VORTEXcobol Precompiler Reference Manual

August 12, 2019

Trifox Inc.

www.trifox.com

Trademarks

Copyright

Contents

Preface

The VORTEXcobol Precompiler Reference Manual explains how to use the precompiler so that COBOL programs can access relational databases through VORTEX. This guide does not discuss principles of writing COBOL, COBOL syntax, or Embedded SQL.

Audience

This reference manual is for programmers who use embedded SQL in existing or new COBOL programs, and want to take advantage of the power and flexibility of VORTEX products in accessing relational database systems without programming directly to the VORTEX Client Library Interface (VORTEXcli). While this document does not assume that you are a database expert, understanding the information you are working with is important. It does assume that you have a working knowledge of your operating system and its conventions, including standard commands, and how to operate your compiler.

Organization

This document is a guide for using VORTEXcobol. 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.

Background

Trifox Inc. has been serving the relational database market since 1984 through consulting and the development of software products.

VORTEX

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.

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.

VORTEX client/server enables your SQL applications to access data on different platforms over one or more network configurations via 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.

DesignVision/TRIM

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, Unx, 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 lets you write powerful applications in no time.

GENESIS

GENESISsql is a SQL processor that accesses low-level data sources such as ISAM, SDMS, ADABAS, and dBASE and makes the data accessible to VORTEX-supported clients. GENESIS data sources can be added 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 Unix 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.

Support and Feedback

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 South Winchester Boulevard
  Campbell, CA 95008
  U.S.A.
- 408/796-1590

Revisions

18 January 2000

Updated *"**Building Your Application**"* on page 9 to reflect the new script procedure.

23 September 2005

Added new -copy option.

Chapter 1 Precompilers

The VORTEX COBOL precompiler, VORTEXcobol, lets you easily turn your existing COBOL source code into applications that can not only access relational databases, but also take advantage of VORTEX features.

VORTEXcobol performs a variety of services including:

- Converting SQL statements into VORTEXcli function calls.
- Converting SQL datatypes to COBOL datatypes.
- Managing multiple database connections
- Optimizing database access by using bulk input and output transfers.

How It Works

Precompilers, as you might guess, perform some processing on a file before it is compiled by the program that creates a binary executable from your ascii source code. VORTEXcobol processes embedded SQL commands that enhance COBOL code and allow it to communicate with relational databases.

You continue to write in the language with which you are familiar, and if your application requires no changes, do nothing. If you're working with existing software, you don't have to recode the entire application just to run against a SQL database.

The Process

1. Edit your COBOL source and add EXEC SQL statements as necessary.
2. Make sure your environment is set up according to the instructions in Chapter 2.
3. Run VORTEXcobol.
4. Run the new source through your COBOL compiler, as you normally would, linking with VORTEX libraries, as well as all the components your compiler requires.
5. Use the new executable. If you are working in a multi-tier environment, and don't have database connectivity, make sure that VORTEXserver is set up correctly, and use that along with VORTEXnet to communicate with the database.

Required Items

To use this product, you must have the following software products installed and working on your machine(s).

- Your own COBOL compiler.
- The support files and linking libraries required by your compiler.
- VORTEXcobol, VORTEX libraries.
- VORTEXserver is optional. You can use your own database's connectivity, or if the application executable is on the same machine as the database, simply use VORTEX's database driver for a direct connection.

Structures

VORTEX precompilers use the ANSI standard `SQLCA` structures to return errors and query result information. See the `cob_sqlca.h` file in "***Include Files***" on page 24 .

Chapter 2
Getting Started

Setting Up

This section itemizes the environment variables and initialization parameters that must be set for precompiling and running VORTEXcobol applications. For operating-system specific instructions, consult the README file in the VORTEXcobol package.

First, download the precompiler evaluation or product package from the Trifox FTP site and install the program.

Environment Variables

Environment variables establish pointers to files and directories that are necessary for precompilation and application execution. In addition to environment variables necessary for your communication with the database, you need the following variables for VORTEXcobol operations.

- VORTEX_HOME or TRIM_HOME

  This variable points to the lib directories. The precompiler searches for `VORTEX_HOME` first, only using `TRIM_HOME` if it can't find `VORTEX_HOME`.

- VORTEX_API_LOGFILE

  This ***optional*** variable, along with the following one, help you troubleshoot applications. This variable represents the base VORTEX log file name. The default is `VORTEX_API_LOGFILE.log` for either abridged or FULL logging. RECORD and PLAY options create a file with the same name and a `.vdf` extension.

- VORTEX_API_LOGOPTS

  This ***optional*** variable indicates the logging level(s) you have selected. You can choose and combine from several options. Review the chapter on *Logging* in the ***Trifox Resource Manual.***

Initialization File (vtxcob.ini)

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, edit a "clean" copy of each `.ini` file, rather than modifying an existing one from your environment.

*vtxcob.ini*The following table details the available options. For most installations, the defaults are adequate and you won't need to make modifications.

Option	Type	Default	Description
columns	number	128	Maximum number of columns allowed in query result.
db_cursors	number	32	Number of actual database cursors to allocate.
dflt_rw_transaction	yes/no	no	Default transaction state.
fetch_buffer_size	number	4096	Database fetch buffer size (bytes)
ft_area_a	number	1	Starting column for area a.
ft_area_b	number	5	Starting column for area b.
hard_close_cursors	yes/no	no	Hard close cursors.
logical_cursors	number	128	Number of logical cursors to allocate.
sql_ext	text	.sco	File extension including "."
uppercase_sql	yes/no	no	Uppercase SQL statements.

Example

```
rem ---------- VORTEXcobol generics
columns          512          -- max # of database columns
db_cursors        64          -- max # of DB cursors
dflt_rw_transaction no            -- initial transaction is read write
fetch_buffer_size  4096          -- fetch buffer size (in bytes)
hard_close_cursors  no           -- soft close by default
logical_cursors    512          -- max # of logical cursors
sql_ext          .pco          -- file extension if not specified
ft_area_a         8            -- area A start
ft_area_b         12           -- area B start
```

Error Messages

You must also include the error messages file, `error.msg`, which is used at both precompile time and application execution time. The error messages are detailed in "***Error Messages***" on page 63.

Include File

VORTEXcobol, like your compiler, pulls in any included files at processing time. COBOL source code needs to have the SQLCA file. For a full listing of the file, see "***Include Files***" on page 24.

Verifying the Installation

Once you've set all the environment variables and included all the necessary support files for the precompiler ***and*** your compiler, (see "***Required Items***" on page 5 for a brief list of items) it's time to verify the installation by precompiling, compiling and linking, and then running the sample application in the `cli` subdirectory.

The program, `vtxtest.pco`, uses the `EMPEMPD` table. Verify that you do not have this table in your test database before you try to run the program. Because it creates a table called `EMPD`, running the sample program destroys any data that exists in a table with that name.

See "***Sample Program***" on page 38 for a listing of the program.

Precompile the Embedded SQL Source File

Before you compile the application, you must "precompile" the source file with the VORTEXcobol precompiler. Simply type:

```
vtxcob source[.extension] [output] [options]
```

where

source The name of the source code file that contains the COBOL code and optional embedded SQL statements.

extension If you don't specify an extension, the precompiler uses the `sql_ext` entry in `vtxcob.ini`. If you specify the output file name, you must specify an extension.

output If you don't specify a file name for the output file, the precompiler uses the same name as the input file with the extension `.cob`.

options Processing options:

- **-copy** Automatically pull in all copy libs.
- **-ft** Terminal format.
- **-h *hdr*** Set the name of the generated header file to *hdr.*
- **-i *dir*** Add *dir* to the EXEC SQL INCLUDE search path. This option can be repeated as many times as necessary.

-ii	Ignore indicator byte-order error
-implicit	Do not require DECLARE SECTIONs.
-ma	AcuCOBOL mode.
-ma32	AcuCOBOL mode (32 bit pointers)
-md	OpenVMS COBOL
-mf	Fujitsu COBOL mode.
-mm	MicroFocus COBOL mode.
-net	Generate code for .NET environment
-s	Dump the symbol table.
-sqlca	Include SQLA in WORKING-STORAGE SECTION
-t	Print tracing information to the screen.
-ts	Generate thread safe code.
-wN	Max width N (default: 72, 0 for max)

For example:

```
vtxcob vtxtest.pco vtxtest
```

Runs VORTEXcobol on the test application and names the output file "vtxtest".

Building Your Application

You can use a variety of COBOL compilers. VORTEXcobol includes a script for you to use. For example,

```
cobvtxcob vtxtest
```

NOTE: *When building on AIX, you must use the -brtl linker option if your application will access a local Informix database.*

Run the Test Application

Now you can run the test application.

```
vtxtest db_login
```

Refer to the ***Trifox Resource Manual*** section on connecting your application for the correct `db_login` syntax.

The results of the vtxtest sample program should look like this:

```
                    <<<   VTXTEST  >>>
CONNECTED TO DATABASE AS: NET:scott/tiger
TABLE 'EMPD' HAS BEEN CREATED IN THE DATABASE
```

```
DATA HAS BEEN LOADED INTO TABLE 'EMPD'
OPEN CURSOR AND FETCH DATA FROM TABLE emp 'EMPD'
 ID                NAME      SALARY   DEPT
=========================================
[101]              Smith   25000.00   10
[102]             Robert   25600.00   10
[103]              Henry   23400.00   10
[104]            Johnson   34000.00   20
[105]             George   33000.00   20
[106]               Wang   33000.00   20
[107]              Sandy   30000.00   30
[108]               Mary   32000.00   30
[109]              Linda   30000.00   30
[110]               Alex   30000.00   30
[111]              Allen   29000.00   30
[112]                Bob   38000.00   40
[113]            Michael   32000.00   40
[114]              April   30000.00   50
[115]              Steve   29000.00   50
[116]             Andrew   30000.00   50
[117]                Dan   31000.00   50
[118]            Tolosky   25000.00   50
[119]           Rotomomo   32000.00   50
[120]             Albert   30500.00   50

*** Modify the salary on [120] Albert from $30500 to $35555

OPEN CURSOR AND FETCH DATA FROM TABLE 'EMPD'
 ID                NAME      SALARY   DEPT
=========================================
[120]             Albert   35555.00   50

*** Delete the record of [119] Rotomomo

OPEN CURSOR AND FETCH DATA FROM TABLE 'EMPD'

 ID                NAME      SALARY   DEPT
 =========================================
[101]              Smith   25000.00   10
[102]             Robert   25600.00   10
[103]              Henry   23400.00   10
[104]            Johnson   34000.00   20
[105]             George   33000.00   20
[106]               Wang   33000.00   20
[107]              Sandy   30000.00   30
[108]               Mary   32000.00   30
[109]              Linda   30000.00   30
[110]               Alex   30000.00   30
[111]              Allen   29000.00   30
[112]                Bob   38000.00   40
[113]            Michael   32000.00   40
[114]              April   30000.00   50
[115]              Steve   29000.00   50
[116]             Andrew   30000.00   50
[117]                Dan   31000.00   50
[118]            Tolosky   25000.00   50
[120]             Albert   35555.00   50
```

```
TABLE 'EMPD' HAS BEEN DROPPED FROM THE DATABASE HAVE A GOOD DAY.
```

Chapter 3
Converting Your Files

You may not have to make any changes to your original COBOL source code for it to precompile successfully with VORTEXcobol. However, if you need to make changes for multi-database functionality, or to use SQL features, VORTEX precompilers make it very easy by implementing a rich set of embedded SQL functions.

