



**VORTEXcli**  
**(Call Level Interface)**  
**Reference Manual**

March 16, 2026

Trifox Inc.  
[www.trifox.com](http://www.trifox.com)



---

## Trademarks

TRIMapp, TRImpl, TRIMqmr, TRIMreport, TRIMtools, GENESISsql, DesignVision, DVapp, DVreport, VORTEX, VORTEXcli, VORTEXc, VORTEXcobol, VORTEXperl, VORTEXjdbc, VORTEX++, VORTEXJava Edition, LIST Manager, VORTEXodbc, VORTEXnet, VORTEXclient/server, VORTEXaccelerator, VORTEXreplicator are all trademarks of Trifox, Inc.

All other brand and product names are trademarks or registered trademarks of their respective owners.

## Copyright

The information contained in this document is subject to change without notice and does not represent a commitment by Trifox Inc. The software described in this document is furnished under a license agreement and may be used or copied only in accordance with the terms of the agreement. No part of this manual or software may be reproduced or transmitted in any form or by any means, electronic or mechanical (including photocopying and recording), or transferred to information storage and retrieval systems without the written permission of Trifox Inc.

Copyright © Trifox Inc. 1986-2022

All rights reserved.

Printed in the U.S.A.



# Contents

---

---

## **Preface iii**

Revisions v

## **1 Overview**

Structures 1

Datatypes 2

Error Handling 4

## **2 Database Function Calls**

VTXBLOB 6

VTXCAN 7

VTXCLOS 8

VTXCMD 9

VTXCOMM 13

VTXCONN 14

VTXDES2 15

VTXDES3 16

VTXDES4 17

VTXDESC 18

VTXEXEC 19

VTXEXIO 20

VTXINIT 21

VTXIPC 22

VTXLOBG 23

VTXLOBP 24

VTXMOVE 25

VTXOPEN 27

VTXREL 28

VTXROLL 29

## **3 Utility Routines**

VTXEMAP 30

VTXGDID 31

VTXGEEM 32

VTXGLEN 33

VTXOPTS 33

## **4 Conversion Routines**

TCVCHFM 35

TCVD2I4 35

TCVD2N 36

TCVD2S 36

TCVDD2N 37

TCVDINI 37

TCVF42N 38

TCVF82N 39

TCVI42D 39

---

TCVI42N	39
TCVI82N	40
TCVN2DD	40
TCVN2D	41
TCVN2F4	41
TCVN2F8	42
TCVN2FS	42
TCVN2I4	43
TCVN2I8	43
TCVN2PD	44
TCVN2S	44
TCVN2U4	45
TCVN2U8	45
TCVN2ZD	46
TCVPD2N	46
TCVS2D	47
TCVS2IB	47
TCVS2N	48
TCVU42N	49
TCVU82N	49
TCVUNIC	49
TCVZD2N	50

#### **Appendix A Include Files 51**

SQLCA	51
SQLDA	52
VORTEX.H	53
VORTEX.X	58



# Preface

---

---

## Background

Trifox Inc. has been serving the relational database market since 1984 through consulting and the development of software products. In 1987, Trifox created SQL\*QMX for Oracle. This easy-to-use, powerful querying and report writing tool, which is based on IBM's QMF, continues to be used at thousands of sites. In 1989, Trifox created TRIMtools, a family of application and reportwriting tools now known as DesignVision. DesignVision was developed in response to the OLTP requirements of several large application vendors.

## Database Access

VORTEX is an integrated family of products that allows nearly any production application to access SQL data:

- On any or all of the major relational databases.
- Across networks.
- Across platforms.
- With a dramatic increase in the number of concurrent users.
- Without any additional hardware.

In a client/server or multi-tier configuration, VORTEX makes it possible for your SQL applications to access data on different platforms over one or more network configurations. Currently it supports only TCP/IP.

Inherent in this approach are services that allow production applications originally written for one relational database (such as Oracle) can access the same data on another database (such as Informix), even if it is spread across different databases.

VORTEX Precompilers for C and COBOL, as well as a variety of program interfaces, allow existing SQL programs to take full advantage of VORTEX services such as performance enhancement, transaction monitoring, and flat-file database access.

With VORTEXaccelerator in your configuration, you dramatically increase the number of concurrent users who can log on to a specific SQL production application. Your users experience faster performance and you won't have to change any programs or add any hardware.

## Application and Report Development

DesignVision DVapp lets you design, generate, and maintain forms-based applications. You can easily port the pop-up windows, customizable menus and submenus, and

---

custom keyboard assignments, in fact the entire application, to Windows .NET, Unix, OpenVMS, or HTML5 with no extra effort.

The reportwriter, TRIMreport, lets you create simple reports quickly, or complex reports with absolute confidence in their power.

When you want to write stand-alone applications (including triggers) without a user interface, the TRIMpl 4GL language gives you the freedom you want. The procedural language has over 100 database-specific functions that help you write powerful applications in very little time.

## Reaching Legacy Data

GENESISsql is a SQL processor that accesses low-level data sources such as ISAM, SDMS, ADABAS, RMS, and MicroFocus and makes the data accessible to VORTEX clients. You can add GENESIS data sources to a VORTEX system in a matter of days, simplifying what used to be an enormous task.

## Conventions

Screen shots in this manual come from the Windows version of our software.

Trifox documentation uses the following conventions for communicating information:

Example	Describes
CHOOSE REPORT > [F3] >	Press [F3] on the CHOOSE REPORT menu and ...
Right-click	Clicking the right mouse button.
Left-click	Clicking the left mouse button.
<i>connect_string</i>	Replace italicized text with your own variable.
<b>vtxnetd</b>	Text in bold typewriter style represents strings that you type exactly as they appear in the manual.

## Support

If you have a question about a TRIFOX product that is not answered in the documentation (paper or online), contact the Customer Support Services group at:

- support@trifox.com
- Trifox Customer Support Services  
2959 Winchester Boulevard  
Campbell, CA 95008  
U.S.A.
- 408-796-1590

## Revisions

### *August 1999*

Added two command codes to VTXCMD.

### *May 2005*

Added VTXDES3 and scrollable cursor information.

### *December 2007*

Added VTXGLEN.

### *January 2008*

Updated VTXLOBG.

### *October 2009*

Added VTXDES4.

### *December 2014*

Added two command codes to VTXCMD.

### *April 2015*

Removed erroneous VTXMOVE parameter. Added DB2 stored procedure cursor result processing to VTXEXIO.

### *August 2015*

Added SSL configuration command to VTXCMD.

### *June 2018*

Added BLK\_MIN, MAX\_COL\_LEN, SET\_CON\_ATTR command codes to VTXCMD.

### *July 2018*

Added TDB\_CMD\_GET\_CON\_ATTR command codes to VTXCMD.

### *March 2026*

Added DESC\_LOBS, DESC\_NULLS, DESC\_PARMS command codes to VTXCMD.



# Chapter 1

## Overview

---

---

VORTEXcli is a thin layer that provides a structured method of accessing the VORTEX API, which is a message-based interface.

Since the API is message based and machine-independent, it moves data very efficiently but isn't easy to use. VORTEXcli provides the following benefits:

- Complex message control structures are hidden from the user.
- Automatic data conversion on input and output.
- Uses familiar SQLCA/SQLDA control structures.
- Added features such as error code mapping.
- Does validity checking (the raw API does no checking).
- Bulk fetch/insert/update/delete can be either record or column oriented.

---

*NOTE: This document assumes that you are familiar with ANSI standard SQL pre-compilers.*

---

## Structures

VORTEXcli uses the ANSI standard SQLCA and SQLDA structures as well as two custom structures, TDB\_VAR and TDB\_HVA2, in its communication. (The structures are described in Appendix A on page 51.)

## TDB\_VAR

TDB\_VAR is the ANSI standard SQLVAR structure without the `sqlname`. This structure, used to describe host variables, is 12 bytes long and located in `vortex.h`.

TDB\_VAR has the following fields:

Field	Description
TDB_DTY	The datatype of the host variable.
TDB_LEN	The length of the host variable.
TDB_VAP	Pointer to the host variable.
TDB_IVP	Pointer to the indicator variable. (optional; set to NULL if not required.)

## TDB\_HVA2

This structure (also defined in `vortex.h`) is an extended version of `SQLVAR`. Created to support `VTXEXIO()`, its first 4 fields are identical to `TDB_HVAR`.

Field	Description
<code>TDB_DTY</code>	The datatype of the host variable.
<code>TDB_LEN</code>	The length of the host variable.
<code>TDB_VAP</code>	Pointer to the host variable.
<code>TDB_IVP</code>	Pointer to the indicator variable. (optional; set to NULL if not required.)
<code>TDB_FLG</code>	Indicates if the variable is an input and/or output variable. <ul style="list-style-type: none"> <li>• Input: <code>TDB_FLG_IN</code></li> <li>• Output: <code>TDB_FLG_OUT</code></li> <li>• Input &amp; output: <code>TDB_FLG_IN   TDB_FLG_OUT</code></li> </ul>
<code>TDB_VNL</code>	The length of the variable's name. This must be the full name as required by the target database.
<code>TDB_CNT</code>	Array count. For scalar variables this must be set to 1.
<code>TDB_VNP</code>	Address of buffer containing the variable name.

## Datatypes

The following datatypes describe input and output host variables.

The complete definitions are in `vortex.h`.

Type	Length(s)	Description
0	1, 2, 4, 8	Integer (8 on 64-bit machines only)
1	64K max	Character
2	22	Internal number (Oracle-compatible)
3	<code>sizeof(char*)</code>	Character address
4	20 max	Packed decimal (COBOL COMP-3 compatible)
5	40 max	Zoned decimal
8	4, 8	Floating point
9	64K max	Variable length character (1st two bytes, a short, contains the length)
10	2**31 max	BLOB (Binary Large Object)
11	2**31 max	CLOB (Character Large Object)
12	4, 7, 10	Internal datetime (Oracle-compatible)

Type	Length(s)	Description
80	2**31 max	Varblob (1st four bytes, an int, contains the length)
99	64K max	Binary

## Identifying Numeric Datatypes

When DESCRIBE contains an internal number (type 2 from the preceding list), you can determine the datatype's precision, scale, and original database datatype by examining the `sql1en` (in `SQLVAR`). The low order byte (`sql1en & 0xFF`) contains precision and the high order byte (`(sql1en >> 8) & 0xFF`) indicates the scale. The following four numeric datatypes are standard:

Datatype	Precision	Scale
1 byte signed integer	3	0
2 byte signed integer	5	0
4 byte signed integer	10	0
4 byte floating point	8	6
8 byte floating point	16	6

You can also use this method to determine some non-standard database datatypes, as well.

