1 MOVE "MESSAGE TOO LONG. RESTART TRANSACTION"
            TO MESS-ERR
              GO TO SEND-ERROR.
   DISPLAY "CALL SUBJOB PENDING" UPON TERMINAL.
   MOVE ZERO TO JOB-SWITCHES.
```

```
  CALL "SUBJOB" USING DESCRPT-JOB STAT JOB-START.
```

```
   IF RESULT NOT=ZERO
      DISPLAY "OH WELL.. RESULT= " RESULT UPON TERMINAL
                      MOVE SPACES TO NEXT-TPR GO TO TPR-END.

SEND-FORM.

******************************************************************
*                                                                *
*          ACTIVATE FORM AND REQUEST ENTRY OF CUSTOMER NUMBER    *
*                                                                *
******************************************************************

CALL "CDGET" USING CDOUT-R MEMBERI IND-ESI.
IF SKO = "AG" DISPLAY "ERROR DATE..TO BE RECOMPILED" UPON CONSOLE
              MOVE SPACES TO NEXT-TPR GO TO TPR-END.
IF SKO NOT="OO" AND NOT="AB" MOVE "10" TO LEV-ERR PERFORM ERR-SKO.
MOVE SPACES TO G-COMCOR-V.
MOVE "S" TO G-L24-V.
MOVE "ENTER CUSTOMER NO" TO G-L24.
CALL "CDSEND" USING CDOUT-R MEMBERR IND-EGI MEMBERV.
IF SKO NOT="00" AND NOT="AB" MOVE "11" TO LEV-ERR
                             GO TO ERR-SKO.
GO TO TPR-END.
```

The following listing shows the results of submitting the job.

```
JOBID + C                    USER = KEHOE             PROJECT = PK


<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-


12:10:46                                            JOB INTRODUCED FROM
                                                        C JBSL BD11

<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-<-


12:10:49  START OF TRANSLATION

**       PARAMETER VALUES                      **

&1       = UT4V00
&2       = JBSL
&3       = DVC=MS/D500 MD=BD111
&XR      = XREF
         $JOB C USER=KEHOE;

WARNING  150 THIS STATEMENT HAS BEEN GENERATED

  COBOL SOURCE=&1 INLIB=(&2 &3) &XR LEVEL=L64;

          RECORD COUNT:   1

12:10:49  END OF TRANSLATION

SUBMITTING A GCL PROCEDURE

IDENTIFICATION DIVISION.
PROGRAM-ID. SDP11.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 IND-ESI PIC 9 VALUE 1.
01 IND-ESI PIC 9 VALUE 3.
01 DESCRPT-JOB.

  02 JOB-LONG-STRUCTURES COMP-1 VALUE 61.
  02 JOB-CLASS PIC X VALUE SPACE.
  02 JOB-PRIORITY PIC 9 VALUE 7.
  02 JOB-SWITCHES.
     03 SW PIC 9 OCCURS 32 INDEXED BY I.
  02 JOB-DELETE PIC X VALUE "N".
  02 JOB-HOST PIC X(24) VALUE SPACE.
```

```
    02 PARAMETERS.
     03 LONG-STRUCTURES COMP-1 VALUE 49.

     03 DESCR-PARAMETERS.
      04 NB-POSITIONALS COMP-1 VALUE 3.

      04 NB-OF-KEYWORDS COMP-1 VALUE 0.
      04 DEFINITION-PARAMETERS.
       05 POSITIONAL-1 COMP-1 VALUE 24.     ---------------------|
       05 VALUE-1 PIC X(24) VALUE "MWINLIB BIN SYS.HBINLIB;"<----|
       05 POSITIONAL-2 COMP-1 VALUE 3 .   -----------|
       05 VALUE-2 PIC X(3) VALUE "DJ ".     <--------|
       05 POSITIONAL-3 COMP-1 VALUE 12.               ------------|
       05 VALUE-3 PIC X(12) VALUE "JOBID = x114".   <----------|

01 STAT.
  02 RESULT PIC 9.
  02 FILER COMP-2
01 JOB-START.                                 -----------------------|
  02 LENGTH-FOLLOWING COMP-1 VALUE 20.                        <-----|
  02 JOB-COORDINATES PIC X(20) VALUE "SYS.HSLLIB..ABSENTEE".
01 MESS-ERR.
  02 FILLER PIC X(11) VALUE "ERROR SKO=".
  02 SKO-ERR PIC XX.
  02 FILLER PIC X(8) VALUE "LEVEL".
  02 LEV-ERR PIC XX.
  02 FILLER PIC X(10) VALUE "VECTORS=".
  02 VECTOR PIC X(12).
01 ZRECEPT PIC X(6).
01 AIG PIC 9 VALUE 0.
01 MEMBERI.
   COPY G-COMCORI.
01 MEMBERV.
   COPY G-COMCORV.
01 MEMBERR.
COPY G-COMCORR.
LINKAGE SECTION
COPY TDS-STORAGE.
COPY CONSTANT-STORAGE.
01 TRANSACTION-STORAGE PIC X(256).
COMMUNICATION SECTION.
CD CDIN FOR INPUT
   SYMBOLIC QUEUE               SQI
   MESSAGE DATE                 MDI
   MESSAGE TIME                 MTI
   SYMBOLIC SOURCE              SSI
   TEXT LENGTH                  TLI
   END KEY                      EKI
   STATUS KEY                   SKI
   MESSAGE COUNT                MCI.
```

```
01 CDIN-R PIC X(87).
CD CDOUT FOR OUTPUT
   DESTINATION COUNT             DCO
   TEXT LENGTH                   TLO
   STATUS KEY                    SKO
   ERROR KEY                     EKO
   SYMBOLIC DESTINATION          SDO.
01 CDOUT-R.
   02 FILLER PIC 9(4) VALUE 1.
   02 FILLER PIC 9(4) VALUE 1.
   02 FILLER PIC X(15).

PROCEDURE DIVISION USING TDS-STORAGE CONSTANT-STORAGE TRANSACTION-
STORAGE.

START.
MOVE SYMBOLIC-QUEUE TO SQI.
RECEIVE CDIN MESSAGE INTO ZRECEP.
IF EKI = ZERO MOVE 1 TO AIG GO TO START.
MOVE SSI TO SDO.
IF AIG = 1 MOVE "MESSAGE TOO LONG. RESTART TRANSACTION" TO MESS-
ERR
          GO TO SEND-ERROR.
DISPLAY "CALL SUBJOB PENDING" UPON TERMINAL.
MOVE ZERO TO JOB-SWITCHES.
```