Using SQL

To issue SQL commands in your COBOL application, you use an industry standard protocol called ***embedded SQL***. You begin each statement with the phrase "`EXEC SQL`" and end it with "`END-EXEC.`" or "`END-EXEC`", depending on whether you want the replacement code to have a terminating ".". Your SQL statements can span multiple lines, but the `END-EXEC` can't be split between lines. The lower case bold text in the following code fragment represents the actual SQL commands.

```
EXEC SQL close c1 END-EXEC.
EXEC SQL declare C0 cursor for select *
                              from staff
                              where id > 50 END-EXEC.
```

Datatype Mappings

The COBOL language does not directly support all database datatypes. VORTEXcobol automatically converts data between the database and COBOL datatypes.

Character Data

Most databases support two types of character data: `CHAR` and `VARCHAR`. `CHAR` data maps directly to the COBOL `PICTURE X` C `char` datatype.

For example, a `CHAR(10)` database column matches a `01 str PICTURE X(10)` COBOL variable. VORTEXcobol null-terminates the variable if there is enough space. Here, if the database column contains "ATENCHAR10," the `str` variable is not null-terminated. If you add an extra char position to the COBOL variable, (for example, `01 str PICTURE X(11)`) VORTEXcobol can null-terminate the string.

On input, the COBOL variable does not have to be null-terminated; however, it does ensure that the application gets correct results, since databases treat trailing blanks differently.

Varchar Data

If you are using VARCHAR data in the form

```
10 MY-VARCHAR
```

```
49 VAR-LEN  PIC S9(4)      COMP-5.
49 VAR-DATA PIC A(20).
```

then the length datatype must be either COMP-4 or COMP-5. Note that the level must be 49 for VORTEXcobol to interpret the variable as a varchar.

Integer Data

VORTEXcobol supports two-, four-, and eight-byte integers where applicable on the hardware.

Fixed and Floating Point Data

Most databases have predefined fixed and floating point datatypes as well as the ability to declare a variety of precision and scale combinations. For example, Oracle has a `NUMBER(p,s)` datatype that can define a numeric column from `NUMBER(1)`, a one-byte integer, to `NUMBER(38,127)`, a very large fixed point number.

You map these values to COBOL's `PICTURE S` datatype. VORTEXcobol converts between the database value and the COBOL variable; however, you lose significant precision if the application does not define a large enough variable.

Date and Time Data

Each database handles date and time information differently.

VORTEXcobol accommodates the variability in date-time with several options. The most simple to implement option has VORTEXcobol using a character array to pass date-time information to and from the database. This method works best with applications that target a single database. The default format for date-time data is DD-MON-YY.

VORTEXcobol can also allow an application to request that date-time information be returned into a fixed point (integer) variable. If the integer is four bytes, then the number of days since January 1, 0000 is returned. If the integer variable is eight bytes, then the number of seconds in the day is returned in the second four-byte integer.

You can change the base date and the default format, if you use a character array, with VORTEXcli's `VTXOPTS()`.

The most flexible and portable technique for using date-time data is the VORTEX internal format. This format lets you take advantage of VORTEX's ability to format date-time values according to the target database. The format allows you to write truly database-independent applications. Applications can use this format directly by declaring a datetime or timestamp variable as follows:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 dt PICTURE X(7).
01 ts PICTURE X(10).
EXEC SQL END DECLARE SECTION END-EXEC.
timestamp ts;
char ts[10];
```

You can also use the `TCVS2D()` and `TCVD2S()` functions to convert date-time information between character strings and the native VORTEX format. Refer to *Conversion Routines* in the ***Trifox Resource Manual*** for details on these functions.

Indicator Variables

Indicator variables simplify NULL value handling by reporting the status of returned data and letting applications know if the database data has been truncated. Indicator variables are short integers and can have the following values:

Value	Description
< 0	Host variable contains a NULL value (input/output).
0	Host variable contains a valid value (output).
> 0	Host variable contains a truncated value (output), the actual database data length is returned in the indicator variable.

To use an indicator variable, simply define it and put it after the host variable in the SQL statement. For example:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 STR PICTURE X(22).
01 IND PICTURE S9(4)COMP.
EXEC SQL END DECLARE SECTION END-EXEC.

EXEC SQL SELECT SPOUSENAME INTO :STR:IND
                FROM STAFF WHERE ID = 42 END-EXEC.
IF IND = -1 THEN DISPLAY "No Spouse"
ELSE IF IND > 0 THEN
   DISPLAY "Spouse name truncated (DBMS length " IND ")".
ELSE DISPLAY str.
…
```

Error Handling

VORTEXcobol uses the ANSI-standard error handling mechanism. It processes error handling commands serially. The statement:

```
EXEC SQL WHENEVER condition action END-EXEC.
```

applies to the entire source file from that point onward. For example, if the application sets

```
EXEC SQL WHENEVER NOT FOUND GOTO no_data END-EXEC.
```

the command applies to every `SELECT` or `FETCH` operation that follows in that file. If the command appears in a function, unless the `no_data` label exists in the other functions in the file, the application is not likely to compile.

In addition, commands are processed serially. Thus,

```
EXEC SQL WHENEVER condition action END-EXEC.
```

statements apply until a following `EXEC` with the same condition replaces it.

Conditions can be:

- NOT FOUND — No records returned from the query.

- SQLERROR — Database error.
- SQLWARNING — Database warning.

Actions available are:

- CONTINUE — Ignore conditions.
- BREAK -- Break out of a loop.
- GOTO *label* — Jump to *label.*

Error messages are returned in two fields:

- `SQLCA.SQLERRM.SQLERRML` contains the length of error message.
- `SQLCA.SQLERRM.SQLERRMC` contains the actual error message.

Some databases return very long error messages that are truncated to fit into the `SQLCA.SQLERRM.SQLERRMC` buffer. If you need to retrieve long messages, use the `VTXGEEM()` function described in the ***VORTEXcli Reference Manual***.

For the greatest portability, though, using the `VTXEMAP()` lets you automatically map different database error codes and messages into a consistent set of error codes and messages.

SQLCA.SQLERRD

VORTEXcobol uses two of the six entries defined for the `SQLCA.SQLERRD` fields in SQLCA. The most useful entry is `SQLCA.SQLERRD[2]`, which returns the number of rows affected by the latest operation. A FETCH into an array of records may return fewer than the maximum number of records. By checking `SQLCA.SQLERRD[2]`, the application knows how many records in the array are valid.

Databases that can return different result sets (like Sybase stored procedures) within the same query use `SQLCA.SQLERRD[0]`.

The possible return codes are:

Value	Description
0	Normal
1	New rows
2	Compute rows

Returning/Receiving Data

VORTEXcobol supports several ways of returning data. You can fetch it into individual variables, or into a single record, or you can fetch into arrays of both host variables and records. Using arrays can improve performance if you are dealing with large amounts of data.

Record Example

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 STAFF-REC.
```

```
   02 ID     PIC S9(9).
   02 NAME   PIC X(11).
   02 DEPT   PIC S9(9).
   02 JOB    PIC X(6).
   02 YEARS  PIC S9(9).
   02 SALARY PIC S9(9).
   02 COMM   PIC S9(9).
01 STAFF-IND PIC S9(4) OCCURS 7 TIMES.
EXEC SQL END DECLARE SECTION END-EXEC.

EXEC SQL SELECT ID,NAME,DEPT,JOB,YEARS,SALARY,COMM
         INTO :STAFF-REC:STAFF-IND
         FROM STAFF WHERE ID = 42 END-EXEC.
```

This code populates the `STAFFREC` record with the correct values, as well as the indicator variables, `STAFFIND`.

Array Example

The following example populates the `STAFF-REC-ARR` array with the correct values, as well as the indicator variables, `STAFF-IND`. Enough indicator values for the `number of record elements * number of array elements` must exist. The indicator variables are arranged in column format: the first 100 `STAFF-IND` values are for the `id` column, the next 100 `STAFF-IND` values are for the `name` column, and so on.

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
   01 STAFF-REC.
      03 STAFF-REC-ARR OCCURS 100 TIMES.
         05 ID     PIC S9(9).
         05 NAME   PIC X(11).
         05 DEPT   PIC S9(9).
         05 JOB    PIC X(6).
         05 YEARS  PIC S9(9).
         05 SALARY PIC S9(9).
         05 COMM   PIC S9(9).
   01 STAFF-IND PIC S9(4) OCCURS 700 TIMES.
   01 MIN-ID   PIC S9(9).
EXEC SQL END DECLARE SECTION END-EXEC.

MOVE 42 TO MIN-ID.
EXEC SQL DECLARE C0 CURSOR FOR
         SELECT ID,NAME,DEPT,JOB,YEARS,SALARY,COMM
         FROM STAFF WHERE ID >= :MIN-ID END-EXEC.
EXEC SQL OPEN C0 END-EXEC.
EXEC SQL WHENEVER NOT FOUND GOTO done END-EXEC.

GET-MORE.
EXEC SQL FETCH C0 INTO :STAFF-REC-ARR:STAFF-IND END-EXEC.
IF SQLCODE IN SQLCA < 0 THEN
   GO TO BOOFOO
   END-IF.
```

```
IF SQLCODE IN SQLCA = 100 THEN
   GO TO DONE
   END-IF.

IF SQLERRD(2) IN SQLCA > 0 THEN
   DISPLAY SQLERRD(2) IN SQLCA ''record(s) returned''
   ELSE DISPLAY ''No records returned''
   END-IF.
GO TO GET-MORE.
DONE.
```

The application could also be written as follows:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 STAFF-REC.
  03 ID     PIC S9(9) OCCURS 100 TIMES.
  03 NAME   PIC X(11) OCCURS 100 TIMES.
  03 DEPT   PIC S9(9) OCCURS 100 TIMES.
  03 JOB    PIC X(6)  OCCURS 100 TIMES.
  03 YEARS  PIC S9(9) OCCURS 100 TIMES.
  03 SALARY PIC S9(9) OCCURS 100 TIMES.
  03 COMM   PIC S9(9) OCCURS 100 TIMES.

