

# Centura Pro

Visit us at [www.ProPublishing.com](http://www.ProPublishing.com)!

Hot Ideas for Centura® Developers

## Drill Deeper with LogView

**Richard Lindberg with Michael Cunningham**

SQLWindows database routers help developers with their ability to log all of the SQL activity in an application; but it still takes some time to figure out what's going on when you read the log text. It's also inconvenient to recreate a sequence of events using the log. Our utility, LogView, does nothing that couldn't also be done manually with a text editor, a query tool, and some spare time. It just does it better and faster.

### The LogView utility

LogView reads the log produced by SQLWindows and displays the SQL statements with the parameter values included. Queries can be executed, with their result sets displayed. **Figure 1** shows the main window of LogView.

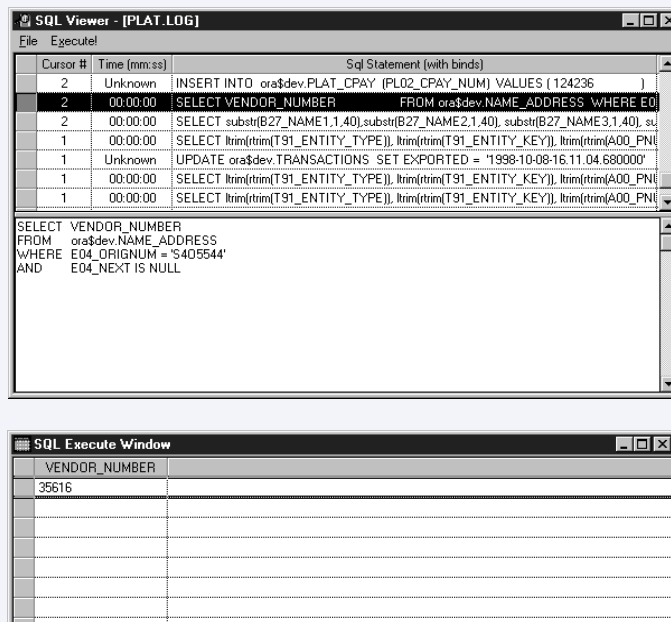
Why would you want to use logging and LogView? We can think of several reasons:

*Continues on page 3*

SQLWindows applications can get complex, and it's not always clear what's occurring in the background, especially with regard to SQL statements. This utility helps SQLWindows developers make better sense of their database log.

SQLWindows

16



**Figure 1.** The main LogView window. The top screen appears before you run LogView. The lower screen shows the results.

## January 1999

Volume 4, Number 1

- 1 Drill Deeper with LogView  
*Richard Lindberg with Michael Cunningham*
- 2 Another Turn of the Wheel  
*Mark Hunter*
- 4 Ace! Centura App Detective: The Case of the Missing Position
- 5 Slicker File Operations  
*Joachim Meyer*
- 5 Centura Tip: A Table Window Bug Fix  
*Michel de Becdelièvre*
- 8 Centura Tip: Comments in SQLTalk Scripts  
*R.J. David Burke*
- 9 Roll Your Own Select Case  
*R.J. David Burke*
- 10 Centura Tip: Converting to Hexadecimal  
*R.J. David Burke*
- 11 More ListView Control Tips  
*R.J. David Burke*
- 11 Centura Tip: Deleting Array Elements the Fast Way  
*R.J. David Burke*

**PRO**  
Publishing

# Drill Deep with LogView . . .

Continued from page 1

- Typically, your application puts up an hourglass icon and stares at you for a long time and finally you cancel the task. Looking at the log can tell you the last SQL statement executed. If there's a loop, you have some starting point for investigating.
- Your application runs to the end of the job and all looks well. Did it really do all the updating it was supposed to? Going through the log will show this.
- A dropdown box contains some unexpected values. How was it populated? LogView shows this quickly.
- A problem shows up in the production environment, but you can't recreate it in the test environment just from screen prints. Reading through the SQL log can help you to set up the conditions causing the error to occur for you.