### *Sybase*

Datatype	Precision	Scale
Bit	1	0
1 byte signed integer1	3	0
Money	16	2

## Converting Datatypes

You can convert from one datatype to another as long as the data makes sense. For example, you can change character data to integer if it contains numeric characters.

Two behavior options (see `vortex.h` and `VTXOPTS()` for additional details) facilitate datetime conversions, which can be tricky since each database vendor has a different format.

*TDB\_DTB = 0;*

When you retrieve a datetime into an integer, *TDB\_DTB* is subtracted from the days returned (typically the number of days from January 1 year 0000). If integer's length has been set to 8, then the 2nd long contains the number of seconds of the day.

*TDB\_DTF = "DD-MON-YY";*

When you retrieve a datetime into a character buffer the format mask "DD-MON-YY" is applied. However, you can use *VTXOPTS()* to select any valid Oracle datetime format mask.

## Error Handling

You should check *SQLCA.SQLCODE* after each *VTXxxxxx()* for the following warnings or conditions:

Value	Indicates
Non-zero	Error or warning condition.
100	End-of-scan.
< 0	Error.
> 0	Warning.
All Others	Actual database return codes, unless you have remapped them with <i>VTXEMAP()</i> .



## Chapter 2

# Database Function Calls

---

---

This chapter describes the VORTEXcli functions in detail. `vortex.x` contains the external declarations.

All the routines except VTXGEEM and VTXEMAP have the first two (2) parameters in common:

- **DB\_index** — Value indicating the database connection to use. If you are writing a multi-threaded application, then the first parameter is a pointer to a handle.
- **sqlca** — Address of the SQLCA structure.

Data-moving routines also have their third (3rd) parameter in common:

- **cursor** — Address of an integer containing the logical cursor number.

Thus, many functions begin with:

```
unsigned int VTXxxxx
(
    hdl      DB_index,
    SQLCA   *sqlca,
    int     *cursor,
    ...
);
```

where `hdl` is either an `int` or a `void *` pointer.

Whenever you open a connection to a database, you must initialize the environment, connect to the database, and then release from the database when the work is completed.

1. VTXINIT() — Initialize VORTEXcli environment.
2. VTXCONN() — Connect to a database.
3. VTXxxxx() — Perform any number of database functions.
4. VTXREL() — Release from the database.

## VTXBLOB

Puts/gets BLOB (Binary Large Object) or CLOB (Character Large Object) data. VTXBLOB is a deprecated interface. New applications should use VTXLOBG and VTXLOBP.

### Syntax

```
unsigned int VTXBLOB
(
    hdl            DB_index,
    SQLCA         *sqlca,
    int           *cursor,
    int           column,
    char          *buf_ptr,
    unsigned int  buf_len
);
```

### Parameters

- DB\_index** Indicates the database connection to use.
- sqlca** SQLCA structure's address.
- column** When putting data, this is set to -datatype, either -1 for character or -99 for binary. When retrieving data, this indicates which BLOB column (0-based) to retrieve.
- buf\_ptr** Points to the BLOB buffer.
- buf\_len** Specifies the BLOB buffer's length.

### Return Value

Length of fetched data.

### Notes

See the VTXMOVE() Notes for an alternative method of fetching blob/clob data.

## VTXCAN

Cancels any outstanding database request.

### Syntax

```
void VTXCAN
(
    hdl      DB_index,
    SQLCA   *sqlca
);
```

### Parameters

**DB\_index** Indicates the database connection to use.

**sqlca** SQLCA structure's address.

### Notes

Typically called from a Ctrl-C handler when a user wants to interrupt a wild query.

## VTXCLOS

Closes a logical cursor.

### Syntax

```
void VTXCLOS
(
    hdl      DB_index,
    SQLCA   *sqlca,
    int     *cursor,
    int     hard
);
```

### Parameters

- DB\_index** Indicates the database connection to use.
- sqlca** SQLCA structure's address.
- cursor** Address of an integer containing the logical cursor number.
- hard** Indicates that the function should perform an actual underlying database cursor close. If false then the cursor is only marked as closed (soft). If true, then the actual database cursor is closed and the integer pointed to by \*cursor is reset to -1.

### Notes

For performance reasons, you should avoid hard closing logical cursors.

## VTXCMD

Executes a database driver-specific command.

### Syntax

```
void VTXCMD
(
    hdl      DB_index,
    SQLCA   *sqlca,
    int     *cursor,
    int     command,
    char    *parms
);
```

### Parameters

- DB\_index** Indicates the database connection to use.
- sqlca** SQLCA structure's address.
- cursor** Address of an integer containing the logical cursor number.
- command** Command code. The following commands are available with the listed parms values and their descriptions (See `vortex.h` for definitions.). If a driver doesn't support a command, the command is simply ignored. The command codes in the table are all prefixed with "TDB\_CMD\_"

Cmd	Parms	DBMS	Duration	Default	Description
TIMEOUT	n	All	Session	Differs	Set resource timeout to <n>, seconds.
SINGLEPID	yes no	Sybase	Session	no	Use a single PID within a transaction.
OOPT	rbopt waitopt	Oracle	Cursor	See oopt()	Rollback option (rbopt) and wait on resource option (waitopt).
USEDDB	dbname	Sybase	Session	N/A	Use database<dbname>.
LANGUAGE	language	Oracle	Session	See oparse() or StmtPrepare()	Set language and datatypes for SQL statements.
RAW_DATETIME	yes no	All	Session	no	Return raw datetime.
CHAR	dtty	Oracle	Session	1	Map VORTEX char type to Oracle type <dtty>.
LOCK	N/A	All	Call	N/A	Lock VORTEXaccelerator slave on next call.
ONEBYTE	yes no	Sybase	Session	no	Return single blank varchar as blank instead of null.

Cmd	Parms	DBMS	Duration	Default	Description
MULTISTMT	yes   no	Sybase	Session	no	Use individual insert statements instead of bulk operations.
AUTOCOMMIT	on   off	GENESIS ODBC	Session	off	Automatically commit after every SQL statement.
DORMANT	N/A	Informix	Until next use	N/A	Sets the current connection to dormant.
GETLDA	N/A	Oracl	Call	N/A	Returns the address of the LDA on Oracle 7 only.
RAISE	raise() value	TRIMrpc	Call	N/A	Cause a raise error condition.
ERROR	Error message	TRIMrpc	Call	N/A	Return the error message.
TMP_RELEASE	N/A	VORTEXwebd	Call	N/A	Execute a temporary release
WORM_AUTH	sdms2_initx() parm	GENESIS/SDMS	Call	N/A	Call sdms2_initx() using the parameter.
TIMEOUT_QUERY	n	GENESIS	Session	2	Set the query timeout value.
TIMEOUT_FETCH	n	GENESIS/SDMS	Session	2	Set the fetch timeout value.
DYN_CURSOR	yes   no	ODBC	Session	connection default	Use dynamic cursors.
SYNTAX_LEVEL	N/A	GENESIS	Session	N/A	Return GENESIS syntax level
RETURN_ROWID	yes   no	All	Session	yes	Return ROWID on insert.
NEW_BLOBS	yes   no	Oracle	Session	no	Use BLOB/CLOB instead of LONG RAW/LONG.
RO_CURSOR	yes   no	ODBC	Session	no	Use read-only cursors.
CURSOR_TYPE	Cursor type	ODBC	Session	SQL_CURS OR_TYPE_ DEFAULT	Set the ODBC cursor type to one of: SQL_CURSOR_STATIC SQL_CURSOR_KEYSET_DRIVEN SQL_CURSOR_DYNAMIC SQL_CURSOR_FORWARD_ONLY
TXN_ISOLEVEL	Isolation type	ODBC	Session	N/A	Set the ODBC cursor isolation level to one of: SQL_TXN_READ_UNCOMMITTED SQL_TXN_READ_COMMITTED SQL_TXN_REPEATABLE_READ SQL_TXN_SERIALIZABLE
SCROLL	Scroll option	Oracle Informix DB2 AdabasD ODBC	Cursor	N/A	Set the scrolling direction or position of a scrollable cursor (see VTXPEN). Scroll options are defined in vortex.h as TDB_CMD_SCROLL_xx
HOSTNAME	N/A	All	N/A	N/A	Return the hostname of the machine where the DBMS driver is running.
FQCOLNAME	yes   no	All	Session	no	Return fully qualified column names.

Cmd	Parms	DBMS	Duration	Default	Description
FREE_LOCKS	N/A	ODBC	Cursor	N/A	Free the locks held by the cursor.
NOUNICHAR	yes no	ODBC SQL Server	Session	no	Describe W[VAR]CHAR columns as CHAR.
ZEROLEN_NULL	yes no	ODBC SQL Server	Session	no	Forces zero length parameter strings to be bound as NULL
EXIO_RS	yes no	DB2	Session	no	Process stored procedure cursors with EXIO
DRV_CONFIG	SSL options	NET	Session	no	Sends SSL connection parameters. Refer to Notes section.
BLK_MIN	n	PostgreSQL	Session	10	Minimum number of bulk insert records needed to switch to SQL statement expansion method.
MAX_COL_LEN	n	PostgreSQL	Session	65535	Maximum described length for char or expression column (max value: 65535).
SET_CON_ATTR	attr value	ODBC SQL Server DB2	Session	N/A	SQLSetConnectAttribute attribute and value to set.
GET_CON_ATTR	attr	ODBC SQL Server DB2	Session	N/A	SQLGetConnectAttribute attribute, value returned as string in message buffer.
DESC_PARMS	yes no	SQL Server	Session	no	Use SQLDescribeParam to avoid filling plan cache with similar stmts, different parameter values.
DESC_LOBS	yes no	ODBC	Session	no	Describe LOB columns as LOBs, else char/binary.
DESC_NULL	n	PostgreSQL	Session	0	Describe column nullability: 0 query actual nullability, 1 set all false, 2 set all true.

