# VORTEXaccelerator
# Users Guide

May 30, 2017

Trifox Inc.
2959 Winchester Blvd
Campbell, CA 95008

www.trifox.com

## Trademarks

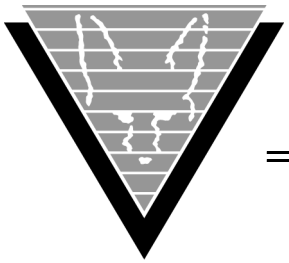
TRIMapp, TRImpl, TRIMqmr, TRIMreport, TRIMtools, GENESISsql, DesignVision, DVapp, DVreport, VORTEX, VORTEXcli, VORTEXc, VORTEXcobol, VORTEXperl, VORTEXjdbc, VORTEX++, VORTEXJava Edition, LIST Manager, VORTEXodbc, VORTEXnet, VORTEXclient/server, VORTEXaccelerator, VORTEXreplicator are all trademarks of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

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

## Copyright



Printed in the U.S.A.



# Contents

# Preface

The VORTEXaccelerator User Guide explains how to use VORTEXaccelerator, Trifox's transaction accelerator and monitor, to improve both the performance and resource consumption of your applications that VORTEX.

This guide does not discuss managing the database; for instructions and information about database management procedures, read your database vendor's documentation.

This guide is designed for the DBA who has some experience with large systems and understands the memory and CPU resource issues involved with hundreds of client applications running against a relational database.

## Organization

This document is a guide for using VORTEXaccelerator. It tells you where to find installation and troubleshooting information and provides suggestions on how to use VORTEXaccelerator to improve your application and system performance.

This document is divided into the following chapters:

- ***Introduction*** — Discusses basic concepts of database access, large system resource problems, and the VORTEXaccelerator solution.
- ***Cofiguration and Setup*** — Describes how to configure and set up the machines to successfully run the VORTEXaccelerator,VORTEXmonitor, and VORTEXdaemon.
- ***Using VORTEXaccelerator*** — Describes how to start, run, and stop VORTEXaccelerator.
- ***Monitoring and Tuning Activity*** — Describes the features of VORTEXmonitor.

Appendixes contain the settings in `mux.ini` and complete listings of VORTEX environment variables.

## 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. |
| *`connect_string`* | Replace italicized text with your own variable. |
| **`vtxnetd`** | 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

### *1 December 1999*

Added introductory information about cursors and connections to database in application development.

### *20 January 2000*

Added information about address mapping for shared memory, page 10.

### *09 November 2015*

Correct log filename to ttc_<name>.log in Chapter 3, Command Summary.

# Chapter 1 Introduction

VORTEXaccelerator is a transaction accelerator and monitor that can help you improve both the performance and resource consumption of applications that use VORTEX.

If most users in a system are running the same applications against the same databases, sharing cursors between users is a simple way to increase database efficiency.

VORTEXaccelerator enables this cursor sharing so a large number of application users are represented by a smaller number of actual database users. VORTEXmonitor allows you to monitor statistics as well as manage slaves and clients.

## What are Cursors?

Cursors, specifically database cursors, are the elements that databases use to point to the data that meets the application's request. Cursors are created by the database, and usually exist only as long as it takes for the database to respond to the application. When the application makes another request, the database creates another cursor for that job and closes it when the request is completed.

As you can imagine, when the application is interactive the database can be very busy creating cursors and closing them. In fact, interactive applications most often ask for information from a database. This information is translated into what relational databases call SQL SELECT statements. Creating the cursor is the most resource-intensive part of the entire process.

## Database Perspective

A database's relationship with an application is momentary: the application requests some service and the database tries to provide it -- without any context to optimize the process. Beyond the statements they receive from client applications, databases have no way of knowing what kind of work the clients are doing, and they make no assumptions.

VORTEXaccelerator, however, can make good "guesses" about the kind of work the applications are doing, and thus optimize the environment for that work.

In batch application where vast amounts of data are processed serially, an extra open cursor does not seem particularly important. But in an interactive application, such as online order entry, especially in a real-time environment, most activities involve user data validation. A single order, can require hundreds of simple validations such as verifying the availability of an item in inventory, the user's permission to order it, and any special circumstances. The database only sees the SELECT, and has no way of knowing which activity the request is associated with.

The database goes through many steps in responding to a simple SELECT statement. First, the database parses the statement to check that the syntax (language) is correct and that it makes sense. Then it must make sure that any tables or columns in the request actually exist. The database is also responsible for making sure that the user making the request is authorized to get information from the tables or columns in the statement. And

finally, the database must determine how to access the data and retrieve it most efficiently.

Only after all these steps have been completed does the database open a cursor and replace the variables in the request with the actual table and column names. The last steps, fetching the data and then closing the cursor complete the transaction.

The first step, which involves many disk operations, is expensive in terms of database resources. Depending on the database engine, the cost can be as high as 10 times more than all the other combined processing steps.

# How It Works

Consider a scenario in which 100 people are running the same set of order entry applications. If each application requires between 50-100 open cursors at any given time, even the most capable database engine's performance will suffer. Assuming that each cursor uses 10KB, a worst-case scenario could easily consume up to 100MB for cursor memory alone.

Since most users are running the same application most of the time, VORTEXaccelerator offers the perfect solution: sharing cursors between users saves database resources and preserves the engine's performance so all 100 users are able to order items efficiently.

How VORTEXaccelerator works:

1. The main process, called `vtxmux`, starts.
2. This "transaction cop" initializes a portion of shared memory that contains a control area, SQL hash table, and client and slave work areas (all user-configurable).
3. When an application (which has already been linked with the VORTEXaccelerator libraries) connects to a database, vtxmux takes control of it as a client.
4. To run a SQL statement, the client sends a request to vtxmux, which assigns the client to a database slave. If it can't find an available slave or start a new one, it places the client in a "first come, first served" queue.
5. When the client's request has been served by the database, the slave determines if the client/slave connection should be broken. For INSERT, UPDATE, or DELETE statements, vtxmux maintains the connection until a COMMIT or ROLLBACK is performed. Otherwise, for a SELECT statement, the database slave breaks the connection.

VORTEXaccelerator also includes a tool to monitor statistics as well as allow a database administrator to free and/or kill slaves and clients.

# Summary

Reusing cursors is a straightforward process: Once the first step is completed, the following transactions use the open cursor and simply begin fetching the data, binding the parameters as necessary. The cursor is closed only when an application exits or the cursor is required for another statement.

Even though well-written applications usually know how to use existing cursors, not all of them take advantage of the facility. When many users are running the same applications independently on different machines, and you just don't have the time to re-engineer all the applications, getting the most out of cursor-reuse requires an independent manager like VORTEXaccelerator.

# VORTEXaccelerator in Multi-Tier Environments

VORTEXaccelerator fits into any existing client/server configuration with ease. Running on Windows, Unix, or VMS, you can boost the performance of your database application with a very simple installation.

The illustrations specify the program names for Windows systems, but the configuration is the same for Unix machines.

## Local Access

This example illustrates a single-machine configuration, specifically Windows connecting to an Oracle database. The sample `mux.ini` file shows settings for a slave group connecting to VORTEXaccelerator with the `vtxapi32.dll`.