01 STAFF-IND.
  03 ID-IND     PIC S9(4) OCCURS 100 TIMES.
  03 NAME-IND   PIC S9(4) OCCURS 100 TIMES.
  03 DEPT-IND   PIC S9(4) OCCURS 100 TIMES.
  03 JOB-IND    PIC S9(4) OCCURS 100 TIMES.
  03 YEARS-IND  PIC S9(4) OCCURS 100 TIMES.
  03 SALARY-IND PIC S9(4) OCCURS 100 TIMES.
  03 COMM-IND   PIC S9(4) OCCURS 100 TIMES.

01 MIN-ID   PIC S9(9).
EXEC SQL END DECLARE SECTION END-EXEC.

MOVE 42 to MIN-ID.
EXEC SQL DECLARE C0 CURSOR FOR
         SELECT ID,NAME,DEPT,JOB,YEARS,SALARY,COMM
         FROM STAFF WHERE ID >= :MIN-ID END-EXEC.
EXEC SQL OPEN C0 END-EXEC.
EXEC SQL WHENEVER NOT FOUND GOTO DONE END-EXEC.

GET-MORE.
EXEC SQL FETCH C0 INTO     :ID:ID-IND,
                           :NAME:NAME-IND,
                           :DEPT:DEPT-IND,
                           :JOB:JOB-IND,
                           :YEARS:YEARS-IND,
                           :SALARY:SALARY-IND,
                           :COMM:COMM-IND
END-EXEC.

IF SQLCODE IN SQLCA < 0 THEN
```

```
   GO TO BOOFOO
   END-IF.

IF SQLCODE IN SQLCA = 100 THEN
   GO TO DONE
   END-IF.

IF SQLERRD(2) IN SQLCA > 0 THEN
   DISPLAY SQLERRD(2) IN SQLCA ''record(s) returned''
   ELSE DISPLAY ''No records returned''
   END-IF.
GO TO GET-MORE.
DONE.
```

VORTEXcobol also enables applications use arrays on input. If the array always has every item filled, then you don't need to add any special embedded SQL. In the first array example, if the application always inserts 100 records, then a simple statement like this works:

```
EXEC SQL INSERT INTO STAFF
         VALUES(:STAFF-REC-ARR:STAFF-IND) END-EXEC.
```

However, if inserting less than the array size of records is possible, then you must use the following syntax:

```
EXEC SQL FOR :CNT INSERT INTO STAFF
               VALUES(:STAFF-REC-ARR:STAFF-IND) END-EXEC.
```

where `CNT` has the number of valid array entries.

Transactions

Some databases have implicit transaction management capabilities, starting new transactions automatically, while other databases require the user to explicitly control transactions. VORTEXcobol supports the following transaction control statements:

Statement	Description
BEGIN WORK	Begin a new transaction.
COMMIT WORK	Commit the current transaction.
ROLLBACK WORK	Abort the current transaction.

For databases that do not require a `BEGIN WORK`, the VORTEX driver ignores the command.

Converting Existing Embedded SQL Files

Converting from another vendor's precompiler to VORTEXcobol is typically transparent as long as the embedded SQL is ANSI and X/OPEN compliant. Some problems you might face in conversion include:

- ***Undocumented Behavior***

VORTEXcobol does not support any undocumented features in any database vendor's precompiler. If your application uses any undocumented features, you have to change them.

- ***Version Mismatches***

  Vendors concentrate on ensuring a high degree of backward compatibility, but often neglect forward compatibility. The ability to take an application written for a newer version of a precompiler and run it using an older version of the same precompiler is usually not an issue for the vendor. However, supporting forward compatibility can become a serious problem for a multiple-database product such as VORTEXcobol.

  When substantial compatibility problems exist, VORTEXcobol typically supports only the latest version of the database.

- ***Incompatible SQL Language Extensions***

  VORTEXcobol supports all ANSI-compliant SQL language statements and an increasing number of proprietary SQL extensions.

  However, if VORTEXcobol does not recognize a SQL language statement, it passes the statement unmodified to the database. At run time, if the database does not recognize your statement, you get an error.

  "SQL language" and "embedded SQL" are not the same set of commands. ANSI and X/OPEN define one standard for the SQL Language and another standard for embedded SQL. The SQL language consists of those SQL statements that are sent to the database at run time (that is, the SELECT statement). Embedded SQL is the set of SQL-like statements used in pre-compilation and/or compilation for example, the CONNECT and DECLARE statements. They are not sent to the database at runtime.

- ***Incompatible Embedded SQL Extensions***

  Some vendors support extensions to the ANSI standard for embedded SQL statements and few vendors flag these extensions in their documentation or at pre-compile time.

  VORTEXcobol supports all ANSI-compliant embedded SQL. In addition, VORTEXcobol parses some non-compliant embedded SQL statements and uses them to create correct and workable code.

  For more information on supported extensions, review "***Embedded SQL***" on page 29 and "***Extensions to SQL***" on page 34.

- ***Different & Incompatible Datatype Representations***

  Because different vendors' precompilers interpret certain datatypes such as `varchar` and `string` differently, even though the variables may be defined within the `EXEC SQL` and its bounding `END-EXEC`, the actual COBOL code used to manipulate data defined by these statements may have to vary depending on vendor.

  When generating these datatypes, VORTEXcobol tries to reconcile this incompatibility by interpreting these datatypes in a specific way dictated by the architecture of VORTEX. You may have to revise those portions of the COBOL code that manipulate these variables.

- ***Different Runtime Behaviors***

Databases can respond to SQL statements with substantially different behavior in areas such as locking, concurrency control, datatype conversion, error handling, transaction control, schema management and security, and cursor handling. While VORTEX hides many of these run-time differences, it cannot, for example, make Oracle look and act exactly like Sybase. VORTEX is designed as a "virtual" database interface. It provides a generous set of functions that are portable across multiple database engines. This virtual interface is different from every underlying database interface and therefore it does not necessarily provide the exact functionality of any specific vendor's database.

Read the ***VORTEXcli Reference Manual*** for a more detailed discussion of the function set.

Chapter 4
Advanced Topics

Using VORTEXaccelerator

VORTEXaccelerator improves system performance by matching SQL statements that are syntactically identical. It compares the SQL statements exactly as they are presented, including case. So, uppercase and lowercase characters are not identical and "Select" is not the same as "SELECT."

Case is usually not important to databases, so to make sure that VORTEXaccelerator makes as many matches as possible, set the `uppercase_sql` entry in the `vtxcob.ini` file to `yes`. All SQL statement identifiers are "upper-cased," creating more identical matches.

NOTE: *If your database is case-sensitive, like Sybase, this setting may cause problems.*

Managing Concurrent Connections

VORTEXcobol allows you keep multiple database connections open at one time. You must declare the connections and assign a connection identifier to each SQL statement.

To declare the connections, use the `DECLARE` *`dbid`* `DATABASE` command:

```
EXEC SQL DECLARE MYORA DATABASE END-EXEC.
EXEC SQL DECLARE MYDB2 DATABASE END-EXEC.
```

In the code, prefix the `AT` *`dbid`* qualifier on all SQL statements. For example,

```
EXEC SQL AT MYORA CONNECT TO :myoraconnect END-EXEC.
EXEC SQL AT MYDB2 CONNECT TO :mydb2connect END-EXEC.
.
.
.
EXEC SQL AT MYORA OPEN C1 END-EXEC.
EXEC SQL AT MYDB2 FETCH C3 INTO :myrec END-EXEC.
.
.
.
```

Host Variable Descriptors

VORTEXcobol uses a shortened version of the `SQLVAR` structure to describe host variables. Refer to the ***VORTEXcli Reference Manual*** for details on this structure. Most applications never manipulate these structures directly however some situations do require adjustment because the COBOL language does not have native datatypes to match certain database datatypes.

Representing Numeric Data

VORTEX uses an internal byte-array to represent numeric data. This format is machine-independent and provides a maximum precision of 38, a scale ranging from -84 to 127, and a magnitude range of 1.0E-129 to 9.99E125. The application can use this format directly by declaring a “number” variable as follows:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 nn PICTURE X(22).
EXEC SQL END DECLARE SECTION END-EXEC.

EXEC SQL SELECT NUMERICDATA INTO :nn
FROM TESTME WHERE PRIMARY_KEY = 42 END-EXEC.
EXEC SQL INSERT INTO TESTME VALUES(:nn) END-EXEC.
```

VORTEXcobol creates the following (abridged) COBOL code for these statements:

```
01 OO-8
   05 HDTY-0 PIC S9(4) COMP VALUE 1.
   05 HLEN-0 PIC S9(4) COMP VALUE 22.
   05 HVAR-0 USAGE POINTER.
   05 HIND-0 USAGE POINTER VALUE NULL.
01 II-8
   05 HDTY-0 PIC S9(4) COMP VALUE 1.
   05 HLEN-0 PIC S9(4) COMP VALUE 22.
   05 HVAR-0 USAGE POINTER.
   05 HIND-0 USAGE POINTER VALUE NULL.

CALL "VTXOPEN" USING BY VALUE 0,BY REFERENCE SQLCA,
      BY REFERENCE VTX-CSR(1),SS7
      BY VALUE 0,BY REFERENCE II-9
      BY VALUE 12,0.

CALL "VTXMOVE" USING BY VALUE 0,BY REFERENCE SQLCA,
      BY REFERENCE VTX-CSR(1),OO-8,
      BY VALUE 12,1,0.

CALL "VTXCLOS" USING BY VALUE 0,BY REFERENCE SQLCA,
      BY REFERENCE VTX-CSR(1),BY VALUE 0.

CALL "VTXEXEC" USING BY VALUE 0,BY REFERENCE SQLCA,
      BY REFERENCE VTX-CSR(2), SS8,
      BY VALUE 1,BY REFERENCE II-8,
      MY VALUE 12,1,0.
```

You must modify the host variable declarations to reflect the use of number instead of character:

```
   05 HDTY-0 PIC S9(4) COMP VALUE 2.
```

To determine the length of the returned data in `nn`, scan for either a NULL (0) byte or the end of the `nn` variable.

On input, the host variable descriptor must be updated with the correct length (for example, 6).

```
SET HLEN-0 IN II-8 TO 6.
```

You can use `TCVN2xx()` and `TCVxx2N()` to easily convert number information between a variety of external formats and the native VORTEX format.

