

Transbase®
Getting Started

Transaction Software GmbH
Willy-Brandt-Allee 2
D-81829 München
Germany
Phone: +49-89-62709-0
Fax: +49-89-62709-11
Email: info@transaction.de
<http://www.transaction.de>

Version 6.8.1.40
November 02, 2010

Contents

1	Overview of Transbase Manuals	3
2	Interactive Use	4
2.1	Setting the Environment	4
2.2	Creating a Database	4
2.3	Accessing the Database interactively	6
2.4	Administrating Databases	7
2.4.1	Changing Database Parameters	7
2.4.2	Boot and Shutdown	8
2.4.3	Logging and Transaction Recovery	8
2.4.4	Extending the Space	9
2.4.5	Information about Database Parameters	10
2.5	Transbase Startup Routines	10
3	Programming Transbase	11
3.1	Making Programs	11
3.2	Handling UTF-8 databases	11
3.3	Connecting to a database	12
3.4	Transaction Control	13
3.5	DDL Statements	14
3.6	DML Statements	14
3.7	SELECT Statements	15
3.8	Parametrized Statements	15
3.9	Other Statements	16
3.10	Complete Programming Example	16

Chapter 1

Overview of Transbase Manuals

The following is a list and a brief description of all Transbase manuals.

Getting Started This manual is a quick introduction into some basic functionality and concepts of Transbase.

System Guide This manual describes the commands and tools for the Transbase Database Administrator and many internal functions and concepts .

SQL Reference Manual Contains the complete TB/SQL kernel language definition.

Transbase Embedded SQL Describes the standardized Embedded SQL programming interface (EXEC SQL ...) in a C language environment.

Programming Interface TBX Describes the native C programming interface to Transbase provided by the library tbx.a and tbx.dll, resp.

TransbaseCD Guide This manual comprises the description of the unique Transbase CD-ROM Databases. Described are the preparation, processing and tuning (e.g. loading and compression) of CD-ROM databases.

Command Control Language CCL An easy-to-learn yet powerful programming language which offers a comfortable SQL interface to the Transbase kernel.

Transbase Interactive Interface TBI Describes the line oriented interactive interface TBI and its table editor reledit.

Transbase Query Optimization and Locking Guide Describes to a far extent how the Transbase query optimizer works and how it can be influenced by commands and additional language features.

UFI User's Guide (only Unix Platforms) Describes the screen oriented interactive interface UFI.

Chapter 2

Interactive Use

2.1 Setting the Environment

Transbase installs into a single directory (called the TRANSBASE directory in the sequel). This directory contains the executables, libraries, include and error message files as well as further information needed by Transbase.

Almost all Transbase executables require the environment variable TRANSBASE to be set to the location of the TRANSBASE directory. The TRANSBASE environment variable should be set either globally by the system administrator or by each user in his shell initialisation.

The most convenient way to access Transbase and its tools is to include the path of the TRANSBASE directory into the PATH setting of your operating system, again either globally or by each user.

In the sequel we assume both settings to be made.

2.2 Creating a Database

Databases are administrated by the tool tbadm (tbadm32 on Windows systems). In particular,

```
tbadmin -cf <dbname>  
tbadm32 -cf <dbname>
```

creates a new database with default parameters. If you want to set some parameters to non-default values, you should use the interactive method:

```
tbadmin -c <dbname>  
tbadm32 -c <dbname>
```

The parameters to be set are discussed briefly below:

The *database type* is standard unless a CD-ROM image of the database shall be created.

Transbase V5.3: The *code page* denotes how character strings are encoded (and transferred between client and server). Selecting SINGLEBYTE code page, usually is the most general setting, unless Multi-Byte-Character have to be stored in the database. In that latter case, use UTF8 code page, but be sure to observe the fact that a character may take more than one byte.

The code page cannot be changed later.

Transbase V5.3: The *locale* setting determines character processing of the database. In particular, the case sensitivity and the mapping to/from upper case are determined by this setting. If the C locale setting is used, only ASCII characters are regarded as characters.

