VORTEXc Precompiler Reference Manual

May 30, 2017

Trifox Inc.

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

Audience

This reference manual is for programmers who use embedded SQL in existing or new C 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 VORTEXc. 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

9 November 1999

Updated the sample application in Appendix D.

18 January 2000

Updated instructions for building your application (page 9).

21 January 2010

Added timestamp datatatype.

4 November 2011

Updated timestamp host variable definition.

Chapter 1 Precompilers

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

VORTEXc performs a variety of services including:

- Converting SQL statements into VORTEXcli function calls.
- Converting SQL datatypes to C 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. VORTEXc processes embedded SQL commands that enhance C 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 C source and add SQL statements as necessary.
2. Make sure your environment is set up according to the instructions in Chapter 2.
3. Run VORTEXc.
4. Run the new source through your C 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 C compiler.
- The support files and linking libraries required by your compiler.
- VORTEXc, 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` and `SQLDA` structures to return errors and query result information. See the `sqlca.h` file in "***Include Files***" on page 20

Chapter 2
Getting Started

Setting Up

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

vtxc.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)
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 ---------- VORTEXc generics
columns            256          -- max # of database columns
db_cursors          64          -- max # of DB cursors
dflt_rw_transaction no             -- initial transaction is read write
fetch_buffer_size   8192           -- fetch buffer size (in bytes)
hard_close_cursors  no             -- soft close by default
logical_cursors     512          -- max # of logical cursors
sql_ext             .pc          -- file extension if not specified
uppercase_sql       yes            -- uppercase SQL
```

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 61.

Include File

VORTEXc, like your compiler, pulls in any included files at processing time. C source code needs to have the SQLCA file. For a full listing of the file, see "***Include Files***" on page 20.

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.pc`, uses the `EMP` 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 `EMP`, 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 VORTEXc precompiler. Simply type:

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

where

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

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

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

options Processing options:

-c	Use lower-case SQLCA and SQLDA.
-cpp	Wrap 'extern "C" {...}' around inserted include files.
-copy	Automatically pull in all copy libs.
-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.
-implicit	Do not require DECLARE SECTIONs.
-include	Process `#include` directives.
-l	Insert `#line` information.
-s	Dump the symbol table.
-ss	Allow // style comments (slash slash).
-t	Print tracing information to the screen.
-tp	Use token pasting for long SQL strings
-ts	Generate thread safe code.

For example:

```
vtxc vtxtest.pc vtxtest
```

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

Building Your Application

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

```
ccvtxc 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  >>>
            OPEN CURSOR AND FETCH DATA FROM emp
    ================================================

     [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

[120]             Albert   35555.00   50


 *** Delete the record of [119] Rotomomo

[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
```

Chapter 3
Converting Your Files

You may not have to make any changes to your original C source code for it to precompile successfully with VORTEXc. 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 C application, you use an industry standard protocol called ***embedded SQL***. You begin each statement with the phrase "`EXEC SQL`" and end it with "a semicolon (;)." Your SQL statements can span multiple lines, but the `EXEC SQL` 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;
EXEC SQL declare C0 cursor for select *
                                        from staff
                                        where id > 50;
```

Datatype Mappings

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

Character Data

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

For example, a `CHAR(10)` database column matches a char str[10]; variable. VORTEXc 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 C variable, (for example, `char str[11];`) VORTEXc can null-terminate the string.

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

Integer Data

VORTEXc 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.

In comparison, C has two floating point datatypes: `float` and `double`. VORTEXc converts between the database value and the C 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.

VORTEXc accommodates the variability in date-time with several options. The most simple to implement option has VORTEXc 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.

VORTEXc can also allow an application to request that date-time information be returned into a C (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;
datetime  dt;
timestamp ts;
EXEC SQL END DECLARE SECTION;
```

After VORTEXc processes the source file, C variables are declared as follows:

```
char dt[7];
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;
char    str[22];
short   ind;
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT SPOUSENAME INTO :str:ind FROM STAFF WHERE ID = 42;
if (ind < 0)
printf(``No spouse name on record\n'');
else if (ind > 0)
printf(``Spouse name truncated (database length %d)\n'',ind);
else …
```

Error Handling

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

```
EXEC SQL WHENEVER condition action;
```

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

```
EXEC SQL WHENEVER NOT FOUND GOTO no_data;
```

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
```

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.*
- *macro* — Execute a macro.

macro is a C macro. It can be as simple as a single function call or more complex.

