Tools and Connectivity Troubleshooting Guide

Please send comments about this *Troubleshooting Guide* to the email alias *tsg@sybase.com*.
We welcome your comments.

SQL Server 4.2 Through 10.0.2
Document ID: 39999-01-1000-02
Last Revised: December 19, 1994

Managing editor: Leigh Ann Hussey

Contributing authors/editors: Mike Allen, Pamela Barnett, Diana Barrett, Chris Black, Thomas Burmann, Claudia Catano, Brian Choy, David Clegg, Kathryn Dumont, Ajit Gupta, Wendy Hoben, Michael Hushour, Leigh Ann Hussey, Vijaya Konakanchi, Suresh Kuppahally, Alan McGhee, Guy Moore, Dave Peterson, John Smith, Gwen Stringer, Uma Subrumanian, Carol Talbeck, David Van Couvering, Rob Weaver, Mark White

Document ID: 39999-01-1000

This publication pertains to SQL Server 4.2 Through 10.0.2 of the SYBASE database management software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

Document Orders

To order additional documents, U.S. and Canadian customers should call Customer Fulfillment at (800) 685-8225, fax (617) 229-9845.

Customers in other countries with a U.S. license agreement may contact Customer Fulfillment via the above fax number. All other international customers should contact their Sybase subsidiary or local distributor.

Upgrades are provided only at regularly scheduled software release dates.

Sybase Trademarks

Restricted Rights

Sybase, Inc., 6475 Christie Avenue, Emeryville, CA 94608.

Table of Contents

Preface

1. About *isql*

2. Bulk Copy Utility

3. Bulk Copy Programming

4. Embedded SQL

5. Open Client DB-Library

6. Open Client Client-Library

7. Open Server Issues

8. PC Issues

9. Mainframe Access Products

10. SQL Monitor Server

11. APT Workbench, APT-Edit, and APT-Build

13. APT-Library and Mixed-Mode Programming

A. Finding Your Library Version

Index

List of Figures

List of Tables

Preface

The *Tools and Connectivity Troubleshooting Guide* describes error conditions and troubleshooting procedures for problems that developers or users may encounter in the Sybase Tools and Connectivity products. The problems addressed here are those that Technical Support staff hear about most often. The guide's purpose is:

- To provide answers to commonly reported questions so that you can resolve problems without calling Sybase Technical Support.
- To provide lists of information that you can gather before calling Technical Support, which will help speed problem resolution.
- To provide you with a greater understanding of Sybase products.

Troubleshooting information about SQL Server is provided in a separate manual, *SYBASE SQL Server Troubleshooting Guide*.

Audience

This guide is intended for the following audience:

- Application developers using SYBASE® software
- Sybase Technical Support Contacts
- Sybase System Administrators

This guide assumes that you are already familiar with Sybase products. If you are unsure about handling any of the procedures described in this guide, call Sybase Technical Support for assistance.

What This Guide Contains

This guide contains the following chapters:

- Chapter 1, "About isql."
- Chapter 2, "Bulk Copy Utility."
- Chapter 3, "Bulk Copy Programming."
- Chapter 4, "Embedded SQL."
- Chapter 5, "Open Client DB-Library."
- Chapter 6, "Open Client Client-Library."

- Chapter 7, “Open Server Issues.”
- Chapter 8, “PC Issues.”
- Chapter 9, “Mainframe Access Products.”
- Chapter 10, “SQL Monitor Server.”
- Chapter 11, “APT Workbench, APT-Edit, and APT-Build.”
- Chapter 12, “Data Workbench and Report Workbench.”
- Chapter 13, “APT-Library and Mixed-Mode Programming.”
- Appendix A, “Finding Your Library Version.”

Most chapters contain information about commonly encountered errors, answers to frequently asked questions, and tips for programming or using products more effectively.

Changes to the *Tools and Connectivity Troubleshooting Guide*

This document contains the following changes:

- Chapter 1, “About **isql**”, contains information on suppressing header information in result tables. VMS information has been updated.
- Chapter 2, “Bulk Copy Utility”, includes answers to frequently asked questions, and new and updated error message information.
- Chapter 4, “Embedded SQL”, has been expanded, including these new sections:
  - Missing Error Message Texts and Negative Return Codes
  - Embedded SQL vs. **isql** Performance with Joins
  - Migrating Embedded SQL Applications to System 10
  - Unresolved Symbols At Link Time
  - Error Numbers Beyond -25000
- Chapter 5, “Open Client DB-Library”, has new errors and frequently asked questions information.
- Chapter 6, “Open Client Client-Library”, has been added; it includes programming and troubleshooting tips.
- Chapter 7, “Open Server Issues”, includes a new code example for creating an application that audits SQL requests.

- Chapter 8, “PC Issues”, has been reorganized by operating system, and has been updated for System 10.
- Chapter 9, “Mainframe Access Products”, has been added; it contains programming tips for Open Client / Mainframe, a list of the most recent Mainframe Access Product documents, and other pertinent items.
- Chapter 10, “SQL Monitor Server,” which addresses SQL Monitor Server issues, has been reorganized into Monitor Server and Monitor Client sections. New material has been added regarding setting the event buffer and various errors.

➤ ***Note***

The term EBF (Emergency Bug Fix) is, over time, going to be changed to SWR (Software Release). This document will be updated to reflect this change when the transition between names has been completed.

Your Comments About the Troubleshooting Guide

In order to ensure the continuous improvement of the *Tools and Connectivity Troubleshooting Guide*, we need your feedback. To help make it easier for you to provide this feedback, a mail alias called *tsg* has been established. The intention of this mail alias is to allow both Sybase customers and employees (via *tsg@sybase.com*) to provide comments about the *Troubleshooting Guide*.

These comments might include:

- Corrections
- Requests for specific additions
- Additions (that is, written material)
- Comments about sections that are particularly helpful
- Comments about sections that are not clear
- Any other input you might have

The goal of this *Troubleshooting Guide* is to assist you in the use and support of Sybase products and to make you more self-sufficient in these activities. To accomplish this, **we need your feedback**.

Style Conventions

Wherever possible, the *Tools and Connectivity Troubleshooting Guide* uses the style conventions of the various product manuals. This section contains a brief summary of those conventions.

Style Conventions in Text

Commands and script names appear in bold type; for example:

> To change the **isql** command terminator...

Object names appear in italics; for example:

> Use the **installmodel** script to complete the installation of the *model* database.

SQL Syntax Conventions

The conventions for syntax statements in this manual are as follows:

Table 1: SQL syntax conventions

Key	Definition
command	Command names, command option names, utility names, utility flags, and other keywords are in bold.
variable	Variables, or words that stand for values that you fill in, are in italics.
{ }	Curly braces indicate that you choose at least one of the enclosed options. Do not include braces in your option.
[ ]	Brackets mean choosing one or more of the enclosed options is optional. Do not include brackets in your option.
( )	Parentheses are to be typed as part of the command.
|	The vertical bar means you may select only one of the options shown.
,	The comma means you may choose as many of the options shown as you like, separating your choices with commas to be typed as part of the command.

SQL syntax statements (displaying the syntax and all options for a command) are printed like this:

```
sp_dropdevice [device_name]
```

Examples showing the use of Transact-SQL® commands are printed like this:

```
1> select * from publishers
2> go
```

Examples of output from the computer are printed like this:

```
pub_id  pub_name              city        state
------  --------------------- ----------- -----
0736    New Age Books         Boston      MA
0877    Binnet & Hardley      Washington  DC
1389    Algodata Infosystems  Berkeley    CA

(3 rows affected)
```

APT Workbench and Data Workbench Style Conventions

Whenever this manual describes an action on a cascading menu, it uses this notation:

/→ Tools→ Customize→ Properties→ Character.

Table names, datatypes, and file names are *italicized* in the text.

`Example syntax is monospaced.`

Text that you type in as a value for a property, variable, or field appears on a separate line and appears in bold monospace font:

```
Type this
```

If You Need Help

Help with your SYBASE software is available in the form of SyBooks™ and AnswerBase CDs, and Sybase Technical Support. If you have any questions about the procedures contained in this guide, ask the designated person at your site to contact Sybase Technical Support.

ISQL and Bulk Copy

1 About *isql*

This chapter contains answers to some common questions about **isql**.

Changing the Command Terminator for *isql*

Question

How do you change the command terminator for **isql**? (The default command terminator is “go”.)

Answer

You do not have to type “go” every time you want to execute a query in **isql**. If you include the command terminator argument–the **-c** flag for UNIX and PC platforms–you can choose your own terminator symbol. “go” is the default value for this option.

Do not change the command terminator on OpenVMS systems.

Action

For example, to use a period as the command terminator, invoke **isql** on UNIX or PC platforms as follows:

```
isql -c.
```

A sample **isql** session with this command terminator looks like this:

```
1> select name from sysusers
2> .

name
-----------
sandy
kim
leslie

(3 rows affected)
```

Keep the following rules in mind when dealing with the isql command terminator:

- Sybase-supplied scripts must always use “go.”
- Your own scripts may continue to use “go” as the command terminator or may use another command terminator.
- Make sure that the selected command terminator does not allow “partial commands.” The following example shows why carriage returns should not be used as the command terminator:

```
1> update TBL set col = 0 <carriage return><carriage return>
9999 rows affected
1> where col = 99
....error message.....
```

The intent was to change only the col=99 rows, but the first command changed all rows because the carriage return actually sends the command. The **where** clause returns an error message.

◆ ***WARNING!***

OpenVMS users: do not change the command terminator. If you do, the “go” terminator no longer works and Sybase-supplied scripts and stored procedures will no longer run.

A description of the **-c** option can be found in the *SQL Server Utility Programs* manual for your platform.

Executing OpenVMS DCL Commands from *isql*

Question

How do you execute OpenVMS DCL commands in **isql**?

Answer

You can execute OpenVMS DCL commands from **isql** by spawning them from an editor. For example, if your default editor is **tpu/eve**, you can spawn a subprocess as follows:

1. From the isql prompt, invoke **tpu/eve**:

   ```
   1> edit
   ```

2. Press the Do key to access the command prompt.
3. At the command prompt, spawn a subprocess:

   ```
   Command: spawn
   ```

4. From the DCL prompt, enter DCL commands or procedures.
5. When finished, log out of the DCL subprocess to return to the editor:

   ```
   $ logout
   ```

6. Exit from **tpu/eve** to return to the **isql** prompt.

➤ ***Note***

Using Ctrl-y from **isql** to spawn a subprocess from DCL is **not** recommended because some DCL commands will not allow you to return to **isql**. This can be particularly dangerous if you have open transactions in **isql** when the subprocess is spawned.

Determining *isql* version on OpenVMS

Question

How do I find out the **isql** release and latest installed EBF on my OpenVMS platform?

Answer

To find out what version of **isql** you are running, enter the following:

```
isql /version
```

Omitting Column Names and Hyphens

Question

Can I suppress table header information (both column names and the line of hyphens) when using **isql**?

Answer

It is not possible to suppress both column names and the line of hyphens from within **isql** itself.

On UNIX platforms, however, you can delete both column names and the line of hyphens by redirecting the **isql** output through the **tail** command as shown below:

```
% isql -Usa - Ppswrd < script | tail +2
```

This strips off the first two lines, but only works for a single select statement in the batch.

Another platform-independent solution is to rename the columns in the result table to null strings. This doesn't suppress any lines, but will cause a blank line to appear in place of the base table column names in your output. For example:

```
1> select ""=title, " "=price from titles
2> go
```

where each string for the column name has a different number of spaces in it (so they are viewed as different column names).

This query returns a result table without column headers, similar to:

```
----------------------------------------------------------------
You Can Combat Computer Stress                             2.99
Straight Talk About Computers                             19.99
Silicon Valley Gastronomic Treats                         19.99
The Gourmet Microwave                                      2.99
But Is It User Friendly?                                  22.95
```

2 Bulk Copy Utility

This chapter contains information about the bulk copy utility (**bcp**) provided with SYBASE software. **bcp** is used to import and export data to and from SQL Server.

Refer to Chapter 3, “Bulk Copy Programming,” for a description of problems and techniques used in bulk copy programming.

Bulk Copy Interfaces

Three different interfaces are available for accessing the bulk copy utility. The following sections outline the capabilities of each of these interfaces.

Copy Table in the Tools Icon of Data Workbench

The Data Workbench® **Copy Tabl**e option provides a visual interface to bulk copy. Use it to learn how to use bulk copy and to perform one-time bulk copy operations.

bcp Command Line Utility

The **bcp** utility program performs automated operations in a script or command file that runs in a batch or in the background.

When run interactively, **bcp** prompts for copy behavior such as the size of columns, prefix length, and so on. You can also specify that a format file be built to store the parameters for reuse.

The **bcp** command line utility with the **-c** flag (“character” host file) provides a quick and easy way to copy data from a table to an ASCII file. With the **-n** flag (“native format” host file), you can quickly copy data in compact format. This method is good for moving tables between servers.

When data is copied out in native format, information other than the data values is also copied. As a result, only **bcp** can read this file. Character format is therefore best when transferring data to and from different applications.

➤ ***Note***

Be careful when copying data from different releases of SQL Server in native format (specified by the **-n** option) because not all releases have the same datatypes. The hardware and operating systems should be identical or at least binary compatible. Internal data representations vary by vendor and release of SQL Server.

See the *SYBASE SQL Server Utility Programs* manual for your operating system for a full discussion of **bcp** syntax and usage.

Programming *bcp* Function Calls in an Open Client Program

Open Client DB-Library provides function calls for **bcp** operations (all prefixed **bcp_**). Although bulk copy programming is a little more complex than the first two methods above, it offers the greatest degree of flexibility and control.

Typically, an Open Client DB-Library bulk copy program is used for special or complex preprocessing such as:

- Inserting or changing field delimiters (you can change field delimiters with **bcp** when copying out).
- Converting or translating special or unknown characters.
- Performing real-time processing where the information going to a table is collected from a file that is dynamically created, from a system mailbox, or from satellite transmissions.
- Embedding functionality inside an application. Rather than stopping an application to bulk copy the data, a function call or external routine does the job.

Other advantages to programming bulk copy calls include the ability to pre-sort the data, eliminate duplicates, include or exclude rows or records, perform additional tasks or actions, and interact with users.

Refer to Chapter 3, "Bulk Copy Programming", and to the *Open Client DB-Library/C Reference Manual*, for more information.

Bulk Copying Data Is Slow

Problem

bcp takes a long time to run and / or the transaction log fills up.

Explanation

If bulk copying data into a table runs slowly, or the database transaction log fills up, it may be because you are copying host file data to a table with indexes, rules, or triggers in place. In this case, **bcp** uses the slow method. Each row copied into the database is recorded as an insert in the database and logged in the transaction log and the index is updated.

You may want to drop triggers, rules, and indexes, and recreate them after the copy is complete. See the *SQL Server Utility Programs* manual for your platform for further details on **bcp** performance.

➤ ***Note***

On the PC, bulk copy to or from an OS/2 server may slow down considerably even during fast **bcp** of a huge table. To avoid this, set the **batch size** option as described in the following section.

Setting the Batch Size Option

Set the batch size using **bcp** "-b" option to a smaller percentage of the number of rows being bcp-ed in, such as ten percent at a time.

For example copy 100,000 rows by entering the following:

```
bcp in -b10000
```

This sets batch size to 10000 and commits each batch as a transaction. The reason for doing this is that if an error should occur during the copy procedure, you can begin again from the current set of 10,000. So if 60,000 rows were successfully copied and an error occurred during the next batch of 10,000, you can restart the **bcp** at row 60,001 and continue copying data (rather than starting from the beginning again). Of course, setting the batch size is a trade-off between recovering from errors and how often the commits occur. This example uses 10 percent as a guideline, but numbers may vary according to your needs.

Adjusting *cschedspins*

Another possible way to increase the speed is to adjust the *cschedspins* parameter. For SQL Server running a single engine, the optimal value is 1. The optimal value for multiple engines in release 10.0.x is 2000.

Follow these steps to change the value of *cschedspins*:

1. Shut down SQL Server.
2. Use the appropriate command to change the *cschedspins* value:

   ```
   UNIX - buildmaster -dMSTR_DEVICE -ycschedspins=32
   ```

   ```
   VMS -
   BUILDMASTER/DISK=MSTR_DEVICE/ALTER=”cschedspins”=32
   ```

   where MSTER_DEVICE is the full path and name of the master device as it appears in the RUNSERVER file (except Novell, which does not use a RUNSERVER file).

If changing the value to these recommended settings does not improve performance, increase the number in increments of 8. The value of *cschedspins* should always be 1 or a multiple of 8. Be sure to use the current version of buildmaster when making this change. Reset the *cschedspins* parameter whenever you rebuild the master device.

3. Restart SQL Server.

For additional information, refer to the SQL Server documents concerning the *cschedspins* parameter.

For more information about either the **sp_dboption** or **checkpoint** command, see the *SQL Server Reference Manual*.

For more information about **bcp** command options and performance issues, refer to the *SYBASE SQL Server Utility Programs* manual for your operating system.

Bulk Copy Aborts Before All Rows Copied

Problem

Bulk copy aborts before all rows have been copied and conversion errors are returned. Why?

Explanation

Problems exist with command or library **bcp** such as a failure to transfer data or errors in data format descriptions. For example, one or more of the following messages may return:

- `DB-Library error: Unexpected EOF encountered in bcp datafile.`
- `The conversion of <type> value to <type2> resulted in an overflow.`
- `You cannot run the non-logged version of Bulk Copy in this database.`

Action

Do the following to create and examine the error file.

1. On **bcp** in, input the following:

```
bcp in -e new_err_file
```

where *new_err_file* is the name of the new error file. Also, you may wish to use the **-m** option to increase the number of errors permitted before the **bcp** operation aborts; the default is ten errors.

In UNIX, you can use the **od** (octal dump) command to view strange or invisible characters, such as stray EBCDIC, tabs or new lines. To use **od** to view a **bcp** error file, use a command similar to:

```
od -cx data_or_error_file
```

od output is large and can be piped into a file for review, or through a screen viewer command such as **pg** or **more**. The *-cx* parameter shown here may vary depending on the type of UNIX. Use parameters that cause **od** to interpret bytes as ASCII characters and interpret short words as hexadecimal.

The od output can also be limited by using the UNIX **head** command. For example, the following command would limit the **od** output to the first 200 lines:

```
od ...| head -200
```

Using *bcp* to Restore Tables: Effects on Performance

Question

When using **bcp** to copy tables in a database, is the copy equivalent to the original in terms of performance?

Answer

If your tables are large and have had many rows deleted throughout them, rebuilding them via **bcp,** and recreating any clustered indexes, may improve I/O performance by reducing the degree of fragmentation. (You cannot use bulkcopy on an entire database at once; **bcp** only operates on a table level.)

Copying via **bcp** will remove the holes and will usually compact the rows more than they were in the original.

Rebuild tables using either **bcp** or **select into**.

Steps using bcp:

1. Use **bcp** to copy a table's data out to a host file.
2. Use the SQL **truncate table** command to truncate the original table.
3. Use bcp to copy the data from the host file back into the table.

Steps using select into:

1. Use the select into command to copy a table's rows into another table,
2. Use the SQL **truncate table** command to truncate the original table.
3. Use the SQL **insert** command to copy the rows from the other table back into the original table.

There are two types of fragmentation: internal and external.

Internal fragmentation refers to gaps occurring within the pages of a table. External fragmentation refers to gaps among a table's allocated extents.

Internal Fragmentation

Meaningful "holes" only exist when larger tables with a clustered index have a small fillfactor, and/or have small groups of rows deleted from areas spanning most of the table. These gaps are not large on a per-page basis, since through all manipulations, pages are always kept at least half full, and rows on a page are always contiguous (free space on any page is kept together at the end of the page.)

Recreating the clustered index will fill these on-page gaps, placing a uniform number of rows on all pages.

External Fragmentation

This is the fragmentation of a table's allocated extents (chains of eight pages or table building blocks). While not impacting table I/O performance, this type of fragmentation is a greater contributor than internal fragmentation to excessive reserved space allocation on a table, (space not yet reused after being partially deallocated in a **delete**).

Fragmented extents occur only when less than one extent (8–12K pages) of continuous rows are ever deleted. Until completely emptied, extents remain allocated to the table indefinitely, effectively reserving small groupings of empty pages which could otherwise be freed for use by other tables in the database.

Loading NULL Dates with Bulk Copy

Question

Can **bcp** load NULL dates?

Answer

bcp can load NULL dates if there is nothing between the delimiters for the columns. If **bcp** encounters a space, it converts that to Jan 1, 1900. For example:

```
create table foo
(seq_no int not null,
date1 datetime null,
date2 datetime null)
```

The following is the contents of a host file that we copy into table foo, which uses the tilde to delimit columns and a percent sign (%) followed by a Return (%\n) as a row terminator.

```
1~  ~%\n
2~~ %\n
3~~%\n
```

Now use **bcp** with the delimiters specified above:

```
bcp foo in foo.dat -c -t”~” -r”~%\n”
```

```
Starting copy....
3 rows copied
Clock Time (ms.): total =37 Avg =12(81.08 rows per
sec.)
```

Using **isql** to select the table rows:

```
1> select * from foo
2> go

seq_no date1 date2
1 Jan 1 1900 12:00AM     NULL
2 NULL                   Jan 1 1900  12:00AM
3 NULL                   NULL
(3 rows affected)
```

➤ ***Note***

The string “ “ is not equivalent to the value of NULL. The “ “ string is a valid ASCII value and not NULL.

Bulk Copy to Tape Not Supported

Question

I want to bulk copy a Sybase table from my UNIX machine to tape. Is there any way to do this?

Answer

You may be able to use **bcp** to copy a table to EXABYTE tapes by setting up a temporary pipe. However this use of **bcp** is not supported.

◆ ***WARNING!***

If your tables contain text or image data, bcp cannot copy them to tape.

Common Error Numbers and Messages

The remainder of this chapter gives explanations and actions you can take to remedy some errors commonly encountered when working with **bcp**.

Error 2601

Severity Level 14

Error Message Text

```
Attempt to insert duplicate key row in object
table_name with unique index index_name.
```

Explanation

This error occurs when data being loaded into a table via **bcp** would have duplicated a unique key in the table.

The duplicate record is not inserted. Additionally, the other host file records in the same **bcp** batch are also rejected. (The entire input file is treated as a single batch unless you have used the **-b** option to define a different batch size.)

Duplicate key errors are reported by SQL Server, not by the client. Therefore, you cannot capture the records causing them to an error file using the **-e** (error file) option, nor can you prevent them from stopping execution of the batch via the **-m** (max errors) option.

Action

Correct or remove any data records in the host file which duplicate values of unique keys in the database table.

One way to find out which records contain such values is to **bcp** the host file into a work table so that you can compare the key values of the original table against the values in the work table.

After you have created the work table and loaded the **bcp** input into it, find the non-unique key values in it by using a **select** statement of the form:

```
1> select w.key
2> from work w,  original o
3> where w.key = o.key
```

where *work* is the name of your work table and *original* is the name of the table into which you were copying data when the 2601 error occurred.

Once you have identified the non-unique values that are causing the error, you are ready to resolve the problems in the host file. In the host file, either remove the records containing those values, or edit the values to be unique, depending on what is most appropriate to your business needs.

Error 4207

Severity Level 16

Error Message Text

```
Msg 4207, Level 16, State 4
DUMP TRANsaction is not allowed while the select
into/bulk copy option is enabled: use DUMP
DATABASE, or disable the option with sp_dboption.
```

Explanation

This message occurs when you do a non-logged operation such as bulkcopy on the database followed by the **dump transaction** command. The **dump transaction** command is not allowed because the log cannot be properly recovered. For example, if you use the **select into** command followed by an **update** command, the **update** command cannot be recovered since the **select into** command was not logged.

Action

You should dump the database before you attempt to dump the transaction log. Merely disabling the **bulkcopy/select into** option using **sp_dboption** does not allow you to dump the log. Although dumping the database is the best solution, it is possible to dump the transaction log with the **truncate_only** option until a full database dump is convenient.

The reason for this requirement is that **bcp** is performing operations that are not logged in the log segment and you can not dump a log transaction because the log segment is empty.

Error 4806

Severity Level 16

Error Message Text

```
You cannot run the non-logged version of bulk copy
in this database. Please check with the DBO.
```

Explanation

This error occurs when the **select/into bulkcopy** option is not set to true when using “fast” bulk copy into a table that has no indexes or triggers.

Action

Use the **sp_dboption** stored procedure to set the **select/into bulkcopy** option as follows:

```
1> sp_dboption database_name,
2> "select into/bulkcopy", true
3> go
```

```
1> use database_name
2> go
```

```
1> checkpoint
2> go
```

Once a minimally-logged operation such as “fast” bulk copy runs in the database, you are not allowed to dump the transaction log to a device, because unlogged changes are not recoverable. Instead, you should do a **dump database** as soon as possible to restore recoverability and allow transaction dumps to devices again.

When you copy into a table that has indexes or triggers, a slower version of **bcp** is automatically used. The slow version, which does log data inserts in the transaction log, can cause the transaction log to become very large. You may need to use **dump transaction with truncate_only** until you can perform a full database dump. If you must do this, you will lose the ability to recover up-to-the-minute changes in data in case of a media failure. You may also consider using smaller batch sizes.

Additional Information

By default, the **select into/bulkcopy** option in newly created databases is set to the same as that in *model*.

Error 4808

Severity Level 10

Error Message Text

```
Msg 4808, Level 10, State 2
The bulk copy of this table has been aborted
because the checkpoint record could not be written
in the log. Please free up space in the database.
```

Explanation

This error occurs when the transaction log is completely full. The database administrator must dump the log using the **truncate_only** or **no_log** options. A corresponding error occurs in SQL Server.

Action

◆ ***WARNING!***

Notify your database administrator; the database needs to be dumped at once, as you have no usable log.

Part of the data from the **bcp** operation may be in the table. The **bcp** output and the table should be checked before retrying the **bcp** to avoid duplicate or missing records.

Error 4810

Severity Level 16

Error Message Text

```
Msg 4810, Level 16, State 2
Expected TEXT token in data stream for bulk copy
of text data.
```

Explanation

This error indicates that **bcp** could not find the text token in the input Tabular Data Stream (TDS).

Action

You can avoid seeing this error by batching **bcp** with **bcp -b**.

Error 4814

Severity Level 16

Error Message Text

```
Msg 4814, Level 20, State 2
Msg Bulk_getschema: Unable to open the system
catalog SYSCOLUMNS in database ”%.*s” while
attempting to retrieve table schema informatiom.
Run DBCC CHECKTABLE on sysobjectsin “%.*s”.
```

Explanation

This error message contains a mistake in the last line of text. It should read as follows:

```
Msg 4814, Level 20, State 2 SYSCOLUMNS in ‘%.*s“.S
Msg Bulk_getschema: Unable to open the system
catalog SYSCOLUMNS in database ”%.*s” while
attempting to retrieve table schema informatiom.
Run DBCC CHECKTABLE on syscolumns in “%.*s”.
```

Action

Run **dbcc checktable** on *syscolumns* and address any errors reported.

”Unexpected eof” or “bcp stopped by data conversion”

Problem

One of the following errors appears and **bcp** stops:

- “Unexpected eof”
- “bcp stopped by data conversion”

Explanation

These messages mean that **bcp** has found a problem with the host data file and cannot proceed. This can happen for a variety of reasons, as described in the following sections.

Action

The specific remedy depends on the cause of the error. However, in general you need to capture the bad rows so that you can determine why they have generated errors. To do this:

- Specify the **-e** option to create an error file.
- Use the **-m** option to raise the number of errors allowed before bcp stops running. (The default is 10; if you are loading large numbers of records and encountering many errors, you will need to raise this significantly. You may want to start by setting it to 100.)

After rerunning bcp with these options, analyze the contents of the error file to determine which of the following issues you have encountered.

Blank Lines

In the host data file, any completely blank line, or one that contains spaces, is interpreted as a null record.

Remove all blank lines and null records.

Unconverted Characters

Converting data from EBCDIC format (typically IBM mainframes) to ASCII can leave some unconverted characters. Utility programs on platforms such as UNIX and VAX VMS can perform most, if not all, of the translation properly. However, if the utility encounters an

EBCDIC character that it can not translate, the character is left untranslated. These are often “overstrike” characters.

When **bcp** encounters the untranslated character, it displays the “unexpected eof” or “bcp stopped by data conversion” error message.

If this only happens occasionally, edit the host data file to remove or replace the problem characters. You must inspect the data to determine where the copy failed, because **bcp** does not tell you. If necessary, cross-reference an EBCDIC collating sequence chart to determine which characters caused the problems.

If this problem occurs frequently, use the **-e** and **-m** options as described under the “Action” heading in this section to capture the bad rows.

If you frequently load translated data, and especially if you load translated data into your server as part of an automated process, you can create your own translation script, program, or command file to perform the EBCDIC-to-ASCII translation as preprocessing to the bulk copy itself.

Programming has two advantages:

- The program can create an exception file to handle data that is not translatable, using a special translation table to handle what the standard translation table does not define. Records not copied are deposited into the exception table for further inspection and resolution.
- The program can perform the **bcp** itself with a finer degree of control using bulk copy function calls.

“Garbage” Characters

The host data file contains control or garbage characters such as Ctrl-m, which represents a hard return. In particular, noisy data transmission can generate characters outside the range of a platform’s collating sequence.

Many garbage characters cannot even be seen with editors such as **vi** and **emacs**. If you suspect garbage in your data, use the UNIX **od** command to produce a hex dump or the <esc>:%1 option under **vi** to be sure of the integrity of the data.

Cross-Platform Row Terminator Differences

In a multi-platform environment with both UNIX and PC applications, there are characters with special meanings in one system which may produce problems in another. For example, on PC platforms, the \r (carriage return) is the common row terminator; On UNIX platforms, the \n (new line character) is the usual row terminator. When using **bcp** to move data between PC and UNIX operating environments, you must take these differences into account. Otherwise, eof or data conversion errors will occur.

Field Terminators Read As Data

Sometimes a character being used as a terminator character for a field or record is read as part of the field itself. The comma is often used as a field termination character when importing data from a PC program. Sometimes the “unexpected eof” error occurs. In other cases, the server data table is wrong due to misalignment of the data.

Carefully check your host data file parameters and content to be sure the delimiters are not included in the data. For automated transfers, you may need to create filters or conversion programs.

VMS File Record Format

On OpenVMS platforms, the “unexpected eof” error can occur due to an incorrect file record format. **bcp** requires a record format **stream_lf**.

To see the record format, type:

```
$ DIR/FULL filename
```

If the file was created by an editor, it is probably “variable length, maximum *xxx* bytes.” Files created by **bcp** when copying a database table to a host file are in the correct format.

Choose delimiters that do not appear in your data when bulkcopying data from SQL Server to a host file.

To change the record format to **stream_lf**, perform the following steps:

1. Create a file *filename.fdl* containing OpenVMS File Description information, parameters and controls:

   ```
   $ ANALYZE/RMS/FDL filename.dat
   ```

2. Edit *filename.fdl*. Under the RECORD section, change the FORMAT specification to **stream_lf**.

3. Create a new version of the *filename.dat* in **stream_lf** record format:

   ```
   $ CONVERT/FDL=filename filename.dat filename.dat
   ```

   .fdl is the default extension in *FDL=filename*. If you want a different file name for the conversion instead of the next version, replace the second *filename.dat* with *newfilename.dat*.

4. Include an error file to capture errors. This makes it easier to find problems.

3 Bulk Copy Programming

This section describes programming tips and techniques for the bulk copy function calls available with Open Client DB-Library.

Use of the Error File

Question

When is the **bcp** error file used?

Answer

Use the error file option during bulk copy operations from host files to SQL Servers. Refer to the **bcp_init** section in the *Open Client DB-Library Reference Manual* for more information.

bcp from Bound Program Variables

If you are performing bulk copy from bound program variables, the error file is not used and the specification of the error file in **bcp_init** is ignored. This is because copies from program variables are done through **bcp_sendrow**, which returns FAIL whenever a given row fails. At that point, the program already has access to the row values which failed. The Open Client DB-Library error handler raises an error to tell the application the reason for failure.

bcp Error File and *bcp_exec*

The whole file is copied through **bcp_exec**, so you must use another method of tracking errors and bad rows when copying from host files. The application must be programmed in such a way that an error does not cause the program to abort.

bcp Programming and Paging

Using **bcp_init, bcp_bind**, and **bcp_done** in a 3GL program can cause each row to be started on a new page, even though the row might not be a full page in length. If your function to load the data into the database is called for each row until EOF, and in that same function are **bcp_init, bcp_bind**(s), and **bcp_done**, you may see this problem.

This is because having initialization (**bcp_init)** occur each time a row is to be added causes each row to be stored on a new page. This is not documented in the *Open Client DB-Library Reference Manual*.

Instead of using all three calls in the same function, run **bcp_init** and **bcp_done** outside the function that does the **bcp_bind**(s). Be aware that now either all the data is sent when **bcp_done** is executed, or nothing is sent if the program crashes before **bcp_done**.

Using *bcp* to Import Data

Question

How do you use Open Client DB-Library **bcp** functions to import **bcp** data from an environment outside SQL Server to a given SQL Server table?

Answer

The example program *bulkcopy.c* (included in the DB-Library samples subdirectory of your SYBASE installation) shows how to import data using **bcp** functions. The section following is an example of real-time use of that program. You may wish to look at the *bulkcopy.c* code and / or compile and run the sample as you read the following material.

Finding Source Code for the *bulkcopy.c* Sample

The source code for a sample bulkcopy program is shipped with the DB-Library samples. The name of the source file is *bulkcopy.c*. A data file, *bcpdata.in*, is also included in the sample directory. The samples are normally located in the *sample/dblibrary* subdirectory of your SYBASE root directory.

If you wish to compile and execute the sample code, copy it to another directory first. Consult your *Open Client/Server Supplement* for detailed information on finding and running this sample.

Executing *bulkcopy.c*

When the program executes, it asks a series of questions about SQL Server and running the program, such as whether to create a database, and if so, on which device to create it. If you create a database, the **select into/bulk copy** database option is enabled. If you are copying into an existing database, be sure to enable the **select into/bulk copy** option before running the program.

The program then asks whether to copy a host file or to simulate real-time. In this example, the host file *bcpdata.in* is copied in. *bcpdata.in* is also distributed in the sample directory of Open Client DB-Library release 4.6. If you choose real-time, *bulkcopy.c* prompts you to enter data to copy in.

Note that everything this program does is dynamic, that is, based on user responses to the questions asked, except the table *alltypes* and the **bcp** error file. The *alltypes* table is predetermined, and therefore hard-coded. The **bcp** error file has also been hard-coded as *bulkcopy.c.errors*. This is actually ignored, however, because this program binds data to program variables. See “Use of the Error File” on page 3-2.

The standard error handler and a modified message handler have been installed. Severe errors cause the program to abort. Informational messages such as “changing database...” are suppressed.

Example: Host File Method

This sample execution first creates the database *bcp_db* on the device *xy0e* and then copies the host file *bcpdata.in* to the *alltypes* table. The **bcp** copy is performed in character mode because host file columns are all character. The default column terminator, <TAB>, and default row terminator, <NEWLINE>, are used. The program reads the host file using the C function **fscanf**, binds the data using the Open Client DB-Library function **bcp_bind** and then sends the data to the *alltypes* table with the Open Client DB-Library function **bcp_sendrow:**

Figure 3-1: Sample output of routine importing data via bcp

```
Create database and 'alltypes' table (y/n) ? y
Name of database to create ? bcp_db
Name of Sybase device to create database on ? xy0e
Use real-time simulation or host file data (r/h) ? h
Name of host file to read from ? bcpdata.in

Creating the 'bcp_db' database...Successfully created !
Creating the 'alltypes' table...Successfully created !
Select/into, bulkcopy option set on 'bcp_db' !

Moving data from host file to database...
```

Figure 3-1: Sample output of routine importing data via bcp (continued)

```
---> 46 206 1005 Douglas 0x22 1 1.234 sometitle Arthur
---> 60 345 2321 Andrea 0x14 1 2.46 freetitle Sally
---> 67 346 1141 Arthur 0x17 1 456.23 fulltitle Sally
---> 47 347 9863 Jonathan 0x34 1 4568.97 smalltitle Sally
---> 76 348 9200 Beatrice 0x32 1 3456.20 largetitle Arthur
---> 26 349 2233 Carol 0x11 0 999.99 newtitle Andrea
---> 27 350 1800 Howard 0x12 1 1887.6 footitle Andrea
---> 45 351 6553 David 0x18 1 19.74 booktitle Ted
---> 67 352 3131 Michael 0x10 1 2166.334 entitle Ted
---> 67 353 3312 Claire 0x16 1 32.06 retitle Carol
---> 67 354 1051 Maggie 0x15 0 678.876 titlebout Andrea
---> 67 355 2520 Ted 0x23 1 111.22 titlerole Carol
---> 67 356 7623 Chuck 0x24 1 3030.0 landtitle Arthur
---> 67 357 1548 Nina 0x25 0 78.90 atitle Sally
---> 67 358 6006 Susan 0x26 1 9.1 subtitle Andrea
---> 67 359 1212 Samuel 0x27 1 7.6 jobtitle Ted

*** 16 Rows Copied Successfully **
```

Example: Real-Time Method

This example simulates real-time bulk copy usage. Enter the data interactively. Next to each column name prompt is the table's column type in square brackets, for example, [*tinyint*]. As with the host file method above, the data is then bound and sent to the *alltypes* table:

Figure 3-2: Sample output of real-time bulk copy routine

```
Create database and 'alltypes' table (y/n) ? n
Name of database to use ? bcp_db
Use real-time simulation or host file data (r/h) ? r

Real-time simulation has been chosen.

To simulate real-time, you will enter your own data in the
'alltypes' table. The data will be bulk copied in 1 row at a
time. To EXIT at any time, type '*' at any field. Note that all
table columns do not allow nulls.
```

Figure 3-2: Sample output of real-time bulk copy routine (continued)

```
Age [tinyint]....................: 22
Userid [smallint]................: 12345
Royalty [int]....................: 98765
Name [char(25)]..................: John Doe
Title ID [varbinary(20)]
(example: 0x123).................: 0x1af3
US Citizen ? (0/1) [bit].........: 1
Account [float]..................: 22.010101
Title [varchar(20)]..............: Programmer
Manager [char(25)]...............: Bill Cohen

--> 22 12345 98765 John Doe 0x1af3 1 22.010101 Programmer Bill
Cohen

Age [tinyint]....................: 34
Userid [smallint]................: 555
Royalty [int]....................: 10101
Name [char(25)]..................: Richard Smith
Title ID [varbinary(20)]
(example: 0x123).................: 0xaaa
US Citizen ? (0/1) [bit].........: 0
Account [float]..................: 298.298
Title [varchar(20)]..............: Sr. Admin
Manager [char(25)]...............: George Hill

--> 34 555 10101 Richard Smith 0xaaa 0 298.298 Sr. Admin George
Hill

Age [tinyint].....................: *

Exiting, this row not inserted !

*** 2 Rows Copied Successfully ***
```

Connectivity/3GL

4 Embedded SQL

This chapter gives explanations and actions you can take to resolve some common error messages displayed by Embedded SQL™. It also contains additional troubleshooting information. The chapter is divided into four sections:

- Material that applies to both pre-10.0 Embedded SQL and 10.0 and later Embedded SQL, including how to migrate Embedded SQL applications to release 10.0 from earlier releases
- Material that applies to 10.0 and later Embedded SQL only
- Material that applies to pre-10.0 Embedded SQL only
- Explanations of commonly encountered errors

You may need to know your programming library and operating system library versions when calling Sybase Technical Support. This is so that Technical Support can ensure that you are not encountering a bug that has already been fixed in a later version of the product, and that you are running on a certified version of the compiler and/or precompiler. For information about finding library version numbers, refer to Appendix A, "Finding Your Library Version".