```
  CALL "SUBJOB" USING DESCRPT-JOB STAT JOB-START.
```

```
IF RESULT NOT=ZERO DISPLAY "OH WELL.. RESULT = " RESULT UPON
TERMINAL
                   MOVE SPACES TO NEXT-TPR GO TO TPR-END.
```

E. TCAM

Note that the features described in this Appendix are not part of standard TDS; to use these features, you need a special MI (Marketing Identifier).

TDS uses TCAM (TDS Communications Access Method) for pass through communications. TCAM can be used to communicate between a Front-End application (which initiates the dialog) and a Back-End application (which does not initiate the dialog). The following diagram shows this communication method.

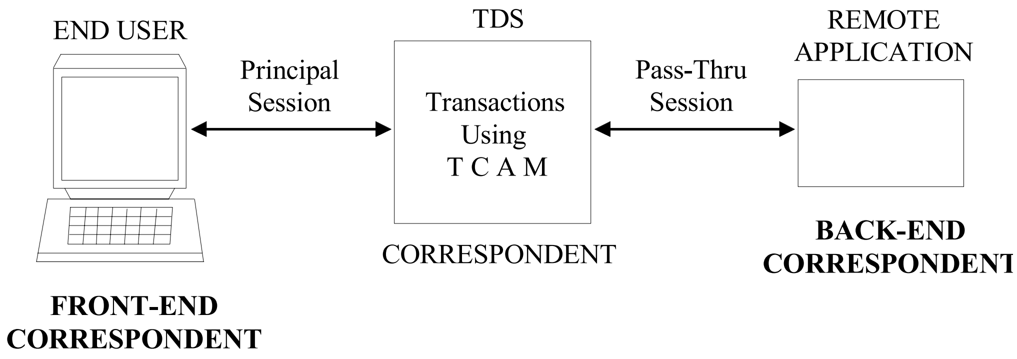


Figure E-1. Using TCAM as Communication between the End User and the Remote Application

A session is the logical link between two correspondents. TDS creates it at the time of connection. Figure E-1 shows a transaction using TCAM with two sessions: the principal session and the pass-thru session. The principal session initiates a transaction that will dialog with a remote application. When the transaction starts, the principal session is already open and belongs to the transaction. A TCAM call statement opens the pass-thru session, and it then belongs to the transaction.

Figure E-1 shows three correspondents: the logged user, the TDS, and the remote application. These three correspondents dialog with each other through two sessions. The correspondents cannot send data at the same time. Instead, they take turns. The correspondent with the turn has the right to send data to another correspondent.

During the session, the TPR sends and receive events and interruptions (also called special events). These are present in the form of a returned status in a call statement.

E.1 The TCAM Call Statements

With TCAM, an end user can initialize a TDS transaction to dialog with another application. This application can be on the same site as the TDS session, or on another site. In this way, the transaction can access the files and databases declared in the TDS and those belonging to the remote application.

TDS uses TCAM in the transaction code of a set of CALL statements (or procedures) to communicate with other applications. The TCAM call statements are as follows:

TERMID	gets the terminal characteristics. (Principal session only.)
TOPEN	opens a TCAM pass through session. (Pass-thru session only.)
TEVENT	returns session events to the transaction. (Both sessions.)
TRECV	receives data from the remote application. (Pass-thru session only.)
TRCVIT	receives interruptions from the remote application. (Pass-thru session only.)
TSEND	sends data to the remote application. (Pass-thru session only.)
TSENDIT	sends interruptions to the remote application. (Both sessions.)
TCLOSE	closes the TCAM pass-thru session. (Pass-thru session only.)

E.2 Using TCAM

A transaction uses TCAM to create a pass-thru session. This pass-thru session provides the communication environment between the TDS and the remote application. However, the end user does not see the TDS and is able to communicate with the remote application as though the TDS were not there.

There is no defined application protocol for the pass-thru session, leaving the user without restrictions in using the pass-thru session.

There is no message recovery function on the pass-thru session. However, the end user can write an application (TDS and remote) to use this function.

A transaction in pass-thru mode manages the pass-thru and the principal session using this set of verbs:

Principal Session	Pass-Thru Session
CALL "TEVENT"	CALL "TOPEN"
RECEIVE	CALL "TEVENT"
SEND	CALL "TRECV"
CALL "TSENDIT"	CALL "TSEND"
CALL "TCLOSE" (with reason 35)	CALL "TSENDIT"
	CALL "TRCVIT"
	CALL "TCLOSE"

NOTE:
In the principal session, there is no verb especially defined to receive the interruption. TDS sends them to the transaction using CALL "TEVENT".

Pass-Thru Session Limits

- A transaction can open only one pass-thru session.
- The transaction that opens a session, manages the session.
- The pass-thru session is closed at the end of the transaction. Either the transaction or the TDS closes it. If it is not closed, TCAM closes it automatically.
- A transaction can not be in a pass-thru mode and XCP1 mode at the same time. (The TOPEN or CP-ALLOCATE call statement in the transaction defines these modes.)
- A transaction cannot be in a pass-thru mode and XCP2 mode at the same time.
- A user connected to an OMH terminal cannot enter a transaction using the TDS pass-thru.
- TDS does not include the pass-thru sessions when checking the maximum number of connections allowed.

Using the Question Mark (?)

The user, when logged to a terminal can use the question mark in a transaction as follows:

- When the user is logged on to a terminal and the transaction is not in pass-thru mode, the question mark displays the most recent message again.
- When the transaction is in pass-thru mode, the question mark does nothing. The transaction will receive (in the RECEIVE verb) a question mark (?), but the message is normal.

Using the Break Signal

In pass-thru mode, a break signal at the terminal sends an interruption in the CALL "TEVENT", using a status of 43 (V-ATTENTION). This is not the normal break signal processing by TDS.

EXAMPLE: Break Signal

The following diagram shows a break signal.

- At the start of this example, the terminal has the turn in the principal session, and the transaction has the turn in the pass-thru session.
- When the break occurs at the terminal, it is sent to the transaction as an interruption message (message 1), which is a status of 43 (V-ATTENTION) in the CALL "TEVENT". TPR N then sends this to the remote in the CALL "TSENDIT", using TYPE = 0. This is message 2.
- The remote does not have the turn. So it sends an interruption message (message 3) to the transaction, asking for the turn.
- TPR N+1 receives a status of 35 (V-INTERRUPT in a pass-thru session) in the CALL "TEVENT", so it sends a CALL "TRCVIT" to the terminal. The result is TYPE = 1 (demand the turn).
- In the principal session, TPR N+1 still does not have the turn, so it sends an interruption message (message 4) to the terminal, asking for the turn.
- The transaction receives the turn after it gets a status of 41 (V-DATA) in the principal session.

❑

The following diagram shows a break signal.

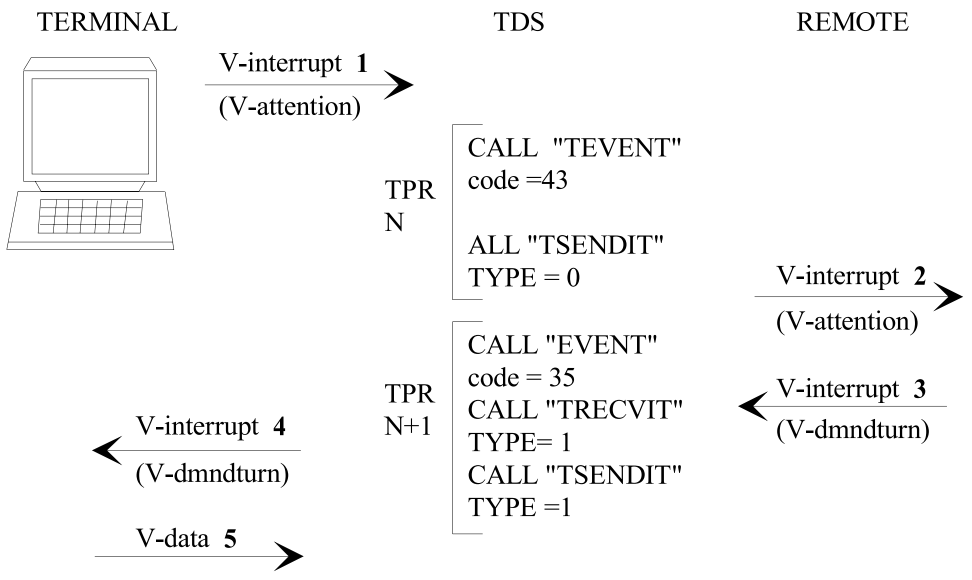


Figure E-2. The Break Signal

E.3 TCAM Call Statements

E.3.1 The CALL "TERMID" Procedure

Syntax

```
CALL "TERMID" USING ADDRESS OF data-name1.
```

Description

This call retrieves the principal correspondent characteristics. These characteristics are for the terminal and are necessary for TDS to open a pass-thru session.

The usage and parameters for this verb are described in the chapter on session management procedures, in the subsection on the CALL "TERMID" procedure.

E.3.2 The CALL "TOPEN" Procedure

Syntax

```
CALL "TOPEN" USING    ADDRESS OF    data-name1,
                                    data-name2,
                                    data-name3,
                                    data-name4,
                                    data-name5,
                      ADDRESS OF    data-name6.
```

Description

This call opens a pass-thru session between the TDS transaction and a target application.

A pass-thru session is in asynchronous mode and TWA (Two-Way Alternate) mode. TOPEN opens an asynchronous pass-thru session. Before opening a pass-thru session, the transaction checks a status of 104 (session pass-thru already opened), which determines if there is one already opened.

TDS verifies the structure and the parameters. If the structure is too small, or if the parameters are invalid, the ARGERR return code aborts the transaction.

The status of 30 or 31 on CALL "TEVENT", not 0, acknowledges that the asynchronous session is open. Status 0 means the transaction is in pass-thru mode.

When the target application is a TDS, TDS fills the SYMBOLIC-SOURCE field with the terminal internal address.

The FILLER field of this call can suppress the transport of the device header and the pending timer event. If the first four characters in this field are set to the NDVH value on the TOPEN statement performed in TDS1, the device headers coming from the terminal and received by TDS1 are not sent to the target, and TDS1 will not receive any device header from the target.

If the last 4 characters in the FILLER field are NOTM, the principal session (with send EMI, or without send) is no longer pending on the wait-time. An event coming from the pass-thru session will start the TPR. This is useful in detecting a transmission failure from the PT session when the target has kept the turn.

data-name1 Usage

data-name1 is an input parameter. It is the name of a structure that must be declared as follows:

```
01  data-name1.
   02  DESTMBX            PIC X(8)    VALUE "IOF".
   02  DESTNODE           PIC X(4)    VALUE SPACES.
   02  PASSWORD           PIC X(12)   VALUE SPACES.
   02  TURN               PIC X       VALUE "A".
   02  OTERMID.
      03  CORRESPID       PIC X(12).
      03  PROJECT         PIC X(12).
      03  BILLING         PIC X(12).
      03  LOCALDVC.
         04  MODEL        COMP-1.
         04  TYPE.
            05  DISPLAY   PIC X.
            05  KEYBRD    PIC X.
            05  PRINTER   PIC X.
         04  PAGEL        COMP-1.
         04  LINEL        COMP-1.
         04  DVFEATR
            05  ROLLUP    PIC X.
            05  WRAPAR    PIC X.
            05  LINFEED   PIC X.
            05  LINFOLD   PIC X.
            05  HTAB      PIC X.
            05  VTAB      PIC X
      03  FILLER          PIC X(12).
   02  USERINFO           PIC X (32).
```

Parameters for data-name1

DESTMBX	is an 8-character alphanumeric field (PIC X(8)) that contains the name of the remote application.
DESTNODE	is a 12-character alphanumeric field (PIC X(4)) that contains the node name of the remote application. If this field is blank, the remote application is on the same site as the local TDS.

PASSWORD	is a 12-character alphanumeric field (PIC X (14)) that contains the target correspondent password. If the CORRESPID field of this structure contains the principal correspondent name and the PASSWORD field is left blank, then TDS pass-thru retrieves the password from the CATALOG.
TURN	is a 1-character alphanumeric field (PIC X) that contains one of the following values: A or I. For A, the transaction specifies that the initial turn be given to the remote application. For I, the transaction specifies that the initial turn stay at the transaction.
OTERMID	Either the CALL "TERMID" statement or the transaction itself can fill this part of the structure. For more information, see the CALL "TERMID" statement.
USER INFO	This field is specific to the remote application, and it can be redefined.

For example, when the remote application is IOF, the structure is as follows:

```
02  USERINFO.
      03   SEPAR1    PIC X value "$".
      03   STATION   PIC X(8)./*station name*/
      03   SEPAR2    PIC X value "!".
      03   OPTION    PIC X(22) value "NSTARTUP"
```

Where:

SEPAR 1	"$" if there is a station. " " (blank) if there is not a station.
SEPAR 2	"!" if there is an IOF option. " " (blank) if there is not.

When the remote application is another TDS, the TDS that is requesting the connection gives "user information" to the remote TDS. The structure is as follows:

```
02  USERINFO.

  03  FILLER  PIC X(9).
  03  SEPAR3  PIC X VALUE "?".
  03  PARAM   PIC X VALUE "A".
  03  SYSLGTH PIC X.
  03  SYSHDR  PIC X(8).
  03  FILLER  PIC X(12).
```

You may pass information to the LOGON transaction of TDS by placing "%" in the SYSLGTH field and up to 8 characters in the SYSHDR field. The LOGON transaction can retrieve the information via a RECEIVE statement.

Where:

SEPAR 3	"?" if there are presentation options (which are specified for a TDS remote). " " (blank) if there is not a TDS option.
PARAM	encodes the two options that are described at terminal level: UNEDIT and NSYSMSG. UNEDIT. In the case of VIP, messages and device procedure headers are visible to the application. NSYSMSG. TDS cannot send system messages in this session.

The encoding is as follows:

“A”	No UNEDIT, no NSYSMES
“B”	UNEDIT, no NSYSMES
“C”	no UNEDIT, NSYSMES
“D”	UNEDIT, NSYSMES
“S”	Same as A, but with the TRACE PRINT option
“T”	Same as B, but with the TRACE PRINT option
“U”	Same as C, but with the TRACE PRINT option
“V”	Same as D, but with the TRACE PRINT option

SYSLGTH — contains one of two values: the effective length of the SYSHDR field ("0", "2", "4", "6", "8"), or "%", which is a particular separator that separates strings from a system header. See below.

SYSHDR — defines the presentation of the service messages that TDS sends. It defines a prefix that is inserted in front of each TDS service message, which overrides the one defined at TDS GENERATION time in the SERVICE-MESSAGE statement. This field is encoded as follows:

It is encoded with each character representing a hexadecimal EBCDIC DIGIT. The user can pass a particular string (of up to 8 alpha-numeric characters) at connection time and the LOGON transaction processes it. This string is sent to the first TPR of the LOGON transaction as a message in the RECEIVE NO DATA statement. The RECEIVE mechanism determines if the user has provided a string. Because the previous system header also has a "%" string, these two mechanisms cannot be used at the same time.

For example, the default prefix, which is CR-LF, is encoded with the 4 characters "OD25".

data-name2 Usage

data-name2 is an output parameter. It is a computational field (COMP-1) that defines the status returned on the CALL "TOPEN" statement. The returned status values are:

0	DONE
1	REJECT-ABN
2	REJECT-DNNOTOP
3	REJECT-DNSAT
4	REJECT-DMUNKN
5	REJECT-DMNOTOP
6	REJECT-DMSAT
7	REJECT-DASAT
9	REJECT-DIAREJ
21	REJECT-TIMEOUT
52	OPERR-RESOV
56	OPERR-DSTUNS
64	REJECT-DNUNKN
65	REJECT-PDNNOTAV
66	REJECT-DUPUSERID
67	REJECT-DUPSTID
68	REJECT-MODNOTAV
85	INV-OPSTRUC
100	ERR-USERID
101	ERR-MODEL
102	ERR-TERMCHAR
103	ERR-XCP
104	ERR-OPALREADY
128	PTARGERR

data-name3 Usage

data-name3 is an output parameter. It is a computational field (COMP-2) used for debugging because it contains wrong status information. The TPR can print it.

data-name4 Usage

data-name4 is a computational field (COMP-1) used to pass four parameters or less. If more than four parameters are passed, this is an output field containing the pass-thru session identifier.

data-name5 Usage

data-name5 is a computational field (COMP-1) that sets the timer on a remote session. For more information, see the data-name5 parameter description of the TSEND procedure.

data-name6 Usage

data-name6 is the name of a structure that extends the capabilities of the CALL "TOPEN", as follows:

```
01 data-name6.
   02 EXTENDED-VERSION     COMP-1.
   02 EXTENDED-MBXEXT      PIC X(4).
```

Parameters for data-name6

EXTENDED-VERSION	must be set to 1.
EXTENDED-MBTEXT	is either the name of the extended remote mailbox or must be set to spaces.

If the timer on remote session is not used, it must be set to 0.

E.3.3 The CALL "TEVENT" Procedure

Syntax