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

VORTEXc 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

VORTEXc 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;
typedef struct {
      int    id;
      char   name[11];
      int    dept;
      char   job[6];
      int    years;
      float  salary;
      float  comm;
      } STAFF_REC;
STAFF_REC staff_rec;
short staff_ind[7];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT ID,NAME,DEPT,JOB,YEARS,SALARY,COMM
      INTO :staff_rec:staff_ind
      FROM STAFF WHERE ID = 42;
```

This code populates the`staff_rec` record with the correct values, as well as the indicator variables, `staff_find`.

Array Example

The following example populates the`staff_rec` 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;
typedef struct {
      int    id;
      char   name[11];
      int    dept;
      char   job[6];
      int    years;
      float  salary;
      float  comm;
      } STAFF_REC;
STAFF_REC staff_rec[100];
short staff_ind[700];
int   min_id;
EXEC SQL END DECLARE SECTION;

min_id = 42;
EXEC SQL DECLARE C0 CURSOR FOR
      SELECT ID,NAME,DEPT,JOB,YEARS,SALARY,COMM
      FROM STAFF WHERE ID >= :min_id;
EXEC SQL OPEN C0;
EXEC SQL WHENEVER NOT FOUND GOTO done;
while (1) {
  EXEC SQL FETCH C0 INTO :staff_rec:staff_ind;
```

```
if (SQLCA.SQLERRD[2])
   printf(``%d record%s returned\n'',
          SQLCA.SQLERRD[2],(SQLCA.SQLERRD[2] == 1 ? ``'' : ``s''));
else printf(``No records returned\n'');
}
done:;
```

The application could also be written as follows:

```
EXEC SQL BEGIN DECLARE SECTION;
       int    id[100];
       char   name[100][11];
       int    dept[100];
       char   job[100][6];
       int    years[100];
       float  salary[100];
       float  comm[100];

       short  id_ind[100];
       short  name_ind[100];
       short  dept_ind[100];
       short  job_ind[100];
       short  years_ind[100];
       short  salary_ind[100];
       short  comm_ind[100];

       int   min_id;
EXEC SQL END DECLARE SECTION;

   min_id = 42;
EXEC SQL DECLARE C0 CURSOR FOR
       SELECT ID,NAME,DEPT,JOB,YEARS,SALARY,COMM
       FROM STAFF WHERE ID >= :min_id;
EXEC SQL OPEN C0;
EXEC SQL WHENEVER NOT FOUND GOTO done;
   while (1) {
       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;

if (SQLCA.SQLERRD[2])
   printf(``%d record%s returned\n'',
       SQLCA.SQLERRD[2],(SQLCA.SQLERRD[2] == 1 ? ``'' : ``s''));
else printf(``No records returned\n'');
}
   done:;
```

VORTEXc 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;staff_ind);
```

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;staff_ind);
```

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. VORTEXc 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 VORTEXc 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***

  VORTEXc 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 VORTEXc.

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

- ***Incompatible SQL Language Extensions***

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

  However, if VORTEXc 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.

  VORTEXc supports all ANSI-compliant embedded SQL. In addition, VORTEXc 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 `semi-colon`, the actual C code used to manipulate data defined by these statements may have to vary depending on vendor.

  When generating these datatypes, VORTEXc 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 C 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.

Threaded Applications

VORTEXc can create thread-safe application code. There are three modifications that you must make to your source file before processing it with VORTEXc using the -ts option. You must declare a pointer to aVORTEX connection handle and then specify that handle name when you initiate the database connection. For example

```
VTX_CB *my_hdl;
exec sql AT my_hdl CONNECT TO :cst;
```

You can continue to use “EXEC SQL AT <name>” in subsequent statements but it is not necessary; the last sequentially processed “EXEC SQL AT <name>” remains the active connection. You must also declare a pointer to a SQLCA structure. For example,

```
SQLCA_DEF sqlca,*SQLCAP;
SQLCAP = &sqlca;
memcpy(SQLCAP,&SQLCA_INIT,sizeof(SQLCA_INIT));
```

Clearly both the VTX_CB and SQLCAP variables must be valid in the scope of any EXEC SQL statements.