**parms** Address of a null terminated parameter string. Multiple parameters must be blank separated.

## Notes

The `cursor` is ignored for commands that are session-oriented.

The `TDB_CMD_DRV_CONFIG (34)` command is used between the `VTXINIT` and `VTXCONN` calls. It is used to set up SSL connection parameters and overrides any SSL connection parameters found in `net.ini`. It is valid only for `net:` connections and otherwise ignored. There are four keywords in the `Parms` string and they are separated by `“;”`:

Keyword	Value
3	TDB_DID_NET. This <u>must</u> be the first item.
SSL	yes/no

---

<b>SSL_Certificate</b>	Filename for SSL Certificate Trust. If specified, the server certificate will be validated.
<b>SSL_Protocol</b>	The desired TLS levels. Currently supported values are 1.0,1.1, and 1.2. Multiple levels are comma separated, e.g. 1.1,1.2.

---

For example, to specify SSL using TLS levels 1.1 and 1.2, the parameter string would be:

```
3;SSL=YES;SSL_Protocol=1.1,1.2
```

The keywords are case-insensitive and can be in any order except that the string must begin with TDB\_DID\_NET;. The target VORTEXserver must also be running in SSL mode and support one of the requested TLS protocols. The Params string must remain unchanged until after the VTXCONN call.

## VTXCOMM

Commits a transaction.

### Syntax

```
void VTXCOMM
(
    hdl      DB_index,
    SQLCA   *sqlca,
    int      start_write
);
```

### Parameters

**DB\_index** Indicates the database connection to use.

**sqlca** SQLCA structure's address.

**start\_write** Specifies to start a write transaction following the commit.

### Notes

`start_write` is ignored by databases that don't support the concept of different transaction states.

## VTXCONN

Connects to a database.

### Syntax

```
void VTXCONN
(
    hdl      DB_index,
    SQLCA   *sqlca,
    char    *connect,
    int     connect_len,
    int     not_used
);
```

### Parameters

- DB\_index** Indicates the database connection to use.
- sqlca** SQLCA structure's address.
- connect\_len** Length of the connect string.
- not\_used** Currently not used. Must be set to 0.

## VTXDES2

Describes a previously opened SELECT statement.

### Syntax

```
void VTXDES2
(
    hdl      DB_index,
    SQLCA   *sqlca,
    int     *cursor,
    SQLDA   *sqlda
);
```

### Parameters

- DB\_index** Indicates the database connection to use.
- sqlca** SQLCA structure's address.
- cursor** Address of an integer containing the logical cursor number.
- sqlda** Address of SQLDA structure.

### Notes

You must initialize `SQLDA` before using this function.

Structure	Set to ...
<code>sqldaid</code>	"SQLDA " (3 trailing blanks)
<code>sqldabc</code>	<code>sizeof(SQLDA) + sizeof(SQLVAR)*(sqln - 1)</code>
<code>sqln</code>	number of allocated SQLVARs

If you already have an open cursor, `VTXDES2 ()` is faster than `VTXDESC ()` because it populates the `SQLDA` directly from the cursor's internal description buffer instead of making a database call.

If you have used the `VTXOPTS ()` call to set the NULL mask bit, you must remove this bit from the `sqltype` fields before using the results of `VTXDES2 ()` in a `VTXOPEN ()`, `VTXMOVE ()`, or `VTXEXEC ()` call.

## VTXDES3

Returns the column count of a previously opened SELECT statement.

### Syntax

```
int VTXDES3
(
    hdl      DB_index,
    SQLCA   *sqlca,
    int     *cursor
);
```

### Parameters

- DB\_index** Indicates the database connection to use.
- sqlca** SQLCA structure's address.
- cursor** Address of an integer containing the logical cursor number.

## VTXDES4

Describes the parameters of a SQL statement.

### Syntax

```
int VTXDES4
(
    hdl      DB_index,
    SQLCA   *sqlca,
    char    *sqlp,
    SQLDA   *sqlda
);
```

### Parameters

- DB\_index** Indicates the database connection to use.
- sqlca** SQLCA structure's address.
- sqlp** Address of the null-terminated SQL statement.
- sqlda** Address of SQLDA structure.

### Notes

You must initialize `SQLDA` before using this function.

---

Structure	Set to ...
<code>sqldaid</code>	"SQLDA " (3 trailing blanks)
<code>sqldabc</code>	<code>sizeof(SQLDA) + sizeof(SQLVAR)*(sqln - 1)</code>
<code>sqln</code>	number of allocated <code>SQLVARs</code>

---

## VTXDESC

Describes a SQL statement.

### Syntax

```
void VTXDESC
(
    hdl      DB_index,
    SQLCA   *sqlca,
    char    *sqlp,
    SQLDA   *sqlda
);
```

### Parameters

**DB\_index** Indicates the database connection to use.

**sqlca** SQLCA structure's address.

**sqlp** Address of the null-terminated SQL statement.

**sqlda** Address of SQLDA structure.

### Notes

You must initialize `SQLDA` before you can use this function.

Structure	Set to ...
<code>sqldaid</code>	"SQLDA " (3 trailing blanks)
<code>sqldabc</code>	<code>sizeof(SQLDA) + sizeof(SQLVAR)*(sqln - 1)</code>
<code>sqln</code>	number of allocated SQLVARs

You can avoid an actual database call if you already have an open cursor. `VTXDES2()` populates the `SQLDA` directly from the cursor's internal description buffer.

If you have set used the `VTXOPTS()` call to set the NULL mask bit, you must remove this bit from the `sqltype` fields before using the results of `VTXDESC()` in a `VTXOPEN()`, `VTXMOVE()`, or `VTXEXEC()` call.

## VTXEXEC

Executes a non-SELECT statement.

### Syntax

```
void VTXEXEC
(
    hdl          DB_index,
    SQLCA      *sqlca,
    int         *cursor,
    char        *sqlp,
    int         sqlvar_cnt,
    SQLVAR     *sqlvarp,
    int         sqlvar_len,
    int         record_cnt,
    int         record_len,
    int         option
);
```

### Parameters

- DB\_index** Indicates the database connection to use.
- sqlca** SQLCA structure's address.
- cursor** Address of an integer containing the logical cursor number.
- sqlp** Address of the null-terminated SQL statement.
- sqlvar\_cnt** Number of SQLVARs.
- sqlvarp** Address of the SQLVARs.
- sqlvar\_len** Length of each SQLVAR (typically 44 bytes for the full length version and 12 bytes for the shortened version).
- record\_cnt** Number of records to process.
- record\_len** Length of each data record. If the data is column oriented, then `record_len` must be set to 0.
- option** 0 - standard, 1 - positioned. If positioned then cursor must be currently opened with the positioned option. (see `VTXOPEN()`).

### Notes

You can use `VTXEXEC()` to call stored procedures, but if the stored procedure returns more than a single scalar value then you need to use `VTXEXIO()`.

`sqlca.sqlerrd[2]` returns the number of records affected (inserted, updated, or deleted) for non-stored procedure statements.

## VTXEXIO

Executes a stored procedure using input and/or output variables.

### Syntax

```
void VTXEXIO
(
    hdl          DB_index,
    SQLCA       *sqlca,
    int         *cursor,
    char        *sqlp,
    int         hostvar_cnt,
    TDB_HVA2    *hostvarp,
);
```

### Parameters

**DB\_index** Indicates the database connection to use.

**sqlca** SQLCA structure's address.

**cursor** Address of an integer containing the logical cursor number.

**sqlp** Address of the null terminated SQL statement.

**hostvar\_cnt** Number of TDB\_HVA2s.

**hostvarp** Address of the TDB\_HVA2s.

### Notes

If a message or status is returned, you can retrieve it using `VTXGEEM()`. The format of this message is database brand-specific.

DB2 stored procedures can return cursor results. To test for pending cursor results, check `SQLCA.SQLERRD[0]`. While `SQLCA.SQLERRD[0] = 1`, use `VTXDES2()` to describe the cursor results and `VTXMOVE()` to retrieve results. After `VTXMOVE()` returns `SQLCA.SQLCODE = 100`, check `SQLCA.SQLERRD[0] = 1`. Once this fails, then the stored procedure has returned all cursor results.

## VTXINIT

Initializes a database instance.

### Syntax

```
void VTXINIT
(
    hdl      DB_index,
    SQLCA   *sqlca,
    int      max_cursors,
    int      max_columns,
    int      max_db_cursors,
    int      fetch_buf_size,
    int      version_number
);
```

### Parameters

<b>DB_index</b>	Indicates the database connection to use. For a multi-threaded application, this is a pointer to a void * pointer variable.
<b>sqlca</b>	SQLCA structure's address.
<b>max_cursors</b>	Maximum number of logical cursors to be used. This should be set to some high number (such as 512).
<b>max_columns</b>	Maximum number of columns to be processed at any given time.
<b>max_db_cursors</b>	Maximum number of actual database cursors to allocate. A good number is 1/8 of max_cursors.
<b>fetch_buf_size</b>	Size of the internal buffer used to pre-fetch data.
<b>version_number</b>	Must be set to TDB_PRE_VER_NUM (see vortex.h).

### Notes

This *must* be the first function called for each database.

## VTXIPC

Sends a message to a non-VORTEX function.

### Syntax

```
void VTXIPC
(
    hdl      DB_index,
    SQLCA   *sqlca,
    byte    *sbufp,
    int     sbuf_len,
    byte    *rbufp,
    int     rbuf_len
);
```

### Parameters

<b>DB_index</b>	Indicates the database connection to use.
<b>sqlca</b>	SQLCA structure's address.
<b>sbufp</b>	Pointer to the send data buffer.
<b>sbuf_len</b>	Length of the send data buffer.
<b>rbufp</b>	Pointer to the receive data buffer.
<b>rbuf_len</b>	Length of the receive data buffer.

### Notes

VTXIPC() communicates with non-VORTEX functions via VORTEXchannel. The function's main entry point must be named TIPMAIN and receive parameters for the input buffer and length and the output buffer and length. TIPMAIN returns the length of the output data. If TIPMAIN returns a (-1) length, then the error message is a null-terminated string stored in the output buffer.

```
int TIPMAIN
(
    byte    *bufp,
    int     buf_len,
    byte    *rbufp,
    int     rbuf_len);
```

## VTXLOBG

Gets LOB data.