All Embedded SQL Releases

This section applies to pre-10.0, 10.0, and later releases of Embedded SQL.

-K Option Results in Invalid SQL

Problem

Using the SQL checking option results in invalid SQL being sent to the SQL Server.

Explanation

The **-K** (UNIX) or /**checksql** (VMS) option checks all user SQL at precompile time on the Server. The pre-10.0.2 COBOL precompiler did not do this reliably. Any attempt to use a variable of type *DISPLAY SIGN LEADING SEPARATE* in a query resulted in invalid SQL being sent to the Server, which would then report an error.

The **-K** option was broken in 10.0.1, but is fixed in 10.0.2.

Action

If using the COBOL precompiler, try the same precompile without using the **-K** option. This will allow valid SQL to be generated. The problem is fixed in EBFs 1574 through 1583 and higher for version 4.0.4.

DISPLAY SIGN LEADING SEPARATE is a COBOL datatype that has no analog in C.

➤ ***Note***

The -K option was -C in Embedded SQL releases before 10.0.

Embedded SQL vs. *isql* Performance with Joins

Problem

A query with table joins returns rows much more quickly when the query is sent via **isql** than when it is sent via Embedded SQL.

Explanation

The optimizer uses different query plans in Embedded SQL than it does in **isql**. If a cursor is opened in Embedded SQL with host variables in the **select** statement, the optimizer will discover type mismatches between the variables and the database columns, and will convert the columns with the result that no index is used.

Action

In the **declare cursor** statement, use an explicit **convert** (**:** ***host_var***). Additionally, if the cursor declarations contain many **or** clauses, performance may improve after you rewrite them using **union**.

Migrating Embedded SQL Applications

The following section is intended to help developers of Embedded SQL Applications understand the issues involved in migrating release 4.x Embedded SQL applications to System 10™.

First check the “Considerations.” If migration makes sense for your applications, then use the guidelines for modifying your release 4.x source code to make the transition a smooth one.

Considerations Before You Begin

- The System 10 Embedded SQL products run **only** against System 10 SQL Servers.
  - The generated code explicitly checks the TDS level from the Server connection, and raises an error (-25013) if it is not at System 10 level. Therefore, do not migrate release 4.x Embedded SQL applications to System 10 unless they will be run exclusively against System 10 SQL Servers.
- System 10 precompilers cannot **insert** / **update** *text* and *image* columns.
  - If your Embedded SQL applications insert or update text and image columns, do not migrate them to System 10.
- Remote server / password combinations and interfaces file names cannot be specified in the **using** ***server_name*** clause of a **connect** statement.
  - The **using** ***server_name*** clause of a **connect** statement in Release 4.x required an ESQL application to put **isql**-style command line syntax into the ***server_name*** string. The ESQL application could specify: SQL Server name, interface filename, and a list of remote-server / password pairs.
  - In System 10, the only information specified in ***server_name*** is the name of the SQL Server to use (from the interfaces file). There is no way to set the interfaces file to use for a connection statement. However, an Embedded SQL / C application can call **ct_config(, CS_SET, CS_IFILE,...)** using the global context handle to achieve this.
  - There is no way to specify remote-server / password combinations in a System 10 ESQL **connect** statement. An ESQL / C application can set remote passwords using the **ct_remote_pwd()** API. However, **ct_remote_pwd(...,SET,...)** must be called before the connection is established with the **ct_connect()**

call. Allocation of the connection handle and the **ct_connect()** call all happen in the same set of code generated for the **connect** statement, so an ESQL / C developer would have to modify the generated code to set remote passwords in this manner.

- System 10 Embedded SQL products require Client-Library™.
  - In addition to having System 10 SQL Server installed, you must also have System 10 Client-Library installed (it is included in the Embedded SQL product package).
- Release 4.x source code requires some modifications before use with System 10 precompilers.
  - Do not attempt to use the System 10 Embedded SQL precompiler on source targeted for a pre-System 10 Server until you have made the source changes described in the following section.

Source Code Changes Required

The following source code changes are required to precompile release 4.x applications with the System 10 Embedded SQL precompiler.

- Change *DBXXXX* datatype declarations in a **declare section** to their analogous *CS_XXXX* typedefs.
- Check for code which takes actions based on SQLCODE values. SQLCODE values have changed. Any code which takes actions based on any specific values of SQLCODE (other than 0 or 100) must be re-written. SQLCODE must be declared appropriately.
- Because SQLCA structure has changed, rewrite references to any of the fields which no longer exist. Refer to the *Open Client Embedded SQL Programmers Guide* for your language for the new definition. These references may appear in **whenever** handlers, or in other code which checks **exec sql** statement completion.
- Change dynamic statement parameter markers from ":dummy" to "?".
- Shorten dynamic statement names to 10 characters or less. Because of a constraint in SQL Server, dynamic statement names are currently limited to 10 characters in length. Previously, the length limit was 63.
- Remove any NoParse, NoProc, and Recompile directives; they are no longer supported.

- Check source files that use static cursors; System 10 allows only one **open** statement per static cursor in a source file, per ANSI standards. The precompiler has no such limitations, but will raise a warning.
- Check **exec sql include *"file_name"*** statements. They now require double quotes (" ") around the file name. Previously, this was not the case.
- Replace **release** statements with **disconnect** statements; the **release** statement is no longer supported. Likewise, remove **release** clauses from **commit** and **rollback** statements and add a **disconnect** statement instead.
- If your source uses double quotes (") to identify literal strings inside SQL statements, you must use the **-d** (**/delimited**) command line switch when you precompile using the System 10 precompiler. By default, System 10 Embedded SQL applications treat double-quoted (") strings inside SQL statements as delimited identifiers rather than as literal string values. The **-d** (**/delimited**) command line switch disables delimited identifiers so that your double quoted literal strings will be correctly interpreted.
- Remove **cancel** statements. The statement is no longer supported because it is no longer necessary; the generated application code automatically cancels any unexpected result data.
- Add return code (SQLCODE) checks after all **prepare** statements. Dynamic **prepare** statements in the Release 4.x product did not fail when they contained SQL syntax errors. In System 10, however, SQL syntax errors cause dynamic **prepare** statements to fail.
- Behavior of *updatable* cursors has changed. For a cursor to be *updatable* in Embedded SQL release 4.x, the underlying table had to be *browsable* (having a unique index and a timestamp column). This restriction no longer applies. Also, release 4.x may have allowed updates of some columns of a table through a cursor where the **select** statement contained a **union**, **order by**, an aggregate function or the **distinct** clause. The System 10 SQL Server does not allow updates on cursors with these clauses.
- Remove any **for browse** clauses that appear in the **select** query **declare cursor** statements. The release 4.x product allowed the **for browse** clause in the select query of a **declare cursor** statement (although documentation said otherwise). The System 10 SQL Server does not allow this.

- The release 4.x **execute immediate** statement was executed as a simple language command. The statement is executed as a dynamic **execute immediate** command in System 10. The System 10 SQL Server applies different restrictions to the **execute immediate** command than it does to a language command. For example, it would have been possible to do a **select into *table_name*** with release 4.x, but this will fail on a System 10 SQL Server.
- Beware of using 0-length character string variables as *input* to execution of a query; changes in the way release 10.0 Embedded SQL handles these variables could result in significant behavior change in applications. 4.0.4 ESQL / C converted such 0-length character string variables to a single space. In release 10.0, a 0-length character variable is passed to the SQL-Server as a NULL value, just as if you had an indicator variable with value -1 associated with the character variable. The 10.0.2 maintenance release re-implements the 4.0.4-style conversion.

Further Reading

Before attempting to migrate Embedded SQL Release 4.x applications to release 10.x, be sure to read the Embedded SQL release 10.x product manuals. These include:

Open Client Embedded SQL/C or *COBOL Programmer's Guide*

- The "Backward Compatibility" section in Chapter 1 is of particular interest.

Open Client Embedded SQL Reference Manual

- Contains the descriptions and syntax of valid Embedded SQL statements.

Open Client/Server Supplements and *Installation Guides*

- Contain information on installing and configuring the precompiler's flags.

Embedded SQL Release 10.0 or Later

The material in this section applies only to Embedded SQL release 10.0 or later.

Double Quotes: Delimited Identifiers

Problem

Starting with Embedded SQL 10.0.1, the precompiler has begun enforcing the concept of **delimited identifiers**: the ability to refer to column names inside double quotes.

As a consequence, SQL syntax that worked in earlier releases of Embedded SQL, and which continues to process successfully through **isql**, will generate a -207 error message. For example:

```
exec sql SELECT
     au_lname
   , "none"
   , au_fname
FROM
     pubs2..authors
WHERE
     au_id = "472-27-2349"
INTO
     :lastnm
   , :middlenm
   , :firstnm
;
```

will result in:

```
** SQLCODE=(-207)
** SQL Server Error
** Invalid column name 'none'.

** SQLCODE=(-207)
** SQL Server Error
** Invalid column name '472-27-2349'.
```

Resolution

Either of these approaches may be used:

1. Specify precompiler option **-d** to turn off the interpretation of double quotes as the identifier delimiter.
2. Use single quotes for character text. This must be done everywhere within the SQL command—simply replacing “none” with ‘none’ would only result in a new -207 error message.

Unresolved Symbols at Link Time

Problem

At link time, the following symbols are listed as undefined:

```
_sqlctdiag
_sqlinitctx
_sqlprolog
_sqlsetinterr
_sqlepilog
```

Explanation

There has been a change in the way the include file *sybtesql.h* is handled between Embedded SQL / C 10.0 and 10.0.1. In release 10.0, the precompiler put the directive **#include sybtesql.h** at the bottom of the generated *.c* file. Release 10.0.1 does not do this. The function of that **#include** is now fulfilled by *sybesql.o*; if *sybesql.o* has not been compiled and linked, the symbols listed are unresolved.

Action

Release 10.0.1 contains a file called *sybesql.c*, located in the *include* subdirectory of your SYBASE directory tree, which must be compiled and linked in with your programs. This file consists only of these two include statements: **#include sybhesql.h** and **#include sybtesql.h**. Make sure that your makefile compiles *sybesql.c* to produce *sybesql.o* and links *sybesql.o* with your application.

Mysterious Program Failure

Problem

A program fails mysteriously; the precompiler gives no warning.

Action

Make sure you did not accidentally leave out the **connect** statement.

Missing Error Message Texts and Negative Return Codes

Problem

A SQLCODE value is returned as a negative integer, and no error message text is available.

Explanation

In release 10.0 and later of Embedded SQL, SQLCODE values in the range (0 > SQLCODE > -25000) are messages from the SQL Server. A **select from sysmessages where error =** ***return_code_converted_to_positive_int*** will show the error message text.

Action

To get Embedded SQL to include the error message text, add the following statement to your application in the working storage section:

```
EXEC SQL INCLUDE SQLCA END-EXEC
```

or

```
EXEC SQL INCLUDE SQLCA;
```

The result is that *sqlerrmc* will have the error message text filled in, and any variables in the message text (*%s*, *%d*, and so on) will also be filled in.

Error Numbers Beyond -25000

Problem

Large (negative) numbers are returned in the *SQLCODE* variable, for example `-084083715`, and if SQLCA has been included, have an accompanying error message, for example:

```
ct_connect(): network packet layer: internal net library error:
Specified server name attribute could not be found
```

Explanation

Embedded SQL programs are built on Open Client Client-Library, which returns these “telephone number” error message numbers when it experiences a problem communicating with a server or handling the data passed to, or returned from, a server.

The error number is a bit map for the program layer, error origin, severity, and internal error number. Such error numbers are of little value to the Embedded SQL application developer, as the bit maps used are primarily defined in internal structures and in libraries.

Fortunately, the error message text contains all the relevant information.

Action

The error message begins with the Client-Library function in which the problem occurred; it is sometimes easy to deduce the Embedded SQL statement which corresponds to that function. For example, **ct_connect** is generated for an **exec sql CONNECT** statement. You can also look at the generated code to determine this.

The layer and protocol information is minimally useful; use them as a context for evaluating the error description.

The error description tells you what problem was encountered by Client-Library in your program; the example message above means that the Server name specified in the **connect** statement, or in the DSQUERY environment variable, could not be found in the interfaces file.

Pre-10.0 Embedded SQL

The material in this section applies only to pre-10.0 releases of Embedded SQL.

Message Processing Routines in 4.0.4

The documentation and samples provided with Embedded SQL release 4.0.4 contain rudimentary routines for processing error and warning message routines. The routines supplied with the Embedded SQL 4.0.4 run-time library provide more information.

Sample output from the Sybase error message routines help Sybase Technical Support to determine what is happening when a problem occurs.

The easiest way to get these routines is to precompile the program using the **-V** (UNIX) or **/VERBOSE** (VMS) option. This puts C code into the program file created by the precompiler that will test SQL code after every **EXEC SQL** statement and call the message printing routines if appropriate.

To avoid dual error messages, remove or comment out any **WHENEVER SQLERROR** or **WHENEVER SQLWARNING** statements that invoke handlers copied from Sybase documentation or examples or already coded by a programmer.

➤ ***Note***

The documentation provided with Embedded SQL outlines this last method, and states that the library routines are invoked via “warning_handler(&sqlca)” and “error_handler(&sqlca)”. They are not invoked in this way and following these directions will result in missing externals.

Alternatively, the program could contain the statements:

```
WHENEVER SQLERROR CALL sqlerror(&sqlca);
WHENEVER SQLWARNING CALL sqlwarning(&sqlca);
```

You would **not** supply any routines named *sqlerror* or *sqlwarning*. The Sybase library error message routines will be invoked, in the same way as with the **-V** precompiler option.

If a program needs its own error or warning processing routine, the program can call *sqlerror* or *sqlwarning* from within its own routine to print the error message, and then do the additional processing necessary.

Common Error Numbers and Messages

The remainder of this chapter gives explanations and actions you can take to remedy some errors commonly encountered when working with Embedded SQL.

Errors -30 Through -35

Following are brief descriptions of some release 4.x Embedded SQL error messages.

These messages apply only to release 4.x.

Error -30
SQLERR_CURSOR_COULDNT_OPEN
Couldn't open a cursor.

Error -31
SQLERR_NO_QUAL
Occurs when **dbqual** fails, usually because a table is not browsable. See "Error -31" on page 4-19 for details.

Error -32
SQLERR_DEADLOCK_DEATH
Indicates that the transaction in progress on this connection has been terminated to break a deadlock. See the *SQL Server Troubleshooting Guide* section on Error 1205 for details.

Error -33
SQLERR_BAD_PARAM_NAME
Undefined (or improperly formatted) parameter appears in **EXECUTE...USING** statement. See Error -34 (below) for related problems.

Error -34
SQLERR_NOMOREPARAMS
Variable value(s) missing from a dynamic **SQL EXECUTE** statement. When using dynamic parameter markers ("?") in a **PREPARE SQL** statement, the **EXECUTE** statement must include host variables or system descriptor names for each "?"; that is, the **PREPARE** and **EXECUTE** statements must look something like:

```
...
exec sql PREPARE  querycmd FROM
"select * from tblx where colB = ?"
...
exec sql EXECUTE  querycmd USING:search_vala;
...
```

Error -35
SQLERR_STPROC_FAILED
Stored procedure failed.

Error -31

Error Message Text

```
Couldn't obtain unique row qualification for
CURRENT OF a cursor.
```

Explanation

Applies to release 4.x only.

This message results when the user is attempting to do a **DELETE...WHERE CURRENT OF...** or **UPDATE...WHERE CURRENT OF...**, and the table in question is not browsable. The implementation of these statements involves invoking the DB-Library function **dbqual**. If this function fails, then this error results.

Action

For a table to be browsable, it must have a unique index and it must have a timestamp column named *timestamp*. The user ordinarily gets such a column on a table by executing the following in **isql** (for a table named *foo* with a column named *bar*):

```
1> alter table foo add timestamp
2> go
```

```
1> update table foo set bar=anything
2> go
```

The **update** statement puts time values into the timestamps.

Error -31 is a run-time error. However, unlike most run-time errors, Error -31 does not print the query that caused the error. This is because for a **FETCH**, the query is not sent to SQL Server. It is handled on the client by the precompiler run-time library. Therefore, no SQL Server query exists to print.

Error 114

Problem

You receive error 114:

```
Attempt to access item beyond bounds of memory.
```

Explanation

Applies to both 4.x and 10.x releases.

When using Embedded SQL with Microfocus COBOL, one common cause of Error 114 is a reference to a host variable with usage of **COMP**. One example is a **FETCH** that looks like the following in the Embedded SQL / COBOL program:

```
1000-MAIN-PROCESS SECTION.
1000-GET-SC.
     DISPLAY "FETCH SC".
     EXEC SQL FETCH sclist INTO
     :F-SC-FIN-ID,
     :F-SC-LOAN-NUM,
     :F-SC-CATEG-ID,
     :F-SC-INTERNAL-ID, etc.
     END-EXEC.
```

where:

```
01 F_SC_LIST.
   05 F-SC-FIN-ID        PIC X(4).
   05 F-SC-LOAN-NUM      PIC X(5).
   05 F-SC-CATEG-ID      PIC X(8).
   05 F-SC-INTERNAL-ID   PIC S9(9)  COMP.
etc...
```

The column *SC_INTERNAL_ID* is declared as an **int** with length 4 in the table on the SQL Server and this column is bound to *F-SC-INTERNAL-ID*.

If *F-SC-INTERNAL-ID* is described as **PIC S9(2)**, the program does not generate the 114 error. If it is defined as anything greater than 2, for example **S9(9)**, you will see this error.

The problem is that *F-SC-INTERNAL-ID* is part of a group item which has some elements whose size is odd. This causes *F-SC-INTERNAL-ID* to start at an odd-numbered address. If it is a **PIC S9(2)**, it takes up only one byte, and it is legal to have it at an odd-numbered address. If it is declared with a larger picture, it takes 2 or 4 bytes. **int**s start on a word boundary (4 bytes for most platforms, but may be as large as 8 bytes). In fact, since level 01 items are always aligned on an even boundary, it is being put at an odd address and Microfocus's run-time library thinks that odd-addressing errors are references to addresses out of range.

Action

Often you can avoid this error by declaring *F-SC-INTERNAL-ID* to have the usage **COMP SYNC** rather than **COMP**. However, the record description (for example, *F_SC_LIST*) may define a file from an external source, so it may not be possible to use **SYNC**. In this case, use a second version of the record definition with the **SYNC** clause. Code **MOVE** statements to transfer data between them.

Also, add the flag **-C IBMCOMP** in the compile line in the makefile.

Error 916

Error Message Text

```
Server user id %d is not a valid user in database
'%.*s'
```

Explanation

Applies to release 10.x and later.

On **open cursor** (dynamic), this error is returned. The problem is that the *guest* user is missing from the database being accessed. The *guest* user does not need any particular privileges, but it must be present in the database. This case of the 916 error raised by absence of the *guest* user only applies to **open cursor**.

Action

Execute the command **sp_adduser guest** in the database being accessed.

This behavior should not appear in the 10.0.2 SQL Server release, in case you should not wish to add a *guest* user.

Error 5702

Severity Level 10

Error Message Text

```
The SQL Server is terminating this process.
```

Explanation

Applies to both 4.x and 10.x releases

This is an informational message, and only appears in conjunction with other errors. For further explanation, see “Suppressing Informational Messages” on page 5-2.

Unresolved External References _sql_chk_xact and _sql_put_tds_vsn

Error Message Text

```
Unresolved external references _sql_chk_xact and
_sql_put_tds_vsn
```

Explanation

Applies to release 4.x only.

Generally, the run-time libraries we supply are used as libraries, with one copy of every necessary routine bound into every program. For efficiency reasons, users sometimes try to create dynamic link libraries from our supplied static library. The 4.0.4 precompiler run-time library contained two modules, *sqlstproc.o* and *sqlflogin.o*, which contain two unresolved externals, *_sqlchkxact* and *_sql_put_tds_vsn*. This makes dynamic link libraries fail because of these two missing entry points. (There is no problem if the libraries are used as static libraries.)

Action

The best way to solve this problem is to remove the two problem routines from the supplied library. On UNIX, the following command will do it:

```
ar dv $SYBASE/lib/libsybesql.a sqlstproc.o
sqlflogin.o
```

Then **make** the dynamic link library.

Another way to proceed is to supply two dummy routines *sqlchkxact* and *sql_put_tds_vsn*, which go into the library. These routines will never be called.

5 Open Client DB-Library

This section covers some of the questions and problems that can occur when using Open Client DB-Library.

Suppressing Informational Messages

Problem

Customers, especially those using Microsoft Access or ODBC, may encounter server messages 5701, 5703 and 5704. You may also see these if you have programs written to connect with a 4.0 SQL Server which are now connecting to a 4.2 or later SQL Server.

5701 Error Message Text

```
Changed database context to ‘%.*s’.
```

Explanation

This message is returned when a user changes databases with the **use database** command.

5703 Error Message Text

```
Changed language seting to ‘%.*s’.
```

Explanation

This indicates a difference in language between the client and the server.

5704 Error Message Text

```
Changed client character setting to ‘%.*s’.
```

Explanation

This indicates a difference in character set between the client and the server.

Action

These are all informational messages and may be ignored. If you wish to suppress them, add the following line to your SERVER message callback function:

```
if (msgno==5701 || msgno==5703 || msgno==5704)
    return(0);
```

These messages cannot be suppressed on the server side, they must be handled on the client side.

Setting Output Line Length of *dbprrow*

Question

How do you set the output line length in the Open Client DB-Library function **dbprrow**?

Answer

In Open Client DB-Library pre-4.2 releases, you can specify line length for your **dbprrow** output. Use the global variable *DbColwidth* to set the desired length.

All global variables were removed as part of Open Client DB-Library release 4.2. Use the function **dbsetopt** with the option **DBPRLINELEN** to set the output line length as follows, substituting your output line length value for *column_width*:

```
dbsetopt(dbproc, DBPRLINELEN, "column_width", -1);
```

Remember to enclose the character string in quotes.

Additional Information

See the *SQL Server Reference Manual* entry for **dbsetopt** for more information.

Quotation Marks Using *dbsafestr*

Problem

Single and double quotation marks around SQL statement strings must be consistent. This can be confusing if you have strings that contain apostrophes or quotation marks.

Action

Suppose you are using DB-Library to build a table of quotations. You might try to use this **while** statement to insert the values:

```
while ((status = get_next_quo()) != NO_MORE_QUO)
{
   dbcmd(dbproc,"insert quotations values('%s')\n",
         my_quotation);
}
```

This statement works well for quotations that contain no apostrophes or quotation marks of their own. It does not work, however, for strings such as the following:

```
It's the water.
I'm Hennery the Eighth, I am.
```

The SQL parser interprets those apostrophes as the end of the string constant, and the rest of the quotation results in a SQL syntax error.

dbsafestr makes these strings safe for use in a SQL statement, by doubling any single or double quotation mark. For example, it changes one single quotation mark to two:

`'` becomes `''`

It also changes one double quotation mark to two:

`"` becomes `""`

The SQL parser then treats the quotation marks as non-syntactic marks, part of a string constant.

Both single and double quotation marks can be used in the same string using the **both** option.

The following code fragment uses **dbsafestr** to check and add any necessary quotation marks to character fields:

Figure 5-1: Sample code checks for quotation marks with dbsafestr

```
{
/*
**   CORPORATENAME
*/

dbsafestr(corporatename, buf2, 200);
dbfcmd(dbproc,
    "select corporateid from corporate
    where corporatename = '%s'",
    buf2);
}
```

Does *dbcancel* Automatically Send a Rollback Transaction Command?

Question

Does the DB-Library **dbcance**l command automatically issue a **rollback transaction** command?

Answer

No, **dbcancel** does not automatically send a **rollback transaction** command. A problem would occur if it did because the transaction may have completed several updates before a **select** is cancelled with **dbcancel**. You may not have wanted the changes rolled back but this would have happened.

You must send the **rollback transaction** explicitly.

DB-Library Memory Problem

Question

A DB-Library program logs into a SQL Server repeatedly and runs out of memory after a few iterations. Is **dbclose** or **dbexit** not freeing memory?

Answer

The memory use may be caused by LOGINREC structures, allocated by calls to **dblogin**, not being freed with calls to **dbloginfree**. If the login information is the same each time, one LOGINREC structure may be used and the call to **dblogin** placed outside the program's loop. If a login structure is no longer needed, it should be deallocated with **dbloginfree** after the call to **dbopen**.

For more information, refer to the *Open Client DB-Library Reference Manual* for a description of each routine.

How to Print Value After *dbbind* With Vartype *VARYCHARBIND*

Question

After doing **dbbind** with a vartype of *VARYCHARBIND*, why is the string value returned from SQL Server not shown when that variable is printed?

Answer

The variable is a structure containing the length and the string value with no NULL terminator. The value can be copied from that structure to a NULL terminated string and then printed as follows:

```
DBVARYCHAR titles;
char strtitle[DBMAXCHAR+1];

....

dbbind(dbproc, column, VARYCHARBIND,(DBINT)0,
      &titles);

....

while (dbnextrow(dbproc)!=NO_MORE_ROWS)
{
    strncpy(strtitle, titles.str, titles.len);
    strtitle[titles.len]='\0';
    printf("The title returned is %s\n",strtitle);
}
```

➤ ***Note***

Add a 1 to *DBMAXCHAR* (as shown in the second line of the above code fragment) to avoid overflow when titles.len=*DBMAXCHAR*.

Which C Datatypes Correspond to SQL Server Datatypes?

Question

When writing a DB-Library application, which datatypes correspond with SQL Server and C programs?

Answer

The datatypes are listed in the "types" section of the *Open Client DB-Library Reference Manual.*

The definitions are in the *sybfront.h* file in the include subdirectory of the standard SYBASE installation.

Programs should use the types as defined in *sybfront.h* and not their C definitions. This avoids problems if one or more declarations change in a later release.

What Is the DBPROCESS Structure?

Question

What does the DBPROCESS structure look like? How can members of that structure be manipulated?

Answer

The declaration of the DBPROCESS structure is included in the file *sybdb.h*. That structure contains all information about SQL Server connections. The members of the structure should only be manipulated through the macros supplied as part of DB-Library such as DBCOUNT and DBISAVAIL. Using these macros will ensure that the applications do not have problems if the DBPROCESS structure changes with a new release of DB-Library.

What Is the Difference Between *dbnextrow* and *dbresults*?

Question

In a DB-Library program it is necessary to include both **dbnextrow** and **dbresults** to process the results returned to the program from SQL Server. Why is this?

Answer

A DB-Library program stores SQL commands in a buffer with the DB-Library call **dbcmd**, then sends the buffer contents to SQL Server being accessed and waits for a response with the call **dbsqlexec** or the calls **dbsqlsend** and **dbsqlok**. The call **dbresults** causes DB-Library to read the results sent by the server. If **dbresults** sees that rows are available, it returns so that **dbnextrow** can read them. **dbnextrow** is then called to process the first row of data returned by the command accessed with **dbresults**. **dbnextrow** must be called once for each row returned by the **dbresults** call.

If more than one command was sent to SQL Server from the buffer, it is then necessary to call **dbresults** again to begin processing the results returned by the second command and **dbnextrow** must then be called again once for each row returned by this command, and so on. This is typically done within a nested loop in the DB-Library program.

Refer to the sample code on page 5-37 where the nested loop is illustrated.

DB-Library Functions for Setting a Program Variable to NULL

Question

When a DB-Library application returns data values and binds them to program variables, it is unable to bind a NULL which has no value. Instead, it binds a zero or blank of some form, depending on the datatype, to the program variable. How can these be distinguished from actual zero values or blanks?

Answer

The zero or blank is a default; you may use the **dbsetnull** function call to specify a user-defined NULL substitution value. It is also possible, with the Open Client 4.2 release and later, to determine when an attempt to bind a NULL occurred using the DB-Library function call **dbnullbind**.

For more information, refer to the *Open Client DB-Library Reference Manual* for details on these calls.

Utilities Available with PC Windows DB-Library/C

Question

Are **isql** and **bcp** utilities included with the Sybase PC Windows Open Client DB-Library/C software?

Answer

Neither **isql** nor **bcp** is available for pre-10.0 release PC Windows clients. For 10.0 and later releases, **wisql.exe** and **bcp** are both available. Some Open Client release bulletins contain an incorrect statement that such utilities exist for pre-10.0 releases.

DB-Library Ignores Interrupts

Question

Do DB-Library routines ignore interrupts at any time?

Answer

DB-Library ignores interrupts only when it is about to send an attention to SQL Server. DB-Library does check for interrupts when reading from the network. If an interrupt occurs, DB-Library just tries to read from the network again. The user can override this behavior by installing an interrupt handler with **dbsetinterrupt**.

Connecting a Macintosh DB-Library Application to OpenVMS SQL Server

Question

A Macintosh Open Client DB-Library application cannot connect to an OpenVMS SQL Server.

Answer

Connection problems are usually caused by one or more of the following items:

- An incorrect version of the network driver
- An incorrectly setup network driver
- Errors in the interfaces file

Follow these steps to resolve the problem:

1. Test the TSSNet NCP network driver to ensure that it is correctly set up. For example, use the NCP Net Copy utility to copy a file to the VAX host. If the copy succeeds, then your network setup is correct.
2. Make sure that your NCP version is 2.4 or later.
3. Check to see whether the interfaces file on your Macintosh has the correct port number for the OpenVMS SQL Server. You can display your Macintosh interfaces file with most Macintosh editors.

DBCOUNT Returns 0 or -1

Problem

The Open Client DB-Library macro **DBCOUNT** returns a 0 or -1 as the result of the execution of a stored procedure that performs an **insert**, **update**, or **delete** operation.

Explanation

DBCOUNT returns the number of rows returned by the latest **select** statement executed by a stored procedure. It does not count rows affected by any other SQL statements. If the stored procedure does not execute any **select** statements, **DBCOUNT** returns -1. However, if the stored procedure calls a procedure that does execute a select statement, **DBCOUNT** returns a meaningful value.

Action

Here are two alternative methods for finding the count of rows affected by an **insert, update**, or **delete** operation:

Select @@rowcount IMMEDIATELY after the modification, within the stored procedure!

1. Explicitly select the value of *@@rowcount* immediately after the data modification within the stored procedure (usually after **select**, **insert**, or **update** statement). This is the fastest option. If other people are modifying the same table, this option is also the most accurate. You can return the value of *@@rowcount* in one of three ways:
   - As an output parameter to the stored procedure (the preferred method),
   - As the return status (precluding its use for other status information), or
   - As a data row.
2. Perform a **select** using the same conditions as the data modification statements before performing the **insert**, **update**, or **delete** operation:

   ```
   select count(*) from table
         where conditions_used_for_data_modification
   ```

   ```
   update table
         where conditions_used_for_data_modification
   ```

   Make sure that this is the last select statement in the query. You will experience some loss of performance using this solution.

This solution is the only way to find out the number of rows to be returned by a **select**, **insert**, **update**, or **delete** operation before the operation is performed. It can do so only if no other processes are modifying the same table.

text or *image* Data Is Truncated

Problem

text or *image* data is truncated to 32K during reading or writing.

Action

This occurs because the default display size for *text* data in SQL Server is 32K. To see the current value, type:

```
select @@textsize
```

The text size is displayed as an integer value. To redefine the SQL Server text size, type:

```
set textsize integer_value
```

For example, to set the text size to 65536, type the following:

```
1> set textsize 65536
2> go
```

Also, Open Client DB-Library provides two options that affect *text* or *image* data size: DBTEXTSIZE and DBTEXTLIMIT. These options are set by the DB-Library function **dbsetopt.**

DBTEXTLIMIT

DBTEXTLIMIT causes DB-Library to limit the size of returned *text* or *image* values. To set this option, supply the length, in bytes, of the longest *text* or *image* value that your program can handle. For example, to set DBTEXTLIMIT to 64K, type:

```
dbsetopt(dbproc, DBTEXTLIMIT, "65536")
```

DB-Library reads but ignores any part of a *text* or *image* value that goes over this limit.

DB-Library's default behavior is to read and return all the data sent by SQL Server (for Windows, DOS, and OS/2 DB-Library, the default value is 4096). To restore this default behavior, set DBTEXTLIMIT to a value less than 1. With very large text values, it may take some time for the entire text value to be returned over the network. To prevent the Server from sending this extra text in the first place, use the DBTEXTSIZE option instead.

DBTEXTSIZE

DBTEXTSIZE changes the value of the SQL Server global variable *@@textsize*, that limits the size of *text* or *image* values that SQL Server returns. DBTEXTSIZE is equivalent to **set textsize**. *@@textsize* has a default value of 32,768 bytes (32K). To set this option, supply the length, in bytes, of the longest text or image value that SQL Server should return. For example, to set *@@textsize* to 64K, type:

```
dbsetopt(dbproc, DBTEXTSIZE, "65536")
```

In programs that allow application users to make ad hoc queries, a user can override this option with the Transact-SQL® **set textsize** command. To set a text limit that the user cannot override, use the DBTEXTLIMIT option instead. DBTEXTLIMIT limits how much *text* and *image* data is returned by DB-Library to the user, and affects how much data the SQL Server sends across the network to DB-Library

Remember that SQL Server stores *text* and *image* data on 2K data pages (4K for Stratus). When using either an **isql** command or **dbsetopt** to control the text size, increase or decrease the size in multiples of the page size (2K = 2048, 4K = 4096). This is because, with text data, SQL Server allocates one page at a time when needed.

Incompatibility Between Releases

Question

What releases of Open Client DB-Library are compatible with APT-Library™?

Explanation

When installing Open Client DB-Library and APT-Library, make sure that the releases are compatible. For example, in a mixed-mode application, a 4GL program such as APT-SQL might call a 3GL program such as a routine that contains some DB-Library calls. In some instances this causes link or run-time failures.

➤ ***Note***

When running APT 5.x on PC platforms, you must use a release 1.x Net-Library. Release 10.x Net-Library is not compatible with APT.

Action

Install separate releases of DB-Library, one for APT mixed-mode applications and one for DB-Library programs. This will work for example, if you have 4.0.2 APT and you want to use 4.6 Open Client in your DB-Library applications.

The following chart shows which releases are compatible.

Table 5-1: Open Client/APT-Library compatibility chart

APT-Library	Open Client DB-Library					
	4.0.1	**4.0.2**	**4.2**	**4.2.5**	**4.6**	**10.0**
4.0.1	Yes	No	Yes	No	No	No
4.0.2	No	Yes	No	Yes	No	No
4.0.3	No	Yes	No	Yes	No	No
5.0	No	No	No	No	Yes	

Raising Exceptions in ADA

Problem

In ADA programs, you can use ADBERRHANDLE and ADBMSGHANDLE to install error- and message-handling routines. Raising exceptions in these routines, however, can result in unpredictable ADA program behavior. This happens because when the user-installed routines are invoked, the call stack contains a mixture of ADA and C calls.

Explanation

Here is an example of how the call stack might look when a user-installed routine is invoked:

Figure 5-2: Sample Call Stack

Main ADA Code	(ADA)
ADBERRHANDLE	(ADA)
ADBMSGHANDLE	(ADA)
ADBSQLEXEC	(ADA)
dbsqlexec	(C)
SYBASE internal routine	(C)
User-installed routine	(ADA)

When the user-installed routine raises an exception, the exception is raised to the SYBASE C function, or related SYBASE internal routines, that was executing at the time of the error.

Action

Do not raise exceptions in the user-installed routines. Instead, set a global variable in the program such that upon return from the user-installed routines, the global variable can be inspected for additional action by the program.

The following program fragment avoids the problem:

```
If ADBSQLEXEC = FAIL then
    If GLOBAL_IS_SET = YES then raise exception;
    End If;
End If;
```

Undefined “_infinity” Symbol

Problem

When linking a C program in the UNIX environment, the symbol “_infinity” is reported as undefined and the link is unsuccessful.

Action

This error occurs because the math library *libm.a* is either missing or not where it should be. *libm.a* is located in */usr/lib*. You can add **-lm** to the end of the link statement if it is in */usr/lib*.

When linking in the UNIX VADS (Verdix ADA Development System) environment, also check the *ada.lib* file to be sure the math library is in the last declared WITH specification. Type the following example, where # is the last sequential number of WITH occurrences:

```
WITH#:LINK:/usr/lib/libm.a
```

Undefined Symbols During a Link

Problem

When linking a C program in the UNIX environment, the symbols "_db_os_sl", "_unsignstrncmp", and "_unsignstrcmp" are reported as undefined and the link is unsuccessful.

Action

This error occurs when Open Client DB-Library, APT-Execute™, APT Workbench™ and Data Workbench have been installed on a host machine in the wrong order. The correct order is:

1. Open Client DB-Library
2. APT-Execute and / or APT Workbench
3. Data Workbench

If you are experiencing these errors, reinstall your software in the correct order.