### *Sample file*

```
rem -------------------------- VORTEXaccelerator specifics
log_directory           C:\Program
Files\Trifox\VORTEXaccelerator\LOG\
dflt_db_id              0
connect0                scott/tiger
dll0                    vtx0
slave0                  vtxslav
```

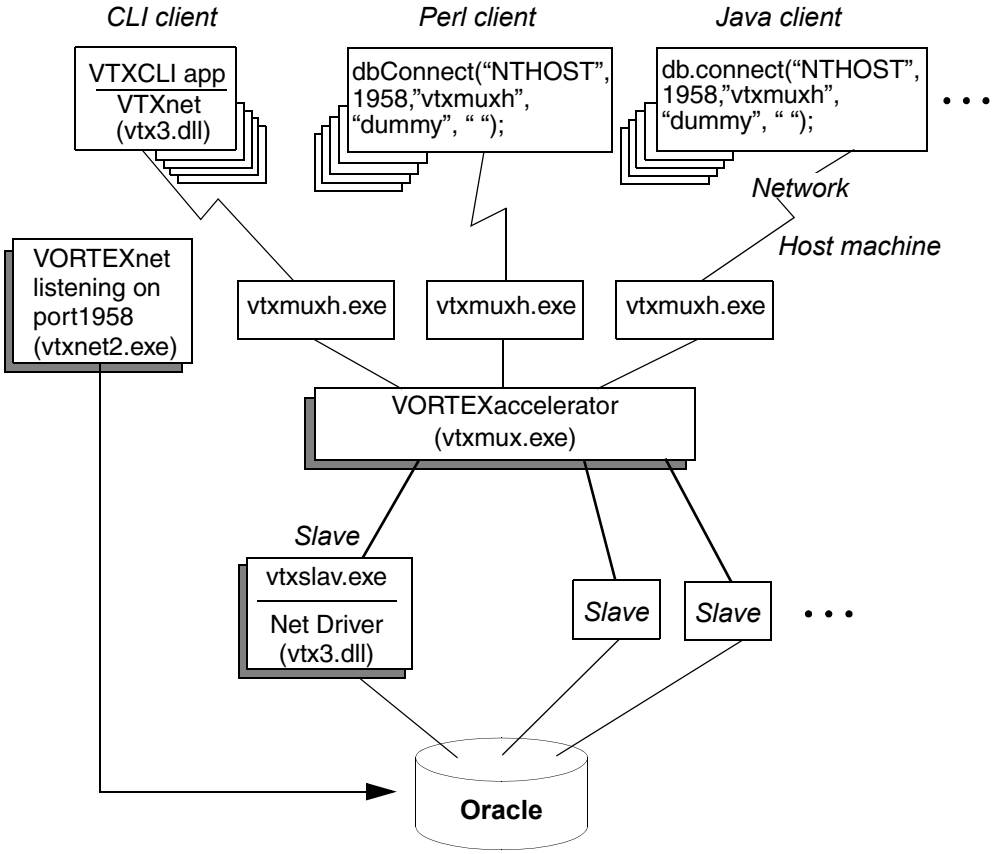
For complete descriptions of each initialization file parameter, consult "***Initialization File***" on page 32.

*The VORTEXaccelerator starts slaves as necessary to provide service to the clients at any given time. In this "snapshot," the VORTEXaccelerator has three slaves running to process the client requests.*

## Remote Access

This example illustrates a two-machine configuration, specifically a Windows computer connecting to another Windows computer, which hosts the Oracle database.

The sample `mux.ini` file that follows shows settings for four clients running the same program and all connecting to VORTEXaccelerator with the `vtxapi32.dll` but using VORTEXnet to reach the database.

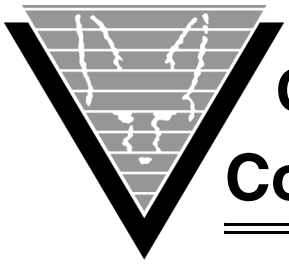



*VORTEXnet acts as a request broker for the clients by starting the vtxmux processes when a connection is requested. Thereafter, the clients connect directly to the VORTEXaccelerator by way of the* `vtxmuxh.exe`.

### *Sample files*

The VORTEXcli client uses the information in the `mux.ini` and `net.ini` files.

```
rem MUX.INI
rem -------------------------- VORTEXaccelerator specifics
log_directory    C:\Program Files\Trifox\VORTEXaccelerator\LOG\
dflt_db_id       0
connect0         scott/tiger
dll0             vtx0
slave0           vtxslav
rem NET.INI
rem ---------------------------- TRIM/NET specifics
hostnamesvc0            NT2!VTX0
packetsize              8192
port                    1958
```



# Chapter 2
# Configuration and Setup

Before you begin using VORTEXaccelerator, you must install the components according to the release notes in the installation package. Then, you set up environment variables, edit initialization files, and if appropriate, create a Windows session.

Note that logins and passwords are only used by VORTEXaccelerator for database access, not for security. Data access security is controlled by the database itself, or by the authentication server, if you choose to use it.

VORTEXaccelerator has three components:

1. ***vtxmux*** — The main process that manages shared memory and coordinates the clients and database slaves.
2. ***vtxmon*** — A monitoring tool.
3. ***vtxmuxd*** — A process that watches over VORTEXaccelerator.

The actual executable names of each component depend on your computer's operating system. For example, on VMS `vtxmux` and `vtxmuxd` are batch jobs within DCL scripts. On Windows the names have an `.exe` extension. This guide uses the Unix program names — `vtxmux`, `vtxmon`, and `vtxmuxd` — in describing how to use VORTEXaccelerator components.

Take note:

1. You must be a "super user" or "root" to complete the installation.
2. VORTEXserver must already be installed on your system.

## Setting Environment Variables

Define the following three environment variables for your operating system:

1. `VORTEX_HOME` — Identifies the directory where VORTEX products are located.
2. `VORTEX_SHM_FILE` — Identifies a file used to find the VORTEXaccelerator shared memory segment. This file must be readable by any user who accesses VORTEXaccelerator.
3. `VORTEX_MUX_NAME` — Identifies the name of the VORTEXaccelerator instance and is used by client applications to find the correct entry in the `VORTEX_SHM_FILE`.

For more information on setting environment variables in different operating systems, and for complete information and examples for these variables, see "Enviroment Variables" in Chapter 5.

If you use an operating system that allows shared memory segments to be mapped to different addresses, you must configure the `VORTEX_SHM_ADDR` variable to define suitable address space.

## Address Mapping and Shared Memory

The VORTEX_SHM_ADDR file is for operating systems that allow a shared memory segment to be mapped to different addresses (for example, Solaris and Linux). Specifying this environment variable ensures that VORTEXaccelerator clients can locate the correct addresses.

Typically, you can determine if your operating system requires this file if VORTEXaccelerator comes up, but you cannot attach to its shared memory using a client program.

Use the tman utility to locate the default addresses your operating system chooses for various types of memory. VORTEX_SHM_ADDR identifies a file that contains instance-address pairs that identify the shared memory for a given instance.

### *Example*

This example line from the file designated by VORTEX_SHM_ADDR instructs VORTEXaccelerator to place the ACCEL1 instance's shared memory segment at (hex) address 0xE0000000.

```
ACCEL1  E0000000
```

## Groups

You can run several copies of vtxmux at the same time on one machine. These different copies, called groups (note that VORTEXaccelerator groups are not the same as Unix groups) allow you to isolate different types of users or applications. For example, management applications may be in one group and data entry applications in another.

Alternatively, you could but read-intensive applications in one group and update-intensive applications in another.

These instructions only include steps for a single group.

# Customizing Initialization Files

You must have a customized mux.ini file, which is typically located in the lib subdirectory of VORTEX_HOME before you can operate VORTEXaccelerator.

If you use VORTEXnet to connect VORTEXaccelerator to the database, you should also have a customized net.ini file.