Appendix A
Include Files

SQLCA

```
/
*****************************************************************************
************
*                              Copyright (C) TRIFOX, Inc. 1989
*  Module Name = SQLCA
*  Description = SQL communication area.
*  Author      = LNB
*  Date        = 03/02/89
*
*****************************************************************************/
#ifndef chad
#ifdef USING_UNICODE
#define chad unsigned short
#else
#define chad char
#endif
#endif

#ifdef MVS
#define SQLCA_INIT {{'S','Q','L','C','A',' ',' ',' '},134,0,0,{0},{0},{0},{0},{0}}
#else
#define SQLCA_INIT {{'S','Q','L','C','A',' ',' ',' '},128,0,0,{0},{0},{0},{0}}
#endif

#ifdef internal_sqlca
typedef struct sqlca {
        char        sqlcaid[8];     /* SQLCA identifier          */
        int         sqlcabc;        /* SQLCA length              */
        int         sqlcode;        /* return code               */
```

```
        unsigned short  sqlerrl;        /* error message length         */
        chad          sqlerrm[70];     /* error message                */
#ifdef MVS
        char          sqlerrp[8];
#endif
        int           sqlerrd[6];      /* error descriptors            */
        char          sqlwarn[8];      /* warning descriptors          */
        char          sqlext[8];       /* reserved for future use      */
        } SQLCA;
#else
static struct {
  char SQLCAID[8];                     /* used for DB2 compatability    */
  int  SQLCABC;                        /* size of SQLCA structure       */
  int  SQLCODE;                        /* status of last sql statement  */
  struct {
    short SQLERRML;                    /* length of error message       */
    chad  SQLERRMC[70];                /* the actual error message      */
    } SQLERRM;
  int  SQLERRD[6];                     /* uses 2,3,4,5 only             */
  struct {
    char  SQLWARN0[1];
    char  SQLWARN1[1];
    char  SQLWARN2[1];
    char  SQLWARN3[1];
    char  SQLWARN4[1];
    char  SQLWARN5[1];
    char  SQLWARN6[1];
    char  SQLWARN7[1];
    } SQLWARN;                         /* for DB2 compatability only    */
  char SQLEXT[8];
  } SQLCA = SQLCA_INIT;
#endif
SQLDA
/*****************************************************************************/
/* SQLDA structure                                                           */
/*****************************************************************************/
```

```
#ifndef chad
#ifdef USING_UNICODE
#define chad unsigned short
#else
#define chad char
#endif
#endif

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

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

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

vortex.h

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

#ifdef __370__
#define SIGNED signed
#endif
#ifdef USING_UNICODE
#define   chad      unsigned short          /* char (defined)                 */
#define   chud      unsigned short          /* unsigned char (defined)        */
#else
```

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

#ifdef __alpha
#ifdef VMS
#define  i8  long long                      /* 8 byte signed                 */
#define  u8  unsigned long long             /* 8 byte unsigned               */
#else
#define  i8  long                           /* 8 byte signed                 */
#define  u8  unsigned long                  /* 8 byte unsigned               */
#endif
#else
#ifdef SYSTEM5
#define  i8  long long                      /* 8 byte signed                 */
#define  u8  unsigned long long             /* 8 byte unsigned               */
#endif
#ifdef _MSWIN
#define  i8  __int64                        /* 8 byte signed                 */
#define  u8  unsigned __int64               /* 8 byte unsigned               */
#endif
#endif

#define  ident_len   30                     /* max identifier length         */

typedef  char  ident[ident_len];

/*****************************************************************************/
/* Datatypes                                                                 */
/*****************************************************************************/
#ifndef  TDT_INTEGER
#define  TDT_INTEGER         0              /* integer                       */
#define  TDT_CHAR            1              /* character                     */
#define  TDT_NUMBER          2              /* internal numeric              */
#define  TDT_NTCHAR          3              /* null terminated string        */
#define  TDT_PACKED          4              /* packed decimal                */
#define  TDT_ZONED           5              /* zoned decimal                 */
#define  TDT_FLOAT           8              /* floating point number         */
#define  TDT_VARCHAR         9              /* var char                      */
#define  TDT_BLOB           10              /* blob (binary large object)    */
```

```
#define  TDT_CLOB           11              /* clob (character large object)  */
#define  TDT_DATETIME       12              /* date time                      */
#define  TDT_VARBLOB        80              /* varlen blob                    */
#define  TDT_UTF8           81              /* Unicode char UTF-8  format     */
#define  TDT_UTF16          82              /* Unicode char UTF-16 format     */
#define  TDT_UCS2           83              /* Unicode char UCS-2  format     */
#define  TDT_UCS4           84              /* Unicode char UCS-4  format     */
#define  TDT_UTF8LOB        85              /* Unicode LOB  UTF-8  format     */
#define  TDT_VARUTF8        86              /* Unicode varchar UTF-8  format  */
#define  TDT_VARUTF16       87              /* Unicode varchar UTF-16 format  */
#define  TDT_VARUCS2        88              /* Unicode varchar UCS-2  format  */
#define  TDT_VARUCS4        89              /* Unicode varchar UCS-4  format  */
#define  TDT_BINARY         99              /* binary                         */
#endif
#endif

