TDS-TCP/IP

User's Guide

DPS7000/XTA
NOVASCALE 7000

REFERENCE
47 A2 37UT 07

DPS7000/XTA
NOVASCALE 7000
TDS-TCP/IP

User's Guide

Software

February 2005

BULL CEDOC

357 AVENUE PATTON

B.P.20845

49008 ANGERS CEDEX 01

FRANCE

REFERENCE
47 A2 37UT 07

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

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

Trademarks and Acknowledgements

Preface

TDS-TCP/IP allows an application running on a Windows, Linux or AIX platform to dialog with a TDS transaction on GCOS 7 in client/server mode via a TCP/IP communication link.

The application on the station is called the **client** application, while the TDS transaction running on GCOS 7 is called the **server** application.

Scope and Objectives

This document presents the configuration commands, variations from standard TDS master commands, and a subset of C-language XATMI functions for client application development. Where applicable, references are given to relevant information in the *TDS Administration Guide* and associated documents.

The list of associated documents is given below with reference numbers.

Intended Readers

This guide is intended for:

- administrators and operators concerned with TDS generation and supervision,
- programmers responsible for developing client/server applications running on a station that dialogs with a TDS transaction through a TCP/IP network.

Structure

This guide first gives an overview of TDS-TCP/IP, then describes the commands required for its installation and use, and the API to be used for client application development.

The summary of the contents of this manual is as follows:

Chapter 1	summarizes the features and requirements of TDS-TCP/IP.
Chapter 2	describes the GCOS 7 TS 9764 added functionalities.
Chapter 3	describes commands executed for TDS administration.
Chapter 4	describes the API to be used for client application development.
Chapter 5	describes error handling.
Chapter 6	describes TCP/IP transactions using the FORMS facility.
Chapter 7	describes the protocol between SA7 and the client application.
Chapter 8	describes the use of a client application on AIX and Linux platforms.

Bibliography

This guide is a member of the set of TDS manuals:

TDS Concepts *47 A2 26UT*
TDS Administrator's Guide *47 A2 32UT*
TDS COBOL Programmer's Guide *47 A2 33UT*
TDS C Language Programmer's Guide.. *47 A2 07UT*
TDS Quick Reference Handbook... *47 A2 04UT*

The following publications give information on topics related to running a TDS application:

For configuring TDS-TCP/IP:
XTI GCOS 7 Name Services User's Guide *47 A2 69UC*
OPEN 7 Administrator's Reference Manual Vol. 1..................................*47 A2 82UC*
OPEN 7 Administrator's Reference Manual Vol. 2..................................*47 A2 83UC*
OPEN 7 Administrator's Reference Manual Vol. 3..................................*47 A2 84UC*

For generating the DPS 7000 network:
Network Overview and Concepts *47 A2 92UC*
Network Generation *47 A2 93UC*

For main console operator commands:

Network User's Guide..*47 A2 94UC*
GCOS 7 System Operator's Guide.. *47 A2 53US*

For status values and return codes:

Messages and Return Codes Directory *47 A2 10UJ*

Syntax Notation

Conventions for entering TP7GEN syntax

<u>UPPERCASE</u>	indicates that this underlined item is a reserved keyword, which must be specified if the clause in which it appears, is required.
UPPERCASE	indicates a reserved keyword that must be coded as shown. It may be omitted.
item	italics indicates a term for which the user supplies a value.
[item]	optional entry
{**item**|item|item}	a list of items within braces means that only one is to be selected if the introducing parameter is specified. The item heading the list and appearing in bold is the default value if the introducing parameter is not specified. This list of parameters can also be vertical: {item} {item} {item}
item . . .	Three dots following an item indicate that more items having the same form may appear.
Clause . . .	Three vertical dots mean that part of a clause has been intentionally omitted.
||A|| ||B|| ||C||	Clauses enclosed in double vertical bars indicate only one occurrence of each.

Conventions for entering the master command syntax

You can use this syntax:

- if you are logged on under a master mailbox
- or if the command is:
  - stored in a subfile for execution by the M EXEC_TDS command
  - to be executed via the Batch Interface
  - to be executed via the spawning mechanism.

Command names and parameters follow the same naming convention. Separate:

- the command name and the first parameter by at least one blank
- parameters from each other by blanks or commas.

Each master command is followed by various position parameters and keywords. A parameter can be specified as:

- a keyword introducing an argument
- a position parameter.

Keywords can be specified in any order. Both keywords and position parameters can be mixed, in which case position parameters are interpreted as the values for parameters whose rank is determined by the preceding keyword, if any.

Parameters must be constants. Use the following data types:

- character
- decimal
- boolean
- name
- star-name
- file.

EXAMPLE:

```
M CLOSE_TDS_FILE IFN=T1, DEASSIGN=1
```

Because full GCL facilities are not available, do not:

- use built-ins or GCL variables
- mix quoted and unquoted strings
- nest parentheses.

❑

Table of Contents

Table of Graphics

1. Overview

1.1 Features of TDS-TCP/IP

1.1.1 General Features

TDS-TCP/IP allows an application located on a Windows, Linux or AIX platform to dialog with a TDS transaction on GCOS 7 according to a Client/Server model.

The application located on the station is said to be the **client application** while the TDS transaction running on GCOS 7 is the **server application**.

This conversational dialog is made up of messages exchanged via the network; it takes place through the "de facto" standard communications network known as TCP/IP that provides native TCP/IP support in a GCOS 7 TDS processing environment.

Access to the TCP/IP network is via a layer of network services named the socket interface (often abbreviated to **socket**). This layer is implemented on GCOS 7 by a component called **SOCKG 7** (SOCKet for GCOS 7), while on the stations, the interface is a part of the operating system .

At application level (i.e., the client application and the server application), the conversational dialog must respect the client/server rules defined by a subset of the **XATMI** interface.

On GCOS 7 (server)

- Existing TDS transactions can be used by a client application; these transactions can be either in **line** mode or in **formatted** mode (i.e., using the **FORMS** facility).
- At TDS transaction level, respect of the XATMI protocol is transparent; this is done at a lower level by a new GCOS 7 component named **SOCKG 7** that converts the TDS specific verbs to the appropriate functions of the socket interface.
- On DPS 7000 / TA, SOCKG 7 has access to lower communications layers (sockets) using either OPEN 7 services or GXTI services. On DPS 7000 / XTA, SOCKG7 is available by using an INTEROP 7 gateway.
- Generation and administration of the TDS TCP/IP link use TP7GEN and TDS master commands.

On the station (client)

- The client application dialogs with a TDS transaction using a specific API provided by BULL and installed on the station in a standard **DLL** (**D**ynamic **L**ink **L**ibrary) on Windows machines or a shared library on Linux or AIX machines.
- The API provided is a set of primitives, written in C language, which are XATMI compliant; this allows the customer to develop the client application with all **Rapid Application Development** tools (**RAD**) that can call the functions provided by this API (such as WINDEV, VB, etc.).

1.1.2 Definitions and Terminology

The new features provided by TDS-TCP/IP introduce the concept of another kind of transaction. These transactions are referred to as TCP/IP transactions to distinguish them from other transactions running for OSI/DSA correspondents (such as TM, XCP1, XCP2, etc.).

A TCP/IP transaction is a TDS transaction located on a DPS 7000 system started by a client program located on a station, which converses with it through the socket interface as described above.

A TCP/IP transaction is executed by a ***TCP/IP correspondent***, during a ***TCP/IP correspondent session***.

A TCP/IP transaction dialogs with a client application located on a station via a ***TCP/IP communication link***.

Other terms and abbreviations are defined in the Glossary.

The following diagrams show the components that interact via TDS-TCP/IP:

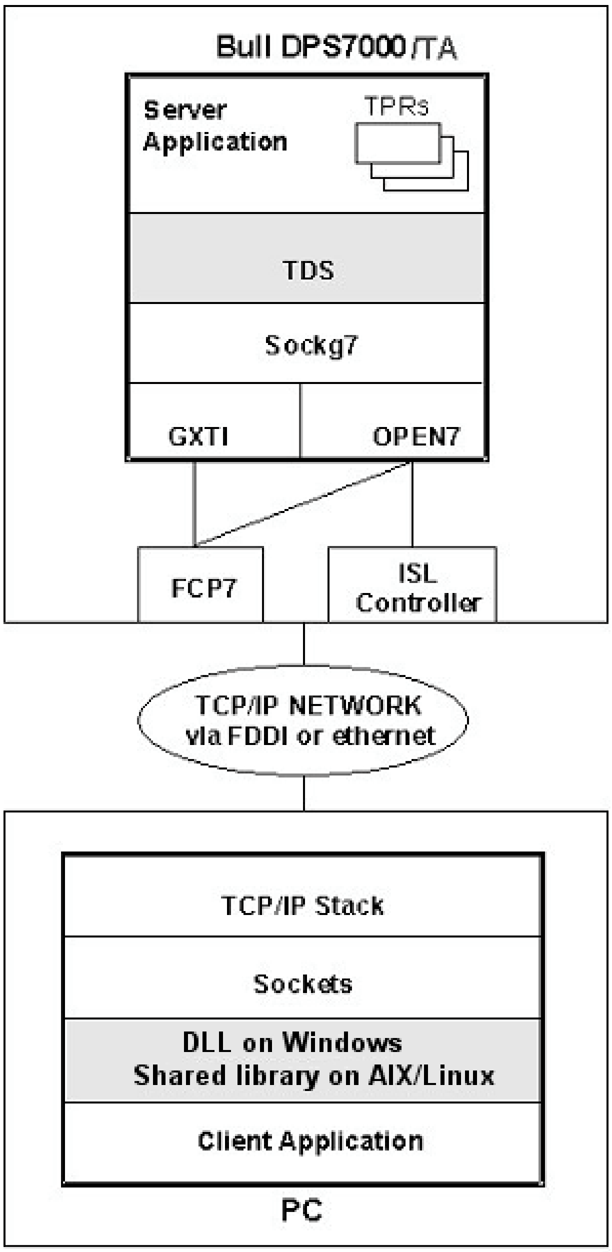

Figure 1-1. Components of TDS-TCP/IP on DPS7000 /TA

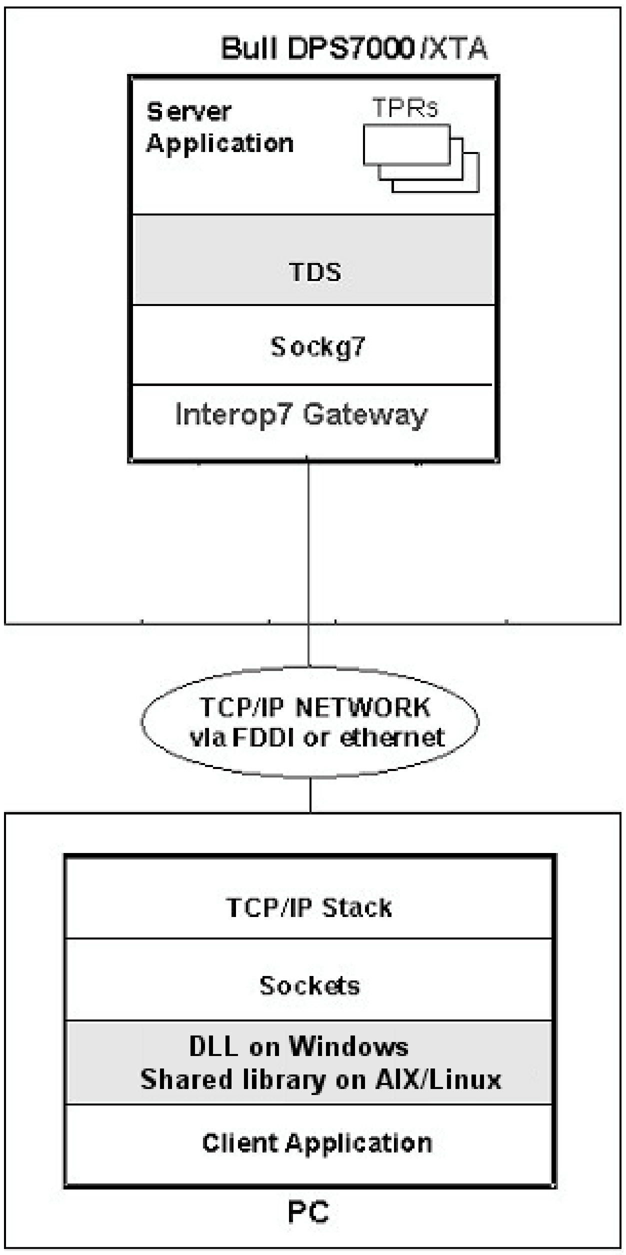

Figure 1-2. Components of TDS-TCP/IP on DPS7000 /XTA

1.1.3 Existing TDS Transactions

TDS-TCP/IP has an impact on TDS at generation level and at administration level, but does not affect TPRs already coded. Such TPRs can be launched by a TCP/IP correspondent without any modification (as they are) and whatever their presentation mode (line mode or formatted mode).

Within the same TDS, OSI/DSA transactions can run simultaneously with TCP/IP transactions.

In a given TDS, a transaction may be running simultaneously for a TCP/IP correspondent and for an OSI/DSA correspondent.

Moreover, a given transaction can be running at the same time for a terminal correspondent and for a TDS-TCP/IP correspondent.

1.1.4 TDS-TCP/IP Client API

Bull provides a **TDS-TCP/IP client API** for the development of client applications that communicate with the TDS via a TCP/IP link. This API accesses functions on the stations via a DLL or a shared library coded in C-Language.

It is the **customer's responsibility** to verify that the tool he wants to use for client application development (C compiler and/or RAD tools) can correctly use the TDS-TCP/IP client API provided by BULL.

The API is composed of the following functions written in C-Language:

- functions for conversational services: *tpconnect*, *tpdiscon*, *tprecv* and *tpsend*,
- functions for typed buffer management: *tpalloc* and *tpfree*.

Some complementary error management functions have been added to the standard XATMI functions: *tperrno* and *tperrdtl*.

The functions provided by this API are available by accessing the DLL on the PC or the shared library on AIX or Linux.

These functions are fully described in the chapter *Client Application Development*.

1.2 Users

Two types of users are directly concerned with TDS-TCP/IP feature(s). They are:

- those in charge of TDS generation, and the associated GCOS 7 configuration and supervision,
- developers of client applications running on a PC that use TDS services.

In addition, several types of users interact to enable the operation of TDS-TCP/IP:

- the TDS administrator, responsible for the generation step and the monitoring of TDS applications,
- the GCOS 7 administrator, responsible for network generation,
- the OPEN 7 administrator, responsible for the installation and configuration of OPEN 7 and SOCKG 7 on DPS7000/TA, or the INTEROP7 administrator, responsible for the installation and configuration of INTEROP7 products on DPS7000/XTA
- the PC administrator, in charge of product installation and system configuration,
- the Telecommunication and Network administrator, responsible for address declarations and coordination for all types of machines, since TDS-TCP/IP uses heterogeneous hardware.

1.3 Prerequisites

1.3.1 Hardware

DPS 7000 Server

Any DPS 7000/TA or DPS 7000/XTA system running:

- GCOS 7 V9 TS 9866
- GCOS 7 V10 TS 9910 and upper

On DPS 7000/TA, an **FCP7** or an **ISL** controller to enable connection to a TCP/IP network using **FDDI** or **Ethernet** technologies.

Windows, AIX or Linux client

Any station having an appropriate communications card allowing connection to a TCP/IP network and capable of running the software listed below.

1.3.2 Software

On DPS 7000/TA server:

GCOS 7

The GCOS 7 component of TDS-TCP/IP is integrated in TDS and is available with:

- GCOS 7 V9 TS 9866
- GCOS 7 V10 TS 9910.and upper

SOCKG 7

TDS-TCP/IP uses SOCKG 7 sockets on GCOS 7. SOCKG 7 offers a standard socket interface that supports two methods of communication:

- one provided by OPEN 7 services which are available from OPEN 7 V5,
- the other provided by GXTI services, available with the GCOS 7 releases listed above.