Appendix A
Include Files

SQLCA

```
01 SQLCA.
 02 SQLCAID    PIC X(8) VALUE "SQLCA   ".
 02 SQLCABC    SIGNED-INT VALUE 128.
 02 SQLCODE    SIGNED-INT VALUE ZERO.
 02 SQLERRM.
  03 SQLERRML UNSIGNED-SHORT VALUE ZERO.
  03 SQLERRMC PIC X(70).
 02 SQLERRD    SIGNED-INT OCCURS 6 TIMES.
 02 SQLWARN.
  03 SQLWARN0 PIC X VALUE ZERO.
  03 SQLWARN1 PIC X VALUE ZERO.
  03 SQLWARN2 PIC X VALUE ZERO.
  03 SQLWARN3 PIC X VALUE ZERO.
  03 SQLWARN4 PIC X VALUE ZERO.
  03 SQLWARN5 PIC X VALUE ZERO.
  03 SQLWARN6 PIC X VALUE ZERO.
  03 SQLWARN7 PIC X VALUE ZERO.
 02 SQLEXT     PIC X(8).
```

Utility routine constants and structures

The `vtxcli.acu.cbl` (AcuCobol ANSI), `vtxcli.act.cbl` (AcuCobol terminal mode), `vtxcli.lib.cbl` (ANSI) and `vtxcli.lit.cbl` (terminal mode) files contain the constants and record definitions used by the Utility routines defined in the VORTEXcli Reference Manual. If you use any of these routines, you must COPY the correct library file.

`VTXCLI.ACU.CBL`

```
01 VORTEX-CLI-DATATYPES.
    05 TDT-INTEGER  UNSIGNED-INT VALUE  0.
    05 TDT-CHAR     UNSIGNED-INT VALUE  1.
    05 TDT-NUMBER   UNSIGNED-INT VALUE  2.
    05 TDT-NTCHAR   UNSIGNED-INT VALUE  3.
```

```
    05 TDT-PACKED   UNSIGNED-INT VALUE  4.
    05 TDT-ZONED    UNSIGNED-INT VALUE  5.
    05 TDT-FLOAT    UNSIGNED-INT VALUE  8.
    05 TDT-VARCHAR  UNSIGNED-INT VALUE  9.
    05 TDT-BLOB     UNSIGNED-INT VALUE 10.
    05 TDT-CLOB     UNSIGNED-INT VALUE 11.
    05 TDT-DATATIME UNSIGNED-INT VALUE 12.
    05 TDT-BINARY   UNSIGNED-INT VALUE 99.

01 VORTEX-CLI-DB-IDS.
    05 TDT-DID-ORACLE    UNSIGNED-INT VALUE   0.
    05 TDT-DID-RDB       UNSIGNED-INT VALUE   1.
    05 TDT-DID-SYBASE    UNSIGNED-INT VALUE   2.
    05 TDT-DID-NET       UNSIGNED-INT VALUE   3.
    05 TDT-DID-GENESIS   UNSIGNED-INT VALUE   4.
    05 TDT-DID-INFORMIX  UNSIGNED-INT VALUE   5.
    05 TDT-DID-ALLBASE   UNSIGNED-INT VALUE   6.
    05 TDT-DID-DB2       UNSIGNED-INT VALUE   7.
    05 TDT-DID-INGRES    UNSIGNED-INT VALUE   8.
    05 TDT-DID-ADABASC   UNSIGNED-INT VALUE   9.
    05 TDT-DID-ADABASD   UNSIGNED-INT VALUE  10.
    05 TDT-DID-ODBC      UNSIGNED-INT VALUE  11.
    05 TDT-DID-SQLSERVER UNSIGNED-INT VALUE  12.
    05 TDT-DID-TERADATA  UNSIGNED-INT VALUE  13.
    05 TDT-DID-ODBX      UNSIGNED-INT VALUE  16.

    05 TDT-DID-MAX       UNSIGNED-INT VALUE  16.
    05 TDT-DID-MASK      UNSIGNED-INT VALUE 255.

01 VORTEX-CLI-DB-DRIVER-CMDS.
    05 TDT-CMD-TIMEOUT      UNSIGNED-INT VALUE  0.
    05 TDT-CMD-SINGLEPID    UNSIGNED-INT VALUE  1.
    05 TDT-CMD-OOPT         UNSIGNED-INT VALUE  2.
    05 TDT-CMD-USEDB        UNSIGNED-INT VALUE  3.
    05 TDT-CMD-LANGUAGE     UNSIGNED-INT VALUE  4.
    05 TDT-CMD-RAW-DATETIME UNSIGNED-INT VALUE  5.
    05 TDT-CMD-VERSION      UNSIGNED-INT VALUE  6.
    05 TDT-CMD-CHAR         UNSIGNED-INT VALUE  7.
    05 TDT-CMD-LOCK         UNSIGNED-INT VALUE  8.
    05 TDT-CMD-ONEBYTE      UNSIGNED-INT VALUE  9.
    05 TDT-CMD-MULTISTMT    UNSIGNED-INT VALUE 10.
    05 TDT-CMD-AUTOCOMMIT   UNSIGNED-INT VALUE 11.

01 VORTEX-CLI-DB-DRIVER-FILE-CMDS.
    05 TDT-FILE-OPEN   UNSIGNED-INT VALUE 0.
    05 TDT-FILE-CLOSE  UNSIGNED-INT VALUE 1.
    05 TDT-FILE-READ   UNSIGNED-INT VALUE 2.
    05 TDT-FILE-WRITE  UNSIGNED-INT VALUE 3.
    05 TDT-FILE-DELETE UNSIGNED-INT VALUE 4.


01 VORTEX-CLI-MISC-DEFINITIONS.
```

```
           05 TDT-PRE-POPEN    UNSIGNED-INT VALUE 1.
           05 TDT-PRE-PEXEC    UNSIGNED-INT VALUE 1.
           05 TDT-PRE-FUO      UNSIGNED-INT VALUE 2.
           05 TDT-PRE-BLOB     UNSIGNED-INT VALUE 4.

           05 TDT-EMAP_OK      UNSIGNED-INT VALUE 0.
           05 TDT-EMAP_NOFILE  UNSIGNED-INT VALUE 1.
           05 TDT-EMAP_BADFILE UNSIGNED-INT VALUE 2.
           05 TDT-EMAP_NOMEM   UNSIGNED-INT VALUE 3.

       01 VORTEX-CLI-TDB-OPTS.
         03 TDB-OPTS.
           05 TDB-DTB          UNSIGNED-INT.
           05 TDB-DTF          PIC X(64).
           05 TDB-NMK          UNSIGNED-SHORT.
```