/*****************************************************************************/
/* DB engine IDs                                                             */
/*****************************************************************************/
#define  TDB_DID_ORACLE      0              /* Oracle                         */
#define  TDB_DID_RDB         1              /* DEC's Rdb                      */
#define  TDB_DID_SYBASE      2              /* Sybase                         */
#define  TDB_DID_NET         3              /* Network                        */
#define  TDB_DID_GENESIS     4              /* Genesis                        */
#define  TDB_DID_INFORMIX    5              /* Informix's RDBMS               */
#define  TDB_DID_ALLBASE     6              /* HP's Allbase RDBMS             */
#define  TDB_DID_DB2         7              /* IBM's DB2                      */
#define  TDB_DID_OLEDB       8              /* OLE DB                         */
#define  TDB_DID_ADABASC     9              /* Software AG's ADABAS           */
#define  TDB_DID_ADABASD    10              /* Software AG's SQLDB            */
#define  TDB_DID_ODBC       11              /* ODBC                           */
#define  TDB_DID_SQLSERVER  12              /* Microsoft's SQL Server         */
#define  TDB_DID_TERADATA   13              /* AT&T's Teradata                */
#define  TDB_DID_MYSQL      14              /* MySQL                          */
#define  TDB_DID_POSTGRESQL 16              /* PostgreSQL                     */
#define  TDB_DID_TRIM       17              /* TRIMpl                         */

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

/*****************************************************************************/
/* DB driver commands                                                        */
/*****************************************************************************/
#define  TDB_CMD_TIMEOUT        0           /* resource timeout               */
#define  TDB_CMD_SINGLEPID      1           /* single transaction pid (Sybase)*/
#define  TDB_CMD_OOPT           2           /* OCI oopt call (Oracle7)        */
#define  TDB_CMD_USEDB          3           /* USE DATABASE (Sybase)          */
#define  TDB_CMD_LANGUAGE       4           /* language flag (Oracle)         */
#define  TDB_CMD_RAW_DATETIME   5           /* Datetime is returned untouched */
#define  TDB_CMD_VERSION        6           /* place version # in message buf */
#define  TDB_CMD_CHAR           7           /* char datatype value (Oracle)   */
#define  TDB_CMD_LOCK           8           /* lock SLAVE on next stmt (MUX)  */
#define  TDB_CMD_ONEBYTE        9           /* blank varchar handling (Sybase)*/
#define  TDB_CMD_MULTISTMT     10           /* batch statements (Sybase)      */
#define  TDB_CMD_AUTOCOMMIT    11           /* autocommit on/off(Genesis,ODBC)*/
#define  TDB_CMD_DORMANT       12           /* connection dormant (Informix)  */
#define  TDB_CMD_GETLDA        13           /* return LDA pointer (Oracle7)   */
```

```
#define  TDB_CMD_RAISE         14          /* do a raise() (driver 99 only)  */
#define  TDB_CMD_ERROR         15          /* return error (driver 99 only)  */
#define  TDB_CMD_TMP_RELEASE   16          /* temporary release (vtxwebd)    */
#define  TDB_CMD_WORM_AUTH     17          /* Genesis/WORM authentication    */
#define  TDB_CMD_TIMEOUT_QUERY 18          /* query timeout (Genesis/SDMS)   */
#define  TDB_CMD_TIMEOUT_FETCH 19          /* fetch timeout (Genesis/SDMS)   */
#define  TDB_CMD_DYN_CURSOR    20          /* use dynamic cursors (ODBC)     */
#define  TDB_CMD_SYNTAX_LEVEL  21          /* GENESIS syntax level (ODBC)    */
#define  TDB_CMD_RETURN_ROWID  22          /* Return ROWID on insert         */
#define  TDB_CMD_NEW_BLOBS     23          /* use new Oracle blobs           */
#define  TDB_CMD_RO_CURSOR     24          /* set cursors readonly (various) */
#define  TDB_CMD_CURSOR_TYPE   25          /* cursor type (ODBC)             */
#define  TDB_CMD_TXN_ISOLEVEL  26          /* xaction isolation level (ODBC) */
#define  TDB_CMD_SCROLL        27          /* set cursor scroll option       */
#define  TDB_CMD_HOSTNAME      28          /* get host name                  */
#define  TDB_CMD_FQCOLNAME     29          /* fully qualified column names   */
#define  TDB_CMD_FREE_LOCKS    30          /* free/release locks (ODBC)      */
#define  TDB_CMD_NOUNICHAR     31          /* desc W[VAR]CHAR as CHAR (ODBC) */
#define  TDB_CMD_ZEROLEN_NULL  32          /* make 0 len strings NULL (SS)   */
#define  TDB_CMD_NOTUSED     0xFF          /* used to check connection       */