The valid settings for locale are highly machine dependent. Typical values are e.g. C, de_DE, de_DE.utf8, en_US, en_US.utf8.

The locale setting cannot be changed later.

The *page size* influences the performance of the database, the maximum size of a database and also the maximal length of records. Generally, the bigger the database shall be, the larger the page size should be set. The page size never should be smaller than the physical page size of the underlying operating system. *A good choice is 4 kBytes which, however, is not the default for historic reasons.*

A database is mapped to a directory with several subdirectories. All *paths* can be chosen separately if needed. In particular, the persistent data can be split between several disk files which can have completely arbitrary locations (in particular on separate disks), so databases can grow across disks. By default, the database is installed in the current directory.

By default, a database consists of a single *disk file* of at most 102'400 pages. You can specify multiple disk files as well as bigger disk files; however, on most systems a disk file cannot be larger than 2 GBytes.

The size of the *cache area* can be influenced by two parameters: the number and size of shared memories that implement the main memory cache. Default setting is: 2 cache areas of 128 kBytes each. Depending on the physical memory size, this setting should be adapted, e.g. to 2 areas of 2 MBytes each resulting in a total cache of 4 MBytes. The maximum cache size supported is 512 MBytes per cache area and the maximum number of cache areas is 40.

Please note that some operating systems (in particular Solaris systems) restrict the number and size of shared memories significantly by default. In those cases, the operating system limits have to be changed appropriately.

The size of the *sorter cache* should be set larger than default 512 kBytes if you frequently sort large amounts of data. A sorter cache is allocated to each Transbase process dynamically upon sort operations.

The Transbase server processes can be run either dedicatedly for each application or multiplexed. In the latter case, a minimum and maximum number of kernels can be set to values > 0 . A minimum/maximum setting of 2/8 e.g. means that two kernel processes are always present, and dynamically up to 8 processes can be created, which are multiplexed between applications on a transaction level. By default, Transbase servers run dedicatedly.

The *number of users* usually is restricted by the license, but can be further restricted if necessary.

The *disk recovery* log by default is set to off. Setting it on, provides recovery for physically corrupted databases (see below).

The *logging file size* can be set to 0 (default) which means the transactions are before-image-logged. Each page produces a before image page before it is modified. This setting is recommended when loading a database from external files. Setting the logging file size to a value > 0 means that transactions are delta-logged from now. This is recommended for transaction processing, in particular for short transactions that modify only small amounts of data. Note that delta-logging is required if disk recovery logging is set.

Transbase can log *accounting events*, e.g. logins, errors, statements etc. The set of events can be selected - entering a question mark lists the possible settings. In a separate setting the account logging can be switched on and off.

If you finally confirm all settings, the database will be created. Initially there will be one user *tadmin* who has unrestricted rights on the database. You are prompted for a password for this user (an empty password is allowed). The database is automatically booted and can be accessed immediately.

2.3 Accessing the Database interactively

The easiest way to access the database is by tbi (tbi32) by one of the following commands:

```
tbi <dbname>
tbi <dbname> tadmin
tbi <dbname> tadmin ""
tbi32 <dbname>
tbi32 <dbname> tadmin
tbi32 <dbname> tadmin ""
```

Following the prompt you can either directly type SQL commands or you can call an editor (just type e) to type and execute the commands (just type x).

A script to create a table, insert some rows and then commit the transaction would be:

```
create table xyz
( a integer not null,
  b varchar(80) default 'new',
  c date default currentdate
);
insert into xyz values (24, 'transbase', datetime(2000-1-1));
insert into xyz(a,b) values (25, default);
insert into xyz(a,c) values ((select max(a)+1 from xyz), default);
ct;
select * from xyz;
```

Note that in the above example no primary key has been specified for the table xyz. By default, all fields are taken for primary key (in the order of writing).

It is always recommended to select a primary key for Transbase tables; note, however, the semantic and performance implications of the key selection.

