# Mirroring with 4D 2004.4, PART II

By Kent Wilbur, Manager Information Systems, 4D, Inc.

Technical Note 06-37

## PartnerWare



## The Mirroring process – Main Server side

The responsibility of actually doing the mirroring falls upon the main server. It periodically creates a new log file then ships that log file off to the mirrored server(s). The method *Mirror_P_MirrorProcess* is a small scheduling method that is simply delayed most of the time. The fact that it is small coupled with the fact that it is always in memory, means this method maintains a relatively small footprint. Most of the code needed resides in other methods that are loaded only when needed. It handles delaying the process and it also handles rescheduling the mirror time based upon changes in the preference settings while the process is delayed. When the client updates the settings, the process wakes up and recalculates when the next mirroring is to take place. When the mirror should run is based upon a date/time stamp mechanism. If the mirroring fails for some reason the method loops and tries again based upon the preferences. Finally, it e-mails any error messages as appropriate.

```
If (False)
    ` Method: Mirror_P_MirrorProcess
    ` 4D Technote on Mirroring a 2004.4 database
    ` Created by: Kent Wilbur
    ` Date: 9/27/2005

    ` Purpose: The Mirroring Process project method

  <>Mirror_f_Version2004x4:=True
  <>Mirror_fK_Wilbur:=True

End if
```

```
Open window(200;200;450;400;0;"Mirror Status")
  ` Declare local varaibles
C_BOOLEAN($fOK)
C_STRING(6;$sVersion)
C_LONGINT($LDelayTicks)
C_LONGINT($LErrorCode)
C_LONGINT($LPosition)
C_LONGINT($LRetryCounter)
C_LONGINT($LSMTPid)
C_TEXT($tDateTimeDelay)
C_TEXT($tMessageText)
C_TEXT($tMirrorScheduleMode)
C_TEXT($Mirror_tLastBackupFolderPath)
C_TEXT($Mirror_tLastBackupID)
C_TEXT($Mirror_tNumberBackupsValue)
C_TEXT($Mirror_tCurrentDiskFreeSpace)
C_TIME($hCurrentTime)
C_TIME($hDelayInterval)
C_TIME($hDelayUntil)

$sVersion:=Application version
Mirror_HandleMirrorPreferences ("Load")

Case of
   : (Application type#4D Server )
      ALERT("Mirroring only works with 4D Server.")
   : (Num($sVersion)<804)
      ALERT("This Mirroring code requires version 2004.4 or higher.")
   Else
      $Mirror_tLastBackupID:=""
      $Mirror_tNumberBackupsValue:=""
      $Mirror_tLastBackupFolderPath:=""
      $Mirror_tCurrentDiskFreeSpace:=""
      Mirror_HandleBackupPreferences ("Load";->$Mirror_tLastBackupID;->$Mirror_tNumberBackupsValue;-
>$Mirror_tLastBackupFolderPath;->$Mirror_tCurrentDiskFreeSpace)
      <>Mirror_LCurrentDiskFreeSpace:=Num($Mirror_tCurrentDiskFreeSpace)

      $tMirrorScheduleMode:=<>Mirror_tScheduleMode
      $hDelayInterval:=<>Mirror_hTimeInterval
      $hCurrentTime:=Current time  ` Set now for use later
      Mirror_tLastBackupTime:=Mirror_tDateTimeStamp (Current date;Current time)

      While ((Not(<>Mirror_fQuit)) & (<>Mirror_tScheduleMode#"Off"))
         Repeat
            Case of
               : ($tMirrorScheduleMode="Interval Only")
                  If ($hDelayInterval#<>Mirror_hTimeInterval)
                     $hDelayInterval:=<>Mirror_hTimeInterval
                  End if

                  If ($hDelayInterval<?00:15:00?)
