

Transbase[®]
PHP Module

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	Introduction	3
1.1	PHP	3
1.2	Introductory Example	3
1.3	Features of Transbase/PHP	4
2	Transbase Functions	6
2.1	Error Handling	6
2.2	Database Functions ("trans_db_")	6
2.2.1	trans_db_open	6
2.2.2	trans_db_close	7
2.2.3	trans_db_state	7
2.3	Transaction Functions ("trans_ta_")	7
2.3.1	trans_ta_open	7
2.3.2	trans_ta_commit	8
2.3.3	trans_ta_abort	8
2.3.4	trans_ta_state	8
2.4	SQL Functions ("trans_sql_")	8
2.4.1	trans_sql_exec	9
2.4.2	trans_sql_open	9
2.4.3	trans_sql_fetch	9
2.4.4	trans_sql_fetch_row	10
2.4.5	trans_sql_fetch_array	10
2.4.6	trans_sql_close	11

2.4.7	trans_sql_state	11
2.5	Field Functions ("trans_field")	11
2.5.1	trans_field_count	11
2.5.2	trans_field_name	12
2.5.3	trans_field_type	12
2.5.4	trans_field_isnull	12
2.5.5	trans_field_string	12
2.5.6	trans_field_value	13
2.6	Global Interface Functions ("trans_tbx")	13
2.6.1	trans_tbx_timeout	13
2.6.2	trans_tbx_isolation	13
3	PHP System Architecture	15
4	Current Limitations	17
4.1	Features not available at PHP interface	17

Chapter 1

Introduction

1.1 PHP

PHP is a popular server-side script processor which allows dynamic Web page generation. In addition to several built-in modules PHP also provides a concept to dynamically load extensions, e.g. to access SQL databases.

The following simple example demonstrates the power of PHP most evidently.

1.2 Introductory Example

The following HTML code shows a heading followed by a list of newsletter subjects directly generated by a Transbase SQL query.

```
<html>
<head>
  <title>Demonstration of Transbase PHP</title>
</head>

<body>

<h1>Newsletters of TransAction Software</h1>
<?php
  $dbid = trans_db_open ( "tas@aix", "web", "web" );
  if ( $dbid >= 0 )
  {
    $quid = trans_sql_open (
      "select nr, date, subject from newsletter" );
    if ( $quid >= 0 ) {
```

```

        while ( trans_sql_fetch () > 0 ) {
            printf ( "%s: %s <BR>\n",
                    trans_field_string(1),
                    trans_field_string(2)
                );
        }
        trans_sql_close ();
    }
    trans_db_close ();
}
else
{
    echo trans_tbx_error ();
}
?>
</body>
</html>

```

1.3 Features of Transbase/PHP

Transbase/PHP is a specialisation of the TBX programming interface which provides the full functionality of Transbase to C programmers. For more detail, please refer to other manuals of Transbase, in particular:

- Transbase SQL reference manual
- Transbase TBX reference manual
- Transbase system guide

Generally speaking, Transbase handles three classes of objects:

- A *database* is a collection of tables which can be queried and/or modified. A database usually is accessed through a TCP/IP socket. To open a connection, a user identification and a password have to be specified. A database is referenced by a database identifier in the range [0 .. MAXDB].
- A *transaction* is a unit of actions performed atomically by a database. A transaction can be committed (all changes are made persistent and visible to other transactions) or aborted (all changes are rolled back as if the transaction had not been executed). Although a transaction usually is processed on a single database, it can also be distributed over several databases (in which case a so-called two-phase commit has to be used). A transaction is referenced by a transaction identifier in the range [0 .. MAXTA].

- A *query* is a set of rows retrieved from the database. A query is formulated by a SQL SELECT statement and processed by the selected database within the selected transaction. At the programming level, a query is retrieved row by row. A query is referenced by a query identifier in the range [0 .. MAXQU].

Chapter 2

Transbase Functions

2.1 Error Handling

Transbase PHP functions principally return a numeric value. Negative values correspond to error situations while non-negative values mean successful completion of a request with additional information. E.g. a (non-negative) database identifier is returned on success.

In case of errors, a textual description of the error can be retrieved by a further call.

2.2 Database Functions (”trans_db_”)

These functions control access to databases, authorization and provide status inquiries.

Databases are referenced by database identifiers. Database identifiers can be omitted. If so, the previously used database identifier is used.

In particular, when only one database is being used, database identifiers are never necessary.