2.4 Administrating Databases

All administration commands, including deleting, changing parameters, shutting down etc. are performed by tbadm. tbadm also provides information about database parameters and state.

2.4.1 Changing Database Parameters

Databases are changed by tbadm. The commands to call are:

```
tbadm -a <dbname>
tbadm32 -a <dbname>
tbadm -af <dbname> <modified options>
tbadm32 -af <dbname> <modified options>
```

In the first case, the user is prompted for all changeable parameters, in the latter case options can be specified on the command line directly, e.g.

```
tbadmin -af <dbname> drl=on
tbadm32 -af <dbname> drl=on
tbadmin -af <dbname> lc=1024 nc=8*1024
tbadm32 -af <dbname> lc=1024 nc=8*1024
```

The first command switches disk recovery logging on, the second command sets the local cache size to 1024 kB and the global cache size to 8 areas of 1024 kB each.

Note that the page size of a database cannot be changed after creation. Instead, the database must be archived, deleted and re-created.

2.4.2 Boot and Shutdown

Databases can be accessed only when they are booted. By shutting a database down, access can be prohibited (e.g. to dump a database, to install a new version etc.). The commands

```
tbadmin -b <dbname>
tbadm32 -b <dbname>
tbadmin -s <dbname>
tbadm32 -s <dbname>
tbadmin -sf <dbname>
tbadm32 -sf <dbname>
```

boot a database, shutdown a database and perform a "forced shutdown" by which running transactions are aborted.

2.4.3 Logging and Transaction Recovery

As mentioned, Transbase provides two different transaction recovery methods: Before-Image-Logging and Delta-Logging. In the first case, before image pages are written before a page is modified. Upon commit, modified pages are written to the database persistently and before image pages are deleted. Upon abort, before image pages are reinstalled into the database persistently. Before-image-logging requires a lot of disk I/O and a lot of synchronisation (a page must not be written to disk before its before image has been safely written) which can be achieved only by expensive disk operations (fsync, sync).

In the other case, page modifications are logged incrementally on a byte basis into the log file. Upon commit, the log file records all changes as well as the commit operation safely on disk. Upon abort, the log file has to be processed reversely to rollback any changes made to the database so far, and the abort operation has

to logged safely to disk. So, the log files are an immanent part of each database, only *disk files plus log files* make up a consistent database state.

The disk files are written safely to disk periodically. Thereby the log files are shortened (from the beginning). Some of the log files (called archive log files) are no longer necessary for building a consistent database state, since their modifications are kept safely in the disk files. Archive log files, by default, are deleted by Transbase, in order to keep the disk usage of log files low.

If, however, disk recovery logging is enabled, those archive log files are kept and renamed (the extension changes from .act to .arc), and the system administrator is responsible to move those .arc files to a safe backup medium. Together with an initial state of the database, these archive log files can be used to redo any changes on the database in case some disk failure has happened.

2.4.4 Extending the Space

After creation, databases are limited to a maximum number of pages. However, when a database grows further, it can be extended by disk files (whose locations can be set independently, so that databases can be split across file systems or disks). The command

```
tbadmin -af <dbname> dx=/db/part2,409600
tbadm32 -af <dbname> dx=c:\db\part2,409600
```

would extend the database by one disk file located under /db/part2 with 409'600 pages. The following command

```
tbadmin -af <dbname> \
  dx=/db2/space,409600,,pinit \
  dx=/db3/space,409600,,pinit
tbadm32 -af <dbname> \
  dx=c:\dbpart2,409600,,pinit \
  dx=d:\dbpart3,409600,,pinit
```

extends the database by two disk files of 409'600 pages each which are physically initialised. Note that initialising a disk file may take a while, depending on the size of file. However, it guarantees that the space actually is allocated by Transbase. Without initialising a disk file, pages would be allocated dynamically on demand, and it could happen that Transbase finds the disk full.

2.4.5 Information about Database Parameters

The command

```
tbadmin -i <dbname>  
tbadm32 -i <dbname>
```

provides all parameter settings about a database.

