4D Debug Log Analyzer

By Josh Fletcher, Technical Services Team Member, 4D Inc.

Technical Note 09-39

Table of Contents

Abstract

This Technical Note and included database give developers a tool for efficient analysis of the 4D Debug Log file. The approach used in this document places emphasis on using the 4D Debug Log file as a focal point for statistical analysis regarding Project, Form and Object methods. A sample database and log file are included.

Introduction

This database is provided to allow efficient analysis of the 4D Debug Log (4DDL) file. For more information on the 4DDL feature, see the **SET DATABASE PARAMETER** command, as well as Technical Note 06-01 (Knowledgebase asset #41182), *The 4DDebugLog.txt File*.

The 4DDL can be useful for troubleshooting crashes and lock-ups because the most recent actions that occur at the time of the issue are logged. However this typical scenario is not what this database was developed for.

The 4DDL is also useful because it logs all Project Method, Form Method, Object Method, plug-in method, and 4D Command calls in the database. Furthermore, for Project, Form, and Object methods, the beginning and end of each method is logged so it is possible to measure the execution time of those commands. This database was created to digest all of this logging information and generate statistics like:

- Total number of calls to each method.
- Average execution time for each method.
- Longest execution time for each method.

Much of this could be accomplished with a spreadsheet but the volume of data that the 4DDL might contain can quickly grow beyond the reasonable bounds of a spreadsheet application like Excel (which only supports ~67,000 lines per sheet in the 2003 version). The example 4DDL included with the database represents roughly 10 minutes of database execution, resulting in 1.6 million log lines. Thus getting this data into a database is far more practical for reporting purposes.

Usage

Launch the database with 4D. By default the database launches in Design Mode. This is not an end-user tool, so if you want to explore the data you'll do it from Design mode using the List of Tables.

Note: the Debug Log File Analyzer database is designed to parse debug log files that were generated by 4D in interpreted mode only. It will not correctly handle debug log files created in compiled mode.

There are two fundamental tasks that can be performed:

- Import 4DDL
- Generate a summary for an imported 4DDL

Import 4DDL

This task is used to import 4DDL files and parse the data they contain. Each 4DDL session is saved in the [Debug_Logs] table, which has a one-to-many relation with the [Log_Lines] table.

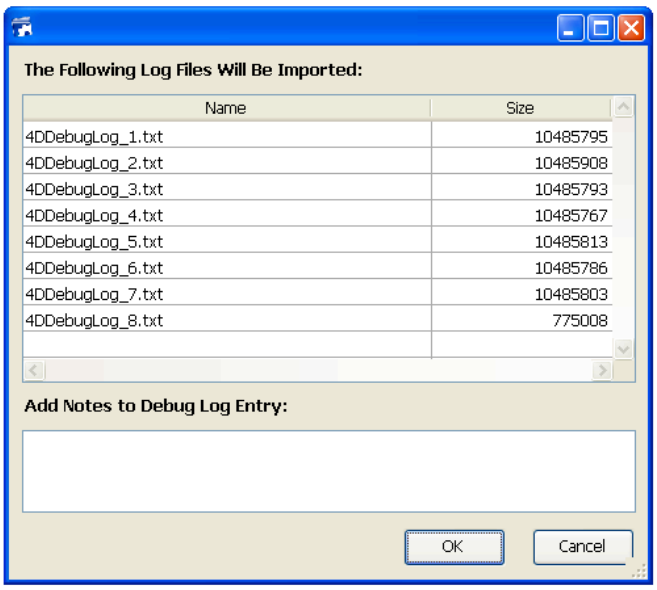
Run the method **DBL_M_Import** or select "Import Debug Log..." from the File menu in Application mode.

You will be asked to choose the folder that contains your debug log files.

Note: *In 4D v11 SQL the log files are automatically segmented into 10 MB chunks. The database will automatically import all segments, in order.*

The database is designed to have a single debugging session inside the selected folder. I don't know what will happen if you choose a folder that contains more than one session, since it wasn't part of the design. There is a test, however, to see if the millisecond timestamp of each log segment matches the first. The database will complain if it does not match and you can cancel the import at that time.

Once the folder is selected, a list of all discovered log segments will be displayed:



This is for confirmation purposes only, you cannot customize the list.

There is a space provided to add notes about this 4DDL session. I recommend entering something that you can remember as it will identify the 4DDL session once it is imported.

When you're ready to import, click "OK".

WARNING: the import process can be lengthy. A progress bar is presented that should give some idea of how long it will take. For the example data (80MB 4DDL session, 1.6 million lines) the import takes some minutes (but not hours).

The import process will import each line of the log file in "raw" form and save it to the [Log_Lines]Data field. Additionally the raw data is de-normalized into corresponding fields in the same table.