```
CALL "TEVENT" USING    dataname1,
                       dataname2,
                       dataname3.
```

Description

In pass-thru mode this call must be at the beginning of each TPR. If not, the UBUGGER return code aborts the transaction.

This procedure returns all of the VCAM or timer events in the pass-thru session to the transaction. However, it returns only some of the VCAM events in the principal session. This tells the user what the next should be.

The status values all correspond to VCAM semaphore messages, VCAM interruptions, and timer semaphore messages in the principal and pass-thru sessions. In the pass-thru session, some of the interruptions are returned to the transaction with the CALL "TRCVIT".

If the structure is too small, or if the parameters are invalid, the ARGERR return code aborts the transaction.

Usage

TEVENT usage implies that a commitment point is forced at the end of the current TPR.

data-name1 is an output parameter. It is a computational field (COMP-1) that defines the status returned on the "TEVENT" procedure.

The following is a list of the status values returned involving the pass-thru session:

29	PT-TIMER
30	PT-ACK-IN
31	PT-ACK-ACC
32	PT-CLOSED
33	PT-DATA
34	PT-CREDIT
35	PT-INTRPT
36	PT-ACK-REJ
70	SYSTERR
71	IGNORE
105	ERR-NOTPT

The following is a list of the status values returned involving the principal session:

40	MN-CLOSED
41	MN-DATA
42	MN-CREDIT
43	MN-ATT
44	MN-ABNTERM
45	MN-TERMREQ
46	MN-TIMEOUT

data-name2 is an output parameter. It is a computational field (COMP-1) that defines the secondary status returned on the "TEVENT" procedure. This field is used only if the value of the first status in data-name1 is 36 (PT-ACK-REJ: pass thru session).

The following is a list of the status values that this field returns:

1	REJECT - ABN
2	REJECT - DNNOTOP
3	REJECT - DNSAT
4	REJECT - DMUNKN
5	REJECT - DMNOTOP
6	REJECT - DMSAT
7	REJECT - DASAT
9	REJECT - DIAREJ
10	REJECT - PRESREJ
21	REJECT - TIMEOUT
23	REJECT - RIGHVIOL
24	REJECT - SECVIOL
52	OPERR - RESOV
55	OPERR - LMBXSAT
56	OPERR - DSTUNS
64	REJECT - DNUNKN
65	REJECT - PDNNOTAV
66	REJECT - DUPUSERID
67	REJECT - DUPSTID
68	REJECT - MODNOTAV
>=128	PTARGERR

data-name3 is an output parameter. It is a computational field (COMP-2) used for debugging because it contains wrong status information. The TPR can print it.

Timeout

The "TEVENT" call returns a status of 46, which is a timeout status. This code is used about the dialog on the principal session only. Remember the following rules about timeout management (TDS is not in pass-thru mode):

- The TDS-STORAGE specifies the wait time.
- After a SEND WITH EGI, when the TPR specifies a wait time, the next TPR can start after the terminal response arrives (even before the delay), or after the wait time delay.
- After a SEND WITH EMI, when the TPR specifies a wait time, the next wait time starts after the wait time delay. However, in this case, the TPR cannot start before the delay, and even later because the V-CREDIT is also waiting.

In pass-thru mode, the timeout status can appear (when a wait time is specified) according to the following two timeout examples.

EXAMPLE: Timeout after a SEND WITH EGI

The diagram below shows how CALL "TEVENT" uses a timeout after a SEND WITH EGI.

- During the processing TPR N, the first event (1) coming from the remote application takes place in the pass-thru session.
- TPR N specifies the wait time delay. As a result, the timer starts at the end of TPR N, when the message (2) is sent to the terminal.
- TPR N+1 can start before the end of the delay because the TDS has received the first event (1), which has not yet returned to the transaction.
- During the processing of TPR N+1, the timer notification occurs, which means that the terminal has not answered. No other event occurs in the pass-thru session. TPR N+2 starts and the timeout is returned on "TEVENT".

❑

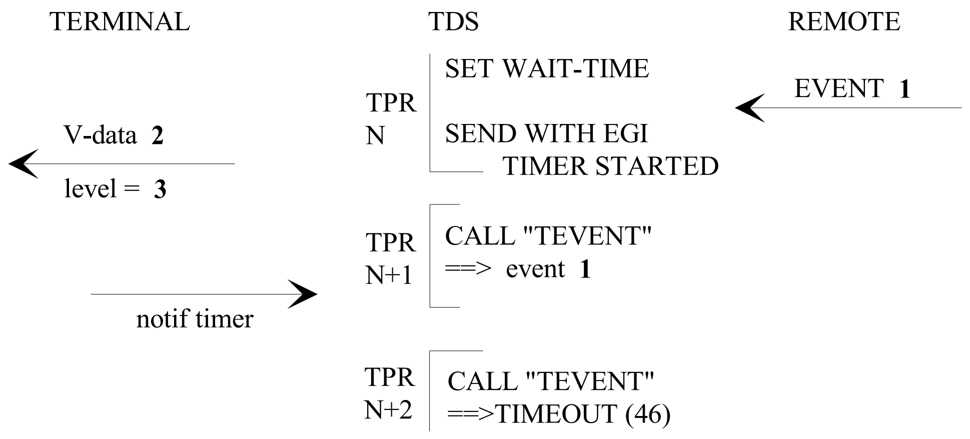


Figure E-3. Timeout after SEND WITH EGI

EXAMPLE: Timeout after a SEND WITH EMI

The following diagram shows how the CALL "TEVENT" uses a timeout after a SEND WITH EMI.

- During the processing of TPR N, the first event (1) takes place in the pass-thru session.
- TPR N specifies the wait time delay. As a result, the timer starts at the end of TPR N, when the message (2) is sent to the terminal.
- Unlike in the SEND WITH EGI, TPR N+1 must wait to start after the delay.
- The CALL "TEVENT" returns the status of 46 (TIMEOUT). The principal session events (in this case, timeout) are returned to the transaction before this pass-thru session (in this case, event 1).
- Note that it is not necessary to wait for the V-CREDIT before starting TPR N+1. This is because this takes place in pass-thru mode.

❑

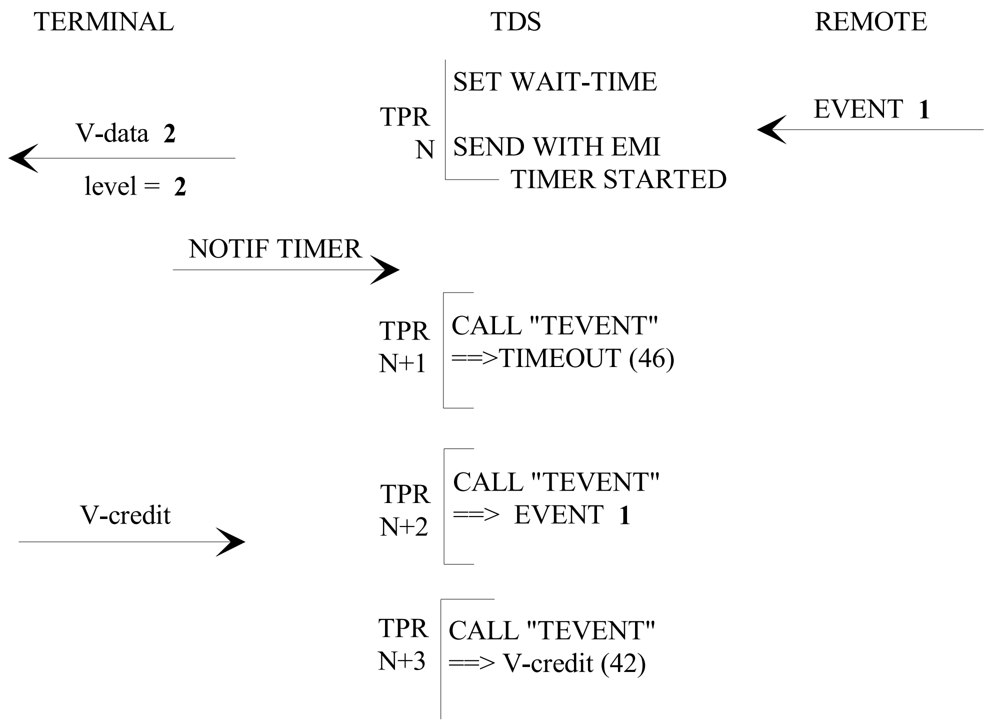


Figure E-4. Timeout after SEND WITH EMI

E.3.4 The CALL "TRECV" Procedure

```
CALL "TRECV" USING ADDRESS OF   data-name1,
                                data-name2,
                                data-name3.
```

Description

Receives the data from the remote application in the pass-thru session.

This call must be used to receive data in the pass-thru session. The ARGERR return code aborts the transaction if the parameters are not valid.

Usage

data-name1 is the name of the structure that is declared as follows:

```
01  data-name1.
  02  LEVEL      COMP-1.
  02  LENGTH     COMP-1.
  02  BUFFER     PIC X(LENGTH).
```

LEVEL	is an output field that contains the retrieved enclosure level. Valid values are: 1: end of record. The remote application keeps the turn 3: end of interaction unit. The transaction receives the turn. 5: end of message group.
LENGTH	is both an input and output field. As an input field, it must contain the maximum buffer length. As an output field, it contains the length of the message received.
BUFFER	is an output field that contains the message received.

data-name2 is an output parameter. It is a computational (COMP-1) field that defines the status that the "TRECV" statement returns. The following is a list of the status values:

0	DONE
35	PT-INTRPT
37	PT-INTRPT
38	PT-TRUNC
70	SYSTERR
71	IGNORE
80	INV-BUFLG
105	ERR-NOTPT

data-name3 is an output parameter. It is a computational field (COMP-2) used for debugging.

E.3.5 The CALL "TRCVIT" Procedure

Syntax

```
CALL "TRCVIT" USING ADDRESS OF    data-name1,
                                  data-name2,
                                  data-name3.
```

Description

Receives the interruptions that come from a remote application in a pass-thru session. The ARGERR return code aborts the transaction if the parameters are not valid.

Usage

data-name1 is the name of the structure that is declared as follows:

```
01  data-name1.
    02 TYPE     COMP-1.
    02 LENGTH   COMP-1.
    02 BUFFER   PIC X(LENGTH).
```

TYPE	is an output field that contains the type of the interruption received. Valid values are: 1: V-DMNDTURN 2: V-PURGE 3: V-ABNTERM 8: V-TERMREQ 9: V-TELEG
LENGTH	is both an input and output field that is used when the type value is 9 (v-teleg). As an input field, it contains the length of a telegram (80 bytes), to avoid a truncation. As an output field, it contains the telegram length.
BUFFER	is an output field that is used when the interruption type is 9. It contains the telegram.

data-name2 is an output parameter. It is a computational (COMP-1) field that defines the status that the "TRCVIT" statement returns. The status values are:

0	DONE
35	PT-INTRPT
37	PT-TRUNC
70	SYSTERR
71	IGNORE
80	INV-BUFLG
105	ERR-NOTPT

data-name3 is an output parameter. It is a computational field (COMP-2) used for debugging because it contains wrong status information. The TPR can print it.

E.3.6 The CALL "TSEND" Procedure

Syntax

```
CALL "TSEND" USING ADDRESS OF    data-name1,
                                 data-name2,
                                 data-name3,
                                 data-name4,
                                 data-name5,
                                 data-name6.
```

Description

Sends data to the remote application in a pass-thru session.

The ARGERR return code aborts the transaction if the parameters are not valid.

Usage

data-name1 is the name of the structure that is declared as follows:

```
01  data-name1.
  02  LEVEL      COMP-1.
  02  LENGTH     COMP-1.
  02  BUFFER     PIC X(LENGTH).
```

Where:

LEVEL	is an input field that contains the enclosure level. Valid values are: 1: end of record. The transaction keeps the turn. 2: end of quarantine unit. The transaction keeps the turn. 3: end of interaction unit. The remote application receives the turn. 5: end of message group.
LENGTH	is an input field that contains the length of the message to be sent.
BUFFER	is an input field that contains the message to be sent.

data-name2 is an output parameter. It is a computational (COMP-1) field that defines the status that the "TSEND" statement returns. The following is a list of the status values:

0	DONE
35	PT-INTRPT
38	PT-TURNVIOL
39	PT-WAITCR
70	SYSTERR
80	INV-BUFLG
81	INV-LEVEL
86	INV-DVCHD
105	ERR-NOTPT
106	ERR-DVCHD

data-name3 is an output parameter. It is a computational field (COMP-2) used for debugging because it contains wrong status information. The TPR can print it.

data-name4 is not valid for this release. It is a computation field (COMP-1).

data-name5 is a computational field (COMP-1) that sets the timer (in seconds) in a remote session. A TPR can set this timer only once, and it detects if the remote application never gives a response. If the remote application does give a response before the time expires, TDS reset the timer and the TPR can be restarted.

data-name6 is the name of a structure that gives the device header to the remote application, as follows:

```
COBOL DECLARATION
01 UDVCHD.
 02 UDVCHDLG COMP-1.
 02 UDVCHDVL PIC X(30).
```

In the following example, no checking is done on the validity of the device header contents compared to the terminal type. The valid characters are in the set of 0 to 9 and A to F:

```
MOVE 6 TO UDVCHDLG.
MOVE "004040" TO UDVCHDVL.
```

The following is a list of the status values:

86	invalid characters in device header
106	function not allowed (DPNS).

E.3.7 The CALL "TSENDIT" Procedure

Syntax

```
CALL "TSENDIT" USING        data-name1,
                            data-name2,
                            ADDRESS OF data-name3,
                            data-name4,
                            data-name5.
```

Description

Required in order to send interruptions to the remote application in a pass-thru session or to a principal correspondent in a principal session.

The ARGERR return code aborts the transaction if the parameters are not valid.

Usage

data-name1 is an input character. It is a 1-character alphanumeric field (PIC X). It contains the interruption destination. Valid values are as follows:

1:	Send to the principal correspondent.
2:	Send to the remote application.

data-name2 is an input parameter. It is a 1-character alphanumeric field (PIC X) and contains the type of interruption to be sent.

Valid values on a pass-thru session are as follows:

0:	V-ATTENTION
1:	V-DMNDTURN
9:	V-TELEG

Valid values on a principal session are as follows:

1:	V-DMNDTURN
9:	V-TELEG

data-name3 is an input parameter. It is the name of a structure used if the type of interruption is 9 (V-TELEG), declared as follows:

```
01  data-name3.
  02  LENGTH  COMP-1.
  02  BUFFER  PIC X(80).
```

LENGTH	contains the length of the telegram to be sent. This length contains the size of device headers (from 1 to 16 bytes) plus one more byte if the DATANET version is DNS-V4. This length must be no more than 80 bytes.
BUFFER	contains the telegram.

data-name4 is an output parameter. It defines the status returned by "TSENDIT".

The following is a list of the status values.

In a pass-thru session:

35:	PT-INTRPT

In a principal session:

43	MN-ATT
44	MN-ABNTERM
45	MN-TERMREQ

In both sessions:

0	DONE
70	SYSTERR
72	TURN-ALREADY
80	INV-BUFLG
82	INV-INTTYP
83	INV-DEST
105	ERR-NOTPT

data-name5 is an output parameter. It is a computational field (COMP-2) used for debugging because it contains wrong status information. The TPR can print it.

E.3.8 The CALL "TCLOSE" Procedure

Syntax