                     $hCurrentTime:=Current time  ` Less than 15 minutes, set to relative to time last
                                                    mirroring _finished rather than  time began
                  End if

                  $hDelayUntil:=$hCurrentTime+$hDelayInterval
                  If ($hDelayUntil>?24:00:00?)
                     $tDateTimeDelay:=Mirror_tDateTimeStamp (Add to date(Current
date;0;0;1);$hCurrentTime+$hDelayInterval-?24:00:00?)
                  Else
                     $tDateTimeDelay:=Mirror_tDateTimeStamp (Current date;$hDelayUntil)
                  End if
```

```
            Else
               $tDateTimeDelay:=<>Mirror_tNextTimeIntervalMode
               Mirror_HandleMirrorPreferences ("SetNextTimeInterval")
               If ($tDateTimeDelay#<>Mirror_tNextTimeIntervalMode)
                  Mirror_HandleMirrorPreferences ("Save")
               End if

               $tDateTimeDelay:=<>Mirror_tNextTimeIntervalMode
            End case


   $hDelayUntil:=Time(Substring($tDateTimeDelay;9;2)+":"+Substring($tDateTimeDelay;11;2)+":"+Substri
ng($tDateTimeDelay;13;2))

            If (Date(Substring($tDateTimeDelay;5;2)+"/"+Substring($tDateTimeDelay;7;2)+"/"+
               Substring($tDateTimeDelay;1;4))>(Current date))
         ` Next schedule after midnight
               $LDelayTicks:=($hDelayUntil+?24:00:00?)-$hCurrentTime*60
            Else
               $LDelayTicks:=$hDelayUntil-$hCurrentTime*60
            End if
            Mirror_MyDelay (Current process;$LDelayTicks)
            $hDelayUntil:=?00:00:00?
              ` It is possible that a Client machine could be changing the delay mode
            If ($tMirrorScheduleMode#<>Mirror_tScheduleMode)
               ` The mode has been changed need to recalculate time
               $tMirrorScheduleMode:=<>Mirror_tScheduleMode
            End if

         Until ((Mirror_tDateTimeStamp (Current date;Current time)>=
$tDateTimeDelay) | (<>Mirror_tScheduleMode="Off"))

         If (<>Mirror_tScheduleMode#"Off")  ` If we are not  turning off the mirroring
            $fOK:=False
            $tMessageText:=""
            Repeat
               MESSAGE("Starting mirroring: "+String(Current time)+Char(13))  ` JYFH
               $LErrorCode:=Mirror_LSendLogFile

               Case of
                  : ($LErrorCode=0)  ` Mirroring Sucessful
                     MESSAGE("Mirror Completed: "+String(Current time)+Char(13))
                     $fOK:=True
                  :
($LErrorCode#1403) | ($LErrorCode#1407) | ($LErrorCode#1410) | ($LErrorCode#1412) | ($LErrorCode#14
22)  ` Fatal mirroring error
                     <>Mirror_tScheduleMode:="Off"  `Stop the mirroring
                     MESSAGE("Error "+String($LErrorCode)+"during mirroring: "+String(Current
time)+Char(13))
                     $fOK:=True  ` JY: If we stop the mirroring, we should stop trying  ` JYFH Modif
                  : ($LErrorCode#1409) & ($LErrorCode#1411)  ` Unexpected error stop trying this cycle
                     $fOK:=True
                     MESSAGE("Error "+String($LErrorCode)+"during mirroring: "+String(Current
time)+Char(13))
                  Else
                     ` An error of 1411 happens when a critical operation prevents creating the new log file
Transactions or Indexing
                     ` jyfh: Same if error = -17053: backup in progress
                     MESSAGE("Error "+String($LErrorCode)+"during log sent: "+String(Current
time)+Char(13))
                     If ($LRetryCounter<>Mirror_LTransactionRetries)
                        $LRetryCounter:=$LRetryCounter+1
```

```
                Mirror_MyDelay (Current process;3600)
        ` Wait one minute before retrying      jyfh was 360
              Else
                $fOK:=True  ` Stop trying