# Installing VORTEXaccelerator

VORTEXaccelerator runs on Windows, Unix, and VMS and can be configured to suit your client and server resources.

### *Installing VORTEXaccelerator as a Windows service*

1. Install the VORTEX service.

   a. Type `vortex -install` on a command line.

   b. You may need to restart Windows to make the VORTEX service available to the system.

   In the control panel, change the `Startup Type` for the VORTEX service from **manual** to **automatic**.

2. Update system variables.

   In the **Control Panel**, open the **System** folder. On the **Environment** tab, ensure that the following variables appear in the **System Variables** section:

```
VORTEX_HOME=C:\Program Files\Trifox\VORTEXserver
VORTEX_MUX_NAME=MUX
VORTEX_SHM_FILE=C:\Program Files\Trifox\VORTEXserver\LIB\TRIM.SHM
VORTEX_SERVICE_FILE=C:\temp\vtxs.srv
```

3. Create the VORTEX service command file.

   Create a file called `vortex.srv` with the following entries. Make sure to end each line ***including the last one*** with an `EOL` character ( [Enter] after each line).

```
#
# Set additional environment variables needed.
#
PATH=C:\ProgramFiles\Trifox\VORTEXserver\
       BIN;C:\WINNT;C:\WINNT\SYSTEM32
TRIM_HOME=C:\Program Files\Trifox\VORTEXserver
TRIM_MUX_NAME=MUX
TRIM_SHM_FILE=C:\Program Files\Trifox\VORTEXserver\LIB\TRIM.SHM
#
# Programs (daemons) to start
#
C:\Program Files\Trifox\VORTEXserver\BIN\VTXRSHM.EXE
C:\Program Files\Trifox\VORTEXserver\BIN\VTXSHM.EXE 00000008 -s
C:\Program Files\Trifox\VORTEXserver\BIN\VTXMSG.EXE 100 32 -s
C:\Program Files\Trifox\VORTEXserver\BINVORTEXaccelerator.EXE
       MUX 64 8 64 64 4 32 log
C:\Program Files\Trifox\VORTEXserver\BIN\VTXNET2.EXE -p1958
```

4. Check services.

   Restart Windows and check in the **Task Manager** to make sure the following processes are running:

   - vtxshm.exe
   - vtxmsg.exe
   - vtxmux.exe

- vtxnet2.exe

If you have problems, check the event log and correct the errors that are reported. You reach the event log by selecting **Administrative Tools** from the Programs list, and choosing **Event Viewer** from the list that appears.

### ☞ *Installing VORTEXaccelerator on VMS*

1. Install the TRIMipc service.

   Create a GLOBAL SECTION for `trimipc` to use in emulating shared memory and IPC by typing

   ```
   trimipc /c 500 50 50
   ```

2. Verify the installation by typing

   ```
   trimipc /?
   ```

3. Start VORTEXaccelerator by typing

   ```
   @startmux
   ```

4. Verify installation.

   Start VORTEXmonitor and view the VORTEXaccelerator control area by typing

   ```
   vtxmon MUX
   ```

5. At the ==> prompt, type the command `DT`.

   You should see a display of the main VORTEXaccelerator control block.

### ☞ *Installing VORTEXaccelerator on Unix*

1. You should have downloaded a `.tar` file from `ftp.trifox.com`. That file includes instructions for proper installation for the specific Unix platform.
2. Change directory to `VORTEX_HOME` and untar the file.

# Chapter 3
# Using VORTEXaccelerator

With your environment variables and initialization files set up, all that is left is determining VORTEXaccelerator operating parameters.

Once determined, you set the values on the command line when you start VORTEXaccelerator for NT, Unix, and VMS.

You set a number of operating parameters when you launch VORTEXaccelerator and fine-tune them using information from VORTEXmonitor to achieve the best performance for your system. When you first start using VORTEXaccelerator you can begin with the operating parameter recommendations offered in this chapter. However, you ***must*** monitor the results and adjust the settings to achieve the full benefit of using VORTEXaccelerator.

## Starting vtxmux

If you have set up vtxmux as an NT service, it should already be running. Otherwise, you must start the program.

To start the VORTEXaccelerator, type:

**vtxmux *name* nc bs he hs ns cu [ns cu …] [log] [snap] [ess]**

Verify installation by starting the VORTEXmonitor. Type

**vtxmon MUX**

At the ==> prompt, type the command `DT`.

You should see a display of the main VORTEXmonitor control block.

The tuning process involves monitoring activity between the client application and the database, and adjusting the operating parameters accordingly. You can change some settings during a session, but most adjustments require that you restart the vtxmux process.

# Stopping vtxmux

Use `vtxmon` to bring down the VORTEXaccelerator in an orderly fashion at the end of a period, when you want to change parameters, or in any non-emergency situation.

1. Bring up the monitor by typing

   **`vtxmon` *`name`***

   where *name* is the symbolic name (`VORTEX_MUX_NAME`) of the VORTEXaccelerator group.

2. Release and lock all slaves by typing

   `SR*`

3. Shut down the group by typing

   `XK`

# Emergency Stop

Even if some applications are running, you can use VORTEXmonitor to bring down the `vtxmux` group in the unlikely event that VORTEXaccelerator has detected memory corruption or another emergency situation occurs.

1. Bring up the monitor with the option "K" to kill by typing

   **`vtxmon` *`name`* `k`**

   where *name* is the symbolic name (`VORTEX_MUX_NAME`) of the VORTEXaccelerator group.

# Running Multiple Instances

You can run several VORTEXaccelerator instances on the same system. Simply specify a different *name* for each `VORTEX_MUX_NAME`. If you are using `VORTEX_SHM_ADDR`, you must put in an entry for each instance.

## Multiple Slave Groups

VORTEXaccelerator allows you to specify up to four database slave groups at startup. These groups are defined as pairs of `ns` and `nc` values.

Each of these pairs has a corresponding set of entries in the `mux.ini` file: `connectn`, `hostenvn`, `slaven`, and `sqlnstmtn`.

By using the different groups, applications can access different databases, or the same one, but with a different connection id.