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

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

/*****************************************************************************/
/* Various VORTEXcli (tdbprep) definitions                                   */
/*****************************************************************************/
#define  TDB_PRE_what "@(T)PCAPI 3.3.2.55"  /* pre-compiler version          */
#define  TDB_PRE_VERSION       "3.3.2.55"  /* pre-compiler version          */
#define  TDB_PRE_W32_VER        3,3,2,55   /* pre-compiler version          */
#define  TDB_PRE_VER_NUM        332        /* pre-compiler version #        */

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

#define  TDB_PRE_POPEN       0x0001        /* is a positioned OPEN           */
#define  TDB_PRE_PEXEC       TDB_PRE_POPEN /* is a positioned EXEC           */
#define  TDB_PRE_FUO         0x0002        /* statement has "FOR UPDATE OF"  */
#define  TDB_PRE_BLOB        0x0004        /* statement involves BLOBs       */
#define  TDB_PRE_SINGSEL     0x0008        /* singleton SELECT (allows 1 row)*/
```

```
#define  TDB_PRE_SCROLL      0x0010         /* scrollable cursor              */
#define  TDB_PRE_STATIC      0x0020         /* static scrollable cursor       */
#define  TDB_PRE_DYNAMIC     0x0040         /* dynamic scrollable cursor      */
#define  TDB_PRE_INSENSITIVE 0x0080         /* insensitive scrollable cursor  */
#define  TDB_PRE_ASENSITIVE  0x0100         /* asensitive scrollable cursor   */

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

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

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

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

/*************************************************************************/
/* Extended host variable descriptor. (used by VORTEXcli)                */
/* NOTE! The first 4 fields match TDB_HVAR                               */
/*************************************************************************/
#define  TDB_FLG_IN   0x01                  /* input variable                */
#define  TDB_FLG_OUT  0x02                  /* output variable               */

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

/*************************************************************************/
/* VTX behaviour options. Use VTXOPTS() to get/set these options.        */
/*                                                                       */
/* Field descriptions:                                                   */
/*                                                                       */
/* TDB_DTB - Datetime base.                                              */
/*           This number is SUBTRACTED from a datetime value when it's   */
/*           fetched into a long output variable. By default (i.e., when */
/*           TDB_DTB is 0), a datetime value fetched into a long output  */
/*           variable will be the number of days since the beginning of AD 1. */
/*           To change the base of counting to a different date, set TDB_DTB  */
/*           to the number of days between the selected date and the start of */
/*           AD 1. For example, to use the Informix datetime base, set TDB_DTB*/
```

```
/*           to 693959. Note that a positive number moves the date base       */
/*           forward to a later (AD) year, and a negative number moves it back*/
/*           to a BC year.                                                    */
/*           Default: 0                                                       */
/*                                                                            */
/* TDB_DTF - Datetime format mask. (Note! must be null terminated)            */
/*           Specifies the format of the datetime value when fetched into a   */
/*           character buffer.                                                */
/*           Default: "DD-MON-RR"                                             */
/*                                                                            */
/* TDB_NMK - NULLs mask.                                                      */
/*           When a column is described and NULLs are allowed this value is   */
/*           bit ORd to the datatype field.                                   */
/*           Default: 0                                                       */
/*                                                                            */
/******************************************************************************/
typedef struct tdb_opts {
          i4         TDB_DTB;               /* datetime base                 */
          chad       TDB_DTF[64];           /* datetime format               */
          i2         TDB_NMK;               /* NULL mask                     */
         } TDB_OPTS;

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

vortex.x

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

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

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

void VTXINIT(db_hdl,void *,int,int,int,int,int);
void VTXINI2(db_hdl,void *,int,int,int,int,int,int,void *);
void VTXCONN(db_hdl,void *,void *,int,int);
```