VTXCLI.ACT.CBL

```
   01 VORTEX-CLI-DATATYPES.
     05 TDT-INTEGER  UNSIGNED-INT VALUE  0.
     05 TDT-CHAR     UNSIGNED-INT VALUE  1.
     05 TDT-NUMBER   UNSIGNED-INT VALUE  2.
     05 TDT-NTCHAR   UNSIGNED-INT VALUE  3.
     05 TDT-PACKED   UNSIGNED-INT VALUE  4.
     05 TDT-ZONED    UNSIGNED-INT VALUE  5.
     05 TDT-FLOAT    UNSIGNED-INT VALUE  8.
     05 TDT-VARCHAR  UNSIGNED-INT VALUE  9.
     05 TDT-BLOB     UNSIGNED-INT VALUE 10.
     05 TDT-CLOB     UNSIGNED-INT VALUE 11.
     05 TDT-DATATIME UNSIGNED-INT VALUE 12.
     05 TDT-BINARY   UNSIGNED-INT VALUE 99.

   01 VORTEX-CLI-DB-IDS.
     05 TDT-DID-ORACLE    UNSIGNED-INT VALUE   0.
     05 TDT-DID-RDB       UNSIGNED-INT VALUE   1.
     05 TDT-DID-SYBASE    UNSIGNED-INT VALUE   2.
     05 TDT-DID-NET       UNSIGNED-INT VALUE   3.
     05 TDT-DID-GENESIS   UNSIGNED-INT VALUE   4.
     05 TDT-DID-INFORMIX  UNSIGNED-INT VALUE   5.
     05 TDT-DID-ALLBASE   UNSIGNED-INT VALUE   6.
     05 TDT-DID-DB2       UNSIGNED-INT VALUE   7.
     05 TDT-DID-INGRES    UNSIGNED-INT VALUE   8.
     05 TDT-DID-ADABASC   UNSIGNED-INT VALUE   9.
     05 TDT-DID-ADABASD   UNSIGNED-INT VALUE  10.
     05 TDT-DID-ODBC      UNSIGNED-INT VALUE  11.
     05 TDT-DID-SQLSERVER UNSIGNED-INT VALUE  12.
     05 TDT-DID-TERADATA  UNSIGNED-INT VALUE  13.
     05 TDT-DID-ODBX      UNSIGNED-INT VALUE  16.

     05 TDT-DID-MAX       UNSIGNED-INT VALUE  16.
     05 TDT-DID-MASK      UNSIGNED-INT VALUE 255.
```

```
   01 VORTEX-CLI-DB-DRIVER-CMDS.
     05 TDT-CMD-TIMEOUT      UNSIGNED-INT VALUE  0.
     05 TDT-CMD-SINGLEPID    UNSIGNED-INT VALUE  1.
     05 TDT-CMD-OOPT         UNSIGNED-INT VALUE  2.
     05 TDT-CMD-USEDB        UNSIGNED-INT VALUE  3.
     05 TDT-CMD-LANGUAGE     UNSIGNED-INT VALUE  4.
     05 TDT-CMD-RAW-DATETIME UNSIGNED-INT VALUE  5.
     05 TDT-CMD-VERSION      UNSIGNED-INT VALUE  6.
     05 TDT-CMD-CHAR         UNSIGNED-INT VALUE  7.
     05 TDT-CMD-LOCK         UNSIGNED-INT VALUE  8.
     05 TDT-CMD-ONEBYTE      UNSIGNED-INT VALUE  9.
     05 TDT-CMD-MULTISTMT    UNSIGNED-INT VALUE 10.
     05 TDT-CMD-AUTOCOMMIT   UNSIGNED-INT VALUE 11.

   01 VORTEX-CLI-DB-DRIVER-FILE-CMDS.
     05 TDT-FILE-OPEN   UNSIGNED-INT VALUE 0.
     05 TDT-FILE-CLOSE  UNSIGNED-INT VALUE 1.
     05 TDT-FILE-READ   UNSIGNED-INT VALUE 2.
     05 TDT-FILE-WRITE  UNSIGNED-INT VALUE 3.
     05 TDT-FILE-DELETE UNSIGNED-INT VALUE 4.

   01 VORTEX-CLI-MISC-DEFINITIONS.
     05 TDT-PRE-POPEN    UNSIGNED-INT VALUE 1.
     05 TDT-PRE-PEXEC    UNSIGNED-INT VALUE 1.
     05 TDT-PRE-FUO      UNSIGNED-INT VALUE 2.
     05 TDT-PRE-BLOB     UNSIGNED-INT VALUE 4.

     05 TDT-EMAP_OK      UNSIGNED-INT VALUE 0.
     05 TDT-EMAP_NOFILE  UNSIGNED-INT VALUE 1.
     05 TDT-EMAP_BADFILE UNSIGNED-INT VALUE 2.
     05 TDT-EMAP_NOMEM   UNSIGNED-INT VALUE 3.

   01 VORTEX-CLI-TDB-OPTS.
    03 TDB-OPTS.
     05 TDB-DTB          UNSIGNED-INT.
     05 TDB-DTF          PIC X(64).
     05 TDB-NMK          UNSIGNED-SHORT.

VTXCLI.LIB.CBL

       01 VORTEX-CLI-DATATYPES.
           05 TDT-INTEGER  PIC S9(4) COMP-5 VALUE  0.
           05 TDT-CHAR     PIC S9(4) COMP-5 VALUE  1.
           05 TDT-NUMBER   PIC S9(4) COMP-5 VALUE  2.
           05 TDT-NTCHAR   PIC S9(4) COMP-5 VALUE  3.
           05 TDT-PACKED   PIC S9(4) COMP-5 VALUE  4.
           05 TDT-ZONED    PIC S9(4) COMP-5 VALUE  5.
           05 TDT-FLOAT    PIC S9(4) COMP-5 VALUE  8.
           05 TDT-VARCHAR  PIC S9(4) COMP-5 VALUE  9.
           05 TDT-BLOB     PIC S9(4) COMP-5 VALUE 10.
           05 TDT-CLOB     PIC S9(4) COMP-5 VALUE 11.
           05 TDT-DATATIME PIC S9(4) COMP-5 VALUE 12.
```

```
    05 TDT-BINARY   PIC S9(4) COMP-5 VALUE 99.

01 VORTEX-CLI-DB-IDS.
    05 TDT-DID-ORACLE    PIC S9(4) COMP-5 VALUE   0.
    05 TDT-DID-RDB       PIC S9(4) COMP-5 VALUE   1.
    05 TDT-DID-SYBASE    PIC S9(4) COMP-5 VALUE   2.
    05 TDT-DID-NET       PIC S9(4) COMP-5 VALUE   3.
    05 TDT-DID-GENESIS   PIC S9(4) COMP-5 VALUE   4.
    05 TDT-DID-INFORMIX  PIC S9(4) COMP-5 VALUE   5.
    05 TDT-DID-ALLBASE   PIC S9(4) COMP-5 VALUE   6.
    05 TDT-DID-DB2       PIC S9(4) COMP-5 VALUE   7.
    05 TDT-DID-INGRES    PIC S9(4) COMP-5 VALUE   8.
    05 TDT-DID-ADABASC   PIC S9(4) COMP-5 VALUE   9.
    05 TDT-DID-ADABASD   PIC S9(4) COMP-5 VALUE  10.
    05 TDT-DID-ODBC      PIC S9(4) COMP-5 VALUE  11.
    05 TDT-DID-SQLSERVER PIC S9(4) COMP-5 VALUE  12.
    05 TDT-DID-TERADATA  PIC S9(4) COMP-5 VALUE  13.
    05 TDT-DID-ODBX      PIC S9(4) COMP-5 VALUE  16.

    05 TDT-DID-MAX       PIC S9(4) COMP-5 VALUE  16.
    05 TDT-DID-MASK      PIC S9(4) COMP-5 VALUE 255.

01 VORTEX-CLI-DB-DRIVER-CMDS.
    05 TDT-CMD-TIMEOUT      PIC S9(4) COMP-5 VALUE  0.
    05 TDT-CMD-SINGLEPID    PIC S9(4) COMP-5 VALUE  1.
    05 TDT-CMD-OOPT         PIC S9(4) COMP-5 VALUE  2.
    05 TDT-CMD-USEDB        PIC S9(4) COMP-5 VALUE  3.
    05 TDT-CMD-LANGUAGE     PIC S9(4) COMP-5 VALUE  4.
    05 TDT-CMD-RAW-DATETIME PIC S9(4) COMP-5 VALUE  5.
    05 TDT-CMD-VERSION      PIC S9(4) COMP-5 VALUE  6.
    05 TDT-CMD-CHAR         PIC S9(4) COMP-5 VALUE  7.
    05 TDT-CMD-LOCK         PIC S9(4) COMP-5 VALUE  8.
    05 TDT-CMD-ONEBYTE      PIC S9(4) COMP-5 VALUE  9.
    05 TDT-CMD-MULTISTMT    PIC S9(4) COMP-5 VALUE 10.
    05 TDT-CMD-AUTOCOMMIT   PIC S9(4) COMP-5 VALUE 11.

01 VORTEX-CLI-DB-DRIVER-FILE-CMDS.
    05 TDT-FILE-OPEN   PIC S9(4) COMP-5 VALUE 0.
    05 TDT-FILE-CLOSE  PIC S9(4) COMP-5 VALUE 1.
    05 TDT-FILE-READ   PIC S9(4) COMP-5 VALUE 2.
    05 TDT-FILE-WRITE  PIC S9(4) COMP-5 VALUE 3.
    05 TDT-FILE-DELETE PIC S9(4) COMP-5 VALUE 4.

01 VORTEX-CLI-MISC-DEFINITIONS.
    05 TDT-PRE-POPEN    PIC S9(4) COMP-5 VALUE 1.
    05 TDT-PRE-PEXEC    PIC S9(4) COMP-5 VALUE 1.
    05 TDT-PRE-FUO      PIC S9(4) COMP-5 VALUE 2.
    05 TDT-PRE-BLOB     PIC S9(4) COMP-5 VALUE 4.

    05 TDT-EMAP_OK      PIC S9(4) COMP-5 VALUE 0.
    05 TDT-EMAP_NOFILE  PIC S9(4) COMP-5 VALUE 1.
    05 TDT-EMAP_BADFILE PIC S9(4) COMP-5 VALUE 2.
```

```
           05 TDT-EMAP_NOMEM   PIC S9(4) COMP-5 VALUE 3.

       01 VORTEX-CLI-TDB-OPTS.
         03 TDB-OPTS.
           05 TDB-DTB          PIC S9(4) COMP-5.
           05 TDB-DTF          PIC X(64).
           05 TDB-NMK          PIC S9(2) COMP-5.
```

VTXCLI.LIT.CBL

```
   01 VORTEX-CLI-DATATYPES.
     05 TDT-INTEGER  PIC S9(4) COMP-5 VALUE  0.
     05 TDT-CHAR     PIC S9(4) COMP-5 VALUE  1.
     05 TDT-NUMBER   PIC S9(4) COMP-5 VALUE  2.
     05 TDT-NTCHAR   PIC S9(4) COMP-5 VALUE  3.
     05 TDT-PACKED   PIC S9(4) COMP-5 VALUE  4.
     05 TDT-ZONED    PIC S9(4) COMP-5 VALUE  5.
     05 TDT-FLOAT    PIC S9(4) COMP-5 VALUE  8.
     05 TDT-VARCHAR  PIC S9(4) COMP-5 VALUE  9.
     05 TDT-BLOB     PIC S9(4) COMP-5 VALUE 10.
     05 TDT-CLOB     PIC S9(4) COMP-5 VALUE 11.
     05 TDT-DATATIME PIC S9(4) COMP-5 VALUE 12.
     05 TDT-BINARY   PIC S9(4) COMP-5 VALUE 99.

   01 VORTEX-CLI-DB-IDS.
     05 TDT-DID-ORACLE    PIC S9(4) COMP-5 VALUE   0.
     05 TDT-DID-RDB       PIC S9(4) COMP-5 VALUE   1.
     05 TDT-DID-SYBASE    PIC S9(4) COMP-5 VALUE   2.
     05 TDT-DID-NET       PIC S9(4) COMP-5 VALUE   3.
     05 TDT-DID-GENESIS   PIC S9(4) COMP-5 VALUE   4.
     05 TDT-DID-INFORMIX  PIC S9(4) COMP-5 VALUE   5.
     05 TDT-DID-ALLBASE   PIC S9(4) COMP-5 VALUE   6.
     05 TDT-DID-DB2       PIC S9(4) COMP-5 VALUE   7.
     05 TDT-DID-INGRES    PIC S9(4) COMP-5 VALUE   8.
     05 TDT-DID-ADABASC   PIC S9(4) COMP-5 VALUE   9.
     05 TDT-DID-ADABASD   PIC S9(4) COMP-5 VALUE  10.
     05 TDT-DID-ODBC      PIC S9(4) COMP-5 VALUE  11.
     05 TDT-DID-SQLSERVER PIC S9(4) COMP-5 VALUE  12.
     05 TDT-DID-TERADATA  PIC S9(4) COMP-5 VALUE  13.
     05 TDT-DID-ODBX      PIC S9(4) COMP-5 VALUE  16.

     05 TDT-DID-MAX       PIC S9(4) COMP-5 VALUE  16.
     05 TDT-DID-MASK      PIC S9(4) COMP-5 VALUE 255.

   01 VORTEX-CLI-DB-DRIVER-CMDS.
     05 TDT-CMD-TIMEOUT      PIC S9(4) COMP-5 VALUE  0.
     05 TDT-CMD-SINGLEPID    PIC S9(4) COMP-5 VALUE  1.
     05 TDT-CMD-OOPT         PIC S9(4) COMP-5 VALUE  2.
     05 TDT-CMD-USEDB        PIC S9(4) COMP-5 VALUE  3.
     05 TDT-CMD-LANGUAGE     PIC S9(4) COMP-5 VALUE  4.
     05 TDT-CMD-RAW-DATETIME PIC S9(4) COMP-5 VALUE  5.
     05 TDT-CMD-VERSION      PIC S9(4) COMP-5 VALUE  6.
```

```
  05 TDT-CMD-CHAR         PIC S9(4) COMP-5 VALUE  7.
  05 TDT-CMD-LOCK         PIC S9(4) COMP-5 VALUE  8.
  05 TDT-CMD-ONEBYTE      PIC S9(4) COMP-5 VALUE  9.
  05 TDT-CMD-MULTISTMT    PIC S9(4) COMP-5 VALUE 10.
  05 TDT-CMD-AUTOCOMMIT   PIC S9(4) COMP-5 VALUE 11.

01 VORTEX-CLI-DB-DRIVER-FILE-CMDS.
  05 TDT-FILE-OPEN   PIC S9(4) COMP-5 VALUE 0.
  05 TDT-FILE-CLOSE  PIC S9(4) COMP-5 VALUE 1.
  05 TDT-FILE-READ   PIC S9(4) COMP-5 VALUE 2.
  05 TDT-FILE-WRITE  PIC S9(4) COMP-5 VALUE 3.
  05 TDT-FILE-DELETE PIC S9(4) COMP-5 VALUE 4.

01 VORTEX-CLI-MISC-DEFINITIONS.
  05 TDT-PRE-POPEN    PIC S9(4) COMP-5 VALUE 1.
  05 TDT-PRE-PEXEC    PIC S9(4) COMP-5 VALUE 1.
  05 TDT-PRE-FUO      PIC S9(4) COMP-5 VALUE 2.
  05 TDT-PRE-BLOB     PIC S9(4) COMP-5 VALUE 4.

  05 TDT-EMAP_OK      PIC S9(4) COMP-5 VALUE 0.
  05 TDT-EMAP_NOFILE  PIC S9(4) COMP-5 VALUE 1.
  05 TDT-EMAP_BADFILE PIC S9(4) COMP-5 VALUE 2.
  05 TDT-EMAP_NOMEM   PIC S9(4) COMP-5 VALUE 3.

01 VORTEX-CLI-TDB-OPTS.
 03 TDB-OPTS.
  05 TDB-DTB          PIC S9(4) COMP-5.
  05 TDB-DTF          PIC X(64).
  05 TDB-NMK          PIC S9(4) COMP-5.
```

Appendix B
Embedded SQL

This section describes the syntax understood by VORTEXcobol when it is in embedded SQL mode. When VORTEXcobol reaches the end of a statement in embedded SQL mode, it switches back to COBOL mode. An embedded SQL statement ends with a semicolon.