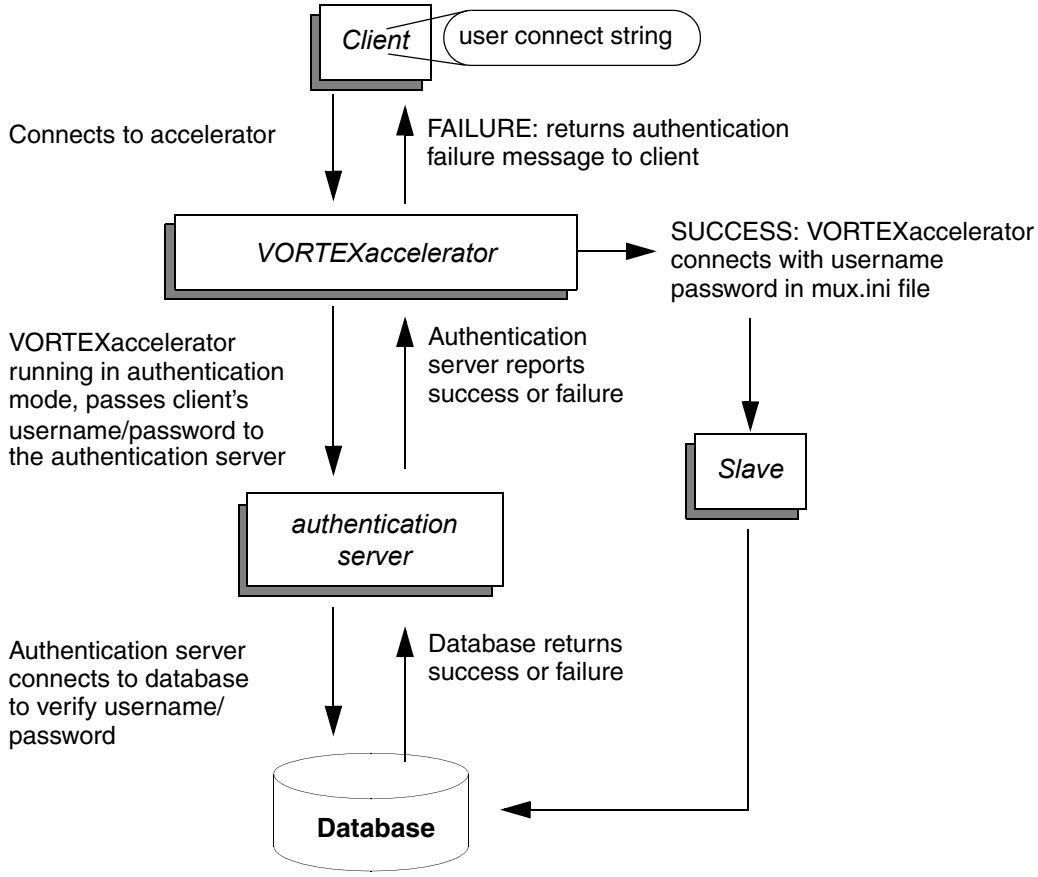
You can specify slave groups in two ways:

- TRIMpl applications use the `connect()` call where the `id` identifies the slave group.
- VORTEXcli applications specify the slave group by passing a string with the group id (0-3) as the `conn` parameter to `VTXCONN()`.

# Using Authentication

To run in authentication mode, you must start the authentication server before the `vtxmux` process. `vtxmux` detects its presence and sends the client application's connect string (username and password) to the authenticator for validation.

Once the username and password have been authenticated, however, `vtxmux` uses the connect string specified in `mux.ini` to connect to the server.




*VORTEXaccelerator running in authentication mode passes a client's username/password to the authentication server. Once the username/password have been authenticated, the VORTEXaccelerator reconnects using the information in* `mux.ini`.

## Operating Settings

### *Instance — name*

When you start VORTEXaccelerator you must give the session an "instance name" by which it is identified to other processes. This name can be any length and can include any alphanumeric character as well as punctuation.

### *Client-slave settings — nc and ns*

The settings for the values of maximum number of clients and maximum number of slaves and the ratio of these two values is probably the single greatest determinator of the performance enhancement you achieve with VORTEXaccelerator.

If the ratio is too high, client applications may experience long delays in getting serviced. If it is too low, improvement in performance or memory usage is reduced.

The max client (`nc`) value determines the greatest number of client applications that can register. If the value is set to 10 then the 11th client application that attempts to register receives an error message saying that too many clients are connected.

The max slave (`ns`) value determines the greatest number of slave processes for a given database connection of a given instance of VORTEXaccelerator.

When adjusting these values, consider the amount of update versus read activity. As soon as a client application begins a transaction, that client is ***locked*** to the slave. To avoid the possibility of an application deadlocking, the lock remains until the transaction commits, aborts, or (during a read) EOS is reached.

The shorter the transaction, the smaller the amount of time the slave is locked to the client. Read activity can also lock a slave to a client. If the select cursor returns more records than can fit in the data buffer, `bs`, then the slave cannot release from the client because the client application may come back to fetch more data. To avoid this situation, use a restrictive `WHERE` clause in your `SELECT` statement.

A good starting point for setting the `nc/ns` relationship is 5:1. You can use VORTEXmonitor's `DL` (see page 23) to monitor VORTEXaccelerator performance and make adjustments as necessary.

If VORTEXaccelerator is configured for more than one database you must specify an `ns` (and a `cu`) value for each database. (The connection configurations are defined in `mux.ini`.)

### *Max database cursors — cu*

The number of actual database cursors that are allocated per slave process should be high enough to service all the clients. If this value is too low, client applications experience a performance degradation as cursors are swapped out.

However, a value too close to one cursor for each application reduces any performance benefits of cursor reuse.

Begin by setting this value to one-third of the total number of distinct logical cursors in your applications and make adjustments as needed. Check the value with VORTEXmonitor's `DL` command (detailed on page 23).

### *Data buffer — bs*

This value, in KB, represents the size of the area used to exchange data between the client application and the database slave process. Set it to be large enough to contain the SQL statement, any bind variables, and the returned data, but no so large that the slave processes return more data records than typically required.

### *Hash table — he and hs*

The settings for maximum number of SQL statements that can be kept in the VORTEXaccelerator hash table and the size of the hash table itself influence performance.

If the hash table fills up because either too many entries are allowed or not enough space is allocated, VORTEXaccelerator returns an error.

You can monitor these values using VORTEXmonitor's `DH` command, as described on page 22.

### *Activity log*

Specify the log file directory in `mux.ini` with the log directory

parameter and use the keyword `log` at process start time to set logging on. If you don't want activity logging, simply omit the keyword at start time. The log file is called `ttc_pid.log`.

### *Memory snapshot — snap*

Specify the directory for snapshot files in `mux.ini` using the log directory parameter. Specify the keyword `snap` at process start time. If a data corruption is detected, vtxmux writes a shared memory snapshot to the filename `ttc_name.ss` in the directory you specified.

### *Existing shared segment — ess*

Specify that the VORTEXaccelerator instance should use an existing shared segment if it can fit. The default is for each instance to be in its own shared segment.

## Command Summary

Most of the configuration settings previously described are set at runtime when you, or another program, start a process. Here are the process commands with their parameters.

```
vtxmux name nc bs he hs ns cu [ns cu …] [log] [snap] [ess]
```

**name** Identifies a given instance of VORTEXaccelerator running on the system. This value is a string of any length of alphanumeric characters and can include punctuation.

The name is stored in the `VORTEX_SHM_FILE` file so that the other VORTEXaccelerator programs, as well as client applications, can find the correct shared memory area.

**nc** The maximum number of client applications that can register with VORTEXaccelerator for service.

**bs** The buffer size, in KB, used to exchange data between the client application and the database slave process.

**he** The maximum number of SQL statements that can be kept in the VORTEXaccelerator hash table.

**hs** The hash table size, in KB, where the SQL statements are stored.

**ns** The maximum number of slave processes to use for a given database connection of a given instance of VORTEXaccelerator.

**cu** The number of actual database cursors allocated per slave process.

**log** Directs VORTEXaccelerator to maintain a log file of activity. The log file location is defined in `mux.ini` and is called `ttc_<name>.log`.

**snap** Directs VORTEXaccelerator to write a shared memory snapshot file if any data corruption is detected. The snapshot file location is defined in `mux.ini` and is called `ttc_<name>.ss`.

**ess** Directs VORTEXaccelerator to use an existing shared memory segment if it can fit. The default is to put each VORTEXaccelerator instance in its own shared memory segment.

# Running VORTEXdaemon

Typically VORTEXdaemon, an overseer process that validates data structures, only runs in a production system. According to the schedule you set, the process “wakes up,” checks that all the VORTEXaccelerator processes are running correctly and reports to vtxmux.

If all the structures are correct, it returns to “sleep.” If any structures are corrupt, vtxmuxd shuts down the vtxmux process. It does not send any alarms to other processes.