The commands

```
tbadmin -i <dbname> connections  
tbadm32 -i <dbname> connections  
tbadmin -i <dbname> space  
tbadm32 -i <dbname> space
```

provide information about currently active connections or the currently usage space in the database.

2.5 Transbase Startup Routines

Under UNIX an executable shell script file `rc.Transbase` is installed in the `TRANSBASE` directory which should be executed at machine boot time. It starts all necessary processes and boots all shared memories for all databases. Reversely, a script file `rc.tbstop` is installed in the `TRANSBASE` directory which shuts down all databases and terminates all `TRANSBASE` processes. This script should be executed upon machine shutdown.

Under WindowsNT Transbase is installed as a "service" which is automatically started when the machine boots or it is installed as a linked-in DLL which is automatically loaded when Transbase is executed. Shared Memories are installed dynamically upon the first `CONNECT` to a database.

Chapter 3

Programming Transbase

3.1 Making Programs

C-programs need to include the file `tbx.h` (which located in the `TRANSBASE` directory). This file contains the data structures, constants as well as the function prototypes.

If the program wants to decode specific error situations by symbolic names, inclusion of `tberror.h` is necessary, too.

The compiler needs to be directed to find the include files in the appropriate directories, usually by the `I` option.

In order to link the program, the library `tbx.a` has to be included. Depending on the operating system, also some standard libraries have to be included. Find below a `cc` comand line to compile and link two source modules:

```
cc -I$TRANSBASE main.c db.c $TRANSBASE/tbx.a -lm -ltcp -ldes
```

If you are using makefiles the following entries automate program generation:

```
.c.o:  
    CFLAGS=$(CFLAGS) -I$(TRANSBASE)  
    LFLAGS=$(LFLAGS) $(TRANSBASE)/tbx.a -lm -ltcp -ldes
```

3.2 Handling UTF-8 databases

Transbase V5.3: For UTF-8 databases, all TBX parameters (except the database name) that refer to strings (`char*`) have to be interpreted as UTF-8 coded. UTF-8

can be either converted into wide character strings (UNICODE) or into machine-dependent single or multi byte character strings.

The conversion from UTF-8 (`unsigned char * src`) to UCS-2 wide characters (`wchar_t *dest`) is given by the following code fragment:

```
while (*src) {
    if ((*src & 0x80) == 0)
        *dest++ = *src;
    else if ((*src & 0xC0) == 0xC0) { // 2 Bytes
        *dest++ = ((*src & 0x1f) << 6) | (*(src+1) & 0x3f);
    } else if ((*src & 0xE0) == 0xE0) { // 3 Bytes
        *dest++ = ((*src & 0x0f) << 12) | (*(src+1) & 0x3f)<<6 | (*(src+2) & 0x3f);
    }
    ++src;
}
*dest = 0;
```

The conversion from UCS-2 wide characters (`wchar_t *src`) to UTF-8 (`unsigned char * dest`) is given by the following code fragment:

```
while (*src) {
    if (*src < 128)
        *dest++ = *src;
    else if (*src < 1<<12) {
        *dest++ = 0xC0 | (*src>>6);
        *dest++ = 0x80 | (*src & 0x3f);
    } else if (*src < 1<<17) {
        *dest++ = 0xE0 | (*src>>12);
        *dest++ = 0x80 | (*src>>6) & 0x3f;
        *dest++ = 0x80 | (*src) & 0x3f;
    }
    ++src;
}
*dest = 0;
```

3.3 Connecting to a database

The following code fragment connects to a single database, authorises and disconnects from a database:

```
static Id dbid;
```

```
static State dbstate;
rc = TbxConnect(dbname, &dbid);
rc = TbxLogin(dbid, "jones", "tiger");
rc = TbxDisconnect(dbid, &dbstate);
```

Although Transbase supports up to 10 concurrent database connections, in most cases a single database connection is sufficient whose id could be held globally in the program.