### Syntax

```
unsigned int VTXLOBG
(
    hdl          DB_index,
    SQLCA       *sqlca,
    int         *cursor,
    int         column,
    int         data_type,
    char        *buf_ptr,
    unsigned int buf_len,
    unsigned int source_len
);
```

### Parameters

- DB\_index** Indicates the database connection to use.
- sqlca** SQLCA structure's address.
- column** This indicates which LOB column (0-based) to retrieve.
- data\_type** If **data\_type** is 0, then the LOB data is moved directly to **buf\_ptr**. If the LOB is type BLOB or CLOB, then **data\_type** can be TDT\_BINARY, TDT\_BLOB, TDT\_CLOB, or TDT\_CHAR; otherwise a BADCONV error is returned. If the LOB is UTF8LOB, then **data\_type** can be any of the TDT unicode types as well as TDT\_CHAR; anything else will return a BADCONV error.
- buf\_ptr** Points to the LOB buffer.
- buf\_len** Specifies the LOB buffer's length in bytes
- source\_len** Specifies the number of bytes to be fetched from the database.

### Return Value

Length of fetched data.

### Notes

VTXLOBG can be called in a loop to fetch the LOB in pieces.

## VTXLOBP

Puts LOB data.

### Syntax

```
unsigned int VTXLOBP
(
    hdl            DB_index,
    SQLCA         *sqlca,
    int           *cursor,
    int           column,
    int           data_type,
    char         *buf_ptr,
    unsigned int  buf_len
);
```

### Parameters

- DB\_index** Indicates the database connection to use.
- sqlca** SQLCA structure's address.
- column** This indicates which LOB column (0-based) to send.
- data\_type** One of TDT\_UTF8, TDT\_UTF16, TDT\_UCS2, TDT\_UCS4, TDT\_CHAR, or TDT\_BINARY.
- buf\_ptr** Points to the LOB buffer.
- buf\_len** Specifies the LOB buffer's length in bytes

### Return Value

True for error, false for success.

### Notes

## VTXMOVE

Moves (fetches) data into host variables.

### Syntax

```
void VTXMOVE
(
    hdl          DB_index,
    SQLCA      *sqlca,
    int         *cursor,
    SQLVAR     *sqlvarp,
    int         sqlvar_len,
    int         record_cnt,
    int         record_len
);
```

### Parameters

<b>DB_index</b>	Indicates the database connection to use.
<b>sqlca</b>	SQLCA structure's address.
<b>cursor</b>	Address of an integer containing the logical cursor number.
<b>sqlvarp</b>	Address of the SQLVARs.
<b>sqlvar_len</b>	Length of each SQLVAR (typically 44 bytes for the full length version and 12 bytes for the shortened version).
<b>record_cnt</b>	Number of records to move.
<b>record_len</b>	Length of each data record. If the data is column oriented, then record_len must be set to 0.

`SQLCA.SQLERRD[2]` stores the number of records returned in the `VTXMOVE()` call.

If `SQLCA.SQLCODE == 100`, then there are no more records unless `SQLCA.SQLERRD[0] > 0`.

If that is the case then `SQLCA.SQLERRD[0] == 1` indicates a new result set and `SQLCA.SQLERRD[0] == 2` indicates a COMPUTE row set.

Sybase and SQL Server are the only databases that support changing result sets and COMPUTE rows.

You can retrieve any return messages (formats are database-specific) or status with `VTXGEEM()`.

### Notes

Most of the host variable types are self-explanatory, eg an integer host variable will receive an integer value from the database. There are two exceptions to this, BLOB and CLOB host variables. Host variables of BLOB/CLOB type point to a four byte integer which receives the length of blob/clob data. The user must then use `VTXBLOB()` calls to

actually fetch the data. If you know the maximum length of the data or you know that you only want a maximum amount returned, then you can use the varblob datatype. It is similar to varchar but with a four byte length.

## VTXOPEN

Opens a SELECT cursor.

### Syntax

```
void VTXOPEN
(
    hdl      DB_index,
    SQLCA    *sqlca,
    int      *cursor,
    char     *sqlp,
    int      sqlvar_cnt,
    SQLVAR   *sqlvarp,
    int      sqlvar_len,
    int      option
);
```

### Parameters

<b>DB_index</b>	Indicates the database connection to use.
<b>sqlca</b>	SQLCA structure's address.
<b>cursor</b>	Address of an integer containing the logical cursor number.
<b>sqlp</b>	Address of the null-terminated SQL statement.
<b>sqlvar_cnt</b>	Number of SQLVARs.
<b>sqlvarp</b>	Address of the SQLVARs.
<b>sqlvar_len</b>	Length of each SQLVAR (typically 44 bytes for the full length version and 12 bytes for the shortened version).
<b>option</b>	0x0000 standard. 0x0001 positioned. 0x0002 statement contains a FOR UPDATE OF clause. 0x0004 select list contains BLOB columns. 0x0008 singleton Select 0x0010 scrolling cursor 0x0020 static scrolling cursor 0x0040 dynamic scrolling cursor 0x0080 insensitive scrolling cursor 0x0100 asensitive scrolling cursor

### Notes

The **options** can be OR'd. The values are also defined in `vortex.h` as `TDB_PRE_xxxx`.

Scrollable cursors are controlled by sending direction or positioning commands via VTXCMD's `TDB_CMD_SCROLL` command.

## VTXREL

Releases (disconnects) from a database connection.

### Syntax

```
void VTXREL
(
    hdl      DB_index,
    SQLCA   *sqlca
);
```

### Parameters

**DB\_index** Indicates the database connection to use.

**sqlca** SQLCA structure's address.

### Notes

All transactions are automatically rolled back.

Reset all cursors used by this connection to -1 before re-connecting.

You must call `VTXINIT` again to initialize the `hdl`.

## VTXROLL

Rollback a transaction.

### Syntax

```
void VTXROLL
(
    hdl      DB_index,
    SQLCA   *sqlca,
    int      start_write
);
```

### Parameters

**DB\_index** Indicates the database connection to use.

**sqlca** SQLCA structure's address.

**start\_write** Indicates that a write transaction should be started following the rollback.

### Notes

**start\_write** is ignored by databases that don't support the concept of different transaction states.



## Chapter 3

# Utility Routines

---

---

Utility routines, which work with `.DLLs`, help you read and write internal information.

These utilities control various VORTEXcli behaviors, such as remapping error codes and formatting datetime data. They are local to VORTEXcli and have no effect on the database.

## VTXEMAP

Maps a database specific error code to a generic error code.

### Syntax

```
int VTXEMAP
(
    int    DB_id,
    int    error_code,
    int    default_code,
    int    *mapped_code,
    char   *map_file
);
```

### Parameters

- DB\_id** Database identification (see `VTXGDID()` for valid values).
- error\_code** Database specific error code. This error code is returned in `SQLCA.SQLCODE`
- default\_code** Code to return if no match was found in the map file.
- mapped\_code** Address where resulting (mapped) error code is placed.
- map\_file** Address of the null-terminated error mapping file name. `DB_id` is appended before the file is opened. This parameter is only used on the first call for each `DB_id`.

## Return Value

Possible return values are:

(see `vortex.h` for `TDB_EMAP_xxxx` definitions)

- 0 — All is OK. Result is in `*mapped_code`.
- 1 — Map file could not be opened.
- 2 — Illegal format in map file.
- 3 — Out of memory.

## Notes

The map file is a regular text file where each line contains two numbers. The first number is the database-specific error code. The second number is the matching generic error code that is to be returned. Note that most error codes are negative.

## VTXGDID

Gets the database identification of the connection.

## Syntax

```
int VTXGDID
(
    int DB_index
);
```

## Parameters

**DB\_index** Value that indicates which database to use.

## Return Value

Possible return values are (see `vortex.h` for `TDB_DID_xxxx` definitions):

ID	Database	Vendor
-1	Not connected	N/A
0	Oracle	Oracle
1	Rdb	Oracle (formerly DEC)
2	Sybase	Sybase
4	Genesis	Trifox
5	Informix	Informxi
6	Allbase	HP

ID	Database	Vendor
7	DB2	IBM
8	Ingres	CA
9	ADABAS C	Software AG
10	ADABAS D	Software AG
11	ODBC	Microsoft
12	SQLServer	Microsoft
13	Teradata	NCR

## VTXGEEM

Gets the extended database error message.

### Syntax

```
int VTXGEEM
(
    int    DB_index,
    char   *msg_buffer,
    int    msg_buffer_len
);
```

### Parameters

**DB\_index** Value indicating which DB to use.

**msg\_buffer** Address of the buffer that receives the message.

**msg\_buffer\_len** Length of the msg\_buffer.

### Return Value

Length of the message.

### Notes

Because the message is always null terminated, the actual message cannot be longer than `msg_buffer_len-1`.

Some databases return multiple message lines. If VORTEXcli receives multiple lines, they are separated by nulls. If `strlen(msg_buffer)` is less than the returned length then multiple lines exist in the buffer.

## VTXGLEN

Returns the lengths of the data items in the current record into the provided integer array.

### Syntax

```
void VTXGLEN
(
    hdl      DB_index,
    SQLCA *sqlca,
    int      *cursor,
    int      *lenp
);
```

### Parameters

<b>DB_index</b>	Value indicating which DB to use.
<b>SQLCA</b>	SQLCA structure's address.
<b>cursor</b>	Address of an integer containing the logical cursor number.
<b>lenp</b>	Array of integers to receive the data item lengths.

### Notes

The array pointed to by lenp must contain as many integers as there are columns in the record.

## VTXOPTS

Gets/sets VORTEXcli behavior options.

### Syntax

```
void VTXOPTS
(
    int      option,
    TDB_OPTS *option_buffer
);
```

## Parameters

**option**            0 — set options; 1 — get options

**option\_buffer** Address of the options buffer.

## Notes

See `vortex.h` for a description of `TDB_OPTS`.



## Chapter 4

# Conversion Routines

---

---

VORTEXserver uses internal numeric and date-time formats to manipulate these datatypes. The numeric format is a variable byte array of up to 22 bytes and the date-time format is a seven-byte array. Of these two, the date-time is of more use to the VORTEXcli and VORTEXcobol developer.