Compiling DB-Library Program on UNIX System V Platform

Problem

When attempting to compile DB-Library programs on UNIX System V platforms, various operating errors occurred as well as the message “unreferenced symbol” for all network calls. Why?

Action

When compiling DB-Library programs using either the makefile procedure supplied with the DB-Library sample programs or a system prompt command, it is standard to include a reference to the Sybase-supplied library of DB-Library calls (libsybdb.a). On UNIX System V platforms, it is also necessary to refer to an operating system network library: ‘libnsl.a’. It can be included to the makefile procedure by performing the following steps:

1. Modify the makefile procedure:

```
DBLIB1 = full_pathname libsybdb.a
DBLIB2 = -lnsl
DBLIBS = $(DBLIB1) $(DBLIB2)
```

Where *full_pathname* is the full pathname to the Sybase-supplied library of the DB-Library call.

This should replace the supplied definition:

```
DBLIBS = full_pathname libsybdb.a
```

➤ ***Note***

AT&T (NCR) System 3000 platforms are not affected by this modification.

Using *dbreadtext*

Problem

You need a good example of using the Open Client DB-Library function **dbreadtext** to read *text* and *image* data.

Action

The example program “Sample Code: example_dbrt” shows how to use **dbreadtext.**

Sample Output Using *dbreadtext*

The sample output below prints each 67-character buffer of data. When an entire row is read, it prints the string “End of Row!”

Figure 5-3: Sample execution with a text column

```
Database to use: test
Table name.....: ttext
Column name....: f2


Working, please wait ...


text1stringtext1stringtext1stringtext1stringtext1stringtext1stringt
ext1stringtext1stringtext1stringtext1stringtext1stringtext1stringte
xt1stringtext1stringtext1stringtext1stringtext1stringtext1stringtex
t1stringtext1stringtext1stringtext1stringtext1stringtext1string
End of Row!

text2stringtext2stringtext2stringtext2stringtext2stringtext2stringt
ext2stringtext2stringtext2stringtext2stringtext2stringtext2stringte
xt2stringtext2stringtext2stringtext2stringtext2stringtext2stringtex
t2stringtext2stringtext2stringtext2stringtext2stringtext2string
End of Row!

text3stringtext3stringtext3stringtext3stringtext3stringtext3stringt
ext3stringtext3stringtext3stringtext3stringtext3stringtext3stringte
xt3stringtext3stringtext3stringtext3stringtext3stringtext3stringtex
t3stringtext3stringtext3stringtext3stringtext3stringtext3string
End of Row!
```

Sample Execution with an *image* Column

The sample output below prints *image* data. Note that *image* text in general is unreadable. When an entire row is read, it prints the string “End of Row!”

Figure 5-4: Sample execution with an image column

```
Database to use: test
Table name.....: itext
Column name....: f2

^Q”3DUfw^H^Y
End of Row!

^?n]L;*
End of Row!

3333333333333333333333
End of Row!
```

Creating the Sample Program

The following steps show how to set up the environment so that the example runs correctly.

1. Create and populate *text* and *image* example tables. Insert several long rows of data into each table.

```
1> create table ttext (f1 int, f2 text)
2> go

1> insert ttext values (1,"text1....text1")
2> go
        .
        .
        .
1> insert ttext values (10,"text10...text10")
2> go

1> create table itext (f1 int, f2 image)
2> go

1> insert itext values (1,0xabcdef...12345)
2> go
        .
        .
        .
1> insert itext values (10,0x98765...43210)
2> go
```

2. Study the following program, **example_dbrt**. If you want to try it out, change the login and password information to a valid login on your SQL Server. The information is in these lines:

   ```
   DBSETLUSER(login, "username");
   DBSETLPWD(login, "password");
   ```

3. Compile and link using SYBASE include and library specifications.
4. Execute the program **example_dbrt**. The program prompts for the database to use, the table to select from and the *text/image* column to display. When an *image* column is displayed, the results vary according to the type of terminal.

Sample Code: *example_dbrt*

Figure 5-5: Sample program using dbreadtext

```
/*
**      example_dbrt.c 1.3 9/12/91 10:45:14
**
**      This program is an example of how to use the dbreadtext()
**      routine.
*/

#include <stdio.h>
#include <sybfront.h>
#include <sybdb.h>

#define BUFSIZE         67

/*
** Forward declarations of the error handler and message handler.
*/
int     err_handler();
int     msg_handler();

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

      LOGINREC        *login;
      DBPROCESS       *dbproc;

      /*
      ** Declare the local variables.
```

```
*/
char            buf[BUFSIZE + 1];
long            bytes;
long            rowcnt = 1;
char            ex_dbtouse[30];
char            ex_tabname[30];
char            ex_colname[30];

RETCODE         result_code; /* to hold the results of
                               dbresults(). */
STATUS          row_code;    /* to hold the results of
                               dbnextrow(). */


/*
** Get program parameters:
**      1) Database to use
**      2) Table name
**      3) Text/image column name
*/
printf("\n\n");                 /* 1 */
printf("Database to use: ");
scanf("%s", ex_dbtouse);

printf("\n");                   /* 2 */
printf("Table name.....: ");
scanf("%s", ex_tabname);

printf("\n");                   /* 3 */
printf("Column name....: ");
scanf("%s", ex_colname);

printf("\n\nWorking, please wait ...\n\n");


/*
** Initialize DB-Library.
*/
if (dbinit() == FAIL)
{
      exit(ERREXIT);
}

/*
** Install the user-supplied error-handling and message-
** handling routines. They are defined at the bottom of
** this source file.
*/
dberrhandle(err_handler);
```

```
 dbmsghandle(msg_handler);

 /*
 ** Set up the login information.
 */
 login = dblogin();

 DBSETLUSER(login, "username");
 DBSETLPWD(login, "password");
 DBSETLAPP(login, "example_dbrt");

 dbproc = dbopen(login, NULL);

 /*
 ** Send a "use database" command.
 */
 dbuse(dbproc, ex_dbtouse);

 /*
 ** Put the SQL statement into the command buffer.
 */
 dbfcmd(dbproc, "select %s from %s", ex_colname, ex_tabname);

 /*
 ** Send the command buffer to sql server for execution.
 */
 dbsqlexec(dbproc);

 /*
 **  Process the results:
 */
while( (result_code = dbresults(dbproc)) != NO_MORE_RESULTS )
 {
     if( result_code == FAIL )
     {
             /* dbresults() failed */
             printf("dbresults() failed.\n");
     }

     else
     {
         while( (bytes = dbreadtext(dbproc, (void *)buf,
                BUFSIZE)) != NO_MORE_ROWS )
         {
             if( bytes == -1 )
             {
                 /* dbreadtext() failed */
                 printf("dbreadtext() failed.\n");
```

```
                  }
                  else if( bytes == 0 )
                  {
                      /* We've reached the end of a row. */
                      printf("End of Row!\n\n");
                  }
                  else
                  {
                      /*
                      **  'bytes' bytes have been placed into our
                      ** buffer.
                      **  Print them:
                      */
                      buf[bytes] = '\0';
                      printf("%s\n", buf);
                  }
              }
          }
      }

      printf("\n\n");
      dbexit();
      exit(STDEXIT);
      }
}

int err_handler(dbproc, severity, dberr, oserr,
      dberrstr, oserrstr)
DBPROCESS       *dbproc;
int             severity;
int             dberr;
int             oserr;
char            *dberrstr;
char            *oserrstr;
{
      if ((dbproc == NULL) || (DBDEAD(dbproc)))
      {
            return(INT_EXIT);
      }
      else
      {
            printf("DB-Library error:\n\t%s\n", dberrstr);
      }
      if (oserr != DBNOERR)
      {
            printf("Operating-system error:\n\t%s\n", oserrstr);
      }
      return(INT_CANCEL);
}

int msg_handler(dbproc, msgno, msgstate, severity, msgtext,
            srvname, procname, line)

DBPROCESS       *dbproc;
DBINT           msgno;
int             msgstate;
int             severity;
char            *msgtext;
char            *srvname;
char            *procname;
DBUSMALLINT     line;

{
      /*
      ** Ignore 'changed database context' messages
      */
      if (msgno == 5701)
      {
            return(0);
      }

      printf ("Msg %ld, Level %d, State %d\n",
                msgno, severity, msgstate);

      if (strlen(srvname) > 0)
      {
            printf ("Server '%s', ", srvname);
      }
      if (strlen(procname) > 0)
      {
            printf ("Procedure '%s', ", procname);
      }
      if (line > 0)
      {
            printf ("Line %d", line);
      }

      printf("\n\t%s\n", msgtext);
}
```

Common Error Numbers and Messages

The remainder of this chapter gives explanations and actions you can take to remedy some errors commonly encountered when working with DB-Library.

Error 20009 (SYBECONN)

Severity Level 9 (EXCOMM)

Error Message Text

```
Unable to connect: SQL Server is unavailable or
does not exist.
```

Explanation

The **dbopen** has failed; the DBPROCESS is dead. There are several possible causes for this:

1. SQL Server is not running.
2. The Server name is not set correctly, whether it is set by the DSQUERY environment variable, by using the **isql -S** flag, or the *server* parameter in the **dbopen** function call.
3. The interfaces file has incorrect server name information.
4. $SYBASE does not point to the correct interfaces file.

Action

The UNIX commands shown below are for the C-shell (**csh**). If you are using a different shell, substitute commands appropriately.

Here are the solutions for the most common problems:

1. Use the **showserver** command to check that the SQL Server is running. If SQL Server is not running, restart it.
2. Invoke **printenv DSQUERY** to check the DSQUERY environment variable. To change it, type:

   ```
   setenv DSQUERY RIGHT_NAME
   ```

   where *RIGHT_NAME* is the correct name of your server. Check the spelling of the name if you are using the **-S** flag for **isql**. If the server name is being provided to the **dbopen** call, check to be sure that the variable is declared as **char ***.
3. If the information for this SQL Server is incorrect in the interfaces file, check the *Installation Guide* for your platform for information on how to change the interfaces file. It is highly recommended that you exercise caution and make a copy of the interfaces file before you change it.

4. Do **printenv SYBASE** to check the environment variable, and change it by using the command:

   ```
   setenv SYBASE /the/right/path
   ```

   where */the/right/path* is the path to your $SYBASE directory.

Error 20011 (SYBEDBPS)

Severity Level 8 (EXRESOURCE)

Error Message Text

```
Maximum number of dbprocesses already allocated.
```

Explanation

The default value for the maximum number of DBPROCESSes is 25. This error is generated when the number of simultaneously open DBPROCESSes exceeds the maximum limit.

Action

Check your current setting for *maxprocs* by calling **dbgetmaxprocs**. Then use the API call **dbsetmaxprocs** to increase the max limit. The syntax is:

```
RETCODE dbsetmaxprocs(maxprocs)
int maxprocs;
```

See your *Open Client DB-Library/C Reference Manual* for details.

Error 20017 (SYBESEOF)

Severity Level 9 (EXCOMM)

Error Message Text

```
Unexpected EOF from SQL Server.
```

Explanation

This error occurs when either the whole SQL Server process goes away or when the *spid* that handles this connection in the SQL Server dies. It will not be clear which of these has occurred when the error reaches the front end client.

Action

Perform the following steps to resolve the issue:

1. Try to reconnect to SQL Server. If the reconnect is successful, then it is the *spid* for the previous connection that has died. Try to understand what may have caused this connection to be broken and then take preventive measures.
2. If a Message 20009 occurs ("SQL Server is unavailable or does not exist"), then check to be sure that the SQL Server process is still running.
3. If **showserver** does not show the SQL Server process, check the SQL Server error log to see why the SQL Server process went away.
4. Restart SQL Server.

Error 20019 (SYBERPND)

Severity Level 7 (EXPROGRAM)

Error Message Text

```
Attempt to initiate a new SQL Server operation
with results pending.
```

Explanation

“Results pending” errors occur when you try to initiate a new SQL Server operation before the results of the previous operation have been processed completely.

This problem and its corresponding solution arise from the way in which 3GL programs process results. Usually programs that get this error do not process results and rows in a **while** loop. As illustrated in the program **example1.c** in the *DB-Library Reference Manual*, here is the proper way to process results:

```
while (dbresults(dbproc) != NO_MORE_RESULTS)
{
    .
    .
    .
    while (dbnextrow(dbproc) != NO_MORE_ROWS)
    {
            .
            .
            .
    }
}
```

This is the best method for processing results even if only one result is expected, because it is consistent. It makes the program generic in case future SQL Server releases change a SQL command’s behavior to return more than one result.

There are other, less common, reasons for error 20019. One has to do with mixed-mode programming. A 3GL program generates 20019 errors if it calls a form to do a “find by example,” retrieves partial results, and then calls another “find by example” form from a menu. If the forms in question are executed by APT-Execute, that is, entirely in 4GL, the error does not occur. It does occur in mixed-mode, however, because results are still pending on the first form when the second form tries to initiate a SQL Server operation.

Action

To avoid the 20019 error, the 3GL program must call **fsdbrec** after **fsdbopen,** so that another login to the Server is established.

This error can also occur if you initiate two SQL Server operations using the same DBPROCESS. A DBPROCESS can only handle one SQL Server operation at a time and cannot handle multiple requests. Here are three alternatives:

- Fill up the command buffer with the **dbcmd** / **dbfcmd** calls and process several sets of results.
- Open a new DBPROCESS for another SQL Server operation.
- Use **dbcanquery** to cancel the last set of results before initiating a new SQL Server operation. Calling **dbcanquery** is equivalent to calling **dbnextrow** until it returns NO_MORE_ROWS, but **dbcanquery** is faster because it allocates no memory and executes no bindings to user data.

Error 20023 (SYBEBTYP)

Severity Level 7 (EXPROGRAM)

Error Message Text

```
Unknown bind type passed to DB-Library function.
```

Explanation

This error occurs when the datatype for which the variable is bound is not an already-defined or existing datatype. It can occur while using the **dbconvert** and **dbsetnull** API function calls.

Action

Use only already-defined or existing datatypes. These datatypes are:

- *TINYBIND*
- *SMALLBIND*
- *INTBIND*
- *FLT8BIND*
- *REALBIND*
- *CHARBIND*
- *BINARYBIND*
- *STRINGBIND*
- *NTBSTRINGBIND*
- *VARYCHARBIND*
- *VARYBINBIND*
- *DATETIMEBIND*
- *SMALLDATETIMEBIND*
- *MONEYBIND*
- *SMALLMONEYBIND*

In System 10, these datatypes were added to the previous list:

- *NUMERICBIND*
- *DECIMALBIND*
- *SENSITIVITYBIND*
- *BOUNDARYBIND*

For details on the bind types and the program variable type, see your *Open Client DB-Library/C Reference Manual*.

Error 20032 (SYBEABNC)

Severity Level 7 (EXPROGRAM)

Error Message Text

```
Attempt to bind to a non-existent column.
```

Explanation

This error most often occurs when the DB-Library program attempts to bind to a column that is not in the select list of the previous **dbcmd** or **dbrpcsend** call. For instance, a situation might occur where you are trying to use a 4 as the second argument of **dbbind** when only 3 columns were selected in the previous **dbcmd** call.

Action

- Check the syntax of the DB-Library program to make sure that the **dbbind** calls do not refer to more column numbers than there are selected columns in the table.
- Check for network-related problems with your network administrator.

Error 20044 (SYBERDNR)

Severity Level 7 (EXPROGRAM)

Error Message Text

```
Attempt to retrieve data from a non-existent row.
```

Explanation

This error is returned by the following routines when there is no row to act upon:

- **dbqual**
- **dbtsput**
- **dbdata**
- **dbdatlen**
- **dbtxplen**
- **dbtxptr**
- **dbtxtimestamp**

This error occurs in the unusual situation where you think a **where** qualifier identified a row, but there was actually no row in the table meeting the qualification. For example, if a table has *image* / *text* data, the procedure is first to insert all the non-*image* columns into the table. Then an insert is done on this row for the *image* / *text* data; if the insert fails because the row cannot be found, then this error is raised.

Action

This is caused by a bad qualification of the row so that not a single matching row can be found. Check the row you actually want, and match the key fields for that row.

Error 20047 (SYBEDDNE)

Severity Level 1 (EXINFO)

Error Message Text

```
DBPROCESS is dead or not enabled.
```

Explanation

The DBPROCESS is no longer available for some reason. DB-Library marks a DBPROCESS dead when an error occurs that makes the DBPROCESS unusable in any future DB-Library call. There are several possible reasons for this:

- In processing commands issued by the client, SQL Server experienced a severe error (for example, Error 803 or Error 1205).
- While returning data to the client, the connection to the client was lost due to possible networking problems.
- The number of allowed user connections for the SQL Server was exceeded.

Action

1. Check the SQL Server error log for error messages corresponding to the time in which the 20047 error was raised.
2. Check for networking problems with your network administrator.
3. If the number of allowed connections is the problem, the number can be increased using **sp_configure**. Check the memory availability on your SQL Server, because more connections means increased use of SQL Server memory.

Error 20050 (SYBECSYN)

Severity Level 4 (EXCONVERSION)

Error Message Text

```
Attempt to convert data stopped by syntax error in
source field.
```

Explanation

Two possible reasons exist for this message:

- The wrong **dbbind** statement is used for a column.
- The input file contains bad data.

Action

Check the **dbbind** statements to make sure that you are using the correct **dbbind** for a column. See the *Open Client DB-Library Reference Manual* for more details.

Check that the input file contains good data. For example, **dbconvert** will fail if the data is corrupted. Check for a return value (-1 or FAIL) Put a check for a return value (-1 or FAIL) in your code, and the direction to continue processing without exiting, and the corrupted data will not be processed.

Error 20180 (SYBETRAN)

Severity Level 1 (EXINFO)

Error Message Text

```
DBPROCESS is being used for another transaction.
```

Explanation

“Transaction” is misleading in this message. It is not a SQL transaction (that is, **begin**, **commit**, **rollback**). In this message, “transaction” refers to an RPC operation, a registered procedure operation, or a passthru operation. For example, the following statement would generate this message since the first RPC was not processed to completion:

```
dbrpcinit(dbproc,”rpcA”,...);
dbrpcinit(dbproc,”rpcB”,...);
```

Action

If you do not want to continue with the current transaction for this DBPROCESS, issue a **dbcancel** call for the DBPROCESS.

If you do want to continue processing the current transaction, then depending on the Open Client commands that have already been executed, the next logical command should be executed.

Error 20188 (SYBETEXS)

Severity Level 1 (EXINFO)

Error Message Text

```
Called dbmoretext with a bad size parameter.
```

Explanation

dbmoretext is used in conjunction with **dbwritetext** to send large image or text data in smaller blocks to SQL Server. This error occurs if the size is set to a value less than zero.

Action

Set the size value to greater than zero.

Error 5702

Severity Level 1 (EXINFO)

Error Message Text

```
The SQL Server is terminating this process.
```

Explanation

This is an informational message, and only appears in conjunction with other errors, for example, 813, 8211, crashes, stack traces, core dumps, and so on. All the errors it accompanies are fatal; 5702 itself is merely an advisory that the SQL Server is terminating the client process. Its severity level on the server side (which you can see by doing **select severity from sysmessages where error=5702**) is 10. This is unexpectedly high for an informational message and indicates the severity of the accompanying errors. It also warns you that the DBPROCESS is dead.

Action

Check for and resolve other errors are raised in addition to this one. Check the error log if they have not been displayed on your screen. You may need the assistance of your SYBASE System Administrator.

6 Open Client Client-Library

This section covers some of the questions and problems that can occur when using Open Client Client-Library.

Using *ct_param* to Pass Values for Host Variables

The *Open Client Client-Library/C Reference Manual* (release 10.0) explains that applications may need to call **ct_param** to define host variable formats for cursor **declare** commands. It further states that you must define host variable formats for **declare** commands when the body of the cursor being declared is a SQL string containing host variables.

A sample call to **ct_cursor** containing host variables looks like this:

```
ct_cursor(cmd, CS_CURSOR_DECLARE, “cursname”,
CS_NULLTERM, “select * from table where column1 >
@var1 or column2 < @var2”, CS_NULLTERM, CS_UNUSED);
```

Following this declaration, you would use **ct_param** to describe the values which you are going to provide for the host variables *@var1* and *@var2*.

The **datafmt* structure passes data to the SQL Server describing the types it can expect for *var1* and *var2* at cursor open:

```
...
dfmt.status = CS_INPUTVALUE;
dfmt.type = ...;
ct_param(cmd, &dfmt, NULL, CS_UNUSED, 0);
```

As a result, you should pass a NULL pointer for the **buffer** argument to the **ct_param** call.

Troubleshooting User-Defined Signals

Problem

A problem can arise when a user signal handler is installed which does not have a function associated with it, for example:

```
sigact.sa_handler=SIG_IGN;
sigaction (SIGIO,&sigact,NULL);
```

The macro SIG_IGN (which has a value of 1) specifies the address of the signal handler, but, when it is run, the program fails with a bus error message.

Explanation

If this user-defined handler is installed before any Embedded SQL statements or Open Client function calls are executed, it can co-exist with the SYBASE-defined handler which monitors both SIGIO and SIGPOLL. The SYBASE-defined handler, on installation, checks to see if another handler has already been installed. If so, the SYBASE-defined handler performs its own operations when called, and then invokes execution of the user-defined handler.

However, if a user-defined handler is installed after the first embedded statement or function calls, it may actually replace or overlay the SYBASE handler. If this new handler does not understand TDS packets or proper protocol behavior, here is what may happen:

1. The SYBASE handler is invoked for the connect statement, and it does its own processing;
2. The SYBASE handler tries to invoke the function whose address was specified in the **sigaction** call;
3. SIG_IGN's value of 1 is not a valid address, and the program fails.

Action

There are three ways to deal with this situation:

1. Upgrade to release 10.0.1, because in that release the code has been changed to recognize SIG_IGN explicitly, and no effort is made to execute this function whose address is 1.
2. Replace SIG_IGN macro with a fully coded signal handler before the first Embedded SQL statement. As SIG_IGN is a "no op", it may be inferred that the programmer intended to install the

"real" handler later anyway. Do the intended replacement, but **before** the first Embedded SQL statement, or the program will be unsuccessful.

3. Use a call to the Client-Library routine **ct_callback** to install a signal handler that Client-Library will invoke, for example:

   ```
   ct_callback(context, conn, CS_SET,
   CS_SIGNAL_CB+SIGIO, some_real_function)
   ```

Message Handling in Client-Library

Question

In the “Error and Message Handling” section of the *Open Client Client-Library/C Reference Manual*, the user is warned not to confuse error messages and message result sets. When or how can an application receive a message result set?

Explanation

In Client-Library, the word “message” has two meanings:

- SQL Server messages: Character strings returned by SQL Server describing an error or providing some kind of information. When Client-Library receives a SQL Server message, it invokes your message callback handler.
- Message streams: A new, System 10 way to communicate between client and server. It allows you to define your own “protocol” on top of TDS. This involves a message number (some arbitrary number) and zero or more parameters. A client can both send and receive such “message streams.”

If you look at **ct_results()**, you will see that one possible result type is *CS_MSG_RESULT*. This is a message result set, which should not be confused with a server message.

The NULL *varchar* and Host Variables

Question

The following statement, issued from within Client-Library through a language command, results in an empty string stored as one blank:

```
insert into table (varchar_col) values ("")
```

However, a second statement, issued through a language command, which uses a host variable and supplies an empty string through **ct_param**, results in the empty string being stored as NULL:

```
insert into table values (@buffer)
```

Why is there a difference?

Explanation

In Client-Library, as in other Sybase software, the empty string (" ") is not the same as NULL in a *varchar* column. SQL Server uses a single space to record the presence of the empty string in a varchar column, as documented in the *SQL Server Reference Manual* under "Datatypes."

However, TDS cannot represent a non-null string of length 0 when sending data to the SQL Server as a character parameter. The 0 in the length field of the variable length character TDS-stream **means** NULL. This is documented in the "ct_param" section of the *Open Client Client-Library/C Reference Manual*. The difference is that when Client-Library sends a language command, there **is** a way to represent the 0-length, non-null string.

7 Open Server Issues

This chapter gives explanations of some of the errors that can occur when you build or use Open Server™ applications. It also provides an example of an Open Server application that could be used to audit SQL requests.

➤ ***Note***

Some Open Server messages that relate to the Backup Server™ are documented in the *SQL Server Troubleshooting Guide*.

Do Open Server Applications Time-Out Their Connections?

Question

Connections between SQL Server and a remote server may be closed by SQL Server, depending on the timeout setting. Is there something similar for an Open Server application?

Answer

No, you cannot set an automatic timeout for the connection between an Open Server application and another server. The connection remains open until one of two events occurs:

- The other server closes the connection, or
- The Open Server application stops.

Thread Control Structure

Problem

How do you keep track of thread control structures to be able to handle attentions for your client connections?

Explanation

In Open Server 1.0 you could use a static array of your thread control structures, and index into the array using the thread control structure ID. With Open Server 2.0 and later, however, the procedure IDs are no longer sequential, so it is difficult to determine how large the array should be.

Action

There are two approaches:

- Estimate the maximum number of connections for your application and multiply that number by four. Declare a static array using that number of elements. This solution should create more than enough space for your thread control structures.
- Use other storage methods, such as a dynamically linked list or a hash table.

SRV_GETSRVPROC

Problem

SRV_GETSRVPROC results in an error.

Explanation

SRV_GETSRVPROC was removed as of Open Server release 2.0 because releases 2.0 and above are multi-threaded.

In a multi-threaded program, threads, or processes, execute one at a time and periodically yield so that other threads can execute. For example, suppose a Server thread sends 1000 data rows to a client. Every 100 rows it yields and lets another thread run so that all the other clients do not have to wait until all 1000 rows are delivered.

In SQL Server, the thread is a process executing a section of SQL Server code. The code a thread executes must be careful not to change things in the environment that can affect other threads. Usually that means that a routine must make sure that no other thread will be destroyed if it alters a piece of shared memory.

Action

In release 10.0, use **srv_props**. This returns the thread control structure of the current thread. (Release 2.0 used **SVR_CURPROC**.)

Protection of Registered Procedures

Problem

You want to restrict the use of certain registered procedures to the "sa" user.

Action

The use of registered procedures in an Open Server can be restricted by installing a callback handler. This ensures that the registered procedures are not executed by any Open Server client and at the same time will also protect the system registered procedures from being dropped. To do this, your Open Server code must include a call back handler. Then, by using appropriate Server Library calls, the handler logic can determine whether to continue or abort.

Following is the code for a sample callback handler that is installed in the **start_handler** function of the Open Server and then passes the control to a function called **callback_handler.** The **callback_handler** function uses the **srv_rpcname** to determine the name of the client's RPC to execute. Then the handler gets the client's user name by using the **srv_pfield** function. This callback handler is written to allow clients logged in as "sa" to execute the system registered procedures **sp_regcreate**, **sp_regdrop**, and **sp_terminate.** It aborts all other client connections if they try to execute to any of the above procedures. It allows clients to execute any other registered procedures. This can be done either by returning a **CS_FAIL** (to abort connections) or a **CS_SUCCEED** (continue executing).

➤ ***Note***

This example was written for 2.0 Open Server; some of the routines in it are no longer supported by 10.0 Open Server. For example, **SRV_PROC**, **SRV_SERVER**, and **SRV_CONFIG** are now hidden structures which may not be directly manipulated. **SRV_CONTINUE** has been replaced by **CS_SUCCEED**,. **SRV_S_INHIBIT** has been replaced by **CS_FAIL**.

Figure 7-1: Sample callback handler code

```
      .....
#define USER_UNKNOWN SRV_MAXERROR+7
      ....
      ....
main()
     { .....
     /* START HANDLER FOR THE OPEN SERVER */
     srv_handle(server, SRV_START, start_handler);
     ....
     ....
     }
RETCODE start_handler(server)
SRV_SERVER *server;
      {
      /*
       * Install the callback handler. This is done within the
       * START_HANDLER so that whenever the Open Server is
       * started, the callback handler is installed.
       */
      srv_callback(srvproc,SRV_C_PROCEXEC,callback_handler);
      .....
      ....
      }


RETCODE callback_handler(srvproc)
SRV_PROC *srvproc;
     {
     DBCHAR *rpcname;
     DBCHAR *username;

    /* Figure out the RPC name*/
    rpcname = srv_rpcname(srvproc,NULL);
    /* Figure out the user name*/
    username = srv_pfield(srvproc,SRV_USER,NULL);

    /*
    * Verify that the user logged in is not “sa”. If user is not,
    * verify the registered procedure name. If the procedure is
    * sp_regcreate, sp_regdrop or sp_terminate, send message to
    * client and abort. Otherwise, allow a user to execute
    * the proc. If the user is logged in as “sa”, continue.
    * /

   if ( strncmp(username,”sa”,2) != 0)
         {
```

Figure 7-1: Sample callback handler code (continued)

```
if ( (strncmp(rpcname,”sp_regcreate”,12) == 0) ||
            (strncmp(rpcname,”sp_regdrop”,10) == 0) ||
            (strncmp(rpcname,”sp_terminate”,12)== 0) )
         {
           SRV__SPRINTF(Msg, “Only the SA user can execute the \
                 procedure%s and you are currenlty logged in as\
                 %s”,rpcname,username);
           srv_sendmsg(srvproc, SRV_MSG_ERROR, USER_UNKNOWN,
                 SRV_INFO, (DBTINYINT) 0, (DBCHAR *)0, 0, 0, Msg,
                 SRV_NULLTERM);
           srv_senddone(srvproc, SRV_DONE_ERROR | SRV_DONE_FINAL,
               (DBUSMALLINT) 0, (DBINT) 0);
            return (SRV_S_INHIBIT);
       }
       else {
            return (SRV_S_CONTINUE);
           }
       }
       else {
       return(SRV_S_CONTINUE);
       }
    }
```

MUTEX and Resource Protection

Problem

What is a MUTEX and how is it used?

Explanation

Mutual Exclusion Semaphores (MUTEXes) are used in any multi-threaded application, like Open Server-based applications, to protect global data and to avert collisions between threads. MUTEX is a logical object created by a unique name associated with some shared resources, which can be accessed by one or more user threads.

MUTEXes can resolve many resource collision issues among multiple user threads in an Open Server. Any thread can lock a MUTEX and gain exclusive use or control of a global resource. This ensures protection of the resource from other threads accidentally changing the values, and other collision points. Other threads that need that particular resource can be forced to sleep until the MUTEX is unlocked.

For example, a typical Open Server would attempt to write to *stdout* and when multiple clients execute, threads could collide with one another when writing to *stdout*. To avoid this problem, code the Open Server so that, before writing to the *stdout*, the thread attempts to a lock a MUTEX associated with the resource. This ensures that only one client can write to the resource. Other threads waiting for this resource would sleep until the MUTEX is available. Server-Library provides a number of API calls that allow a user application to create, lock, and delete MUTEXes:

- **srv_createmutex**

  This call creates a MUTEX.

- **srv_lockmutex**

  This call is used to lock a MUTEX and the resource associated with it. It also returns the following unique return codes to tell the thread attempting to lock the MUTEX about the status of their attempt:

  SRV_I_SYNC – The thread got the lock immediately (synchronously).

  SRV_i_GRANTED – The thread got the lock after waiting.

  SRV_I_INTERRUPTED – The thread was interrupted while waiting for another thread to release this lock.

- **srv_unlockmutex**

  This API allows you to unlock a MUTEX.

  An associated API that is useful is the call **srv_getobjid**, which allows you to get the ID of a MUTEX from a name. The ID is required to perform both the locking and unlocking of a MUTEX. A thread can acquire the same MUTEX multiple times, if it has already been granted to it. The important thing is that the same thread must call **srv_unlockmutex** the same number of times that it called **srv_lockmutex**. Until the unlocking is complete, no other thread will be able to acquire that MUTEX.

  See the *Open Server Reference Manual*, for reference pages on specific commands. It also contains a “Multithread Programming” topic section.

Notification from an Open Server

A registered procedure is a special type of procedure available in Open Server 2.0 and later. The Open Server run-time system has a list of registered procedures created and registered either by the Open Server or by client programs using DB-Library or Client-Library API calls. Usually clients can request notification after a particular registered procedure executes. In this way, multiple client programs can communicate with one another via the Open Server. The Open Client libraries have a variety of APIs that serve this purpose.

Following is a DB-Library program that receives a notification from an Open Server when a specific registered procedure is executed. The program first connects to an Open Server, and then waits for a notice anytime an RPC called "*select1*" is executed. The main DB-Library function calls used in this code fragment include:

- **dbreghandle**

  Tells the program which handler function should be executed when a notification comes through.

- **dbregwatch**

  Tells the program which registered procedure to watch for. This API can watch both synchronously and asynchronously, and can instruct the Open Server to get one or multiple notifications.

- **dbpoll**

  Polls the network to see if there is something to be read for this specific client.

Figure 7-2: Sample program using notification APIs

```
/************* Sample *****************/

#include <stdio.h>
#include <sybfront.h>
#include <sybdb.h>

/*

/** Forward declarations of the error handler and message
handler routines. */
int   err_handler( );
int   msg_handler( );
int   myhandle( );
```

Figure 7-2: Sample program using notification APIs (continued)

```
main()
{
      LOGINREC  *login;
      DBPROCESS *dbproc;
      DBINT     ret;
      RETCODE   return_code;

      if (dbinit() == FAIL)
          exit(ERREXIT);

      dberrhandle(err_handler);
      dbmsghandle(msg_handler);

      login = dblogin( );
      DBSETLUSER(login, “sa”);
      DBSETLPWD(login, ““);

      /* Open a connection to the Open Server*/

      dbproc = dbopen(login, “OPENSERVER”);

      /* Identify which is the handler function to be called when
            a notification arrives */

      dbreghandle(dbproc,”select1”,DBNULLTERM,myhandle);
       /*
      ** Identify the procedure to watch. The fourth parameter
      ** can also be varied to do the following:
      ** DBNOWAITALL -> Return immediately, used primarily to get
      **          asynch notifications.
      ** DBWAIT -> Wait till a notification arrives. Used to
      **          get synch notifications
      */
      ret = dbregwatch(dbproc,”select1”,DBNULLTERM,DBNOWAITALL);
      if (ret == FAIL)
            {
            printf(“ DBREGWATCH Failed \n”);
            dbexit();
            exit(ERREXIT);
            }
      else if(ret == DBNOPROC)
            {
            printf(“ The Reg Proc doesn't Exist \n”);
            dbexit();
            exit(ERREXIT);
            }
```

Figure 7-2: Sample program using notification APIs (continued)

```
     else {

     /* Start the polling loop as we have requested asynch
     ** notification in the DBREGWATCH call*/

          while (1)
                {
                dbpoll(NULL,1000,NULL,&ret);
                if (ret == DBNOTIFICATION)
                    {
                    printf(“Procedure executed \n”);
                    }
                }

           dbexit();
           exit(STDEXIT);
           }
}


/* This function installed as a handler for the registered proc;
** this proc attempts to email a user when the notification
** arrives.*/
int myhandle(dbproc,proc_name,res1,res2)
DBPROCESS *dbproc;
DBCHAR *proc_name;
DBUSMALLINT res1,res2;
{
     FILE        *mailfp;
     DBCHAR      mailcmd[256];
     DBCHAR      mailto[20];
     DBCHAR      mailfrom[20];
     FILE        *popen();

      strncpy(mailto,”user”,4);
      strncpy(mailfrom,”watcher”,7);
      sprintf(mailcmd, “/usr/ucb/mail %s”, mailto);
      if ((mailfp = popen(mailcmd, “w”)) == (FILE *) NULL)
      {
           fprintf(stderr, “popen failed \n.”);
           return (INT_CONTINUE);
      }

      fprintf(mailfp, “~s “);
      strcpy(mailcmd,”SOMEONE EXECUTED SELECT1\n”);
      fputs(mailcmd,mailfp);
```

Figure 7-2: Sample program using notification APIs (continued)

```
/*
     ** Now sign it.
     */
     fprintf(mailfp, “\t\t\t-- %s\n”, mailfrom);
     fprintf(mailfp, “\t** Mail sent via Open Client **\n”);
     pclose(mailfp);
     return(INT_CONTINUE);
}

int err_handler(dbproc, severity, dberr, oserr, dberrstr,
oserrstr)
DBPROCESS  *dbproc;
int        severity;
int        dberr;
int        oserr;
char       *dberrstr;
char       *oserrstr;
{
       if ((dbproc == NULL) || (DBDEAD(dbproc)))
             return(INT_EXIT);
       else
       {
         fprintf (stderr, “DB-Library error:\n\t%s\n”, dberrstr);

          if (oserr != DBNOERR)
              fprintf (stderr, “Operating-system error:\n\t%s\n”,
                       oserrstr);
          return(INT_CANCEL);
       }
}

int msg_handler(dbproc, msgno, msgstate, severity, msgtext,
         srvname, procname, line)

DBPROCESS   *dbproc;
DBINT       msgno;
int         msgstate;
int         severity;
char        *msgtext;
char        *srvname;
char        *procname;
DBUSMALLINT line;
```

Figure 7-2: Sample program using notification APIs (continued)

```
{

     fprintf (stderr, “Msg %ld, Level %d, State %d\n”,
      msgno, severity, msgstate);

     if (strlen(srvname) > 0)


           fprintf (stderr, “Server '%s', “, srvname);
     if (strlen(procname) > 0)
           fprintf (stderr, “Procedure '%s', “, procname);
     if (line > 0)
           fprintf (stderr, “Line %d”, line);

     fprintf (stderr, “\n\t%s\n”, msgtext);

     return(0);
}
```

Number of Open Server Connections

Question:

How many Open Server connections canI set in release 10.0?

Answer:

In the initial 10.0 release, you are limited by operating system resources. For example, Sun Solaris 1.x (SunOS release 4.x) restricts SYBASE to 256 file descriptors, unless the kernel has been modified.

In release 10.0.1 and later, there is no connection limit.

The 2.0 Open Server release had a SYBASE imposed limit of 256 connections, but as of release 2.0.2, this restriction has been removed.

Compatibility of Open Server, DB-Library, and Client-Library

Question:

Can I build 10.0 Open Server applications with 10.0 DB-Library or do I need to use 10.0 Client-Library?