Transbase V5.3: Please note that for UTF-8 databases all (char*) parameters have to be interpreted as UTF-8 coded. The only exception here is the database name which is restricted to be pure ASCII. After a database connection is made, the code page (determining the encoding) can be queried as follows:

```
static Id dbid;
static short cp;

rc = TbxConnect(dbname, &dbid);
rc = TbxCodePage(dbid, &cp);

if (cp == CPG_UTF8) ...
```

3.4 Transaction Control

The following code fragment shows how to control database transactions. Note that databases and transactions are kept completely separate in Transbase, i.e. a transaction can span multiple databases (distributed transaction) or a separate transaction can be allocated for each database. However, on a single database at most one transaction can be active at a time.

```
Static Id taid;
Static State tastate;
rc = TbxBt(&taid);
rc = TbxCt(taid, &tastate);
if (rc || tastate != TA_COMMITTED)
    rc = TbxAt(taid, &tastate);
```

Since most programs are restricted to a single database, they also are restricted to a single transaction whose id could be held globally in the program.

The following procedure shows an automatic BT operation that starts a new transaction only when no transaction is active:

```

{
    int rc = TbxGetTaState(taid, &tastate);
    if (rc == 0 && tastate == TA_ACTIVE) ; // TA already active
    else rc = TbxBt(&taid);
}

```

3.5 DDL Statements

The first SQL statement to execute usually is a DDL (Data Definition Language) statement, e.g. to create a new table or an index. The following fragment shows the interaction with tbx:

```

{
    char * ddl1 = "create table ADR \
        (ANO integer, NAME char(*), DATUM datetime[yy:dd], , \
        primary key (ANO))";
    char * ddl2 = "create index XNAME on ADR(NAME)";

    int rc = TbxRun(dbid, taid, ddl1, &qd, NULL);
    int rc = TbxRun(dbid, taid, ddl2, &qd, NULL);
}

```

Note that DDL statements underly the usual transaction recovery and no automatic COMMIT or ABORT is executed by Transbase.

3.6 DML Statements

Usually, data insertion is the next step to take place. The following code fragment inserts a record into ADR and checks it is really inserted:

```

{
    Query_descr qd;
    Result res;
    char * ins = "insert into ADR(ANO, NAME, DATUM) values \
        ( 1, TransAction Software, currentdate )";

    int rc = TbxRun(dbid, taid, ins, &qd, &res);
    if (rc == 0 && res._var.count.ntuples == 1) // ok
    else // not ok
}

```

3.7 SELECT Statements

By far the most frequent operation is to retrieve data from tables, via SELECT statements which is shown below:

```
{
    Query_descr qd;
    char * sel = "select ANO, NAME, DATUM from ADR WHERE ANO=15";

    int rc = TbxDml(dbid, taid, sel, &qd);
    while (rc==0 && qd.eod != NO_TUPLE) {
        rc = TbxEval(&qd, NULL);
        if (rc == 0 && qd.eod != NO_TUPLE) {
            Integer *ano = (Integer *) qd.field[0].fieldpointer;
            char *name = (char *) qd.field[1].fieldpointer;
            Datetime *dt = (Datetime *) (qd.field[2].fieldpointer);
            char dtstring[24];
            tb_dt_dt (dtstring, ISO, dt);
            printf("%3d %-20s %s\n", *ano, name, dtstring);
        }
    }
}
```

Transbase V5.3: Please note that fieldpointers of type CHAR have to be converted from UTF-8 on UTF-8 databases.

3.8 Parametrized Statements

DML statements also can be parametrized so that the statement can be executed several times with different parameters. The parameters usually are bound to program variables as shown in the following program fragment:

```
static Integer ano; // variable bound to ANO
static char name[80]; // variable bound to NAME

static Parameters p;

{
    Id sid;
    Query_descr qd;
    Result res;
    int rc;
```

```

rc = TbxStore(dbid,
    "insert into ADR(ANO, NAME, DATUM) values (?, ?, currentdate)",
    &sid, &qd);
p.param_no = 2;
p.param = (Param *) calloc(2, sizeof(Param));
p.param[0].type = TB__INTEGER;
p.param[0].value = (char*) &ano;
p.param[1].type = TB__CHAR;
p.param[1].value = (char*) name;
ano = 0;
while (1) {
    ++ano;
    if (gets(name) != name) break;
    rc = TbxRunStored(dbid, taid, sid, &p, &qd, &res);
    if (rc || res._var.count.ntuples != 1) break;
}
TbxDropStored(dbid, sid);
}

```

Transbase V5.3: Please note that parameters of type CHAR have to be converted into UTF-8 on UTF-8 databases.