Since every database vendor uses a different date-time format, it is difficult to use character strings to manipulate date-time values. It is simpler to convert date-time strings to the internal format when communicating with VORTEXserver.

See *Format Masks* in the *VORTEX Installation and Usage Guide* for details on format masks.

The conversion routines are written in C. Your COBOL code must use the correct calling conventions. See your COBOL documentation for complete details.

## TCVCHFM

TCVCHFM validates the format mask `maskp` for the datatype `dtty`.

TCVCHFM returns `mask_len` if valid, otherwise 0 is returned.

### *Syntax*

```
int TCVCHFM
(
    int    dtty,
    char  *maskp,
    int    mask_len
);
```

### *Parameters*

**dtty**        Datatype  
**maskp**      Mask pointer  
**mask\_len**   Length of maskp buffer

## TCVD2I4

Convert internal date-time value into integer days from year 0 and integer seconds in the day. The number of days is returned from TCVD2I4 and, if `secp` is not NULL, the number of seconds in the day is returned.

*Syntax*

```
int TCVD2I4
(
    unsigned char *dt,
    long          dt_len,
    long          *secp
);
```

*Parameters*

**dt** Internal date-time value.

**dt\_len** Length of the internal date-time value.

**secp** Seconds in day (returned).

**TCVD2N**

TCVD2N converts an internal date-time value into an internal numeric value containing days from year 0 and fractional day. The length of the internal number is returned by TCVD2N.

*Syntax*

```
int TCVD2N
(
    unsigned char *dt,
    long          dt_len,
    unsigned char *nump
);
```

*Parameters*

**dt** Internal date-time value.

**dt\_len** Length of the internal date-time value.

**nump** Pointer to internal numeric buffer.

**TCVD2S**

TCVD2S converts an internal date-time value into a character string. The length of the string is returned.

*Syntax*

```
int TCVD2S
(
    unsigned char *dt,
    long          dt_len,
    char          *maskp,
    int           mask_len,
    char          *bufp
);
```

```
);
```

### Parameters

**dt** Internal date-time value.

**dt\_len** Length of the internal date-time value.

**maskp** Format mask.

**mask\_len** Format mask length.

**bufp** Character string buffer (returned).

## TCVDD2N

Convert DIBOL decimal to internal numeric format. The length of the internal numeric is returned.

### Syntax

```
int TCVDD2N
(
char          *ddp,
int           dd_len,
int           dd_scale,
unsigned char *nump
);
```

### Parameters

**ddp** DIBOL decimal to convert.

**dd\_len** DIBOL decimal length.

**dd\_scale** DIBOL scale.

**nump** Number buffer (returned).

## TCVDINI

TCVDINI provides a method for the user to change the English names of the months and days used by TCVD2S and TCVS2D to any other language. TCVDINI returns true on success, false on errors.

### Syntax

```
int TCVDINI
(
char *fname
);
```

### *Parameters*

**fname**      Name of file containing month and day strings.

### *Notes*

The file pointed to by `fname` must contain 19 strings: 12 month names and 7 day names. The strings must be at least 3 characters long and in uppercase.

For example, the French file would contain:

```
JANVIER
FEVRIER
MARS
AVRIL
MAI
JUIN
JUILLET
AOUT
SEPTEMBRE
OCTOBRE
NOVEMBRE
DECEMBRE
DIMANCHE
LUNDI
MARDI
MERCREDI
JEUDI
VENDREDI
SAMDI
```

## **TCVF42N**

TCVF42N converts a four byte float value to an internal numeric. The length of the numeric is returned.

### *Syntax*

```
int TCVF42N
(
    float          *fp,
    unsigned char  *nump
);
```

*Parameters*

**fp**            Pointer to four-byte float value.  
**nump**        Internal numeric value (returned).

**TCVF82N**

TCVF82N converts an eight byte double value to an internal numeric. The length of the numeric is returned.

*Syntax*

```
int TCVF82N
(
    double          *fp,
    unsigned char   *nump
);
```

*Parameters*

**fp**            Pointer to eight-byte double value.  
**nump**        Internal numeric value (returned).

**TCVI42D**

TCVI42D converts in two integer values, days from year 0 and number of seconds in the day, into an internal date-time value. The length of the internal date-time value is returned.

*Syntax*

```
int TCVI42D
(
    long           num_days,
    long           num_secs,
    unsigned char  *dt
);
```

*Parameters*

**num\_days**    Number of days since year 0.  
**num\_secs**    Number of seconds in the day.  
**dt**           Internal date-time value (returned).

**TCVI42N**

TCVI42N converts a four byte integer into an internal numeric. The length of the numeric is returned.

*Syntax*

```
int TCVI42N
(
    long          ii4,
    unsigned char *nump
);
```

**TCVI82N**

TCVI82N converts an eight-byte integer into an internal numeric. The length of the numeric is returned. Only applicable to platforms that support eight-byte integers.

*Syntax*

```
int TCVI82N
(
    long long ii8,
    unsigned char *nump
);
```

*Parameters*

**ii8** Eight-byte integer to convert.

**nump** Internal numeric value (returned).

**TCVN2DD**

Convert internal numeric to DIBOL decimal. The length of the DIBOL decimal is returned.

*Syntax*

```
int TCVN2DD
(
    unsigned char *nump,
    int          num_len,
    char         *ddp,
    int          pas
);
```

*Parameters*

**nump**     Number to convert.

**num\_len**   Length of number.

**ddp**       (returned) DIBOL decimal.

**pas**       Precision and scale. Scale is the upper byte, precision is in the lower byte.

*Parameters*

**ii4**       Integer to convert.

**nump**       Internal numeric value (returned).

**TCVN2D**

TCVN2D converts an internal numeric value into an internal date-time value. The length of the numeric is returned.

*Syntax*

```
int TCVN2D
(
  unsigned char *nump,
  int          num_len,
  unsigned char *dt
);
```

*Parameters*

**nump**     Number of days, including fractional day.

**num\_len**   Length of nump buffer.

**dt**       Internal date-time value (returned).

**TCVN2F4**

TCVN2F4 converts an internal format numeric to a four byte float.

*Syntax*

```
void TCVN2F4
(
  unsigned char *nump,
  int          num_len,
  float        *fp
);
```

*Parameters*

**nump** Internal numeric buffer  
**num\_len** Length of nump buffer.  
**fp** Address of four-byte float (returned).

*Notes*

Care must be taken to ensure that fp points to a correctly aligned float address.

**TCVN2F8**

TCVN2F8 converts an internal format numeric to an eight byte double.

*Syntax*

```
void TCVN2F8
(
    unsigned char *nump,
    int          num_len,
    double       *fp
);
```

*Parameters*

**nump** Internal numeric buffer  
**num\_len** Length of nump buffer.  
**fp** Address of four-byte float (returned).

*Notes*

Care must be taken to ensure that fp points to a correctly aligned double address.

TCVN2FS converts an internal format numeric into a formatted string.

**TCVN2FS***Syntax*

```
int TCVN2FS
(
    unsigned char *nump,
    int          num_len,
    char         *maskp,
    int          mask_len,
    int          jus,
    char         *fsp
);
```

*Parameters*

<b>nump</b>	Internal numeric buffer
<b>num_len</b>	Length of nump buffer.
<b>maskp</b>	Pointer to format mask.
<b>jus</b>	Justification (“L,” “C,” or “R”)
<b>fsp</b>	Address of formatted buffer (returned).

**TCVN2I4**

TCVN2I4 converts an internal format numeric to a four byte integer. The integer value is returned from the function.

*Syntax*

```
long TCVN2I4
(
    unsigned char *nump,
    int          num_len,
    int          *status
);
```

*Parameters*

<b>nump</b>	Internal numeric buffer
<b>num_len</b>	Length of nump buffer.
<b>status</b>	Conversion status. <ul style="list-style-type: none"> <li>• -1 — truncation</li> <li>• 0 — ok</li> <li>• 1 — overflow</li> </ul>

*Notes*

Care must be taken to ensure that status points to a correctly aligned integer address.

**TCVN2I8**

TCVN2I8 converts an internal format numeric to an eight-byte integer. The integer value is returned from the function. Only applicable to platforms that support eight-byte integers.

*Syntax*

```
long long TCVN2I8
(
    unsigned char *nump,
    int num_len,
    int *status
```

```
);
```

### Parameters

**nump** Internal numeric buffer.

**num\_len** Length of nump buffer.

**status** Conversion status:  
 -1 — truncation  
 0 — ok  
 1 — overflow

### Note

Take care to ensure that the status points to a correctly aligned integer address.

## TCVN2PD

TCVN2PD converts an internal numeric to a packed decimal format. The length of the packed decimal is returned.

### Syntax

```
int TCVN2PD
(
  unsigned char *nump,
  int          num_len,
  unsigned char *pdp,
  int          pas
);
```

### Parameters

**nump** Internal numeric buffer

**num\_len** Length of nump buffer.

**pdp** Packed decimal buffer (returned).

**pas** Precision and scale of packed decimal (scale <<8 — precision)

## TCVN2S

TCVN2S converts an internal numeric into a character string. The length of the string is returned.

### Syntax

```
int TCVN2S
(
  unsigned char *nump,
```

```

int          num_len,
char         *p,
int          max_len,
int          eee
);

```

### Parameters

**nump** Internal numeric buffer

**num\_len** Length of nump buffer.

**p** Character string buffer (returned)

**max\_len** Length of pBuffer

**eee** Set to true of scientific notation desired.

## TCVN2U4

TCVN2U4 converts an internal format numeric to a four byte unsigned integer. The integer value is returned from the function.

### Syntax

```

unsigned long TCVN2U4
(
    unsigned char *nump,
    int          num_len,
    int          *status
);

```

### Parameters

**nump** Internal numeric buffer

**num\_len** Length of nump buffer.

**status** Conversion status.

- -1 — truncation (negative or between 0 and 1)
- 0 — ok
- 1 — overflow

### Notes

Ensure that status points to a correctly aligned integer address.