Answer:

10.0 Open Server works with both 10.0 DB-Library and 10.0 Client-Library.

Open Server “Half Passthrough” Auditing Gateway

You may want to create a “half passthrough” Open Server application, which can audit SQL requests, for one or more of the following reasons:

- To audit requests on the SQL Server
- To search for a word in a request, for example, “create”, and refuse the request if it contains this word
- To run a timing cost on the query and refuse it if the cost is too high
- To log requests

The following code gives an example of how to create a “half passthrough” Open Server.

➤ ***Note***

This code is untested and is meant to be used only as a general guideline.

Figure 7-3: Sample code for creation of “half passthrough” Open Server

```
#include<stdio.h>
#include<string.h>
#include<strings.h>
#include<signal.h>
#include<sybfront.h>
#include<sybdb.h>
#includeospublic.h>
#include<ossample.h>
#include <stdlib.h>
#include <ctpublic.h>
#define SERVERNAME “FTTOR1”

#if (USE_SCCSID)
static charSccsid[] = {“@(#) lang.c 1.9 8/5/93”};
#endif /* (USE_SCCSID) */

CS_RETCODE ex_clientmsg_cb();
CS_RETCODE ex_servermsg_cb();

/*
** Prototypes.
*/
```

```
CS_RETCODE  pass_lang_handler PROTOTYPE ((SRV_PROC
*sp
    ));
CS_RETCODE  start_handler PROTOTYPE ((
    SRV_SERVER*server
    ));
CS_RETCODE  connect_handler PROTOTYPE ((
    SRV_PROC*sp
    ));
CS_RETCODE  disconnect_handler PROTOTYPE ((
    SRV_PROC*sp
    ));
CS_RETCODE  Slang_handler PROTOTYPE ((
    SRV_PROC*sp
    ));
CS_VOID done_error PROTOTYPE ((
    SRV_PROC*sp
    ));

int main( argc, argv )
CS_INTargc;
CS_CHAR**argv;
{
    CS_CONTEXT*cp; /* Context structure. */
    SRV_SERVER*server;/* Server process itself. */
    CS_CHAR logname[CS_MAX_CHAR];/* Log file name*/
    CS_CHAR sname[CS_MAX_CHAR];/* Server name. */
    /*
    ** Get a context.
    */
    if( cs_ctx_global(CS_VERSION_100, &cp) ==
CS_FAIL )
    {
        (CS_VOID)fprintf(stderr,
        “ERROR: cs_ctx_global() failed!\n”);

        cs_ctx_drop(cp);
        exit(1);
    }


    /*
    ** Install a CS-Library error handler.
    */
    if( cs_config(cp, CS_SET, CS_MESSAGE_CB,
        (CS_VOID *)cs_err_handler,
        CS_UNUSED, (CS_INT *)NULL) == CS_FAIL )
    {
        (CS_VOID)fprintf(stderr,
            “ERROR: cs_config(CS_MESSAGE_CB)
                failed!\n”);

        cs_ctx_drop(cp);
        exit(1);
    }

    /*
    ** Initialize Open Server.
    */
    if( srv_version(cp, CS_VERSION_100) == CS_FAIL
)
    {
        (CS_VOID)fprintf(stderr, “ERROR:
            srv_version() failed!\n”);

        cs_ctx_drop(cp);
        exit(1);
    }

    /*
    ** Process the command-line arguments.
    */
    if( proc_args(cp, argc, argv, sname,
CS_MAX_CHAR, (CS_CHAR *)NULL,
        (CS_INT)0) == CS_FAIL )
    {
        /*
        ** An error was already raised.
        */
        cs_ctx_drop(cp);
        exit(1);
    }

    /*
    ** Install the Open Server error handler.
    */
    if( srv_props(cp, CS_SET, SRV_S_ERRHANDLE,
        (CS_VOID *)server_err_handler,
sizeof(CS_VOID *),
        (CS_INT *)NULL) == CS_FAIL )
    {
        /*
        ** An error was already raised.
        */
        cs_ctx_drop(cp);
```

```
            exit(1);>}>
        /*
        ** Define the log file.
        */> if( sname[0] != (CS_CHAR)’\0’ )>{
            (CS_VOID)strcpy(logname, sname);
            (CS_VOID)strcat(logname, “.log”);
        }
        else
        {
            (CS_VOID)strcpy(logname, “os.log”);
        }

        if( srv_props(cp, CS_SET, SRV_S_LOGFILE,
    logname,
            CS_NULLTERM, NULL) == CS_FAIL )
        {
            /*
            ** An error was already raised.
            */
            cs_ctx_drop(cp);
            exit(1);
        }

        /*
        ** Initialize Open Server.
        */
        if( (server = srv_init((SRV_CONFIG *)NULL,
    sname,
                CS_NULLTERM)) == (SRV_SERVER *)NULL )
        {
            /*
            ** An error was already raised.
            */
            cs_ctx_drop(cp);
            exit(1);
        }


        /*
        ** Install the event handlers.
        */
        srv_handle(server, SRV_DISCONNECT,
    disconnect_handler);
        srv_handle(server, SRV_START, start_handler);
        srv_handle(server, SRV_LANGUAGE, lang_handler);
        srv_handle(server, SRV_CONNECT,
    connect_handler);
        /*
        ** Print a version message.
        */
        print_version(cp, sname);

        /*
        ** Start up Open Server.
        */
        if( srv_run((SRV_SERVER *)NULL) == CS_FAIL )
        {
            fprintf(stderr,”Failed srv_run\n”);
            /*
            ** An error was already raised.
            */
            cs_ctx_drop(cp);
            exit(1);
        }

        /*
        ** Clean up and exit.
        */
        cs_ctx_drop(cp);
        exit(0);
    }

    CS_RETCODE start_handler(server)
    SRV_SERVER    *server;
    {
            CS_INT      info;
            SRV_PROC    *sp;
    /*
            ** Create a temporary SRV_PROC to register
            ** the procedure.
            */
            sp = srv_createproc(server);
            if ( sp == (SRV_PROC *)NULL )
            {
                return(CS_FAIL);
            }
    }
    CS_RETCODEdisconnect_handler(sp)
    {
        CS_RETCODE retcode;
        CS_CONNECTION   *Cn;
        CS_CONTEXT   *Cx;
        CS_INT  size;
```

```
        /*
        ** Retrieve the Connection handler stored
        ** in the client thread’s user data space.
        */
        if ( srv_thread_props(sp, CS_GET,
SRV_T_USERDATA, &Cn,sizeof(CS_CONNECTION *),
&size) == CS_FAIL)
    {
        fprintf(stderr, “srv_thread_props
failed\n”);
        return(CS_FAIL);
    }
    if (ct_con_props(Cn, CS_GET, CS_PARENT_HANDLE,
&Cx,
        CS_UNUSED, &size) == CS_FAIL )
    {
        fprintf(stderr, “ct_con_props failed\n”);
        return(CS_FAIL);
    }
    ex_con_cleanup(Cn, retcode);
    ex_ctx_cleanup(Cx, retcode);
}

/*
** lang_handler
** This routine is the SRV_LANGUAGE event handler.
** All we do here is get the incoming language
** string, and return it to the client via an
** informational message.
*/
CS_RETCODElang_handler(sp)
SRV_PROC*sp;
{
    char    msg[1024];
    CS_CHAR *CmdText;
    CS_INT  len; /* the length of the message. */
    CS_RETCODEretcode;
    CS_INT  cost;
    CS_BOOL dead;

    /*
    ** Initialization.
    */
    srv_bzero(&msg, sizeof(msg));
    /*
    ** Get the length of the language string.
    */
    if( (len = srv_langlen(sp)) == -1 )
    {
        done_error(sp);
        return CS_FAIL;
    }

    /*
    ** Allocate enough space to hold the language
    ** string.
    */
    if( (CmdText = (CS_CHAR *)malloc(len+1)) ==
(CS_CHAR *)NULL )
    {
        done_error(sp);
        return CS_FAIL;
    }

    /*
    ** Get the language string itself.
    */
    strncpy(CmdText,srv_langptr(sp,0),len);
    CmdText[len] = ‘\0’;

    log_function(sp,CmdText);
    half_passthru(sp,CmdText);

    free(CmdText);

    /*
    ** All done.
    */
    return CS_SUCCEED;
}

int half_passthru(sp,CmdText)
SRV_PROC *sp;
char    *CmdText;
{
    CS_CONNECTION*Cn;
    CS_COMMAND*Cmd;
    CS_RETCODEretcode;

    if ( srv_thread_props(sp, CS_GET,
SRV_T_USERDATA, &Cn,
        sizeof(CS_CONNECTION *), (CS_INT *)NULL)
== CS_FAIL )
    {
        fprintf(stderr, “srv_thread_props
error\n”);
        done_error(sp);
        return(CS_FAIL);
    }
    retcode = ct_cmd_alloc(Cn,&Cmd);
    if (retcode != CS_SUCCEED)
    {
        fprintf(stderr,”Failed To allocate command
structure\n”);
        done_error(sp);
        return CS_FAIL;
    }

    retcode = ct_command
(Cmd,CS_LANG_CMD,CmdText,CS_NULLTERM,CS_UNUSED);
    if (retcode != CS_SUCCEED)
    {
        fprintf(stderr, “Language Handler: Failed
to store command. Failing...\n”);
        done_error(sp);
        return CS_FAIL;
    }
    retcode = ct_send (Cmd);
    if (retcode != CS_SUCCEED)
    {
        fprintf(stderr, “Language Handler: Failed
to send command. Failing...\n”);
        done_error(sp);
        return CS_FAIL;
    }
    pass_lang_handler(sp,Cmd);
    retcode = ct_cmd_drop(Cmd);
    if (retcode != CS_SUCCEED)
    {
        fprintf(stderr,”Failed To deallocate
command structure\n”);
        return CS_FAIL;
    }
}
```

```
/*
**  done_error
**  This routine is used to send a done-error
**  message to client threads when some client-
**  originated login or command has failed.
*/

CS_VOID done_error(sp)
SRV_PROC*sp;
{
    /*
    ** All we need to do is send the done. If this
    ** fails, print error to the screen and return.
    */
    if( srv_senddone(sp, SRV_DONE_ERROR |
SRV_DONE_FINAL,
        CS_TRAN_COMPLETED, (CS_INT)0) == CS_FAIL )
    if( srv_senddone(sp, SRV_DONE_FINAL,
        CS_TRAN_COMPLETED, (CS_INT)0) == CS_FAIL )
    {
        (CS_VOID)fprintf(stderr, “lang: Failed to
send a done!\n”);
    }
    return;
}
/**************************************************
** Utility Functions
**
**************************************************/
/*
**  ex_con_cleanup()
**  Type of function:
**  example program utility api
**  Purpose:
**  The routine closes a connection and
**  deallocates the CS_CONNECTION structure.
**  Parameters:
**  connection- Pointer to connection structure.
**  status  - status of last interaction with this
**  Client-Library
**  If not ok, this routine will perform a
**  force close.
**  Returns:
**  Result of function calls from CT-Lib.
*/
```

```
CS_RETCODE CS_PUBLIC
ex_con_cleanup(connection, status)
CS_CONNECTION*connection;
CS_RETCODEstatus;
{
    CS_RETCODEretcode;
    CS_INT  close_option;
    close_option = (status != CS_SUCCEED) ?
CS_FORCE_CLOSE : CS_UNUSED;
    retcode = ct_close(connection, close_option);
    if (retcode != CS_SUCCEED)
    {
        fprintf(stderr,”ex_con_cleanup: ct_close()
failed”);
        return retcode;
    }
    retcode = ct_con_drop(connection);
    if (retcode != CS_SUCCEED)
    {
        fprintf(stderr,”ex_con_cleanup:
ct_con_drop() failed”);
        return retcode;
    }
    return retcode;
}
/*
**  ex_ctx_cleanup()
**  Type of function:
**  example program utility api
**  Purpose:
**  The routine exits Client-Library deallocates
**  the CS_CONTEXT structure.
**  Parameters:
**  context - Pointer to context structure.
**  status  - status of last interaction with
**  Client-Library.
**  If not ok, this routine will perform a forced
**  exit.
**  Returns:
**  Result of function calls from CT-Lib.
*/
CS_RETCODE CS_PUBLIC
ex_ctx_cleanup(context, status)
CS_CONTEXT*context;
CS_RETCODEstatus;
```

```
{
    CS_RETCODEretcode;
    CS_INT  exit_option;
    exit_option = (status != CS_SUCCEED) ?
CS_FORCE_EXIT : CS_UNUSED;
    retcode = ct_exit(context, exit_option);
    if (retcode != CS_SUCCEED)
    {
        fprintf(stderr,”ex_ctx_cleanup: ct_exit()
failed”);
        return retcode;
    }
    retcode = cs_ctx_drop(context);
    if (retcode != CS_SUCCEED)
    {
        fprintf(stderr,”ex_ctx_cleanup:
cs_ctx_drop() failed”);
        return retcode;
    }
    return retcode;
}
server_login(Cx,Cn,username,password,server)
CS_CONTEXT    **Cx;
CS_CONNECTION  **Cn;
char    *username, *password, *server;
{
    CS_RETCODE retcode;
    char cur_username[31], cur_password[31],
cur_server[31];
    CS_INT len, status, logout;

    if (*Cx == (CS_CONTEXT *) NULL)
    {
        retcode = create_context(Cx);
        if (retcode == CS_FAIL)
            return CS_FAIL;
    }
    if (*Cn == (CS_CONNECTION *)NULL)
    {
        retcode = ct_con_alloc(*Cx,Cn);
        if (retcode != CS_SUCCEED)
        {
            fprintf(stderr,
            “Failed to allocate connection.
Failing...\n”);
            return CS_FAIL;
        }
    }
    else
    {
        retcode = ct_con_props(*Cn, CS_GET,
CS_LOGIN_STATUS, &status,
            CS_UNUSED,&len);
        if (retcode != CS_SUCCEED)
        {
            fprintf(stderr,
            “Failed to get login status.
               Failing...\n”);
            return CS_FAIL;
        }
        if (status == CS_SUCCEED)
        {
/*
** If already connected, check for a name/server/
** password change.
*/
            retcode = ct_con_props(*Cn, CS_GET,
CS_USERNAME,
                cur_username,31,&len);
            if (retcode != CS_SUCCEED)
            {
                fprintf(stderr, “Failed to get
                  username. Failing...\n”);
                return CS_FAIL;
            }
            retcode = ct_con_props(*Cn, CS_GET,
CS_PASSWORD,
                cur_password,31,&len);
            if (retcode != CS_SUCCEED)
            {
                fprintf(stderr, “Failed to get
                  password. Failing...\n”);
                return CS_FAIL;
            }
            retcode = ct_con_props(*Cn, CS_GET,
CS_SERVERNAME,
                cur_server,31,&len);
            if (retcode != CS_SUCCEED)
            {
                fprintf(stderr,
                “Failed to get password.
                    Failing...\n”);
                return CS_FAIL;
            }
            logout = 0;
            if (strcmp(username,cur_username))
                logout = 1;
            if (strcmp(password,cur_password))
                logout = 1;
            if (strcmp(server,cur_server))
                logout = 1;
            if (logout == 1)
            {
                retcode = ct_close(*Cn, CS_UNUSED);
                if (retcode == CS_FAIL)
                {
                    retcode = ct_close(*Cn, CS_
                      FORCE_CLOSE);
                    if (retcode == CS_FAIL)
                    {
                        fprintf(stderr, “Failed to
                          close connect\n”);
                        return CS_FAIL;
                    }
                }
            }
        }
    }

    retcode = ct_con_props(*Cn, CS_GET,
CS_LOGIN_STATUS, &status,
        CS_UNUSED,&len);
    if (retcode != CS_SUCCEED)
    {
        fprintf(stderr, “Failed to get login
          status. Failing...\n”);
        return CS_FAIL;
    }
/*
** Already Logged In.
*/
    if (status == CS_TRUE)
    {
        fprintf(stderr, “Already Logged In\n”);
        return CS_SUCCEED;
    }
    retcode = ct_con_props(*Cn, CS_SET,
CS_USERNAME, username,
        CS_NULLTERM,NULL);
    if (retcode != CS_SUCCEED)
    {
        fprintf(stderr,
        “Failed to change username. Failing...\n”);
        return CS_FAIL;
    }
    retcode = ct_con_props(*Cn, CS_SET,
CS_PASSWORD, password,
        CS_NULLTERM,NULL);
    if (retcode != CS_SUCCEED)
    {
        fprintf(stderr,
        “Failed to change username. Failing...\n”);
        return CS_FAIL;
    }
    retcode = ct_con_props(*Cn, CS_SET, CS_APPNAME,

        “Costing Routine”,CS_NULLTERM,NULL);
    if (retcode != CS_SUCCEED)
    {
        fprintf(stderr, “Failed to change
           application name. Failing...\n”);
        return CS_FAIL;
    }
    retcode = ct_connect(*Cn,server,strlen
        (server));
    if (retcode != CS_SUCCEED)
    {
        fprintf(stderr,”Failed To Connect (%d)\n”,
          retcode);
        return CS_FAIL;
    }
    return CS_SUCCEED;
}
create_context (Cx)
CS_CONTEXT **Cx;
{
    CS_RETCODE retcode;
    retcode = cs_ctx_alloc(CS_VERSION_100,Cx);
    if (retcode != CS_SUCCEED)
    {
        fprintf(stderr, “Failed to allocate
context structure. Failing...\n”);
        return CS_FAIL;
    }
    retcode = ct_init(*Cx,CS_VERSION_100);
    if (retcode != CS_SUCCEED)
    {
        fprintf(stderr,
        “Failed to initialize context structure.
Failing...\n”);
        return CS_FAIL;
    }
    ct_callback(*Cx, (CS_CONNECTION *) NULL,CS_SET,
        CS_SERVERMSG_CB, ex_servermsg_cb);
    ct_callback(*Cx, (CS_CONNECTION *) NULL,CS_SET,
        CS_CLIENTMSG_CB, ex_clientmsg_cb);
}
CS_RETCODE pass_lang_handler(sp,cmd)
SRV_PROC*sp;
CS_COMMAND*cmd;
{
    CS_CONNECTION*conn;/* connection handle. */
    CS_BYTE *packet;
    CS_INT  srvinfo;
    CS_RETCODEctinfo;
    CS_BOOL passthru_ok;
    /*
    **  Get the connection handle for this
    **  connection.
    */
    if ( srv_thread_props(sp, CS_GET,
SRV_T_USERDATA, &conn,
        sizeof(CS_CONNECTION *), (CS_INT *)NULL)
== CS_FAIL )
    {
        fprintf(stderr, “srv_thread_props
error\n”);
        done_error(sp);
        return(CS_FAIL);
    }
    passthru_ok = CS_TRUE;
    /*
    **  We finished passing the packets from the
    **  client to the remote server.  Now we are
    **  going to read the results from the remote
    **  server and pass them along to the client.
    */
    ctinfo = CS_PASSTHRU_MORE;
    while ( ctinfo == CS_PASSTHRU_MORE )
    {
        /*
        **  Read results from the remote server.
        */
        ctinfo = ct_recvpassthru(cmd, (CS_VOID
**)&packet);
        /*
        **  Make sure that ct_recvpassthru didn’t
        **  return something other than
        **  CS_PASSTHRU_MORE and CS_PASSTHRU_EOM.
        */
        if ( (ctinfo != CS_PASSTHRU_MORE)
            && (ctinfo != CS_PASSTHRU_EOM) )
        {
            passthru_ok = CS_FALSE;
            break;
        }
        /*
        **  Now send the packet to the client.
        */
        if (srv_sendpassthru(sp, packet, &srvinfo)
== CS_FAIL )
        {
            passthru_ok = CS_FALSE;
            break;
        }
    }
    /*
    **  If all went well, srv_sendpassthru should
    **  set srvinfo to SRV_I_PASSTHRU_EOM.
    */
    if ( srvinfo != SRV_I_PASSTHRU_EOM )
    {
        passthru_ok = CS_FALSE;
    }

    /*
    **  If there’s any error, clean up, and stop
    **  the server.
    */
    if ( !passthru_ok )
    {
        done_error(sp);
        return(CS_FAIL);
    }
    return(CS_SUCCEED);
}
```

```
send_message(sp, message)
SRV_PROC*sp;
char    *message;
{
    CS_CONTEXT*cp;  /* Context structure. */
    CS_SERVERMSGmsg;/* The message we’ll send. */
    CS_CHAR sname[CS_MAX_NAME];/* The server 
name.*/
   CS_INTslen;/* The server name length.*/
   CS_INTlen;/* the length of the message.*/
   /*
   ** Initialization.
   */
   srv_bzero(&msg, sizeof(msg));
   /*
   ** Get the CS_CONTEXT structure.
   */
   if( cs_ctx_global(CS_VERSION_100, &cp) == 
CS_FAIL )
   {
    fprintf(stderr,”cs_ctx_global Failed\n”);
    done_error(sp);
    return CS_FAIL;
   }
   /*
   ** Get the name of the server.
   */
   {
    fprintf(stderr,”srv_props Failed\n”);
    done_error(sp);
    return CS_FAIL;
   }
   strcpy(msg.text,message);
   msg.textlen = strlen(msg.text);
   msg.msgnumber = 20999;
   /*
   ** Fill in the server name field as well.
   */
   (CS_VOID)strncpy(msg.svrname, sname, slen);
   msg.svrnlen = slen;
   /*
   ** Send the message to the client.
   */
   msg.status = (CS_FIRST_CHUNK|CS_LAST_CHUNK);
   if( srv_sendinfo(sp, &msg, CS_TRAN_UNDEFINED) == 
CS_FAIL )
```

```
   {
    fprintf(stderr,”srv_sendinfo Failed\n”);
    done_error(sp);
    return CS_FAIL;
   }
   return CS_SUCCEED;
}
CS_RETCODE connect_handler(sp)
SRV_PROC*sp;
{
   CS_INTgeneral_csint;
   CS_CONNECTION*conn;
   CS_CHARgeneral_str[CS_MAX_CHAR];
   CS_LOGINFO*loginfo;
   CS_INTstatus;
   CS_CONTEXT*context;
   /*
   **Assume that everything is ok to start.
   */
   status = SRV_DONE_FINAL;
   /*
   **If this is a site handler, just send a
   **DONE and return.
   */
   if ( srv_thread_props(sp, CS_GET, SRV_T_TYPE,
    (CS_VOID *)&general_csint, sizeof(CS_INT),
    (CS_INT *)NULL) == CS_FAIL )
   {
    done_error(sp);
    return(CS_FAIL);
   }
   if ( general_csint == SRV_TSITE )
   {
    done_error(sp);
    return(CS_SUCCEED);
   }
   /*
   **Retrieve the context for this Open Server
   **application.
   */
   create_context (&context);
   /*
   **Get a connection handle for the connection
   **attempt.Note that the context used here is
   **the global CS_CONTEXT data.  The context
   **structure returned by ct_ctx_global
   **doesn’t seem to work.  The client
   **got “Login incorrect” error message.
   */
   if ( ct_con_alloc(context, &conn) == CS_FAIL )
   {
    done_error(sp);
    return(CS_FAIL);
   }

   /*
   **Get and set username, user password,
   **application name, TDS packet size, etc.
   */
   if ( srv_getloginfo(sp, &loginfo) == CS_FAIL )
   {
    (CS_VOID)ct_con_drop(conn);
    return(CS_FAIL);
   }

   /*
   **We check for CS_SUCCEED because this call
   **could return CS_BUSY.
   */
   if ( ct_setloginfo(conn, loginfo) != CS_SUCCEED )
   {
    ct_con_drop(conn);
    return(CS_FAIL);
   }
   /*
   ** Get and print the TDS packet size.  This is
   **only for informational purposes, so we will
   **continue even if the call fails.
   */
   if ( srv_thread_props(sp, CS_GET,
SRV_T_PACKETSIZE,
    &general_csint, sizeof(CS_INT), (CS_INT *)NULL)
    == CS_SUCCEED )
   {
    /*
    ** Print the client’s requested packet
    ** size to the Open Server application
    ** log file.
    */
    (CS_VOID)sprintf(general_str,
       “Packet size requested %d\n”, general_csint);
    (CS_VOID)srv_log( (SRV_SERVER *)NULL, CS_TRUE,
general_str,
       CS_NULLTERM);
   }
   /*
   ** Try a connection to the remote DBMS
   ** defined in the DSQUERY enviroment variable.
   */
   if ( ct_connect(conn, SERVERNAME, CS_NULLTERM)
== CS_FAIL )
   {
    (CS_VOID)sprintf(general_str,
       “Can’t connect to remote server %s\n”,
SERVERNAME);
    (CS_VOID)srv_log( (SRV_SERVER *)NULL, CS_TRUE,
general_str,
       CS_NULLTERM);
    done_error(sp);
    (CS_VOID)ct_con_drop(conn);
    return(CS_FAIL);
   }
   (CS_VOID)sprintf(general_str, “Connected to
remote server %s\n”,
    SERVERNAME);
   (CS_VOID)srv_log( (SRV_SERVER *)NULL, CS_TRUE,
general_str,
    CS_NULLTERM);
   /*
   **Get the login reponse from the remote
   **server, and pass it to the client.
   */
   if ( ct_getloginfo(conn, &loginfo) == CS_FAIL )
   {
    (CS_VOID)ct_close(conn, CS_FORCE_CLOSE);
    (CS_VOID)ct_con_drop(conn);
    done_error(sp);
    return(CS_FAIL);
   }
   if ( srv_setloginfo(sp, loginfo) == CS_FAIL )
   {
        (CS_VOID)ct_close(conn, CS_FORCE_CLOSE);
        (CS_VOID)ct_con_drop(conn);
        done_error(sp);
        return(CS_FAIL);
    }
    /*
    **  Save the connection handle in SRV_PROC’s
    **  user data space.  This is done almost at
    **  the last minute so that we don’t have to
    **  clear this user pointer if something goes
    **  wrong before returning.
    */
    if ( srv_thread_props(sp, CS_SET,
SRV_T_USERDATA, conn,
        sizeof(CS_CONNECTION *), (CS_INT *)NULL)
== CS_FAIL )
    {
        (CS_VOID)ct_close(conn, CS_FORCE_CLOSE);
        (CS_VOID)ct_con_drop(conn);
        done_error(sp);
        return(CS_FAIL);
    }
    /*
    **Send client the connection acknowlegement.
    */
    done_error(sp);
    return(CS_SUCCEED);
}
/*
**  ex_clientmsg_cb()
**  Type of function:
**  example program client message handler
**  Purpose:
**  Installed as a callback into Open Client.
**  Returns:
**  CS_SUCCEED
**  Side Effects:
**  Since this routine uses stdio routines, it may
**  not be safe to call it at interrupt level.
*/
CS_RETCODE CS_PUBLIC
ex_clientmsg_cb(context, connection, errmsg)
CS_CONTEXT*context;
CS_CONNECTION*connection;
CS_CLIENTMSG*errmsg;
{
    fprintf(stderr, “\nOpen Client Message:\n”);
    fprintf(stderr, “Message number: LAYER = (%ld)
ORIGIN = (%ld) “,
        CS_LAYER(errmsg->msgnumber),
CS_ORIGIN(errmsg->msgnumber));
    fprintf(stderr, “SEVERITY = (%ld) NUMBER =
(%ld)\n”,
        CS_SEVERITY(errmsg->msgnumber),
CS_NUMBER(errmsg->msgnumber));
    fprintf(stderr, “Message String: %s\n”, errmsg-
>msgstring);
    if (errmsg->osstringlen > 0)
    {
        fprintf(stderr, “Operating System Error:
%s\n”,
            errmsg->osstring);
    }
    fflush(stderr);
    return CS_SUCCEED;
}
/*
**  ex_servermsg_cb()
**  Type of function:
**  example program server message handler
**  Purpose:
**  Installed as a callback into Open Client. If
**  extended error data is detected, the routine
**  will call ex_fetch_data() to display the rows.
**  Returns:
**  CS_SUCCEED
**  CS_FAIL If we could not get the extended error
**  command handle.
**  Side Effects:
**  Since this routine uses stdio routines, it may
**  not be safe to call it at interrupt level.
*/
CS_RETCODE CS_PUBLIC
ex_servermsg_cb(context, connection, srvmsg)
CS_CONTEXT*context;
CS_CONNECTION*connection;
CS_SERVERMSG*srvmsg;
{
    CS_COMMAND*cmd;
    CS_RETCODEretcode;

    fprintf(stderr, “\nServer message:\n”);
    fprintf(stderr, “Message number: %ld, Severity
%ld, “,
        srvmsg->msgnumber, srvmsg->severity);
    fprintf(stderr, “State %ld, Line %ld\n”,
        srvmsg->state, srvmsg->line);

    if (srvmsg->svrnlen > 0)
    {
        fprintf(stderr, “Server ‘%s’\n”, srvmsg-
>svrname);
    }

    if (srvmsg->proclen > 0)
    {
        fprintf(stderr, “ Procedure ‘%s’\n”,
srvmsg->proc);
    }
    fprintf(stderr, “Message String: %s\n”, srvmsg-
>text);
    fflush(stderr);
    return CS_SUCCEED;
}
int log_function(sp,CmdText)
SRV_PROC*sp;
char    *CmdText;
{
/*
** In reality, this is where one would log the
** request to a file with a username and timestamp.
*/
    printf(“LOG >> %s\n”,CmdText);
}
```

Common Error Numbers and Messages

The remainder of this chapter gives explanations and remedies for some errors commonly encountered when developing or using Open Server applications.

Error 16011

Error Message Text

```
SERVER: FATAL SERVER ERROR: 16011/20/0: Bad
interfaces file.
```

Explanation

This error number occurs only with releases earlier than 10.0.

When you try to start an Open Server, it does not start and displays Error 16011.

➤ ***Note***

As of 10.0, the scenarios described below are more likely to result in error 16240. Check AnswerBase Technical Notes for the information on error 16240.

Open Server initializes one or more listening services at start-up time. These listening services are network listeners bound to a specific network address. An Open Server application on most platforms gets the network information for starting these listeners from a SYBASE network address file: on UNIX platforms, the interfaces file; on OS/2 the *os2.ini* or *sql.ini* file. This error means that the Open Server has found that the information in the network address file is incorrect or bad.

The error's severity is 20 because it prevents Open Server from starting.

There are several potential reasons for this error:

- The master entry for this specific Open Server is missing in the network address file. The Open Server uses the master entry in this file to determine the network address and port number at which to start listeners. Failure to find a master entry raises a 16011 error.

➤ ***Note***

The above information is true only for Open Server releases 2.0 and later. Open Server release 1.0 uses the query entry in the interfaces file.

- The query entry is missing in the network address file. Whenever another Server attempts to send an RPC to an Open Server, it attempts to connect to the Open Server's query listener service.

Therefore if the query entry is missing, the other server cannot connect and this error is raised.

- The structure of the network address file is incorrect.

Action

Verify that the master and query entries in your network address file for the Open Server are correct. Make sure that the file adheres to the correct format and entries are delimited properly.

To find the correct format for 10.x releases, refer to the *Open Client/Server Supplement* for the platform on which Open Server is running. For earlier releases, refer to your installation manuals.

Make sure the above entries are the same and are unique on a given host.

Use SYBASE utilities like **sybinit** (for UNIX) or **sqledit** (for some PC platforms), to create, modify, or add entries to network address files. This ensures that the file is properly formatted.

Make sure the SYBASE environment variable is set properly; by default Open Server always uses the interfaces file located in the SYBASE root directory.

Error 16027

Error Message Text

```
Can't bind socket.
```

Explanation

This error only occurs in SQL Server releases previous to 10.0.

When you try to start an Open Server, it does not start and displays Error 16027.

This error may occur for the following reasons:

- When Open Server is started the operating system detects, correctly or not, that the port address for the socket is still in use by Open Server. This may be because the port address specified in the network address file (interfaces file on UNIX platforms) is already in use by either another Open Server, a SQL Server, or a non-SYBASE application on the network.
- This error can also occur at run time on platforms using Transport Layer Interface (TLI) networking as a result of network or operating system resource problems. The error is raised when Open Server tries to bind a client connection end point to a transport provider assigned address.

Action

If the error occurs during Open Server start-up, it prevents Open Server from starting. The following steps may help you resolve the error:

1. Check to be sure that Open Server is not already running.
2. If you just stopped Open Server, wait 60 seconds and then restart it. (After you free a port address, most operating systems impose a 60-second waiting period before the port address can be used again.)
3. If some other server or non-SYBASE application is using the port specified in the network address file, you may need to choose a different port for Open Server. Change the Open Server's port in the network address file, and in any other network address files used by applications that connect to Open Server.

If the error occurs while Open Server is running, the message is informational only. However, the client whose connection failed needs to retry the connection. If the message occurs frequently while

Open Server is running, check for network and other operating system problems.

➤ ***Note***

In Open Server 2.0 and later, the network address file should contain both query and master entries.

Error 16059

Error Message Text

```
server: server error: 16059/20/0: Network Select
found no sockets had send SIGURG -- SIGURG Ignored
```

Explanation

This error occurs only in SQL Server releases previous to 10.0.

Open Server installs a default handler for trapping the SIGURG signal. This signal normally indicates the arrival of an attention packet. The Open Server internal handler figures out which socket the attention came in and cancels any pending read requests on that specific socket. Then, if there is a user-installed attention handler, it gets called. This error message indicates that the Open Server could not locate the socket for which the attention was raised. This could be due to events such as the client disconnecting completely or network-level inconsistencies.

Since the Open Server's default handler is invoked at an interrupt level, a user-installed attention handler is preferable to handle application-specific actions on such signals.

The default signal handler for SIGURG flushes out any preceding data "in the pipe," and attempts to read the Out-Of-Band packet from the client. Next, it attempts to call the user attention handler. In general, the Open Server application will be in one of the three states explained below. Recommended actions are given for each state.

Action

Table 7-1: Error 16059 recommended actions

Open Server Application	Attention Handler Recommended Action
Sending row replies	Set ATTN flag and test it after each row sent. Mainline should issue srv_senddone(SRV_DONE_ATTN)
Processing RCVD query (e.g. waiting for I/O from remote system)	Set ATTN flag and periodically test it before returning control to Open Server Server-Library for next event. Mainline should issue srv_senddone(SRV_DONE_ATTN)
Waiting for next event	Open Server Server-Library will automatically swallow the OOB, discard the event and data, and send SRV_DONE_ATTN.

Error 16145 (SRV_EPROCIODEAD)

Error Message Text

```
Server process has an invalid i/o.
```

Explanation

Applies to both 2.x and 10.x releases.

This occurs when the client connection is lost when the user sends an interrupt (Ctrl-c), powers off a client PC, or initiates a similar, unorthodox shutdown. The gateway has results to send to the client; if the client is not available, SQL Server gives a 16145 error for every row that is to be sent.

- The SQL Server process (or thread control structure that was allocated when the connection was established) is dead when the client connection goes away.
- The user must kill the process associated with the lost client connection and restart the client application. The application is not affected by this, but there will be an entry in the error log.

Action

Trap this error in order to avoid it. There are two possible methods:

- Trap the error while results are being returned.

  Write a function ("user-written function") that will check to see if the thread control structure is dead (using **SRV_IODEAD**). If the process is dead, queue a disconnect event using **srv_event(srvproc, DISCONNECT, NULL)** and quit the event handler for that particular thread.

- Trap the error before SQL Server sends back results.

  Check to see if the thread control structure is dead before processing every query. If it is dead, cancel the query in the language handler. For an example of how this can be done, see the **handle_results** function of the *lang.c* sample program that is provided with the Open Server Server-Library product.

8 PC Issues

This chapter covers problems that might arise when you use SQL Server in a PC environment. It is organized by operating system, in the following sections:

- “General Information” starting on page 8-2
- “DOS-Specific Information” starting on page 8-11
- “OS/2-Specific Information” starting on page 8-14
- “Windows-Specific Information” starting on page 8-16

Save the *Installation Guide* for your OS/2, Novell or Windows NT SQL Server, and the *Installation Guide* for your Sybase Open Client/Server or Net-Library products. These contain valuable troubleshooting sections that will be useful after installation.

General Information

PC Connectivity Checklist

If you are unable to solve your PC connectivity problem with the information in this chapter, collect the following information before having your primary site contact call Sybase Technical Support:

- SQL Server release version and operating system.
- Client operating system and version. Be as specific as possible.
- The exact error message that notified you of the problem.
- The network vendor and version of the network products involved.
- DB-Library library name, size and date.
- Client-Library library name, size and date.
- Net-Library library name, size and date.
  - On all PC platforms except DOS and NetWare, libraries are implemented as DLLs. Check your product supplement manuals for library names.

➤ ***Note***

The 10.0.1 production release for Windows did not have a System 10 DB-Library product. Instead, Sybase bundled DB-Library release 4.2.5 and Net-Library release 1.0.2 with the System 10 product; a beta release of Open Client is available now, which includes System 10 DB-Library, eliminating the bundling requirement. A similar bundling was used for DOS. System 10 Net-Libraries will soon be bundled with the Open Client and Open Server products.

- Sybase or third-party application being used, version, and other vendor-specific information.
- Compiler name, version, and platform.
- Whether the problem is reproducible or intermittent.
- Whether this is a new or old installation.

General Connectivity Troubleshooting Information

This table contains general connectivity troubleshooting information. Windows-specific information is contained in“Windows connectivity—frequently asked questions” starting on page 8-25

Table 8-1: PC connectivity troubleshooting

Question	Answer
wdbping, **dbping**, or **sybping** does not work. What do I do?	If the network vendor-supplied **ping** works, see the information about **wdbping** in “Errors Common with wdbping” starting on page 8-28. Troubleshooting information about **wsybping** (or **sybping** on Windows NT and DOS) for 10.0 and later releases is in the *Open Client/Server Supplement for Windows* If the network vendor’s **ping** does not work, call the network vendor.
What do I need to connect to SQL Server?	Based on the client and server operating system and protocol desired at both ends, determine the appropriate Net-Library. Beginning with System 10, you do not need to purchase different Net-Library drivers separately. You need to ensure that your interfaces files (*sql.ini* and *libtcl.cfg*, or whatever the name is for your platform) are properly configured based on available networking software. Refer to the “Setting up the Environment” chapter in the appropriate *Supplement* for your platform for examples of interfaces file entries. Applications must make a call to open the connection, for example: DB-Library: **dbopen** Client-Library: **ct_connect** The DSQUERY environment variable must be set.
Cancellation of queries takes a while before getting control back. Why?	This is an out-of-band data issue. Check if the protocol stack supports out-of-band data, of what flavor (RFC 793 or RFC 1122), and how it is supported. Check for the **urgent** parameter in the connection string.
Sleeping processes are left behind on the server. Why?	When clients are suddenly lost, for example, through a GP fault or a reboot without closing the application, they sometimes leave sleeping processes behind. See the information on **keepalive**, on page 8-10.
Sleeping processes are left behind on the server even after the **keepalive** period. Why?	Many network vendors are TSR-based, so if a GP fault appears in Windows, and you restart Windows, the process is not cleared on the client. A reboot is required. As of November 1994, Netmanager and SuperTCP are the only fully DLL-based PC protocol stacks.