              End if
           End case

           If (MirrorSOAP_tErrorMessage#"")  ` An error occurred add to the error message
              $LPosition:=Position(MirrorSOAP_tErrorMessage;$tMessageText)
              If ($LPosition=0)
                    ` If the error message is not already in the outgoing message add it to the message
                 $tMessageText:=$tMessageText+MirrorSOAP_tErrorMessage+Char(Carriage return )
              End if
           End if

        Until ($fOK)

         ` Check the backup Disk Size is the minimium size is set
        If (<>Mirror_LMinimumDiskFreeSpace>0)
           $Mirror_tLastBackupID:=""
           $Mirror_tNumberBackupsValue:=""
           $Mirror_tLastBackupFolderPath:=""
           $Mirror_tCurrentDiskFreeSpace:=""
           Mirror_HandleBackupPreferences ("Load";->$Mirror_tLastBackupID;-
>$Mirror_tNumberBackupsValue;->$Mirror_tLastBackupFolderPath;->$Mirror_tCurrentDiskFreeSpace)
           <>Mirror_LCurrentDiskFreeSpace:=Num($Mirror_tCurrentDiskFreeSpace)

           If (<>Mirror_LCurrentDiskFreeSpace<<>Mirror_LMinimumDiskFreeSpace)
              $LErrorCode:=-17054
              Mirror_SOAP_ErrorHandling ($LErrorCode)
              $tMessageText:=$tMessageText+Char(Carriage return
)+MirrorSOAP_tErrorMessage+Char(Carriage return )
              $tMessageText:=$tMessageText+"Structure File: "+Structure file
              $tMessageText:=$tMessageText+"Drive Used for log files and the last log file:
"+<>Mirror_tLastLogNumber
              $tMessageText:=$tMessageText+"Available Space Remaining on the Drive:
"+$Mirror_tCurrentDiskFreeSpace+" MB"
           End if
        End if

        Case of
           : ($LErrorCode=0)  ` Mirroring Sucessful nothing else to do
           : (Length(<>Mirror_tSMTPServer)=0) & (<>Mirror_tSMTPServer#"None")
              ` Nothing to do no place to send error message
           : (Position(".";<>Mirror_tSMTPServer)<1) | ((Position(".";<>Mirror_tSMTPServer))=
(Length(<>Mirror_tSMTPServer)))
      `invalid format for mail server exists (must have a period to be valid and not only a period at the end)
           Else
              `E-Mail Error message
              ` Note: This might be added to an existing e-mail or other messaging system in the database.
              ` If such a system involves saving records
              ` But be sure it is NOT done on the mirrored server side only on the Master server

              ` The problem with this solution is if the SMTP server is down, the message will not be sent,
              ` so an existing messaging system already in the database might be a better solution

           $LErrorCode:=SMTP_New ($LSMTPid)
           $LErrorCode:=SMTP_Host ($LSMTPid;<>Mirror_tSMTPServer)
           $LErrorCode:=SMTP_From ($LSMTPid;<>Mirror_tErrorEMailAccount)
           $LErrorCode:=SMTP_Subject ($LSMTPid;"Error with Mirroring for
"+<>Mirror_tDatabaseName)
           $LErrorCode:=SMTP_To ($LSMTPid;<>Mirror_tErrorEMailAccount)
```

```
            $LErrorCode:=SMTP_Body ($LSMTPid;$tMessageText)
            If (<>Mirror_tAuthenticationRequired="Yes")
               $LErrorCode:=SMTP_Auth
($LSMTPid;<>Mirror_tAuthenticationUserName;<>Mirror_tAuthenticationPassword)
            End if
            $LErrorCode:=SMTP_Send ($LSMTPid)
            $LErrorCode:=SMTP_Clear ($LSMTPid)

         End case

         $hCurrentTime:=Current time  ` Reset for next time interval
      End if
   End while

End case
`End of method
```

Be sure to note that it is possible fatal errors might occur which would make future mirroring impossible. In this case I have stopped the mirroring process from running causing further problems. But, if you haven't entered someplace to send an e-mail, you will never know that the mirroring process has stopped.