SOCKG 7 and OPEN 7 V5 can be installed with the INTEROP 7 facility called **ISI 7** minimum tape version I5310.

On DPS 7000/XTA server:

- The supported operating systems are GCOS 7 V10 TS 9870 and upper
- V7000 version 1.5 minimum
- INTEROP7_BASIC version 2.3.0 minimum (which includes SOCKG7 and the sharable module H_SM_DCM)

On AIX Computer:

- The operating system supported is AIX 4.3

On Linux Computer

- The shared library is currently being built on Red Hat 7.3 system

1.4 Delivery

1.4.1 Software for Windows

The Windows operating systems that allow development and execution of client applications are Windows 95/98/ME and Windows NT/2000/XP.

From version 4.2.1, the package TDS_TCP_APIW is delivered with Interop7 from CD ID330. It includes an executable file, which will install the product TDS_TCP/IP on windows.

Five files are delivered:

- atmi.ini which contains DLL options for trace and time-out. It must be located in the Windows directory (usually C:\WINNT).
- atmi32.dll which contains the DLL itself. It should be located in the Windows system directory (usually C:\WINNT\SYSTEM32). The DLL is a 32-bit version for Windows .
- atmi.h, an include file which contains the description of the connection structures, the external function prototypes, the event types, and error codes.
- atmi32.lib that contains the XATMI standard definition (this file is needed for development purposes but not for execution of the client application).

The atmi.h and the atmi32.lib files must be put in the client working directory used for application development

- Atmi32.bas is a model to program an application VB and may be put in the client working directory used for application development.

1.4.2 Software for Linux

The package TDS_TCP_APIX, which includes :

- A shared library XATMI
- An include file ‘Atmi.h’

is an Interop7 product. The release 2.1.1 is delivered from CD ID330 (for DPS7000/XTA) and from CD I5322 (for DPS7000/TA) as a ‘tar’ archive.

The shared library is currently being built on Red Hat 7.3 system.

1.4.3 Software for AIX

The package TDS_TCP_APIX, which includes :

- A shared library XATMI
- An include file ‘Atmi.h’

is an Interop7 product. The release 2.1.1 is delivered from CD ID330 (for DPS7000/XTA) and from CD I5322 (for DPS7000/TA) as a ‘tar’ archive.

The shared library is currently being built on AIX 4.3 system.

1.5 Configuration

1.5.1 TDS Generation

To enable the TDS to communicate via TCP/IP, the TDS SECTION of the STDS file must be adapted. The STDS is the input file for generation processing, and describes the characteristics of the resulting TDS-TCP/IP, a transactional application.

TDS generation incorporates two important activities:

- declaration of the TCP/IP protocol,
- selection of the SOCKG 7 communication link (either OPEN 7 or GXTI).

1.5.2 SOCKG 7 Sockets

Two different sets of files are usable for configuration purpose according to the SOCKG 7 communication link to be used, so the socket selection is important.

SOCKG 7 configuration incorporates two important activities:

- declaration of the TCP/IP service,
- providing details of relevant host-client information.

TDS generation and SOCKG 7 configuration are explained in the chapter *Server Administration*.

1.5.3 Client stations

The configuration of the stations incorporates two important activities:

- declaration of the TCP/IP service,
- providing details of relevant host-client information.

1.6 Security

For each TDS-TCP/IP connection, the client application must provide:

- a user identification,
- a password,
- a project,
- a billing.

Verification is then made by comparison with the GCOS 7 catalog information.

Before GCOS 7 TS 9764, a TDS running TCP/IP transactions cannot be controlled by SA7 (Secur'Access). GCOS 7 TS 9764 enables the control by Secur'Access. The Access Master functionality is not supported.

1.7 API Programming

The client API allows the client application to dialog with TDS transactions.

Two functions provide the buffer management and are local to the client applications:

- tpalloc which allocates typed buffers,
- tpfree which frees typed buffers.

The interface between the client application and TDS includes six functions that are called by the client:

- tpconnect which logs the TCP/IP client to TDS,
- tprecv which receives TDS messages,
- tpsend which sends message to TDS,
- tpdiscon which disconnects the client from TDS in strong mode,
- tperrno and tperrdtl for error management (extension of the XATMI protocol).

API programming is fully explained in the chapter *Client Application Development*.

1.8 Limitations

Although most TDS services can be used by TDS-TCP/IP transactions, restrictions apply to:

- Some TPR programming functions.
- Some session management procedures.
- The terminal BREAK function which is not supported.
- Service messages, SEND verbs with routing addresses and M SNDTU commands; none of these are sent to TCP/IP correspondents.
- There is no header and trailer in message presentation.

Since GCOS 7 TS 9764 the passthru function is allowed, for details please report to the **GCOS 7 TS 9764 added functionalities** part of this document.

1.8.1 TPR Programming Functions

Existing TPRs can be executed as they are and without any modification.

New TPRs may be developed following the same method as used previous to the TDS-TCP/IP interface, but the following restrictions need to be taken into account:

Since GCOS 7 TS 9764 the SPAWNING on a TCP/IP correspondent is allowed using CALL DSPAWN/SPAWN/SPAWNTX/TSPAWN. The behaviour of the spawning, related to this correspondent type, is described in the **GCOS 7 TS 9764 added functionalities** part of this document.

1.8.2 Session Management Procedures

The following session management procedures cannot be used:

- RECONNECT-OPTION. An error status: - wrong session type - is returned for a TCP/IP correspondent.

Since GCOS 7 TS 9764, the CALL SET-ACTIVE/SET-PASSIVE functions related to a TCP/IP correspondent are allowed. For details please report to the **GCOS 7 TS 9764 added functionalities** part of this document.

1.8.3 Configurability Rules

TDS is able to support up to 4000 active sessions including OSI/DSA sessions and TCP/IP sessions.

The maximum number of TCP/IP users that can be connected simultaneously to a TDS application is 1000. These 1000 sockets are shared by all the TDS applications of a same GCOS 7 system. Starting from GCOS 7 TS 9910, this number is raised to 3000 on DPS 7000 / XTA.

The maximum number of TCP/IP connections per client process is 64. From the DLL version 3.1.0, this number is raised to 500.

2. Added functionalities

2.1 AIX and Linux Clients

An AIX version and a Linux version of the TDS-TCP/IP API are provided. They allow a client application located on an AIX or Linux machine to dialog with a TDS transaction on GCOS7.

The API is installed on the AIX or Linux machine in a standard shared library.

The Chapter 8 of this document describes the use of TDS-TCP/IP for AIX and Linux clients.

2.2 GCOS 7 TS 9764 added functionalities

2.2.1 Spawning on a TCP/IP correspondent

A call SPAWN/DSPAWN/TSPAWN/SPAWNTX to a TCP/IP correspondent is allowed. If the correspondent is frozen, a status 3 (or 17 wrong type on SPAWNTX) is returned.

The reconnection of a frozen TCP/IP correspondent using the TDS master command ALNTC is without effect; the correspondent must be reconnected to accept a new spawning.

2.2.1.1 Active TCP/IP correspondent

A spawned transaction can be started onto this correspondent type, only after the termination of a running transaction.

If a running transaction, related to this kind of correspondent, makes a call SPAWN toward itself, this spawned transaction will be automatically started at the end of the running transaction.

A transaction spawned from another correspondent (TCP/IP or other type) can be started on the TCP/IP active correspondent at the end of a running transaction. If no transaction is running, the further transaction started by *tpsend* () will be executed, the spawned transaction will be started at the end of the transaction.

2.2.1.2 Passive TCP/IP correspondent

When a spawned transaction is started by TDS for this correspondent type, the client receives the message sent by the spawned transaction using the *tprecv* () function. The client is not allowed to launch a transaction by *tpsend* () because the correspondent is passive.

However the client using *tprecv* (), may be awaiting a message sent by a spawned transaction.

The *tprecv* function is synchronous so the process executing the function is blocked until either the reception of the data coming from the server or the expiration of the time-out value given in the Atmi.ini file. During this time, the execution of the program is stopped, next actions generated in the program are differed.

2.2.2 Call SET-ACTIVE/SET-PASSIVE

These TDS functions are allowed.

Note that the spawning behaviour is different for an active or a passive TCP/IP correspondent, see above the TCP/IP spawning.

A TCP/IP correspondent is always connected or reconnected in ACTIVE mode.

2.2.3 MAXIMUM IDLE TIME

Since GCOS 7 TS 9764, the maximum idle time applied to a TCP/IP correspondent leads to an abnormal disconnection (the disconnect transaction, if any, is launched).

2.2.4 WAIT-TIME

The WAIT-TIME for a TCP/IP correspondent does not enable the automatic execution of the next TPR when the current TPR ends with a SEND EGI.

In this case, the execution of the next TPR is started by the client response message to the SEND EGI (*tpsend* ()).

If the TPR ends without SEND or with a SEND EMI, the time given in the WAIT-TIME is normally taken into account.

2.2.5 TERMINAL_ID

Since GCOS 7 TS 9764, using a DLL version 3.0.6, a terminal identification may by supplied to the *tpconnect* function using the *termid* field in the *data* parameter. If the termid field is supplied with a value different from spaces, it is moved in the TERMINAL_ID field of the TRANSACTION-STORAGE of the LOGON transaction. If the termid field is filled with spaces, or not used, or null, or if the DLL version is 3.0.5 the TERMINAL_ID field of the TRANSACTION-STORAGE of the LOGON transaction is initialized by TDS to the specific value **H-TCPIP-CLI**.

2.2.6 DISCONNECTIONS issued by TDS

Since GCOS 7 TS 9764, the maximum idle time of the TDS generation applied to a TCP/IP correspondent leads to an abnormal disconnection issued by TDS.

TDS disconnects abnormally the correspondent if there is no incoming message from the client during 2 minutes after TDS has requested the turn (because the action to start needs the turn).

2.2.7 SECUR'ACCESS

A TDS TCP/IP application can be protected by Secur'Access.

A TCP/IP correspondent connected to a TDS controlled by Secur'Access, executes the dedicated Secur'Access transactions and the dialogs must be done according to the format of their messages. Please report to the part **Protocol between SA7 and the Client application** of this document for details.

The Access Master functionality is not supported.

2.2.8 PASSTHRU functionality

Since GCOS 7 TS 9764, the Passthru functionality can be activated from a TCP/IP correspondent. The requested connection to a TDS or IOF application is done in a DSA environment using a DKU7105 terminal model (and a TM correspondent type for the target TDS).

If the target application is TDS, the messages are received without header and trailer in line mode. When the formatted mode (FORMS) is activated in a transaction, the messages are received in the TDS-TCP/IP FORMS format.

If the target application is IOF, only the line mode must be used because the TDS-TCP/IP FORMS format is only used for TDS. The messages sent by IOF in line mode contain some presentation characters corresponding to the DKU7105 model as the terminal identification, the cursor position, i.e. these messages have not the FORMS TCP/IP format.

2.3 GCOS 7 TS 9920 added functionalities

2.3.1 Reconnect with option Force

In certain cases, TDS does not detect that a TCP/IP connection is broken. The user cannot reconnect to TDS because from TDS point of view, he is still connected. From now, the application can use the option "logon force" (see paragraph 4.3.2) to disconnect the user, then to reconnect him with same the logon context.

3. Server Administration

The server administration of TDS-TCP/IP consists of:

- TDS administration for the TDS generation and the control of the TDS session using master commands (TDS preparation is not impacted by TDS-TCP/IP).
- GCOS 7 administration for the configuration of the TDS-TCP/IP services.

3.1 TDS Generation

The use of the TCP/IP communication link must be declared at TDS generation.

To generate the TDS:

- Modify the TDS source member (STDS) to include:
  - the TCP-IP PROTOCOL clause (see below),
  - the ATTACH SHARABLE MODULE clause (see below).
- Generate the TDS using TP7GEN (see the *TDS Administrator's Guide* for details).

3.1.1 TCP-IP PROTOCOL Clause

Syntax

```
[TCP-IP PROTOCOL [USED] [WITH {OPEN7 |GXTI }].]
```

This clause specifies the use of the TCP/IP protocol and enables selection of OPEN 7 or GXTI as the communication link. OPEN7 is the default value.

On DPS 7000 / XTA, just specify :

```
TCP-IP PROTOCOL.
```

The position of the TCP-IP PROTOCOL clause is just before the first USE statement (if any) and after the XA-RESYNC-DELAY clause in the TDS SECTION of the STDS.

3.1.2 ATTACH SHARABLE MODULE Clause

Syntax

```
[ATTACH SHARABLE MODULE H_SM_DCM.]
```

This (already existing) clause must also be declared to attach SOCKG 7.

The position of the ATTACH SHARABLE MODULE clause is just after the RESERVE AREAS clause in the TDS SECTION of the STDS. See the *TDS Administrator's Guide* for more details.

3.1.3 Errors and Responses

The following TP7GEN error message can occur:

TG28 *sev* **WRONG SYNTAX IN CLAUSE OR STATEMENT:** *clause.*

where *clause* is the following:

TCP-IP PROTOCOL

Type:	Information: *sev*=2
Meaning:	Indicates the clause or statement in which a syntax error has been detected.
Action:	Correct the syntax of the clause or statement, and re-run TP7GEN.

3.2 GCOS 7 Environment Configuration

TDS-TCP/IP, like other Internet services, has to be declared at GCOS 7 level. Two configurations are possible depending on the communication link used for the socket, either OPEN 7 or GXTI.

3.2.1 Description

Two types of information must be declared in the configuration files, **/etc/hosts** (for OPEN 7) or **ETC_HOSTS** (for GXTI):

- the IP addresses of client hosts,
- the service names.

3.2.2 Declarations for OPEN 7

Update the **/etc/hosts** file with the IP address of the host name, which is the symbolic name used to address the DPS 7000 system. See the *OPEN 7 Administrator's Reference Manual.*

Update the **/etc/services** file of the OPEN 7 subsystem with the service names, their related port number, and the protocol used (i.e., TCP). See the *OPEN 7 Administrator's Reference Manual*. The TDS-TCP/IP service is identified by concatenation of the host name and the TDS name. A port number is associated with each service. It must be the same as the one declared on the PC for the same service, and must not be in conflict with the port numbers of other services. The value of the port number must be greater than 1024.

EXAMPLE:

If a client application wants to connect to the TDS `TDS1` located on the DPS 7000 system referred to by the host name `BC0F` the following line must appear in **/etc/services**:

```
bc0ftds1   10100/tcp
```

❑

3.2.3 Declarations for GXTI

The **ETC_HOSTS** subfile in the GCOS 7 library **SYS.DSACONF** must contain the same information as the /etc/hosts file located in the OPEN 7 subsystem. See *XTI GCOS 7 Name Services User's Guide*. To comply with GXTI conventions, the host name is `DPS7000_name-xti` in lower case.

EXAMPLE:

```
193.128.250.85  bc0f-xti
```

❑

The **ETC_SERVICES** subfile in the GCOS 7 library **SYS.DSACONF** must contain the same information as /etc/services file located in the OPEN 7 subsystem. See *XTI GCOS 7 Name Services User's Guide*. Check that the standard service : "echo" is also declared in ETC_SERVICES.

EXAMPLE:

If a client application wants to connect to the TDS `TDS1` located on the DPS 7000 system referred to by the host name `bc0f-xti` the following lines must appear in **ETC_SERVICES**:

```
echo          7/tcp
bc0f-xtitds1  10100/tcp
```

❑

3.3 Master Commands

3.3.1 Correspondent Management

The client applications executing TDS transactions via TCP/IP are TCP/IP correspondents. The following master commands deal with this type of correspondent:

- ALLOW_NEW_TDS_COR (allows connection of new TCP/IP correspondents).
- CANCEL_TDS_COR (cancels one or all TCP/IP correspondents).
- DISPLAY_TDS (displays information about the current number of active TCP/IP sessions).
- LIST_TDS_COR (lists information about the correspondent state and the number of running transactions and TPRs).
- PREVENT_NEW_TDS_COR (prevents connection of new TCP/IP correspondents).

NOTE:
A TCP/IP correspondent cannot be a master operator.