## Setting up SQLWindows logging

If you've never done it before, you should know that logging requires some setup. On my system, I defined a folder and decided on the log name:

```
c:\LOGVIEW\SQL.log
```

I like to separate the logging files from the GUPTAxxx directory so that it won't be affected if that directory is deleted and SQLWindows is reinstalled.

To tell SQLWindows where to log, add a section to the SQL.INI file.

```
[winclient.oraw]
log=c:\LOGVIEW\SQL.log
```

The first part of the section name is fixed, "winclient." But the "oraw" will likely be different for your system. To find the right suffix for your use, locate the "[winclient.dll]" section of the SQL.INI file:

```
[winclient.dll]
comdll=sqloraw
comdll=sqlsgsw
```

I wanted to log only the Oracle sessions. I took the characters after "sql" and that becomes the suffix. You can define different logs for each "comdll" line.

## Network logging

If the SQL.INI file is on the network and is shared by everyone, then you'll have an extra setup. Not everyone needs to log, but those who do need their own log. To handle this, define a small file that has your own "winclient.oraw" information. I call mine SQLTmpl.INI

and put it in the c:\LogView directory with SQLWindows. SQLTmpl.INI contains:

```
[winclient.oraw]
log=c:\LOGVIEW\SQL.log
```

To add this section to a local copy of the SQL.INI file, I wrote a batch file, called, "make\_ini.bat". Adapt the contents to your system.

```
type c:\logview\sqltmpl.ini > c:\apps\gupta503\sql.ini
type j:\swin503\sql.ini >> c:\apps\gupta503\sql.ini
```

For Windows 95, place a shortcut to this batch file in c:\windows\Start Menu\Programs\StartUp. Set the properties (Program tab) to "Close on exit." This will copy the network SQL.INI every time you start and will reflect changes that may have been made to it. If changes are made during the day (as happens during development), you can rerun make\_ini.bat to recreate your local SQL.INI file.

Add the following line to your AUTOEXEC.BAT, and you're ready to log the next time you boot up.

```
SET SQLBASE=C:\APPS\GUPTA503
```

## Installing LogView

LogView comes in the form of a self-extracting .ZIP file. Load it into the c:\LogView directory and execute it by double-clicking. It contains:

```
lv.exe
ParseLog.dll
LogView.hlp
LogView.ini (This must be placed in
              your Windows directory)
LogView.ico
```

Set up a shortcut to C:\LogView\LV.exe, and the installation is complete.

## Sample source code in LV.APP

LV.APP is a SQLWindows application. The general design is very straightforward: A log file is selected to open. The name of that file along with the name of a work file (.lvf) are passed as parameters to ParseLog(.). The work file is opened and displayed in a table window. The SQL statement with binds on the selected row is displayed in a box. Here it can be copied to the Clipboard or edited. The currently displayed statement is what will be executed by the "Execute" menu item.

## ParseLog.DLL

ParseLog is written in C. Since SQLWindows is a 16-bit environment, it was set up as a DLL project in Visual C++ 1.0. Debugging and testing was done as a DOS .EXE stand-alone program, replacing the SQLWindows-specific code with a program wrapper to open the files and get the file pointers to use as parameters.

Parameters are the SQLWindows log file name and

the name of the work file where the parsed log will be written. Return value is zero for successfully processing the files. The function is defined to SQLWindows as:

```

External Functions
Library name: ParseLog.dll
Function: ParseLog
Export Ordinal: 2
Returns
Number: LONG
Parameters
String: HSTRING
String: HSTRING

```

As the log is parsed, a structure of linked lists is built:

```

CURSOR-->CURSOR-->CURSOR-->CURSOR
|
| | |
| | | ERROR-->ERROR-->ERROR
| | |
| | | SQL FRAGMENT-->SQL FRAGMENT-->SQL FRAGMENT
| | | |
| | | | FRAGMENT TEXT
| | |
| | | PARAMETER-->PARAMETER-->PARAMETER
| | | |
| | | | PARAMETER TEXT
|
MERGED SQL AND PARAMETERS

```

CURSORS get built as they're encountered in the log.  
 SQL fragments are added starting after "[compile]" and ending before "After compiling..."  
 PARAMETERS follow "[execute]" until the first line that's not a parameter (see the "IsParameter()" function).  
 MERGED SQL AND PARAMETERS is built at the

end of the parameter list or just before output if there were no parameters.