Compatibility Charts

Here is a chart detailing the certified combinations of Net-Library, DB-Library, Client-Library, Open Server and APT.

Table 8-2: Compatibility chart for connectivity products and others

Net-Library	DB-Library	Client-Library	Open Server	APT
1.x	4.2	N / A	1.0	5.x
2.x (OS/2 only)	4.6	N / A	2.0	N / A
10.x	10.x	10.x	10.x	N / A

◆ ***WARNING!***

You cannot use APT with any Net-Library other than 1.x.

Net-Library Certifications

New Net-Libraries are certified regularly and frequently. This list is partial.

DOS

Table 8-3: DOS Net-Library certifications

Version	Protocol	Cat. No.	Bit 16/ 32	Vendor	File Name
1.0.2	TCP/IP	11590	16	3Com 3+Open TCP 1.2	*db3optcp.exe*
1.0.2	TCP/IP	11580	16	FTP PC/TCP 2.1, 2.2, 2.3	*dbftptcp.exe*
1.0.2	TCP/IP	11630	16	HP ARPA Services 2.0, 2.1, 2.11	*dbhptcp.exe*
1.0.2	TCP/IP	11650	16	Microsoft TCP/IP 1.0	*dbmstcp.exe*
1.0.2	TCP/IP	11870	16	Novell LAN WorkPlace 4.1	*dbnovtcp.exe*
1.0.2	TCP/IP	11870	16	Novell LAN WorkGroup 4.1	*dbnovtcp.exe*
1.0.2	TCP/IP	11550	16	Sun PC-NFS 4.0, 5.0	*dbsuntcp.exe*
1.0.2	TCP/IP	11630	16	Walker, Richey, Quinn 2.01	*dbhptcp.exe*
1.0.2	TCP/IP	11640	16	Wollongong PathWay 2.2./Kernel 1.2.0.1	*dbwoltcp.exe*
1.0.2	IPX/SPX	11600	16	Novell NetWare 3.1	*dbnovspx.exe*
1.0.2	IPX/SPX	11600	16	Novell VLM NetWare Client 4.0.1	*dbnovspx.exe*
1.0.1	Named pipes	11610	16	Banyan VINES 4.1	*dbnmpipe.exe*
1.0.1	Named pipes	11610	16	IBM LAN Server 1.3 w/ESD, wr04097+	*dbnmpipe.exe*
1.0.1	Named pipes	11610	16	Microsoft LAN Manager 2.2	*dbnmpipe.exe*
1.0.1	Named pipes	11610	16	Novell NetWare Requester 1.0+	*dbnmpipe.exe*
1.0.1	Named pipes	11610	16	Ungermann-Bass Net/One LAN Mgr 1.0+	*dbnmpipe.exe*
1.0.2	DECnet	11560	16	DEC Pathworks: DECnet 4.1	*dbdecdec.exe*
1.0	StarGROUP	11620	16	AT&T StarGROUP 3.2+	*dbatttcp*

Windows

Table 8-4: Windows Net-Library certifications

Version	Protocol	Cat. No.	Bit 16/ 32	Vendor	File Name
1.0.2	TCP/IP	11830	16	Wollongong PathWay 2.2/Kernel 1.2.0.1	*wdbwoltc.dll*
1.0.2	TCP/IP	10840	16	Novell LAN WorkPlace 4.1	*wdbnovtc.dll*
1.0.2	TCP/IP	10840	16	Novell LAN WorkGroup 4.1	*wdbnovtc.dll*
1.0.2	TCP/IP	11820	16	FTP PC/TCP 2.1, 2.2, 2.3	*wdbftptc.dll*
1.0.2	TCP/IP	11810	16	HP ARPA Services 2.1, 2.11	*wdbhptcp.dll*
1.0.2	TCP/IP	11810	16	Walker, Richey, Quinn 2.01	*wdbhptcp.dll*
1.0.2	TCP/IP	11850	16	Microsoft TCP/IP 1.0	*wdbmstcp.dll*
1.0.2	TCP/IP	11870	16	NetManage NEWT TCP/IP 3.1.1	*wdbnewtc.dll*
1.0.2	TCP/IP	11880	16	Sun PC-NFS 4.0, 5.0	*wdbsuntc.dll*

Table 8-4: Windows Net-Library certifications (continued)

Version	Protocol	Cat. No.	Bit 16/ 32	Vendor	File Name
1.0.2	TCP/IP	11802	16	WinSock: IBM TCP DOS 2.1.1	*wdbwsktc.dll*
1.0.2	TCP/IP	11802	16	WinSock: Frontier SuperTCP 3.0	*wdbwsktc.dll*
1.0.2	TCP/IP	11802	16	WinSock: FTP TCP/IP 2.2	*wdbwsktc.dll*
1.0.2	TCP/IP	11802	16	WinSock: NetManage NEWT 3.1.1	*wdbwsktc.dll*
1.0.2	TCP/IP	11802	16	WinSock: Wollongong PathWay 2.2/Krnl 1.2	*wdbwsktc.dll*
1.0.2	DECnet	11800	16	DEC Pathworks: DECnet 4.1	*wdbdecde.dll*
1.0	StarGROUP	10850	16	AT&T StarGROUP 3.3+	*wdbatttc.dll*
1.0.2	SPX/IPX	11840	16	Novell NetWare 3.1	*wdbnovsp.dll*
1.0.2	Named pipes	10860	16	IBM LAN Server 1.3 w/CSD WR04097+	*dbnmp3.dll*
1.0.2	Named pipes	10860	16	Novell NetWare Requester 1.3	*dbnmp3.dll*
1.0.2	Named pipes	10860	16	3Com 3+Open 1.1+	*dbnmp3.dll*
1.0.2	Named pipes	10860	16	Ungermann-Bass Net/One LAN Mgr 1.0+	*dbnmp3.dll*
1.0.2	Named pipes	10860	16	Microsoft LAN Manager 2.2	*dbnmp3.dll*
1.0.2	Named pipes	10860	16	Banyan VINES 4.1	*dbnmp3.dll*
10.0	Named pipes	11895	16	Same as 1.0.2	*wnlnmp.dll*
10.0	SPX/IPX	11895	16	Novell NetWare 3.1	*wnlnovsp.dll*
10.0	TCP/IP	11895	16	Same as 1.0.2	*wnlwnsck.dll*
10.0	TCP/IP	11895	16	NetManage Newt TCP/IP 3.1.1	*wnlnewtc.dll*
10.0	TCP/IP	11895	16	FTP PC/TCP 2.1, 2.2	*wnlftptc.dll*
10.0	TCP/IP	11895	16	Novell LAN WorkPlace 4.1	*wnllwptc.dll*

OS/2 Versions 1.3 and 2.0

Table 8-5: OS/2 1.3 and 2.0 Net-Library certifications

Version	Protocol	Cat. No.	Bit 16/ 32	Vendor	File Name
1.0.2	TCP/IP	16200	16	FTP PC/TCP 1.1, 1.2 on OS/2 2.0	*dbftptcp.dll*
1.0.2	TCP/IP	16150	16	Novell LAN WorkPlace 2.0, 3.0	*dbnovtcp.dll*
1.0.2	TCP/IP	16160	16	3Com 3+Open TCP 1.2	*db3optcp.dll*
1.0.2	TCP/IP	16210	16	Microsoft TCP/IP 1.0	*dbmstcp.dll*
1.0.2	TCP/IP	16140	16	IBM TCP/IP 1.2, 2.0	*dbibmtcp.dll*
1.0.2	DECnet	16190	16	DEC PathWorks 1.2	*dbdecdec.dll*
1.0.2	SPX/IPX	16135	16	Novell NetWare Requestor OS/2 1.3	*dbnovspx.dll*
1.0.2	Named pipes	16170	16	IBM LAN Server 2.0, 3.0	*dbnmpp.dll*
1.0.2	Named pipes	16170	16	Novell NetWare Requestor 1.1+	*dbnmpp.dll*
1.0.2	Named pipes	16170	16	3Com 3+Open 1.1+	*dbnmpp.dll*

Table 8-5: OS/2 1.3 and 2.0 Net-Library certifications (continued)

Version	Protocol	Cat. No.	Bit 16/ 32	Vendor	File Name
1.0.2	Named pipes	16170	16	Banyan VINES 4.1	*dbnmpp.dll*
1.0.2	Named pipes	16170	16	Microsoft LAN Manager 2.0+	*dbnmpp.dll*
1.0.2	Named pipes	16170	16	Ungermann-Bass Net/One LAN Manager 1.0+	*dbnmpp.dll*

OS/2 Version 2.1 Only

Table 8-6: OS/2 Version 2.1 only Net-Library certifications

Version	Protocol	Cat. No.	Bit 16/ 32	Vendor	File Name
2.0	TCP/IP	16125	32	IBM TCP/IP 1.2, 2.0	*nlibmtcp.dll*
2.0	Named pipes	16121	32	IBM LAN Server 2.0, 3.0	*nlmsnmp.dll*
2.0	Named pipes	16121	32	Novell NetWare Requestor 1.1+	*nlmsnmp.dll*
2.0	Named pipes	16121	32	3Com 3+Open 1.1+	*nlmsnmp.dll*
2.0	Named pipes	16121	32	Ungermann-Bass Net/One LAN Mgr 1.0+	*nlmsnmp.dll*
2.0	Named pipes	16121	32	Microsoft LAN Manager 2.0+	*nlmsnmp.dll*
2.0	Named pipes	16121	32	Banyan VINES 4.1	*nlmsnmp.dll*
2.0	SPX/IPX	16145	32	Novell NetWare Requestor OS/2 2.0.1	*nlnovspx.dll*

Windows NT

Table 8-7: Microsoft Windows NT Net-Library certifications

Version	Protocol	Cat. No.	Bit 16/ 32	Vendor	File Name
10.0	Named pipes	16602	32	Native NT Interface 3.1	*nlmsnmp.dll*
10.0	TCP/IP	16604	32	Win Sock (native NT TCP) 1.1	*nlwnsck.dll*
10.0	SPX/IPX	16603	32	WinSock NWLink 1.1	*nlnwlink.dll*

How SYBASE Servers and Clients Use Named Pipes

Question

What are named pipes, and how does SQL Server use them?

Answer

A named pipe is an interprocess communication mechanism that is similar to sockets in UNIX.

Named pipes are available on the following PC operating systems:

- Windows
- Windows NT
- DOS
- OS/2

Only Windows NT and OS/2 are SQL Server platforms.

Both Windows NT and OS/2 SQL Servers open a pipe for user connections. Version 10.x SQL Servers use a pipe named *\pipe\sql10\query*; version 4.*x* SQL Servers use a pipe named *\pipe\sql\query*. Client applications can access this pipe in several ways. A pipe may be either local or remote. A local pipe allows communication between applications on the same machine; a remote pipe allows communication across machines, over a network.

Assume, for example, that an OS/2 SQL Server is running on a machine named "valkyrie." One of these scenarios can happen:

- If the client is running on a remote computer, it connects to SQL Server across the network by requesting the remote named pipe *\\valkyrie\pipe\sql\query*.
- If the client is also running on "valkyrie," it may connect to SQL Server through the local named pipe *\pipe\sql\query*, whether or not LAN Manager is running on "valkyrie."
- If the client is running on "valkyrie", and "valkyrie" is configured under LAN Manager as a network server machine, the client can connect to SQL Server through the remote named pipe *\\valkyrie\pipe\sql\query*.

➤ ***Note***

For OS/2, the network is driven by LAN Manager. There is no Windows NT version of LAN Manager; there is a Named Pipes Driver among Windows NT's network drivers, which fulfills the same functions.

How the *keepalive* Server Option Detects Client Disconnects

Question

What is the **keepalive** option, and how does it work?

Answer

The **keepalive** option instructs TCP to probe the connection periodically if it becomes idle for a long period of time. If the other end does not respond, the connection is broken.

keepalive requires operating system-level support on the server machine. It is normally a configurable TCP/IP option. OS/2 and Windows platforms do not support this option, but Novell NetWare does.

There are **keepalive** patches and fixes for different platforms. You may have to rebuild/reconfigure the operating system kernel to modify the **keepalive** parameters.

In certain situations, SQL Server is able to detect abnormal disconnects of client machines and will time out the client, thereby cleaning any host server processes, releasing locks, and logging transactions that involved the client.

The ability of the server to detect and disconnect “absent” front-end processes depends on the hardware vendor’s implementation of TCP/IP. Normally, a number of **keepalive** packets are sent at specified intervals during periods of inactivity.

Failure to receive an acknowledgment packet results in the operating system informing the server that it should clean up the stranded connection associated with that client. The process may take from 10 minutes to 2 hours.

DOS-Specific Information

Connecting DOS Clients to SQL Server

Question

How do DOS clients connect to SQL Server?

Answer

DOS clients connect to SQL Server in different ways, depending on whether or not the network uses named pipes, and whether the SQL Server name passed to **dbopen** (DB-Library) or **ct_connect** (Client-Library) is NULL or a character string.

All client applications, including both client tools such as **isql** and your own programming code, must call one of the above connect routines.

Check that one of the environment variables discussed below, either DSQUERY or <*server name*>, is set.

After setting the environment variable, check that the appropriate Net-Library is loaded for the network type being used. Only one Net-Library can be loaded at one time, but multiple server connections are possible if all are accessible through the same network type.

For Networks Using Named Pipes

If the server name passed to the connect routine is NULL, Net-Library looks for an environment variable called DSQUERY and uses its value as the pipe to open. For example, DSQUERY might be set to *\\server_name\ pipe\sql\query*. If you are on the same machine as the server, DSQUERY may be set to *\ pipe\sql\query*. If Net-Library cannot find the Server named in DSQUERY, the connection fails.

If the server name passed to the connect routine is a character string, Net-Library attempts to open a pipe *\\server_name\ pipe\sql\query*. If this fails, then the Net-Library looks for an environment variable called *server_name* and uses its value as the pipe to open. For example, *server_name* might be set to *\\server_name\pipe\sql\query1*. If a variable with that name does not exist, Net-Library looks for the DSQUERY environment variable and tries to connect; if the lookup fails, the connection fails.

For Networks Not Using Named Pipes

If the server name passed to the connect routine is NULL, Net-Library looks for the DSQUERY environment variable and uses its value to connect. If the variable is not set, then the connection fails.

If the server name passed to the connect routine is a character string, then Net-Library looks for an environment variable that matches the string. For example, executing the command **dbopen(dbproc,"MYSERVER")** causes Net-Library to look for an environment variable MYSERVER. If an environment variable with that name does not exist, then Net-Library tries to match the string with the value for DSQUERY. Again, if the lookup fails, the connection fails.

➤ ***Note***

The *urgent* specification shown in the following examples is optional.

Format of Connect Strings

- Named pipes
  - *server_name = \\server_name\ pipe\sql\query*

    example: **DSQUERY = \\valkyrie\pipe\sql\query**
- TCP-IP
  - *server_name = machine_name, query_port*[, urgent]

    example: **DSQUERY = 131.250.21, 9010, urgent**

    or: **DSQUERY = mercury, 9010, urgent**
- SPX
  - *server_name = network_number, node_number, port*[, urgent]

    or *server_name = advertised_server_name*

    example: **DSQUERY = 16, 1, 83BD, urgent**

    or: **DSQUERY = PROTON** where PROTON is the advertised server name
- DECnet
  - *server_name = node_name, port*

    example: **DSQUERY = 1.11, 27**

TSR Tips

Note that the environment variable should be set before the Net-Library TSR is loaded. If it is not, the connection will fail.

If you need to add a server name after loading the Net-Library TSR you can unload the TSR using the **unloadnl** utility provided on the Net-Library diskette, located in the *SYBASE_root\bin* directory.

Additional Information

The documentation shipped with your DOS Open Client/Server products, especially the *Installation Guide*, *Supplement*, and *Release Bulletins*, contain more information about DOS clients and SQL Server.

OS/2-Specific Information

Connecting OS/2 Clients to SQL Server

Question

How do OS/2 clients connect to SQL Server?

Answer

SQL Server name parsing is done at **dbopen** by DB-Library or **ct_connect** by Client-Library, and is the same no matter what network is loaded.

OS/2 and Multiple Server Connections

OS/2 uses **Dynamic Load Libraries** (DLLs) that allow a properly configured network environment to support connections to multiple SQL Servers on the same or different network types. You must make sure the DLL files are accessible to OS/2 by specifying the directory where they are stored in the LIBPATH directive of the *config.sys* file.

Net-Library and Server Configuration Information

If you use Net-Library release 1.x, the logical names and connection information for the accessible SQL Servers are stored in the binary file *os2.ini*. Use **sqlsetup** to read or write entries to *os2.ini*.

If you use Net-Library releases 2.x or 10.x, SQL Server configuration information is managed by the **sqledit** utility, which stores the information in a file called *sql.ini*, in the *ini* subdirectory of the root directory specified by the *SYBASE* environment variable

Net-Library 1.x: DB-Library and Server Connections

If the server name is NULL, DB-Library looks for a DSQUERY entry in the SQLSERVER section of the *os2.ini* file. If this entry is not found, the connection fails.

If the server name is a character string, DB-Library looks for an entry of the same name in the SQLSERVER section of the *os2.ini* file. If this entry is not found, it proceeds as if the server name had been NULL.

Once an entry is found, it is assumed to have the following form:

```
dll_Name,connection_info
```

Note that there is no space surrounding the comma. For example:

```
dbnovtcp,131.214.1.40,2025
```

DB-Library then attempts to load the dynamic link library and passes the connection information to **dbopen**.

Format of Connect Strings

- Named pipes
  - *server_name = dll_name, \ \ server name \ PIPE \ SQL \ QUERY*

    example: **DSQUERY = dbnmpp, \\valkyrie\PIPE\SQL\QUERY**
- TCP-IP
  - *server_name = dll_name, machine_name, query_port[, urgent]*

    example: **DSQUERY = dbibmtcp, 131.250.21, 9010, urgent**

    or: **DSQUERY = dbibmtcp, mercury, 9010, urgent**
- SPX
  - *server_name = dll_name, network_number, node_number, port[, urgent]*

    example: **DSQUERY = dbnovspx, 16, 1, 83BD, urgent**
- DECnet
  - server_name = *dll_name, node name, port*

    example: **DSQUERY = 1.11, 27**

The correct format for entries in *sql.ini*, for Net-Library 2.x and 10.x, is described in the documentation accompanying those products.

Additional Information

The documentation shipped with your SQL Server, especially the *Installation Guide*, *Supplement*, and *Release Bulletins*, contain more information about OS/2 and SQL Server.

Windows-Specific Information

Connecting 1.x Windows Clients to SQL Server

See the Appendix of the *Open Client Net-Library Installation and Reference Manual* release 1.x for more complete information.

Following are sample connection string formats for Net-Library in the SQL Server section of *win.ini*:

TCP/IP

```
moliere=wdbnovtc,130.214.121.96,4096,urgent
```

or

```
moliere=wdbnovtc,moliere,4096,urgent
```

where:

- The port number (4096) used in the Net-Library string is in decimal (equals 1000 in hex)
- The *urgent* parameter is for out-of-band data support.

IPX/SPX

```
moliere=wdbnovsp,0131,1,83bd,urgent
```

where:

- 0131 is the network address,
- 1 is the node number,
- 83bd is the default IPX port number
- The *urgent* parameter is for out-of-band data support.

DECnet

```
decsyb=wdbdecde,1.33,239
```

where:

- 1.33 is the DECnet address, and
- 239 is the object number (range 128 to 255).

Use **sethost** *DECnet address* to see if the network connection is there.

Out-of-band data is not supported.

Named Pipes

```
camus=dbnmp3,\\camus\pipe\sql\query
```

or:

```
camus=dbnmp3,\pipe\sql\query
```

(for standalone client/server PC)

where:

- *camus* is the machine name
- *\pipe\sql\query* is the local pipe.

Do not use the machine name (\\camus) for a standalone client/server.

The *urgent* parameter is not required because there is implicit out-of-band data support.

Use **net view** to check whether SQL Server is listed and available.

Troubleshooting Windows Network Software

Problem

When you test the SYBASE connection, you get an error message that the server is unavailable or does not exist. Or, when you try to do a **wdbping**, the PC client hangs

If you are having trouble with the **wdbping** command, the following sections may help you to identify and fix the problem. If you are unable to solve the problem, see “PC Connectivity Checklist” starting on page 8-2 before your primary site contact calls Sybase Technical Support.

➤ ***Note***

Troubleshooting **wsybping** for 10.0 release Net-Library products is covered in the *Open Client/Server Supplement for Windows*.

Action for Pre-10.0 Releases

1. Check the *win.ini* file and make sure the correct Net-Library Dynamic Link Library name is specified for the network type and that the connection information required for that type of network is correct.
2. Check the *win.ini* file and make sure that the pipe address or socket is correct. The port address in *win.ini* should be the query port address of the server to connect to.

Checking *sytem.ini* for necessary entries

Use this chart to determine what the *system.ini* file in Windows 3.1 requires. The chart also shows what directories you need to have in your path to ensure a successful connection from PC Windows.

Table 8-8: Network hints, pre-10.0 releases

Network	Check *system.ini* For

FTP Network Software for DOS	`[386Enh]`
	`device=c:\ftp\vpctcp.386`
	Make sure that *pctcpapi.dll* is in a directory in your path statement and that *vpctcp.386* is present.
	NOTE: Works in Enhanced mode only.

Digital DECnet	`[boot]`
	`network.drv=LanWORKS.drv`
	`[386Enh]`
	`network=vdnet.386,vnetbios.386, *dosnet,`
	Make sure that *lanworks.drv* is in a directory in your search path.

Microsoft LAN Manager	Use **setup.exe** and choose LAN Manager version.

Microsoft TCP/IP	`[386Enh]`
	`TimerCriticalSection=5000`
	`UniqueDOSPSP=TRUE`
	`PSPIncrement=2`

Novell LAN WorkPlace	`[386Enh]`
	`device=c:\xln\bin40\vtcpip.386`
	Make sure that *wlibsock.dll* is in your path statement and that *vtcpip.386* is present.

Table 8-8: Network hints, pre-10.0 releases (continued)

Network	Check *system.ini* For

Novell NetWare 386	Use **setup.exe** and choose Novell NetWare 3.26 or later.
	`[boot]`
	`network.drv=netware.drv`
	`[386Enh]`
	`network=*vnetbios, vnetware.386`
	Windows **setup.exe** copies *netware.drv* and *vnetware.386* to the *\windows\system* directory so be sure it is included in the path.
	Make sure *nwconn.dll*, *nwcore.dll* and *nwipxspx.dll* are included in your path, or if you are running release 10.0, just *nwnetapi.dll*.
	NOTE: If you are not running Windows in Enhanced mode, you must run *tbmi.com*.

Wollongong PathWay	Make sure that *sockdos.dll* is in a directory in your path statement. NOTE: Works in Enhanced mode only.

Action for 10.0 and Later Releases

1. Verify that an entry exists in *SYBASE_root:\sql10\ini\sql.ini* for the master or query connection you are attempting.
2. Ensure that the connection address information matches the connection format for the appropriate network software. Port numbers must be valid, pipe addresses must be correct, etc.
3. Ensure that the network software is installed.
4. Verify that an entry for the Net-Library driver is properly listed in *SYBASE_root:\sql10\ini\libtcl.cfg*.

TCP/IP Network Tips

Novell LAN WorkPlace

Check all of the following that apply to your site.

Important files for Novell LAN WorkPlace 4.1 include *vtcpip.386* and *wlibsock.dll* (Windows will cause a general protection fault if *vtcpip.386* does not exist or is not in the 386enh section of *win.ini*).

If you are using TCP / IP for NLM SQL Server, convert the IP port number from hex to decimal in the connection string on the client side. The value in the **interfac** editor for the IP Port Number is in hex. 1000 (hex) equals 4096 (decimal). The IP port number used in the Net-Library is in decimal.

Check to ensure that device *vtcpip.386* is installed in file *system.ini*:

1. Open *system.ini* with a file editor and go to the "386enh" section. Look for *vtcpip.386*. If it is not there, add this entry to the bottom of the "386enh" section:

   ```
   device=c:\path_value\vtcpip.386
   ```

   path_value is the path to the PC network software, for example *C: \ XLN \ bin \ vtcpip.386*.

2. Restart Windows and try the **wdbping** again. If this does not solve the problem, continue to "Check Connection to Novell NetWare SQL Server" starting on page 8-21.

Check Connection to Novell NetWare SQL Server

If you are trying to communicate to an NLM SQL Server, check the connection string in the *win.ini* file. If the port number is 1000, change it to 4096. For example, the *win.ini* entry may look like this:

```
DSQUERY = WDBNOVTC, 123.234.12.35, 1000
```

Change "1000" to "4096," the correct decimal version. Users often mistakenly enter the hexadecimal version instead. You can verify that this entry is correct on the SQL Server side by looking at the **interfac** file on the file server. See the last item in "Novell IPX / SPX Network Tips" starting on page 8-22 for instructions.

FTP TC/PIP

Important files for FTP PC / TCP Version 2.3 are *pctcpapi.dll*, *pctcp.ini*, and *vpctcp.386*.

Use **ethdrv.exe -B** (for Ethernet) for out-of-band data RFC 793 support (the default is RFC 1122).

Check that the user has installed *vpctcp.386*. See the first item in "Novell LAN WorkPlace" starting on page 8-20 for instructions (remember to substitute *vpctcp.386* for *vtcpip.386*).

Microsoft TCP

Check that *sockets.exe* or *socktsr.exe* is installed in the network software:

1. Look for the *sockets.exe* file or the *socktsr.exe* file in the subdirectory that contains the software for the network. Add a line to the *autoexec.bat* file as follows:

   ```
   c:\path_value\sockets
   ```

   where *path_value* is the path to the PC network software; for example *C: \ lanman.dos \ netprog \ sockets*.

2. Retry **wdbping**.

Novell IPX/SPX Network Tips

Check all of the following items that apply to your site:

- Is the client connected to the Novell server? Is it using IPX or IPXODI?
- Check the output of **slist** or **nlist** to verify the network address and node number.
- **rconsole** to the Novell server, load **sybinst**, and choose utilities and the **interfac** editor. Verify the values of **spx** and **ipx** port number.
- Check whether the *ipx.com* version is 3.1 or later (use **ipx -i** to find the version information).
- Run **isql** from the Novell prompt to see if NLM SQL Server is up and running.
- Check to see if all the NLM's are there and are of the correct size and date. Use modules at the Novell prompt for listing all the loaded NLM's.
- 386enh section in *system.ini* should have:

  ```
  network=*vnetbios,vnetware.386,vipx.386
  ```

- In the Windows setup screen, the network option should be:

  ```
  novell netware (shell version 3.26 and above)
  ```

- *netapi.dll*, *nwnetapi.dll*, and *nwipxspx.dll* are Novell files. Get them from CompuServe's Novfiles forum: the files are *winup9.exe* and *dosup9.exe*.

Generating a New ipx.com

Check the version of the *ipx.com* (`ipx -i`), it must be 3.1. If it is 3.1, continue to “Check IPX Internal Network Number” starting on page 8-23.

If the version is 3.02, you must create a new IPX with the **wsgen** utility supplied by Novell:

1. Search in the *windows/system* subdirectory for *ipx.obj*.
2. If *ipx.obj* is not there, select the Setup icon in the Main window.
3. Select Novell NetWare 3.26 and above for the network. The setup utility will then prompt you to insert diskette number 2 to copy over the *ipx.obj* file into the *windows/system* subdirectory.
4. Use the Novell **wsgen** utility to generate an *ipx.com* file from the *ipx.obj* file (your Novell system administrator should be familiar with this utility). The utility will ask questions about the adapter card.
5. After generating the *ipx.com*, check *autoexec.bat* to make sure it will load *ipx.com*. Note that Novell suggests that customers use *ipxodi.com* rather than *ipx.com*.
6. Try **wdbping** again. If this does not solve the problem, continue to “Check IPX Internal Network Number.”

Check IPX Internal Network Number

Check that the IPX internal network number has an even number of digits. Following is a sample of the connection string in the *win.ini* file located in the SQL Server section of the file:

```
DSQUERY = WDBNOVSP, 123, 1, 83bd
```

The IPX internal network number “123” is three digits long. Change it to an even number of digits, for example:

```
DSQUERY = WDBNOVSP, 0123, 1, 83bd
```

Save the file and try **wdbping** again, it should work. If not, continue to “Verify IPX Internal Network Number and Node Number” starting on page 8-24.

Verify IPX Internal Network Number and Node Number

Verify that the IPX internal network number and node number are correct for the file server in which the SQL Server is installed:

1. Type **slist** at the DOS prompt. This command lists the available file servers on the system.
2. Load the *SYBINST.NLM* file at the console prompt:

   ```
   :LOAD SYBINST
   ```

3. Select option 3 (Utilities) and then option 1 (Interfac Editor) to display the **interfac** file entry. The following information should appear:
   - Server name
   - IPX Address
   - IP address (Blank if the site is using only IPX)
   - IP Port number 1000
   - IPX Port number 83bd
4. Check to be sure that this information matches the information on the client side by comparing it to the DSQUERY line in the *win.ini* file (see “Check IPX Internal Network Number” starting on page 8-23 for more information).

Troubleshooting Windows Clients and SQL Server

This section contains answers to some common questions about Windows and SQL Server connectivity. For troubleshooting information about **wdbping**, see "Troubleshooting Windows Network Software" starting on page 8-18. For information about **wsybping**, please refer to your *Open Client/Server Supplement for Windows*.

Table 8-9: Windows connectivity—frequently asked questions

Question	Answer
Are **isql** and **bcp** utilities available on PC Windows platforms?	Neither **isql** nor **bcp** is available for pre-10.0 release PC Windows clients. For 10.0 and later releases, **wisql** is available; **wbcp** is currently in beta. Some Open Client *Release Bulletins* contain an incorrect statement that such utilities exist for pre-10.0 releases.
Why doesn't my 4.2.5 DB-Library application run from a PC Windows client?	Use the sample program *sqltest3.c*, provided in the directory *\sql\sample\dblib* with Open Client for Windows, to verify your development environment. To compile and link *sqltest3.c* using the *sqltest3* makefile, execute a command similar to: `nmake sqltest3` There is no equivalent sample program for 10.0 and later releases. There are, however, other sample programs.

Why can't I connect from PC Windows clients to the Novell NetWare SQL Server?	Check that you have the latest *nw* files from CompuServe's by checking the date of *nwipxspx.dll*. If it is dated before 1/29/92 you may want to dial in to CompuServe and download *winup9.exe*
	Check that you can connect to SQL Server (the **SQLSRVR** NLM) with the **isql** NLM. To check this, type the following from RCONSOLE on the file server:
	`:LOAD ISQL -Usa -P`
	If this command doesn't work, SQL Server is set up incorrectly or it isn't running.
	Check the SQL Server section in the *win.ini* (or *sql.ini* for 10.x releases) file and look at the "DSQUERY" entry. It should look similar to the following:
	`DSQUERY=WDBNOVSP,0x0131,0x01,83bd`
	The element in this example, 0x0131, is the network address. You can verify that it is correct by running the DOS command **slist**, which prints a network address without the "0x." If you are having trouble, try adding "0x" to the address in the *win.ini* (or *sql.ini*) file "DSQUERY" entry, if it is missing.
	The number following the network address is the node number, also shown by the **slist** command, and it is normally 1.

Troubleshooting Third-Party Windows Applications

If you are working with third-party software such as Q+E or Powerbuilder, or if you have other user applications problems, use this section as a quick reference guide for resolutions.

Check Windows Mode

Check to be sure that Windows is running in 386 Enhanced Mode.

Check Open Client Version

Ensure that the application is using the current version of the Open Client/C DB-Library product (catalog number 10930):

1. Open the File Manager utility in Windows. Select the File menu at the top of the screen and select the search option. In the dialog box, enter the file name *w3dblib.dll*.
2. Check the next line to make sure the search starts from the root directory *C:\* and that the check-box which includes all subdirectories in the search is checked.
3. If more than one instance of the file exists on the hard disk, check the size of all the files. Try using the latest *w3dblib.dll*. If this does not work, then find a *w3dblib.dll* with a size of 167,232K. Make a copy of this file in the *windows\system* subdirectory so the application picks up this file.
4. Change the name of all the other occurrences of *w3dblib.dll*. Third-party vendors sometimes supply their own version of the DLL and often it does not integrate properly with Net-Library.
5. After making these changes, test the application again to see if this resolves the problem.

Access and Visual BASIC and Microsoft ODBC Drivers

If the error message number is -7745, install *instcat.sql* (93834 bytes) on the server.

With System 10 SQL Server, installing these files should not be necessary. **installmaster** installs the necessary catalog stored procedures.

instcat.sql comes with the ODBC disk with Microsoft Access and Visual BASIC. *instcat.sql* must be installed before using Microsoft

Access and Visual BASIC or any other ODBC-supported applications.

sqlsrvr.dll, *odbc.dll*, and *odbcinst.dll* are the files used by the Sybase client for connecting to SQL Server.

Powerbuilder

Before using Powerbuilder version 3.0 with SQL Server, you must run the script *pbsyb.sql* against SQL Server.

Open Server

Things to check include:

- Can the client **wdbping** (**sybping** or **wsybping** for release 10.0) Open Server?
- Can the client **wdbping** (**sybping** or **wsybping** for release 10.0) SQL Server?

Can SQL Server talk to Open Server (through **isql** and so on)?

Errors Common with *wdbping*

The remainder of this chapter gives explanations and remedies for errors that are commonly encountered while debugging network and application connection problems.

Cannot Find <*xxx*> File (or *xxx.dll*)

Files not found include:

- *netapi.dll*, *nwnetapi.dll*, *nwipxspx.dll*, and so on. (You can get these files in *winup9.exe* or *dosup9.exe* from CompuServe in the Novfiles forum.)
- *pctcpapi.dll* (FTP PC / TCP file)
- *wlibsock.dll* (Novell LAN WorkPlace for DOS file)

General Protection Fault

Possible causes include:

- Device driver may not be loaded in the “386enh” section of *system.ini* (for PCTCP, it is *vpctcp.386*; for Novell l LAN WorkPlace, it is *vtcpip.386*).
- Illegal memory operations.

PC Hangs (*wdbping* Does Not Return)

Possible causes include:

- Network problem. Determine the relevant files.
- Multiple DLLs (*w3dblib.dll*) or DLL files from different vendors. Verify the date and size.
- Interrupt conflict. DOS Net-Libraries are TSRs and use interrupt 62h.

rmineti.exe is used only in DOS as the interface between Net-Library (real mode) and protected mode applications built using *emineti.exp*.

Server Is Not Responding or Unable to Open Network Connection

Possible causes include:

1. SQL Server is not installed or is not running. To determine if this is the case, see if you can run **isql** locally, on the machine where SQL Server runs.
2. If you can access SQL Server locally, but not from another machine, you may have network problems. Check to see if **telnet**, **ftp**, and **ping** work.
3. If network utilities such as **telnet**, **ftp**, and **ping** can connect, check for incorrect connection string parameters in *win.ini* or *sql.ini*.

Unable to Load Net-Library <*xxx.dll*> Dynamic Link Library

Things to do include:

- Check path (**libpath** in *config.sys* for OS/2) for this particular DLL.
- Check if the DLL exists in the directory specified.
- Check if the name of the DLL is correct.
- Check whether this is the right DLL for the desired protocol and vendor.
- Check *win.ini*, *os2.ini*, or *sql.ini* for connection string format and syntax.
- Check if all the other DLLs required for the Net-Library are loaded.

9 Mainframe Access Products

This chapter covers various Mainframe Access Products (MAP™), including:

- Open Client / Mainframe
- OmniSQL Access Module for DB2-CICS

It also contains a section on the latest versions of Mainframe Access Products documents.

Current Versions of Mainframe Documents

Make sure your documents are up to date by checking them against these tables. These are the current MAP document titles and versions as of November, 1994.

Table 9-1: Generic MAP documents

Document ID Number	Document Title
70230-01-0300-02	*MAP Overview*
36450-01-0300-02	*MAP Messages and Codes*

Table 9-2: Net-Gateway documents

Document ID Number	Document Title
36820-01-0300-02	*MAP Net-Gateway™ Administration*
36692-01-0300-03	*MAP Net-Gateway Installing and Getting Started for HP-UX*
36680-01-0300-02	*MAP Net-Gateway Installing and Getting Started for SunOS Release 4 (BSD)*
36693-01-0300-01	*MAP Net-Gateway Installing and Getting Started for SunOS Release 5.x (SVR4)*
36690-01-0300-03	*MAP Net-Gateway Installing and Getting Started for IBM RISC System 6000*
36830-01-0300-02	*MAP Net-Gateway Installing and Getting Started for IBM OS/2*
36691-01-0300-01	*MAP Net-Gateway Installing and Getting Started for NCR SVR4*
36694-01-0300-01	*MAP Net-Gateway Installing and Getting Started for NetWare*
34014-01-0300-01	*MAP Net-Gateway Installing and Getting Started for NT*

Table 9-3: Open Client/Mainframe documents

Document ID Number	Document Title
36470-01-0200-03	*MAP Open Client/Mainframe COBOL Programmer's Reference*
36460-01-0200-02	*MAP Open Client/Mainframe PL/I Programmer's Reference*
34012-01-0200-01	*MAP Open Client/Mainframe Installation and Administration for CICS LU6.2*

Table 9-4: Open Server/Mainframe documents

Document ID Number	Document Title
36520-01-0300-03	*MAP Open Server/Mainframe COBOL Programmer's Reference*
36560-01-0300-03	*MAP Open Server/Mainframe PL/I Programmer's Reference*

Table 9-4: Open Server/Mainframe documents (continued)

Document ID Number	Document Title
36510-01-0300-02	*MAP Open Server/Mainframe Installation and Administration for CICS*
34368-01-0300-01	*MAP Open Server/Mainframe Installation and Administration for IMS TM*

Table 9-5: Omni SQL Access Module for DB2 documents

Document ID Number	Document Title
36087-01-1010-01	*MAP OmniSQL Access Module™ Client Guide for DB2*
34245-01-1010-01	*MAP OmniSQL Access Module Installation and Administration for DB2-CICS*
34369-01-1010-01	*MAP OmniSQL Access Module Installation and Administration for DB2-IMS TM*

Open Client/Mainframe Results

Question

What types of results are supported by Open Client / Mainframe?