3.3.1.1 ALLOW_NEW_TDS_COR

Purpose

Cancels a previous [M] PREVENT_NEW_TDS_COR command and reconnects all passive TM, XCP1, XCP2 and TCP/IP correspondents. According to the parameters specified, this command:

- allows new terminal or TCP/IP correspondents to log on to a TDS application,
- establishes new XCP1 and XCP2 session pools,
- increases the number of allocated sessions for pools already opened.

Syntax

```
[ M ] { ALLOW_NEW_TDS_COR | ALNTC }
        [ { TMC | TM_COR } = bool }       ]
        [ { X1C | XCP1_COR } = bool }     ]
        [ { X2C | XCP2_COR } = bool }     ]
        [ TDS = { name4 | #WTDS }         ]
        [ { TCPIPC | TCPIP_COR } = bool } ]
```

Parameters

TCPIP_COR	TCP/IP correspondents.

See *TDS Administrator's Guide* for other parameters.

Constraints

Specifying no parameter is equivalent to specifying all correspondents regardless of their type.

Output

See *TDS Administrator's Guide.*

Example

```
[M]   ALNTC TCPIPC;
```

Allows all TCP/IP correspondents to log on to the TDS.

3.3.1.2 CANCEL_TDS_COR

Purpose

Forces the specified correspondent(s) to log off even if the user is frozen.

Syntax

```
[ M ]  { CANCEL_TDS_COR | CTC }
        { COR | USER } = star12
        [ STRONG = { 0 | bool }    ]
        [ FROZEN = { 0 | bool }    ]
        [ TDS = { name4 | #WTDS } ]
```

Parameters

STRONG	Forces the correspondent to log off immediately or allows any active transactions to complete. The effect of STRONG=1 depends on the type of correspondent: For TCP/IP correspondents, if a transaction is in progress, the executing TPR aborts, the ON-ABORT-TPR is executed, and the transaction terminates. The remote client application is aware of the disconnection via the returned status obtained on the next API verb.

See *TDS Administrator's Guide* for other parameters.

Constraints

None.

Output

See *TDS Administrator's Guide*.

Example

```
[M] CTC COR=LAUTIER TDS=PL ;
```

3.3.1.3 DISPLAY_TDS

Purpose

Displays information about the current TDS session according to the parameters specified. Additional information is displayed when SIMUL and/or STATUS are required.

Syntax

```
[ M ] { DISPLAY_TDS | DTDS }
        [ { STATUS | STAT } = bool          ]
        [ SIMUL = { 0 | bool }              ]
        [ SMLIB = { 0 | bool }              ]
        [ SWAP = { 0 | bool }               ]
        [ TDS = { name4 | #WTDS }           ]
        [ { RPC_STAT | RPC_STATUS } = bool ]
```

Parameters

See *TDS Administrator's Guide*.

Constraints

See *TDS Administrator's Guide*.

Output

The current number of TCP/IP sessions is displayed as part of the general statistics, if the TDS is TCP/IP.

See *TDS Administrator's Guide* for a complete description.

Example

Display the General statistics of an application named PL:

```
S: DTDS STATUS
              ----------------------------------------
              --   TDS = PL 11:45:51 NOV 14, 1997   --
              -----    GENERAL TDS STATISTICS    -----
              ----------------------------------------
      INIT.SIMU.COUNT   = 5    CUR.SIMU.COUNT    = 4
      ACC.SESS.ALLOC    = 0    ACC.SESS.REJEC    = 0
      USED TX COUNT     = 16   TX ABORT. COUNT   = 0
      USED TPR COUNT    = 27   TPR ABORT COUNT   = 0
      COMMIT COUNT      = 16   DIALOG COUNT      = 10
      TPR ELAPSED TIME  = 3    TPR CPU TIME      = 0
      DEADLOCK COUNT    = 0    NON CONCUR WAIT   = 0
      TABOV ABT COUNT   = 0    WDNAV ABT COUNT   = 0
      LGWAITABT COUNT   = 0    DIRTY READ ABORT  = 0
      BUFOVABT COUNT    = 0    SERIALIZATION     = 0
      MAX TM SES        = 10   CUR TM SES        = 0
      MAX XCP1 SES      = 5    CUR XCP1 SES      = 0
      MAX XCP2 SES      = 50   CUR XCP2 SES      = 0
      MAX VIRT SES      = 3    CUR VIRT SES      = 1
      PMOS COR COUNT    = 1    MAX IDLE TIME     = 2000
      POOL USED (KB)    = 160  POOL SIZE (KB)    = 500
      PSEUDO BUFFERS    = 0
      WAITING TPR MEAN  = 0    MAX CPU TIME      = 9000
      CUR TCP SES       = 2
      TDS: PL, DTDS COMMAND COMPLETED
```

3.3.1.4 LIST_TDS_COR

Purpose

Displays information about all correspondents or about specified correspondent(s).

Syntax

```
[ M ] { LIST_TDS_COR | LSTC }
       { COR | USER } = star12
        [ TYPE = { * | TM | XCP1 | XCP2 | DUMMY | TCPIP } ]
        [ { NG | NETGEN } = { 0 | bool }                  ]
        [ { LOG | LOGGED } = bool                         ]
        [ { DTLD | DETAILED } = { 0 | bool }              ]
        [ SORT = { 0 | bool }                             ]
        [ { PRTMB | PRINT_MEMBER } = name31               ]
        [ TDS = { name4 | #WTDS }                         ]
```

Parameters

See *TDS Administrator's Guide*.

Constraints

NETGEN=1 and LOGGED are mutually exclusive.

Output

The new *cortype* TCP/IP is processed by the command.

See *TDS Administrator's Guide* for a complete description.

Example

For a list of all types of correspondents known by TDS, when a TCP/IP correspondent is present:

```
S: LSTC * *
  -----------------------------------------
  --    TDS = PL 10:38:40 NOV 14, 1997   --
  -----    LIST OF CORRESPONDENTS     -----
  -----------------------------------------

  STATE: UNSPEC / TYPE: TM / LIST: SHORT / OPTION: DYNAMIC
  LEVENEZ-P H___PMS

  STATE: UNSPEC / TYPE: DUMMY / LIST: SHORT / OPTION: DYNAMIC
  DUMMY

  STATE: LOGGED / TYPE: TCPIP / LIST: SHORT / OPTION: DYNAMIC
  LAUTIER
  TDS: PL, LSTC COMMAND COMPLETED
```

For a detailed list of all TCP/IP correspondents:

```
S: LSTC * TCPIP DTLD
  ----------------------------------------
  --   TDS = PL 10:49:12 NOV 14, 1997   --
  -----    LIST OF CORRESPONDENTS    -----
  ----------------------------------------

  STATE: LOGGED /TYPE: TCPIP /LIST: DETAILED /OPTION: DYNAMIC
  CORRESPONDENT   ADDRESS   TX_COUNT   TPR_COUNT   STATUS   TX_NM
  LAUTIER                      3           8          P     ESSAI
  TDS: PL, LSTC COMMAND COMPLETED
```

3.3.1.5 PREVENT_NEW_TDS_COR

Purpose

Prevents new correspondents, including TCP/IP correspondents, from logging on to a TDS application. Prevents allocation of new XCP1 and XCP2 sessions by [M] MODIFY_COR_POOL, [M] OPEN_COR_POOL or corresponding CALLs described in *TDS COBOL Programmer's Guide*. For XCP2 correspondents, new conversations can be allocated using pools already opened.

When all current correspondents log off normally, TDS remains idle until [M] ALLOW_NEW_TDS_COR or [M] TERMINATE_TDS is issued.

Syntax

```
[ M ] { PREVENT_NEW_TDS_COR | PVNTC }
        [ { TMC | TM_COR } = bool }        ]
        [ { X1C | XCP1_COR } = bool }      ]
        [ { X2C | XCP2_COR } = bool }      ]
        [ TDS = { name4 | #WTDS }          ]
        [ { TCPIPC | TCPIP_COR } = bool } ]
```

Parameters

TCPIP_COR	TCP/IP correspondents.

See *TDS Administrator's Guide* for other parameters.

Constraints

None.

Output

See *TDS Administrator's Guide.*

Example

```
[M]   PVNTC TCPIPC TDS=PL;
```

Prevents the TCP/IP correspondents from logging onto the TDS application named PL.

3.3.2 Socket Management

Three new master commands are supplied to manage the closing, display and opening of the TDS socket:

- CLOSE_TDS_SOCKET
- DISPLAY_TDS_SOCKET
- OPEN_TDS_SOCKET

At TDS start, the TDS socket is automatically opened. TCP/IP correspondents can log on to the application as soon as the TDS is ready. The CLOSE_TDS_SOCKET and OPEN_TDS_SOCKET commands allow dynamic re-initialization of the communication link without stopping the TDS application. OPEN_TDS_SOCKET is intended for use after SOCKG 7 has been stopped or after a network failure.

3.3.2.1 CLOSE_TDS_SOCKET

Purpose

Closes the TDS socket. This command disconnects all TCP/IP correspondents and closes the socket. The CLOSE_TDS_SOCKET command is not remanent at TDS warm restart.

Syntax

```
[ M ] { CLOSE_TDS_SOCKET | CLTS}
        [ STRONG = { bool | 0 }    ]
        [ TDS = { name4 | #WTDS } ]
```

Parameters

STRONG	When STRONG = 1, forces disconnection of TCP/IP correspondents even if they are executing a transaction. In this case, the current TPR is aborted. When STRONG = 0 (the default value), closure of the TDS socket is deferred until the last TCP/IP correspondent ends its current transaction.

Constraints

None.

Output

After the command is successfully executed, the TX92 message appears on the master console:

```
TDS SOCKET CLOSED.
```

Example

```
S: CLTS STRONG=1 TDS=PL
   TX54 TDS: PL, CLTS COMMAND COMPLETED
-->   TX92 TDS SOCKET CLOSED
```

When the command cannot be performed, the message TX55 is issued:

TX55 TDS: PL, CLTS COMMAND NOT PERFORMED *rc*.

3.3.2.2 DISPLAY_TDS_SOCKET (DTDSS)

Purpose:

Display socket informations such as socket interface configuration through OPEN7 or GXTI, correspondent name and its peer address and port.

Syntax :

```
[ M]    { DTDSS                    }
        { DISPLAY_TDS_SOCKET }
        [{ STATUS | STAT }=bool ]
        [{ COR | USER } =star12]
        [ SORT=bool]
        [{ PRINT_MEMBER | PRTMB }=name31]
        [ TDS = { name4 | #WTDS } ]
```

Parameters :

STATUS alias STAT	Socket interface configuration (OPEN7 or GXTI). Default STAT=1.
COR alias USER	TCPIP TDS correspondents whose peer port and tcpip address is to be displayed. Star convention assumed. If blank (default value), no correspondent information is displayed. Only logged correspondents (not the FROZEN ones) are taken into account.
SORT	Correspondents are sorted in alphabetical order.

Default SORT=0 (random order).

PRINT_MEMBER :	Member in *tds-name*.DEBUG file for storing output to be printed. This member is always opened in output mode and is erased on cold restart of TDS. Default : Result are displayed on the screen.

Output :

Depending on the parameter specified, the message returned can be TX93,TX94, TX23,TX24,TX56,TX57.

See Appendix I.

Example :

DTDSS STATUS=1 COR=* SORT=0 ;

```
 ------------------------------------------
 -- TDS = JDB    10:29:49  MAR 31, 2004 --
 -----   CURRENT TCP/IP PARAMETERS      ---
 ------------------------------------------
SOCKET INTERFACE IS : OPEN7
CORRESPONDENT     PEER ADDRESS       PORT
 JDB2           129.182.197.67       1142
 JDB3           129.182.197.67       1143
 JDB4           129.182.197.67       1144
 JDB1           129.182.197.67       1141
TDS : JDB, DTDSS COMMAND COMPLETED
```

3.3.2.3 OPEN_TDS_SOCKET

Purpose

Opens the TDS socket. This command re-opens the TDS socket after a CLOSE_TDS_SOCKET command has been issued.

Syntax

```
[ M ] { OPEN_TDS_SOCKET | OTS }
        [ UPON = { OPEN7 | GXTI } ]
        [ TDS = { name4 | #WTDS } ]
```

Parameters

UPON	Specifies the option for the communication link: OPEN7 or GXTI. It must correspond to the option set in the TDS-TCP/IP PROTOCOL generation clause. The default value is GXTI.

Constraints

None.

Output

After the command is successfully executed, the message TX92 appears on the master console:

```
TDS SOCKET OPENED.
```

Example

```
S: OTS UPON=OPEN7 TDS=PL
  TX54 TDS: PL, OTS COMMAND COMPLETED
-->   TX92 TDS SOCKET OPENED
```

When the command cannot be performed, the message TX55 is issued:

```
TX55 TDS: PL, OTS COMMAND NOT PERFORMED rc.
```

4. Client Application Development

Client application development on the PC concerns:

- Configuration of the client machine; i.e., PC configuration,
- Calls to C-language XATMI functions invoked by the TDS-TCP/IP API in order to access one or more TDS applications from a PC application.

4.1 PC Environment Configuration

4.1.1 Description

Client PCs must be configured for each TDS application. Two files hold the relevant information:

- the address for remote host mapping is held in the **HOSTS** file located in the Windows installation directory(usually C:\WINNT\system32\drivers\etc),
- the service declaration is held in the **SERVICES** file located in the Windows installation directory(usually C:\WINNT\system32\drivers\etc).

4.1.2 Configuration Declaratives

The **HOSTS** file must be updated to include the IP addresses of the names of the hosts running the TDS applications. A host name is the symbolic name used in the client application to address the DPS 7000 system (corresponding to the *hostid* parameter of the *tpconnect* verb).

EXAMPLE:

129.182.50.50	bc0f	#host name of the DPS 7000 on which the TDS **tds1** runs

❑

The **SERVICES** file must be updated to include the required service names, the port number associated to each service, and the protocol used. The service name is the concatenation of the host name and the TDS name. The port number must be the same as the one declared on the GCOS 7 server.

EXAMPLE:

If a client application wants to connect to the TDS **tds1** located on the DPS 7000 referred to by the host name **bc0f**, the following line must appear in the SERVICES file:

```
bc0ftds1                10100/tcp
```

❑

NOTE:
The value of the port number must be greater than 1024.

4.2 TDS-TCP/IP Client API

A PC user connected to TDS via TCP/IP, accesses a TDS application in the same way as a terminal user connected through DSA. In particular, the turn is managed.

The TDS-TCP/IP client API provided for the development of client applications enables a conversational service. This API uses XATMI functions adapted to GCOS 7 requirements in terms of connection, transactional dialog with turn, abnormal events, and disconnection.

The LOGON transaction is activated each time a TCP/IP client successfully connects to TDS. In the TRANSACTION-STORAGE of the LOGON transaction, the TERMINAL-ID field is initialized by TDS to the specific value **H-TCPIP-CLI**.

Since GCOS 7 TS 9764 and DLL version 3.0.6, the *tpconnect* function supports a *termid* field with 13 characters in the *data* parameter. If the termid field is supplied with a value different from spaces, it is moved in the TERMINAL-ID field of the TRANSACTION-STORAGE of the LOGON transaction. If the termid field is filled with spaces, or null, or not used, or if the DLL version is before 3.0.6 the TERMINAL-ID field of the TRANSACTION-STORAGE of the LOGON transaction is initialized by TDS to the specific value **H-TCPIP-CLI.**

Transactions are started by the *tpsend* function. *tpsend* and *tprecv* are coded in the client application to correspond to the RECEIVE and SEND verbs coded in the TPRs and the messages sent and received by the TDS monitor.

Before GCOS 7 TS 9764, if there is no SEND verb at the end of a transaction, a final *tprecv* function is issued to get back the turn. A null length message is then returned. Since GCOS 7 TS 9764, if there is no SEND verb at the end of a transaction, a final *tprecv* function receives the READY message from TDS (or the related service message defined at TDS generation time).

If a transaction aborts, the message sent by the ON-ABORT-TPR is retrieved by the next *tprecv* verb.

If *tpsend* parameters are not correct from the TDS-TCP/IP dialog's point of view, an error status is returned on the next *tprecv* verb.