```
void VTXREL (db_hdl,void *);
void VTXCAN (db_hdl,void *);
void VTXCOMM(db_hdl,void *,int);
void VTXROLL(db_hdl,void *,int);
void VTXEXEC(db_hdl,void *,void *,void *,int,void *,int,int,int,int);
void VTXEXIO(db_hdl,void *,void *,void *,int,void *);
void VTXOPEN(db_hdl,void *,void *,void *,int,void *,int,int);
void VTXMOVE(db_hdl,void *,void *,void *,int,int,int);
void VTXGLEN(db_hdl,void *,void *,int *);
void VTXCLOS(db_hdl,void *,void *,int);
void VTXDESC(db_hdl,void *,void *,void *);
void VTXDES2(db_hdl,void *,void *,void *);
int  VTXDES3(db_hdl,void *,void *);
void VTXCMD (db_hdl,void *,void *,int,void *);

void VTXOPTS(int,void *);

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

Appendix B
Embedded SQL

This section describes the syntax understood by VORTEXc when it is in embedded SQL mode. When VORTEXc reaches the end of a statement in embedded SQL mode, it switches back to ANSI C 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 VORTEc 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

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

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***.
- ***`#include` or `#define` statements***; however, the `#define` statement must be on a single line.
- ***Typedefs***. When you declare a typedef variable you must include its definition in a DECLARE SECTION also. You can also use types with typedefs.
- ***Function arguments.*** You can use new or old style prototypes for simple functions, but again, the function declaration must also appear in the DECLARE SECTION. You cannot use complex types, such as functions that return pointers to functions, because the return type of a function must be a valid type for a host variable.

  Old Style

```
EXEC SQL BEGIN DECLARE SECTION;
int f(a, b, c)
char *a;
long b;
int c[];
{
   char d[20];
   long e;
   EXEC SQL END DECLARE SECTION;
   /* … */
}
```

  New Style

```
EXEC SQL BEGIN DECLARE SECTION;
int f(char *a, long b, int c[])
{
   char d[20];
   long e;
```

```
        EXEC SQL END DECLARE SECTION;
    /* … */
}
```

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>;
```

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;

struct emp_t
{
   int id;
   char name[11];
   int dept;
   char job[11];
   int years;
   int salary;
   int comm;
};

struct emp_struct emp;
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT * INTO :emp FROM STAFF WHERE ID = 20;
```

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     :emp.id, :emp.name,
```

```
                                    :emp.dept, :emp.job, :emp.years,
                                    :emp.salary, :emp:comm
                                    FROM STAFF WHERE ID = 20;
```

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;

struct emp_t
{
   int id;
   char name[11];
   int dept;
   char job[11];
   int years;
   int salary;
   int comm;
};

struct emp_struct emp;
short int emp_indicator[7]; /* INDICATOR VARIABLE */

EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT * INTO :emp:emp_indicator FROM STAFF;
```

In this example

```
ELEMENT                    INDICATOR

emp_id                     emp_indicator[0];
emp_name                   emp_indicator[1];
emp_dept                   emp_indicator[2];
emp_job                    emp_indicator[3];
emp_years                  emp_indicator[4];
emp_salary                 emp_indicator[5];
emp_comm                   emp_indicator[6];
```

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

```
EXEC SQL BEGIN DECLARE SECTION;

struct emp_t
{
   int id;
   char name[11];
   int dept;
   char job[11];
   int years;
   int salary;
```

```
   int comm;
};
struct emp_struct emp_array[10]; /* ARRAY OF STRUCTURES */

EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT ID, NAME DEPT, JOB, YEARS, SALARY,COMM
         INTO :emp_array
         FROM STAFF WHERE ID = 20;
```

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;

struct emp_t
{
   int id;
   char name[11];
   int dept;
   char job[11];
   int years;
   int salary;
   int comm;
};
struct emp_struct emp_array[10]; /* ARRAY OF STRUCTURES */
short int emp_ind[70];          /* 7 STRUCTURE MEMBERS X
                                        10 ARRAY ELEMENTS */

EXEC SQL END DECLARE SECTION;
```

In this example

```
ELEMENT                    INDICATOR

emp_array[0].id            emp_ind[0]
emp_array[0].name          emp_ind[10]
   .                           .
   .                           .
emp_array[0].comm          emp_ind[60]
   .                           .
   .                           .
emp_array[1].id            emp_ind[1]
emp_array[1].name          emp_ind[11]
   .                           .
   .                           .
emp_array[1].comm          emp_ind[61]
   .                           .
```

```
    .                           .
emp_array[2].id             emp_ind[2]
emp_array[2].name           emp_ind[12]

    .                           .
    .                           .
emp_array[9].comm           emp_ind[69]
```

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 (:emp);
EXEC SQL INSERT INTO STAFF VALUES (:emp_array);
```

SQLCA

You can include the SQLCA inside a function, but then the scope of the SQLCA is limited to that function and you must reinclude it in any disjoint scope where it is needed. If you include the SQLCA in a function, don't forget to precede any executable statement in the function.