In the syntax chart below, vendor extensions are indicated with a notation such as `Rdb` or `Oracle` and unsupported constructs are printed in italics.

Any SQL statements that are not recognized by VORTEXcobol are passed to the database unchanged.

```
embedded-sql :==
   EXEC SQL sql-statement ;
   $ sql_statement ; <Informix>

sql-statement :==
   BEGIN DECLARE SECTION
   | END DECLARE SECTION
   | BEGIN <Ingres>                              /* start select loop */
   | END <Ingres>                                /* end select loop */
   | ENDSELECT <Ingres>                          /* break out of select loop */
   | BEGIN WORK
   | CLOSE { cursor-name | parameter <Rdb> }
   | commit-statement
   | connect-statement <Oracle>
   | DATABASE {db-name | :host-string-variable} [EXCLUSIVE]<Informix>
   | declare-database-statement <Oracle>
   | declare-cursor-statement
   | declare-list-cursor-statement <Rdb>
   | dynamic-declare-cursor
   | declare-schema-statement
   | [ AT db-name ] <Oracle> DECLARE statement-name-list STATEMENT
   | describe-statement
   | execute-statement
   | execute-immediate-statement
   | fetch-statement
   | FLUSH { cursor-name | parameter }  <Informix>
   | FREE { statement-name | cursor-name | parameter } <Informix>
   | disconnect-statement
   | include-statement
   | open-statement


   | prepare-statement
   | put-statement <Informix>
   | release-statement-statement
   | rollback-statement
   | savepoint-statement
   | set-sql-statement
```

```
   | set-transaction-statement
   | whenever-statement

statement-name-list :==
   statement-name [ , statement-name-list ]

commit-statement :==
   [ AT db-name ] <Oracle>
   COMMIT [ WORK ] [ RELEASE <Oracle> ]

<Oracle>
connect-statement :==
   CONNECT { string | :host-string-variable }
     [ IDENTIFIED BY { string | :host-string-variable } ]
     [ [ AT db-name ] USING { string | :host-string-variable }]

<Ingres>
ingres-connect-statement :==
   CONNECT dbname
     [ SESSION integer ]
     [ IDENTIFIED BY username ]

declare-cursor-statement :==
   [ AT db-name ] <Oracle>
     DECLARE cursor-name
     [ INSERT ONLY | READ ONLY ] <Rdb>
     [ TABLE ] <Rdb>
     [ SCROLL ] <Informix>
     CURSOR
     [ WITH HOLD ] <Informix>
     FOR
     { select-expression
         [ order-by-clause]
         [ limit-to-clause ] <Rdb>
         [ FOR UPDATE OF column-list ]
       | insert-statement   <Informix>
       | statement-name  <Informix> }

<Oracle>
declare-database-statement :==
   DECLARE db-name DATABASE

select-expression :==
   { select-clause | ( select-expression ) }
     [ UNION [ ALL ] select-expression ]


select-clause :==
   SELECT [ ALL | DISTINCT ] select-list
     FROM select-table-list
     [ WHERE predicate ]
     [ GROUP BY column-list ]
     [ HAVING predicate ]

select-table-list :==
   table-or-view [ alias ] [ , select-table-list ]
```

```
order-by-clause :==
   ORDER BY order-by-list

order-by-list :==
   { column-name | integer } [ ASC | DESC ] [ , order-by-list ]

column-list :==
   column-name [ , column-list ]

<Rdb>
limit-to-clause :==
   LIMIT TO row-limit ROWS

<Rdb>
declare-list-cursor-statement :==
   DECLARE cursor-name
     [ INSERT ONLY | READ ONLY ]
     LIST CURSOR FOR SELECT column-name WHERE CURRENT OF table-cursor-name

<Rdb>
dynamic-declare-cursor :==
   DECLARE parameter1
     [ INSERT ONLY | READ ONLY ]
     [ TABLE | LIST ]
     CURSOR FOR parameter2

<Rdb>
declare-schema-statement :==
   DECLARE SCHEMA FILENAME vms-file-specification

describe-statement :==
   DESCRIBE [ SELECT LIST FOR ] <Oracle> { statement-name | parameter <Rdb> }
     [ SELECT LIST [FOR <Oracle>]
       | MARKERS <Rdb>
       | BIND VARIABLES FOR <Oracle> ]
     { INTO | USING <Ingres> } descriptor-name
     [ USING NAMES ] <Ingres>



execute-statement :==
   [ AT db-name ] <Oracle>
     [FOR :host-integer] <Oracle>
     EXECUTE { statement-name | parameter <Rdb> }
     [ USING { parameter-list | DESCRIPTOR descriptor-name } ]

execute-immediate-statement :==
   [ AT db-name ] <Oracle>
     EXECUTE IMMEDIATE parameter
     [ { INTO variable-list
         | USING [ DESCRIPTOR ] descriptor-name }
       [ EXEC SQL BEGIN ;
         host-language-code
         EXEC SQL END ;
       ]
     ]  <Ingres>
```

```
fetch-statement :==
   FETCH
     [ NEXT | PREVIOUS | PRIOR | FIRST | LAST | CURRENT
       | RELATIVE integer | ABSOLUTE integer ] <Informix>
     { cursor-name | parameter <Rdb> }
     [ INTO parameter-list | USING DESCRIPTOR descriptor-name ]

parameter-list :==
   parameter [ , parameter-list ]

parameter :==
   : host-variable
   : host-variable : indicator ]
   | $ host-variable $ indicator       <Informix>
   | $ host-variable : indicator       <Informix>
   | : host-variable $ indicator       <Informix>
   | : host-variable INDICATOR : indicator  <Ingres>

disconnect-statement :==
   CLOSE DATABASE <Informix>
   | FINISH <Rdb>
   | DISCONNECT [ SESSION integer | ALL ] <Ingres>
   | RELEASE

include-statement :==
   INCLUDE { SQLCA | SQLDA | file-spec }

open-statement :==
   [ FOR :host-integer ] <Oracle>
     OPEN { cursor-name | parameter <Rdb> }
     [ USING { parameter-list | DESCRIPTOR descriptor-name } ]
     [ FOR READONLY ] <Ingres>



prepare-statement :==
   PREPARE { statement-name | parameter <Rdb> }
     [ [ SELECT LIST ] INTO descriptor-name <Rdb> <Oracle> <Ingres>
       | USING DESCRIPTOR descriptor-name <Ingres> ]
     FROM { statement-string | parameter }

<Informix>
put-statement :==
   PUT cursor-name
     [ USING DESCRIPTOR descriptor-name | FROM host-variable-list ]

release-statement-statement :==
   RELEASE { statement-name | parameter }

rollback-statement :==
   [ AT db-name ] <Oracle>
     ROLLBACK [ WORK <Oracle> ]
     [ TO [ SAVEPOINT ] savepoint-name ] <Oracle> <Ingres>
     [ RELEASE ] <Oracle>

<Ingres> <Oracle>
savepoint-statement :==
```

```
   [ AT db-name ] <Oracle>
     SAVEPOINT savepoint-name

<Ingres>
set-sql-statement :==
   SET_SQL ( SESSION = { integer | :host-integer } )

<Rdb>
set-transaction-statement :==
   SET TRANSACTION [ READ ONLY | READ WRITE ]

whenever-statement :==
   WHENEVER
     { NOT FOUND | SQLERROR | SQLWARNING }
     { CONTINUE
       | GOTO label
       | GO TO label
       | STOP <Oracle>
       | CALL procedure <Ingres>
     }
```

Appendix C
Extensions to SQL

VORTEXcobol supports the following extensions to Embedded SQL. These extensions work with all supported databases.

Declaring a Transaction

The DECLARE TRANSACTION statement is supported syntactically but performs no function at this time.

Declaring a Table

The DECLARE TABLE statement is supported syntactically but performs no function at this time.

DECLARE SECTION

The DECLARE SECTION supports and may include:

- ***Structures and arrays of structures.*** When you declare a structure variable in a DECLARE SECTION, you must also declare the the definition of the structure itself in a DECLARE SECTION.
- ***Host language comments***.
- ***Union definitions***.

VORTEX commands

VORTEX supports passthrough commands that set certain VORTEX driver options. There are several VORTEX driver option that are outside of the ANSI embedded SQL standard. These are set using the VTXCMD function. While it is possible to code these calls directly, it is more convenient to use the extension. The syntax is

```
EXEC SQL VORTEX [CURSOR <name>] <cmd> <int>|<string>END-EXEC.
```

where [CURSOR <name>] is optional and is currently only used for the Oracle OOPT command, <cmd> is the actual command as found in the vortex.h file but without the 'TDB_CMD_' prefix and <int>|<string> are the parameters for the command. <string> must be enclosed within quotes. For example, in order to get the hostname of the server where the database driver is running,

```
EXEC SQL VORTEX HOSTNAME "" END-EXEC.
```

Then check the SQLERRMC buffer.

Please refer to the VORTEXcli document for appropriate values for <cmd>.

Host Variables

Host variable scope follows the normal scoping rules for host language variables. Structure members may be used as input/output host variables.

Structures may be used as output (INTO) host variables or in the VALUES list of an INSERT clause. For example,

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.

01  STAFF-REC.
05  H-ID           PIC 9(3)     COMP-5 VALUE ZERO.
05  H-NAME         PIC X(10)           VALUE SPACES.
05  H-DEPT         PIC 9(2)     COMP-5 VALUE ZERO.
05  H-JOB          PIC X(6)            VALUE SPACES.
05  H-YEARS        PIC 9(2)     COMP-5 VALUE ZERO.
05  H-SALARY       PIC S9(9)V99 COMP-3 VALUE ZERO.
05  H-COMM         PIC S9(9)V99 COMP-3 VALUE ZERO.

EXEC SQL END DECLARE SECTION END-EXEC.