Answer

Open Client / Mainframe supports only three types of results:

- Regular row results
- Return parameter results
- Return status results

It does not support other results types, such as compute results or message results.

Lowercase Required for *charset* Parameter

When installing the OmniSQL Access Module for DB2-CICS, the value specified for **CHARSET=*parameter*** in the **SYGWMCXL** macro **must** be in lowercase.

For example, enter:

```
CHARSET=us_english
```

not

```
CHARSET=US_ENGLISH
```

If you are using ISPF, do not use CAPS ON when configuring this product.

Changing Host Password from a PC

Problem

If you have an OS/2 requester using CICS OS/2 to communicate with CICS on the mainframe, with security handled by RACF on the host and undefined on the CICS OS/2 gateway, you have to log onto the host machine to change password. However, you would like to change it from the client side.

Response

There are two ways to allow an OS/2 workstation user to change the RACF host password:

- Via a 3270 emulation session
- Via an APPC connection between CICS OS/2 and a host subsystem that supports APPC Option Set 222, Receive Sign-on/Change Password.

This APPC option set is supported by CICS/ESA V3, R3 and is documented in a separate manual, the *CICS/ESA CICS-APPC Password Expiration Management Guide*. It is not supported by CICS/MVS V2 or CICS/ESA V3, R2.1.

A sample transaction documented in the *CICS/ESA CICS-APPC Password Expiration Management Guide* demonstrates how to send a message in an architected format to invoke an architected transaction program within CICS/ESA, which in turn determines whether the RACF password value included in the message has expired.

The reply message sent to the requester program in the OS/2 workstation from this architected transaction program indicates either that the current RACF password supplied has expired or that the password is still valid (or invalid if the password value supplied is incorrect).

The reply message from the transaction program in CICS/ESA to the requester program also includes the current host date and time, the date and time of the last prior successful sign-on, and the date and time when the current password will expire.

CICS/ESA provides APPC password expiration management server code, which does the following:

- Enables users to update passwords on APPC links

- Provides APPC users with additional information regarding their signon status; for example, the date and time at which they last signed on

It is possible to run a PEM requester program on your OS/2 workstation (a sample program is supplied) which can communicate with CICS to change the password for users of CICS OS/2. This PEM requester could not execute under CICS OS/2, since it uses basic APPC conversations. CICS OS/2 is restricted to mapped conversations.

For more information on PEM function, see the *CICS-APPC Password Expiration Management Guide*, SC33-0921. The sample PEM requester code is provided in Appendix A of that document.

Changing a password from the PC using this method is much like changing a password through TSO or CICS during the security signon. The sample PEM requester provides for a signon with new password when the user enters:

```
PEM hostcics pslua lu62ss userid oldpw newpw
```

where PEM is the OS/2 command to invoke the PEM requester, *hostcics* is the CM/2 partner_LU_alias, *pslua* is the CM/2 LU_alias, and *lu62ss* is the CM/2 mode_name.

You may also be interested in IBM Network SignON Coordinator/2. This product is designed to provide a single signon and password coordination for the LAN, Database Manager (or DB2/2) and CICS. If the host CICS is at CICS/ESA V3.3, NSC/2 will use the PEM server code which is part of that CICS release. If the host CICS is at a lower level, NSC/2 can communicate using HLLAPI on top of a 3270 session. The workstation user need not be aware that the 3270 session exists. NSC/2 runs on both DOS and OS/2 workstations. See announcement letter 293-322 for more information.

Use of TDSNDDON in Open Server/Mainframe COBOL Applications

A call to **TDSNDDON** signifies the completion of transmission of a result set to a client. It can be used to indicate that the result set just completed is the final result set or that subsequent result sets will follow. It can provide other information to aid the client end of the conversation in anticipating the actions that will subsequently take place on the connection. Further, **TDSNDDON** can indicate whether or not a row-count for the result set will be conveyed.

The *Open Server/Mainframe COBOL Programmer's Reference* provides the following model call for **TDSNDDON**:

```
01 TDPROC                    PIC S9(9)   USAGE COMP SYNC.
01 RETCODE                   PIC S9(9)   USAGE COMP SYNC.
01 STATUS                    PIC S9(9)   USAGE COMP SYNC.
01 ROW-COUNT                 PIC S9(9)   USAGE COMP SYNC.
01 RETURN-STATUS-NUMBER      PIC S9(9)   USAGE COMP SYNC.
01 CONN-OPTIONS              PIC S9(9)   USAGE COMP SYNC.

CALL 'TDSNDDON' USING  TDPROC, RETCODE, STATUS, ROW-COUNT,
                       RETURN-STATUS-NUMBER, CONN-OPTIONS
```

TDPROC is an input parameter in which the programmer identifies the handle returned from the prior **TDACCEPT** call.

RETCODE is an output parameter wherein the programmer checks the success or failure of this **TDSNDDON** call.

STATUS is an input parameter used to signify whether or not this call is a "final" call, whether or not a *ROW-COUNT* is included, and can be used to indicate that the client's last request contained a logical error from the viewpoint of the Open Server application. The valid representations of *STATUS* are:

• **TDS-DONE-FINAL**	(0x0000)
• **TDS-DONE-CONTINUE**	(0x0001)
• **TDS-DONE-ERROR**	(0x0002)
• **TDS-DONE-COUNT**	(0x0010)

For brevity's sake, we will refer to these representations in this document by their last word: *FINAL*, *CONTINUE*, and so on.

Note that these *STATUS* representations can be combined; the COBOL programmer can add these values together to produce the desired result. For example, you could add *CONTINUE* and *COUNT* to produce a *STATUS* which will convey that a result set is terminated, that a valid *ROW-COUNT* for that result set is included, and that other result sets will follow.

It is important to note that if the low-order (rightmost) position of *STATUS* does not contain a “1” (**CONTINUE -** 0x0001), the **TDSNDDON** will be treated as *FINAL*. When the program executes a **TDSNDDON** *CONTINUE*, subsequent iterations of **TDSNDDON** can be executed, that is, the application remains in “send” state. If the program executes a **TDSNDDON FINAL**, the application switches back to “receive” state. Some further action must be taken at that point to put the application back into “send” state, if that is required. In release 3.0 of Open Server CICS Mainframe COBOL, long-running transactions are supported, allowing the Open Server application to switch between “send” and “receive” states as required to carry on conversations with a client. An Open Server Application is initially placed in “send” state upon successful completion of the **TDACCEPT**.

ROW-COUNT is an input parameter. If you indicated *COUNT* in *STATUS*, this parameter must contain a valid count of rows sent in a just-completed result set. The contents of this parameter are not automatically determined by any other action in the program; the programmer is expected to keep a count of the rows sent and then insert that count in *ROW-COUNT* prior to executing **TDSNDDON**.

RETURN-STATUS-NUMBER is a completion code that can be sent to the client to indicate a **FINAL** status. It cannot be sent with a **TDSNDDON CONTINUE**, only with a **TDSNDDON FINAL**. That is, if a program sends several result sets, each followed by a **TDSNDDON CONTINUE**, it cannot convey a *RETURN-STATUS-NUMBER* with the **TDSNDDON CONTINUE** to indicate a completion status for each result set. Again, no value is automatically inserted in *RETURN-STATUS-NUMBER*; it is the programmer’s responsibility to insert a value in this parameter. For example, after completing a static DB2 SQL request, it is the programmer’s responsibility to insert the SQLCODE return status in *RETURN-STATUS-NUMBER*, if that is desired.

CONN-OPTIONS is an input parameter used to qualify *FINAL* and *CONTINUE* usages of **TDSNDDON**. The possible entries are:

• **TDS-ENDREPLY**	(1) (new in release 3.0)
• **TDS-ENDRPC**	(3)
• **TDS-FLUSH**	(7)

In release 2.0, sending **TDSNDDON FINAL** with *CONN-OPTIONS ENDRPC* meant that the Open Server application had completed all tasks and was about to execute a **TDFREE** to place the application in “reset” state just before coming to the end of the program.

ENDREPLY is a new release 3.0 *CONN-OPTIONS* value that allows the Open Server application to convey to the client that although a **TDSNDDON FINAL** has been sent, the application will remain in “receive” state awaiting another request from the client. **CONN-OPTIONS ENDREPLY** and **ENDRPC** can only be used with **TDSNDDON FINAL**. **CONN-OPTIONS FLUSH** is used only with **TDSNDDON CONTINUE** and confirms that all prior output rows should be flushed across the connection before this **TDSNDDON** flows.

Return Parameters and Return Status in Open Server/MF COBOL Programs

Open Server / Mainframe release 3.0 (CICS) supports an application program's ability to pass return parameters and a return status back to the invoking client. This capability was also supported in release 2.0, but that implementation required the client to load return parameters with a value appropriate to the datatype before conveying the return parameters behind an RPC call.

The following stored procedure example illustrates the use of return parameters and return status when invoking an RPC with Open Server / Mainframe release 3.0:

Figure 9-1: Sample code for Open Server/Mainframe COBOL RPC invocation

```
create proc retparm_example as
declare @p1 char(4), @p2 int, @p3 char(8),
   @pret4 int, @pret5 char(4), @retstatus int,
   @ans char(4)
select @p1 = “AAA”
select @p2 = “123
select @p3 = “BBB”
exec @retstatus = gwname...rpcname @p1, @p2, @p3,
@pret4 output, @pret5 output
select @ans = convert(char(8),@retstatus,1)
print @ans
select @ans = convert(char(8),@pret4,1)
print @ans
print @pret5
return
```

This stored procedure passes five parameters. Note that the fourth and fifth parameters (@pret4, @pret5) are used as return parameters; they are declared, but no values need be inserted into them. The return status parameter (@retstatus) is declared without a value and is not passed; its use in the **exec** statement positions it to receive a return status passed back from the RPC. The **exec** statement passes the five parameters, with the fourth and fifth marked as “output.” The rest of the stored procedure simply prints the return parameters and return status for demonstration purposes.

The COBOL application sends back two single-column rows. The first row contains a valid integer (99), the second row contains a NULL integer. This is followed by the message content of a TDSNDMSG. The stored procedure then prints the return status (0), the integer content of @pret4 (50), and the character content of @pret5 (OK):

```
1> retparm_example
2> go
 STUFF                 * Header for one column returned
------
    99                 * First row returned
  NULL                 * Second (last) row returned
MSG: IT WORKED         * TDSNDMSG output
(2 rows affected)      * Row-count sent with TDSNDDON
0                      * Print of @retstatus sent with TDSNDDON
50                     * Print of @pret4 flushed by TDSNDDON
OK                     * Print of @pret5 flushed by TDSNDDON
(return status = 0)
```

The COBOL program flow that produced this output is shown in the following table:

Table 9-6: Sample program flow for Open Server/Mainframe COBOL RPC invocation

Action	Purpose
PROCEDURE DIVISION.	
CALL 'TDINIT'	
CALL 'TDACCEPT'	
CALL 'TDRESULT'	To check if parameters are present.
CALL 'TDNUMPRM'	To determine the number of parameters received. Knowing that five parameters were received, and the purpose of each parameter, this example just executes TDINFPRM against the parameters expected to be return parameters. A more flexible program might examine each parameter with TDINFPRM.
CALL 'TDRCVPRM'	To receive the first (input) parameter.
CALL 'TDRCVPRM'	To receive the second (input) parameter.
CALL 'TDRCVPRM'	To receive the third (input) parameter.
CALL 'TDINFPRM'	To check / establish the fourth parameter as an output parameter.
CALL 'TDRCVPRM'	To receive the fourth (output) parameter.
CALL 'TDINFPRM'	To check / establish the fifth parameter as an output parameter.
CALL 'TDRCVPRM'	To receive the fifth (output) parameter.
CALL 'TDESCRIB'	As often as necessary, to describe output columns.

Table 9-6: Sample program flow for Open Server/Mainframe COBOL RPC invocation

Action	Purpose
CALL ‘TDSNDROW’	As necessary, to send output rows while counting rows.
CALL ‘TDSNDMSG’	Optionally, to describe results.
CALL ‘TDSETPRM’	To prepare an int return parameter (50) and insert value.
CALL ‘TDSETPRM’	To prepare a character return parameter (OK) and insert value.
CALL ‘TDSNDDON’	With TDS-DONE-FINAL + TDS-DONE-COUNT status, TDS-ENDRPC conn option. TDSNDMSG, row-count and the two return parameters will precede TDSNDDON indication in TDS flow.
CALL ‘TDFREE’	To reset the connection.

SQL Server Administration Tools

10 SQL Monitor Server

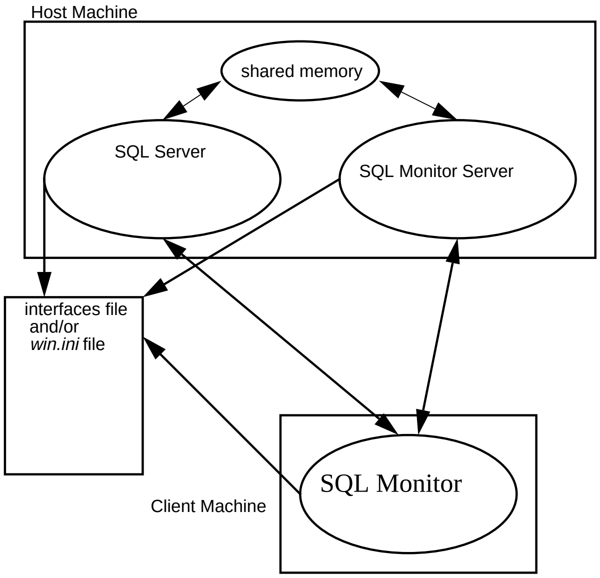
This chapter provides general troubleshooting information about SQL Monitor Server™ and SQL Monitor Server Client, and lists error messages that may appear in the SQL Monitor Server error log.

Overview

SQL Monitor is a performance monitoring tool for SQL Server. It is an Open Client / Open Server Graphical User Interface (GUI) application. It allows you to monitor device I / O, processes, transactions, SQL Server cache, and so on. The data that SQL Monitor displays has been read from the SQL Server's shared memory and therefore contains more information than stored procedures such as **sp_who** or **sp_lock** can provide.

SQL Monitor consists of two parts: SQL Monitor Client, written in Open Client, and SQL Monitor Server, which is an Open Server application. There are different versions of SQL Monitor Client and SQL Monitor Server for the different SQL Server versions on individual operating systems. Refer to the following illustration for communication routes between SQL Server and SQL Monitor.

SQL Monitor Map



Determining Software Releases

Since release compatibility is crucial to resolving many problems occurring with SQL Monitor, the following shows how to determine the releases of the software you are running:

Table 10-1: How to determine software releases

To Get the Version Of	Type	From
SQL Server	**select @@version**	Within **isql** Version is also found at the top of the SQL Server error log.
SQL Monitor Server	**monserver -v**	UNIX command line
	monserver/version	VMS command line Version is also found at the top of the SQL Monitor log file.
SQL Monitor Client	**sqlmon -sv**	UNIX command line
	monserver/version	VMS command line
	select "About" from the Help Menu	PC

SQL Server and SQL Monitor Server Compatibility

Following is a compatibility matrix for SQL Server and SQL Monitor Server:

Table 10-2: SQL Server and SQL Monitor Server compatibility

SQL Monitor Server	Monitors	SQL Server	Condition
4.9.x	Yes	4.9.x	None
4.9.x	No	10.0.0	This may work but is not certified.
4.9.x	No	10.0.1	
10.0.0	Yes	4.9.x	While starting SQL Monitor Server, the SYBASE environment variable must point to a System 10 SQL Server installation.

Table 10-2: SQL Server and SQL Monitor Server compatibility (continued)

SQL Monitor Server	Monitors	SQL Server	Condition
10.0.0	Yes	10.0.0	None
10.0.0	No	10.0.1	None
10.0.1	No	4.9.1	None
10.0.1	No	4.9.2	None
10.0.1	No	10.0.0	None
10.0.1	Yes	10.0.1, 10.0.2	None

Starting SQL Monitor Server

The following sections describe issues that may arise when you start SQL Monitor Server on UNIX, VMS or PC Windows platforms.

SQL Monitor Start-Up Steps

The order in which to start the three applications involved in monitoring your SQL Server is:

1. SQL Server
2. SQL Monitor Server
3. SQL Monitor Client

Anytime SQL Server is restarted, both SQL Monitor Server and SQL Monitor Client must be restarted, too.

➤ ***Note***

A one-to-one correlation exists between SQL Servers and SQL Monitor Servers. Only one SQL Monitor Server can monitor one SQL Server.

Permissions Needed

A user must have **sa_role** privilege to access release 10.0 or 10.0.1 SQL Monitor Server or SQL Server. Enter the following command to grant that privilege to a user:

sp_role “grant”, sa_role, *username*

where *grant* lists the privilege and *username* is the name of the user. For example,

sp_role “grant”, sa_role, kate

grants the role of System Administrator to Kate.

UNIX Tips

The process is as follows:

1. Start the SQL Server as user “sybase”. Use the **-M** flag to specify the location of the *server_name.krg* file. (See the *SQL Server Installation Guide* for your platform.) For example:

   ```
   setenv SYBASE /usr/local/system10
   setenv DSLISTEN SYBASE10
   $SYBASE/bin/dataserver -\
   d/sun4db/system10db/SYBASE10_master.dat
   -e$SYBASE/install/errorlog_SYBASE10 -M$SYBASE &
   ```

2. Start up the SQL Monitor Server as the same “sybase” user, on the same host machine, with **no spaces** between the flag and the flag’s value. Using the **-i** flag, point to an interfaces file that has both the SQL Server entry and the SQL Monitor Server entry; specify the same location of the SQL Server *.krg* memory file using the **-m** flag. Be sure to start SQL Monitor Server on the same machine as SQL Server because SQL Monitor Server has to access the same shared memory region.

   ```
   $SYBASE/bin/monserver -MMON10_SYBASE10 -SSYBASE10 -\
   Usa_role -Ppassword -i$SYBASE/interfaces -\
   l$SYBASE/install/MON10_SYBASE10.LOG -n -m$SYBASE -\
   O &
   ```

   The two flags, **-M** and **-m**, have different meanings. See the *SQL Monitor Server Supplement* for complete start-up instructions

3. Start the client as any user who has been granted the “sa” role on any client machine, **using spaces** between each flag and its value. The **-i** flag must point to an interfaces file that has both the SQL Monitor Server entry and the corresponding SQL Server entry that was passed on the **monserver** command line. The SQL Server name **must** be the same name that appears in the interfaces file. It cannot be an entry that has the same host and port number but a different SQL Server name.

   (See the *SQL Monitor Release Reference Manual* for complete start-up instructions.)

```
/usr/directory/bin/sqlmon -M MON10_SYBASE10 -U sa_user -\
P password -I /usr/directory/interfaces &
```

➤ ***Note***

When you start SQL Server in UNIX using the **-M** flag to specify a location of the shared memory file that is different from the default of *$SYBASE*, you also need to start SQL Monitor Server with the **-m** flag specifying the same directory location. That way, SQL Monitor Server knows where to find the shared memory file it is reading. If you have more than one SQL Server on your host machine, it is especially important that each SQL Monitor Server points to the correct memory file.

Windows Tips

Enter the following:

```
C:\SERVMON\SERVMON.EXE
```

The *win.ini* file must have entries for both the SQL Monitor Server and the SQL Server that were indicated on the command line for the SQL Monitor Server. They must use the same names that were used to start the SQL Monitor Server.

OpenVMS Tips

In the VAX OpenVMS environment, SQL Monitor Server name should be less than or equal to 11 alpha-numeric characters. If you use a name that is greater than 11 alpha-numeric characters in length, SQL Monitor Server will not start.

Starting SQL Monitor Client

The following sections describe issues that come up when you start SQL Monitor Client.

Performance Changes During SQL Monitor Client Invocation

When you start SQL Monitor Client, it issues a **dbcc memusage** command to SQL Server to get information about how memory is allocated. Depending upon its size and configuration, SQL Server's performance may be degraded during this initialization process.You can bypass the above process by using the **-nomem** command line option:

UNIX

```
sqlmon -U sa -P password -M monitor_server_name -nomem
```

➤ ***Note***

When you use the **-nomem** option, UNIX SQL Monitor Clients will not provide memory allocation information on the start-up screen.

Windows

For PC Windows, do not check the box “obtain memory allocation.”

➤ ***Note***

Nothing will be displayed if you open the “memory allocation” window when you have not checked the “obtain memory allocation” box.

UNIX Password

If you run the **sqlmon** utility from the command line, SQL Monitor Client reads a start-up file called *Sqlmonitor* in your home directory. For security reasons, you cannot hard code the “sa” password in the start-up file. To specify the “sa” password, use the following SQL Monitor command line option to run SQL Monitor Client:

```
sqlmon -U sa_role -P password -M monitor_server_name
```

Alternatively, you can write a shell script to accept the user name and password and then run the **sqlmon** command within the shell script.

UNIX Command Line Help on *sqlmon*

To get command line help and display the options that may be specified when running **sqlmon**, type:

```
sqlmon -h
```

Shutting Down SQL Monitor Server

If the SQL Server being monitored goes down at any time, SQL Monitor Server automatically shuts itself down and returns the following message:

```
SQL Server seems to be down. Shutting down SQL
Monitor.
```

Once SQL Server is down, SQL Monitor Server cannot capture its activities. Therefore, SQL Monitor Server periodically checks for the existence of the SQL Server it monitors. The frequency of this check is called the monitor interval. This interval can be changed, if desired. Refer to the section "SQL Server seems to be down" in the "Common Error Numbers and Messages" section of this chapter.

Restarting SQL Monitor Server or SQL Monitor Client

You need to restart SQL Monitor Client after either SQL Server or SQL Monitor Server has been restarted. This is because the SQL Monitor client makes a connection to the SQL Server to get process and database identification information. Also, if the SQL Server goes down for any reason, the SQL Monitor Server needs to be restarted. This is because SQL Server allocates new shared memory each time it starts. You must restart SQL Monitor Server to stop monitoring the old shared memory and start monitoring the shared memory that has been newly allocated by SQL Server.

Normally, SQL Monitor Server will shut itself down when SQL Server is taken down. However, if SQL Server is immediately restarted, SQL Monitor may not recognize that SQL Server went down. In that case, SQL Monitor Server should shut down via **isql** as described in the *SQL Monitor Server Supplement*.

Then follow the order and instructions for starting SQL Monitor Server and SQL Monitor Client, as described earlier in the chapter.

Using SQL Monitor Server

The following sections describe issues that come up when using SQL Monitor Server.

Identifying SQL Server for a SQL Monitor

To find out which SQL Server a SQL Monitor is monitoring, run **isql** against SQL Monitor Server, and then use the **msv_status server** command to determine the name of the SQL Server that SQL Monitor Server is monitoring. Following is an example:

```
% isql -Usa -Ppassword -Smonitor_server_name

1> msv_status server
2> go
```

This command will print the name of the SQL Server being monitored.

Monitoring Multiple SQL Servers

One SQL Monitor Server can only monitor a single SQL Server. If you want to monitor multiple SQL Servers, then you must run multiple SQL Monitor Servers, each one monitoring one SQL Server. Similarly, each SQL Monitor Client can only connect to a single SQL Monitor Server at a time.

Using SQL Monitor Client

Connecting from a SQL Monitor Client

UNIX

The SQL Monitor Client connection you specify when starting the client should be to the SQL Monitor Server, although the Monitor Client also connects to the SQL Server. This is why both the SQL Monitor Server and the SQL Server entries must be in the same interfaces file found in the SYBASE root directory or the directory specified with the **-I** flag.

Windows

If there is no appropriate SQL Server entry in the PC *win.ini* file, then SQL Monitor Client will use the DSQUERY entry, if it exists, and may connect to the wrong SQL Server.

These are the steps for connecting from SQL Monitor Client to SQL Server:

- You specify the SQL Monitor Server name when connecting or logging in.
- SQL Monitor Server will be queried for the name of SQL Server.
- SQL Monitor Client will log on to SQL Server.
- The *win.ini* file must contain an entry for the SQL Server that matches the entry in the interfaces file used to start SQL Monitor Server.

Windows: Network Connection Limitations

SQL Monitor Client opens up two network connections for each open window: one to SQL Monitor Server and one to SQL Server. PC users in particular should be aware of this fact since PC network software is often configured for a limited number of connections.

For example, if you are using FTP's PC/TCP, with its default configuration of six network connections, this limit will be reached with three SQL Monitor windows. If you are running other applications that require network connections, then the limit will be reached sooner. An attempt to open another SQL Monitor window returns the following series of error messages:

```
Unable to connect: SQL Server is unavailable or
does not exist.

You may have exceeded the maximum number of
network connections configured for this machine.

Cannot access Monitor Server.
```

You may also see these error messages when the limit of open connections for the SQL Monitor Server has been exceeded. In that case, restart the SQL Monitor Server with a larger **-n** value (the default value is 5).

To increase the number of PC network connections, using the FTP PC/TCP example, you must edit the "tcp-connection=" entry in the [pctcp kernel] section of the *\PCTCP\PCTCP.INI* file and then restart your PC. Please refer to your network documentation for specific information on how to increase your network connections.

How Do I Save the Sampled SQL Monitor Statistics?

At present, it is not possible to save the SQL Monitor statistics because they are overwritten. However, you may take a screen snapshot of the graphs.

SQL Monitor and the Event Buffer

Some of the information provided by SQL Monitor is obtained from counters that SQL Server maintains in shared memory. SQL Monitor Server reads this information from shared memory and then makes it available to SQL Monitor Client for display. Other information, however, is not available via the counters maintained by SQL Server. The information not available from counters is:

Table 10-3: Information obtained from the event buffer

Information	Client Window Where Displayed
Page I/O	Cache
Data cache hit rate	Performance Summary
Data cache hit rate	Performance Trends
CPU usage and page I/O rate	Process Activity
Reads and writes	Process Detail
Page I/O	Process List

To obtain this information, SQL Monitor must tell SQL Server to save information about these events in a separate event buffer that can be read by SQL Monitor Server.

Whenever SQL Monitor Client displays any of these windows, it notifies the SQL Server that it must start writing information about these events to the event buffer. If SQL Monitor Client is not displaying any of these windows, then the SQL Server does not save event information and does not use the event buffer.

How the Event Buffer Works

The event buffer is a **wrap-around buffer**; that is, when the buffer is full, the SQL Server wraps around to the beginning of the buffer to write new events, overwriting the previously recorded events. This overwriting of events is not a problem as long as the events have already been read by SQL Monitor before SQL Server overwrites them. The frequency with which the events are read is determined by the sample interval set on SQL Monitor Client.

Several factors contribute to whether events are overwritten or not:

- The amount of page I/O (reads and writes to or from disk and cache) occurring during a given interval

- The size of the event buffer
- The sample interval of SQL Monitor Client

If events are overwritten in the event buffer, SQL Monitor Client will not have complete information to display. For example, it will report smaller values for page I/O and CPU usage than are really the case.

In the case of the Process Detail, Process Activity, and Process List windows, the significance of this discrepancy is lessened because SQL Monitor detects that unread events have been overwritten, and extrapolates what the missing data would have been based on the events that are available to be read. This happens automatically, and SQL Monitor Client provides no indication that event data is being lost or extrapolated.

There are two methods you can use to reduce or eliminate the loss of events. Use them in combination, since their use involves making trade-offs between using more resources in exchange for losing fewer events.

1. Increase the size of the event buffer
2. Decrease the sample interval of SQL Monitor Client.

Increasing the Size of the Event Buffer

The default size of the event buffer is 100 events (where each event requires 100 bytes). If there are a large number of page I/O events occurring and the sample interval of the Monitor Client is not very frequent, events are likely to be lost. They will be overwritten before they have been read. For example, if the sample interval is set to 15 seconds, then if more than 100 reads and writes occur in 15 seconds, events will be lost.

The event buffer size can be increased to allow more space for events to be recorded. The *SQL Monitor Server Supplement* gives instructions about using **buildmaster** to change the configured size of the **ceventbuf** parameter. The *Supplement* also gives instructions for determining a size for the event buffer, but this calculation should only be used as an upper limit since it simply calculates a value that is 5 percent of the combined size of the procedure and data caches.

A more appropriate method for determining the buffer size is to use the *sp_monitor* stored procedure to count the amount of page I/O during an interval equal to the Monitor Client's sample interval. To use *sp_monitor* in this way, follow these steps:

1. Run **sp_monitor** once to initialize its counters.
2. Wait for a period equal to the desired sample interval.
3. Run **sp_monitor** again and note the values in parentheses under the *total_read* and *total_write* fields: these are the number of reads and writes that occurred in the period since **sp_monitor** was last run.

The total reads and writes (page I/O) gives an indication of how many events would be written to the event buffer while the Monitor Client was monitoring this activity. Of course, this test should be run several times, and during periods of peak activity.

Trade-Off Between Event Buffer Size and Procedure Cache

Another consideration that must be taken into account when changing the size of the event buffer is the fact that event buffer memory is allocated from the memory that has been set aside for the procedure cache.

Smaller SQL Servers that have a small procedure cache may not have enough memory to run some stored procedures if you create too large an event buffer. In the case of an 8MB SQL Server, the event buffer should not be any larger than 1000; otherwise, there will not be enough procedure cache to run some of the larger system stored procedures. (You may need an even smaller event buffer size if you have large user-defined stored procedures.)

For larger SQL Servers, an event buffer of 2000 is probably a good average size; however, on SQL Servers where there is a large amount of activity, that value should be confirmed by using **sp_configure** during periods of peak activity.

If you must significantly increase the size of the event buffer, then you should consider using **sp_configure** to increase the overall size of the SQL Server or the percentage of cache dedicated to procedures so that there is sufficient procedure cache available.

Decreasing the Sample Interval of SQL Monitor Client

Since the loss of events is due to the fact that event data is not read before it is overwritten, another way to avoid event loss is to increase the frequency of the sample interval so that the event data is read more frequently.

Be aware, however, that increasing the sample interval has the side-effect of increasing the amount of network traffic between SQL Monitor Server and SQL Monitor Client as well as increasing the overall workload of the SQL Monitor Server and SQL Monitor Client processes.

Common Error Numbers and Messages

The remainder of this chapter documents some error numbers and messages that may appear in the SQL Monitor Server error log or be displayed by SQL Monitor Client. Possible causes and resolutions follow each error message.

Error 16008

Error Message Text

```
Open Server: Fatal Error: 16008/20/0: Memory
allocation for ‘143680’ bytes failed in ‘srv_init’
for ‘msgpool’ allocation.
```

Explanation

SQL Monitor Server was unable to allocate needed memory because of insufficient available VMS memory, insufficient quota values in the quota file, insufficient quota limits for the user account that is starting SQL Monitor Server, or insufficient system-wide quota settings.

Action

Adjust the page file quota. See the discussion of quotas under “Create Quota Specification File” in the *SQL Monitor Server Supplement*.

Error 16011

Error Message Text

```
Fatal Error: 16011/20/0: Bad Interface file.
```

Explanation

The entries in the interfaces file are in a format not recognized by the Monitor Server.

Action

Refer to the *System Administration Guide Supplement* or the *SQL Monitor Server Supplement* for information about the correct format of the interfaces files. Be sure that both SQL Server and Monitor Server entries have both a master and query line that are preceded by a single tab character and which share the same port number. Make sure all comments and blank lines are between entries.

Error 16012

Error Message Text

```
No server log file open; Using stderr for log.
Open Server:  Error: 16012/10/0: Can't open log
file file '$SYBASE/install/ms.log'

Openserver srv_init call failed.
Monitor Server has encountered a fatal error and
is quitting
```

Explanation

The user starting SQL Monitor Server does not have permission to write to the log file in the indicated directory.

Action

Start SQL Monitor Server as a user that has permission to read and write the SQL Monitor Server error log file in the indicated directory. If the log should be created in a different directory, use the **-l** flag for the **monserver** command in the script used to start SQL Monitor Server.

Error 16029

Error Message Text

```
Open Server: Fatal Error: 16029/20/0: Failed to
start any network listeners
```

Explanation

A user either attempted to start SQL Monitor Server while it was already up and running or specified a port number in the interfaces file that is already in use by another application.

Action

Enter the following to try to connect to SQL Monitor Server:

isql

OR:

Use the UNIX **ps** command to see if SQL Monitor Server is already up.

OR:

Use the following command:

```
netstat -a | grep port_number
```

where *port_number* is the port you have designated for SQL Monitor Server. If the command returns results, then another process is already using that port.

If another process is using the SQL Monitor Server port number:

1. Change the port number in the interfaces file.
2. Restart SQL Monitor Server.

Error 16104

Error Message Text

```
Warning: 262144 bytes allocated for process buffer
instead of 524288 bytes
Warning: 65536 bytes allocated for process buffer
instead of 524288 bytes
Open Server:  Error: 16104/10/1: Unable to
allocate stack, size 34816, for new thread
```

Explanation

The starting address of the shared memory segment that SQL Monitor Server is sharing with SQL Server is too low. It is not a fatal error for SQL Monitor Server, but additional SQL Monitor Client windows cannot be opened.

Action

Start SQL Server with a higher virtual memory starting address, which allows SQL Monitor Server to do the same. This allows more SQL Monitor Client connections to SQL Monitor Server. For example:

1. Shut down SQL Server.
2. Set **buildmaster -dmaster_device -yckmemrange.mrstart=163577856** [1]
3. Restart SQL Server.
4. Restart SQL Monitor Server.
5. Restart SQL Monitor Client.

The problem is independent of the amount of memory SQL Server as dictated by the **sp_configure memory** command. It is dependent on the amount of memory available between the size of the SQL Monitor Server binary and the starting address of shared memory.

[1] The mrstart number may differ on different platforms.

Error 16240

Error Message Text

```
OpenServer: Fatal Error: 16240/20/0: Net-Library
routine net_address_get() failed in srv_start_net
Network error: status = 112 - Specified server
name attribute could not be found.
```

Explanation

(System 10 only)

SQL Monitor Server cannot find an entry in the interfaces for SQL Server and / or SQL Monitor Server.

Action

Check the interfaces file (specified by **-i** on the **monserver** command line, or by the default *$SYBASE/interfaces* file) to be sure that there is an entry for SQL Server that contains at least a query line and an entry for SQL Monitor Server that contains at least a master line.

Error Message Text

```
OpenServer: Fatal Error: 16240/20/0: Net-Library
routine net_dict_open() failed in
srv_open_dictionary

Network error: status = 111 - Could not find
addressing dictionary.
```

Explanation

SQL Monitor Server cannot find or open either the *$SYBASE/interface* file or the file specified by **-i** on the **monserver** command line.

Action

Check the path and existence of the file, as well as the read permissions.

Can’t Open Log File

Error Message Text

```
Can’t open log file
```

Explanation

SQL Monitor Server creates a default log file called *ms.log* where the **RUN** script for SQL Monitor Server was executed. This error occurs when some operation tries to open the log file. This might happen, for example, if you choose to bring up SQL Server and SQL Monitor Server from an *rc* file so that, whenever the system is restarted, the SQL Server and SQL Monitor Server automatically come up.

Action

If you encounter this error in this context, make sure you change the operating system userid from *root* to *sybase* before bringing up SQL Server and Monitor Server. Also, the **RUN** script for SQL Monitor Server should have the **-l***$SYBASE/install/errlog_MON_SRV_NAME* option set to write the log file to a directory where the “sybase” login account has write permission.

Could not create shared memory region for Monitor Server's use

Error Message Text

```
Warning: Attempt to create shared memory at
addr=0x7d000000 failed now trying at
addr=0x7e000000
Warning: Attempt to create shared memory at
addr=0x7e000000 failed now trying at
addr=0x7f000000
Warning: Attempt to create shared memory at
addr=0x7f000000 failed now trying at
addr=0x80000000
Could not create shared memory region for monitor
server's use. Giving up!
Fatal error -1 detected
```

Explanation

SQL Monitor Server was unable to allocate needed memory because:

- Insufficient VMS memory was available
- Insufficient quota values were used in the SQL Monitor Server quota file
- Insufficient quota values are set up in the user's account or
- System-wide quota parameter settings are insufficient

Action

To determine the virtual pages needed for SQL Monitor Server, use the **show proc/continuous/id=00000xxx** command where the *id* is that of the process ID of SQL Server. Observe the *Virtual pages* value in the upper right-hand corner. SQL Monitor Server needs up to 50 percent more than this value. Confirm that the *page_file* quota value in the SQL Monitor Server quota file, the *Pgflquo* quota value of the SYBASE account, and the SYSGEN parameter *VIRTUALPAGECNT*, are set to a value higher than what is needed by the SQL Monitor Server process.

CSLIB c_ctx_alloc

Error Message Text

```
CSLIB c_ctx_alloc: Check whether locales and
charsets files are in place.
```

Explanation

SQL Monitor Server needs the SYBASE environment variable to point to a directory containing the appropriate locales and charsets subdirectories.

Action

Refer to the section “Special Considerations - Locales and Charsets Files” in the *SQL Monitor Server Supplement*.

DBCC TRACEON 3604; timeslice -201

Error Message Text

```
server: DBCC TRACEON 3604, SPID 11
kernel: timeslice -201 current process infected
```

Explanation

After a SQL Monitor Client connects to SQL Server, SQL Server may start refusing connections, as well as displaying the above error message in the SQL Server error log file when SQL Monitor Client attempts to obtain memory information from SQL Server.

Action

Start SQL Monitor Client without retrieving the memory information.