The name of the structure is SQLCA and all of its elements have uppercase names unless the -s option is used in the command line.

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.

C Variables and Declarators

A C variable can be:

- A singly-indirected pointer.
- An element of a structure or union if the structure or union was defined previously in a DECLARE SECTION of the current scope.

A C DECLARATOR can specify an initializor.

Appendix D
Sample Program

```
/
**********************************************************************
***********/
/*
*/
/*                                                    Trifox, Inc. 1996
YJW       */
/*
*/
/*   Program Name:  vtxtest.pc
*/
/*   Description:   create table emp, and insert records for table
emp       */
/*              select data from table emp
*/
/*
*/
/
**********************************************************************
***********/
#include <stdio.h>
#include <string.h>
#define   true   1
EXEC SQL BEGIN DECLARE SECTION;
     long   i;
     int    id;
     char   name[15];
     float  salary;
     char   dept[5];
     char errmsg[101];
     char db_login[50];

struct EMP {
             int    id;
             char   name[15];
             float  salary;
             char   dept[5];
           } emp[20] = {
            { 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" }};

EXEC SQL END DECLARE SECTION;
EXEC SQL INCLUDE SQLCA;

main(argc,argv)
int argc ;
char **argv ;
{

     if (argc < 2) {
       printf("\n usage: vtxtest <db login> \n");
       exit(0);
       }
     else {
       printf("\n                <<<   VTXTEST  >>>
\n");
       strcpy(db_login,argv[1]);
       }

     printf("          OPEN CURSOR AND FETCH DATA FROM emp \n");

printf("================================================\n\n");

     i = 20;

     EXEC SQL WHENEVER SQLERROR GOTO  error;
     EXEC SQL CONNECT TO :db_login;
     EXEC SQL BEGIN WORK;
     EXEC SQL WHENEVER SQLERROR continue;
     EXEC SQL DROP TABLE emp;

     EXEC SQL WHENEVER SQLERROR GOTO  error;
     EXEC SQL CREATE TABLE emp (id integer not null,name char(15),
                                salary decimal(8,2),dept char(5));

     EXEC SQL FOR :i INSERT INTO emp (id,name,salary,dept)  VALUES
(:emp);
     EXEC SQL COMMIT WORK;
```

```
     EXEC SQL DECLARE empcsr CURSOR FOR
        SELECT id,name,salary,dept
        FROM emp;

     EXEC SQL OPEN empcsr;

     while(i--) {
        EXEC SQL FETCH empcsr INTO :id, :name, :salary, :dept;
        printf(" [%d]   %15s   %8.2f   %s\n",id, name, salary,
dept);
        }

     EXEC SQL CLOSE empcsr;

     printf("\n\n  *** Modify the salary on [120] Albert from
$30500 to $35555\n\n");
     salary = 35555.0;
     id     = 120;
     EXEC SQL UPDATE emp set salary = :salary where id = :id;
     EXEC SQL COMMIT WORK;

     EXEC SQL DECLARE updcsr CURSOR FOR
        SELECT id,name,salary,dept
        FROM emp where id = :id;

     EXEC SQL OPEN updcsr;

     EXEC SQL FETCH empcsr INTO :id, :name, :salary, :dept;
     printf(" [%d]   %15s   %8.2f   %s\n",id, name, salary, dept);

     EXEC SQL CLOSE updcsr;

     printf("\n\n  *** Delete the record of [119] Rotomomo\n\n");
     id     = 119;
     EXEC SQL delete from emp where id = :id;
     EXEC SQL COMMIT WORK;

     EXEC SQL OPEN empcsr;
     while(true) {
        EXEC SQL WHENEVER NOT FOUND GOTO done;
        EXEC SQL FETCH empcsr INTO :id, :name, :salary, :dept;
        printf(" [%d]   %15s   %8.2f   %s\n",id, name, salary,
dept);
       }
done:;
     EXEC SQL CLOSE empcsr;
     EXEC SQL RELEASE WORK;
exit(0);
error:;
     {
     char buf[300];
```

```
            memcpy(buf,SQLCA.SQLERRM.SQLERRMC,i=SQLCA.SQLERRM.SQLERRML);
            buf[i] = 0;
            printf("Error: %s\n",buf);
            }
            EXEC SQL WHENEVER SQLERROR continue;
            EXEC SQL RELEASE WORK;
       exit(1);
       }

/*************************************************************************/
/*                                                                       */
/*                                          Trifox, Inc. 1996 YJW        */