2.2.1 trans_db_open

```
int = trans_db_open( string dbname, string username, string password )
```

trans_db_open establishes a connection to a database. The database must exist and be booted (i.e. accessible). It is referenced by its name. For security, a program must authorize through its username (which determines the access rights to the database) and an optional password. If no password is supplied, the empty string is assumed.

All subsequent calls use the database identifier returned by a successful connect.

2.2.2 trans_db_close

```
int = trans_db_close ( int dbid )
```

The database referenced by dbid is closed. dbid can be omitted if only one database is active. On success, the function returns the database status (see below).

As a side effect, the active transaction is aborted.

2.2.3 trans_db_state

```
int = trans_db_state ( int dbid )
```

The connection status for the database referenced by dbid is retrieved. It can be:

DB_CONN	connection is active
DB_LOGGED	connection is active
DB_DCONN	connection is inactive
DB_KILLED	connection is inactive

2.3 Transaction Functions ("trans_ta")

These functions control transactions run on databases. In the current implementation, only one transaction may be active at a time. For future extensions, however, transactions are referenced by transaction identifiers.

Transaction identifiers can be omitted. If so, the previously used transaction identifier is used.

In particular, when only one transaction is being used, transaction identifiers are never necessary.

Since a transaction is automatically opened when needed and an active transaction is aborted upon trans_db_close, the programmer has to care about committing transactions only.

2.3.1 trans_ta_open

```
int = trans_ta_open ( )
```

trans_ta_open opens a new transaction. A transaction must be open before statements or queries can be processed on a database.

On success, the function returns a transaction identifier that is to be used for subsequent calls to reference to this transaction.

2.3.2 trans_ta_commit

```
int = trans_ta_commit ( int taid )
```

The transaction referenced by taid is committed and all its changes are made persistent and visible to other transactions. taid can be omitted if only one transaction is active. As a side effect, all queries opened in this transaction are closed, and all locks acquired by this transaction are released.

On success, the status of the transaction is returned (see below).

Note that due to locking, a transaction may be forced to wait for commit.

2.3.3 trans_ta_abort

```
int = trans_ta_abort ( int taid )
```

The transaction referenced by taid is rolled back and all its changes are made undone as if the transaction never had started. taid can be omitted if only one transaction is active. As a side effect, all queries opened in this transaction are closed, and all locks acquired by this transaction are released.

On success, the status of the transaction is returned (see below).

2.3.4 trans_ta_state

```
int = trans_ta_state ( int taid )
```

The status of the transaction referenced by taid is returned. taid can be omitted if only one transaction is active.

The state is encoded as follows:

TA_ACTIVE	TA is active
TA_COMMITTED	TA is successfully committed (not active)
TA_ABORTED	TA is aborted (not active)
TA_UNDEF	state of TA cannot be determined

2.4 SQL Functions ("trans_sql")

This class of functions controls the execution of queries on databases. Queries only can be run within a transaction that must be opened before.

The function trans_sql_open is intended for SELECT queries that retrieve a set of result tuples which is referenced by a non-negative query identifier.

Query identifiers can be omitted. If so, the previously used query identifier is used.

The function `trans_sql_exec` is intended for all other SQL queries, in particular for INSERT, UPDATE, DELETE queries and for all data definition statements.

2.4.1 `trans_sql_exec`

```
int = trans_sql_exec ( string query, int dbid, int taid )
```

`trans_sql_exec` executes INSERT, UPDATE, DELETE statements on the database referenced by `dbid` within the transaction referenced by `taid`. On success, the (non-negative) number of tuples inserted, updated, deleted is returned.

`trans_sql_exec` executes also DDL statements (CREATE TABLE, DROP INDEX etc.) on the database referenced by `dbid` within the transaction referenced by `taid`. On success, 0 is returned.

`dbid` can be omitted if only one database is active. `taid` can be omitted if only one transaction is active. A transaction is automatically opened when necessary.

2.4.2 `trans_sql_open`

```
int = trans_sql_open ( string query, int dbid, int taid )
```

`trans_sql_open` opens a SELECT query on the database referenced by `dbid` within the transaction referenced by `taid`.

`dbid` can be omitted if only one database is active. `taid` can be omitted if only one transaction is active. A transaction is automatically opened when necessary.

On success, the function returns a (non-negative) query identifier which is used to reference this query subsequently.

2.4.3 `trans_sql_fetch`

```
int = trans_sql_fetch ( int quid )
```

The next result row of the query referenced by `quid` is retrieved. `quid` can be omitted if only one query is active.