3.9 Other Statements

Other statements include e.g. local settings or Transbase server settings. The following call e.g. set the timeout period to 2 seconds. Thereby interactive queries wait at most 2 seconds for a lock:

```
int rc = TbxSetTimeout(2L);
```

3.10 Complete Programming Example

```

#include <stdio.h>
#include <stdlib.h>
#include "tbx.h"

static Id dbid, taid;
static State dbstate, tastate;
static char * dbname = "sample";
static char * ddl1 =

```

```

    "create table ADR \
      (ANO integer, NAME char(*), DATUM datetime[yy:ss], \
       primary key (ANO))";
static char *ddl2 =
    "create index XNAME on ADR(NAME)";
static char *ins =
    "insert into ADR(ANO, NAME, DATUM) values (?, ?, currentdate)";

static char *selmax =
    "select max(ANO) from ADR";
static char *seladr =
    "select ANO, NAME, DATUM from ADR";

static int tberr (int rc)
{
    if (rc) {
        fprintf(stderr, "Transbase Error %d:\n%s\n", rc, tb_errtxt);
        exit(rc);
    }
    return rc;
}

static int doinsert()
{
    Id sid;
    Parameters p;
    Query_descr qd;
    int rc;
    Integer ano = 0;
    char name[80];

    TbxBt(&taid);
    tberr(TbxRun(dbid, taid, selmax, &qd, NULL));
    if (qd.field[0].fieldpointer)
        ano = *(Integer*) qd.field[0].fieldpointer;

    tberr(TbxStore(dbid, ins, &sid, &qd));
    p.param_no = 2;
    p.param = (Param *) calloc(2, sizeof(Param));
    p.param[0].type = TB__INTEGER;
    p.param[0].value = (char*) &ano;
    p.param[1].type = TB__CHAR;
    p.param[1].value = (char*) name;
    while (1) {

```

```

        ++ano;
        printf("Ano %ld : ", ano);
        if (gets(name) != name || strlen(name)==0) break;
        tberr(TbxRunStored(dbid, taid, sid, &p, &qd, NULL));
    }
    rc = tberr(TbxCt(taid, &tastate));
    free(p.param);
    TbxDropStored(dbid, sid);
    return rc;
}

static int showtable()
{
    Query_descr qd;
    Integer *ano;
    char *name;
    char dtstring[24];

    TbxBt(&taid);
    tberr(TbxDml(dbid, taid, seladr, &qd));
    while (1) {
        tberr(TbxEval(&qd, NULL));
        if (qd.eod==NO_TUPLE) break;
        ano = (Integer*) qd.field[0].fieldpointer;
        name = (char*) qd.field[1].fieldpointer;
        tb_dt_dt (dtstring, ISO, (Datetime*) qd.field[2].fieldpointer);
        printf("%3d %-40s %s\n", *ano, name, dtstring);
    }
    TbxClose(&qd);
    return tberr(TbxAt(taid, &tastate));
}

int main()
{
    Query_descr qd;
    tberr(TbxConnect(dbname, &dbid));
    tberr(TbxLogin(dbid, "tbadadmin", ""));
    tberr(TbxBt(&taid));
    TbxRun(dbid, taid, ddl1, &qd, NULL);
    TbxRun(dbid, taid, ddl2, &qd, NULL);
    tberr(TbxCt(taid, &tastate));

    doinsert();
    showtable();
}

```

```
    tberr(TbxDisconnect(dbid, &dbstate));  
    return 0;  
}
```