/*                                                                       */
/*   Program Name:  vtxtest.pc                                           */
/*   Description:   create table emp, and insert records for table emp   */
/*                  select data from table emp                           */
/*                                                                       */
/*************************************************************************/
#include <stdio.h>
#include <string.h>
#define   true   1
EXEC SQL BEGIN DECLARE SECTION;
     long   i;
     int    id;
     char   name[15];
     float  salary;
     char   dept[5];
     char errmsg[101];
     char db_login[50];

struct EMP {
             int    id;
             char   name[15];
             float  salary;
             char   dept[5];
           } emp[20] = {
            { 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" }};
```

```
EXEC SQL END DECLARE SECTION;
EXEC SQL INCLUDE SQLCA;

main(argc,argv)
int argc ;
char **argv ;
{

     if (argc < 2) {
       printf("\n usage: vtxtest <db login> \n");
       exit(0);
       }
     else {
       printf("\n                <<<   VTXTEST  >>>             \n");
       strcpy(db_login,argv[1]);
       }

     printf("          OPEN CURSOR AND FETCH DATA FROM emp \n");
     printf("================================================\n\n");

     i = 20;

     EXEC SQL CONNECT TO :db_login;
     EXEC SQL BEGIN WORK;
     EXEC SQL DROP TABLE emp;

     EXEC SQL WHENEVER SQLERROR GOTO  error;
     EXEC SQL CREATE TABLE emp (id integer not null,name char(15),
                               salary decimal(8,2),dept char(5));

     EXEC SQL FOR :i INSERT INTO emp (id,name,salary,dept)  VALUES (:emp);
     EXEC SQL COMMIT WORK;

     EXEC SQL DECLARE empcsr CURSOR FOR
        SELECT id,name,salary,dept
        FROM emp;

     EXEC SQL OPEN empcsr;

     while(i--) {
        EXEC SQL FETCH empcsr INTO :id, :name, :salary, :dept;
        printf(" [%d]   %15s   %8.2f   %s\n",id, name, salary, dept);
        }

     EXEC SQL CLOSE empcsr;

     printf("\n\n  *** Modify the salary on [120] Albert from $30500 to
$35555\n\n");
     salary = 35555.0;
     id     = 120;
     EXEC SQL UPDATE emp set salary = :salary where id = :id;
     EXEC SQL COMMIT WORK;

     EXEC SQL DECLARE updcsr CURSOR FOR
        SELECT id,name,salary,dept
        FROM emp where id = :id;
```

```
     EXEC SQL OPEN updcsr;

     EXEC SQL FETCH empcsr INTO :id, :name, :salary, :dept;
     printf(" [%d]   %15s   %8.2f   %s\n",id, name, salary, dept);

     EXEC SQL CLOSE updcsr;

     printf("\n\n  *** Delete the record of [119] Rotomomo\n\n");
     id     = 119;
     EXEC SQL delete from emp where id = :id;
     EXEC SQL COMMIT WORK;

     EXEC SQL OPEN empcsr;
     while(true) {
        EXEC SQL WHENEVER NOT FOUND GOTO done;
        EXEC SQL FETCH empcsr INTO :id, :name, :salary, :dept;
        printf(" [%d]   %15s   %8.2f   %s\n",id, name, salary, dept);
       }
done:;
     EXEC SQL CLOSE empcsr;
     EXEC SQL RELEASE WORK;
exit(0);
error:;
     printf(" Error ...\n");
     EXEC SQL RELEASE WORK;
exit(1);
}
```

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 VORTEXc 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.

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

Parameters

dt Internal date-time value (returned)

dt_len Size of internal date-time buffer. Values are 4 - date, 7 - datetime, 10 - timestamp.

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

VORTEXc 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.
Too many right braces (})	Remove a brace.
Missing *item*.	Insert the missing *item*.
Symbol already defined.	Change the cursor or host variable name.
Symbol not found.	Define the cursor or host variable.
Host variable must be char.	Change the variable declaration to a char.

Message	Remedy
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 `struct` declaration	Correct the struct declaration.
Undefined structure.	Ensure that the structure is defined.

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

V