`trans_sql_fetch` returns zero when the result set is exhausted, otherwise a positive value.

The following program fragment shows the evaluation of a query using simple fetch calls.

```

$quid = trans_sql_open ("select nr, date, subject from newsletter" );
if ( $quid >= 0 ) {
    while ( trans_sql_fetch () > 0 ) {
        printf ( "%s: %s <BR>\n",
                trans_field_string(1),
                trans_field_string(2)
                );
    }
    trans_sql_close ();
}

```

2.4.4 trans_sql_fetch_row

```
int = trans_sql_fetch_row ( indexed_array row, int quid )
```

The next result row of the query referenced by quid is retrieved into the indexed array row. The array is indexed by column position, starting from 0. quid can be omitted if only one query is active.

trans_sql_fetch_row returns zero when the result set is exhausted, otherwise a positive value.

The following program fragment shows the evaluation of a query using arrays.

```

$quid = trans_sql_open ("select nr, date, subject from newsletter" );
if ( $quid >= 0 ) {
    while ( trans_sql_fetch_row (& $row) > 0 ) {
        printf ( "%s: %s <BR>\n",
                $row[1],
                $row[2]
                );
    }
    trans_sql_close ();
}

```

2.4.5 trans_sql_fetch_array

```
int = trans_sql_fetch_array ( associative_array row, int quid )
```

The next result row of the query referenced by quid is retrieved into the associative array row. The array is indexed by column names. quid can be omitted if only one query is active.

trans_sql_fetch_row returns zero when the result set is exhausted, otherwise a positive value.

The following program fragment shows the evaluation of a query using associative arrays.

```
$quid = trans_sql_open ("select nr, date, subject from newsletter" );
if ( $quid >= 0 ) {
    while ( trans_sql_fetch_array (& $row) > 0 ) {
        printf ( "%s: %s <BR>\n",
                $row["date"],
                $row["subject"]
        );
    }
    trans_sql_close ();
}
```

2.4.6 trans_sql_close

```
int = trans_sql_close ( int quid )
```

The query referenced by quid is closed. quid can be omitted if only one query is active.

2.4.7 trans_sql_state

```
int = trans_sql_state ( int quid )
```

The status of the query referenced by quid is returned. quid can be omitted if only one query is active.

The state is encoded as follows:

QU_ACTIVE	query is active
QU_NOT_ACTIVE	query is not active

2.5 Field Functions (“trans_field”)

Field functions provide information about fields of query results. In particular, they provide flexible access to field values.

2.5.1 trans_field_count

```
int = trans_field_count ( int quid )
```

trans_field_count returns the number of columns for the query referenced by quid. quid can be omitted if only one query is active.

2.5.2 trans_field_name

```
string = trans_field_name ( int pos, int quid )
```

This function returns the name of the column indexed by pos in the query referenced by quid. pos ranges from 0 to trans_field_count () - 1 and is a mandatory parameter. quid can be omitted if only one query is active.

2.5.3 trans_field_type

```
int = trans_field_type ( int pos, int quid )
```

This function returns the type of the column indexed by pos in the query referenced by quid. pos ranges from 0 to trans_field_count () - 1 and is a mandatory parameter. quid can be omitted if only one query is active.

TB__TINYINT	1 byte integer
TB__SMALLINT	2 byte integer
TB__INTEGER	4 byte integer
TB__NUMERIC	numeric value with fixed precision and scale
TB__FLOAT	4 byte numeric value with variable scale
TB__DOUBLE	8 byte numeric value with variable scale
TB__CHAR	variable or fixed-size string
TB__DATETIME	date, time or timestamp
TB__TIMESPAN	date or time intervals
TB__BOOL	boolean values (true or false)
TB__BINCHAR	variable or fixed-size data of arbitrary type
TB__BLOB	binary large object of unlimited size
TB__BITSS	bit vectors of variable or fixed size

For details of Transbase data types see the TBX manual (Query_descr).

2.5.4 trans_field_isnull

```
int = trans_field_isnull ( int pos, int quid )
```

This function returns TRUE if the column indexed by pos in the query referenced by quid is NULL, otherwise FALSE. pos ranges from 0 to trans_field_count () - 1 and is a mandatory parameter. quid can be omitted if only one query is active.

2.5.5 trans_field_string

```
string = trans_field_string ( int pos, int quid )
```

This function returns the string converted value of the column indexed by `pos` in the query referenced by `quid`. `pos` ranges from 0 to `trans_field_count () - 1` and is a mandatory parameter. `quid` can be omitted if only one query is active. Null values return the empty string.