EXEC SQL SELECT * INTO :STAFF-REC FROM STAFF
   WHERE ID = 20
END-EXEC
```

When a structure is used as a host variable, the result is the same as if you had used a list of all the fields in the structure. Thus, the SELECT statement above is the same as this:

```
EXEC SQL SELECT * INTO     :H-ID, :H-NAME, :H-DEPT, :H-JOB,
                           :H-YEARS, :H-SALARY, :H-COMM
                           FROM STAFF WHERE ID = 20
END-EXEC
```

When using a structure host variable, an indicator variable, if present, must be an array of short integers with at least as many elements as there are members of the structure. The array elements are the individual indicators for the structure members, starting at array element 0 for the first structure member, element 1 for the second, and so on. For example,

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.

01  STAFF-REC.
   05  H-ID           PIC 9(3)     COMP-5 VALUE ZERO.
   05  H-NAME         PIC X(10)           VALUE SPACES.
   05  H-DEPT         PIC 9(2)     COMP-5 VALUE ZERO.
   05  H-JOB          PIC X(6)            VALUE SPACES.
   05  H-YEARS        PIC 9(2)     COMP-5 VALUE ZERO.
   05  H-SALARY       PIC S9(9)V99 COMP-3 VALUE ZERO.
   05  H-COMM         PIC S9(9)V99 COMP-3 VALUE ZERO.
01  STAFF-IND          PIC S9(4)    COMP-5 OCCURS 7 TIMES.
```

```
EXEC SQL END DECLARE SECTION END-EXEC.

EXEC SQL FETCH STAFF INTO :STAFF-REC:STAFF-IND END-EXEC.
```

In this example

```
ELEMENT                    INDICATOR

H-ID                       STAFF-IND(1)
H-NAME                     STAFF-IND(2)
H-DEPT                     STAFF-IND(3)
H-JOB                      STAFF-IND(4)
H-YEARS                    STAFF-IND(5)
H-SALARY                   STAFF-IND(6)
H-COMM                     STAFF-IND(7)
```

Arrays of structures may be used as output (INTO) host variables. For example,

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.

01  STAFF-REC.
   03  STAFF-REC-ARR OCCURS 10 TIMES.
      05  H-ID          PIC 9(3)     COMP-5 VALUE ZERO.
      05  H-NAME        PIC X(10)           VALUE SPACES.
      05  H-DEPT        PIC 9(2)     COMP-5 VALUE ZERO.
      05  H-JOB         PIC X(6)            VALUE SPACES.
      05  H-YEARS       PIC 9(2)     COMP-5 VALUE ZERO.
      05  H-SALARY      PIC S9(9)V99 COMP-3 VALUE ZERO.
      05  H-COMM        PIC S9(9)V99 COMP-3 VALUE ZERO.

EXEC SQL END DECLARE SECTION END-EXEC.

      EXEC SQL SELECT ID,NAME,DEPT,JOB,YEARS,SALARY,COMM
         INTO :STAFF-REC-ARR FROM STAFF WHERE ID = 20
      END-EXEC
```

A structure may be used as an input host variable where a list is required.

The indicators for an array of structures must be an array of short integers with at least as many elements as there are members of all the structures in the array. The array elements are the individual indicators for the structure members, starting at array element 0 for the first structure member of the first array element, element arraysize for the second, element arraysize*2 for the third, and so on. For example,

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.

01  STAFF-REC.
   03  STAFF-REC-ARR OCCURS 10 TIMES.
      05  H-ID          PIC 9(3)     COMP-5 VALUE ZERO.
      05  H-NAME        PIC X(10)           VALUE SPACES.
      05  H-DEPT        PIC 9(2)     COMP-5 VALUE ZERO.
      05  H-JOB         PIC X(6)            VALUE SPACES.
      05  H-YEARS       PIC 9(2)     COMP-5 VALUE ZERO.
```

```
      05  H-SALARY      PIC S9(9)V99 COMP-3 VALUE ZERO.
      05  H-COMM        PIC S9(9)V99 COMP-3 VALUE ZERO.
01  STAFF-IND             PIC S9(4)    COMP-5 OCCURS 70 TIMES.

EXEC SQL END DECLARE SECTION END-EXEC.

EXEC SQL SELECT * INTO :STAFF-REC-ARR FROM STAFF
   WHERE ID = 20
END-EXEC
```

In this example

```
ELEMENT                     INDICATOR

H-ID(1)                     STAFF-IND(1)
H-NAME(1)                   STAFF-IND(11)
    .                           .
    .                           .
H-COMM(1)                   STAFF-IND(61)
    .                           .
    .                           .
H-ID(2)                     STAFF-IND(2)
H-NAME(2)                   STAFF-IND(12)
    .                           .
    .                           .
H-COMM(2)                   STAFF-IND(62)

    .                           .
    .                           .
H-ID(3)                     STAFF-IND(3)
H-NAME(3)                   STAFF-IND(13)

    .                           .
    .                           .
H-COMM(10)                  STAFF-IND(70)
```

An integer host variable may be used where a cursor name is normally required.

A structure array may be used in the VALUES clause of an INSERT statement. For example,

```
EXEC SQL INSERT INTO STAFF VALUES (:STAFF-REC) END-EXEC.
EXEC SQL INSERT INTO STAFF VALUES (:STAFF-REC-ARR) END-EXEC.
```

SQLCA

The name of the structure is SQLCA and all of its elements have uppercase names .

SQLERRM is an additional member. It is a structure consisting of:

- SQLERRML, a short integer which contains the length of the text of the most recent error message returned by the database,
- SQLERRMC, an array of 70 characters which contains the text of the message.

Appendix D
Sample Program

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  VTXTEST.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

    EXEC SQL BEGIN DECLARE SECTION END-EXEC.

01  DB-LOGIN          PIC X(80)            VALUE SPACES.
01  EMPD-REC-VARS.
    05  H-ID          PIC 9(3)     COMP-5 VALUE ZERO.
    05  H-NAME        PIC X(15)            VALUE SPACES.
    05  H-SALARY      PIC S9(9)V99 COMP-3 VALUE ZERO.
    05  H-DEPT        PIC 9(2)     COMP-5 VALUE ZERO.

    EXEC SQL END DECLARE SECTION END-EXEC.

    EXEC SQL INCLUDE SQLCA END-EXEC.

01  DISPLAY-VARIABLES.
    05  D-ID          PIC Z(2)9.
    05  D-NAME        PIC X(10).
    05  D-SALARY      PIC Z(4)9.99.
    05  D-DEPT        PIC 9(2).

01  COUNTER          PIC S9(9)    COMP-5 VALUE 0.
01  EMPD.
    05 ID-01          PIC 9(3)     COMP-5 VALUE 101.
    05 NAME-01        PIC X(15)            VALUE "Smith".
    05 SALARY-01      PIC S9(9)V99 COMP-3 VALUE 25000.00.
    05 DEPT-01        PIC S9(2)    COMP-5 VALUE 10.
    05 ID-02          PIC 9(3)     COMP-5 VALUE 102.
    05 NAME-02        PIC X(15)            VALUE "Robert".
    05 SALARY-02      PIC S9(9)V99 COMP-3 VALUE 25600.00.
    05 DEPT-02        PIC 9(2)     COMP-5 VALUE 10.
    05 ID-03          PIC 9(3)     COMP-5 VALUE 103.
    05 NAME-03        PIC X(15)            VALUE "Henry".
    05 SALARY-03      PIC S9(9)V99 COMP-3 VALUE 23400.00.
    05 DEPT-03        PIC 9(2)     COMP-5 VALUE 10.
    05 ID-04          PIC 9(3)     COMP-5 VALUE 104.
    05 NAME-04        PIC X(15)            VALUE "Johnson".
    05 SALARY-04      PIC S9(9)V99 COMP-3 VALUE 34000.00.
    05 DEPT-04        PIC 9(2)     COMP-5 VALUE 20.
    05 ID-05          PIC 9(3)     COMP-5 VALUE 105.
```

```
05 NAME-05       PIC X(15)           VALUE "George".
05 SALARY-0      PIC S9(9)V99 COMP-3 VALUE 33000.00.
05 DEPT-05       PIC 9(2)     COMP-5 VALUE 20.
05 ID-06         PIC 9(3)     COMP-5 VALUE 106.
05 NAME-06       PIC X(15)           VALUE "Wang".
05 SALARY-06     PIC S9(9)V99 COMP-3 VALUE 33000.00.
05 DEPT-06       PIC 9(2)     COMP-5 VALUE 20.
05 ID-07         PIC 9(3)     COMP-5 VALUE 107.
05 NAME-07       PIC X(15)           VALUE "Sandy".
05 SALARY-07     PIC S9(9)V99 COMP-3 VALUE 30000.00.
05 DEPT-07       PIC 9(2)     COMP-5 VALUE 30.
05 ID-08         PIC 9(3)     COMP-5 VALUE 108.
05 NAME-08       PIC X(15)           VALUE "Mary".
05 SALARY-08     PIC S9(9)V99 COMP-3 VALUE 32000.00.
05 DEPT-08       PIC 9(2)     COMP-5 VALUE 30.
05 ID-09         PIC 9(3)     COMP-5 VALUE 109.
05 NAME-09       PIC X(15)           VALUE "Linda".
05 SALARY-0      PIC S9(9)V99 COMP-3 VALUE 30000.00.
05 DEPT-09       PIC 9(2)     COMP-5 VALUE 30.
05 ID-10         PIC 9(3)     COMP-5 VALUE 110.
05 NAME-10       PIC X(15)           VALUE "Alex".
05 SALARY-10     PIC S9(9)V99 COMP-3 VALUE 30000.00.
05 DEPT-10       PIC 9(2)     COMP-5 VALUE 30.
05 ID-11         PIC 9(3)     COMP-5 VALUE 111.
05 NAME-11       PIC X(15)           VALUE "Allen".
05 SALARY-11     PIC S9(9)V99 COMP-3 VALUE 29000.00.
05 DEPT-11       PIC 9(2)     COMP-5 VALUE 30.
05 ID-12         PIC 9(3)     COMP-5 VALUE 112.
05 NAME-12       PIC X(15)           VALUE "Bob".
05 SALARY-12     PIC S9(9)V99 COMP-3 VALUE 38000.00.
05 DEPT-12       PIC 9(2)     COMP-5 VALUE 40.
05 ID-13         PIC 9(3)     COMP-5 VALUE 113.
05 NAME-13       PIC X(15)           VALUE "Micheal".
05 SALARY-13     PIC S9(9)V99 COMP-3 VALUE 32000.00.
05 DEPT-13       PIC 9(2)     COMP-5 VALUE 40.
05 ID-14         PIC 9(3)     COMP-5 VALUE 114.
05 NAME-14       PIC X(15)           VALUE "April".
05 SALARY-14     PIC S9(9)V99 COMP-3 VALUE 30000.00.
05 DEPT-14       PIC 9(2)     COMP-5 VALUE 50.
05 ID-15         PIC 9(3)     COMP-5 VALUE 115.
05 NAME-15       PIC X(15)           VALUE "Steve".
05 SALARY-15     PIC S9(9)V99 COMP-3 VALUE 29000.00.
05 DEPT-15       PIC 9(2)     COMP-5 VALUE 50.
05 ID-16         PIC 9(3)     COMP-5 VALUE 116.
05 NAME-16       PIC X(15)           VALUE "Andrew".
05 SALARY-16     PIC S9(9)V99 COMP-3 VALUE 30000.00.
05 DEPT-16       PIC 9(2)     COMP-5 VALUE 50.
05 ID-17         PIC 9(3)     COMP-5 VALUE 117.
05 NAME-17       PIC X(15)           VALUE "Dan".
05 SALARY-17     PIC S9(9)V99 COMP-3 VALUE 31000.00.
05 DEPT-17       PIC 9(2)     COMP-5 VALUE 50.
05 ID-18         PIC 9(3)     COMP-5 VALUE 118.
```