4.2.1 Composition of the API

The TDS-TCP/IP client API is composed of the following subset of C-Language XATMI functions:

- four functions for conversational services: *tpconnect*, *tpdiscon*, *tprecv* and *tpsend*
- two functions for typed buffer management: *tpalloc* and *tpfree*
- two functions for error return: *tperrno* and *tperrdtl,* these functions being extensions of XATMI.

The client application is always the initiator of the TDS connection, but the disconnection can be made by either the client (PC) or the server (DPS 7000).

4.2.2 Compiling the Client Program

For a client coded in C, an atmi.h file is delivered containing the description of the connection structures: service_id and subtype, the extern function prototypes, the event types, and error codes.

The atmi.h file must be put in the client working directory.

In the source code using API functions, the following directive may be provided:

```
#include <atmi.h>
```

4.3 C-Language XATMI Functions

This section contains the C-language description of the XATMI functions for the TDS-TCP/IP client API. It is compliant with the Windows system API.

4.3.1 tpalloc

This function allocates a typed buffer for use for connection and dialog with TDS. The only *type* permitted for TDS-TCP/IP is X_C_TYPE. The buffer allocated is at least as large as *size*.

Syntax

```
char * tpalloc(char *type, char *subtype, long size)
```

Input Parameters

type	**"X_C_TYPE"**. This parameter is char[16] null or blank terminated.
subtype	Name of the subtype associated to the type. The subtype is application dependent. It indicates the meaning or interpretation of the data in the application. This parameter is char[16], null or blank terminated.
size	Allocated buffer size in bytes.

Constraints

The buffers must be allocated by *tpalloc* and freed by *tpfree*. Other functions for memory management such as *malloc()* must not be used.

Return Value

Upon successful completion, *tpalloc* returns a pointer to a buffer of the appropriate type aligned on a long word. Otherwise, it returns NULL and *tperrno* must be called to get the error condition.

The possible values for *tperrno* on *tpalloc* are the following:

- invalid arguments given [TPEINVAL]
- system error [TPESYSTEM], [TPEOS]

EXAMPLE

See *Client Program Examples* later in this chapter.

❑

4.3.2 tpconnect

This function makes the connection to the TDS application which is identified by the couple (host name, TDS name) associated with a unique port number.

Since the version 3.0.6 of the DLL and using a TDS with GCOS 7 TS 9764, the *tpconnect* function supports and added field named *termid* in the input **data** parameter. If this field is filled with a value different from spaces, it is sent to TDS and becomes the terminal identification in the TERMINAL-ID field of the TRANSACTION-STORAGE of the LOGON transaction.

Syntax

```
int tpconnect (char *svc,char *data, long len, long flags)
```

Input Parameters

svc

Name of the service which is the concatenation of the host name and the TDS name described in the SERVICES file.

```
struct service_id {
                   char hostid[16];
                   char tdsname[5]} svc;
```

hostid
the DPS 7000 host name (a string, null terminated or blank terminated).

tdsname
Name of the TDS application (a string, null terminated or blank terminated).

data

Buffer allocated by *tpalloc* function with the type X_C_TYPE. Before the 3.0.6 version of the DLL it contains the following structure:

```
struct subtype {
                char name[13];
                char project[13];
                char billing[13];
                char password[13];} data;
```

Since the 3.0.6 version of the DLL it may contain the following structure:

```
 struct subtype {
                char name[13];
                char project[13];
                char billing[13];
                char password[13];
                char termid[13];}   data;
```

Since the DLL version 3.0.8, the data structure may be the following :

```
struct subtype {
                    char name[13];
                    char project[13];
                    char billing[13];
                    char password[13];
                    char termid[13];
                    char dataconvert [2];}
             data;
```

Since the DLL version 4.2.0, the data structure may be the following :

```
struct subtype {
                    char name[13];
                    char project[13];
                    char billing[13];
                    char password[13];
                    char termid[13];
                    char dataconvert [2]
                    char relogonforce[2];
                    } data;
```

This structure contains the identification of the GCOS 7 user, known in the GCOS 7 catalog, which will allow the foreign client to execute transactions.

All structure variables are strings, null or blank terminated.

name
Name of the TDS user. There is no default value.

project
Name of the project. When connected, the client application can launch all the transactions permitted for this project according to PROJECT/TDS code of

the GCOS 7 catalog. If this parameter is filled with blanks, the GCOS 7 default project is taken.

billing
The billing is checked in the GCOS 7 catalog. If set to blanks, the GCOS 7 default billing for this user's project will be taken.

password
GCOS 7 password of the user. There is no default value.

termid
Terminal Identifier. The termid field, if any, enables the TDS to transmit this value into the TERMINAL-ID field of the TRANSACTION-STORAGE of the LOGON transaction.

dataconvert
Data conversion indicator. If set to blank or not specified, ASCII / EBCDIC conversion will be performed for the data buffers exchanged on tprecv and tpsend verbs.
If dataconvert = "Y" data buffers are converted.
If dataconvert = "N" data buffers are not converted. This latter value must be used only from GCOS 7 TS 9866.

relogonforce
Force logon indicator. If set to blank or "N" or not specified, tpconnect fails with error 90 if the user is already connected.
If relogonforce = "Y", if the transaction state is commited or it is an end of transaction, the user already connected is disconnected, then it is connected with the same context; else tpconnect fails with error 90.
This latter value must be used only from GCOS 7 TS 9920.

len
Before the version 3.0.6 of the DLL, the len value must be 0. Since the version 3.0.6 of the DLL, if the termid field is specified the data structure length must be 65. If the termid field is not specified the len value must be either 0 or 52.
Since the DLL version 3.0.8, if the dataconvert field of the data structure is specified the len value must be 67. Since the DLL version 4.2.1, if the relogonforce field

of the data structure is specified the len value must be 69.

flags	Not used.

Constraints

The client application must perform a *tprecv* after the *tpconnect* function in order to receive the result message of the connection and get back the turn.

The *tpconnect* function can be protected by a timer to avoid blocking situations that may occur in case of an incorrect configuration, a non-response from the network or TDS. The default value of the time-out is 0. It means no time-out mechanism applies. This value can be modified by changing the TIMEOUT key in the Atmi.ini file as follows:

TIMEOUT = *MaxDelay*

MaxDelay is expressed in milliseconds. For example, TIMEOUT = 300000 fixes a time-out to 5 minutes.

When a time-out occurs, the socket is closed causing the disconnection of the client.

Return Value

The output is a communication descriptor *cd*, which is the input parameter for the *tprecv*, *tpsend* and *tpdiscon* verbs. The *cd* descriptor is used to refer to the connection in subsequent calls.

If the value is -1, the connection has failed and *tperrno* must be called to get the error condition.

tperrno on *tpconnect*:

- invalid arguments given [TPEINVAL]
- maximum number of connections reached [TPELIMIT]
- protocol error [TPEPROTO]

EXAMPLE

See *Client Program Examples* later in this chapter.

❑

4.3.3 tpdiscon

This function abnormally terminates the connection specified by *cd*, generates an abnormal disconnection to the server from the client part and cleans the client context. It can be activated at any time by the client application.

To disconnect normally, a client must issue a *tpsend* with the “BYE” transaction in order to notify TDS to execute the LOGOUT TPR. In this case, *tpdiscon* is used to clean the client context. It must be also called after error upon *tprecv* and *tpsend*.

Syntax

```
int tpdiscon (int cd)
```

Input Parameter

cd	descriptor returned by *tpconnect* function.

Constraints

None.

Return Value

If the value is 0, the function is successful.

If the value is -1, the function has failed and *tperrno* must be called to get the error condition.

tperrno on *tpdiscon*:

- invalid arguments given [TPEINVAL]
- protocol error [TPEPROTO]

EXAMPLE

See *Client Program Examples* later in this chapter.

❑

4.3.4 tpfree

This function frees a typed buffer allocated by the *tpalloc* function. This function does not return a value to the client program. Therefore, it is declared as a void.

Syntax

```
void tpfree (char *ptr)
```

Input Parameter

ptr	Pointer to a buffer previously obtained by the *tpalloc* function. This pointer can be NULL in which case no action occurs. Undefined results occur if *ptr* does not point to a typed buffer.

EXAMPLE

See *Client Program Examples* later in this chapter.

❑

4.3.5 tprecv

This function allows a client application to receive a message from the server.

Syntax

```
int tprecv (int cd, char **data, long *len, long flags,
           long *revent)
```

Input Parameters

cd	Descriptor obtained through *tpconnect* function. Specifies on which open connection data is to be received.
flags	Not used.

Output Parameters

data	Buffer allocated through the *tpalloc* function with the type X_C_TYPE and which contains the received message.
len	Contains the length of the received message. If *len* is 0, no data is received, and neither **data* nor the buffer it points to have been modified.
revent	revent is significant if tpermo is set to TPEEVENT. Valid events for *tprecv()* are as follows:

TPEV_SENDONLY: 0x0020
The recipient of this event has the turn and is allowed to send data but cannot receive any data until it relinquishes control.

TPEV_SVCERR: 0x0002
This event indicates that the TDS server has encountered an error while processing the client request. When it is the first *tprecv()* after *tpconnect()* it means that the connection was accepted by TCP/IP but was rejected afterwards by TDS. Upon an SVCERR event, no data is returned by *tprecv()*. The connection been terminated by TDS, so *tpdiscon()* must be called to clean the client context.

Constraints

The client application must not have the turn when *tprecv()* is issued.

The *tprecv* function can be protected by a timer to avoid blocking situations that may occur in case of an incorrect configuration, a non-response from the network or TDS. The default value of the time-out is 0. It means no time-out mechanism applies.

For Windows, this value can be modified by changing the TIMEOUT key in the Atmi.ini file as follows:

TIMEOUT = *MaxDelay*

MaxDelay is expressed in milliseconds. For example, TIMEOUT = 300000 fixes a time-out to 5 minutes.

For AIX or Linux, this value can be modified by positioning the environment variable : ATMI_TIMEOUT with a delay expressed in milliseconds.

When a time-out occurs, the socket is closed causing the disconnection of the client.

Return Value

If the value is 0, the function is successful.

If the value is -1, *tperrno* must be called to get the error condition. If tperrno is set to TPEEVENT and revent to TPEV_SENDONLY, the received message has been returned in the buffer and the client application now has the turn. In the other error cases, the *tprecv* function has failed and no data is available.

tperrno on *tprecv*:

- invalid arguments given [TPEINVAL]
- protocol error [TPEPROTO]
- system error [TPESYSTEM], [TPEOS]
- an event occurred and its type is available in revent [TPEEVENT]

EXAMPLE

See *Client Program Examples* later in this chapter.

❑

4.3.6 tpsend

This function allows the client application to send a message to the TDS. The turn is sent with each *tpsend()*.

Syntax

```
int tpsend ( int cd, char *data, long len, long flags, long
          *revent)
```

Input Parameters

cd	Descriptor obtained through *tpconnect* function. Specifies on which open connection to send data.
flags	Not used.
data	Buffer allocated through *tpalloc* function with the type X_C_TYPE containing the message to send.
len	Contains the length of the buffer to send. Its value must be greater than 0.

Output Parameters

revent	Not used.

Constraints

The client application must have the turn when *tpsend()* is issued.

Return Value

If the value is 0, the function is successful.

If the value is -1, the function has failed and *tperrno* must be called to get the error condition.

tperrno on *tpsend*:

- invalid arguments given [TPEINVAL]
- protocol error [TPEPROTO]
- system error [TPESYSTEM], [TPEOS]

EXAMPLE

See *Client Program Examples* later in this chapter.

❑

4.3.7 tperrno

This function retrieves the tperrno value set by the previous API call. It is an extension of the XATMI primitives.

Syntax

```
int tperrno ( void )
```

Input Parameters

None.

Output Parameters

None.

Constraints

None.

Return Value

If the value is 0, it means that the previous function has encountered no error.

If the value is not 0, it is the tperrno value.

The tperrno values are as follows:

`TPEINVAL`	4	Invalid arguments given
`TPELIMIT`	5	Maximum number of connections reached
`TPEOS`	7	System error
`TPEPROTO`	9	Protocol error
`TPESYSTEM`	12	System error
`TPEEVENT`	22	Event occurred and its type is available in the revent parameter

EXAMPLE

See *Client Program Examples* later in this chapter.

❑

4.3.8 tperrdtl

This function retrieves the tperrno value and the detailed error value set by the previous API call. It is an extension of the tperrno function. The detailed error corresponds to the client error message number.

Syntax

```
int tperrdtl ( int *suberr )
```

Input Parameters

None.

Output Parameters

suberr	Detailed error code set by the last API call.

Constraints

tperrdtl and *tperrno* are mutually exclusive. These 2 verbs cannot be used in sequence to test the result of the same API call.

Return Value

If the value is 0, it means that the previous function has encountered no error.

If the value is not 0, it is the tperrno value. See the tperrno values above.

4.4 Client States

The following table represents the client state with respect to XATMI routines during a conversational service with TDS.

Each line contains an XATMI routine, each column a client state.

An entry under a particular state in the table shows which XATMI routine can be called in that state, and lists the resulting state. A blank indicates that it is an error to call the XATMI routine in that state.

state **routine**	**init**	**send**	**receive**
tpconnect	receive		
tprecv			send (*1)
			receive (*2)
tpsend		receive	
tpdiscon		init	

(*1) resulting state if the client has received the turn
(*2) resulting state if the client has not received the turn

Error cases:

Upon an error returned by an XATMI routine, the tpdiscon routine must be called.

4.5 Client Program Examples

Here is an example of a client PC-TDS dialog:

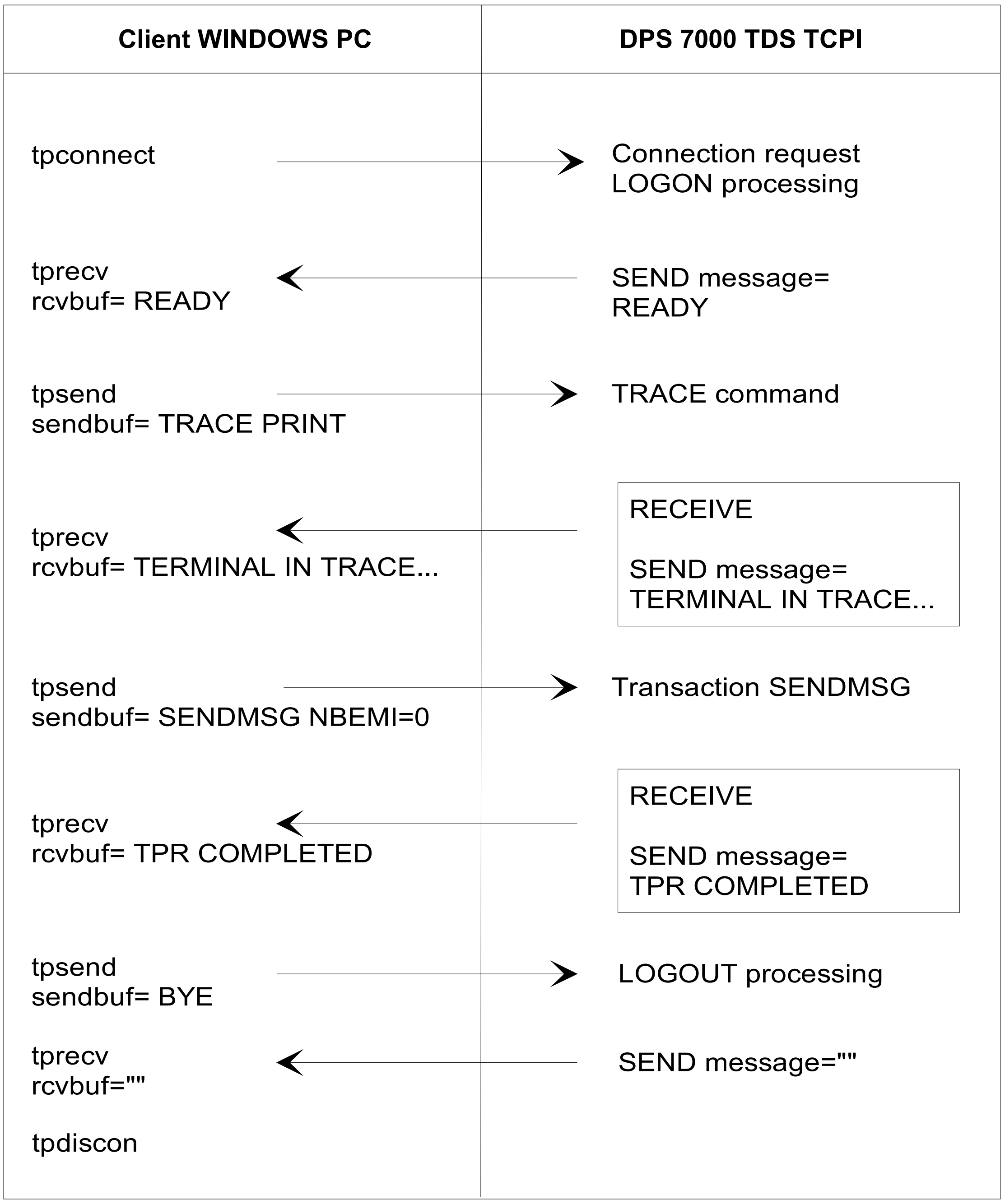

Figure 4-1. Client PC-TDS Dialog

4.5.1 Application Main Program

```
The following example is a program written in VB (Visual Basic).

==> forms description
*********************
Begin VB.Form test_vbatmi32
   Caption         =   "Form1"
   ClientHeight    =   1935
   ClientLeft      =   60
   ClientTop       =   345
   ClientWidth     =   2685
   LinkTopic       =   "Form1"
   ScaleHeight     =   1935
   ScaleWidth      =   2685
   StartUpPosition =   3  'Windows Default

   Begin VB.CommandButton CmdFin
      Caption         =   "FIN"
      Height          =   495
      Left            =   360
      TabIndex        =   1
      Top             =   960
      Width           =   2055
   End

   Begin VB.CommandButton CmdConnect
      Caption         =   "CONNECT"
      Height          =   495
      Left            =   360
      TabIndex        =   0
      Top             =   240
      Width           =   2055
   End

End
***********
* CODE *
***********
    Option Explicit
    Dim ConnectIdent As Long
    Dim ReturnStatus As Long
    Dim data As String
```

```
Private Sub CmdConnect_Click()

    Dim ConnectData As TP_CONNECTION_DATA
    Dim ConnectService As TP_CONNECTION_SERVICE
    Dim ConnectIdent As Long
    Dim ReturnStatus As Long
    Dim data As String
    Dim turn As Long

    ConnectData.Name = "LAUTIER"
    ConnectData.Project = "TCP"
    ConnectData.Billing = "TCP-V7"
    ConnectData.Password = "JA"
    ConnectService.HostId = "bcde-xti"
    ConnectService.TdsName = "tcpi"

==> connection to the tds
******************************
    ATMI_ConnectTds ConnectService, ConnectData, ConnectIdent, ReturnStatus
    If ReturnStatus <> 0 Then
        MsgBox "connect tds code : " & ReturnStatus, vbOKOnly, "*** ERROR ***"
        Exit Sub
    End If

    receivedata
    If ReturnStatus <> 0 Then
        Exit Sub
    End If

==> execution of the TRACE command
**********************************************
    data = "TRACE PRINT"
    ATMI_SendData ConnectIdent, data, ReturnStatus
    If ReturnStatus <> 0 Then
        MsgBox "send data code : " & ReturnStatus, vbOKOnly, "*** ERROR ***"
        Exit Sub
    End If

    receivedata
    If ReturnStatus <> 0 Then
        Exit Sub
    End If
```

```
==> execution of the SENDMSG transaction
****************************************************
    data = "SENDMSG NBEMI=0"
    ATMI_SendData ConnectIdent, data, ReturnStatus
    If ReturnStatus <> 0 Then
        MsgBox "send data code : " & ReturnStatus, vbOKOnly, "*** ERROR ***"
        Exit Sub
    End If

    receivedata
    If ReturnStatus <> 0 Then
        Exit Sub
    End If

==> normal disconnection from TDS
******************************************
    data = "BYE"
    ATMI_SendData ConnectIdent, data, ReturnStatus
    If ReturnStatus <> 0 Then
        MsgBox "send data code : " & ReturnStatus, vbOKOnly, "*** ERROR ***"
        Exit Sub
    End If

    ATMI_ReceiveData ConnectIdent, data, turn, ReturnStatus
    If ReturnStatus = 0 Then
        MsgBox "erreur : receive ok after BYE", vbOKOnly, "*** ERROR ***"
    Else
        MsgBox "disconnection ok : status for BYE : " & ReturnStatus, vbOKOnly,
                                                                "*** OK ***"
    End If

    ATMI_DisconnectTds ConnectIdent, ReturnStatus
    If ReturnStatus <> 0 Then
        MsgBox "disconnect tds code : " & ReturnStatus, vbOKOnly,
                                                             "*** ERROR ***"
        Exit Sub
    End If
==> end of main program
******************************

    MsgBox "==> test OK", vbOKOnly, "BYE BYE"

End Sub

Private Sub CmdFin_Click()
    End
End Sub
```

```
==> sub program for receiving data sent by the TDS transaction
**************************************************************************
Private Sub receivedata()
    Dim turn As Long
    Dim title As String
    Dim ReturnStatusDisc As Long
    turn = 0
    While turn = 0
        ATMI_ReceiveData ConnectIdent, data, turn, ReturnStatus
        If ReturnStatus <> 0 Then
            MsgBox "receive data code : " & ReturnStatus, vbOKOnly,
                                                            "*** ERROR ***"
            ATMI_DisconnectTds ConnectIdent, ReturnStatusDisc
            If ReturnStatusDisc <> 0 Then
                MsgBox "disconnect tds code : " & ReturnStatusDisc, vbOKOnly,
                                                                 "*** ERROR ***"
            End If
            Exit Sub
        End If
        If turn = 0 Then
            MsgBox data, vbOKOnly, "receive noturn length=" & Len(data)
        Else
            MsgBox data, vbOKOnly, "receive with turn length=" & Len(data)
        End If
    Wend
End Sub
```

4.5.2 TDS API Interface

```
Attribute VB_Name = "Module1"
Rem ======= TDS TCP/IP interface defintion in Visual Basic

Rem types definition
Rem ******************

Option Explicit

Type TP_CONNECTION_SERVICE
    HostId As String * 16
    TdsName As String * 5
End Type


Type TP_CONNECTION_DATA
    Name As String * 13
    Project As String * 13
    Billing As String * 13
    Password As String * 13
    Termid As String * 13
    Dataconv As String * 2
    RelogonForce As String * 2
End Type

Rem Constants
Rem ************

Const ATMI_BUFFERTYPE As String * 8 = "X_C_TYPE"
Public Const ATMI_CONNECTDATASIZE As Long = 52
Public Const ATMI_MAX_RECEIVED_DATA_LENGTH = 32767
Public Const ATMI_TPEEVENT As Long = 22
Public Const ATMI_TPEV_SENDONLY As Long = 32

Rem ATMI32.DLL entry points declarations
Rem ********************************************

Declare Function tpalloc Lib "Atmi32.dll" _
        (ByVal sType As String, ByVal sSubType As String, ByVal lsize As Long)
                                                                        As Long

Declare Sub tpfree Lib "Atmi32.dll" _
        (ByVal pBuffer As Long)

Declare Function tpconnect Lib "Atmi32.dll" _
        (ByRef sSvc As TP_CONNECTION_SERVICE, ByVal pBuffer As Long, _
         ByVal lsize As Long, ByVal lFlags As Long) As Long
```

```
Declare Function tpdiscon Lib "Atmi32.dll" _
        (ByVal lCd As Long) As Long

Declare Function tprecv Lib "Atmi32.dll" _
        (ByVal lCd As Long, ByRef pBuffer As Long, ByRef lsize As Long, _
         ByVal lFlags As Long, ByRef lEvent As Long) As Long

Declare Function tpsend Lib "Atmi32.dll" _
        (ByVal lCd As Long, ByVal pBuffer As Long, ByVal lLen As Long, _
         ByVal lFlags As Long, ByRef lRevent As Long) As Long

Declare Function tperrno Lib "Atmi32.dll" () As Long

Rem Other Win 32 API functions used as Services to transfert data
Rem ***********************************************************************

Declare Sub CopyConnectDataToBuffer Lib "KERNEL32" Alias "RtlMoveMemory" _
        (ByVal pDest As Long, ByRef HpSource As TP_CONNECTION_DATA, ByVal
                                                             cbCopy As Long)

Declare Sub CopyStringToBuffer Lib "KERNEL32" Alias "RtlMoveMemory" _
        (ByVal pDest As Long, ByVal HpSource As String, ByVal cbCopy As Long)

Declare Sub CopyBufferToString Lib "KERNEL32" Alias "RtlMoveMemory" _
        (ByVal sString As String, ByVal pSource As Long, ByVal cbCopy As Long)

Rem function interface to call XATMI DLL entry points
Rem *********************************************************

Public Sub ATMI_ConnectTds(TdsIdent As TP_CONNECTION_SERVICE, UserIdent As
TP_CONNECTION_DATA, _
                           ConnectIdent As Long, ReturnStatus As Long)
    Dim ConnectFlags As Long
    Dim ConnectBuffer As Long
    ConnectBuffer = tpalloc(ATMI_BUFFERTYPE, "connect",
    ATMI_CONNECTDATASIZE)
    If ConnectBuffer = 0 Then
        ReturnStatus = tperrno()
    Else
        CopyConnectDataToBuffer ConnectBuffer, UserIdent, ATMI_CONNECTDATASIZE
        ConnectIdent = tpconnect(TdsIdent, ConnectBuffer, ATMI_CONNECTDATASIZE,
        ConnectFlags)
        If ConnectIdent = -1 Then
            ReturnStatus = tperrno()
        Else
            ReturnStatus = 0
        End If
        tpfree ConnectBuffer
    End If
End Sub
```

```
Public Sub ATMI_DisconnectTds(ConnectIdent As Long, ReturnStatus As Long)
    ReturnStatus = tpdiscon(ConnectIdent)
    If ReturnStatus <> 0 Then
        ReturnStatus = tperrno()
    End If
End Sub

Public Sub ATMI_SendData(ConnectIdent As Long, DataToSend As String,
ReturnStatus As Long)
    Dim SendFlags As Long
    Dim SendEvent As Long
    Dim DataLength As Long
    Dim SendBuffer As Long
    DataLength = Len(DataToSend)
    SendBuffer = tpalloc(ATMI_BUFFERTYPE, "send", DataLength)
    If SendBuffer = 0 Then
        ReturnStatus = tperrno()
    Else
        CopyStringToBuffer SendBuffer, DataToSend, DataLength
        SendFlags = 0
        ReturnStatus = tpsend(ConnectIdent, SendBuffer, DataLength, SendFlags,
        SendEvent)
        If ReturnStatus <> 0 Then
            ReturnStatus = tperrno()
        End If
        tpfree SendBuffer
    End If
End Sub
```

```
Public Sub ATMI_ReceiveData(ConnectIdent As Long, ReceivedData As String, _
                            TurnIndicator As Long, ReturnStatus As Long)
    Dim ReceiveBuffer As Long
    Dim DataLength As Long
    Dim receiveFlags As Long
    Dim ReceiveEvent As Long
    DataLength = ATMI_MAX_RECEIVED_DATA_LENGTH
    TurnIndicator = 0
    ReceiveBuffer = tpalloc(ATMI_BUFFERTYPE, "receive", DataLength)
    If ReceiveBuffer = 0 Then
        ReturnStatus = tperrno()
    Else
        ReturnStatus = tprecv(ConnectIdent, ReceiveBuffer, DataLength,
                                    receiveFlags, ReceiveEvent)
        If ReturnStatus = -1 Then
            ReturnStatus = tperrno()
            If ReturnStatus = ATMI_TPEEVENT Then
                If ReceiveEvent = ATMI_TPEV_SENDONLY Then
                    ReturnStatus = 0
                    TurnIndicator = 1
                Else
                    ReturnStatus = ReceiveEvent
                End If
            End If
        End If
        If ReturnStatus = 0 Then
            If DataLength <> 0 Then
                ReceivedData = String(DataLength, " ")
                CopyBufferToString ReceivedData, ReceiveBuffer, DataLength
            Else
                ReceivedData = ""
            End If
        End If
        tpfree ReceiveBuffer
    End If
End Sub
```

5. Error Handling

5.1 Sequence Integrity

A TDS transaction is written to receive and send messages in a pre-defined order. Consistency in receive-send sequences is necessary.

The TCP/IP client must know the semantics of each message expected and sent by the transaction. These sequences can be disrupted by the following situations:

- during a socket closure in abnormal cases,
- a commitment unit rollback,
- a TDS re-initialization,
- a TDS failure.

Note: When a TDS RESTART (COLD or WARM) is done or when a TDS REINIT occurs, the previous sessions of TCP/IP correspondents are lost.
Thus, at reconnection of the TCP/IP client, the session is in first logon processing; Its eventual previous interrupted transaction is not restarted.

5.1.1 Socket Closure in Abnormal Cases

Socket closure can occur in abnormal situations, due to a master command or disconnection. If the transaction has not been completed when the socket closure occurs, the TCP/IP correspondent may connect again to the TDS, and the transaction will be re-started. Two situations are possible:

- If the failure occurs at the commitment point when the correspondent reconnects to TDS, NEXT-TPR is started and the receive-send sequence is not disrupted.
- If the failure occurs during the transaction, the commitment unit is restarted and the receive-send sequence is disrupted, as in the commitment unit rollback case below.

5.1.2 Commitment Unit Rollback

When a commitment unit is restarted, all the commitment unit messages are sent again. The TDS-TCP/IP client may receive the same data twice.

5.1.3 TDS Re-initialization

When sockets close, all TDS-socket operations are interrupted, causing the disconnection of all TCP/IP sessions. A client application must reconnect to the TDS, and the TDS is re-initialized. The receive-send sequence is disrupted.

5.1.4 TDS Failure

If TDS fails, sockets close, and all TDS-socket operations are interrupted.

5.1.5 TDS-HA Takeover

If an HA takeover occurs, all TDS-socket operations are interrupted. Moreover, the TCP/IP correspondents are not automatically reconnected to the backup TDS.

5.1.6 GCOS 7 Warm Restart after Crash

If the TDS step is repeatable it may happen after a GCOS 7 crash that TDS restarts before the socket communication link is re-established. It this case, the TCP/IP communication cannot be initialized by TDS and TCP/IP clients cannot be reconnected. So it is recommended to issue an OPEN_TDS_SOCKET master command as soon as the TDS becomes ready after step repeat.

5.2 Transaction Programming

If a TCP/IP client is disconnected while executing a transaction, this transaction will be automatically restarted at its last commitment point when reconnecting to TDS. The client may receive the same messages twice.

The recovery after receive-send sequence failures is complex, if it assumes that the client application must forecast all the possible messages that can be received, and must be sent due to the failure cases described above.

So, it is recommended to code a CANCELCTX procedure in the LOGON transaction in the case of a TCP/IP correspondent in order to prevent current transactions restarting at reconnection time.

Note: After a TDS Reinit or a TDS restart (COLD or WARM, after GCOS7 crash or not), the previous session of TCP/IP correspondents are lost.
At the reconnection of the TCP/IP client, the session is in first logon processing and its eventual previous interrupted transaction is not restarted.

5.3 Client Error Messages

Client error messages are logged in the file Atmitds.log. This file is dynamically created by the DLL. Messages are recorded in the log each time an error occurs during the execution of an XATMI function. It is used for trace purposes and to find details of tperrno values.