UNIX

For UNIX clients, use the **-nomem** command-line parameter.

Windows

For PC clients, do not select “Obtain Memory Allocation” from the Main menu before you refresh.

Reconfigure SQL Server to correct the timeslice error. Consult with Sybase Technical Support for assistance.

Read from SQL Server failed

Error Message Text

```
Read from SQL Server failed.
Connection reset by peer.
Unable to retrieve data.
```

Explanation

(UNIX client only)

If SQL Server is shut down and SQL Monitor Server remains running, and a user opens a new SQL Monitor Client data window, then SQL Monitor Client displays the above error messages.

Action

Perform the following steps:

1. Shut down SQL Monitor Server.
2. Shut down SQL Monitor Client.
3. Restart SQL Server.
4. Restart SQL Monitor Server.
5. Restart SQL Monitor Client(s).

Whenever SQL Server is shut down and restarted, SQL Monitor Server must also be shut down and restarted.

SQL Server seems to be down

Error Message Text

```
SQL Server seems to be down. Shutting down the
monitor server.
```

Explanation

If SQL Server is down for approximately 2 to 4 minutes, SQL Monitor Server shuts down automatically.

Action

Start up SQL Server, and then start up SQL Monitor Server.

There are instances where the SQL Server may appear to the SQL Monitor Server to be down when in fact it is still running. In these instances when SQL Monitor Server shuts itself down unexpectedly, you may want to increase the time intervals at which SQL Monitor Server checks to see if SQL Server is running. The default interval is 120 seconds. This may be too low for some configurations. To change the interval, use the following **monserver** command line parameter.

UNIX

Although the SQL Monitor default monitoring interval is 120 seconds, you can change this default to *n* seconds minimum, *n* seconds maximum, using 10-minute intervals. One drawback to using a large time interval is that when the SQL Server actually does shut down, SQL Monitor Server continues monitoring and holds onto the SQL Server's shared memory segment. You may need to shutdown SQL Monitor Server manually before you restart SQL Server.

The UNIX parameter is **-L***config_file*, where *config_file* is a file in the current directory. Create a file and use the -L option, adding one line as follows:

```
heartbeat_interval nn
```

where *nn* is the value in seconds.

The following example starts a SQL Monitor Server that performs 10-minute checks to see if SQL Server is running:

```
monserver -Usa_role -Ppassword -MMON_1001 -
SSYB_1001 \
-n5 -O -Lmonserver_config_file &
```

where *monserver_config_file* file has the following line in it:

```
heartbeat_interval 600
```

VMS

The VMS command-line parameter is */localconfig=<config_file>*, where *<config_file>* would have the same line in it:

```
heartbeat_interval nn
```

One drawback in using a large time interval is that when SQL Server does shut down, SQL Monitor Server continues to monitor and to hold the large shared memory segment that SQL Server had created. If the old shared memory segment added to the new shared memory segment is greater than the available system memory, then you need to manually shut down the SQL Monitor Server in order to start SQL Server again.

Windows

Refer to the *v10.1 SQL Monitor User's Guide for Microsoft Windows.*

Windows NT

Follow the example located in the UNIX section.

Unable to attach with shared memory

Error Message Text

```
Unable to attach with shared memory.
```

Explanation

SQL Server creates the *servername.krg* file at boot time. The date of this file should coincide with the date the last time SQL Server was started. Verify this by looking in the SQL Server error log file. If it is not the same date, then SQL Monitor Server was trying to access an old version of this *.krg* file. Do not move the *.krg* file after SQL Server starts up.

This may also mean that you are attempting to start Monitor Server under a different user from the user that started SQL Server.

Action

The value specified for the **-M** flag when starting SQL Server must match the value for the **-m** flag when starting SQL Monitor Server.

You must start Monitor Server using the same account that started SQL Server.

You may need to restart SQL Server in order to have it create shared memory with the proper permissions.

Unknown Network Type

Error Message Text

```
Unknown network type
```

Explanation

If your client is running from a platform, such as Sun Solaris or AT&T (NCR) that uses the Transport Layer Interface (TLI) network interface format, then the entries in your interfaces file must provide the machine name in TLI format.

Action

Use the **sybinit** utility to create entries with the proper format. You need to convert if you have been using Sun Solaris 1.x (SunOS release 4.x) and are switching to Sun Solaris 2.x. Refer to the *System Administration Guide.*

APT and Data Workbench

11 APT Workbench, APT-Edit, and APT-Build

This chapter contains APT Workbench run-time errors and APT-Edit™ compiling and editing errors. It also presents various questions and problems concerning the front-end tool APT Workbench.

APT Workbench Run-Time Errors

There are four levels of severity for APT errors. They are as follows::

Table 11-1: APT Workbench run-time error severity levels

Level Number	Severity	Meaning
0	Informational	Execution continues.
4	Warning	Execution continues.
8	Severe	Current APT-SQL routine exits.
12	Fatal	All execution stops.

For a list of error messages, see the *APT-SQL Reference Manual*.

Custom Profiles and SQL Toolset Customizer

Question

How can I create my own customization files for SYBASE SQL Toolset™ 5.0?

Answer

With release 5.0 of SQL Toolset, you can create your own customization files (custom profiles) using SQL Toolset Customizer (**tsc**), which allows you to control many of the default settings used by Sybase client tools. Use **tsc** to change settings such as the display format of each datatype, the language used, and the edit mode (insert or overstrike).

A *fesa* file is created in the *$SYBASE/custom* directory during installation. A copy of the *fesa* file must be in the same directory where you create your custom profile because **tsc** uses the *fesa* file to initialize the display defaults specified by your custom profile.

You can locate the custom profile in one of two ways:

- Put the custom profile in the directory pointed to by the environment variable (UNIX) or logical name (VMS) SYBCUSTOM.
- If SYBCUSTOM is not set, place the custom profile in *$SYBASE/custom* and set CUSTPROFILE to the name of the custom profile (not the whole path name as is erroneously reported in other manuals, but just the file name).

If you do not have a copy of the *fesa* file in the same directory as your custom profile, you will see this error message:

```
Profile fesa not found.
```

Check the directory indicated by the value of SYBCUSTOM. If the *fesa* file is not present, you can either:

- Copy the *fesa* file into the directory and try again. The *fesa* file is placed in *$SYBASE/custom* during installation. Check with your SQL Toolset System Administrator if you cannot locate it. Do not move the original *fesa* file from its current location.
- Place your custom profile in the same directory as the *fesa* file, usually *$SYBASE/custom*. Then reset CUSTPROFILE to that file name.

◆ ***WARNING!***

If you reinstall TSC and your custom profile is in the directory $*SYBASE/custom*, all the files in that directory may be overwritten.

Additional Information

For more information, see your *SQL Toolset Administration Guide* and *Supplement*.

Changing Fonts in X Windowing Environments for SQL Toolset 5.x

You can change the default font used in SQL Toolset to any fixed-width font provided by your X windowing system such as MIT's X11, OpenWindows, Motif, or DECWindows.

This article explains how SQL Toolset interacts with your X resource files. The last section provides a step-by-step explanation of how to change your default font.

About SQL Toolset, Fonts, and X Windowing Systems

SQL Toolset uses a default font that can be overridden by specifying a different font either in an X resource file, or from the command line. SQL Toolset supplies font aliases that you can use instead of specifying the entire font name, which can be quite long. The aliases are:

Table 11-2: Font name aliases for SQL Toolset

Alias Name	O/S	Alias Definition
"small"	VMS	-Adobe-Courier-Medium-r-Normal--10*
	UNIX	-ADOBE-COURIER-MEDIUM-R-NORMAL--10*
"medium"	VMS	-DEC-Terminal-Medium-R-Normal--14*
	UNIX	-ADOBE-COURIER-MEDIUM-R-NORMAL--14*
"large"	VMS	-Bitstream-Terminal-Medium-R-Normal--18*
	UNIX	-ADOBE-COURIER-MEDIUM-R-NORMAL--18*

The SQL Toolset default font is medium. The asterisk at the end of each line is a wildcard character. It allows the alias to match up with the first font available on your X server that matches the string, since actual font names are a long string of attributes. An example of a font name is:

```
-adobe-courier-medium-r-normal--12-120-75-75-m-70-iso8859-1
```

Each element is a particular attribute of the font. Especially important for the SQL Toolset is the single “m” near the end of the name. This “m”, in uppercase or lowercase, or an uppercase or lowercase “c”, means that the font is fixed-width. **SQL Toolset uses only fixed-width fonts.**

How to Change SQL Toolset Fonts in X Windowing Systems

The following instructions present a straightforward method for overriding the 5.x SQL Toolset default fonts in UNIX or VMS operating system environments:

1. Choose a fixed-width font, using your operating system tools:

   UNIX Environment

   Use either the *xlsfonts* or the *xfontsel* utility. These are provided with X11, and are usually located in the *bin* directory containing other X binaries (*xterm*, *xhost*, etc.). When run from the command line, *xlsfonts* prints out a list of all the fonts available on your system. *xfontsel* shows you the appearance of the font as you select the size and style components. *xfontsel* is a point-and-click style utility.

   VMS Environment

   On most VMS systems there are two files that contain lists of fonts that can be used with DECWindows:

```
SYS$COMMON:[SYSFONT.DECW.100DPI]DECW$FONT_DIRECTORY_100DPI.DAT
SYS$COMMON:[SYSFONT.DECW.75DPI]DECW$FONT_DIRECTORY_75DPI.DAT
```

   These files contain a list of mappings between the font files (in the first column) and the long font names (second column). Choose from among the long font names found in these files. If these two files are not on your system, check with your operating system administrator, or choose from the font listing in the *VMS DECWindows Guide to Xlib Programming*. This guide is also available via the VMS Bookreader Utility, which allows you to view documentation online.

2. Add the font as an X resource to your X resource file, usually the *.Xdefaults* file (UNIX environments) or the *decw$xdefaults.dat* file (VMS environments). These files are usually located in your home directory. You may indicate that you would like to use this font for all SYBASE sessions, or (for UNIX environments only), all APT, APT-Execute or DWB sessions. The syntax is:

SYBASE[1] sessions (UNIX, VMS):	**Sybase.font:** ***font_name***
APT sessions (UNIX):	**apt.font:** ***font_name***
APT-Execute sessions (UNIX):	**aptexec.font:** ***font_name***
DWB sessions (UNIX):	**dwb.font:** ***font_name***
User-named application sessions (UNIX, VMS):	**app_name.font:** ***font_name***

   If you want, substitute the SQL Toolset aliases "small," "medium," or "large" wherever you specify *font_name*.

3. Load these changes into the X resource database to activate them. To do this, do one of the following:

   - Exit and reenter the X windowing environment.
   - Define the XENVIRONMENT logical name (VMS) or environment variable (UNIX) to point to your X resource file.
   - For UNIX only, type this at the operating system prompt:

     ```
     xrdb .X_resource_file_name
     ```

   Start up your SQL Toolset application(s), and the specified font will be used. If you are invoking a user-named application, perform the additional steps below, remembering that in the examples *app_name* is identical to the *app_name* portion of the *app_name*.font entry in the X resources file in step 2 above:

   UNIX Environment

   Add the following flag when you invoke the application:

   ```
   -name app_name
   ```

   VMS Environment

   Define the SYB_NAME logical name before you invoke the application:

   ```
   define SYB_NAME app_name
   ```

[1] Mixed-mode applications, using APT-LIB, also use the font specified by this line.

Overriding SQL Toolset Fonts in X Windowing Systems

To override the settings in your X resource files, give the font name on the command line (UNIX) or define the font logical name (VMS) when you start your SQL Toolset application. Use one of the following procedures, depending on your operating system:

UNIX Environment

Add the following flags when you invoke the tool:

```
-font font_name
```

OR:

```
-fn font_name
```

VMS Environment

Define the SYB_FONT logical name before you invoke the tool:

```
define SYB_FONT font_name
```

Problems with Terminal Types

Problem

Terminal type problems occur when trying to start APT Workbench or Data Workbench in SQL Toolset 5.0.

Error Message Text: Pre-5.0 SQL Toolset

```
Environment variable 'TERM' is not set to 'x11'.

The X display could not be successfully opened.
Please rerun with '-Wl' command line option to
create log.

Couldn't init screenio system . . . please check
terminal type.
```

Sample Error Message Text: Pre-5.0 SQL Toolset

```
Can't initialize the windowing system. Check the
terminal type or the TERM environment variable.
```

➤ ***Note***

Other error messages may appear, depending on which variable is not set properly. See the “Action” section for 5.0 SQL Toolset.

Explanation

APT Workbench and Data Workbench need to know which SYBASE *termdef* file to use. SYBASE tools use a combination of the TERM (SYBASE_TERM starting with 5.0) and SYBASE environment variables (UNIX) or logical names (VMS) to locate the correct terminal definition files in the SYBASE installation directory. There are other environment variables or logical names that may produce this problem in SQL Toolset release 5.0 or later. See the “Action” section below that corresponds to your release level.

Action for Pre-5.0 SQL Toolset

Check the TERM environment variable (UNIX) or logical name (VMS). You may have tried to access a pre-5.0 X11 version of APT without setting TERM to “x11.” To set TERM on UNIX (C shell), type:

```
setenv TERM x11
```

To set TERM on VMS, type:

```
define term "x11"
```

Check to see that the SYBASE environment variable is defined correctly. If SYBASE does not point to the SYBASE Tools directory, APT Workbench and Data Workbench cannot find the *termdef* files. On UNIX, type:

```
setenv SYBASE full_path_to_SYBASE_directory
```

On VMS, type:

```
define sybase_system sybase_directory_location
```

Action for 5.0 SQL Toolset

There are several environment variables or logical names which, if incorrectly set, may cause terminal type error messages. Check which values are currently set with this command:

```
syfechek
```

These environment variables or logical names may cause errors if set incorrectly.

SYBASE_TERM

In the 5.0 Toolset, use SYBASE_TERM instead of TERM. If SYBASE_TERM is defined, the 5.0 Toolset ignores the value in TERM and uses the value in SYBASE_TERM. This is useful when other applications require that TERM be set to “xterm.” To set SYBASE_TERM for UNIX, type:

For example, **`xterm.sun`** or **`vt100`**.

```
setenv SYBASE_TERM term_definition_name
```

For VAX VMS, type:

```
define SYBASE_TERM "term_definition_name"
```

Additional Information

For more information about *fesa*, see “Custom Profiles and SQL Toolset Customizer” on page 11-3, and the *SQL Toolset Administration Guide Supplement*.

Environment Variables for APT Workbench

SYBASE or SYBASE_SYSTEM and the Interfaces File

The SYBASE environment variable (UNIX) or SYBASE_SYSTEM logical name (VMS) must point to the directory where SQL Toolset is installed, or SQL Toolset will not find the *termdef* files, *locales* and *custom* subdirectories.

SYBASE / SYBASE_SYSTEM need not point to the SQL Server software. SQL Toolset and SQL Server can be in the same directory or in different directories, as long as SYBASE / SYBASE_SYSTEM is correctly set:

- When you start SQL Server, set it to the location of SQL Server software.
- When you use SQL Toolset, it should point to the location of the SQL Toolset software.

➤ ***Note***

The location of the *termdef* and *custom* subdirectories can be separately indicated by using the SYBTERMDEF and SYBCUSTOM environment variables/ logical names. If these are not set, then SQL Toolset uses the value of SYBASE.

To set SYBASE for UNIX (C shell), type:

```
setenv SYBASE full_path_to_SYBASE_directory
```

To set SYBASE for VAX VMS, type:

```
define sybase sybase_system rooted_device_location
```

SQL Toolset must have an interfaces file containing an entry for each SQL Server to which it needs to connect. By default, SQL Toolset expects to find an interfaces file in the directory pointed to by the SYBASE environment variable or SYBASE_SYSTEM logical name. If SQL Server and SQL Toolset software are located in separate directories, then a copy of the interfaces file must exist in each directory. (On UNIX systems you can symbolically link a single copy of the file so that it appears in multiple locations.)

On UNIX systems you can specify the location of the interfaces file at run time with the **-I** option.

SYBCUSTOM and CUSTPROFILE

The SYBCUSTOM and CUSTPROFILE environment variables or logical names are used to override the default customization profile, which is in the file *fesa* in the *custom* subdirectory (*custom* is pointed to by SYBASE or SYBASE_SYSTEM). If you do not want to use *fesa*, use SYBCUSTOM to point to the directory of the correct customization file, and CUSTPROFILE should point to the name (not the full path) of the customization file you are using instead of *fesa*.

Do not move the *fesa* file. If SYBCUSTOM is set, a copy of the *fesa* file must be in that directory, even if it is not being used.

SYBTERMDEF

SYBTERMDEF contains the full path name of the directory that contains your terminal definition file. **syfechek** does not reflect changes to SYBTERMDEF, so check this separately.

Open Windows Keyboard Input

Problem

Keyboard input does not work under Open Windows, the X11 window manager.

Explanation

When running SYBASE X11 front ends APT Workbench and Data Workbench under Open Windows, you must tell the OpenLook window manager that the SYBASE X11 front ends are not ICCCM-compliant.

Action

Add the following line to the *.Xdefaults* file for UNIX operating systems or the *decw$xdefaults.dat* file for VMS:

```
% OpenWindows.FocusLenience: True
```

Then either restart the X server to initialize this change, or type one of the following commands:

UNIX:

```
% xrdb .Xdefaults
```

VMS:

```
$ define xenvironment decw$xdefaults.dat
```

APT 5.x, Appearance Names and Color

Question

How can my application take advantage of APT 5.x's ability to manipulate color?

Answer

Be sure you are familiar with the sections in the *SQL Toolset Administration Guide* that have examples you can refer to. These examples are in Chapter 2 and Chapter 4 of the 5.2 *SQL Toolset Administration Guide*.

Before you use the rest of this chapter, you should be aware of the following facts about default appearance names:

- Eleven default appearance names are listed in the *termdef* manager. They cannot be deleted.
- Default appearance names can be used as the appearances of user-defined fields, but they also have an effect on other objects on the form, such as menus and choice fields.
- The appearance editor in the appearance name manager has, by default, all 11 default appearance names, and an additional 6 that may be deleted.

Using Color and Appearance Names

Examine your application and determine where color could be used to enhance it. Color can make an application more attractive and easier to understand, but it can become a distraction if overused. After you determine how to use color in your application, you can begin setting up appearance definitions for your site. Follow the instructions below to change the appearance of your application.

Step One: Check Permissions

You can add or change appearances only if you have the appropriate permissions at the operating system level. Appearance names are stored in the *~sybase/termdef/appdefs* file (or in VMS, the file name is *sybase_system:[sybase.termdef]appdefs)*. If you cannot update this file, you will not be able to save the changes made under SQL Toolset Customizer.

UNIX users can check permissions on the *appdefs* file by using the **ls -l** command. VAX VMS users can check the protections of *appdefs* with the **directory/security** command. These commands display the access at the system level. The operating system account you are using to alter the appearances must have permission to change the file.

These files are owned by the user *"sybase"* at installation.

Step Two: Start SQL Toolset Customizer

After you have set up your environment correctly, including the definition of UNIX environment variables or VAX VMS logical names, start SQL Toolset Customizer by typing "tsc" at the operating system prompt. This is usually located in the $*SYBASE/bin* directory on UNIX operating systems.

If you cannot find **tsc** on your system, contact your System Administrator to ensure that you have 5.0 SQL Toolset and to find out where SQL Toolset Customizer is located.

Step Three: Add the Appearance Name

It is best to create Appearance Names in SQL Toolset Customizer which reflect their use. Programmers refer to the appearances across applications, so make the descriptions generic enough to be used in many applications and descriptive enough to give a clear understanding of their use.

1. Choose **Appearance Names** from the menu.

   You are now in the System Appearance Names form. Some system-supplied appearance names are already available for use.

2. Select **Create!** from the menu.

   The Create Appearance Name window opens. Enter the name you will use to refer to this appearance. Apply the name you have chosen using the menu items ***I*** → **Close** → **Apply**. A message appears, confirming that the appearance name has been added to the list. Now return to the main Toolset Customizer menu.

Step Four: Add Details About the Appearance

After the appearance name has been added, specify the details of what this means for each terminal definition.

1. Select **Termdef Manager** from the main Toolset Customizer menu.

   The list of available terminal definition files is displayed in the Terminal Definitions Manager window. Highlight the terminal definition you want to alter, for example *xterm_c.sun* in UNIX or *xterm_c.dec.* in VAX VMS. Select **Modify** from the menu.

2. Select **Appearance Names**.
3. Select **Create!**.

   For the *xterm_c.sun* or *xterm_c.dec* terminal definition, enter the appearance name you defined previously.

   For X terminals, the *start sequence* is the foreground color of the field and the prompt, that is, what color the letters will be against the field's background.

   The *end sequence* is the background color for the field and the prompt.

   The *comment* field is provided so you can place a description of what the sequence will do on a particular terminal.

   For terminals other than *xterm*, the start and end sequences help define such things as underlining and reverse video.

   Choose ***/*** ⟶ **Close** ⟶ **Apply** from the menu to add the appearance definition for this terminal definition.

Additional Information

Add an appearance definition for each *termdef* file that will be used while running the application. The appearance name used in the application remains the same, but how it affects the application is controlled by how the appearance is defined for each terminal type. If you do not add an appearance definition, the application runs correctly, but the field does not have a special appearance.

If the application is to run on both color and monochrome X terminals, define an appropriate appearance in the *termdef* file for both the monochrome and color terminal.

You can then assign the appearance of FIELDERROR in APT-SQL procedures with the statement:

```
fieldname:appearance="FIELDERROR"
```

You can also assign an appearance to a field in APT-Edit by choosing **Fields** ⟶ **Attributes** from the menu and entering an appearance name in the definition field. To see a list of valid appearance names, position the cursor in the **Appearance Definition** field and press Ctrl-v to obtain a values list.

Mapping APT 5.0 Processing to Control Keys

Question

How can I map APT processing to a control key in APT version 5.0?

Answer

Use both SQL Toolset Customizer and the APT Workbench to map a control key to an APT procedure.

Step One: Associate Key Map Name with Keyboard Action

More than 75 key map names are supplied for your use. Select a key map name and add it to each *termdef* file that will use the functionality. For example, if your application will run on vt100 terminals, Sun color workstations, and Sun monochrome workstations, all three *termdef* files must be modified. The three *termdef* files are *vt100, xterm_c.sun,* and *xterm.sun*. To modify a terminal definition:

1. Select **Termdef Manager** from the main Toolset Customizer screen.
   - You can also map APT procedures to function keys. The procedure is similar to this one, except that you begin by choosing **Function Key Def / Map** from SQL Toolset Customizer screen.
2. Select a *termdef* file and choose **Modify!**
3. For this example, select **Control Key Mapping** from the Modify Terminal Window.
4. Select the control key name that will be tied to the APT processing and select **Modify!** from the menu.
5. Enter an existing key map name from the list of values in the field, or a key map name you created using "Creating a New Key Map Name" on page 11-19.
6. Select **/** → **Close** → **Apply** from the menu of the Control Key Mapping Detail screen.
7. Select **/** → **Close** → **Apply** from the menu of the Control Key Mapping screen.
8. Select **/** → **Close** → **Apply All Changes** from the menu bar of the Modify Terminal Definition screen.The control key is now tied to a key map name. The next step is to link the key map name to processing in APT Workbench.

Step Two: Link Key Map Name to Processing

1. Display the form in APT Edit.
2. Select **Forms** → **Keyboard Processing.**
3. Fill in the Key Map Name Processing window:
   - Enter the key map name to tie to the processing. Remember, this key map name must already be set up in SQL Toolset Customizer as described above.
   - Enter a description for the key name.
   - Choose the type of processing to be used.
   - Enter the name of the processing or form.
   - Enter the version of the form or processing.
4. If you are attaching your own APT-SQL procedure:
   - Select **Edit!** from the menu and create the APT procedure, if it does not yet exist.
   - Select **Compile!** from the menu.
5. Apply the changes by selecting ***/*** → **Close** → **Apply.**
6. Save the form by selecting **Forms** → **Save.**

Creating a New Key Map Name

If for some reason the supplied key map names do not suit your needs, or if you need more names, you can create your own key map names. Add the new name to the list of valid names as follows:

1. Select **Key Map Names** from the main Toolset Customizer screen.
2. Select **Create!** from the menu on the System Key Map Names screen. Enter a unique key map name. The name is case-sensitive.
3. Select ***/*** → **Close** → **Apply** from the menu in the Create Key Map Name screen.
4. Select ***/*** → **Close** → **Apply** from the menu in the System Key Map Names screen.

After you have created a key map name, you can use it in a *termdef* file, as described above.

Using Scientific Notation in APT

Question

How do you control the display of numbers in scientific notation?

Answer: Change Precision for the Current Session (4.0.2 and 4.0.3)

In SQL Toolset releases 4.0.1 and 4.0.2, SYB_MINDECIMAL and SYB_MAXDECIMAL are environment variables (VMS logical names) that control how a *float* field is displayed.

SYB_MINDECIMAL and SYB_MAXDECIMAL define a range where decimal notation is used. If the absolute value of the *float* field falls within this range, the field is displayed in decimal notation. If the absolute value falls outside this range, it is displayed in scientific notation. For example, if you set SYB_MINDECIMAL to the value 1.00e-02, *float* values greater than .01 are displayed in standard notation, and values less than or equal to .01 are displayed in scientific notation.

If SYB_MINDECIMAL is not set, the shorter of standard decimal notation or scientific notation is used.

Define SYB_MINDECIMAL or SYB_MAXDECIMAL as follows, where *float_value* is the absolute value. On UNIX, type:

```
setenv SYB_MINDECIMAL float_value
```

On VAX VMS:

```
define syb_mindecimal float_value
```

The minimum and maximum values for these variables are .99e37 and 1.7e38. You can use either scientific or decimal notation.

All *float* fields are affected by this environment variable. You can set the environment variable / logical name differently for each user's needs. In VAX VMS systems, the system level logical name can be superseded by setting the process level logical name on a user-by-user basis.

Answer: Change Default Precision (5.0 or Later)

Use the new Toolset Customizer (**tsc**)in SQL Toolset 5.0 to define the default precision. You can also set the precision on a field-by-field basis from within APT-Edit. **tsc** is located by default in your *$SYBASE/bin* directory.

1. Type “tsc” to start SQL Toolset Customizer.
2. Select **Customize Profiles** from the main menu of SQL Toolset Customizer.
3. Highlight the profile to be altered and then select **Modify** from the menu of the Custom Profile Manager screen.
4. Choose **Float** to get to the Float Property Sheet and make the desired changes.
5. Select ***/*** → **Close** → **Apply** from the Float Format screen.
6. Select ***/*** → **Close** → **Apply All Changes** from the Modify Customize Profile screen.

In order to take effect, these changes must be made to the *fesa* profile in the custom subdirectory. If another profile is modified, then use SYBCUSTOM and CUSTPROFILE environment variables or logical names to indicate which profile should be used.

Setting *float* Properties in APT_Edit (5.0 and above)

You can also set *float* properties, including scientific vs. standard notation, in the Float Property Sheet from within APT-Edit. This allows you to specify the upper and lower bounds for scientific notation in the *notation* field, for example:

```
10** +4
10** –3
```

Additional Information

See “Change Precision of float Field While in APT-Edit” on page 11-23 for information about changing the precision for scientific notation.

Controlling Displayed *float* Precision

Question

How do you control the number of decimal places displayed for a *float* field when running SQL Toolset?

Answer

You can change the number of decimal places displayed (precision) for either the current session, or the default precision for all applications.

Change Precision for the Current Session (4.0.1 and 4.0.2)

Define the SYB_MAXPRECISION environment variable (UNIX) or logical name (VMS) to be the number of decimal digits in *float* fields, where `integer_value` is the number of decimal digits to be displayed. On UNIX, type:

```
setenv SYB_MAXPRECISION integer_value
```

On VAX VMS, type:

```
define syb_maxprecision integer_value
```

If the specified precision is greater than what can appear on the screen or less than what is being stored in the database, APT will round the value instead of truncating.

All *float* fields are affected by SYB_MAXPRECISION. You can set SYB_MAXPRECISION differently for each user's needs. In VMS systems, the system-level logical name can be superseded by setting the process-level logical name on a user-by-user basis.

Change Default Precision (5.0 or Later)

Use the new Toolset Customizer in the 5.0 SQL Toolset to define the default precision. You can also set the precision on a field-by-field basis from within APT-Edit. **tsc** is usually located in the *$SYBASE/bin* directory.

1. Type "tsc" to start SQL Toolset Customizer.
2. Select **Customize Profiles** from SQL Toolset Customizer.
3. Highlight the profile to be altered and then select **Modify** from the menu of the Custom Profile Manager screen.
4. Select **Float**.

5. On the Float Format screen, enter the number of digits in the *precision* field.
6. Select ***l*** → **Close** → **Apply** from the Float Format screen.
7. Select ***l*** → **Close** → **Apply All Changes** from the Modify Customize Profile screen.

➤ ***Note***

This procedure works only for SQL Toolset. It does not work for **isql**, DB-Library, or **bcp**.

Change Precision of *float* Field While in APT-Edit

You can change the precision of a *float* field while in APT-Edit, if it is already defined as a *float* datatype:

1. Highlight the field, then select **Field** → **Attributes** → **Format.**
2. The Float Property Sheet displays. Set the **precision** option.

If the field is not defined as a *float* datatype you can define it at the **Field** → **Attributes** level, then choose **Format** to display the Float Property Sheet. Changes made in this way will override settings in **tsc.**

For more information, see the *APT-Edit Reference Manual*.

➤ ***Note***

This procedure works only for SQL Toolset. It does not work for **isql**, DB-Library, or **bcp**.

Determining SQL Sent to Server

Question

How do you examine the text of a SQL statement sent to the SQL Server by APT Workbench or Data Workbench?

Answer for 4GL Applications

To see the exact SQL text being sent by 4GL applications to SQL Server, use the environment variable (UNIX) or logical name (VMS) RECTFOS. This will record the SQL text in a specified file which can be reviewed for debugging a new application or diagnosing other problems.

To do this, set the environment variable or logical name RECFTOS to the file name where you want the text to be saved. Set RECFTOS before starting APT Workbench or Data Workbench. In UNIX, type:

```
setenv RECFTOS filename
```

In VAX OpenVMS type:

```
define recftos filename
```

VMS and UNIX operating systems create a new file for each connection made to SQL Server. The files are named sequentially as each connection is opened: *filename*.0;1, *filename*.1, etc.

This feature is present in releases of APT 4.0.1 dated 5/3/90 or later and Data Workbench releases 2.2 or later.

Use **unsetenv** (UNIX) or **deassign** (VAX OpenVMS) to turn off logging when finished, otherwise the information will continue to be logged.

Answer for 3GL Applications

To see the exact SQL text being sent by the 3GL application, use the **dbrecftos** call. See the *Open Client DB-Library Reference Manual* for more information.

Using *submit* vs. *sqlrows*

Question

When do you use **submit** and when do you use *sqlrows*?

Answer

Use the **submit** statement to bring over the results of a query all at once, sending results directly to APT form fields (in pre-5.0 releases, data cannot be assigned to APT-SQL variables or choice fields using submit.). Use **sqlrow** when you want to be able to look at or manipulate one row at a time.

The *submit* Statement

Using **submit** to send results to APT-SQL variables is available with release 5.x of the front-end tools. This can be done using the **submit local** syntax.

The **submit** command automatically transfers the results of a query to form fields when the column headings are the same as the field names. By default, column headings are the same as the column names, but you can override the default by specifying the column heading in the select statement.

For example, to have the data in the *title_id* and *title* fields in the *titles* table bound to the fields *title_id* and *name* respectively, use the following:

```
submit
sqlbegin
    select title_id, name=title from titles
sqlend
```

➤ ***Note***

The actual SQL text must be contained within a **sqlbegin...sqlend** construct.

sqlrows

sqlrows are APT-SQL objects that store individual rows of a query to provide greater flexibility in manipulating data. *sqlrows* are particularly useful in the following situations:

- You want to be able to buffer rows so you can move backward and forward through the results.
- A query returns a large number of rows.
- You want to be able to look at all the rows, one at a time.
- You want to evaluate the data returned by the query before binding it to form fields, or alter the data before transferring it to the form.
- In pre-5.0 releases, you want to assign database values to APT-SQL variables, which you cannot do using **submit**.

Defining vs. Declaring *sqlrows*

Defining a *sqlrow* associates it with a certain SQL query, or batch of queries, and with a certain channel. *sqlrows* cannot be defined using the default channel (*$channel*); they require a separate connection to the database. Note that the channel must be connected before using a *sqlrow*.

Declaring a *sqlrow*, on the other hand, simply establishes the name of the *sqlrow* as an APT-SQL object. Any routine that opens a *sqlrow* must define rather than declare it.

How to Declare or Define a *sqlrow*

Declare or define the *sqlrow* in the APT-SQL routine that uses it. Use this syntax to declare the *sqlrow*:

```
sqlrow myrow
```

Use this syntax to define the *sqlrow*:

```
define myrow sqlrow on mychannel [nomsg[12] ]
sqlbegin
.
.
.
sqlend
```

You can use the same *sqlrow* in more than one APT-SQL routine, but you must define and open it in the same APT-SQL routine. Once the *sqlrow* is defined and opened, you may use it in subsequent APT-SQL routines by declaring the *sqlrow*. Make sure that you always call the routine that defines the *sqlrow* before calling any routine that declares it.

Using *sqlrows*

Once a *sqlrow* is defined, you can open it using the **opensql** command. This executes the query. Use **fetchsql** to retrieve result rows, one by one, into a single-row *sqlrow* array that the APT-SQL routine can read.

To read the columns from a *sqlrow* array, use the **transfer** statement or assign them individually with assignment statements. **transfer** uses the same methods as **submit** to assign values to form fields. In other words, *sqlrow* column values are transferred to form fields that have the same names as the *sqlrow* columns.

In pre-5.0 releases, **transfer** can only transfer values onto form fields, not APT-SQL variables. Transfer to APT-SQL variables must be done by direct assignment. With the release of 5.0, however, direct transfer to variables is possible using the **transfer local** statement.

The syntax for referring to a specific column in a result row is:

```
myrow (column_heading)
```

sqlrow column names are also APT-SQL objects, so it is legal syntax to refer to a column’s attributes. For example, the following is allowed:

```
myvar = myrow(title_id):value
```

This statement assigns the *title_id* column value to the *myvar* variable. To put the next result row into the array for access by the APT-SQL routine, you must again issue a **fetchsql** command. A *sqlrow* is available as long as it is open.

The **value** attribute in the example above is optional. If it is not specified, the compiler automatically assumes that the *sqlrow* column’s value attribute is being referenced. See the *APT SQL Reference Manual* for information about other *sqlrow* column attributes.

To close a *sqlrow*, issue the following command:

```
closesql myrow
```

Other Useful Tips

Test to see if the *sqlrow* is already open before doing an **opensql** by checking:

```
if myrow:open = TRUE
```

To detect errors when the *sqlrow* was opened, that is, when the SQL statement was actually executed, check to see if *myrow:*`errors` is not null. If it is not null, you may want to abort the subsequent **fetchsql** and transfer of rows. Clear *myrow:*`errors` before you do the **opensql**.

To test if there are any more rows, see if *myrow* `:eoq = TRUE. eoq` stands for end of query and is set to TRUE if there is no more data for the current query in the **sqlrow**. A good way of doing this is to do a **fetchsql**, test for eoq, then do a transfer if *myrow* `:eoq` is false. See *Figure 11-1: Using sqlrows example code* on page 11-30.

To rerun a query on a **sqlrow**, you must close the **sqlrow** and reopen it. Use **closesql** to close the **sqlrow**. For example:

```
if myrow:open = TRUE
    closesql myrow
opensql myrow
```

Use another **closesql** at the end of the processing to close the **sqlrow** after all the processing has finished. It is also good practice at this time to close the channel on which the **sqlrow** is defined.

You can use multiple **select** statements in one **sqlrow**. Relevant **sqlrow** attributes are:

Table 11-3: sqlrow properties for multiple SQL statements

Attribute	Description
query	The number of the current query; it starts at 1.
eod	Abbreviation for “end of data.” Its value is TRUE if there are no more queries in the sqlrow.
eoq	Abbreviation for “end of query.” True if there are no more rows for the current query.

See the *APT-SQL Reference Manual* for more details.

If multiple **select** statements have been defined for a **sqlrow**, then the command **nextquery** terminates the current query and moves to the **sqlrow**'s next query. If there are any un-fetched rows in the current query, they are discarded. The syntax is:

```
nextquery sqlrow_name
```

foreach loops can be a very simple way of reading the result rows. For example, to transfer all appropriate rows in a **sqlrow** to a group on the form:

```
opensql myrow
foreach myrow
begin

   if myrow(my_column) < 50
   begin
        transfer myrow
    end
end
closesql myrow
```

Transfer the row only if it meets basic requirement.

foreach only loops through the rows of a single query. It does not loop through multiple queries.

Keep in mind that SQL Server may issue locks when an application selects data from a table. As each row is fetched, the lock will "move" through the table until the last row is fetched. Only when all rows have been fetched will the lock be removed.