```
    05 NAME-18       PIC X(15)           VALUE "Tolosky".
    05 SALARY-18     PIC S9(9)V99 COMP-3 VALUE 25000.00.
    05 DEPT-18       PIC 9(2)     COMP-5 VALUE 50.
    05 ID-19         PIC 9(3)     COMP-5 VALUE 119.
    05 NAME-19       PIC X(15)           VALUE "Rotomomo".
    05 SALARY-19     PIC S9(9)V99 COMP-3 VALUE 32000.00.
    05 DEPT-19       PIC 9(2)     COMP-5 VALUE 50.
    05 ID-20         PIC 9(3)     COMP-5 VALUE 120.
    05 NAME-20       PIC X(15)           VALUE "Albert".
    05 SALARY-20     PIC S9(9)V99 COMP-3 VALUE 30500.00.
    05 DEPT-20       PIC 9(2)     COMP-5 VALUE 50.
01  EMPD-DATA REDEFINES EMPD.
    03  EMPD-DATA-ARR OCCURS 20 TIMES.
        05  ID-A     PIC 9(3)      COMP-5.
        05  NAME-A   PIC X(15).
        05  SALARY-A PIC S9(9)V99  COMP-3.
        05  DEPT-A   PIC 9(2)      COMP-5.

01  ABC-123-DEF-AREA PIC X(10).
01  GHI-456-JKL-AREA PIC X(10).

PROCEDURE DIVISION.

BEGIN-PGM.

    EXEC SQL WHENEVER SQLERROR
        GOTO SQL-ERROR
    END-EXEC.
    MOVE "NET:scott/tiger" TO DB-LOGIN.
    PERFORM LOGON.
    PERFORM CREATE-TABLE.
    PERFORM LOAD-TABLE.
    PERFORM SHOW-TABLE.
    PERFORM UPDATE-TABLE.
    PERFORM SHOW-TABLE.
    PERFORM SIGN-OFF.

CREATE-TABLE.

    EXEC SQL CREATE TABLE EMPD (ID SMALLINT NOT NULL,NA CHAR(15),
                                SALARY DECIMAL(7,2),
                                DEPT SMALLINT) END-EXEC.
    EXEC SQL COMMIT WORK END-EXEC.
    DISPLAY " "
    DISPLAY "TABLE 'EMPD' HAS BEEN CREATED IN THE DATABASE".

LOAD-TABLE.

    PERFORM INSERT-ROW
        VARYING COUNTER FROM 1 BY 1 UNTIL COUNTER > 20.
    EXEC SQL COMMIT WORK END-EXEC.
    DISPLAY " "
```

```
    DISPLAY "DATA HAS BEEN LOADED INTO TABLE 'EMPD'".

INSERT-ROW.

    MOVE ID-A(COUNTER) TO H-ID.
    MOVE NAME-A(COUNTER) TO H-NAME.
    MOVE SALARY-A(COUNTER) TO H-SALARY.
    MOVE DEPT-A(COUNTER) TO H-DEPT.
    EXEC SQL INSERT
             INTO EMPD (ID, NA, SALARY, DEPT)
             VALUES (:H-ID,  :H-NAME, :H-SALARY, :H-DEPT)
    END-EXEC.

SHOW-TABLE.

    EXEC SQL WHENEVER NOT FOUND
        GOTO SIGN-OFF
    END-EXEC.
    EXEC SQL DECLARE EMPDCSR CURSOR FOR
             SELECT ID,NA,SALARY,DEPT
             FROM EMPD END-EXEC.
    EXEC SQL OPEN EMPDCSR END-EXEC.
    DISPLAY " ".
    DISPLAY "OPEN CURSOR AND FETCH DATA FROM TABLE 'EMPD'".
    DISPLAY " ".
    DISPLAY "ID   NAME        SALARY  DEPT".
    DISPLAY "============================".
    PERFORM GET-ROW
        VARYING COUNTER FROM 1 BY 1 UNTIL COUNTER > 20.
    EXEC SQL CLOSE EMPDCSR END-EXEC.

GET-ROW.

    MOVE SPACES TO H-NAME
    EXEC SQL FETCH EMPDCSR INTO :H-ID, :H-NAME,
                                :H-SALARY, :H-DEPT
    END-EXEC.
    MOVE H-ID TO D-ID
    MOVE H-NAME TO D-NAME
    MOVE H-SALARY TO D-SALARY
    MOVE H-DEPT TO D-DEPT
    DISPLAY D-ID, "  ", D-NAME, "  ", D-SALARY, "  ", D-DEPT.

UPDATE-TABLE.

    DISPLAY " ".
    DISPLAY "*** Modify the salary on [120] Albert from",
            " $30500 to $35555".
    MOVE 35555.00 TO H-SALARY.
    MOVE 120 TO H-ID.
    EXEC SQL UPDATE EMPD SET SALARY = :H-SALARY
             WHERE ID = :H-ID END-EXEC.
```

```
    EXEC SQL COMMIT WORK END-EXEC.
    EXEC SQL DECLARE UPDCSR CURSOR FOR
             SELECT ID,NA,SALARY,DEPT
             FROM EMPD WHERE ID = :H-ID END-EXEC.
    EXEC SQL OPEN UPDCSR END-EXEC.
    MOVE SPACES TO H-NAME
    EXEC SQL FETCH UPDCSR INTO :H-ID, :H-NAME,
                               :H-SALARY, :H-DEPT END-EXEC.
    DISPLAY " ".
    DISPLAY "OPEN CURSOR AND FETCH DATA FROM TABLE 'EMPD'".
    DISPLAY " ".
    DISPLAY "ID   NAME        SALARY  DEPT".
    DISPLAY "============================".
    MOVE H-ID TO D-ID
    MOVE H-NAME TO D-NAME
    MOVE H-SALARY TO D-SALARY
    MOVE H-DEPT TO D-DEPT
    DISPLAY D-ID, "  ", D-NAME, "  ", D-SALARY, "  ", D-DEPT.
    EXEC SQL CLOSE UPDCSR END-EXEC.
    DISPLAY " ".
    DISPLAY "*** Delete the record of [119] Rotomomo".
    MOVE 119 TO H-ID.
    EXEC SQL DELETE FROM EMPD WHERE ID = :H-ID END-EXEC.
    EXEC SQL COMMIT WORK END-EXEC.
    EXEC SQL CLOSE UPDCSR END-EXEC.

LOGON.

    EXEC SQL
        CONNECT TO :DB-LOGIN
    END-EXEC.
    DISPLAY " ".
    DISPLAY "             <<<   VTXTEST   >>>".
    DISPLAY " ".
    DISPLAY "CONNECTED TO DATABASE AS: ", DB-LOGIN.

SIGN-OFF.

    EXEC SQL DROP TABLE EMPD END-EXEC.
    EXEC SQL COMMIT WORK END-EXEC.
    DISPLAY " ".
    DISPLAY "TABLE 'EMPD' HAS BEEN DROP FROM THE DATABASE".
    DISPLAY " ".
    DISPLAY "HAVE A GOOD DAY.".
    DISPLAY " ".
    EXEC SQL COMMIT WORK RELEASE END-EXEC.
    STOP RUN.

SQL-ERROR.

    EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
    DISPLAY " ".
```

```
DISPLAY "DATABASE ERROR DETECTED:".
DISPLAY " ".
DISPLAY SQLERRMC.
EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
STOP RUN.
```

Appendix E
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 `dty`.

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

Syntax

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

Parameters

dty 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-bye 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-bye 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

nump Internal numeric buffer

num_len Length of nump buffer.

maskp Pointer to format mask.

jus Justification (“L,” “C,” or “R”)

fsp 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

nump Internal numeric buffer

num_len Length of nump buffer.

status Conversion status.
- -1 — truncation
- 0 — ok
- 1 — overflow

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

nump Internal numeric value (returned).

pdp Packed decimal value.

pas 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,
  char           *maskp,
  int             mask_len,
  char           *bufp,
  int             buf_len
  );
```

Parameters

dt Internal date-time value (returned)

maskp Character string mask.

mask_len Length of maskp buffer.

bufp Character string.

buf_len 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 sdty;
  unsigned int slen;
  unsigned char *sp;
  int tdty;
  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).

Appendix F
Error Messages

VORTEXcobol Messages

Message	Remedy
Input and output files have the same name.	Change the output filename.
Illegal EXEC SQL statement.	Correct the invalid embedded SQL statement.
SQLERROR, SQLWARNING, or NOT FOUND expected.	Correct the WHENEVER condition.
FOUND expected following NOT.	Correct the statement.
CONTINUE or GOTO expected.	Correct the WHENEVER condition.
Label expected.	Insert a label following GOTO.
Keyword *keyword* expected.	Insert the missing keyword.
Identifier expected.	Insert the missing identifier.
Illegal type declaration.	Correct the host variable declaration.
Unknown type or modifier.	Correct the host variable declaration.
Cursor name expected.	Insert the missing cursor name.
Host variable expected.	Insert the missing host variable.
Comma (,) expected.	Insert the missing comma.
Keyword *keyword* or *keyword* expected.	Insert the missing keyword.
Invalid FOR clause.	Correct the syntax.
Illegal item in INTO list.	Ensure that the item has been declared.
DESCRIBE INTO SQLDA not supported in COBOL	Remove the DESCRIBE.
Missing *item*.	Insert the missing *item*.
Symbol already defined.	Change the cursor or host variable name.

Message	Remedy
Symbol not found.	Define the cursor or host variable.
Host variable must be char.	Change the variable declaration to a char.
Indicator variable must be 2 byte (short) integer.	Correct the datatype.
Symbol table overflow (while processing *'item'*)	Contact Support.
Not yet implemented yet.	Contact Support.
INTERNAL ERROR: *error*	Contact Support.
FILE ERROR: *errof*	Contact Support.
Too many dimensions in array.	Correct the array.
Arrays not same dimensions.	Correct so all host variable arrays are the same dimension.
FOR must be integer.	Correct the FOR type.
Invalid PIC[TURE] definition	Correct the definition.

T

U

V