The error messages are as follows:

TDS-TCP/IP Related Errors

Error = 1	**Internal error in ATMI DLL** DLL system error.
Error = 2	**Connect error: ID out of range** Invalid cd input parameter.
Error = 3	**Connect error: Wrong session ID** Invalid cd input parameter.
Error = 4	**Invalid argument given** Parameter not addressable.
Error = 5	**Maximum number of connections reached** – Attempt to perform a new connection while 500 connections are already active in the client application. – No new connection is allowed due to a PREVENT_NEW_TDS_COR command issued on the TDS application.
Error = 9	**Protocol error** – Function called in an improper context. – User cannot be connected. – Buffer too small to receive data.
Error = 10	**Invalid connection buffer** DLL system error
Error = 12	**System error** DLL system error
Error = 20	**Communication error during connection** Communication link failure.
Error = 22	**Event occurred** An event occurred and its type is available in the revent parameter of *tprecv.*

Error = 30	**Internal error TDS** Attempt to connect a user already known by TDS as a non-TCP/IP correspondent.
Error = 31	**Invalid length** Buffer length too small to receive data.
Error = 32	**TDS socket being closed** Attempt to connect a user while TDS socket being closed.
Error = 33	**Invalid Client version** Incompatibility between client version and GCOS 7 Technical Status.
Error = 60	**Catalog GCOS error** Connection parameters: Name, Project, Billing do not match GCOS 7 catalog information.
Error = 70	**Invalid password** Password connection parameter is invalid.
Error = 80	**Internal TDS error (table allocation)** TDS system error during connection.
Error = 81	**Connection error : (Table or swap allocation)** TDS system error during connection.
Error = 90	**User already connected** Attempt to connect a user already connected.
Error = 95	**Connection error : (Invalid multiple users)** - The length of the correspondent’s name exceed nine characters. - There is already 999 connected multiple users using this name.
Error =96	**Connection error : (Wrong multiple users name)**
Error = 100	**Incomplete disconnection** Attempt to connect a user already being disconnected.
Error = 110	**Connection error (invalid state)** TDS system error during connection.
Error = 120	**Invalid reconnection** TDS system error during connection.

Error = 130	**Reconnection (invalid state)** TDS system error during connection.
Error = 140	**Master not allowed** Attempt to connect the TDS master operator as a TCP/IP correspondent.
Error = 150	**TDS not ready, retry later** Attempt to connect a user while TDS is not at the ready state.
Error = 200	**Send error (invalid header: session)** TDS system error while sending data.
Error = 210	**Send error (invalid header)** TDS system error while sending data.
Error = 220	**Send error (invalid header: sequence)** TDS system error while sending data.
Error = 230	**Send error (invalid datatype or bad turn)** TDS system error while sending data.
Error = 235	**Send error (session being disconnected)** Attempt to send data while the user session being disconnected.
Error = 240	**Send error (internal error)** TDS system error while sending data.
Error = 245	**Send error (no data)** Attempt to send a null length buffer while the session is idle.
Error = 250	**Send error (flow state)** TDS system error while sending data.
Error = 260	**Send error (timeout)** Time-out occurred before TDS receives all the sent data.
Error = 280	**User canceled by TDS operator** Attempt to receive data while the user being canceled by the TDS master operator.
Error = 300	**Socket connection closed during receive** Communication link failure.
Error = 1001	**Unable to create window (internal error)** DLL system error.

Error = 1002	**Too many thread in process (internal error)** DLL system error.
Error = 1003	**Unable to retrieve threadId (internal error)** DLL system error.
Error = 1004	**Buffer still allocated at end of thread or process** Some buffers were still allocated at end of thread or process.
Error = 1005	**Connection still opened at end of thread or process** Some connections were still opened at end of thread or process.
Error = 1006	**Abnormal termination received from peer** Socket disconnection on TDS initiative, it may happen when: – The user has sent a message "BYE" to TDS. – The user is cancelled by the master command CTC. – The user is disconnected because the maximum idle time is expired.
Error = 1007	**Invalid length (header and received length different)** DLL system error.
Error = 1008	**Buffer overflow (received data too long)** Allocated buffer too short to receive all the data.
Error = 1009	**Socket connection closed on receive** Socket disconnection occurred while receiving data.
Error = 1010	**Timeout event occurred** Timeout event occurred while trying to connect or to receive data.
Error = 1011	**Unable to create timer thread (internal error)** DLL system error.
Error = 1012	**Unable to create semaphore table (internal error)** DLL system error.
Error = 1013	**Unable to initialize semaphore (internal error)** System error.
Error = 1014	**Unable to register exit function (internal error)** System error.
Error = 1015	**Error on trace file access (internal error)** System error.
Error = 1016	**Error on trace file creation (internal error)** System error.

Error = 1017	**Error on trace file opening (internal error)** System error.
Error = 1018	**Receive error (timeout)** System error.
Error = 1019	**Receive error (internal error)** System error.
Error = 1020	**Error on semaphore locking (internal error)** System error.
Error = 1021	**Error on semaphore unlocking (internal error)** System error.
Error = 1022	**File not found:** ***file_name***

Socket Related Errors : Windowssockets

(Refer to the Microsoft Sockets documentation)

Error = 10004	Socket error: WSAEINTR
Error = 10009	Socket error: WSAEBADF
Error = 10013	Socket error: WSAEACCES
Error = 10014	Socket error: WSAEFAULT
Error = 10022	Socket error: WSAEINVAL
Error = 10024	Socket error: WSAEMFILE
Error = 10035	Socket error: WSAEWOULDBLOCK
Error = 10036	Socket error: WSAEINPROGRESS
Error = 10037	Socket error: WSAEALREADY
Error = 10038	Socket error: WSAENOTSOCK
Error = 10039	Socket error: WSAEDESTADDRREQ
Error = 10040	Socket error: WSAEMSGSIZE
Error = 10041	Socket error: WSAEPROTOTYPE
Error = 10042	Socket error: WSAENOPROTOOPT
Error = 10043	Socket error: WSAEPROTONOSUPPORT
Error = 10044	Socket error: WSAESOCKTNOSUPPORT
Error = 10045	Socket error: WSAEOPNOTSUPP
Error = 10046	Socket error: WSAEPFNOSUPPORT
Error = 10047	Socket error: WSAEAFNOSUPPORT
Error = 10048	Socket error: WSAEADDRINUSE
Error = 10049	Socket error: WSAEADDRNOTAVAIL
Error = 10050	Socket error: WSAENETDOWN
Error = 10051	Socket error: WSAENETUNREACH
Error = 10052	Socket error: WSAENETRESET
Error = 10053	Socket error: WSAECONNABORTED
Error = 10054	Socket error: WSAECONNRESET
Error = 10055	Socket error: WSAENOBUFS

Error = 10056	Socket error: WSAEISCONN
Error = 10057	Socket error: WSAENOTCONN
Error = 10058	Socket error: WSAESHUTDOWN
Error = 10059	Socket error: WSAETOOMANYREFS
Error = 10060	Socket error: WSAETIMEOUT
Error = 10061	Socket error: WSAECONNREFUSED
Error = 10062	Socket error: WSAELOOP
Error = 10063	Socket error: WSAENAMETOOLONG
Error = 10064	Socket error: WSAEHOSTDOWN
Error = 10065	Socket error: WSAEHOSTUNREACH
Error = 10066	Socket error: WSAENOEMPTY
Error = 10067	Socket error: WSAEPROCLIM
Error = 10068	Socket error: WSAEUSERS
Error = 10069	Socket error: WSAEDQUOT
Error = 10070	Socket error: WSAESTALE
Error = 10071	Socket error: WSAEREMOTE
Error = 10091	Socket error: WSASYSNOTREADY
Error = 10092	Socket error: WSAVERNOTSUPPORTED
Error = 10093	Socket error: WSANOTINITIALISED
Error = 11001	Socket error: WSAHOST_NOT_FOUND
Error = 11002	Socket error: WSATRY_AGAIN
Error = 11003	Socket error: WSANO_RECOVERY
Error = 11004	Socket error: WSANO_DATA

Socket Related Errors : AIX sockets

Error = 10002	Socket error: ENOENT
Error = 10004	Socket error: EINTR
Error = 10009	Socket error: EBADF
Error = 10013	Socket error: EACCES
Error = 10014	Socket error: EFAULT
Error = 10022	Socket error: EINVAL
Error = 10024	Socket error: EMFILE
Error = 10052	Socket error: ESTALE
Error = 10054	Socket error: EWOULDBLOCK
Error = 10055	Socket error: EINPROGRESS
Error = 10056	Socket error: EALREADY
Error = 10057	Socket error: ENOTSOCK
Error = 10058	Socket error: EDESTADDRREQ
Error = 10059	Socket error: EMSGSIZE
Error = 10060	Socket error: EPROTOTYPE
Error = 10061	Socket error: ENOPROTOOPT
Error = 10062	Socket error: EPROTONOSUPPORT
Error = 10063	Socket error: ESOCKTNOSUPPORT
Error = 10064	Socket error: EOPNOTSUPP
Error = 10065	Socket error: EPFNOSUPPORT

Error = 10066	Socket error: EAFNOSUPPORT
Error = 10067	Socket error: EADDRINUSE
Error = 10068	Socket error: EADDRNOTAVAIL
Error = 10069	Socket error: ENETDOWN
Error = 10070	Socket error: ENETUNREACH
Error = 10071	Socket error: ENETRESET
Error = 10072	Socket error: ECONNABORTED
Error = 10073	Socket error: ECONNRESET
Error = 10074	Socket error: ENOBUFS
Error = 10075	Socket error: EISCONN
Error = 10076	Socket error: ENOTCONN
Error = 10077	Socket error: ESHUTDOWN
Error = 10078	Socket error: ETIMEDOUT
Error = 10079	Socket error: ECONNREFUSED
Error = 10080	Socket error: EHOSTDOWN
Error = 10081	Socket error: EHOSTUNREACH
Error = 10083	Socket error: EPROCLIM
Error = 10084	Socket error: EUSERS
Error = 10085	Socket error: ELOOP
Error = 10086	Socket error: ENAMETOOLONG
Error = 10087	Socket error: ENOTEMPTY
Error = 10088	Socket error: EDQUOT
Error = 10093	Socket error: EREMOTE
Error = 10115	Socket error: ETOOMANYREFS
Error = 11001	Socket error: HOSTNOTFOUND
Error = 11002	Socket error: TRY_AGAIN
Error = 11003	Socket error: NO_RECOVERY
Error = 11004	Socket error: NO_DATA

Socket Related Errors : Linux sockets

Error = 10002	Socket error: ENOENT
Error = 10004	Socket error: EINTR
Error = 10009	Socket error: EBADF
Error = 10011	Socket error: EAGAIN, EWOULDBLOCK
Error = 10013	Socket error: EACCES
Error = 10014	Socket error: EFAULT
Error = 10022	Socket error: EINVAL
Error = 10024	Socket error: EMFILE
Error = 10036	Socket error: ENAMETOOLONG
Error = 10039	Socket error: ENOTEMPTY
Error = 10040	Socket error: ELOOP
Error = 10066	Socket error: EREMOTE
Error = 10087	Socket error: EUSERS
Error = 10088	Socket error: ENOTSOCK

Error = 10089	Socket error: EDESTADDRREQ
Error = 10090	Socket error: EMSGSIZE
Error = 10091	Socket error: EPROTOTYPE
Error = 10092	Socket error: ENOPROTOOPT
Error = 10093	Socket error: EPROTONOSUPPORT
Error = 10094	Socket error: ESOCKTNOSUPPORT
Error = 10095	Socket error: EOPNOTSUPP
Error = 10096	Socket error: EPFNOSUPPORT
Error = 10097	Socket error: EAFNOSUPPORT
Error = 10098	Socket error: EADDRINUSE
Error = 10099	Socket error: EADDRNOTAVAIL
Error = 10100	Socket error: ENETDOWN
Error = 10101	Socket error: ENETUNREACH
Error = 10102	Socket error: ENETRESET
Error = 10103	Socket error: ECONNABORTED
Error = 10104	Socket error: ECONNRESET
Error = 10105	Socket error: ENOBUFS
Error = 10106	Socket error: EISCONN
Error = 10107	Socket error: ENOTCONN
Error = 10108	Socket error: ESHUTDOWN
Error = 10109	Socket error: ETOOMANYREFS
Error = 10110	Socket error: ETIMEDOUT
Error = 10111	Socket error: ECONNREFUSED
Error = 10112	Socket error: EHOSTDOWN
Error = 10113	Socket error: EHOSTUNREACH
Error = 10114	Socket error: EALREADY
Error = 10115	Socket error: EINPROGRESS
Error = 10116	Socket error: ESTALE
Error = 10122	Socket error: EDQUOT
Error = 11001	Socket error: HOSTNOTFOUND
Error = 11002	Socket error: TRY_AGAIN
Error = 11003	Socket error: NO_RECOVERY
Error = 11004	Socket error: NO_DATA

5.3.1 TDS ERROR MESSAGES

The following TDS error messages are specific to TDS-TCP/IP and may occur at TDS execution time.

```
MV79.  TCP/IP COMMUNICATION IS NOT INITIALIZED ERRNO=XXXXX ON XXXXXXXXXXXXX
```

Meaning:	An error occurred during TDS-TCP/IP initialization: ERRNO=XXXXX is the error code returned by the SOCKG 7 function invoked. XXXXXXXXXXXXXX is the name of the invoked SOCKG 7 function which encountered a problem.
Action:	Check "host" and "services" configuration files used for the chosen TCP/IP configuration. Check that OPEN 7 and SOCKG 7 products are active and ready on the site.

```
MV80.  TCP/IP FUNCTION IS NOT AVAILABLE
```

Meaning:	The product TDS-TCP/IP (Marketing Identifier) has not been purchased for the site. The H_SM_DCM SM of SYS.DCM.SYSTEM is not loaded.
Action:	Verify if the product was purchased or not. Ensure that the SM containing H_SM_DCM is loaded.

5.4 Client Trace

The ATMITDS.TRC file allows to debug TDS-TCP/IP client applications. If the trace mode is activated (either TRACE_API or TRACE_SOC not null), this file is dynamically created, in the user project directory, by the DLL.

Since the DLL version 3.1.0, a path name can be specified to store the files ATMITDS.TRC and ATMITDS.LOG in a chosen folder. The path name is defined by the PATH key in the ATMI.INI file.

EXAMPLE :

PATH=D:\MY_TRACE_FOLDER

If the atmi.ini file is not found in the system directory (usually C:\WINNT), only the ATMITDS.LOG is always created in the current working directory with the event "1022 File not found" logged and possibly other errors.

If in the atmi.ini file, the keyword SUPPRESS_LOG is found to 1, the ATMITDS.LOG is **never created.**

If the path is not specified in the atmi.ini file, the ATMITDS.LOG and ATMITDS.TRC files can be created in the current working directory.

If the file ATMITDS.TRC can not be created or reached, this event is written in ATMITDS.LOG

❑

5.4.1 Activating the Trace

To activate the trace, keys must be set as follows in the atmi.ini file:

```
DEBUG = {0 | 1 | 2}
TRACE_API = {0 | 1 | 2}
TRACE_SOC = {0 | 1 | 2}
```

DEBUG	relates to the debug mode. If 0, no error messages are echoed at the console. If 1, all messages except disconnection messages are displayed at the console. If 2, all messages are displayed.
TRACE_API	relates to the trace of the API calls. Its value indicates the trace value. If its value is 1, only the first 32 bytes of the buffers are dumped. If its value is 2, the whole contents of the buffers is dumped.

TRACE_SOC	relates to the trace of the socket verbs. Its value indicates the trace value. TRACE_SOC is reserved for system debugging.

The Trace Format

```
 First field: time / second field: thread number / third field: DLL load event
      11:36:21.710 0000 ***** ATMI TDS TRACE ***** (version: 3.0.3.1) 20/11/1998


→     : implies an event or API call received by the DLL
      11:36:21.710  0001 → DLL new thread   :   winThreadId=FFFC9CDD ProcessId=1
      11:36:39.390  0001 → tpalloc bufsize=2000


>>>   : implies API input parameter values
      11:36:39.390  0001 >>> type  : addr=0041B1C0 lg=8
                    0001     0000 585F435F  54595045
      X_C_TYPE
      11:36:39.390  0001 >>> subtype  : addr=00412580 lg=4
                    0001     0000 617A7177
      azqw


<<<   : implies API output parameter values
      11:36:39.390  0001 <<< tpalloc bufptr=0055000c


===   : implies a socket trace event


***   : implies a DLL error not related to a specific thread
```

6. TCP/IP Transactions Using FORMS Facility

6.1 Generality

A TDS transaction running on GCOS 7 in **formatted** mode can be used by a client application located on a PC as well as by a terminal. That implies no modification of the SDPI forms routines (refer to the *FORMS User's Guide*).