The daemon can also be configured to “abdicate” slaves that have not been used for a certain amount of time. Abdicated slaves are released. Slaves can also be “freed,” a state in which the connection is broken but the slave persists. This feature is particularly useful for badly written applications that keep a client-slave connection open during user input.

You need to set several parameters for VORTEXdaemon to access the correct shared memory object and perform its "overseer" duties. Again,you set the values for these parameters on the command line when you start the process for NT, Unix, and VMS.

## Operating Settings

### *Instance — name*

You need to give vtxmuxd the name of the vtxmux instance you want it to watch. If you are running multiple vtxmux processes, you must launch a daemon for each instance. (See "***Instance — name***" on page 15 for more information.) none?

### *Sleep — sleep*

vtxmuxd is generally set to "wake up" every so many seconds to check the data structures, clean up memory, report to `vtxmux`, and go back to "sleep." "Sleep," measured in seconds, refers to the interval between checks.

60 seconds is a safe sleep setting.

### *Controlling slave activity*

The rest of the vtxmuxd parameters control the actions that the daemon takes when it wakes up: terminating and releasing slave processes.

Example

```
vtxmuxd hunter0 60 -a2  -f90  -m120
```

The VORTEXdaemon here is overseeing an instance of vtxmux called ***hunter0*** that:

- Sleeps for 60 seconds between checking the process.
- Terminates slaves that have not been used in the past 2 hours.
- Breaks client-slave connections that have been inactive for 90 seconds.
- Terminates unresponsive slaves after 120 seconds.

### *Slave termination (hour) — [-an]*

Terminating (also called abdicating) unused processes is always a good idea. You can terminate slaves who haven't been used to access to a database after a certain number of hours. For example, `-a2`, means that any slave that has not been used for two hours is terminated.

### *Slave not active — [-fn]*

Clients should not be allowed to remain attached to slaves without activity. If you are unable to redesign applications that allow long connections (which lock up the slave and make it unavailable for other clients' use), you can set a time (in seconds) after which the client-slave connection is broken and the slave is freed to perform other work. Once the connection is broken, the client must establish a new connection with a new slave. If it attempts to fetch more data through the broken connection, the client application receives an error message.

### *Slave wait — [-mn]*

You also control how long the VORTEXaccelerator waits for slaves to respond to a control function. Set time to wait (in seconds) that VORTEXaccelerator waits before terminating the slave.

### *Command summary*

Most of the configuration settings previously described are set at runtime when you, or another program, starts a process. Here are the process commands with their parameters.

```
vtxmuxd name sleep [-an] [-fn] [-mn]
```

**name** The name used to find the correct instance of VORTEXaccelerator.

**sleep** The sleep period, in seconds, for VORTEXdaemon.

**-a*n*** The number of hours, *n*, a slave can be idle before being terminated. For example, `-a2`, means that any slave that has not been used for two hours is terminated.

**-f*n*** The number of seconds, *n*, a slave can be connected to a client with no activity. If this time is exceeded, then the client-slave connection is broken. If the client application attempts to fetch more data, it receives an error message.

**-m*n*** The number of seconds, *n*, the VORTEXaccelerator waits for a slave to respond to a control function before terminating the slave.

# Chapter 4
# Monitor & Tune Activity

VORTEXmonitor allows you to monitor VORTEXaccelerator and make some changes to VORTEXaccelerator's operation while it is running, as well as gather statistics to help you make better choices when starting VORTEXaccelerator. You also kill the vtxmux process from the monitor.

The vtxmon process' commands are divided into three categories:

1. ***Display*** — prefixed with a "D," these commands show you values.
2. ***Slave*** — prefixed with an "S," these commands act on slaves.
3. ***Execute*** — prefixed with an "X," these commands perform various actions.

## Running VORTEXmonitor

To start the monitor vtxmon, you only need the vtxmux instance name:

```
vtxmon hunter0
```

starts a VORTEXmonitor application for the instanced identified as ***hunter0***.

To end vtxmon simply type `R` on the command line where you are instructed to "`Enter option`."

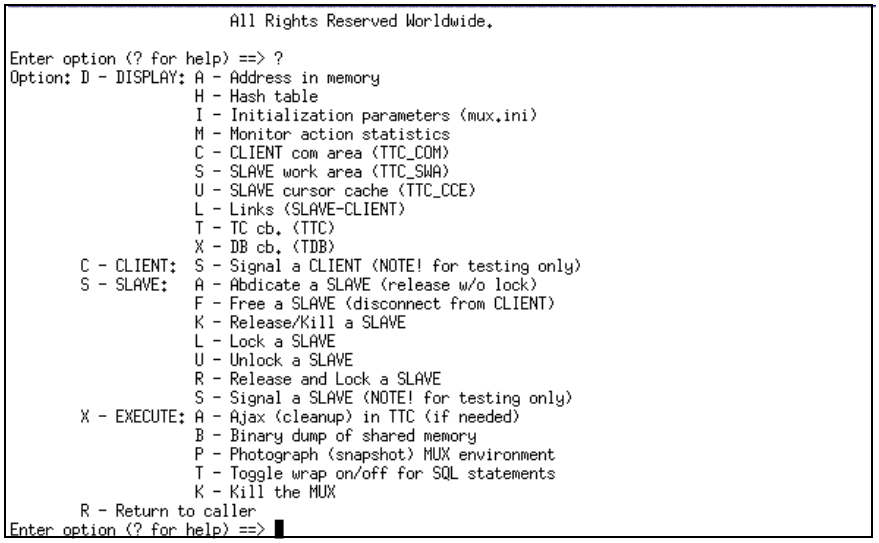
You also use vtxmon to stop the VORTEXaccelerator application:

```
vtxmon hunter0 k
```

kills the ***hunter0*** process.

## Getting Help

To see a list of commands, type?:




## Tuning Activity

The purpose of monitoring the application database connections is to provide information about where to focus your performance tuning efforts.

VORTEXmonitor shows status information about all important areas of consideration.

## Hash Table

By looking through the hash table, you may find SQL statements that can be adjusted to improve performance. Some changes are straightforward: perhaps several SQL statements repeat the same request but use different case or spacing. These can be easily consolidated.

Other changes may require more editing. For example, several SELECT lists can contain the same items in a different order. You can reorder the lists and modify applications so they all use the same list.

You can control whether the SQL statement text wraps or not with the `XT`, which toggles wrapping on and off, for easier viewing.

In addition, you can spot poorly written SQL, such as unbounded `WHERE` clauses, and clean them up.

To see the hash table, type DH:

```
HID  Len Statement
--- ---- ----------------------------------------------------------------
 62   57 select * from GENESIS_VIEWS order by V_OWNER,V_NAME,V_SEQ
276   85 select * from GENESIS_COLUMNS where C_OWNER = :1 and C_TABLE = :2 orde
641   81 select * from GENESIS_INDEXES where I_OWNER = :1 and I_TABLE = :2 orde
666   63 select * from GENESIS_TABLES order by T_DATABASE,T_OWNER,T_NAME
679   27 select * from GENESIS_USERS
767  103 select * from GENESIS_XCOLUMNS where X_OWNER = :1 and X_TABLE = :2 and

6 entries,  6 buckets used (997 available),  Max chain is 1
Enter option (? for help) ==>
```

## Slave-Client Links

An important component of performance is the slave-client link activity. Note that the left side shows slaves processes and the right side shows the client currently connected to that slave.

