# The 4DDebugLog.txt file

By Hugo Fournier, Technical Support Manager, 4D, Inc.
TN 06-01

## Introduction

Starting with version 2004.3, a debug log file can now be created by 4D Server, 4D Client and 4th Dimension. This technical note discusses the basic principles and possible uses of this log file.

## Principles

If you have spent enough time writing code, odds are that you have encountered more than your fair share of bugs in your code. When it comes to troubleshooting bugs, most of them can be found by tracing the code in the debugger or simply reviewing your code closely. The idea at work is that if you get to reproduce the problem, it is just a matter of technique, time and patience before you can narrow it down to a statement or a piece of code.

Unfortunately, there are exceptions to that. Problems may occur in compiled mode only or happen once a day and trigger a server crash. The idea behind the log file is to keep track of what is executed by the 4D application. When the application crashes, it should provide you with detailed information on the sequence of commands that lead to the crash.

## The Log file

### Creating the Log file

This option can be activated for any type of 4D application (4th Dimension single user, 4D Server, 4D Client, 4D Runtime), in interpreted or compiled mode. To create the Log file, simply execute the SET DATABASE PARAMETER command using the selector 34:

SET DATABASE PARAMETER (Selector;Option)
Selector is 34
Possible values for Option: 0, 1 or 2 (0 = do not record, 1 = record, 2 = record in detailed mode).

Description: Starts or stops the sequential saving of events occurring at the 4D programming level, intended for debugging the application. By default, the value is 0 (events are not recorded).

**What is recorded?**

Various types of information can be recorded, more particularly:
- For each event, the number of milliseconds since the creation of the file and the process number ([n]).
- The execution of each 4D command (cmd) and each calling of a plugin (plugInName); in this case, the stack level is indicated ( (n) ).
- Each calling of project methods (meth), object methods (obj) and form methods (form).
- When the detailed mode is activated (value = 2), additional information concerning the plug-ins are recorded: events in the plug-in areas (EventCode) and calls to 4D by the plug-ins (externCall). For example, if you implement a drag and drop system between 4D View areas, only option 2 will allow you to log the commands that directly are involved with the drag and drop.

This file is erased and rewritten each time the application is launched. If the 4D application is not relaunched, and you interrupt the recording of the log and restart it, the recording resumes in the original file. The log file remains locked until either the application is shutdown or a SET DATABASE PARAMETER (34;0) is issued.

**Note:** Each event is systematically recorded in the file before its execution, which guarantees its presence in the file even when the application quits unexpectedly.

**Where is the log recorded?**
The events are stored in a file named "4DDebugLog.txt" that is automatically placed:

- next to the database structure file ford 4D Server and 4th Dimension
- in the 4D Preferences folder (C:\Documents and Settings\User\Application Data\4D\DB_Name.4DB_255_255_255_255 on Windows or User/Library/Application Support/4D/DB_Name.4DB_255_255_255_255 on Mac OS) .

**When to use the log?**

This option is intended solely for the purpose of debugging and must not be put into production since it may lead to deterioration of the application performance as well as saturation of the hard disk. A way to work around the hard disk limitation is discussed later in this technical note.

Keep in mind that, even in the basic mode, for each command that is executed by 4D, a new entry is made in the log, which explains the

deterioration of performance. Typically though, you will use the log for situations that cannot be investigated easily through the debugger. A good example would be a server crash when several clients are connected and where knowing what the sequence of commands being executed is at any given time.

## Example

As we mentioned earlier, one of the drawback of the debug log file is that it tends to overwhelm your hard disk in no time on either a busy server or a busy client. In addition to that, most of the time, the only parts of the log you want to take a look at are actually the last hundred lines or so. Based on that, we implemented a piece of code that interrupts the logging, archives the log file and creates a new file. At any given time there should not be more than five consecutive log files on the disk.

This code performs the following:

- Each time the application is launched, the debug file is archived. The reason for doing that is that the log that lead to the crash has to be preserved. Launching the application after the crash and triggering the debug mode again will otherwise erase that file.

- Files are archived with a name that indicates the date and time of the archiving.

- After the first archiving, a process constantly monitors the size of the log. Once it goes past a threshold (500kb, roughly) the debug mode is deactivated, the debug file is archived and the debug mode is enabled again, which creates a new debug file. The code maintains an array of the archive which warrants, through automatic deletion, that you will not have more than five archives at any given time. Please note that, in most cases, only the last archive is of use since it includes the sequence of commands that lead to the crash.