The server and client applications process the FORMS dialog by the exchange of structured messages:

- Each message will contain as many fields as the number of named fields (NF) defined in the form generation. The unnamed fields (UF) are never sent to the client application.
- Only the contents of the named fields, without any qualification or rendition attributes, are sent to the client application.
- All fields, transmittable or not transmittable, are exchanged in the same order in which they are displayed on a terminal screen (i.e., left to right, top to bottom). If several forms are activated in APPEND mode, the messages of each active form are added in the same order as the form activation.
- Each field is separated from the preceding one by a delimiter character. This delimiter is currently the TAB character (i.e., "05" EBCDIC from server application, "09" ASCII from client application).
- Two consecutive delimiter characters in a message mean that the corresponding field is cleared or empty.
- Pluri Language West characters (PLW) are supported according to the ISO 8859-1 specification (PLW in ASCII 8-bits mode).
- If a client application sends a function-key, the key profile FC1 FC2 is placed at the beginning of the message. It is preceded by the ESC CSI characters (i.e., "274A" EBCDIC from the server application, "1B5B" ASCII from the client application), and followed by a lower-case u character. FC1 FC2 takes the values "01" to "24".

- Fields with the pseudo-graphics characters (CSPS) attribute, contain only CSPS or blank characters. A serial of CSPS characters begins always by a leading SHIFT-OUT character (i.e., "0E" EBCDIC or ASCII).

On the GCOS 7 server, *Formname*_OF_DKU7107 object form file must be present in the binary library or UFAS files, or *Formname*_OF_ANY object form file in the binary library.

6.2 SDPI Verbs Particularities

Refer to *FORMS User's Guide*, Appendix A.

6.2.1 Forms Activation (CDGET)

For a TM transaction, this action will place the fixed form in the message sent to the terminal in order to format the screen.

For a TCP/IP transaction, none of this information is placed in the message sent to the client application. When a level 3 CDGET is performed, all the fields of the activating forms are sent to the client application. All these fields are blank.

6.2.2 Forms Send (CDSEND)

This action causes data to be transferred to the client application. All named fields, either specified in the selection-vector or not, are put in the message.

6.2.3 Forms Receive (CDRECV)

This action causes data to be received from the message sent by the client application to the user data record according to the user selection vector. The message contains all the named fields.

The message sent by the client application is checked:

When the received field number is different from the form field number, a status key A7 (RECARERR return code) is set, and no data transferred to the server application.

When a received protected field has been modified by the client application, the corresponding selection vector is set to "A". If this field is transmittable, it is not delivered. If it is the last field in error in the form, a status key AB (ALMOST return code) is set.

When a received field length exceeds the form field length, the corresponding selection vector is set to "O". If this field is transmittable, it is not delivered. If it is the last field in error in the form, a status key AB (ALMOST return code) is set.

6.2.4 Forms Release (CDRELS)

This action does not put any data in the message.

6.2.5 Forms Purge Input Data (CDPURGE)

This action purges all pending input messages and gives the control to the server application.

6.2.6 Forms Attribute or List Attribute Selection (CDATTR or CDATTL)

This action does not put any data in the message, except if a CLEAR action has been specified by the server application in the selection vector.

In this case, all the named fields (NF) are put in the message to be sent to the client application.

Else, an empty message is sent.

6.2.7 Forms Identification (CDFIDI)

This action returns the name of the form having data pending from the last client received message.

When the received field number is different from the form field number, a status key A7 (RECARERR return code) is set, and no data is transferred to the server application.

6.2.8 Forms Mechanism Function (CDMECH)

This action activates the control function mechanism.

This action does not put any data in the message, except in the following case. If a mechanism CLEAR, INITAT or INIT is performed, and if a CLEAR action has been specified by the server application in the selection vector, then all the named fields are put in the message to be sent to the client application, else, an empty message is performed.

6.3 Limitations

Most of the TDS FORMS functionalities are supported by TDS-TCP/IP transactions with the following restrictions:

- TELETRANS applications, running with the special <<T>> mode activation are not supported. Under TDS-TCP/IP they lead to a transaction abort with a status key AB (FUNCNAV return code).
- The CDMECH CPON/CPOFF mechanism, which notifies the cursor position at <<TRANSMIT>> key press to the TDS application, is not processed. TCP/IP transactions with such a mechanism, run as for a terminal without this functionality support.
- The DETECTABLE, IMMEDIAT-TRANSMIT (running on terminals with a light-pen device) , and INPUT-DEVICE-BD (running on terminals with a badge reader) attributes are not processed.
- The initial data, specified at form generation, is not retrieved from the form internal structure during CDGET processing. It is lost.

6.4 Data Flow Example

Suppose an application running with a form F1.

This form begins with an unnamed field UF1, preceding a named field NF1 with protected and not transmittable attributes, followed by a UF2 preceding an unprotected NF2, terminated by an UF3 followed by an unprotected NF3.

This transaction starts by:

a TPR1 which performs:
  CDGET F1 level=1
  CDATTR F1 level=1 attribute = PR on NF3
  CDSEND F1 level=3 "ABC" to NF1, "123" to NF3
a TPR2 performing a CDRECV on all 3 NFs

At TPR termination of TPR1, the message sent to the terminal or the client is the following:

To the terminal

UF1	NF1 definition
UF2	NF2 definition
UF 3	NF3 definition
CP1 "ABC"	CP2 " "
CP3 PR "123"	

To the TCP/IP client application

"ABC" TAB	" " TAB
"123"	

At TPR beginning of TPR2, if the response is "STOP" on NF2, the message sent from the terminal or the client is the following:

From the terminal

"STOP" TAB	"123"

From the TCP/IP client application

"ABC" TAB	"STOP" TAB
"123"	

❑

7. Protocol between SA7 and the client application

The application on PC connected to a secured TDS shall expect messages from SA7 and be able to answer them.

This protocol is available from SA7 V3.5 which can be installed with the INTEROP7 facility called ISI7 minimum tape version IS230.

For a TCP/IP correspondent, the messages from SA7 are in the following format:

<header>:<function>:<status>:<message>

where (the character « **:** » being used as a delimiter):

<header> = « SA7TCP »:	a sequence of 6 characters indicating that the message is issued from SA7.
<function>,	a 3-character code: = « 001 »: end of check = « 002 »: request for password = « 003 »: request for password change
<status>,	a 3-character code (status codes are those used by SECUR'ACCESS for a TM type correspondent; they are defined in the SAMES1 member of the SA7.LIV.SL library): = « 000 »: connection accepted if function = 001 password change accepted if function = 003 = a letter followed by 2 digits indicating the reason for function request - e.g.: D41: « initial password » D73: « wrong password, 2 attempts left » D74: « wrong password, last attempt » E48: « password expired » E50: « password soon expired » .....

	= 3 digits indicating the reason for connection denied - e.g.: 060: password expired 068: user lacking security rights 098: user stuck for password 108: validity date exceeded ...
<message>:	gives the message associated with <status>; this message is in the language defined for the user.

The answers expected by SA7 are in the following format:

when a password is requested,

<header>:<function>:<action-code>:<user-name>:<parameter1>

when a password change is requested,

<header>:<function>:<action-code>:<user-name>:<parameter1>:<parameter2>

where:

<header>	(6 characters) takes the same value as in the request from SA7
<function>	(3 characters) is the value of the request from SA7
<action-code>	(1 character) is a code used for validating or ignoring the function: = « A »: function ignored = « V »: function validated = « C »: password change requested when answering a password request
<user-name>	(12 characters) is the name of the user, if different from the name used for connection
<parameter1>	(12 characters) is the user password
<parameter2>	(12 characters) is the new user password.

In GCOS 7 TS 9764 the messages from the SA7 TPRs are considered as application type messages, the client API must therefore comply with the protocol described above.

Like for a DSA terminal, the option retained for the TS 9764 is to authorize a maximum of 3 password attempts before denying connection to TDS.

EXAMPLES OF DIALOGS:

1. Connection with a wrong password:

   SA7 issues:
   (English)
   SA7TCP :002 :D73 :ERRONEOUS PASSWORD : TWO TRIES AGAIN
   (French)
   SA7TCP :002 :D73 :MOT DE PASSE ERRONE : ENCORE 2 ESSAIS
   SA7 receives:
   SA7TCP :002 :V : :SA7U :
   SA7 issues: (SA7U password correct)
   SA7TCP :001 :000 :

2. Connection with a password correct but to be changed soon:

   SA7 issues:
   (English)
   SA7TCP :003 :E50 :YOUR PASSWORD WILL EXPIRE SOON
   (French)
   SA7TCP :003 :E50 :MOT DE PASSE BIENTOT PERIME
   SA7 receives:
   SA7TCP :003 :A :
   SA7 issues (connection accepted, password change not mandatory):
   SA7TCP :001 :000 :

3. Connection with an expired password:

   SA7 issues:
   (English)
   SA7TCP :003 :E48 :PASSWORD HAS EXPIRED : MUST BE CHANGED
   (French)
   SA7TCP :003 :E48 :MOT DE PASSE PERIME
   A7 receives:
   SA7TCP :003 :V : :SA7U2 :USA7 :
   SA7 issues (password changed from SA7U2 to USA7):
   SA7TCP :001 :000 :

4. Password change requested by the user, SAUTIL1 transaction:

   SA7 issues:
   (English)
   SA7TCP :003 :A31 :PASSWORD CHANGE
   (French)
   SA7TCP :003 :A31 :CHANGEMENT DE MOT DE PASSE
   SA7 receives:
   SA7TCP :003 :V : :SA7U2 :AZER :
   SA7 issues (password changed from SA7U2 to AZER)
   SA7TCP :003 :000 :

5. Connection denied for a user who entered 3 wrong passwords in succession:

   SA7 issues:
   (English)
   SA7TCP :001 :098 : USER BLOCKED FOR PASSWORD
   (French)
   SA7TCP :001 :098 : UTILISATEUR BLOQUE POUR MOT DE PASSE

❑

8. AIX or Linux Client

8.1 Installation

The AIX or Linux machine must have an appropriate communications card allowing connection to a TCP/IP network and must be able to run the software listed below.

The AIX or Linux operating system must allow development and execution of client applications.

Two files are delivered for the product :

- Atmi.h, an include file required for compilation of the client application program
- XATMI, the shared library providing the interface with TDS-TCP / IP transactions.

The AIX or Linux machine administrator may decide where to install the shared library XATMI, and the include file Atmi.h.

8.2 Configuration

AIX or Linux machines must be configured for each TDS application. Two files hold the relevant information:

- the address for remote host mapping is held in the **/etc/hosts,**
- the service declaration is held in the **/etc/services** file.

The **/etc/hosts** file must be updated to include the IP addresses of the names of the hosts running the TDS applications. A host name is the symbolic name used in the client application to address the DPS 7000 system (corresponding to the *hostid* parameter of the *tpconnect* verb).

EXAMPLE:

129.182.50.50	bc0f	#host name of the DPS 7000 on which the TDS **tds1** runs

❑

The **/etc/services** file must be updated to include the required service names, the port number associated with each service, and the protocol used. The service name is the concatenation of the host name and the TDS name. The port number must be the same as the one declared on the GCOS 7 server.

EXAMPLE:

If a client application wants to connect to the TDS **tds1** located on the DPS 7000 referred to by the host name **bc0f**, the following line must appear in the /etc/services file:

```
bc0ftds1                10100/tcp
```

❑

NOTE:
The value of the port number must be greater than 1024.

8.3 Programming

From the shared library XATMI version 1.2.0, a data conversion indicator may be supplied to the *tpconnect* function, using the *dataconvert* field in the *data* parameter.

A new environment variable is created, ATMI_CONNECT_TIMEOUT. Used during a tpconnect function call, it gives the maximum waiting time for the connection establishment, expressed in milliseconds. Its default value is tcp_keepinit network attribute value (usually 75 seconds).The maximum number of TCP/IP connections per client process is 64. From the shared library XATMI version 1.3.0, this number is raised to 500.

The TDS-TCP/IP client API is composed of the following functions:

- four functions for conversational services: *tpconnect*, *tpdiscon*, *tprecv* and *tpsend*
- two functions for typed buffer management: *tpalloc* and *tpfree*
- two functions for error return: *tperrno* and *tperrdtl,* these functions being extensions of XATMI.

The functions prototypes are described in **Chapter 4.3**.

For a client application program coded in C, an include file Atmi.h is delivered containing the description of the connection structures, the external function prototypes, the event types, and error codes.

The Atmi.h file must be put in the client working directory.

In the source code using API functions, the following directive may be provided:

```
#include "Atmi.h"
```

In order to access the shared library XATMI from a AIX or Linux client application program, a Makefile of the following form is required :

```
VPATH= <path-of-source-code>
DASHO= -g
CFLAGS= $(DASHO)
LDFLAGS= XATMI -L '.'

atmi_test.o: $(VPATH)/atmi_test.c
     $(CC) ${CFLAGS} -c $(VPATH)/atmi_test.c
test_atmi: atmi_test.o
     $(CC) -o test_atmi atmi_test.o $(LDFLAGS)
```