```
CALL "TCLOSE" USING      data-name1,
                         data-name2,
                         data-name3.
```

Description

Required to close the pass-thru session if the close is abnormal. This procedure is also required in order to inform the TDS, with a status of 35, when the principal session disconnects. For more information, see the subsection describing abnormal disconnections, later in this appendix.

The ARGERR return code aborts the transaction if the parameters are not valid.

Usage

data-name1 is an input parameter. It is a computational field (COMP-1) that contains the code giving the reason for the close. The code is given to the remote application. Valid values are from 35 through 44.

data-name2 is an output parameter. It is a computational (COMP-1) field that defines the status that the "TCLOSE" statement returns. The following is a list of the status values:

0	DONE
73	PTCLS-ER
84	INV-CLCODE
105	ERR-NOTPT

data-name3 is an output parameter. It is a computational field (COMP-2) used for debugging because it contains wrong status information. The TPR can print it.

E.4 Return Status List and Definitions

Status Code	Definition
0	Successful completion of the verb

The following status codes occur when the pass-thru session rejects the open verb. (V-REJECT)

Status Code	Definition
1	Abnormal rejection reason
2	Destination node not operable
3	Destination node saturated
4	Destination mailbox unknown
5	Destination mailbox not operable
6	Destination mailbox saturated
7	Destination application saturated
9	Dialog rejection (as a result of negotiation)
10	Presentation rejection (as a result of negotiation)
21	Timeout waiting for acknowledgement
23	Access right violation (object access)
24	Security violation (subject access)
30	Acknowledgement of open in pass-thru session. The transaction gets the turn and can call the TSEND statement.
31	Acknowledgement of open in pass-thru session. The remote application gets the turn. The transaction must wait, using TEVENT in the next TPR, for status V-DATA (33) from the remote application.
32	V-MGCLOSED received in the pass-thru session. The transaction is no longer in pass-thru mode.
33	V-DATA received in the pass-thru session. The TPR can call the "TRECV" statement in order to receive the data from the remote application.
34	V-CREDIT received in the pass-thru session. The TPR can call the "TSEND" statement in order to send some data to the remote application.
35	V-INTERRUPT received in the pass-thru session. The TPR must call the "TRCVIT" statement in order to receive the interruption coming from the remote application.
36	V-OPENACK received in the pass-thru session, but the open is rejected. The second parameter of "TEVENT" gives more details on the rejection reasons.

37	The buffer length is insufficient, resulting in a truncation. Some data is lost.
38	TURNVIOL. No data is received or sent.
39	A V-CREDIT must arrived before sending data toward the remote application.

The following status codes occur during only the principal session.

Status Code	Definition
40	V-MGCLOSED received in the principal session, causing it to close.
41	V-DATA received in the principal session. The TPR can call the RECEIVE CD-IN to receive the data from the principal correspondent.
42	V-CREDIT received in the principal session. The TPR can use the SEND to the principal session later.
43	V-ATTENTION received in the principal session. If the principal correspondent is a terminal, this event indicates that a break has occurred in it.
44	V-ABNTERM received in the principal session, causing it to disconnect abnormally. The transaction can use "TCLOSE" with code 35.
45	V-TERMREQ received in only the principal session. This could occur, for example, after a TT operator command.
46	Timeout received for the transaction.

The following status codes occur when the pass-thru session open fails.

Status CodeDefinition

52	V-RESOV. System overload. Retry later with "TOPEN".
55	V-LMBXSTAT. Maximum number of sessions on the local mailbox is reached.
56	V-DSTUNSPEC. Destination unspecified.

The following status codes occur when the pass-thru session open is rejected. (V-REJECT)

Status Code	Definition
64	Destination node unknown
65	Path to the destination node is not available
66	Duplicate user identifier
67	Duplicate station identifier
68	Telecom module not available (line or datanet)

The following status codes occur at any time.

Status Code	Definition
70	A system error has occurred. The transaction must close the pass-thru session using "TCLOSE"
71	Ignore this event
72	ALREADY. The transaction already has the turn.
73	There is an error during the close of the pass-thru session.

The following status codes occur when there is an invalid parameter.

Status Code	Definition
80	Invalid buffer length
81	Invalid enclosure level
82	Invalid interruption type
83	Invalid destination parameter
84	Invalid close code
85	Invalid value in OPEN structure
86	Invalid character in the device header

The following status codes occur at any time.

Status Code	Definition
100	Error during the user identification research in the catalog
101	Unknown value for model field in the OPEN structure
102	Error during the access to terminal characteristics
103	Pass-thru mode is not possible, due to XCP usage.
104	Session pass-thru is already open
105	Verb not allowed. The transaction is not in pass-thru mode
106	Function not allowed (DPNS)
>=128	V-ARGERR on open in the pass-thru session.

E.5 Status Codes and Statement Cross Reference

Codes	Statements							
	TERMID	TOPEN	TEVENT	TRECV	TRCVIT	TSEND	TSENDIT	TCLOSE
0		X		X	X	X	X	X
1		X	X					
2		X	X					
3		X	X					
4		X	X					
5		X	X					
7		X	X					
9		X	X					
10		X	X					
21		X	X					
23		X	X					
24		X	X					
30			X					
31			X					
32			X					
33			X					
34			X					
35			X	X	X	X	X	
36			X					
37				X	X			
38				X		X		
39						X		
40			X					
41			X					
42			X					
43			X			X		
44			X			X		
45			X			X		
46			X					
52		X	X					
55		X	X					
56		X	X					
64		X	X					
65		X	X					
66		X	X					
67		X	X					
68		X	X					
70			X	X	X	X	X	
71			X	X	X			
72							X	
73								X
80				X	X	X	X	
81						X		
82							X	
83							X	
84								X
85		X						
100		X						
101		X						
102		X						
103		X						
105			X	X	X	X	X	X
128		X	X					

E.6 Using the TCAM Call Statements

E.6.1 Opening and Closing a Pass-Thru Session

The CALL "TOPEN" statement opens the pass-thru session. However, the transaction must wait until the CALL "TEVENT" returns a status of 30 (PT-ACK-IN) or 31 (PT-ACK-ACC), which ensure that the session is open. A session can close in two ways: normally or abnormally.