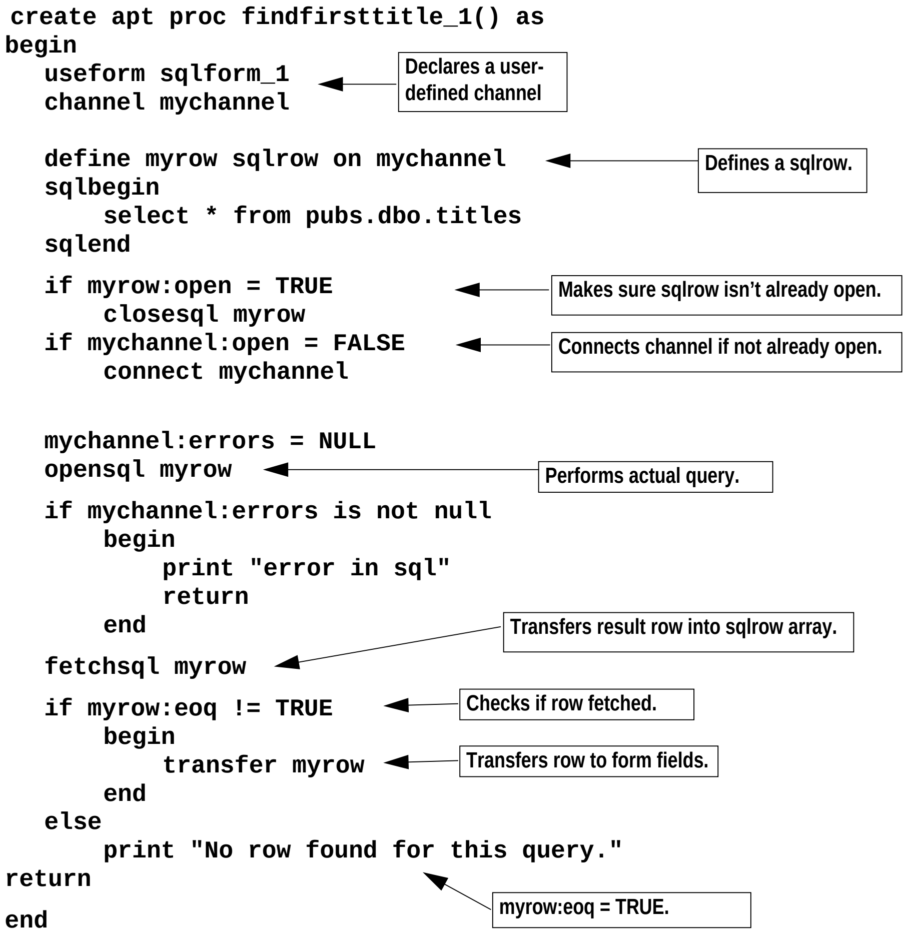
If you do not need to fetch all the rows, you can use the **interruptsql** command to cancel the current query and any remaining queries. This terminates the current query, and releases its lock on the table data, but it leaves the **sqlrow** open so that you may continue to access the data you have fetched. Releasing the lock by using **interruptsql** allows other processes to insert and / or update data in the table.

Two examples of APT-SQL procedures using **sqlrow**s follow. The first defines the **sqlrow**, queries the database, and transfers the first row to a form. The second is an example of a procedure that retrieves the next row from the database using the same **sqlrow**.

Example Code: Using *sqlrows*

This procedure defines the **sqlrow**, opens it (that is, executes the query), and transfers the first row to the form. Subsequent rows in the **sqlrow** are returned by the *findnexttitle* procedure on the next page.

Figure 11-1: Using sqlrows example code



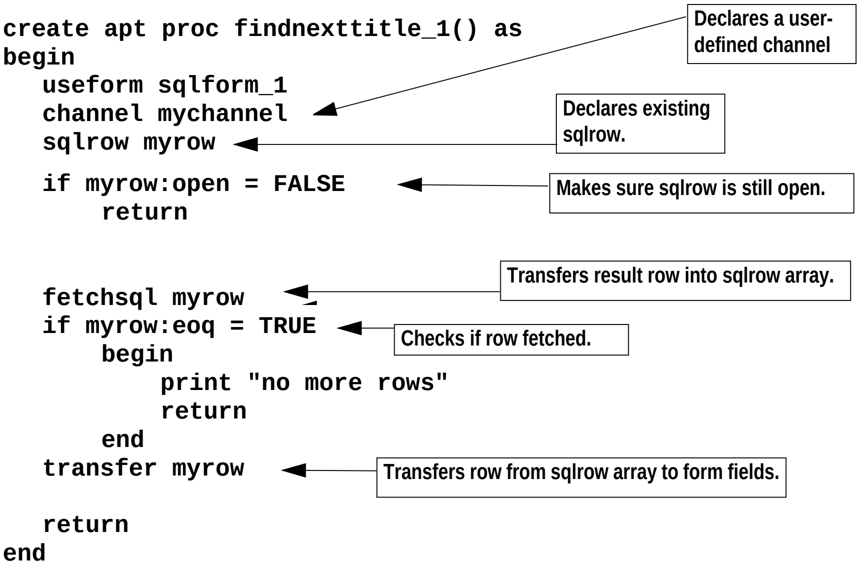
```
create apt proc findfirsttitle_1() as
begin
   useform sqlform_1
   channel mychannel

   define myrow sqlrow on mychannel
   sqlbegin
       select * from pubs.dbo.titles
   sqlend
   if myrow:open = TRUE
       closesql myrow
   if mychannel:open = FALSE
       connect mychannel

   mychannel:errors = NULL
   opensql myrow
   if mychannel:errors is not null
       begin
           print "error in sql"
           return
       end
   fetchsql myrow
   if myrow:eoq != TRUE
       begin
           transfer myrow
       end
   else
       print "No row found for this query."
return
end
```

Declares a user-defined channel
Defines a sqlrow.
Makes sure sqlrow isn’t already open.
Connects channel if not already open.
Performs actual query.
Transfers result row into sqlrow array.
Checks if row fetched.
Transfers row to form fields.
myrow:eoq = TRUE.

Example Code: Transferring Row to a Form

This procedure declares the previously opened **sqlrow** and transfers the subsequent row to the form.

Figure 11-2: Transferring rows to the form



```
create apt proc findnexttitle_1() as
begin
   useform sqlform_1
   channel mychannel
   sqlrow myrow

   if myrow:open = FALSE
       return


   fetchsql myrow
   if myrow:eoq = TRUE
       begin
           print "no more rows"
           return
       end
   transfer myrow

   return
end
```

Declares a user-defined channel

Declares existing sqlrow.

Makes sure sqlrow is still open.

Transfers result row into sqlrow array.

Checks if row fetched.

Transfers row from sqlrow array to form fields.

Buffering Rows in APT-SQL

Question

What does it mean to buffer rows in an APT-SQL routine?

Answer

sqlrows are APT-SQL objects that store individual rows of a query to provide greater flexibility in looking at and manipulating data. They are especially useful with queries that return large numbers of rows, or when each row must be evaluated before you update.

You can put **sqlrow**s into a buffer and access previous **sqlrow**s without having to resubmit the query. The buffer is filled with **sqlrow**s as they are fetched. This is very useful for allowing users to review rows previously retrieved by a query.

A **sqlrow** buffer is created by setting the *bufrows* property of the **sqlrow** after opening the **sqlrow**. The *bufrows* property determines the size of the buffer that will contain the result rows. Suppose, for example, you want users to be able to work with 10 rows at one time. The command for setting this is as follows:

```
myrow:bufrows = 10
```

From the point that this command is executed, **fetchsql** will buffer the rows retrieved from SQL Server until the buffer is full. The next call to **fetchsql** causes the oldest row in the buffer to be discarded to make room for the new row. When the buffer is full, it always contains the last rows that were retrieved from SQL Server.

To retrieve a particular row from within the buffer, use the **nextrow** attribute, which indicates the number of the next **sqlrow** to be retrieved (rows are numbered, starting with 1, as they are retrieved from SQL Server). For example, to retrieve the seventh result row from the buffer, use:

```
myrow:nextrow = 7
fetchsql myrow
```

Any time the **nextrow** attribute is set to a value less than or equal to the value of the count attribute (which tracks the total number of rows that have been retrieved from SQL Server), then **fetchsql** tries to retrieve the row from the buffer.

The previous example assumes that the seventh row is already in the buffer. If it is not, an error is generated. Use the following check to determine whether the desired row is in the buffer or not:

```
if ((myrow:nextrow <= (myrow:count - myrow:bufrows))or
 (myrow:nextrow > myrow:count))
```

If either condition is true, you cannot fetch the row; if both conditions are false, you can fetch the row from the buffer. If *myrow*:**nextrow =** *myrow*:count+1, then **fetchsql** will retrieve the next row from SQL Server.

How to Use Row Buffering

To support row buffering in the example “Example Code: Using sqlrows” on page 11-29, add the following line just before “**fetchsql myrow**” (line 20 in the example):

```
myrow:bufrows = 10
```

With this addition, the last 10 rows are buffered as they are retrieved from SQL Server.

The following example transfers the contents of the next row to the fields on the form. If the current row is not the last row in the buffer, then the next row will be retrieved from SQL Server, otherwise the row will just be retrieved from the buffer.

Figure 11-3: Transferring contents of the next row

```
create apt proc findnext_1() as
begin
        useform sqlform_1
        channel mychannel
        sqlrow testrow

        if mychannel:open = false
             return
        if testrow:open = false
             return
        testrow:nextrow = testrow:rowid+1
        fetchsql testrow
        if (testrow:eoq = true) and
             (testrow:count = testrow:rowid)
            print "No more rows for this query."
        return
end
```

The following sample code is a procedure that uses row buffering to retrieve the row previous to the current row.

Figure 11-4: Retrieving previous row

```
create apt proc findprev_1() as
begin
     useform sqlform_1
     variable first int
     channel mychannel
     sqlrow testrow

     if mychannel:open = FALSE
          return
     if testrow:open = FALSE
          return
     if testrow:count < testrow:bufrows
          first = 1
     else
          first = (testrow:count - testrow:bufrows) + 1
     if testrow:rowid = first
          begin
              print "Already at beginning of buffer, can't get
                  previous row."
          return
          end
     testrow:nextrow = testrow:rowid - 1
     fetchsql testrow
     transfer testrow
     return
end
```

Variable to track first row buffer.
Declare user-defined channel.
Declare previously defined sqlrow.

If not open, sqlrow can't be open.

If not open, no rows to fetch.

Buffer isn't full yet so first row is still in buffer.

Already on first row in buffer.

Set nextrow to previous row.

APT Memory Leaks

Problem

APT hangs or crashes with a fatal XRAISE message, or APT consumes more memory the longer it runs.

Description

These problems are usually due to memory leaks. A memory leak occurs in either of the following situations:

- A routine repeatedly requests memory from the operating system and does not free the memory when it is no longer needed.

  As more and more memory is acquired by the process, but not used, overall system performance usually degrades until the application hangs or crashes, or another process is unable to get the memory it needs to run. This problem may occur after only a few iterations of the routine, or it may take many hours. The speed of performance degradation depends on how much memory is being allocated and how much overall system memory is available.

- A routine tries to write outside of the memory that has been allocated to it.

  This is usually due to a program bug, either in the APT internal routines or in user-created routines. A segmentation error from the operating system usually accompanies the APT error message. The most common message number from APT in this case is “M4-200.”

Action

1. Make sure you have the latest version of the software. Many memory leaks have been fixed since 5.0.

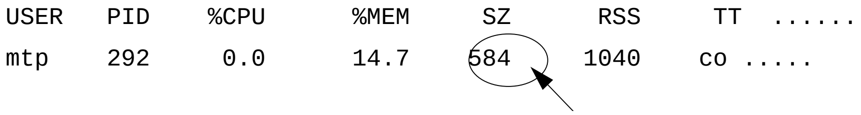
2. Find out if a memory leaks exists by monitoring the memory as the application goes through one of its cycles.

   - On UNIX systems, start the application, and in another window or terminal, use a command similar to one of the following:

   ps -ux | (line; grep aptexec) [For SunOS]

   ps -l | (line; grep aptexec) [For System V]

The command produces output similar to this:



```
USER   PID    %CPU      %MEM     SZ      RSS     TT  ......
mtp    292     0.0      14.7     584     1040    co .....
```

Figure 11-5: Sample “ps” (UNIX) output

The value of SZ gives the size of the process, stack and data only, in kilobytes. As the application runs, repeat the command to monitor the process size.

- On VMS, use the following command before you start the application to be monitored:

`sho proc`

The output will display the process name or ID. Start the application to be monitored, and then in another window or at another terminal, type:

`sho proc/cont/id=`*`process_id`*

The output is a continuously updated screen that displays information about the application process, including a value for “virtual pages” (in VAX 2K pages). Monitor the virtual pages value as you run the application to see if the value is increasing as the application runs.

Any application will grow as new forms (*.frm* files) and APT-SQL routines (*.fpo* files) are loaded into memory. If there is enough cache to hold these all at once the process size should reach a plateau where the amount of memory does not grow any more. Depending on the number of forms and routines used, APT applications may range in size from 500K to 2MB. If the process size continues to grow during manipulation of data and groups in the application when forms and routines are not being loaded, there is probably a memory leak.

3. The errors attribute of channels and **sqlrow**s can consume growing amounts of memory if it is not set to NULL after use. It is a good idea after finishing an action on a channel or **sqlrow**, to do the following:

   1. Check the errors attribute.

   2. Perform the appropriate response.

   3. Set the errors attribute to NULL so that it does not continue to accumulate messages.

-6 Internal Parse Error

Question

What is the -6 Internal Parse Error in APT-Compile?

Explanation

The two most common reasons for this error are:

- A submit structure that exceeds 3K (3072 bytes)
- A symbolic table overflow

The APT-SQL parser creates the submit structure by expanding all the substitution variables and aliases between a pair of **sqlbegin** and **sqlend** statements. If this fully expanded submit structure exceeds 3K, the -6 error is displayed.

APT-SQL allocates space in a symbolic table for all variables and fields used on a form. This table includes system variables and user-defined variables, and also requires space for each field on a form. When you do a **useform**, APT-Compile considers each field on the form to be a variable, in addition to any user-defined or system variables. Using a large number of fields on a form and complicated join structures could cause the -6 internal parse error.

If you use the **-e** flag (or **/list** in VAX VMS) when compiling the procedure, an error file (*.fpe* or *.lis*) is produced and errors encountered during the compilation are placed there. These error files can help determine what internal limit was overrun. For example, suppose the UNIX file contains a *symbolic symbol table overflow* message in it. The message refers to a line number that indicates where the limit was exceeded.

Answer

One solution is to write the SQL code into a stored procedure. By calling the stored procedure you can make the submit structure much smaller and also make the query more efficient.

If this is not practical, you can split the procedure or the submit structure into two parts, thus reducing each submit structure’s length. Create a more modular procedure and limit the complexity of the processing to get around these limitations.

APT Workbench Internal Limits

The following table shows the internal limits of APT Workbench as well as an explanation of each limit. The limits listed below are not adjustable.

Table 11-4: APT-Workbench internal limits

Parameter Name	Limit	Description
FKEYS	32	Maximum number of function keys.
MAXCHANNEL	10	Maximum number of channels that can be opened by the APT application.
MAXCOLLEN	256	Maximum length of a column in a line in the form file.
MAXFORMHEIGHT	99	Maximum scrollable height.
MAXGROUPS	50	Maximum number of groups per form.
MAXLINELEN	2000	Maximum length of a line in the form file.
MAXNEST	10	Maximum nesting level for groups.
MAXPHRASE	256	Maximum length of a phrase in a menu.
MAXTITLELEN	30	Maximum length of a title in a menu.
SYSTEXTLEN	9999	Maximum length for a text field.

APT-Compile Internal Limits

The following table shows the internal limits of APT-Compile as well as an explanation of each limit. The limits listed below are not adjustable.

Table 11-5: APT-Compile internal limits

Parameter Name	Limit	Description
YYMAXDEPTH	150	Limit of the parse stack level. You cannot nest statements deeper than this limit. Depth refers to the number of active constructs on the stack, such as **if**, **foreach**, etc., and individual clauses of statements that have their own definitions. In general, stay clear of the limit on normal nesting. Clause definition is not usually a problem. Note that the nesting is cumulative across all types of nested statements, including **begin, if, while, foreach**, and so on.
MAXBCNEST	40	Cumulative maximum nesting level for APT-SQL statements affected by **break** or **continue** statements, such as **while** and **foreach**.
MAXBSTRING	4096	Limit of size of the string table. The string table includes all literals in the *.fpl*, including the field and variable names, quoted string constants and much of what is in **sqlbegin** / **sqlend** blocks.
MAXCONSTANTS	50	Limit of the number of constants that can be used in any list specification. Examples of lists that might be used are with “in” or “not in” statements.
MAXEXPLEVEL	10	Limit of nesting, implied or explicit, allowed in any APT-SQL arithmetic or logical expression.
MAXEXPRESSIONS (Pre-5.0) (5.0 and above)	 256 512	Limit of the number of expressions that can be used in an *.fpl* file. Refer to the *Transact SQL User’s Guide* for the definition of an expression.
MAXAGGLEVEL	10	Limit of nesting of APT-SQL aggregate functions.
MAXRANGES	50	Limit on the number of items that can be used in any range specification. Examples of ranges that might be used are with “between” and “not between” statements.
MAXFORMAL	20	Maximum number of parameters allowed for any APT-SQL procedure.
MAXSYM	1024	This is the maximum number of symbols allowed in an *.fpl* file. Note that when *useform* is used, all symbols on the form are defined in the *.fpl* whether they are actually referenced later or not.
MAXCODE	4096	This is the maximum size of the *.fpo* object code in words.
MAXSUBMT	3072	This is the maximum length of an APT-SQL submit structure contained between a pair of **sqlbegin** and **sqlend** statements.

APT-Build Internal Limits

The following table shows the internal limits of APT-Build as well as an explanation of each limit. The limits listed below are not adjustable.

Table 11-6: APT-Build internal limits

Parameter Name	Limit	Description
MAXCHANNELS	3	Maximum number of channels per form.
MAXCOMPOSITEFIELD	3 5 in 5.2.2	Maximum number of fields in composite key or join definition.
MAXDETAILTABLES	3	Maximum number of nested groups.
MAXFIELDNAMELEN	128	Maximum length of fully qualified name for a given field.
MAXFORMNAMELEN	30	Maximum length of the name for a form.
MAXFPLOBJLEN	30	Maximum length of the *.fpl* object name.
MAXFPONAMELEN	30	Maximum length of the name for an *.fpo* file.
MAXLOOKUPSPERTABLE	3	Maximum number of look-ups per updatable table.
MAXTABLENAMELEN	256	Maximum length of the name for a table.
MAXTOTALFLDSTRLEN	2048	Maximum length of the string which contains all the column names in a given table.
MAXVERSIONLEN	6	Maximum length of the version for a form.

12 Data Workbench and Report Workbench

This chapter contains a list of some common errors displayed by Data Workbench and Report Workbench and their meanings. It also describes the steps you can take to resolve these problems.

The ideas offered in this section are only suggestions. If you continue to have problems with Data Workbench or Report Workbench after trying the solutions shown here, contact Sybase Technical Support.

Report Workbench Internal Limits

The following table shows the internal limits of Report Workbench as well as an explanation of each limit. The limits listed below are not adjustable in release 4.0 or later.

Table 12-1: Report Workbench internal limits

Parameter Name	Limit	Description
MAX_BAT_COUNT	100	Maximum number of batches for a report.
MAX_RES_CNT	100	Maximum number of results for a batch.
MAX_ALT_CNT	256	Maximum number of **alts** for a result. "alts" are columns in compute rows.
MAX_BRK_CNT	1000	Maximum number of breaks in a report.
MAX_LYT_COUNT	1000	Maximum number of layouts in a report.
MAX_FLDS_LAYOUT	500	Maximum number of fields in a layout.
MAX_VAR_COUNT	10000	Maximum number of variables in a report.

Report Workbench Errors

For a complete list of Report Workbench errors, their text and recommended actions, see the *Report Workbench User's Guide*.

The rest of this chapter contains discussions of some of these Report Workbench errors.

Error 14

Error Message Text

```
Error 14 running report.
```

Explanation

Under OpenVMS, this error occurs if you place quotation marks around the output file name when you use the */file* qualifier on the **runrw** command line.

Action

Run the report without quotation marks around the output file name.

Error 46

Error Message Text

```
Invalid mapped variable for field.
```

Explanation

A select list column or a compute row column has been deleted from a SQL batch statement, but any fields associated with that column are still in their respective layouts. In the layout editor, the field is represented by question marks, like the following:

```
????
```

Action

While working in the layout editor, delete the field using the **Edit** button. Choose **Re-run** from the **Report** menu and your query should produce results.

Alternative Action

If the following is true for your application, you can perform an alternative action:

- You have changed the table definition so that the relevant column has been modified, or
- You have chosen not to include the original column in the select list and are substituting another column with the same data type, and
- You have modified the field associated with the original column, and you would like to use that field for the new column.

If this describes your situation, remap the unmapped field to the variable associated with the new column, and delete the new field created by the batch modification.

Error 69

Error Message Text

```
Error submitting cmd(SQL batch text).
```

Explanation

This error occurs when Report Workbench submits the text of a batch (after making all substitutions) to SQL Server via **dbsqlexec**, and **dbsqlexec** returns anything other than SUCCEED. Usually, this occurs when a database object such as a table or a stored procedure is missing, or there is a permission violation on a database object.

Action

If this error is received while running a report using *runrw*, try running the report using Report Workbench. Report Workbench often reports more details about the cause of the error.

If there are substitutions in the batch, be sure the variables contain the correct, expected values. Test the exact same batch in **isql** or **dwb**, logging in as the same user that is trying to run the report and executing the batch in the same database.

Error 79

Error Message Text

```
Can't allocate line pointer array.
```

Explanation

There is not enough operating system memory available to run the report.

Action

On OpenVMS systems, increase the working set information including the *wsdef*, *wsextent*, and *wsquo* and the page, file, and quota for the swap space by two to four times its current level.

Error 131

Error Message Text

```
Column definition for result differs.
```

Explanation

The stored detail of the result property (the query plan for the report batch that is stored by Report Workbench) does not match the current result. This occurs for one of the following reasons:

- The structure and / or column datatypes of the table from which the data is selected has been changed, or
- The report batch does not produce the same number and datatype of results each time it is run.

This problem is often caused because different **select** statements are in the two parts of an **if-else** clause, or a **select** statement is in one part, but not the other. If a loop such as a **while** loop is being used, then make sure that the same number of loops are executed every time the report is run.

Action

If the error occurs while using *runrw*, run the report using Report Workbench. Note which batch is causing the error as shown on the error screen in the local batch editor, and then exit the batch editor with **Apply**. Answer “no” to **Reformat** if you want to preserve formatting changes. This re-synchronizes the report definition with the data actually returned by the batch.

Then, if needed, remap or delete any fields that have become unmapped as the result of the reapplication of the batch in the batch editor.

If the problem is that the batch does not return the same results regardless of run-time conditions, then you must rewrite the batch so the number and nature of the results returned by the batch are always the same.

Error 154

Error Message Text

```
Error loading report definition.
```

Explanation

User cannot access report, either because they do not have correct permission or because they are using incorrect syntax.

Action

Check to ensure that user has correct permissions to access the report.

If permissions are correct, make sure the user specifies the report owner in addition to the database and report name.

If user is on OpenVMS, make sure they use quotation marks as shown in the example below to avoid a syntax error:

```
runrw "pubs.dbo.myreport" /username="myid"
/password="mypasswd"
/reportparameter=("param1"="type","param2"=5)
/file=report.out
```

The output file name, *report.out*, should not have quotation marks.

Report Workbench or Report Definitions Do Not Exist

Error Message Text

```
There is no report definition in this database.
```

or:

```
Report Workbench does not exist.
```

Explanation

The error appears after selecting the **Reports** menu option in Data Workbench.

Action

Verify that the current database is the one that contains the report tables by selecting ***I*** ⟶ **Tools** ⟶ **Customize DB**. If no report tables are in the current server, ask the System Administrator to run the *installrw* script to install them.

To use your site's database, edit a copied version of *installrw* by replacing **use** *pubs2* with **use** *your_database_name*.

➤ ***Note***

If you run *installrw* in the model database, all databases that you create in the future will already have Report Workbench installed.

Using *installrw*

Usually, if you see the following message, you need to run *installrw*, because the report tables do not exist in the current server:

```
The report definition tables don’t exist in this
database.
```

The *installrw* script creates stored procedures as well as tables. See the *SQL Toolset Administration Guide* for more information.

Before running *installrw*, check to be sure you are using the most recent version of the script. Run *installrw* in each database in which you want to run or save reports.

installrw is in the SYBASE *scripts* directory.

◆ ***WARNING!***

If you run *installrw* in a database that already contains report tables, any existing reports will be lost and you will have to recreate new ones.

If you want to know how much space Report Workbench is taking up, run **dbcc checktable** on each of the tables that were installed with the *installrw* script. For release 4.8 or later, run **sp_spaceused** instead of **dbcc checktable**.

Data Workbench Error

Information on the 702 error follows. For more information about Data Workbench, see the *Data Workbench User’s Guide*

Error 702

Severity Level 20

```
Memory request for %d bytes exceeds the size of
single page of %d bytes.
```

Explanation

SQL Server Error 702 can occur for several reasons unrelated to Data Workbench (see "Error 702" in the *SQL Server Troubleshooting Guide*). However, there is an attribute of Data Workbench's Data Entry facility that prompts Error 702 to occur.

When modifying database tables, Data Workbench sends every updated table column as a search condition to the SQL Server. Therefore any Data Workbench table that has over 128 columns and is updated though the "Modify Data" option generates this error.

Action

If a table has over 128 columns, avoid updating or deleting rows from that table via the "Modify Data" option of the Data Workbench.

Instead, run queries using SQL statements and take into consideration primary, unique keys on that table when you define the rows that are to be updated or deleted.

For example, if a table has a unique key on column *column1*, run the following query in order to delete the row in that table which contains the unique key *unique_key1*:

```
1> begin tran
2> delete table_name
3> where column1 = unique_key1
4> go
```

If the key is unique in the table, only one row will be deleted by the above query. If only one row is being returned by the above query, commit the transaction with the following query:

```
1> commit tran
2> go
```

Otherwise, roll back the transaction by typing:

```
1> rollback tran
2> go
```

If you are not sure if the key *unique_key1* is unique in *column1*, you can check by running the following query:

```
1> select * from table_name
2> where column1 = unique_key1
3> go
```

Similarly, if the table has a unique index on columns *column1* and *column2*, you can delete a row in that table by running the following query:

```
1> delete table_name
2> where column1 = unique_key1
3> and column2 = unique_key2
4> go
```

Additional Information

For more information, see the following:

- *Data Workbench User’s Guide*
- *SQL Server Reference Manual* entry for **delete**
- “Error 702” in the *SQL Server Troubleshooting Guide*

13 APT-Library and Mixed-Mode Programming

This chapter addresses some of the questions that come up when using mixed-mode programming with APT-Library. Mixed-mode programming is the mixture of APT Workbench 4GL forms and 3GL programming, with function calls to APT-Library and Open Client DB-Library.

For more information about mixed-mode applications, refer to *APT Workbench User's Guide*.

For examples of APT mixed-mode programs, see the *gen_main.c* file in your sample directory.

Mixed-Mode Application Parameter Passing At SQL Server Start-Up

Question

How do you pass parameters from a mixed-mode application to an APT form when starting SQL Server?

Answer

The following program fragment demonstrates how to pass parameters when starting SQL Server.

If you are using APT Workbench release 5.0, use **fsinit** to set up the customization flags. See the *APT Workbench User's Guide* for more information.

Program Fragment

Figure 13-1: Sample program for mixed-mode application parameter passing

```
#include <sybfront.h>
#include <sybfrs.h>
#include <sybdb.h>
#include <sys/types.h>
#include <signal.h>

extern LOGINREC  *Login;
extern DBPROCESS *Process;

FORM *CurForm;
static char *parm1, *parm2;
int PassParam();

static FSPROCEDURE procarray[] =
{
        {"PassParam", PassParam, NULL, SYB_C},
        {NULL, NULL, NULL, SYB_C},
};

int ExecForm(formname, version, p1, p2)
        char *formname;
        char *version;
        char *p1, *p2;
{
FORM *fsgetform();
FSPROCEDURE *oldprocarray;
POINTER argarray[3];
int Cleanup();

parm1 = p1;
parm2 = p2;
fsopenscreen(NULL, NULL, NULL);
fsdbproc(Process);
fsdbrec(Login);
CurForm = fsgetform(formname,version);
oldprocarray = fsinstallproc(procarray);
fsprocargs("PassParam", argarray);
fscallform(CurForm);
fsinstallproc(oldprocarray);
fsfreeform(CurForm);
signal(SIGINT, Cleanup);
return(TRUE);

}
```

Start function "ExecForm". Called from main().

Copy the parameters passed to Main Program to Global Statics.

Link APT Lib process to DB Lib process.

Give APT Lib the DB Lib login data.

Attach argument array to APT Lib.

Execute APT Form.

Free the form pointer.

Initialize the APT forms environment.

Invoke a local pointer to the form.

Hook the PassParam() call-back routine to APT Lib and save the old proc array pointer.

On return from the form, replace our callback with old proc array pointer.

Set the UNIX signal SIGINT back to our handler since APT has it.

Return to main().

Figure 13-1: Sample program for mixed-mode application parameter passing (continued)

```
int PassParam(context, argarray)
        FRSCONTEXT *context;
        POINTER argarray[1];
{
fsfwriteval(CurForm, "ASTAB", 1, NULL, -1, SSOFF,
            NULL, parm1, -1, FALSE);
fsfwriteval(CurForm, "OPR", 74, NULL, -1, SSOFF,
            NULL, parm2, -1, FALSE);
return(0);
}
```

Start PassParam function. Called from ExecForm.

Initialize the APT forms environment.

Return to ExecForm.

APT-SQL

Finally, set up APT-SQL preprocessing of the form to include:

`callextern PassParam`

The form must have the hidden fields for the parameters to be passed by this **callextern.**

Channel Error Messages

Problem

Channel error messages occur in mixed-mode programming.

Error Message Text

One of the following:

```
APT_SEVERE (S8,M500), channel for submit is not
open
```

or:

```
APT_SEVERE (S8,M303), unable to obtain login
record for channel #
```

Action

A channel is a connection to SQL Server over which **isql** statements are sent and results are returned.

When using APT-SQL with a 3GL program such as APT-Library, you must call both **fsdbproc** and **fsdbrec**. **fsdbproc** defines the default channel. **fsdbrec** gives the forms run-time system access to a LOGINREC that it can use to open its own DBPROCESS structures.

The "channel for submit is not open" error refers to the default channel in a mixed-mode application and occurs because **fsdbproc** was not called.

The "unable to obtain login record" error can be solved by including a call to **fsdbrec** immediately after the call to **fsdbproc**.

These errors can occur in mixed-mode, even though an application that is entirely 4GL runs without errors.

callextern Fails

Problem

Nothing happens when an APT-SQL form performs a **callextern** to a 3GL program or function.

Action

The 3GL part of a mixed-mode application must “register” any functions that will be called by APT forms while the application is running. Registering is similar to declaring variables and routines in a 3GL program. If the functions are not registered in the 3GL’s **main** section, the application will not know what the functions are or how to find them. See the *APT-Library Reference Manual* for more information and examples.

OpenVMS C Allocation Routines

Problem

In OpenVMS mixed-mode programming (calls made to the APT-Library, LIBSYBAPT.OLB), various errors such as an APT XRAISE or a OpenVMS "access violation" occur when parameters are passed from the form (4GL) to the program (3GL).

Action

Starting with the OpenVMS APT-Library version 4.0.2, the allocation routines **malloc, free, calloc, cfree**, and **realloc** were changed to VAXC$MALLOC_OPT, VAXC$FREE_OPT, VAXC$CALLOC_OPT, VAXC$CFREE_OPT and VAXC$REALLOC_OPT respectively, to improve performance and efficiency on OpenVMS. Program calls must be compatible with those used internally with APT-Library starting at release 4.0.2.

There are two different solutions to this problem:

- Change all mixed-mode programs to replace the standard C allocation routines with the OpenVMS-specific optimized allocation routines.
- Either put the following define statements at the top of each program that is affected, or put them in a ".h" file and change each program to include the ".h" file as needed:

Figure 13-2: Sample define statements for OpenVMS C allocation routines

```
#define   malloc   VAXC$MALLOC_OPT
#define   free     VAXC$FREE_OPT
#define   calloc   VAXC$CALLOC_OPT
#define   cfree    VAXC$CFREE_OPT
#define   realloc  VAXC$REALLOC_OPT
```

Parameters Not Passed

Problem

The value of an APT-SQL variable can be passed to an external procedure. However, when passing a form field as a parameter from an APT procedure to an external procedure, the value is not passed.

Explanation

If the parameter being passed is an APT-SQL variable, only its value is passed. It can be used in **printf** and other commands like a normal C variable.

If, on the other hand, the parameter is a form field, the OBJDSC pointer (the address of the object descriptor) is passed instead of the value itself. This way the application can use the pointer as a parameter to other APT Library routines that require an OBJDSC as a parameter to access the form variable.

Action

Use **fsfgetval** to get the value of the form field from the OBJDSC**.**

APT Error and Message Handlers

Problem

The default APT handler is invoked, even though the DB-Library error and message handlers are installed and in place.

Explanation

A call to **fsopenscreen** installs APT's error and message handlers. If the calls to **dberrhandle** and **dbmsghandle** are made before the call to **fsopenscreen**, the APT handlers will be in use.

Action

Make the call to **dberrhandle** and **dbmsghandle** after the call to **fsopenscreen.**

Mixed-Mode Error Handling

Problem

Various problems can occur when you install your own message and error handlers in a mixed-mode application that calls the forms library routines.

Action

These are the three types of errors and messages that can occur:

- SQL Server
- APT (run-time)
- DB-Library

Each type is explained in detail in the following sections.

SQL Server

The message handler installed via **dbmsghandle** handles Server errors and messages. APT installs its own message handler by default. Your message handler takes the place of the APT message handler. This creates a problem, because the APT message handler updates *channel:errors*, an APT-SQL structure that handles errors within APT-SQL. The best way to avoid this problem is to invoke the default APT message handler from your own. To do this, use the following code fragment:

Figure 13-3: Sample code for mixed-mode error handling

```
/*  In main program, declare a variable that will
   save address of default APT message handler */

   int     (*apt_msg_handler)();
   int     user_msg_handler();
   .
   .
   .
   apt_msg_handler = dbmsghandle( user_msg_handler );
   .
   .
   .
   /* now, in message handler, invoke aptmsghandler */
```

Figure 13-3: Sample code for mixed-mode error handling (continued)

```
/*----------------------------------------------*/
int user_msg_handler(dbproc, msgno, msgstate,
        severity, msgtext, srvname, procname, line)
DBPROCESS       *dbproc;
DBINT           msgno;
int             msgstate;
int             severity;
char            *msgtext;
char            *srvname;
char            *procname;
DBUSMALLINT     line;
{
.
.
.
[*apt_msg_handler]( dbproc, msgno, msgstate,
   severity, msgtext, srvname, procname, line);
.
.
.
}
```

Substitute the name of your own APT message handler for *[*apt_msg_handler]* in the above code.

APT Run-Time Errors

Internal APT-Library routines handle APT run-time system errors. There is no mechanism for installing a user-defined handler for APT run-time messages and errors, so their handling cannot be modified. APT error messages are usually displayed on screen. Execution is terminated depending on the severity of the error. See “APT Workbench Run-Time Errors” on page 11-2 for details.

APT messages are listed in the *APT-SQL Reference Manual*.

APT errors generated by the APT run-time system are handled by APT-Library routines that are not affected by **dbmsghandle** or **dberrhandle**. You cannot, for example, submit requests to SQL Server on a channel that is not open.

Open Client DB-Library

The error handler installed by **dberrhandle** typically only processes DB-Library errors. To ensure correct handling of severe DB-Library errors, use the method explained above for the message handler.

That is, save the address of the default APT error handler, and invoke it when your message handler is installed.

Action for Redundant Messages

After you invoke the APT message handler, you may see redundant errors or warnings. This happens when the APT message handler and then your own error handler process the same errors. These redundant error messages should not interfere with normal operations and you can ignore them, or avoid them using any of the following strategies:

- Do not print anything in your own handler.
- Anticipate which errors will cause the default handler to display errors, and don't display the errors in your own handler.
- Avoid redundant error messages by distinguishing between DB-Library queries to SQL Server and APT forms queries in the same program. Use the calls **dbsetuserdata** and **dbgetuserdata** in DB-Library release 4.0 or later. **dbsetuserdata** connects data to **dbproc**s. **dbgetuserdata** can be called in the message handler (**dbproc** is one of the arguments to the message handler) to determine whether this **dbproc** is a DB-Library **dbproc** or an APT-created **dbproc**.

A Finding Your Library Version

You may need some or all of the following information when calling Sybase Technical Support.

Programming Libraries

- To find your library version for Open Server, use **SRV_GETVERSION**.
- To find your library version for Open Client DB-Library, use **dbversion**.
- To find your library version for Open Client Client-Library, use **ct_config(...CS_VERSION...)**.

Operating System Libraries

To find a UNIX operating system library version, use:

```
strings library | grep version
```

For example:

```
strings $SYBASE/libsybdb.a | grep 4.6.1
```

To find a VMS operating system library version, use:

```
search library version
```

For example:

```
SEARCH SYBASE_SYSTEM:[SYBASE.LIB]LIBSYBDB.OLB -
 "4.6.1"
```

DB-Library Versions

There is a file containing the copyright for the DB-Library product that includes the software version.

On UNIX systems, this file is *$SYBASE/msgs/copyright_dblib*.

On VMS systems, there will be a file for each language:

Language	File Name
C	*SYBASE_SYSTEM:[SYBASE.MSGS]COPYRIGHT_DBLIB.*
Ada	*SYBASE_SYSTEM:[SYBASE.MSGS]COPYRIGHT_ADBLIB.*
COBOL	*SYBASE_SYSTEM:[SYBASE.MSGS]COPYRIGHT_CDBLIB.*
FORTRAN	*SYBASE_SYSTEM:[SYBASE.MSGS]COPYRIGHT_FDBLIB.*

Language	File Name
Pascal	*SYBASE_SYSTEM:[SYBASE.MSGS]COPYRIGHT_PDBLIB.*

Embedded SQL Versions

When you call Sybase Technical Support, please provide the analyst with product, compiler, and precompiler versions.

To find the precompiler version under release 10.x, type **cpre -v** or **cobpre -v** (UNIX and PC platforms) or **cpre /version** or **cobpre /version** (VMS). This prints the version string.

To find the version of Embedded SQL run-time libraries in 10.x environments, use the following commands (UNIX only):

```
% cd $SYBASE/lib
% strings library_name | grep Sybase | grep /
```

where *library_name* is the name of the library you are interested in (or replace *library_name* with '*' for information about all libraries). This will print out a line showing the version of the run-time library.

In pre-10.0 environments, use the following command:

```
% strings library_name | grep Sybase
```

or

```
% strings library_name | grep Copyright
```

➤ ***Note***

libcobct.a has no version information before the 10.0.2 release level.

Index

Index