OUTPUT is written at "[end of fetch]" for this cursor or the next "[compile]" or "[execute]" of ANY cursor.

All the SQLWindows-specific code has been separated from the ParseFile() function. This was done as good practice and to make testing of the ParseFile() function easier, since it could be tested completely separately from the rest of the LogView application.

Here's the source for ParseLog.DLL:

```

PARSELOG.H

#include "stdlib.h"
#include "string.h"
#include "stdio.h"

int __export FAR PASCAL ParseLog(HSTRING
    hInputFilePath, HSTRING hOutputFilePath);

void ParseFile(FILE *in, FILE *out);

PARSELOG.C

#define STRICT
#include <windows.h>
#include "swtype.h"
#include "swin.h"
#include "ParseLog.h"

int CALLBACK LibMain (HANDLE hInstance,
    WORD wDataSeg, WORD wHeapSize,
    LPSTR lpszCmdLine)
{
    if (wHeapSize > 0)
        UnlockData (0); //Unlocks the data
                        segment of the library.

    return 1;
}

```

```

int __export FAR PASCAL ParseLog(HSTRING
    hInputFilePath, HSTRING hOutputFilePath)
{
    FILE * fpInputFile;
    FILE * fpOutputFile;
    char huge * lpInputFilePath;
    char huge * lpOutputFilePath;
    LONG    lLenInputFilePath;
    LONG    lLenOutputFilePath;
    lpInputFilePath=SWinHStringLock(hInputFilePath,
        &lLenInputFilePath);
    lpOutputFilePath=SWinHStringLock(hOutputFilePath,
        &lLenOutputFilePath);

    if (fpInputFile = fopen( lpInputFilePath,"rb"))
        ==NULL)
        return (1);

    if ((fpOutputFile = fopen( lpOutputFilePath,"wb"))
        ==NULL)
        return (2);

    ParseFile(fpInputFile,fpOutputFile);

    if (fclose(fpInputFile)!=0)
        return (3);

    if (fclose(fpOutputFile)!=0)
        return (4);

    SWinHStringUnlock(hInputFilePath);
    SWinHStringUnlock(hOutputFilePath);
    return (0);
}

```

## Known problems

- We suggest renaming the log file before running LogView. LV.EXE is a SQLWindows application and, therefore, writes to the log when it runs. In addition, as you run tests, it may be easier to have each test logged in a separate file. The easy way to clear out an existing log file is to "Select all" and "delete" within a text editor.
- The log file doesn't contain all occurrences of a FETCH on every cursor. LOGVIEW can display only what has been logged.

- SQLWindows has a limit of 32,000 rows in a table window. If there are more than 32,000 lines selected to be displayed, only the first 32,000 will be put in the table window. The log can be erased and restarted by closing SQLWindows and all applications written in SQLWindows. This means 32,000 SQL statements, not 32,000 lines in the log.

## Future enhancements

We plan to release a 32-bit version of LogView in the future, as well as enhancing the 16-bit version. New versions will have the ability to show only specific SQL statements (in other words, only UPDATES or only certain table names), and the ability to search for a particular character string. **CP**

*Download LogView.ZIP from this issue's Table of Contents at [www.propublishing.com](http://www.propublishing.com) or find it on this month's Companion Disk.*

*Source code for the ParseFile function is available on request. E-mail the author at [RichardOL@aol.com](mailto:RichardOL@aol.com).*

Richard Lindberg soldered together his first PC in 1977. Now he works as a Senior Programmer/Analyst with a special interest in the creation of "package interfaces"—getting data into and out of software modules and other topics electronics related. When he's not working in SQLWindows, he codes in C. Reach him at [RichardOL@aol.com](mailto:RichardOL@aol.com).

Michael Cunningham is an independent consultant doing SQLWindows and Centura work in Napa, California. Reach him at [mcunning@napanet.com](mailto:mcunning@napanet.com).