Normally, the session is closed after the TDS and the remote application exchange "data of level 5" in the session. Either the transaction or the remote application can initiate this exchange (using TRECV and TSEND).

Abnormally, the transaction closes the session with a "TCLOSE" call. The "TCLOSE" call should be used when the transaction receives either a return status of 70 (SYSTERR) or a warning that the principal session is disconnected.

E.6.2 Starting and Ending Pass-Thru Mode

To start pass-thru mode, the CALL "TOPEN" statement declares that a transaction is in pass-thru mode with a return status of done. To end pass-thru mode, the following calls allow a transaction to exit:

CALL "TEVENT"	This call returns a status of 36 (PT-ACK-REJ), which indicates the pass-thru session is not open, or a status of 32 (PT-CLOSED), which indicates the pass-thru session is closed.
CALL "TSEND"	The transaction exits pass-thru mode when the transaction first receives "data of level 5" ("TRECV") and then sends "data of level 5" ("TSEND") to the remote application.
CALL "TRECV"	The transaction also exits pass-thru mode when it first sends "data of level 5" ("TSEND") to the remote application and then receives "data of level 5" ("TRECV") from the remote application.

CALL "TCLOSE"	A CALL "TCLOSE" (with a close code of 35 as the first parameter) warns the TDS about the disconnection of the principal session and starts the DISCONNECT transaction. The transaction waiting for the principal session to reconnect is no longer in pass-thru mode. Note that if the close reason in the CALL "TCLOSE" is not 35, the transaction remains in pass-thru mode until the next CALL "TEVENT" sends a return status of 32 (PT-CLOSED).
END OF TRANSACTION	The TDS closes the pass-thru session with code 36.

Programming Rules

- Each TPR must begin with a CALL "TEVENT" in order to receive all the events and interruptions that occur during the two sessions (principal and pass-thru) and determine which statement must follow.
- The return status of 44 (MN-ABNTERM) on CALL "TEVENT" or CALL "TSENDIT" tells the transaction that the principal session has disconnected, and not to attempt to dialog with it. The transaction can continue, however, and dialog in the pass-thru session. Then, at the end of the dialog with the remote application, the transaction must inform the TDS of the disconnection with the CALL "TCLOSE" statement with code 35.

  Even if the transaction does not use "TCLOSE" with code 35, the TDS is automatically informed about the principal session disconnection.

  If the transaction does attempt to dialog with the disconnected principal session, it receives one of two statuses. The transaction can receive a wrong status returned (TFAILED) on the RECEIVE or SEND verb that is not the last Send of the TPR. The transaction can also receive a UBUGERR return code (on the last SEND verb) that aborts the transaction.

E.6.3 Command Sequencing

The following call statements must meet these conditions.

In Either Session

"TEVENT"	This call is mandatory at the beginning of a TPR.
"TSENDIT"	This call can be used at any time.

In the Principal Session

SEND	The transaction has the turn in the principal session, after a RECEIVE.
SEND	42 (V-CREDIT) Additional SEND verbs (with EMI) can be sent in the principal session. However, the transaction must receive the V-CREDIT first, if the previous SEND was with EMI.
RECEIVE	41 (V-DATA) With status 41, the transaction can use this verb.

In the Pass-Thru Session

"TSEND"	The transaction has the turn in the pass-thru session. This is after a "TRECV" with level 3, or after status 30 (PT-ACK-IN) on "TEVENT".
"TSEND"	34 (V-CREDIT) Additional "TSEND" verbs (with level 1 or 2) can be used in a pass-thru session. However, a V-CREDIT (status 34) must be received before sending the next "TSEND".
"TRCVIT"	35 (V-INTERRUPT) If the transaction receives the status 35, a CALL "TRCVIT" must be performed. Status 35 is returned on "TEVENT", "TRECV", "TRCVIT", "TSEND", and "TSENDIT".
"TRECV"	33 (V-DATA) Status 33 allows the transaction to use the "TRECV" call statement.

Command Sequencing Rules

- TDS does not control the command sequencing. However, the programming rules should be respected. If not, a wrong status is returned or the transaction is aborted. If these rules are not followed:
  - either the wrong statuses are returned on the call statement or SEND/RECEIVE verbs
  - or the transaction is aborted (for example a SEND on a disconnected principal session).

- Two conversations can be initialized in the same TPR: one for each session. This is shown in the diagram below.

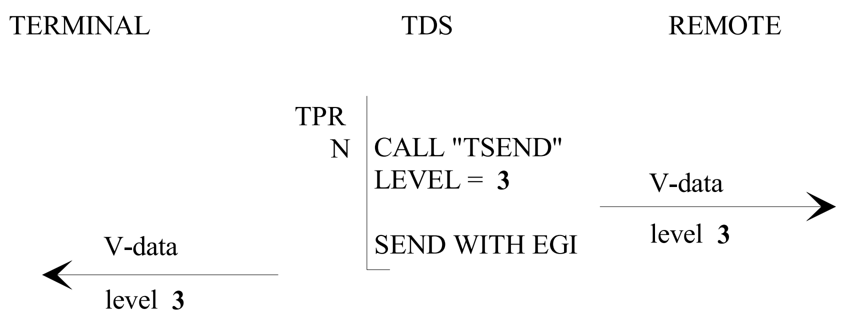


Figure E-5. Two Conversations in the Same TPR

- While the RECEIVE verb (when successful) always give the turn to the transaction in the principal session, the CALL "TRECV" (when successful) does not always give the turn to the transaction in the pass-thru session.
- If the programming rules are not applied, the TDS ensures that data that the transaction receives is not lost.
  - The transaction must use the RECEIVE verb in the principal session, after receiving a status of 41 on the CALL "TEVENT". The transaction can do this either in the same TPR or in succeeding TPRs. For example, this could occur after a call statement in the pass-thru session. The message is not lost, and no other message from the terminal can be received first.
  - The transaction must perform a CALL "TRECV" in the pass-thru session after it receives a status of 33. For example, this could occur after dialoguing in the principal session. The transaction can do this either in the same TPR or in succeeding TPRs, and the data is not lost.

Command Sequencing Examples

EXAMPLE: Opening the Pass-Thru Session and the Remote Application Keeps the Turn

The following diagram shows how the CALL "TOPEN" opens the pass-thru session. The remote application keeps the turn.

- The CALL "TOPEN" requests to open the pass-thru session. The CALL "TEVENT", with status 31, indicates that the session is open and that the turn goes to the correspondent accepting the connection. In this case, that is the remote application.
- After the CALL "TOPEN", the transaction must wait for the CALL "TEVENT" to give the connection results. The CALL "TEVENT" must be in the next TPR.
- The CALL "TEVENT" of TPR N+1 can return the connection results, but it also returns some principal session event information. For example, it returns a status of 42 (V-CREDIT) if the TPR sends a message to the terminal (SEND with EMI), or it returns a status of 43 (V-ATTENTION) if a break occurs at the terminal.

❑

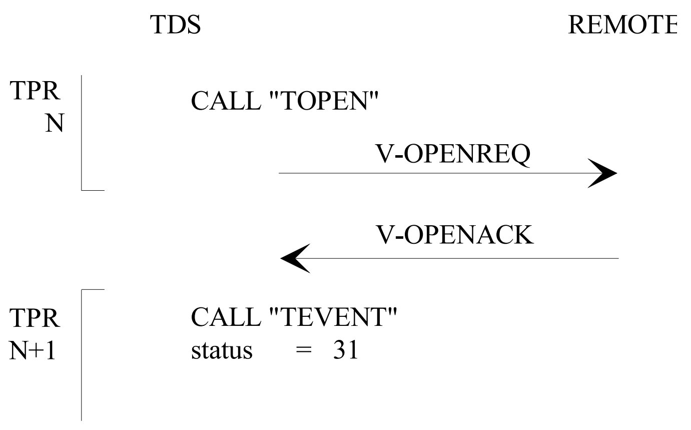