To see the slave-client link information, type DL.

```
Enter option (? for help) ==> DL
------------------------------------------------------------------------ Links
<---------------- SLAVE -----------------> <---------- CLIENT --------->
Id   Pid #Used  %  #DBs #Curs  % #WrtX WFM  Id   Pid Cur Re Status
-- ----- ----- -- ----- ----- -- ----- --- --- ----- --- -- ------
 0    -1    12  0    28    12  0     0 000
 1    -1  5639 99  8530  5639 49     0 010
 2    -1  5462 99  8311  5462 48     0 010
 3    -1  5432 99  8276  5432 47     0 010
Enter option (? for help) ==>
```

In this illustration no client is active. The slave is identified by its **Id** and **Pid**, which is -1 if the slave is not being used. If the values for #Used,%, #DBs, #Curs, and so on, are 0 (zero), then the slave has never been used. If those other values are different numbers, then the slave ceased to exist.

**#Used**, which shows the number of times the slave has been used, is the prime indicator of effective slave reuse. If the #Used count drops dramatically as the slave Id increases, consider allocating fewer slaves the next time you start vtxmux.

Two other important columns to examine are the %s. The first represents the percentage of calls where the requested cursor was already cached and second shows the percentage of calls where the EXEC or OPEN was already cached. An effective system shows above 95% for these two values.

The other values are:

- #DBs — The number of actual database calls.
- #Curs —The number of actual EXECs and OPENs performed.
- #WrtX — The number of update transactions.
- W — Displays a 1 if a write is in progress. Zero (0) indicates no write in progress.
- F — Shows the cursor cache status. If it is full (1), you may want to allocate more cursors. Zero (0) indicates cursors. (See "***Max database cursors — cu***" on page 16.)
- M — Displays any nonzero number if a monitor function is in progress. 1 in this column indicates a locked slave.

On the client side of the screen, the most important value to look at is ***status***, the current status of the link, which is either

- `Client has control` — indicating that the client application is running while holding a slave connection. The application's hold times should be so short as to make it difficult to see this status.
- `DB has control` — indicating that the database has control, which is more likely.

Other values are:

- `Cur` — The cursor currently in use
- `Re` — The reference count for the link. If non-zero, the slave is locked to the client.

## Reviewing Action Summary

Typing `DM` shows a summary of the VORTEXaccelerator's actions (also called statistics). The last action, `SLAVE unlink of CLIENT`, is the number of times a slave serviced a client request and was released. In a well-written OLTP application, this number should increase rapidly.

```
Enter option (? for help) ==> DL
------------------------------------------------------------------------ Links
<---------------- SLAVE -----------------> <---------- CLIENT --------->
Id   Pid #Used  %  #DBs #Curs  % #WrtX WFM  Id   Pid Cur Re Status
-- ----- ----- -- ----- ----- -- ----- --- --- ----- --- -- ------
 0    -1    12  0    28    12  0     0 000
 1    -1  5639 99  8530  5639 49     0 010
 2    -1  5462 99  8311  5462 48     0 010
 3    -1  5432 99  8276  5432 47     0 010
Enter option (? for help) ==> DM
------------------------------------------------- Monitor stats
Monitor Action               Requested Completed
---------------------------- --------- ---------
CLIENT allocate ..............       753       753
Exiting CLIENT ...............       655       655
Dead CLIENT found ............        98        98
CLIENT canceled (ctrl-C) .....         0         0
Kill a SLAVE .................         0         0
Release a SLAVE ..............         0         0
Abdicate a SLAVE .............         5         5
Dead SLAVE found .............         0         0
SLAVE failed (died) ..........         0         0
SLAVE exec failed ............         0         0
Forced SLAVE/CLIENT unlink ...         0         0
Lock a SLAVE .................         0         0
Unlock a SLAVE ...............         0         0
SLAVE confirm MF .............        86        86
SLAVE unlink of CLIENT .......     22258     22258
Enter option (? for help) ==>
```

- `CLIENT allocate` — Number of client connections.
- `Exiting CLIENT` — Number of client releases.
- `Dead CLIENT found` — vtxmuxd or `XA` found a dead client.
- `CLIENT canceled` — Client sent interrupt signal.
- `Kill a SLAVE` — vtxmon issued an SK command.
- `Release a SLAVE` — vtxmon issued an SR command.
- `Abdicate a SLAVE` — Client issued a vtxmuxd -a*n* or vtxmon `SA` command.
- `Dead SLAVE found` — vtxmuxd or vtxmon's `XA` (Ajax) command found a
- `SLAVE failed (died)` — Slave process died unexpectedly.
- `SLAVE exec failed` — Slave process could not be started, check log file.

- `Forced SLAVE/CLIENT unlink` — vtxmuxd -f*n* or vtxmon `SF` command issued.
- `Lock a SLAVE` — vtxmon `SL` command issued.
- `Unlock a SLAVE` — vtxmon `SU` command issued.

Typically the values in the **Requested** column match those in the **Completed** column. In a heavily loaded system, you may see a brief instant where the numbers do not match, but every request should be completed unless you use the "wildcard," or "all" (*) argument to kill, lock, or unlock a slave. When you send a command to "all," processes to which it doesn't apply simply ignore the command.

## Viewing Control Area

Type `DT` to see the VORTEXaccelerator control area, which gives you an overall view of the VORTEXaccelerator configuration.

```
Enter option (? for help) ==> DT
------------------------------ TTC(3,1,6,4), Name: 'MUX', Owner: 'pade'
total shared memory size ..... 1111792
user ID ...................... 501
process ID ................... 13918, message queue ID: 2001
authorization process ID ..... -1, message queue ID: 0
deamon process ID ............ 14112, Free: 0, MF: 0, Age: 3600
# of DBs in use .............. 1
max # of allowed clients ..... 200
max # of concurrent clients ... 200
current # of clients ......... 0
client work-area length ...... 5300
max # of slaves .............      4 Total: 4
current # of slaves ..........      0 Total: 0
busy # of slaves .............      0 Total: 0
# of locked slaves ............      0 Total: 0
slave work-area length ........    524 Total: 524
max # of DB cursors ..........     16 Total: 16
internal request queue count .. 120694
max number of hash entries .... 64
total hash buffer length ..... 32768
number of hash entries ........ 34
number of free bytes (hash) ... 30308
Enter option (? for help) ==>
```

The two most important values displayed in the control area are:

- The internal request queue count.
- The SQL statement hash statistics at the bottom.

The `internal request queue count` is the number of times a client requested service and no slave was available. As long as application response times are acceptable, this count is not a problem. If the count rises rapidly then you may want to allocate a few more slaves.

The SQL statement hash statistics are located at the bottom of the display. If the `number of hash entries` equals the `max number of hash entries` or the `number of free bytes (hash)` is close to 0, then the SQL statement cache is full and new SQL statements are not cached for reuse. The only way to remedy this is to stop the VORTEXaccelerator and restart it with larger `he` or `hs` values.

## Other Display Commands

Other commands, although less frequently used, are available:

`DA` Address. Displays the contents of memory.

`DC` Client. Displays the client's communication area.

`DS` Slave Communication Area. Displays the slave's communication area.

`DU` Slave Cursor Cache. Displays the slave's cursor cache. You can look at all the cached cursors or just the open ones. If a client-slave link is staying open for a long time, you can find the statement(s) by looking at the open cursors.

Using the `Cur` value on the client side of the `DL` command and the slave's `ID`, typing `DU<id> <cur>` shows you the offending SQL statement.

`DX` Client Database Communication Area. Displays the client's database communication area. This command can be useful for checking a client process' last action.

The `DA`, `DC`, and `DS` commands are typically only used by Trifox support personnel.

# Controlling Slaves

The slave commands, prefixed with `S`, are useful for aggressive tuning and to explore behaviors in extreme conditions.