2.5.6 `trans_field_value`

```
mixed = trans_field_value ( int pos, int quid )
```

This function returns the value of the column indexed by `pos` in the query referenced by `quid`. `pos` ranges from 0 to `trans_field_count () - 1` and is a mandatory parameter. `quid` can be omitted if only one query is active.

2.6 Global Interface Functions ("trans_tbx_")

2.6.1 `trans_tbx_timeout`

```
int = trans_tbx_timeout ( int timeout )
```

This function sets the global timeout for the database session to `timeout` seconds. This setting will be valid from the next `trans_ta_open` call.

The timeout determines how long a query may wait for locked database objects; a value of 0 means queries may wait infinitely. The default setting is 60 seconds. For interactive database access (like in Web servers) a shorter timeout value of 3 seconds might be more recommendable.

On success, the previous setting (a non-negative value) is returned.

2.6.2 `trans_tbx_isolation`

```
int = trans_tbx_isolation ( int isolation )
```

This function sets the global isolation level for reading transactions.

On success, the previous setting (a non-negative value) is returned.

Please note that transactions always are raised to level 3 once they modify the database, independently of their isolation level. This setting will be valid from the next `trans_ta_open` call.

The following table shows the encoding of isolation levels:

3	serializability
2	cursor stability
1	

In level 3 (serializability) all read locks are held until the end of a transaction. Read reproducibility is achieved.

In level 2 (cursor stability) all read locks are automatically released when a query is closed. If the same query is re-executed, different results may be seen.

In level 1 reading transactions do not request locks at all. They do not return a consistent state of the database if concurrent update transactions are active on the tables referenced by the SELECT statement.

By default, Transbase PHP starts in level 3 (serializability).

Chapter 3

PHP System Architecture

PHP4 usually is linked into Web servers dynamically. On Windows based systems this is accomplished by a so-called DLL (file extension `.dll`), on UNIX based systems by a so-called shared library (file extension `.so`). Some systems however, prefer to compile and link their Web server statically.

The Transbase PHP module is provided as a shared library, too. This shared library either has to be linked statically or dynamically into the PHP library.

Depending on the linking policy we have the following situations with increasing performance, but decreasing flexibility (shown for a Linux system):

- `httpd` dynamically links `libphp4.so` which in turn dynamically links `libtransbase.so`
- `httpd` dynamically link `libphp4.so` which has been statically linked with `libtransbase.la`
- `httpd` is statically recompiled and linked with both `libphp4.la` and `libtransbase.la`

In case of dynamic linking, the libraries can be loaded either when PHP4 is started or explicitly on demand by the following php statement:

```
d1 ( "libtransbase.so" );
```

Independently of static or dynamic linking, please note that Transbase requires some environment variables be set for the process by which Transbase is called. In case of PHP this is typically the `apache` process.

There are several ways how to provide `apache` with the correct settings, depending on the operating system. For Suse-Linux as shown below:

- The environment could be set within the `apache` start script.

```
export TRANSBASE=/opt/transbase
export TRANSBASE_SERVICENAMES=2024:2025
...
/usr/sbin/http ...
```

- The environment could be set before calling the `apache` start script. This would require to identify potentially many places where `apache` is being started.

```
export TRANSBASE=/opt/transbase
export TRANSBASE_SERVICENAMES=2024:2025
/etc/rc.d/apache start
```

- The environment could be set globally at the end of the file `/etc/rc.config` which provides the setting effectively for all start procedures (not just `apache`).

```
export TRANSBASE=/opt/transbase
export TRANSBASE_SERVICENAMES=2024:2025
```

Chapter 4

Current Limitations

4.1 Features not available at PHP interface

The following is a list of features that is missing as opposed to TBX interface:

- Currently, so-called prepared statements cannot be stored or executed. Since PHP only has a loose type concept, it seems sufficient to support dynamic queries only. Also, since the lifetime of prepared statements is restricted by the connection lifetime, it seems to be no significant performance loss.
- So-called scrollable cursors are not supported. Those cursors allow to position the cursor within a result set on arbitrary relative or absolute positions. They also allow to determine the number of rows in a result set without the need to fetch all rows.
- So-called updatable cursors ("SELECT ... FOR UPDATE") are not supported. Those cursors allow to update or delete the last tuple fetched. Instead, a searched update/delete statement can be used. Note however, that any cursors on the corresponding table must be closed before such statements are accepted.