Figure E-6. Opening the Pass-Thru Session

EXAMPLE: Receiving Data without Turn in the Pass-Thru Session and Sending It to the Terminal

The following diagram shows the TDS receiving data from the remote application, without having the turn. The TDS then sends data to the terminal.

- The TPR N has to receive data in the pass-thru session with CALL "TRECV" because the TPR N returns the status of 33 (V-DATA) on CALL "TEVENT".
- The TPR N then sends this data to the terminal (SEND with EMI). The principal session waits for a V-CREDIT.
- The remote application can send a new V-DATA level 1 in the next TPR (TPR N+1) in code 33 on the CALL "TEVENT". However, a V-CREDIT must be received before sending a new message.
- At the end of TPR N, the transaction has the turn in the principal session, and the remote application has the turn in the principal session. This is the same as at the start of TPR N.

❑

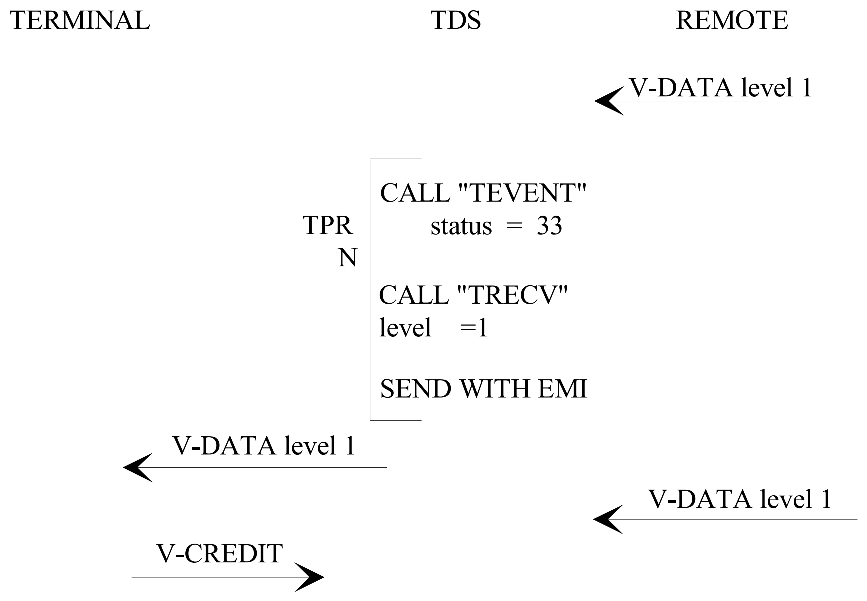


Figure E-7. Receiving Data without Turn in the Pass-Thru Session

EXAMPLE: Transaction Receives Data with the Turn in the Pass-Thru Session and Sends It to the Terminal

The following diagram shows the transaction receiving data, with the turn, from the remote application. The transaction then sends data to the terminal.

- The TPR can use a CALL "TRECV" because TPR N contains a status of 33 on the CALL "TEVENT". In this way, the transaction receives the data coming from the remote application.
- TPR N sends the data to the terminal (SEND with EGI). First however, the transaction must ensure that, if the preceding verb in the principal session was a SEND with EMI, it received a status of 42 (V-CREDIT) on CALL "TEVENT".
- After TPR N sends the data to the terminal, the transaction waits for a V-DATA level of 3 in the principal session.
- At the end of TPR N, the terminal has the turn in the principal session, and the transaction has the turn in the pass-thru session.

❑

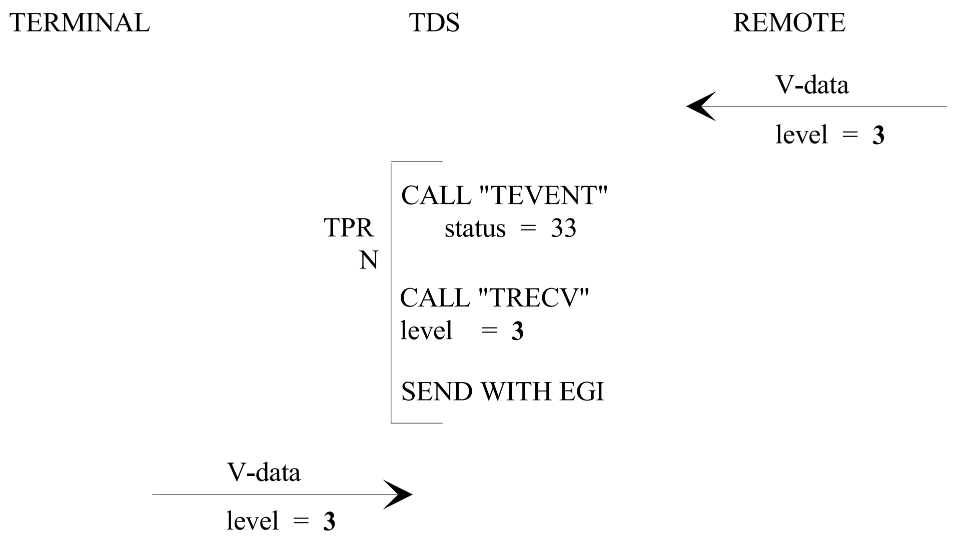


Figure E-8. Transaction Receives Data with Turn in the Pass-Thru Session

EXAMPLE: Transaction Receives Data with the Turn in the Principal Session and Sends It to the Remote Application

The following diagram shows the transaction receiving data, in the principal session, from the terminal. The transaction has the turn. The transaction then sends data to the remote application.

- The TPR can use a RECEIVE verb because TPR N contains a status of 41 on the CALL "TEVENT". In this way, the transaction receives the data coming from the terminal.
- TPR N sends the data to the remote application, using call "TSEND". At the same time, the remote application gets the turn.
- After TPR N, the pass-thru session waits for a status of 33 (V-DATA).

❑

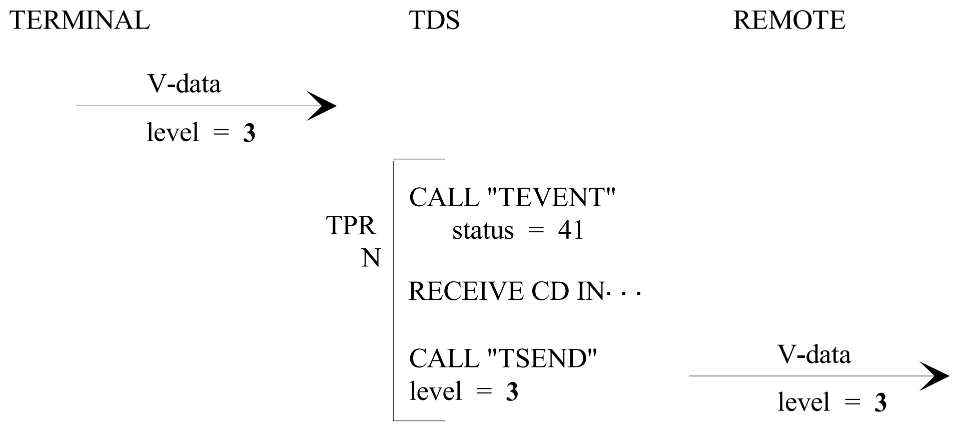


Figure E-9. Receiving Data without the Turn in the Principal Session

EXAMPLE: Normal End of the Pass-Thru Session

The following diagram shows a pass-thru session ending normally. This example uses the TCAM.

- The TCAM (TDS Communication Access Method) performs a pass-thru that makes TDS become transparent to the end user. In this way, message 1, "BYE", is addressed to the remote.
- TPR N receives the data in message 1 with the RECEIVE verb. It then uses "TSEND" to send "BYE" (message 1) to the remote application. This becomes message 2. TPR N sends the turn along with the message.
- The remote application receives message 2, and closes the session. The remote then sends "data of level 5", which becomes message 3.
- TPR N+1 can perform a call "TRECV" because it receives a status of 33 on the call "TEVENT". This is in accordance with the programming rules shown above.
- TPR N+1 sends "data of level 5" in the next message (which is message 4). This closes the session. Message 3 is a request to close, and message 4 is the acknowledgement of the close.

  The pass-thru session is now closed, and the transaction is no longer in pass-thru mode.
- TPR N+1 owns the turn in the principal session, and send data towards the terminal in message 5.

❑

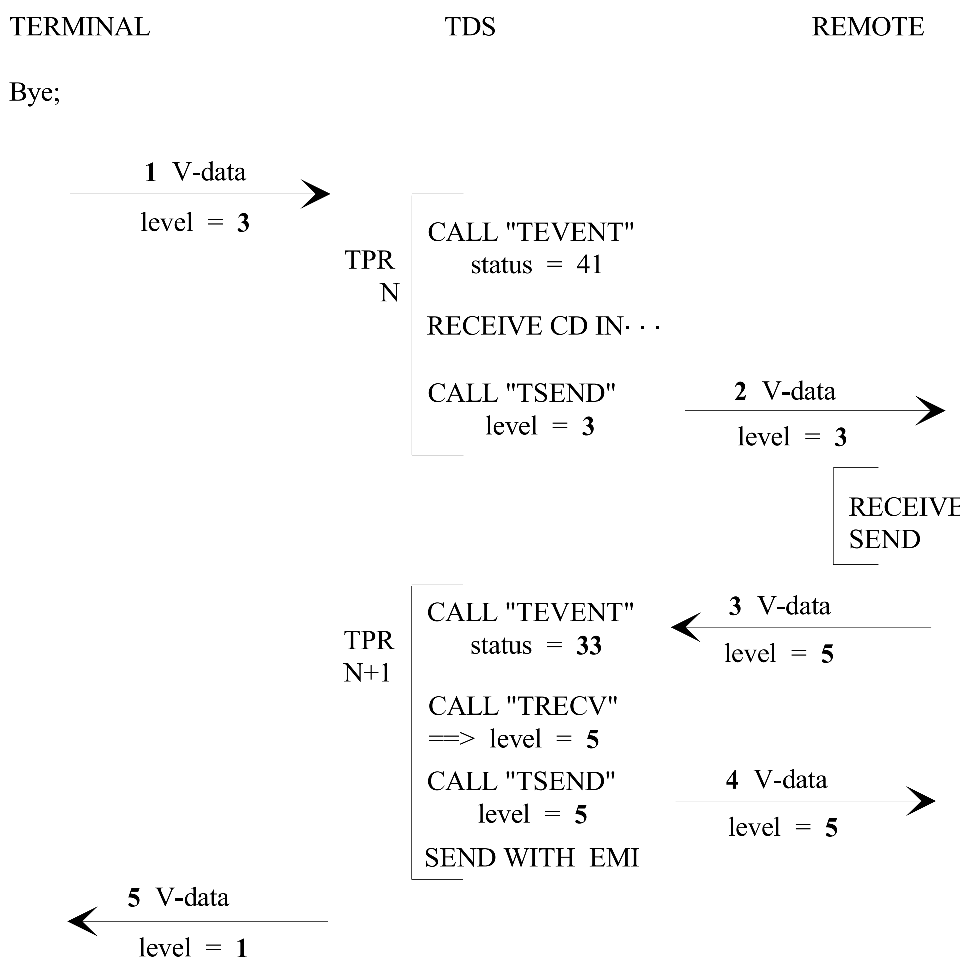


Figure E-10. Normal End of Pass-Thru Session

E.7 Abnormal Disconnections

The terminal can be abnormally disconnected during either the principal or pass-thru session.

E.7.1 Principal Session Disconnections

When a terminal disconnects abnormally, it sends two events on the CALL "TEVENT" to the TDS:

- status code 44 (V-ABNTERM), which is an interruption. This information should be stored in TRANSACTION STORAGE.
- status code 40 (V-MGCLOSED), which is an event. After receiving this, the pass-thru session should be closed using code 35 on the CALL "TCLOSE" statement. This warns the TDS about the principal session disconnection, and the pass-thru session is closed.

These two events are shown in the diagram below.

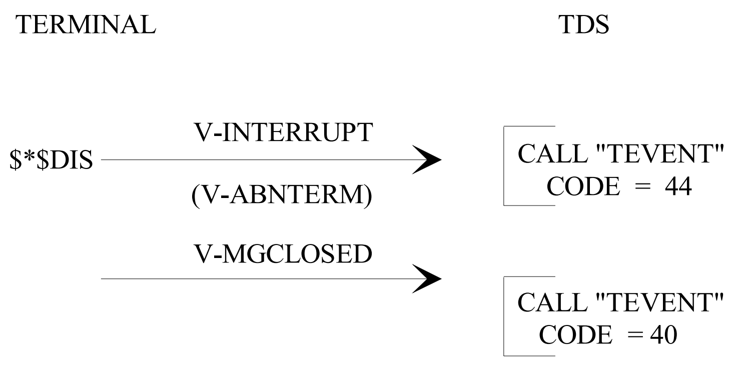


Figure E-11. Disconnecting the Principal Session

NOTE:
The principal session disconnection does not force the user to close the pass-thru session, too. The transaction can continue to dialog in the pass-thru session. However, no principal session verbs are allowed.

E.7.2 Pass-Thru Session Disconnections

When the pass-thru session disconnects abnormally as follows:

- The TPR receives a status of 35 (V-INTERRUPT) in the CALL "TEVENT".
- The TPR must send a CALL "TRCVIT". This returns a type = 3, which indicates that the pass-thru session has disconnect abnormally.
- Before leaving pass-thru mode, the transaction must wait for the last event of this session, which is a status of 40 (V-MGCLOSED) in the CALL "TEVENT".
- After this, the transaction is no longer in pass-thru mode.

These events are shown in the diagram below.

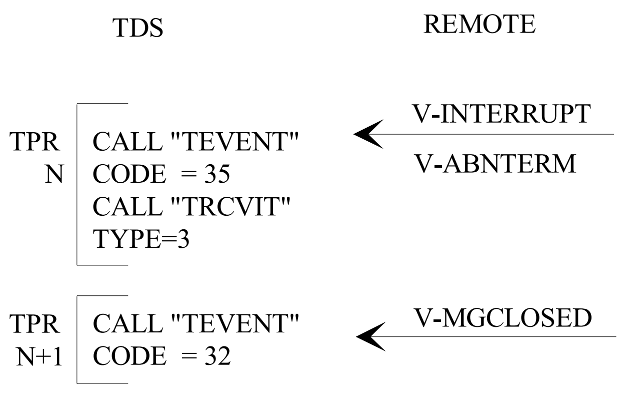


Figure E-12. Disconnecting the Pass-Thru Session

E.7.3 Aborts, GCOS 7 Crashes, and Recovery

The pass-thru mode follows the same rules as the TDS when a transaction aborts.

When there is a GCOS 7 crash, the TDS closes all sessions (pass-thru or not).

There is no recovery of a pass-thru session, and no message recovery. The pass-thru session are not automatically opened again after a master command ALLOW_NEW_TDS_COR.

Index

Technical publication remarks form

Title :	DPS7000/XTA NOVASCALE 7000 TDS COBOL Programmer's Guide

Reference N° :	47 A2 33UT 08	**Date:**	**February 2005**

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 D$^{ept.}$
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	**Phone:**	+33 (0) 2 41 73 72 66
B.P.20845	**FAX:**	+33 (0) 2 41 73 70 66
49008 ANGERS CEDEX 01	**E-Mail:**	srv.Duplicopy@bull.net
FRANCE		

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 33UT 08