8.4 Example

The following example is a program written in C.

```
/*******************************************************************/
/* Complete test of all entry points (connection with termid)      */
/*******************************************************************/
#include <stdio.h>
#include <sys/types.h>

/*******************************************************************/
/* To be included in all  client programmes                    */
/*******************************************************************/
#include "Atmi.h"
/*******************************************************************/

char *tpbuffer = NULL;
long tplen;
long tpflags;
long tprevent;
int tpcd;
int tpret;

/*******************************************************************/
/* subroutine to send data to TDS                                  */
/* return value :                                                  */
/*  0 : normal exit                                                */
/* -1 : error signalled and tpdiscon OK                            */
/* -2 : error signalled and tpdiscon KO                            */
/*******************************************************************/
int send_to_tds()
{
  tpret = tpsend(tpcd, tpbuffer, tplen, tpflags, &tprevent);
  if (tpret == -1)
        {
        tpret = tperrno();
        fprintf (stdout, "tpret after send : %d\n", tpret);
        tpret = tpdiscon(tpcd);
if (tpret == -1)
{
tpret = tperrno();
fprintf (stdout, "tpret after disconnect : %d\n", tpret);
return (-2);
}
return (-1);
}
  return (0);
}
```

```
/*****************************************************************/
/* subroutine to receive data from TDS                           */
/* return value :                                                */
/*  0 : normal exit                                              */
/* -3 : error signalled and tpdiscon OK                          */
/* -4 : error signalled and tpdiscon KO                          */
/*****************************************************************/
int receive_from_tds()
{
  tpret = tprecv(tpcd, &tpbuffer, &tplen, tpflags, &tprevent);
  if (tpret == -1)
         {
         tpret = tperrno();
         if ((tpret == TPEEVENT)&& (tprevent == TPEV_SENDONLY))
               return (0);
         else
            {
            fprintf (stdout, "tpret after receive : %d\n", tpret);
            tpret = tpdiscon(tpcd);
            if (tpret == -1)
               {
               tpret = tperrno();
               fprintf (stdout, "tpret after disconnect : %d\n", tpret);
               return (-4);
               }
            return (-3);
            }
         }
  return (0);
}
/*****************************************************************/
#define min(a,b)  ( ((a) < (b))? (a): (b) )
/*****************************************************************/

main(argc, argv)
  int             argc;
  char           *argv[];
{
char tptype[16] = "X_C_TYPE";
char tpsubtype[16] = " ";
long tpsize=32784;
long tpbuflen;
int outlg;
char outbuff[50];
char inbuff[50];
int ret;

struct service_id {
  char hostid[16];
  char tdsname[5];
} tpsvc;
```

```
struct subtype {
  char name[13];
  char project[13];
  char billing[13];
  char password[13];
  char termId[13];
} tpdata;
/*******************************************************************/
/* allocation of buffer for communication with TDS                 */
/*******************************************************************/
  tpbuffer = tpalloc(tptype, tpsubtype, tpsize);
  if (tpbuffer == NULL)
        {
        fprintf(stdout, "tpalloc did not work");
        exit(-5);
        }
/*******************************************************************/
/* effect connection to TDS                                        */
/*******************************************************************/
  strcpy(tpsvc.hostid,"bc06-42");
  strcpy(tpsvc.tdsname,"espt");
  strcpy(tpdata.name,argv[1]);
  memcpy(tpdata.password,argv[2], sizeof(argv[1]));
  strcpy(tpdata.project,"ESP7");
  strcpy(tpdata.billing, "ESP7");
  strcpy(tpdata.termId, "MyTerm");
  memcpy(tpbuffer,&tpdata,sizeof(tpdata));
/*******************************************************************/
/* set tpbuflen = 65 to allow passage of terminal identifier       */
/*******************************************************************/
  tpbuflen = 65;
  tpflags = 0;
  tpcd = tpconnect((char *) &tpsvc, tpbuffer, tpbuflen, tpflags);
  if (tpcd == -1)
        {
        tpret = tperrno();
        fprintf (stdout, "tpret after connect : %d\n", tpret);
        tpfree(tpbuffer) ;
exit(-6);
        }
/*******************************************************************/
/* retrieve READY message after connection                         */
/*******************************************************************/
  do
        {
        ret = receive_from_tds();
        if (ret != 0)
              {
tpfree(tpbuffer) ;
exit (ret);
}
        outlg = min(tplen, 50);
        memset(outbuff,(char) 0, 50);
        memcpy(outbuff, tpbuffer, outlg);
        fprintf(stdout, "returned buffer : %s\n", outbuff);
  } while (tprevent != TPEV_SENDONLY);

/*******************************************************************/
/* use tpsend to request TRACE PRINT                               */
/*******************************************************************/
  memset(tpbuffer,(char) 0, 256);
  memcpy(tpbuffer, "TRACE PRINT", sizeof("TRACE PRINT"));
  tplen = 11;
  memset(inbuff,(char) 0, 50);
  memcpy(inbuff, tpbuffer, tplen);
  fprintf(stdout,"sent buffer : %s\n", inbuff);
  ret = send_to_tds();
  if (ret != 0)
        {
tpfree(tpbuffer) ;
exit (ret);
}
/*******************************************************************/
/* retrieve message after TRACE PRINT                              */
/*******************************************************************/
  do
        {
        ret = receive_from_tds();
        if (ret != 0)
              {
tpfree(tpbuffer) ;
exit (ret);
}
        outlg = min(tplen, 50);
        memset(outbuff,(char) 0, 50);
        memcpy(outbuff, tpbuffer, outlg);
        fprintf(stdout, "returned buffer : %s\n", outbuff);
  } while (tprevent != TPEV_SENDONLY);
/*******************************************************************/
/* use tpsend to execute user TPR (STYLEADD)                       */
/* STYLEADD adds a record to a UFAS sequential indexed file        */
/*******************************************************************/
  memset(tpbuffer,(char) 0, 256);
  memcpy(tpbuffer, "STYLEADD  STDeee¤¤¤", sizeof("STYLEADD  STDeee¤¤¤"));
  tplen = sizeof("STYLEADD  STDeee¤¤¤") -1;
  memset(inbuff,(char) 0, 50);
  memcpy(inbuff, tpbuffer, tplen);
  fprintf(stdout,"sent buffer : %s\n", inbuff);
  ret = send_to_tds();
  if (ret != 0)
        {
tpfree(tpbuffer) ;
exit (ret);
}
```

```
/*****************************************************************/
/* retrieve reply from STYLEADD                                  */
/*****************************************************************/
  do
        {
        ret = receive_from_tds();
        if (ret != 0)
              {
tpfree(tpbuffer) ;
exit (ret);
}
         outlg = min(tplen, 50);
         memset(outbuff,(char) 0, 50);
         memcpy(outbuff, tpbuffer, outlg);
         fprintf(stdout, "returned buffer : %s\n", outbuff);
  } while (tprevent != TPEV_SENDONLY);

/*****************************************************************/
/* use tpsend to execute user TPR (STYLLST) which reads          */
/* successive records from a UFAS sequential indexed file        */
/*****************************************************************/
  memset(tpbuffer,(char) 0, 256);
  memcpy(tpbuffer, "STYLLST   ", sizeof("STYLLST   "));
  tplen = sizeof("STYLLST   ") -1;
  memset(inbuff,(char) 0, 50);
  memcpy(inbuff, tpbuffer, tplen);
  fprintf(stdout,"sent buffer : %s\n", inbuff);
  ret = send_to_tds();
  if (ret != 0)
        {
tpfree(tpbuffer) ;
exit (ret);
}
/*****************************************************************/
/* retrieve replies from STYLLST                                 */
/*****************************************************************/
  do
        {
        ret = receive_from_tds();
        if (ret != 0)
              {
tpfree(tpbuffer) ;
exit (ret);
}
         outlg = min(tplen, 50);
         memset(outbuff,(char) 0, 50);
         memcpy(outbuff, tpbuffer, outlg);
         fprintf(stdout, "returned buffer : %s\n", outbuff);
  } while (tprevent != TPEV_SENDONLY);
```

```
/*****************************************************************/
/* use tpsend to execute user TPR (STYLEDEL)                     */
/* STYLEDEL deletes a record to a UFAS sequential indexed file   */
/*****************************************************************/
  memset(tpbuffer,(char) 0, 256);
  memcpy(tpbuffer, "STYLEDEL  STDe", sizeof("STYLEDEL  STDe"));
  tplen = sizeof("STYLEDEL  STDe") -1;
  memset(inbuff,(char) 0, 50);
  memcpy(inbuff, tpbuffer, tplen);
  fprintf(stdout,"sent buffer : %s\n", inbuff);
  ret = send_to_tds();
  if (ret != 0)
        {
tpfree(tpbuffer) ;
exit (ret);
}
/*****************************************************************/
/* retrieve reply from STYLEDEL                                  */
/*****************************************************************/
  do
         {
         ret = receive_from_tds();
         if (ret != 0)
         {
tpfree(tpbuffer) ;
exit (ret);
}
         outlg = min(tplen, 50);
         memset(outbuff,(char) 0, 50);
         memcpy(outbuff, tpbuffer, outlg);
         fprintf(stdout, "returned buffer : %s\n", outbuff);
  } while (tprevent != TPEV_SENDONLY);

/*****************************************************************/
/* use tpsend to execute TRACE OFF                               */
/*****************************************************************/
  memset(tpbuffer,(char) 0, 256);
  memcpy(tpbuffer, "TRACE OFF", sizeof("TRACE OFF"));
  tplen = sizeof("TRACE OFF") -1;
  memset(inbuff,(char) 0, 50);
  memcpy(inbuff, tpbuffer, tplen);
  fprintf(stdout,"sent buffer : %s\n", inbuff);
  ret = send_to_tds();
  if (ret != 0)
        {
tpfree(tpbuffer) ;
exit (ret);
}
```

```
/*******************************************************************/
/* retrieve reply from TRACE OFF                                   */
/*******************************************************************/
  do
         {
         ret = receive_from_tds();
         if (ret != 0)
               {
tpfree(tpbuffer) ;
exit (ret);
}
         outlg = min(tplen, 50);
         memset(outbuff,(char) 0, 50);
         memcpy(outbuff, tpbuffer, outlg);
         fprintf(stdout, "returned buffer : %s\n", outbuff);
  } while (tprevent != TPEV_SENDONLY);
/*******************************************************************/
/* use tpsend to execute the BYE transaction                       */
/*******************************************************************/
  memset(tpbuffer,(char) 0, 256);
  memcpy(tpbuffer, "BYE", sizeof("BYE"));
  tplen = sizeof("BYE") - 1;
  memset(inbuff,(char) 0, 50);
  memcpy(inbuff, tpbuffer, tplen);
  fprintf(stdout,"sent buffer : %s\n", inbuff);
  ret = send_to_tds();
  if (ret != 0)
        {
tpfree(tpbuffer) ;
exit (ret);
}
/*******************************************************************/
/* received buffer should be empty after BYE                       */
/*******************************************************************/
  ret = receive_from_tds();
  if (ret != 0)
        {
tpfree(tpbuffer) ;
exit (ret);
}
  if (tplen != 0)
         {
         outlg = min(tplen, 50);
         memset(outbuff,(char) 0, 50);
         memcpy(outbuff, tpbuffer, outlg);
         fprintf(stdout, "returned buffer : %s\n", outbuff);
         }
/*******************************************************************/
/* liberate buffer                                                 */
/*******************************************************************/
  tpfree(tpbuffer);
}
```

8.5 Debugging

8.5.1 Client Trace

The file ATMITDS.TRC_< process id>, where <process id> represents the process identifier for the client application program, allows you to debug TDS-TCP/IP client applications.

If the trace mode is activated, this file is dynamically created, in your current directory, by the shared library.

Trace information is first accumulated in a trace buffer of 512K bytes, and then transferred to the trace file, which is maintained in circular fashion. Each entry is preceded by a trace header. Note that the size of the trace file is 1024K plus the twice the length of the header information.

Trace information is transferred to the trace file from the trace buffer :

1. whenever the trace buffer becomes full
2. whenever an error is detected
3. on execution of the last tpfree call of an application; the number of tpalloc calls are counted, and, whenever a tpfree call is executed, the counter is decremented. Tracing to the trace file takes place whenever the counter becomes zero.

The client trace facility is activated by positioning one or both of the environment variables ATMI_TRACE_PATH, ATMI_TRACE_SIZE, ATMI_SUPPRESS_LOG,ATMI_TRACE_API (for API tracing) and ATMI_TRACE_SOC (for internal socket functions).

These environment variables have the following significance :

ATMI_TRACE_PATH	Used during trace and log files creation, it gives the path (absolute or relative) of the directory in which trace and log files will be stored. If the given path is wrong, default value is applied. Default value : path of the current directory where the application is launched.
ATMI_TRACE_SIZE	Used during writing in the circular trace file, it gives the maximum trace file size, expressed in MegaBytes. Default value : 1
ATMI_SUPPRESS_LOG	Used to suppress the logging into log file. Default value: 0 (log file is created if an error occurs)

ATMI_TRACE_API	relates to the trace of the API calls. Its value indicates the trace value. If its value is 0 (default value), no tracing is effected. If its value is 1, only the first 32 bytes of the buffers are dumped. If its value is 2, the whole contents of the buffers is dumped.

Note that for normal functioning, a value of 0 is recommended. The value of 1 is recommended if a problem arises where the content of send / receive buffers is not required. Value 2 should be reserved for the case of a problem when the send / receive buffer contents need to be checked.

ATMI_TRACE_SOC	relates to the tracing of the socket verbs. Its value indicates the trace value . If its value is 0 (default value), no socket tracing is effected. If its value is 1, the socket verbs are traced. If its value is 2, the send / receive headers exchanged with TDS are traced. ATMI_TRACE_SOC is reserved for system debugging.

Note that these environment variables have to be exported in the standard way in order for the application to have access to their values.

Each trace file entry (0 or 1) begins with a header of the following format :

```
**********************************************
*** Write to trace file : length = 5586
**********************************************
Mon Sep 13 17:10:36 1999 ***** ATMI TDS TRACE ***** (version : AIX 2.0.0)
```

Subsequent fields in the trace file have the following syntax :

-> : implies an event or API call received, for example :

```
Mon Sep 13 17:10:36 1999(00000000) --> tpalloc bufsize=32784
```

Note that the thread index (in this case, 0) is given in brackets .

```
              >>>   : implies API input parameter values, for example :

(00000000)>>> type : addr=2FF229E8 lg=8
(00000000)    0000 585F435F 54595045          X_C_TYPE
(00000000)>>> subtype : addr=2FF229F8 lg=1
(00000000)    0000 20

              <<<   : implies API output parameter values, for example :

Mon Sep 13 17:10:36 1999(00000000) <<< tpalloc bufptr=200007f8

        ===   : implies a socket trace event, for example :

Mon Sep 13 17:10:45 1999(00000000) === socket : socketId=00000004 lasterr=0

***   : implies any other type of  error
```

8.5.2 Client Log

The file ATMITDS.LOG_< process id>, where <process id> represents the process identifier for the client application program, is automatically produced when an error occurs throughout the XATMI functions if the environment variable ATMI_SUPPRESS_LOG is set to 0.

This file is dynamically created in ATMI_TRACE_PATH if any or in your current directory.

8.5.3 Logging of error messages at the console

Independently of the trace file mechanism, the user may request logging of error messages at the console. This is done by setting the environment variable ATMI_DEBUG.

This environment variable has the following significance :

ATMI_DEBUG	If the value is 0, there is no console display
	If the value is 1, only abnormal disconnection messages are displayed at the console
	If the value is 2, all errors are displayed at the console.

Note that this environment variable has to be exported in the standard way in order for the application to have access to its values.

❑

Glossary

API
Application Program Interface.

client application
In the client/server model, it is the application that requests a service to be provided by the server application.

Client/Server model
Generic name defining a standard way for applications to co-operate. The two co-operating applications can be located on the same system or on two different systems; in the latter case, the two applications co-operate via a communication link.

correspondent
The user of a PC application that dialogs with GCOS 7 transaction via a TCP/IP (or OSI/DSA) link.

DLL
Windows Dynamic Link Library

Ethernet
One of the best known technologies used for Local Area Networks (LAN).

FDDI
Fiber Distributed Data Interface: a technology based on optical fiber and used for Local Area Networks (LAN).

GXTI
GCOS 7 XTI: GCOS 7 component implementing the X/Open Transport Interface.

OPEN 7
GCOS 7 sub-system that provides a subset of UNIX system functions.

OSI/DSA
Open Systems Interconnection/Distributed Standard Architecture:
DSA is the name of the implementation by Bull of the OSI communication model.

PC
Personal Computer

RAD tool
Rapid Application Development tool.

SA7
Secur'Access7: a GCOS 7 TDS that provides security facilities.

server application
In the client/server model, it is the application that provides a service requested by the client application.

shared library
Any code module that can be accessed and used by many applications. Shared libraries are used primarily for sharing common code between different executable files or for breaking an application into separate components, thus allowing easy upgrades.

sockets
Berkeley sockets: one of the two most prevalent communications APIs, originally developed for UNIX systems (the second being TLI, Transport Layer Interface) and supporting various communication protocols, in particular TCP/IP. The socket interface was first implemented by Berkeley in the 1980s.

SOCKG 7
SOCKet GCOS 7 component that provides a standard socket interface.

'tar' archive
A single file, created by the command tar, which stores the contents of many files.

TCP/IP
Transmission Control Protocol/Internet Protocol: name of an international "de facto" standard set of communication protocols; the Internet network is made up of the interconnection of thousands of different networks all using TCP/IP.

TCP/IP protocols the result of developments done in the late 1960s and 1970s for the DARPA (Defense Advanced Research Projects Agency).

TCP stands for Transmission Control Protocol; it is a connection-oriented protocol that provides a reliable, full duplex, byte steam for user processes; TCP defines the set of protocols to be used at the Transport Layer level (OSI layer 4).

IP stands for Internet Protocol and provides the packet delivery for TCP (and others such as UDP, ICMP,...): IP defines the set of protocols to be used at the Network Layer level (OSI layer 3).

TDS-HA
TDS High Availability

TM correspondent
One of the OSI/DSA correspondents.

XATMI
X/Open Application Transaction Manager Interface:
one of the three levels of the DTP model of X/Open (Distributed Transaction Processing).

XCP1 correspondent
XCP2 correspondent
OSI/DSA correspondents.

❑

Index

❑

Technical publication remarks form

Title :	DPS7000/XTA NOVASCALE 7000 TDS-TCP/IP User's Guide Transaction processing: General

Reference N° :	47 A2 37UT 07	**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 37UT 07