## TCVN2U8

TCVN2U8 converts an internal numeric to an unsigned eight byte integer. The integer value is returned from the function. Only applicable to platforms that support eight byte integers.

```

unsigned long long TCVN2U8
{
    unsigned char *nump;
    int          num_len;

```

```

    int          *status;
};

```

### Parameters

**nump** Internal numeric buffer.

**num\_len** Length of nump buffer.

**status** Conversion status:  
 -1 — truncation  
 0 — ok  
 1 — overflow

## TCVN2ZD

TCVN2ZD converts an internal numeric into a zoned decimal value. The length of the zoned decimal is returned.

### Syntax

```

int TCVN2ZD
(
    unsigned char *nump,
    int          num_len,
    unsigned char *zdp,
    int          pas
);

```

### Parameters

**nump** Internal numeric buffer

**num\_len** Length of nump buffer.

**zdp** Zoned decimal value (returned)

**pas** Precision and scale of zoned decimal number (scale << 8 — precision)

## TCVPD2N

TCVPD2N converts a packed decimal value into an internal numeric. The length of the numeric is returned.

### Syntax

```

int TCVPD2N
(
    unsigned char *pdp,
    int          pas,
    unsigned char *nump
);

```

*Parameters*

<b>nump</b>	Internal numeric value (returned).
<b>pdp</b>	Packed decimal value.
<b>pas</b>	Precision and scale of decimal number (scale << 8 — precision)

**TCVS2D**

TCVS2D converts a character string to an internal date-time. The length of the numeric is returned upon success; otherwise false is returned.

*Syntax*

```
int TCVS2D
(
  unsigned char *dt,
  int dt_len,
  char *maskp,
  int mask_len,
  char *bufp,
  int buf_len
);
```

*Parameters*

<b>dt</b>	Internal date-time value (returned)
<b>dt_len</b>	Size of internal date-time buffer. Values are 4 - date, 7 - datetime, 10 - timestamp.
<b>maskp</b>	Character string mask.
<b>mask_len</b>	Length of maskp buffer.
<b>bufp</b>	Character string.
<b>buf_len</b>	Length of bufp buffer.

**TCVS2IB**

TCVS2IB converts a character string to an integer. Returns true upon success.

*Syntax*

```
int TCVS2IB
(
  char *bufp,
  int buf_len,
  int *ip
);
```

### *Parameters*

- bufp** Character string buffer.
- buf\_len** Length of bufp buffer.
- ip** Integer error code value (returned).
- max\_int — overflow
  - 0 — non-digit found

### *Notes*

Care must be taken to ensure that ip points to a correctly aligned integer address.

## **TCVS2N**

TCVS2N converts a character string to an internal numeric. The length of the numeric is returned.

### *Syntax*

```
int TCVS2N
(
    int          ip_len,
    char         *ip,
    unsigned char *nump
);
```

## TCVU42N

TCVU42N converts a four byte unsigned integer into an internal numeric. The length of the numeric is returned.

### *Syntax*

```
int TCVU42N
{
    unsigned int uu4;
    unsigned char *nump;
};
```

### *Parameters*

**uu4** Length of character string.  
**nump** Numeric value (returned).

## TCVU82N

TCVU82N converts an eight-byte unsigned integer into an internal numeric. The length of the numeric is returned. Only applicable to platforms that support eight byte integers.

### *Syntax*

```
int TCVU82N
{
    unsigned long long uu8;
    unsigned char *nump;
};
```

### *Parameters*

**uu8** Character string.  
**nump** Numeric value (returned).

## TCVUNIC

TCVUNIC converts character strings between various Unicode formats. All lengths are in bytes. The length of the converted string is returned.

### *Syntax*

```
unsigned int TCVUNIC
{
    int sdtty;
    unsigned int slen;
    unsigned char *sp;
    int tdtty;
    unsigned int tlen;
    unsigned char *tp
```

```
int *rcp;  
};
```

### *Parameters*

**sdty** Source datatype. One of TDT\_CHAR, TDT\_UCS2, TDT\_UTF8, TDT\_UTF16.

**slen** Length of the source string.

**sp** Pointer to the source string.

**tdty** Target datatype.

**tlen** Length of the target buffer.

**tp** Pointer to the target buffer.

**rcp** Pointer to the returned status. One of 0 - Ok, 1 - Truncated, (-1) - Bad data.

## TCVZD2N

TCVZD2N converts a zoned decimal value into an internal numeric. The length of the numeric is returned.

### *Syntax*

```
int TCVZD2N  
(  
    unsigned char *zdp,  
    int pas,  
    unsigned char *nump  
);
```

### *Parameters*

**zdp** Packed decimal value.

**pas** Precision and scale of decimal value (scale << 8 — precision).

**nump** Internal numeric value (returned).



```

    short SQLERRML;                /* length of error message */
    chad  SQLERRMC[70];           /* the actual error message */
} SQLERRM;
int  SQLERRD[6];                /* uses 2,3,4,5 only */
struct {
    char  SQLWARN0[1];
    char  SQLWARN1[1];
    char  SQLWARN2[1];
    char  SQLWARN3[1];
    char  SQLWARN4[1];
    char  SQLWARN5[1];
    char  SQLWARN6[1];
    char  SQLWARN7[1];
} SQLWARN;                       /* for DB2 compatability only */
char  SQLEXT[8];
} SQLCA = SQLCA_INIT;
#endif

```

## SQLDA

```

/*****
/* SQLDA structure
/*****
#ifndef chad
#ifdef USING_UNICODE
#define chad unsigned short
#else
#define chad char
#endif
#endif

#ifndef sqlllen_type
#define sqlllen_type short        /* must by 2 bytes [un]signed */
#endif

typedef struct sqlvar {
    short      sqltype;           /* datatype */
    sqlllen_type sqlllen;        /* length */
    unsigned char *sqldata;      /* address of data */
    short      *sqlind;          /* address of indicator variable */
    struct sqlname {
        short      length;       /* length of name of data */
        chad      data[30];      /* name of data */
    } sqlname;
} SQLVAR;

typedef struct sqllda {
    chad      sqldaid[8];        /* SQLDA identifier */
    int       sqldabc;           /* SQLDA length */
    short     sqln;              /* # of SQLVARs allocated */
    short     sqld;              /* # of SQLVARs used */
    SQLVAR    sqlvar[1];        /* array of descriptors */
} SQLDA;

```

**VORTEX.H**

```

/*
\begin{verbatim}
*****
*
*                               Copyright (C) TRIFOX, Inc. 1992
*   Module Name = VORTEX
*   Description = VORTEXchannel defines, declarations, and macros
*   Author      = LNB
*   Date       = 05/05/92
*
*****/
#ifndef ib
/*****/
/* Some generic typedefs and defines */
/*****/
#define __(X) ()

#ifdef __370__
#define SIGNED signed
#endif
#ifdef USING_UNICODE
#define chad      unsigned short      /* char (defined) */
#define chud     unsigned short      /* unsigned char (defined) */
#else
#define chad      char                 /* char (defined) */
#define chud     unsigned char        /* unsigned char (defined) */
#endif
#ifndef SIGNED
#ifdef __STDC__
#define SIGNED signed
#else
#define SIGNED
#endif
#endif
#define i1  SIGNED char                /* 1 byte signed */
#define i2  short                      /* 2 byte signed */
#define i4  int                        /* 4 byte signed */
#define ib  int                        /* best int (loop variables) */
#define u1  unsigned char              /* 1 byte unsigned */
#define u2  unsigned short            /* 2 byte unsigned */
#define u4  unsigned int              /* 4 byte unsigned */
#define ub  unsigned int              /* best unsigned int */
#define f4  float                     /* 4-byte floating-point */
#define f8  double                    /* 8-byte floating-point */

#ifdef __alpha
#ifdef VMS
#define i8  long long                 /* 8 byte signed */
#define u8  unsigned long long       /* 8 byte unsigned */
#else
#define i8  long                      /* 8 byte signed */
#define u8  unsigned long            /* 8 byte unsigned */
#endif
#else
#ifdef SYSTEM5
#define i8  long long                 /* 8 byte signed */

```

```

#define u8 unsigned long long          /* 8 byte unsigned          */
#endif
#ifdef _MSWIN
#define i8 __int64                    /* 8 byte signed           */
#define u8 unsigned __int64          /* 8 byte unsigned         */
#endif
#endif

#define ident_len 30                  /* max identifier length    */

typedef char ident[ident_len];

/*****
/* Datatypes
*****/
#ifndef TDT_INTEGER
#define TDT_INTEGER 0                 /* integer                  */
#define TDT_CHAR 1                   /* character                 */
#define TDT_NUMBER 2                 /* internal numeric        */
#define TDT_NTCHAR 3                 /* null terminated string   */
#define TDT_PACKED 4                 /* packed decimal          */
#define TDT_ZONED 5                  /* zoned decimal           */
#define TDT_FLOAT 8                  /* floating point number    */
#define TDT_VARCHAR 9                /* var char                 */
#define TDT_BLOB 10                  /* blob (binary large object) */
#define TDT_CLOB 11                  /* clob (character large object) */
#define TDT_DATETIME 12              /* date time                */
#define TDT_BIT 66                   /* ODBC bit (SQL_C_BIT)    */
#define TDT_VARBLOB 80               /* varlen blob              */
#define TDT_UTF8 81                  /* Unicode char UTF-8 format */
#define TDT_UTF16 82                 /* Unicode char UTF-16 format */
#define TDT_UCS2 83                  /* Unicode char UCS-2 format */
#define TDT_UCS4 84                  /* Unicode char UCS-4 format */
#define TDT_UTF8LOB 85               /* Unicode LOB UTF-8 format */
#define TDT_VARUTF8 86               /* Unicode varchar UTF-8 format */
#define TDT_VARUTF16 87              /* Unicode varchar UTF-16 format */
#define TDT_VARUCS2 88               /* Unicode varchar UCS-2 format */
#define TDT_VARUCS4 89               /* Unicode varchar UCS-4 format */
#define TDT_ROWID 98                 /* ROWID/DBKEY/TID         */
#define TDT_BINARY 99                /* binary                   */
#endif
#endif

/*****
/* DB engine IDs
*****/
#define TDB_DID_ORACLE 0              /* Oracle                   */
#define TDB_DID_RDB 1                 /* DEC's Rdb                 */
#define TDB_DID_SYBASE 2              /* Sybase                   */
#define TDB_DID_NET 3                 /* Network                   */
#define TDB_DID_GENESIS 4             /* Genesis                   */
#define TDB_DID_INFORMIX 5            /* Informix's RDBMS         */
#define TDB_DID_MUX 6                 /* MUX                       */
#define TDB_DID_DB2 7                 /* IBM's DB2                 */
#define TDB_DID_OLEDB 8               /* OLE DB                    */
#define TDB_DID_ADABASC 9             /* Software AG's ADABAS     */
#define TDB_DID_ADABASD 10            /* Software AG's SQLDB      */