  Also, due to the multi-process nature of 4D, there is a slim chance that the crash occurs when the log is off. Although this is unlikely, you need to always keep this in mind. This is a trade off on the fact that, to reset the log, you have to first disable the logging which releases the access to the log. Once moved, the application can resume and create a new log file.

- The code archives the bug where 4D/4D Server/4D Client creates it. Per se, only the client requires a specific handling since it handles the location of the log differently from the other 4D Applications.

To install this code in your database:

- For 4D Client and 4D Stand alone, simply copy the contents of the On Startup and On exit database methods and re-create the M_Monitor_Process method.

- For 4D Server, simply copy the contents of the On Server Startup and On Server shutdown methods and re-create the M_Monitor_Process method.

### On Startup/On Server Startup method:

This method initializes interprocess variables and detects if 4D Client is used.

```
ARRAY TEXT(<>aPathList;0)
C_TEXT(<>vClientPath)
  `4D Client is looking at a different location
C_BOOLEAN(<>Startup)
C_BOOLEAN(<>vEnd)
SET DATABASE PARAMETER(34;2)
<>vEnd:=False
<>Startup:=True
If (Application type=4)
     `detecting the case of the client
   <>vClientPath:=Get 4D folder(3)
End if
New process("M_Monitor_Process";512*1024)
`starting the archiving monitoring process
```

### On Exit/On Server Shutdown method:

This method initializes an interprocess variables that terminates the monitoring process.

```
<>vEnd:=True
```

### The M_Monitor_Process method:

This method handles the entire archiving process.

```
If (Test path name(<>vClientPath+"4DDebugLog.txt")=1) & (<>Startup)
     ` if log exists at first pass-> archiving
   <>Startup:=False
     `no longer first pass
   $Str_Time:=String(Current time)
   $Str_Date:=String(Current date)
     `getting current save info
   $Str_Time:=Replace string($Str_Time;":";"_")
   $Str_Date:=Replace string($Str_Date;"/";"_")
     `getting rid of problem characters
   SET DATABASE PARAMETER(34;0)
     `disabling the log to release the lock
   MOVE DOCUMENT(<>vClientPath+"4DDebugLog.txt";<>vClientPath+"Archived on "+$Str_Date+" at
"+$Str_Time+".txt")
     `archiving log
   INSERT ELEMENT(<>aPathList;1)
   <>aPathList{1}:=<>vClientPath+"Archived on "+$Str_Date+" at "+$Str_Time+".txt"
     `bye bye log
```

```
   SET DATABASE PARAMETER(34;2)
     `log is back ASAP

   If (Size of array(<>aPathList)=6)
      DELETE DOCUMENT(<>aPathList{6})
      DELETE ELEMENT(<>aPathList;6)

   End if
Else
   ALERT("Log file is absent, the debug mode is not activated")
     `if it were activated the log file would test present
   <>vEnd:=True

End if
Repeat
   DELAY PROCESS(Current process;60)

   If (Get document size(<>vClientPath+"4DDebugLog.txt")>500000)
      $Str_Time:=String(Current time)
      $Str_Date:=String(Current date)
        `getting current save info
      $Str_Time:=Replace string($Str_Time;":";"_")
      $Str_Date:=Replace string($Str_Date;"/";"_")
        `getting rid of problem characters
      SET DATABASE PARAMETER(34;0)
        `  `disabling the log to release the lock
      MOVE DOCUMENT(<>vClientPath+"4DDebugLog.txt";<>vClientPath+"Archived on "+$Str_Date+" at
"+$Str_Time+".txt")
        `archiving log
      INSERT ELEMENT(<>aPathList;1)
      <>aPathList{1}:=<>vClientPath+"Archived on "+$Str_Date+" at "+$Str_Time+".txt"
        `bye bye log
      SET DATABASE PARAMETER(34;2)
        `log is back ASAP

      If (Size of array(<>aPathList)=6)
         DELETE DOCUMENT(<>aPathList{6})
         DELETE ELEMENT(<>aPathList;6)
           `no more than 6 logs
      End if

   End if

Until (<>vEnd)
  `<>vEnd is for clean death of process
```

## Summary

Starting with version 2004.3, a debug log file can now be created by 4D Server, 4D Client and 4th Dimension. This technical note discusses the basic principles and possible uses of this log file. It also illustrates a technique that archives logs on the fly and works around the risk to saturate you hard disk.