ODBC Logging

Lars Karlsson
c011kn@cs.umu.se
6th June 2005

Abstract
This article discusses issues with logging in ODBC libraries. The standard way of doing this is to write the contents of input and output parameters for each call. This type of logging becomes incomplete when dealing with deferred inputs and outputs¹, which in the case of ODBC creates a big gap in the logging capabilities since deferred parameters are used for important buffers. Furthermore, the logging does not match well with the higher level needs of the reader of the log. One logical task may often span several function calls and the log for the task is hence split into several pieces. This article has examined a way to remedy this.

1 Introduction
When using ODBC² to connect to databases it is in some cases beneficial to see a trace of what calls were made, which arguments they were passed and what they returned. For this purpose some implementations of the ODBC Driver Manager provides a means of getting this information. In some situations, especially when developing and analyzing applications, there is a need to have a logging that better matches the semantics of the calls. The individual calls to SQLBindParameter, SQLPrepare and SQLExecute together form the high level operation of executing a statement with dynamic parameters. To create a logging that logs at this higher level is the goal of this project.

Interesting higher level logging functionality include:

- Combination of bound parameters with their statement text
- Printing of data fetched from the database
- Timing function calls
- Reporting errors for failed calls
- Discovering bad programming practice, like forgetting to free handles or closing cursors
- Pretty printing complex SQL statements

This project has focused on these features, except discovering bad programming practice.

¹A deferred input/output is a buffer who's content is not used by the function it was passed to but rather by a later function call, thus rendering the content of the buffer invalid when it is still known and therefore impossible to log in a straightforward manner.
²Open Database Connectivity
## 2 Approach

This is a suitable place to give a review of the ODBC architecture. The application programmer links his executable with a Driver Manager library. The purpose of the Driver Manager is to load and unload drivers as needed and therefore completely hides the details of which driver to use from the programmer. This information is instead provided by the user running the application in a file typically called `odbc.ini`. The Driver Manager has absolutely no control over any database and for most of the time simply routes the calls to the appropriate driver. This scenario is outlined in figure 1. [1]

![ODBC Architecture Diagram](image)

Figure 1: ODBC architecture, with the Driver Manager and Drivers all adhering to the ODBC API specification. The communication between drivers and databases is completely opaque to the ODBC application.

The new logging mechanism should be readable in plain text by a human, and store information to file as soon as it’s available to avoid information loss due to a crash. Each function call shall get one log record and it should contain the low level information such as parameter names and values as well as higher level information if it is available. Furthermore, the logging must be configurable to limit the information overload otherwise experienced.

There are multiple options available when implementing this new logging functionality. The approach taken was to create a DSO\(^3\) which interposes every ODBC function. It is important not to affect the logged system, which is why the DSO must be carefully crafted to reduce the risk of such things as unintended symbol interposing [2]. The dynamic object can then be used in two modes:

1. Loaded before other objects by setting the `LD_PRELOAD` environment variable used by the dynamic linker in UNIX-like environments [3].

2. Explicitly linked against by the executable.

---

\(^3\)Dynamic Shared Object is a piece of code that might be loaded and used dynamically by other executables
3 RESULTS

The first mode is temporary in nature, while the second option is better suited for long term usage.

Configuring the library is done by either using the graphical tool developed as part of this project or by modifying the configuration file by hand. The file is brought in when the dynamic object is loaded, which triggers the call to the dynamic object constructor [2]. The path to the file is by default "./odbcl.log.cfg" but can be changed by an environment variable.

To get information needed to log parameters and fetched tuples as well as errors, calls are issued to the ODBC library. This is the only way to get error messages and one of the two ways to get buffer pointers and other required data. This goes beyond passive logging and although this may be risky, care has been taken to minimize the chance of state changes that may affect the logged application.

Complex SQL statements must be presented in a format that is readable, which is why a pretty printer is necessary. For this purpose, common parts of the SQL grammar were implemented using yacc and flex. The parser was used to break down and rebuild the statement with indentation information. If the parser succeeded, the resulting statement is then output according to the indentation information contained within it. Each line is viewed as a set of indentation fields of equal length. The indentation directives put into the statement during parsing are:

$+ prints subsequent characters with the next indentation field border as the left margin.
$- prints subsequent characters with the previous indentation field border as the left margin.
$= prints subsequent characters with the left margin set to the current line offset.
$n inserts a new line

These four directives gives enough power to produce highly readable logs of complex SQL queries (see for example figure 3), even though they do not take context into consideration at all.

3 Results

The system creates plain text files as described before. These have a format suitable for both humans and machines. Therefore, it's easy to create a tool to convert the plain text into something easier to browse. The project involves such a tool called log2xhtml which produces XHTML powered by CSS and JavaScript. The result samples presented here are taken from the XHTML form of the log files.

Figure 2 shows parameter substitution and figure 3 shows the power of the pretty printer.

Configuration makes it possible to include or exclude specific function calls in the log, with the potential of substantially reducing the number of log records. Timing of function calls provides performance monitoring.
5 CONCLUSIONS

33. 11:13.24 (10.72ms) - SQLExec

Figure 2: Result of a parameter substitution, the four and ten were substituted instead of parameter markers

39. 10:18.27 (3.49ms) - SQLExecDirect

Figure 3: Pretty printing of complex SQL queries by filling in indentation information during parsing

4 Discussion

The system was implemented in C for UNIX-like systems. It relies on a patched version of the compiler suit GCC\(^4\). The project has been struck by some setbacks due to library versions. For instance, the psqlodbc driver for ODBC access to PostgreSQL has in recent years gone through substantial improvements, implementing otherwise unsupported ODBC functions. The current implementation of the system relies on some of these new functions to get buffer pointers and other such information.

Logging is all about making internal operations of a system visible to humans. The process of deriving a good logging mechanism is primarily guided by the wish to get the most useful representation of the operations. This project has highlighted that at some point, the logging may get too intrusive and change the behavior of the application. This is highly unwanted as the quality of the logging result reduces sharply if it’s not trustworthy. Even for this little project, there is in some cases no other option than to be intrusive.

5 Conclusions

Logging function calls by printing values for input and output parameters is clearly useful, which is why it is a part of ODBC Driver Managers. Higher level

\(^4\)The patch is for changing the default visibility of symbols, merged into GCC starting with version 4.0
logging, as shown by this project, adds to the usefulness of the logging mecha-
nism. In some cases, as with deferred inputs and outputs, it adds strictly more
functionality. This kind of logging demands more from the library programmer
in terms of effort and lines of code. The Driver Manager implementation iODBC
[4] has a goal of providing dynamic loading of custom trace libraries. This would
leave open the possibility to have a high level logging without placing the bur-
den of writing it on the library authors. This project has further shown that
certain desirable features (like error reporting) is impossible to implement by
looking only at function inputs and outputs.

References

       us/odbc/html/odbcabout_this_manual.asp>


[3] Roland McGrath, Ulrich Drepper, and others, “ld.so man page”, 2000-10-