## Specifying a Slave

You can issue slave commands to specific slaves, a list of slaves that meet certain criteria, or all slaves.

- *n* — Sends a command to an actual slave id numbers.
- > *n* — Sends a command to all slaves with an ids greater than *n*.
- < *n* — Sends a command to all slaves with an ids less than *n*.
- * — All slaves.

## Slave Activity

The commands you can give a slave fall into three groups:

- Release
- Free
- Lock/Unlock

### *Release*

You can release a slave without or without locking it, or you can kill it.

| | |
|---|---|
| **Release no lock** | SA sends a message to the slave to exit as soon as it is not being used. The slave process can be restarted. This action is similar to the vtxmuxd's -a*n* parameter. |
| **Release and lock** | SR releases and locks a slave. |
| **Release and kill** | SK releases a slave and kills it. Use this command only if the slave process is corrupted and is not responding SA or SR. |

### *Free*

SF frees a slave from a client connection. This command is similar to vtxmuxd's -f*n* action.

### *Lock*

SL locks a slave, which prevents it from being linked to a client. While you are experimenting with the client-slave ratio, you might lock a number of slaves to see if the ratio could be set higher.

### *Unlock*

SU unlocks a previously locked slave.

## Housekeeping Activities

### *Cleanup processes*

You can "Ajax" (clean up) the shared memory area by typing XA.

VORTEXdaemon performs this task every time it wakes up, checking the integrity of the data structures to verify that no corruption has occurred

### *Killing the process*

You kill vtxmux with a simple XK.

### *Snapshots*

You can write a snapshot of shared memory as either a binary or ascii dump by issuing the command XB or XP. You must specify a file name in either case. This command is typically only used for troubleshooting by Trifox personnel.

### *Display control*

Use XT to control the displays of DH and SU by turning the text wrap on or off for SQL statements.

# Chapter 5
# Environment Variables

Setting environment variables is an operating system-specific task. If you are responsible for setting up your environment and installing Trifox products, but are not familiar with the procedures for your operating system, consult the operating system manuals.

Note that several of the environment variables point to the same objects. For example, both `TRIM_MUX_NAME` and `VORTEX_MUX_NAME` contain the VORTEXaccelerator shared memory identifier. In all such cases, the DesignVision environment variable is checked last. In the previous example, `VORTEX_MUX_NAME` is searched for first and if it is not found, then `TRIM_MUX_NAME` is checked.

| Name | Products | Description |
|---|---|---|
| `DV_CONFIRM_FILE_ERROR` | DV | Display an error message when a file error occurs. The default is to not display an error. Set to any value. |
| `DV_PREFIX` | DV | Replaces the default "**dv**" prefix with the value of DV_PREFIX. This affects the .kma, .img, .ini, .xaml files. If DV_PREFIX<.filename> is not found, then DV uses the default "**dv**" prefix. |
| `DV_TRANSPARENT_COLOR` | DV | RGB color used to make button icons transparent. For example, DV_TRANSPARENT_COLOR=255,0,255 uses magenta as the background to make buttons appear transparent. |
| `EDITOR` | DV/ TRIM | Name of editor program to use in DVapp and DVreport designers, as well as the TRIM equivalents. |
| `GENESIS_HOME` | GENESIS | Locator for the GENESIS DS description files. |
| `TRIM_HOME` | DV/ TRIM | Locator for lib, term, and qmr directories. |
| `TRIM_MUX_NAME` | DV/ TRIM | VORTEXaccelerator shared memory identifier. |

| Name | Products | Description |
|---|---|---|
| TRIM_SHM_ADDR | DV/ TRIM | Name of the shared memory preferred address file. This variable only applies to operating systems that allow a shared memory segment to be mapped to different addresses. Specifying this variable ensures (for those operating systems) that vtxmux clients can locate the correct address. |
| TRIM_SHM_BASE | DV/ TRIM | Defines a base address for shared memory loading. TRIM_SHM_ADDR takes precedence over TRIM_SHM_BASE if the object name is in TRIM_SHM_ADDR. Value is an address, either 8 or 16 digits and in the correct format for the underlying architecture. For example, on x86 systems, 00000008 means address 80000000. |
| TRIM_SHM_FILE | DV/ TRIM | Name of the shared memory description file. |
| VORTEX_API_LOGFILE | VORTEX | Name of the file to use for VORTEX logging. |
| VORTEX_API_LOGOPTS | VORTEX | Keyword that specifies logging options for VORTEX:<br>• FLIP specifies that PLAY must byte flip integers in the playback file.<br>• FULL specifies detailed VORTEX logging.<br>• MULTI specifies unique log filenames.<br>• PLAY specifies data playback.<br>• RECORD specifies data recording.<br>• SQL specifies SQL file creation.<br>• TIME specifies time to complete each call. |
| VORTEX_CCMAP_FILE | VORTEX | Name of the file to use to translate ASCII to EBCDIC and back again. This is set on the server system. The file contains 512 blank separated hexadecimal bytes, e.g. 0x7B 0x41, that define the ASCII to EBCDIC mappings. |
| VORTEX_DDT_MASK | DV/ TRIM/ VORTEX | The date/time format to use for formatting datetime data into strings and for interpretting strings for conversion into datetime data. The default is “DD-MON-RR”. |
| VORTEX_HOME | VORTEX | Locator for client lib directory. |
| VORTEX_HOST_HIDEGPF | VORTEX | If false, then SEM_FAILCRITICALERRORS and SEM_NOOPENFILEERRORBOX are set. If true, then the above plus SEM_NOGPFAULTERRORBOX are set. See the VORTEX_HOST_NOSEM env variable. |

| Name | Products | Description |
|---|---|---|
| VORTEX_HOST_LOGFILE | VORTEX | Name of the file to use for VORTEX logging on the server. |
| VORTEX_HOST_LOGOPTS | VORTEX | Keyword that specifies logging options for VORTEX on the server:<br>• FULL specifies details VORTEX logging.<br>• MULTI speifies unique log filenames.<br>• PLAY specifies data playback.<br>• RECORD specifies data recording.<br>• SQL specifies SQL file creation.<br>• TIME specifies time to complete each call. |
| VORTEX_HOST_NOSEM | VORTEX | If set to true, then no special error handling is done in the VORTEX drivers. If false, then see the VORTEX_HOST_HIDEGPF env variable. |
| VORTEX_HOST_SYSLOG | VORTEX | If set to true, then VORTEX host error messages are sent to the system log. |
| VORTEX_MUX_NAME | VORTEX (Mux) | VORTEXaccelerator shared memory identifier. |
| VORTEX_ODBC_CHAR | VORTEX ODBC | Set the ODBC datatype (integer) to be returned for a described char column. The default is SQL_CHAR. (1) |
| VORTEX_ODBC_DATETIME | VORTEX ODBC | Set the ODBC datatype (integer) to be returned for a described timestamp column. The default is SQL_TIMESTAMP (11). |
| VORTEX_ODBC_NUMBER | VORTEX ODBC | Set the ODBC datatype (integer) to be returned for a described numeric column. If datatype < 0, then VORTEXodbc maps the ODBC datatype based on precision and scale. |
| VORTEX_ODBC_TIME | VORTEX ODBC | Set the ODBC datatype (integer) to be returned for a described time column. The default is SQL_TIME (10). |
| VORTEX_ORACLE_FOOLISH | VORTEX | Allows NULL values to be bound for SELECT statement parameters |
| VORTEX_ORACLE_TS_CLEN | VORTEX | Sets the default TIMESTAMP character string length |
| VORTEX_SERVICE_FILE | VORTEX | Specifies a file that lists NT services to start as well as environment variables for that operating system. |

| Name | Products | Description |
|---|---|---|
| VORTEX_SHM_ADDR | VORTEX (Mux) | Name of the shared memory preferred address file. This variable only applies to operating systems that allow a shared memory segment to be mapped to different addresses. Specifying this variable ensures (for those operating systems) that vtxmux clients can locate the correct address. |
| VORTEX_SHM_BASE | VORTEX /DV | Defines a base address for shared memory loading. VORTEX_SHM_ADDR takes precedence over VORTEX_SHM_BASE if the object name is in VORTEX_SHM_ADDR. Value is an address, either 8 or 16 digits and in the correct format for the underlying architecture. For example, on x86 systems, 00000008 means address 80000000. |
| VORTEX_SHM_FILE | VORTEX (Mux)/ DV | Name of the shared memory description file. |

