The 4DDebugLog.txt file

By Hugo Fournier, Technical Support Manager, 4D, Inc.

TN 06-01

Introduction

バージョン 2004.3 からは、4D Server、4D Client、4th Dimension でデバッグログファイルが出力できるようになりました。このテクニカルノートでは、同ファイルの基本的な仕組みや、考えられる用途について論じています。

Principles

ある程度の期間、プログラミングに携わっている人であれば、相当の時間をバグの退治に費やしてきたはずです。たいていのバグは、デバッガでトレースを実行したり、単純にコードを注意深く調べたりすることによって発見できます。そのようなバグは、再現することができ、時間と技術と忍耐さえあれば、コードの問題部分を特定できるのが特徴です。

残念ながら、そのような分類には属さないバグもあります。コンパイルモードでしか再現しなかったり、いつ起きるかが予測できなかったり、発生と同時にクラッシュを誘発するバグは、上記の方法では退治することができません。そこで登場するのがログファイルです。ログファイルには、アプリケーションが何を実行したかが記録されています。アプリケーションがクラッシュした場合、ログファイルを調べれば、クラッシュに至った経緯を詳しく知ることができ、問題の原因を特定するのに役立てることができます。

The Log file

Creating the Log file

4D アプリケーションは、シングルユーザ、サーバ、クライアント、およびランタイムタイプのいずれであっても、またコンパイル/インタプリタの両モードにおいて、このオプションを利用することができます。デバッグログファイルを出力するには、SET DATABASE PARAMETER コマンドにセレクタ番号 34 を渡します。

SET DATABASE PARAMETER (Selector;Option)
Selector には 34 を渡します。
Option には、0、1、2 のいずれかを渡します。(0=記録しない、1=記録する、2=詳細モードで記録する)

4D プログラミングレベルで発生するイベントの時系列に沿った記録を開始または停止します。主にアプリケーションのデバッグ目的で使用されます。デフォルトの設定は 0(記録しない)です。

What is recorded?

様々な種類の情報が記録の対象になっています。具体的には次とおりです。

- イベントが発生した時刻(ログファイル作成からの経過ミリ秒)、プロセス番号([n])。
- 4D コマンドの実行(cmd)、プラグインコール(plugInName)、スタックレベル((n))
- プロジェクトメソッドコール(meth)、オブジェクトメソッドコール(obj)、フォームメソッドコール(form)

詳細モードで記録する場合は、プラグインに関する付加的な情報が追加されます。

- プラグインエリアイベント(EventCode)、4D コール(externCall)

たとえば、複数の 4D view エリア間でドラッグ&ドロップを実施しようとしている場合、実際の操作に関係のある情報を取得するには、詳細モードでログを記録する必要があります。

デバッグログファイルは、アプリケーションを起動するたびに作成されて上書きされます。アプリケーションを終了することなくログを中断して再開した場合は、元のファイルが使用されます。デバッグログファイルは、アプリケーションをシャットダウンする、または SET DATABASE PARAMETER (34;0)を実行するまでロックされています。

Note:イベントの記録は、実行直前にファイルに書き込まれます。したがって、アプリケーションが予期しない理由で終了した場合でも、ログが記録されていることになります。

Where is the log recorded?

デバッグログは、4DDebugLog.txt という名前のファイルに書き込まれます。ファイルの保存場所は次のとおりです。

- 4D Server、4th Dimension の場合は、データベースストラクチャファイルと同じ階層
- ローカルリソースフォルダ(Windows であれば C:¥Documents and Settings¥User¥Application Data¥4D¥DB_Name.4DB_255_255_255_255、Macintosh であれば User/Library/Application Support/4D/DB_Name.4DB_255_255_255_255)

When to use the log?

デバッグログファイルは、開発目的のオプションであり、パフォーマンスの低下とディスクスペースの逼迫を来たすため、運用時に使用するべきではありません。ハードディスクスペースの問題を回避する方法については、テクニカルノートの後半で論じています。

デバッグログは、基本モードであったとしても、4D のコマンドを実行するたびに更新されるた

め、相応のパフォーマンス低下を予期しなくてはなりません。通常は、デバッガで解決できない問題を追求するためにこのオプションを使用します。たとえば、複数のクライアントが接続している最中にサーバがクラッシュする場合、コマンド実行の流れを調べる場合に効果的です。

Example

前述したように、デバッグログファイルの短所のひとつは、大量のアクセスがあるサーバまたはクライアントであれば、またたく間にディスクスペースを占拠してしまうという点です。さらに加えるならば、ほとんどの場合、肝心な情報は終わりの 100 行程度に集中しています。このことを踏まえ、ログを中断してファイルをアーカイブし、新しいファイルを作成するコードを用意しました。このコードを使用すれば、残されるデバッグログファイルの数が最新の 5 個だけにとどめられるようになります。

コードの内容は次のとおりです。

- アプリケーションが起動すると、デバッグログファイルをアーカイブします。クラッシュ後にアプリケーションを再起動し、デバッグログを再開した時点で、クラッシュに至ったログの記録が失われてしまうため、この処理が必要です。

- アーカイブには、作成日時および時刻を示すファイル名がつけられます。

- 最初のアーカイブが作成されると、ファイルのサイズを監視するプロセスが起動します。サイズが上限の約 500KB を超えた時点で、デバッグログが中断され、ファイルがアーカイブされた後に新しいデバッグログファイルで再開されます。ファイルの数は配列で管理され、最新 5 件を超えるアーカイブファイルは順次、削除されてゆきます。これは、ほとんどの場合、クラッシュに至るコマンドシーケンスを知るには最後のアーカイブだけがあれば充分であるためです。

- 4D のマルチプロセスの仕組み上、ログを中断している間にクラッシュが発生する可能性が拭えません。非常に稀なケースですが、この点を頭の片隅に置いておく必要があります。ログを初期化するためには、一時的にデバッグログを中止する必要があります。アーカイブを作成した後、デバッグログが再開され、新たなデバッグログファイルが作成されることになります。

- デバッグログファイルのアーカイブは、4D/4D Server/4D Client が作成したデバッグログファイルと同じ場所に作成されます。4D Client だけはこの場所が他と異なっています。

データベースにインストールするには、次の手順を踏みます。

- 4D Client および 4th dimension の場合、On Startup データベースメソッドと On Exit データベースメソッドをコピーし、*M_Monitor_Process* メソッドを追加します。

- 4D Server の場合、On Server Startup データベースメソッドと On Server Shutdown データベースメソッドをコピーし、*M_Monitor_Process* メソッドを追加します。

On Startup/On Server Startup method

インタープロセス変数を初期化し、4D Client アプリケーションであるかを確かめます。

```
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

監視プロセスを終了するためにインタープロセス変数をセットします。

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

The M_Monitor_Process method

アーカイブを処理します。

```
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

バージョン 2004.3 以降では、4D Server、4D Client、4th Dimension でデバッグログファイルを出力できるようになりました。このテクニカルノートでは、ファイルの基本的な仕組みや、考えられる用途について論じました。加えて、ログのアーカイブを作成し、ハードディスクの容量を効率良く使用するための回避策についても取り上げました。