```

```

#define TDB_DID_ODBC          11          /* ODBC */
#define TDB_DID_SQLSERVER    12          /* Microsoft's SQL Server */
#define TDB_DID_MARIADB      13          /* Mariadb */
#define TDB_DID_MYSQL        14          /* MySQL */
#define TDB_DID_POSTGRESQL   16          /* PostgreSQL */
#define TDB_DID_TRIM         17          /* TRIMpl */

#define TDB_DID_MAX          16
#define TDB_DID_MASK         0xFF        /* for masking out the DID */

/*****
/* DB driver commands
*****/
#define TDB_CMD_TIMEOUT      0          /* resource timeout */
#define TDB_CMD_SINGLERPID   1          /* single transaction pid (Sybase)*/
#define TDB_CMD_OOPT         2          /* OCI oopt call (Oracle7) */
#define TDB_CMD_USEADB       3          /* USE DATABASE (Sybase) */
#define TDB_CMD_LANGUAGE    4          /* language flag (Oracle) */
#define TDB_CMD_RAW_DATETIME 5          /* Datetime is returned untouched */
#define TDB_CMD_VERSION      6          /* place version # in message buf */
#define TDB_CMD_CHAR         7          /* char datatype value (Oracle) */
#define TDB_CMD_LOCK         8          /* lock SLAVE on next stmt (MUX) */
#define TDB_CMD_ONEBYTE      9          /* blank varchar handling (Sybase)*/
#define TDB_CMD_MULTISTMT    10         /* batch statements (Sybase) */
#define TDB_CMD_AUTOCOMMIT   11         /* autocommit on/off (Genesis, ODBC)*/
#define TDB_CMD_DORMANT      12         /* connection dormant (Informix) */
#define TDB_CMD_GETLDA       13         /* return LDA pointer (Oracle7) */
#define TDB_CMD_RAISE        14         /* do a raise() (driver 99 only) */
#define TDB_CMD_ERROR        15         /* return error (driver 99 only) */
#define TDB_CMD_TMP_RELEASE  16         /* temporary release (vtxwebd) */
#define TDB_CMD_WORM_AUTH    17         /* Genesis/WORM authentication */
#define TDB_CMD_TIMEOUT_QUERY 18        /* query timeout (Genesis/SDMS) */
#define TDB_CMD_TIMEOUT_FETCH 19        /* fetch timeout (Genesis/SDMS) */
#define TDB_CMD_DYN_CURSOR   20        /* use dynamic cursors (ODBC) */
#define TDB_CMD_SYNTAX_LEVEL 21        /* GENESIS syntax level (ODBC) */
#define TDB_CMD_RETURN_ROWID 22        /* Return ROWID on insert */
#define TDB_CMD_NEW_BLOBS    23        /* use new Oracle blobs */
#define TDB_CMD_RO_CURSOR    24        /* set cursors readonly (various) */
#define TDB_CMD_CURSOR_TYPE  25        /* cursor type (ODBC) */
#define TDB_CMD_TXN_ISOLEVEL 26        /* xaction isolation level (ODBC) */
#define TDB_CMD_SCROLL       27        /* set cursor scroll option */
#define TDB_CMD_HOSTNAME     28        /* get host name */
#define TDB_CMD_FQCOLNAME    29        /* fully qualified column names */
#define TDB_CMD_FREE_LOCKS   30        /* free/release locks (ODBC) */
#define TDB_CMD_NOUNICHAR    31        /* desc W[VAR]CHAR as CHAR (ODBC) */
#define TDB_CMD_ZEROLEN_NULL 32        /* make 0 len strings NULL (SS) */
#define TDB_CMD_EXIO_RS      33        /* EXIO can return a resultset */
#define TDB_CMD_DRV_CONFIG   34        /* set up SSL params (VTX3 only) */
#define TDB_CMD_BLK_MIN      35        /* min rows for bulk insert (PSQL)*/
#define TDB_CMD_MAX_COL_LEN  36        /* max col length (PSQL) */
#define TDB_CMD_SET_CON_ATTR 37        /* SQLSetConnectAttr "attr value" */
#define TDB_CMD_GET_CON_ATTR 38        /* SQLGetConnectAttr "attr" */
#define TDB_CMD_DESC_PARMS   39        /* Use SQLDescribeParam for parms */
#define TDB_CMD_DESC_LOBS    40        /* describe LOB columns */
#define TDB_CMD_GET_SPROC_RV 41        /* get stored proc return value */
#define TDB_CMD_DESC_NULL    42        /* actual/dflt nullability (PSQL) */
#define TDB_CMD_NOTUSED      0xFF       /* used to check connection */

```

```

#define TDB_CMD_SCROLL_NEXT      0          /* cursor scroll next          */
#define TDB_CMD_SCROLL_PRIOR     1          /* cursor scroll prior         */
#define TDB_CMD_SCROLL_FIRST     2          /* cursor scroll first         */
#define TDB_CMD_SCROLL_LAST      3          /* cursor scroll last          */
#define TDB_CMD_SCROLL_CURRENT   4          /* cursor scroll current       */
#define TDB_CMD_SCROLL_ABSOLUTE  5          /* cursor scroll absolute      */
#define TDB_CMD_SCROLL_RELATIVE  6          /* cursor scroll relative      */
                                        /* (position passed as second */
                                        /* integer after option)      */

/*****
/* Additional return status. If SQLCODE == 100 and SQLERRD[0] == TDD_ARS_ROWS,*/
/* then there is a new resultset pending.                                     */
*****/
#define TDB_ARS_ROWS              1          /* new rows                   */
#define TDB_ARS_COMPUTE           2          /* compute rows               */

/*****
/* DB driver file (piggy-back) commands                                     */
*****/
#define TDB_FILE_OPEN             0          /* open a file                */
#define TDB_FILE_CLOSE           1          /* close a file               */
#define TDB_FILE_READ            2          /* read from a file           */
#define TDB_FILE_WRITE           3          /* write to a file            */
#define TDB_FILE_DELETE          4          /* delete a file              */

/*****
/* Various VORTEXcli (tdbprep) definitions                                 */
*****/
#define TDB_PRE_what "@(T)PCAPI 3.3.2.76" /* pre-compiler version      */
#define TDB_PRE_VERSION         "3.3.2.76" /* pre-compiler version      */
#define TDB_PRE_W32_VER         3,3,2,76 /* pre-compiler version      */
#define TDB_PRE_VER_NUM         332      /* pre-compiler version #    */

#ifndef TDB_PRE_MAXCONN
#define TDB_PRE_MAXCONN         64          /* pre-compiler max # connections */
#endif

#define TDB_PRE_POPEN            0x0001     /* is a positioned OPEN      */
#define TDB_PRE_PEXEC           TDB_PRE_POPEN /* is a positioned EXEC     */
#define TDB_PRE_FUO             0x0002     /* statement has "FOR UPDATE OF" */
#define TDB_PRE_BLOB            0x0004     /* statement involves BLOBs  */
#define TDB_PRE_SINGSEL         0x0008     /* singleton SELECT (allows 1 row)*/
#define TDB_PRE_SCROLL          0x0010     /* scrollable cursor         */
#define TDB_PRE_STATIC          0x0020     /* static scrollable cursor   */
#define TDB_PRE_DYNAMIC         0x0040     /* dynamic scrollable cursor  */
#define TDB_PRE_INSENSITIVE     0x0080     /* insensitive scrollable cursor */
#define TDB_PRE_ASENSITIVE      0x0100     /* asensitive scrollable cursor */

#define TDB_ERR_SINGSEL         (-2000000) /* more than one row returned */

#define TDB_EMAP_OK             0          /* all is ok                  */
#define TDB_EMAP_NOFILE        1          /* file could not be opened   */
#define TDB_EMAP_BADFILE       2          /* bad format in map file     */
#define TDB_EMAP_NOMEM         3          /* out of memory              */

```

```

#define TDB_GENESIS_SYNTAX_LEVEL 1          /* ret'ed by TDB_CMD_SYNTAX_LEVEL */

/*****
/* host variable descriptor. (used by VORTEXcli)
/* NOTE! This structure matches the first 4 fields in sqlvar in the sqllda
/* *****/
typedef struct tdb_hvar {
    i2          TDB_DTY;          /* data type
    u2          TDB_LEN;          /* column length
    u1          *TDB_VAP;          /* address of variable
    i2          *TDB_IVP;          /* address of indicator variable
} TDB_HVAR;

/*****
/* Extended host variable descriptor. (used by VORTEXcli)
/* NOTE! The first 4 fields match TDB_HVAR
/* *****/
#define TDB_FLG_IN  0x01          /* input variable
#define TDB_FLG_OUT 0x02          /* output variable
#define TDB_FLG_RV  0x04          /* sproc return value variable
#define TDB_FLG_DIR(A) ((A) & (TDB_FLG_IN | TDB_FLG_OUT))

typedef struct tdb_hva2 {
    i2          TDB_DTY;          /* data type
    u2          TDB_LEN;          /* column length
    u1          *TDB_VAP;          /* address of variable
    i2          *TDB_IVP;          /* address of indicator variable
    u1          TDB_FLG;          /* flags
    i1          TDB_VNL;          /* length of variable name
    u2          TDB_CNT;          /* array count
    char        *TDB_VNP;          /* address of variable name
} TDB_HVA2;

/*****
/* VTX behaviour options. Use VTXOPTS() to get/set these options.
/*
/*
/* Field descriptions:
/*
/* TDB_DTB - Datetime base.
/*
/* This number is SUBTRACTED from a datetime value when it's
/* fetched into a long output variable. By default (i.e., when
/* TDB_DTB is 0), a datetime value fetched into a long output
/* variable will be the number of days since the beginning of AD 1.
/* To change the base of counting to a different date, set TDB_DTB
/* to the number of days between the selected date and the start of
/* AD 1. For example, to use the Informix datetime base, set TDB_DTB*
/* to 693959. Note that a positive number moves the date base
/* forward to a later (AD) year, and a negative number moves it back*
/* to a BC year.
/* Default: 0
/*
/*
/* TDB_DTF - Datetime format mask. (Note! must be null terminated)
/*
/* Specifies the format of the datetime value when fetched into a
/* character buffer.
/* Default: "DD-MON-RR"
/*
/*
/* TDB_NMK - NULLs mask.