# Appendix A
# Initialization File

Most of the Trifox tools and sub-systems read configuration and initialization data from special `.ini` files. These files typically have the same format:

```
option                value
```

The *option* is the name of the initialization option, setting name, or parameter. Lines with un-recognized options are ignored.

*Value* is the value of the option. Depending on *option* the *value* can be a number, a yes/no, or a text string. The value can also represent one or more environment variables expressed as:

```
$(name)
```

The environment variable(s) are expanded before the value is evaluated.

The files support text strings as values, but they must be enclosed in double quotes ("), SQL-style, if blanks or quotes are part of the string. If no ending quote mark is provided, the string is terminated with a `\n`.

If an option is not found in the file, then the default value is used.

The various relevant `.ini` files are described in detail in the following section(s).

Edit them using any ascii-based text editor. If you are reinstalling a product, we recommend you edit a "clean" copy of each `.ini` file, rather than modifying an existing one from your environment.

## mux.ini

`mux.ini` is used by VORTEXaccelerator.

### connect*n*

**Type** text

**Default** none

**Description** This required parameter defines the connect string to use for a connection. You can specify up to 4 connections (values 0 to 3, inclusive). Each of these connections may have associated values for **hostenv**, **slave**, and up to 4 **SQL statements**. The login must also be in the authorization table or dictionary for the database. It does not require special privileges.

When connecting to a database through VORTEXaccelerator, this parameter overrides the connect string coded in the application. It has no affect on applications not connecting through VORTEXaccelerator.

**Example** The following specifies that `connect0` is connected with the user name `SYSUSER` and password `SECRET`:

```
connect0      SYSUSER/SECRET
```

### cmd_char

**Type** number

**Default** 1

**Description** This optional parameter is Oracle-specific and depends on the value established by the particular version of the Oracle database. It represents the value to send as Oracle's TDB_CMD_CHAR parameter when a slave is started.

Oracle 6 and 7 have a variety of character data types from which you can choose, depending on the behavior (trimmed blanks, stored blanks, and so on) you want. For detailed information about behaviors and corresponding values, refer to the Oracle OCI documentation. If no value is specified, VORTEXaccelerator uses 1.

**Example** The following specifies that Oracle's data type represented by 96 should be used for all text data. The value of 96 depends on the version of Oracle and its operating parameters, as determined by Oracle:

```
cmd_char      96
```

### cmd_language

**Type** number

**Default** 1

**Description** This optional parameter is Oracle-specific and depends on the value established by the particular version of the Oracle database. It represents the value to send as Oracle's TDB_CMD_LANGUAGE parameter when a slave is started. For detailed information about behaviors and responding values, refer to Oracle documentation on the oparse command. If no value is specified, VORTEXaccelerator uses 0.

**Example** The following specifies that 1 is sent as TBD_CMD_LANGUAGE when a slave is started:

```
cmd_language      1
```

### dflt_db_id

**Type** number

**Default** 0

**Description** This optional parameter specifies the initial database ID setting. Since connecting to the VORTEXaccelerator does not actually create a database connection, the database ID value in VORTEX is not set until a SQL action occurs. Applications that require this information must set the value to operate correctly.

**Example** The following specifies that at connect time the database ID should be set to 2, which is identified as Sybase in Trifox's `vortex.h` header file. For a complete list of the numbers corresponding to databases, refer to your `vortex.h` file:

```
dflt_db_id      2
```

### display_banner

**Type** yes/no

**Default** yes

**Description** This optional parameter specifies product information banner display. If unspecified, the banner appears.

**Example** The following specifies that the banner screen should not appear when VORTEXaccelerator is started:

```
display_banner      no
```

### dll*n*

**Type** text

**Default** none

**Description** This parameter specifies the dynamic link library (DLL) or share library file to use for the connection.

**Example** The following specifies that `vtxslav.exe` should use an Oracle database on NT for connection3:

```
dll3            vtx0
```

### errfile

**Type** text

**Default** none

**Description** This optional parameter is useful for debugging applications. You can create a file that contains database return values that should close the connection. The file is checked and if the driver receives one of these values, it closes the connection.

**Example** The following Unix example specifies that the connection should be closed if the driver receives any of the errors listed in the file `errfile` located in the directory `/usr2/vortex`:

```
errfile      /usr2/vortexXD/errfile
```

### hostenv*n*

**Type** text

**Default** none

**Description** This optional parameter specifies the environment variable(s) to send to the host for connection*n*.

**Example** The following specifies that `ORACLE_HOME` and `ORACLE_SID` should be sent to the host (presumably an Oracle database on Unix) for connection0:

```
hostenv0  ORACLE_HOME=/usr4/oracle,ORACLE_SID=A
```

### log_directory

**Type** text

**Default** None

**Description** This optional parameter specifies where to place log files. It must contain the trailing directory separator. The filename is `ttc_`*`MUX`*`.log` where *MUX* is the VORTEXaccelerator instance name.

**Example** The following specifies that the log should be written to a directory called /tmp/ on a Unix machine:

```
log_directory     /tmp/
```

### slave*n*

**Type** text

**Default** None

**Description** This *required* parameter specifies the program to execute for starting a slave on the VORTEXaccelerator machine for connection*n*. If the value is vtxslav.exe (the only option possible for NT), then you must specify a **dll** value.

**Example** The following specifies that `vtxslav.net`, the VORTEXaccelerator slave for the network version of VORTEX on Unix, starts slave processes for connection0:

```
slave0     vtxslav.net
```

### sql*n*stmt*n*

**Type** text

**Default** None

**Description** This optional parameter specifies SQL statement(s) to execute when a given **slave** is started. You can specify up to 4 statements (from 0 to 3, inclusive) for each connection.

**Example** The following specifies that the SQL statement UPDATE STATUS SET usercount=usercount + 1 is the first command issued when a slave is started on connection0:

```
sql0stmt0 ‘‘UPDATE STATUS SET usercount=usercount+1’’
```

## Sample Unix mux.ini file

```
rem ---------- VORTEXaccelerator specifics
log_directory  /tmp/                        -- where to put optional log files
```

```
errfile         /usr3/rad/tbu/src/errfile
errfile_sleep   120
dflt_db_id      4                           -- default database ID
#
cmd_char0       96
cmd_language0   1
#connect0        niklas/back/sdms:trifox0
#connect0        niklas/back/vision:vision.ini
connect0        niklas/back
#connect0        niklas/back/list:list.ini
slave0          vtxslav
dll0            VTX0
#
cmd_char1       96
cmd_language1   1
connect1        niklas/back
slave1          vtxslav
dll1            VTX3
```

## Symbols



## A



## B



## C



## D



## E



## F



## G



## H

## U



## V



## W



## X