The *Mirror_SOAP_LHandleEvents* method actually does most of the work. But it is controlled by the *Mirror_LSendLogFile* method. if first calls *Mirror_SOAP_LHandleEvents* to verify that at least one mirror is present. If a mirror is not currently available, a new log file should not be created. After creating the log file the method determines the type of log file integration that should take place. Integration alone, or integration and backup of the mirrored server.

```
If (False)
      ` Method: Mirror_LSendLogFile
      ` 4D Technote on Mirroring a 2004.4 database
      ` Created by: Kent Wilbur
      ` Date: 5/10/2006

      ` Purpose: Handles creating and sending the log file to mirroring server(s)

   <>Mirror_f_Version2004x4:=True
   <>Mirror_fK_Wilbur:=True

End if

   ` Declare parameters
C_LONGINT($0)

   ` Declare local variables
C_LONGINT($LError)
C_TEXT($tCurrentTime)
C_TEXT($tLastLogFile)
C_TEXT($tIntegrateType)
C_TIME($hAbortTime)
C_TIME($hCurrentTime)

MirrorSOAP_LErrorNumber:=0
MirrorSOAP_tErrorMessage:=""
$LError:=0
```

```
If (Not(<>Mirror_fQuit))
   $hCurrentTime:=Current time
   Case of
      : (Mirror_SOAP_LHandleEvents (<>Mirror_tDatabaseName;"VerifyPresent";
                  <>Mirror_tServerIPAddress)#0)  ` Correct database is not available
         $LError:=-17050
         MirrorSOAP_tErrorMessage:="Mirror database is not available. Mirroring at "+String(
                  Current time;HH MM )+" on "+String(Current date;Short )+" did not take place."

      Else
         $hAbortTime:=Current time+<>Mirror_hTransactionDelay
                              ON ERR CALL("Mirror_HandleMirrorError")
         While (Semaphore("MirroringInProgress"))  ` This semphore is to hold processes from starting
                              transactions while the new log file is being created
            Mirror_MyDelay (Current process;1)  ` Technicaly this line of code should never happen since all
                              other code should only do a test semaphore, but it is a safe way to write the code
         End while
         Repeat
            If (MirrorSOAP_LErrorNumber#0)
               Mirror_MyDelay (Current process;15)  ` If the New log file below was not created wait before
                                                      trying again.
            End if
            MirrorSOAP_LErrorNumber:=0
            $tLastLogFile:=New log file  ` Get the full path name of the log file that has just been closed
         Until ((MirrorSOAP_LErrorNumber=0) | (Current time>$hAbortTime))
         CLEAR SEMAPHORE("MirroringInProgress")  ` Let any process waiting for the semaphore to clear
                                                   continue once the file is created we don't need to lock eveyone
                                                   out to send the file to the mirror machines
               ON ERR CALL("")
   End case

   Case of
      : (MirrorSOAP_LErrorNumber=1409) | (MirrorSOAP_LErrorNumber=1411)  ` Transaction or other
                                          critical operation in progress
         $LError:=MirrorSOAP_LErrorNumber
         Mirror_SOAP_ErrorHandling ($LError)

      : (Length($tLastLogFile)>0)
         Case of
            : (<>Mirror_tBackupScheduleMode="Use Mirror Machine Scheduler")
               $tIntegrateType:="IntegrateLog"

            : (<>Mirror_tBackupScheduleMode="via Main Server Time")
               Case of
                  : (<>Mirror_tScheduleMode="Time Only")
                     $tIntegrateType:="IntegrateLog&Backup"