```

```

/*          When a column is described and NULLs are allowed this value is      */
/*          bit ORD to the datatype field.                                     */
/*          Default: 0                                                         */
/*                                                                                   */
/*****
typedef struct tdb_opts {
    i4          TDB_DTB;                /* datetime base          */
    chad       TDB_DTF[64];            /* datetime format       */
    i2          TDB_NMK;                /* NULL mask             */
} TDB_OPTS;

/*****
\end{verbatim}
*/

```

## VORTEX.X

```

/*
\begin{verbatim}
*****
*                                                                                   Copyright (C) TRIFOX, Inc. 1993
*   Module Name = VORTEX
*   Description = VORTEXchannel API external function declarations
*   Author      = LNB
*   Date        = 09/07/93
*
*****/
#ifdef m_thread
#define db_hdl void *
#else
#define db_hdl int
#endif

int VTXGDID(db_hdl);
int VTXGEEM(db_hdl,void *,int);
int VTXEMAP(int,int,int,int *,void *);

unsigned int VTXBLOB(db_hdl, void *, void *, int, void *, unsigned int);
unsigned int VTXLOBG(db_hdl, void *, void *, int, int, void *,
                    unsigned int, unsigned int);
unsigned int VTXLOBP(db_hdl, void *, void *, int, int, void *, unsigned int);

void VTXINIT(db_hdl,void *,int,int,int,int,int);
void VTXINI2(db_hdl,void *,int,int,int,int,int,int,void *);
void VTXCONN(db_hdl,void *,void *,int,int);
void VTXREL (db_hdl,void *);
void VTXCAN (db_hdl,void *);
void VTXCOMM(db_hdl,void *,int);
void VTXROLL(db_hdl,void *,int);
void VTXEXEC(db_hdl,void *,void *,void *,int,void *,int,int,int,int);
void VTXEXIO(db_hdl,void *,void *,void *,int,void *);
void VTXOPEN(db_hdl,void *,void *,void *,int,void *,int,int);
void VTXMOVE(db_hdl,void *,void *,void *,int,int,int);
void VTXGLEN(db_hdl,void *,void *,int *);
void VTXCLOS(db_hdl,void *,void *,int);
void VTXDESC(db_hdl,void *,void *,void *);

```

```
void VTXDES2(db_hdl,void *,void *,void *);
int  VTXDES3(db_hdl,void *,void *);
void VTXDES4(db_hdl,void *,void *,void *);
void VTXCMD (db_hdl,void *,void *,int,void *);

void VTXOPTS(int,void *);

/*****
\end{verbatim}
*/
```

---

A  
address  
    variable name's buffer 2  
array count 2  
B  
behavior  
    getting/setting 33  
binary datatype 3  
BLOB  
    datatype 2  
    fetching 25  
    putting/getting 6  
buf\_len  
    VTXBLOBparameter 6, 23, 24  
buf\_ptr  
    VTXBLOB parameter 6, 23, 24  
buffer  
    address 2  
C  
cancelling  
    database requests 7  
changing result sets 25  
character  
    converting from datetime 4  
    datatype 2  
    variable length datatype 2  
character address 2  
CLOB  
    datatype 2  
    fetching 25  
    putting/getting 6  
closing  
    logical cursor 8  
COBOL 2  
column  
    VTBLOB parameter 6, 23, 24  
commands  
    database driver-specific 9  
committing  
    transaction 13  
COMPUTE rows 25  
connect string  
    VTXCONN 14  
connect\_len  
    VTXCONN parameter 14

---

connecting  
  to a database 14

connections  
  gets identification 31

converting datatypes 3

count  
  array 2

cursor  
  closing logical 8  
  hard close 8  
  opening for SELECT 27  
  parameter 5

D

data  
  moving 25

database  
  cancelling requests 7  
  connecting 14  
  disconnecting connection 28  
  error codes 30  
  identifying 5

databases  
  getting error messages 32  
  identifying connections 31  
  initializing instances 21

datatype  
  BLOB 2  
  character 2  
  CLOB 2  
  host variable 1  
  integer 2  
  packed decimal 2  
  varblob 3  
  variable length character 2  
  zoned decimal 2

datatype floating point 2

datatypes 2  
  BLOB 6  
  CLOB 6  
  converting 3

datetime  
  converting to character 4  
  converting to integer 4  
  internal 2

DB\_index 5

---

multi-threaded 5, 21

decimal

- packed 2
- zoned 2

default\_code

- VTXEMAP parameter 30

describing

- SELECT 15, 16
- SQL statement 18
- SQL statements 18

disconnecting

- database connection 28

drivers

- executing commands for 9

E

error messages

- getting 32

errors 4

- mapping codes 30

executing

- non-SELECT statement 19
- stored procedure 20

executing commands

- driver-specific 9

F

fetch

- see moving

float4 datatype 3

float8 datatype 3

floating point

- datatype 2

format mask 4

functions

- data-moving 5

G

getting

- database error messages 32

H

handling errors 4

hard

- VTXCLOS parameter 8

host variable

- datatype 1
- length 1
- pointer 1

---

hostvar\_cnt  
    VTXEXIO parameter 20

hostvarp  
    VTXEXIO parameter 20

I

identifying  
    database connections 31

indicator variable  
    pointer 1

initializing  
    database instance 21  
    SQLDA 18

input variable  
    indicator 2

int1 datatype 3

int2 datatype 3

int4 datatype 3

integer  
    converting from datetime 4  
    datatype 2

internal datetime 2

internal number datatype 2

L

large objects  
    see BLOB or CLOB

length  
    host variable 1  
    variable name 2

M

map\_file  
    VTXEMAP parameter 30

mapped\_code  
    VTXEMAP parameter 30

mapping  
    database error codes 30

messages  
    multiple lines 32

moving data 25

multi-threaded  
    DB\_index 5, 21

N

not\_used  
    VTXCONN parameter 14

null mask bit  
    set 18

---

numeric  
    Sybase datatypes 3

O

opening  
    SELECT cursor 27

option  
    VTXEXEC parameter 19

options  
    VTXOPEN parameter 27

Oracle-compatible 2

Oracle-compatible datatypes 2

output variable  
    indicator 2

P

packed decimal  
    datatype 2

parameters  
    cursor 5  
    DB\_index 5  
    sqlca 5

pointer  
    host variable 1  
    indicator variable 1

precision  
    determining 3

R

rbuf\_len  
    VTXIPC parameter 22

rbufp  
    VTXIPC parameter 22

record\_cnt  
    VTXEXEC parameter 19  
    VTXMOVE parameter 25

record\_len  
    VTXEXEC parameter 19  
    VTXMOVE parameter 25

rollback  
    transactions 29

S

sbuf\_len  
    VTXIPC parameter 22

scalars 2

scale  
    determining 3

SELECT

---

opening cursor 27  
SELECT statement  
    describing 15, 16  
sending messages 22  
sqdabc 18  
SQL  
    describing statement 18  
SQL statement  
    describe 18  
SQLCA 1, 5  
SQLCA.SQLCODE 4, 30  
SQLCA.SQLERRD 25  
sqlca.sqlerrd 19  
SQLDA 1  
sqlda  
    VTXDES2 parameter 15, 16, 17  
    VTXDESC parameter 18  
sqldaid 18  
sqlen 3  
sqln 18  
sqlp  
    VTXEXEC parameter 19  
    VTXEXIO parameter 20  
SQLServer 25  
sqlvar\_cnt  
    VTXEXEC parameter 19  
    VTXMOVE parameter 25  
    VTXOPEN parameter 27  
sqlvar\_len  
    VTXEXEC parameter 19  
    VTXMOVE parameter 25  
    VTXOPEN parameter 27  
sqlvarp  
    VTXEXEC parameter 19  
    VTXMOVE parameter 25  
    VTXOPEN parameter 27  
start\_write 29  
    VTXCOMM parameter 13  
stored procedures  
    calling 19  
    with variables 20  
structures  
    used for communication 1  
Sybase 25  
Sybase numeric types 3

---

## T

- TDB\_CNT 2
- TDB\_DTB 4
- TDB\_DTF 4
- TDB\_DTY 1
- TDB\_FLG 2
- TDB\_HVA2 2
  - addressing 20
  - counting 20
- TDB\_IVP 1
- TDB\_LEN 1
- TDB\_OPTS 34
- TDB\_PRE\_MAXCONN 5
- TDB\_VAP 1
- TDB\_VAR 1
- TDB\_VNL 2
- TDB\_VNP 2
- TIPMAIN 22
- transactions
  - committing 13
  - rollback 29

## V

- varblob
  - datatype 3
- variable
  - input or output 2
- variable length character 2
- variable name
  - buffer address 2
  - length 2
- variables
  - in stored procedures 20
- vortex.h 5
- VORTEXcli
  - definition 1
- VTXBLOB 6
- VTXCAN 7
- VTXCLOS 8
- VTXCMD 9
- VTXCOMM 13
- VTXCONN 14
- VTXDES2 15
- VTXDES3 16
- VTXDES4 17
- VTXDESC 18

---

VTXEMAP 30  
VTXEXEC 19  
VTXEXIO 2, 20  
VTXGDID 31  
VTXGEEM 32  
    retrieving message or status 20  
VTXINIT 21  
VTXIPC 22  
VTXLOBG 23  
VTXLOBP 24  
VTXMOVE 25  
VTXOPEN 27  
VTXOPTS 4, 33  
    null mask bit 18  
VTXREL 28  
VTXROLL 29  
W  
warnings 4  
wild query  
    interrupting 7  
Z  
zoned decimal 2  
    datatype 2