These fields are:

- **ID** – a unique incrementing identifier for each log line.
- **DL_ID** – foreign key field to tie the log line to a particular 4DDL session.
- **Data** – the raw, imported line.
- **MS_Stamp** – each line in the 4DDL has a millisecond timestamp. This value is saved here.
- **PID** – the 4D process ID for the process that called the command.
- **UPID** – the Universal Process ID for the process that called the command. UPIDs are just a unique incrementing number for every process the database has created, so that you can differentiate individual instances of the same process.
- **Stack_Level** – the 4DDL contains the stack level for some types of commands. This information is saved in this field, but not currently used.
- **Command** – the actual command that was executed.
- **Execution_Time** – for those commands that have a begin and end entry in the log file, the execution time is calculated and saved here.
- **IsStart** – for those commands that have a begin and end entry, the "begin" entry is indicated by this field (to differentiate it from the end, and also from other commands that don't have begin/end).
- **Cmd_Type** – the type of command that was executed. One of: Project Method; Form Method; Object Method; Plug-in Method; or 4D Command.

After import and de-normalization is complete, the import process calculates the execution times as appropriate. This is done at import time because it is a fairly slow process (you don't want to have to do it more than once, e.g. every time you run a report).

Note: *The 4DDL is accurate to the millisecond. Many commands will execute faster than 1 millisecond so you may notice a lot of 0's in the Execution_Time column. It is important to understand that a 0 is still a valid time. For any commands that do not have an execution time the value will be -1 (so that you can tell the difference).*

Generate Summary for 4DDL

This task generates a report for a given 4DDL session.

Run the method **DBL_M_LogSummary** or select "Debug Log Summary..." from the File menu in application mode.

You will be asked to select the 4DDL session to generate the summary for. This is where the Notes field comes in handy since the only other identification is a unique ID, date, time, and millisecond stamp.

This report generates the following data:

- Top 20 Project methods by execution count
- Top 20 4D Commands by execution count
- Top 20 Plug-in methods by execution count
- Top 20 Form methods by execution count
- Top 20 Object methods by execution count
- Top 20 Project methods by average execution time
- Top 20 Form methods by average execution time
- Top 20 Object methods by average execution time
- Top 20 Project methods by longest execution time
- Top 20 Form methods by longest execution time
- Top 20 Object methods by longest execution time

Note: *If there are less than 20 of any particular kind of method, the list will be shortened (you won't see a bunch of blanks). If you want more than 20, you can change the limit in the method* ***DBL_P_LogSummary****. The total is passed as a parameter to each subroutine.*

The report is generated as a tab-delimited text file. This file can, of course, be opened in a spreadsheet application for further refinement.

When the report is complete, the folder containing the report will be shown. The report files are saved the "Logs" folder of the database. They have a unique name as follows:

```
LogSummary_<4DDL record ID>_<date>_<time>.txt
```

For example:

```
LogSummary_47_7-17-2009_16_12_56.txt
```

API

There are several commands that may be useful for generating different reports. First, here is a description of the prefixes:

- **DBL_** - Prefix for all methods specific to this database
- **DBL_D_** - code that displays dialogs
- **DBL_DATA_** - code that modifies the data
- **DBL_IMPORT_** - code related to importing debug log files
- **DBL_M_** - menu item methods
- **DBL_P_** - process methods
- **DBL_RPT_** - code related to creating reports
- **IAS_** - generic code from our Internal Application Shell

Useful commands:

DBL_DATA_DenormalizeData

You can call this method whenever you want to re-de-normalize the data. Just make sure you've created a selection for the [Log_Lines] table first.

DBL_DATA_CalcExecTimes

This method can be called to re-calculate the execution times of any appropriate methods. Pass the id of the 4DDL session you want to do it for.

DBL_M_Import

Execute this to import a 4DDL session.

DBL_M_LogSummary

Execute this to generate a 4DDL session summary (described previously).

DBL_RPT_GetAverageExecutions

Execute this to generate an array of average execution times for every command in a given 4DDL session.

DBL_RPT_GetCallCounts

Execute this to generate an array of number of executions for every command in a given 4DDL session.

DBL_RPT_GetWorstExecutions

Execute this to generate an array of slowest executions for every command in a given 4DDL session.

DBL_P_LogSummary

Modify this method if you want to change the 4DDL summary report.

IAS_

Anything with this prefix is intended to be a generic method that can be used in any database.

Conclusion

This Technical Note provides a means for using the 4DDL as a method analysis tool. Instructions for the usage of the included database as well as descriptions for used commands are provided to help developers take advantage of this approach and apply it to their own applications.