                  : (<>Mirror_tScheduleMode="Time & Interval") | (<>Mirror_tScheduleMode="Interval Only")
                     Case of
                        : (String($hCurrentTime;HH MM )=String(<>Mirror_hBackupTime;HH MM ))
                           $tIntegrateType:="IntegrateLog&Backup"
                        : (($hCurrentTime+<>Mirror_hTimeInterval)<<>Mirror_hBackupTime)  ` Still time for
                                                                                           another backup
                           $tIntegrateType:="IntegrateLog"
                        : ((<>Mirror_hBackupTime+<>Mirror_hTimeInterval)<$hCurrentTime)  ` Backup already
                                                                                           occurred for this date
                           $tIntegrateType:="IntegrateLog"
                        : (($hCurrentTime><>Mirror_hBackupTime) & ((<>Mirror_hBackupTime+
                                 <>Mirror_hTimeInterval)<$hCurrentTime))  ` This is the time for the backup
                           $tIntegrateType:="IntegrateLog&Backup"
                        Else
                           $tIntegrateType:="IntegrateLog"
```

```
            End case

        End case


    : (<>Mirror_tBackupScheduleMode="via Main Server Interval") |
                    (<>Mirror_tBackupScheduleMode="via Main Server Time & Interval")
      ` Find the next scheduled backup time from the last time it actually took place
      $tCurrentTime:=Mirror_tDateTimeStamp (Current date;$hCurrentTime)
      If ($tCurrentTime><>Mirror_tNextBackupIntervalMode)
         Mirror_HandleMirrorPreferences ("SetNextBackupInterval")
         $tIntegrateType:="IntegrateLog&Backup"
      Else
         $tIntegrateType:="IntegrateLog"
      End if

  End case

  <>Mirror_tLastLogNumber:=$tLastLogFile
  Mirror_HandleMirrorPreferences ("Save")

  $LError:=Mirror_SOAP_LHandleEvents (<>Mirror_tDatabaseName;$tIntegrateType;
                    <>Mirror_tServerIPAddress;$tLastLogFile)  ` Send the log file to the server

End case

End if

$0:=$LError
 `End of method
```

The *Mirror_LHandleEvents* method prepares for the actual connections to the mirroring database(s). Since our code can handle multiple mirrors, it became necessary to create a mechanism to handle the situation of one the mirrors being down, while others remain operating. The situation could be temporary or permanent. Therefore, just because one specified mirrors is down mirroring should continue as long as one is present. The VerifyPresent routine checks until it finds one of the mirrors active. If found it returns to *Mirror_LSendLogFile* (above) which will create the new log file.

```
If (False)
    ` Method: Mirror_LHandleEvents(text;text;text;text)
    ` 4D Technote on Mirroring a 2004.4 database
    ` Created by: Kent Wilbur
    ` Date: 5/10/2006

    ` Purpose: Handles creating and sending the log file to mirroring server(s)

  <>Mirror_f_Version2004x4:=True
  <>Mirror_fK_Wilbur:=True

End if

 ` Declare parameters
C_LONGINT($0)
C_TEXT($1;$tMirrorDatabaseName)
C_TEXT($2;$tMirrorAction)
C_TEXT($3;$tMirrorIPAddress)
C_TEXT($4;$tLastLogFile)
```

```
  ` Declare local variables
ARRAY TEXT($atMirrors;0)
ARRAY TEXT($atLogFiles;0)
C_BOOLEAN($Mirror_fBackupIncluded)
C_BLOB($Mirror_oLogFile)
C_LONGINT($i;$j)
C_LONGINT($LError)
C_LONGINT($LSizeOfArray)
C_TEXT($tLogFileNames)
C_TEXT($tLogFilePath)

  ` Reassign for readability
$tMirrorDatabaseName:=$1
$tMirrorAction:=$2
$tMirrorIPAddress:=$3
If (Count parameters>3)
   $tLastLogFile:=$4
End if

  ` Set default values
$LError:=0

Mirror_Text2Array ($tMirrorIPAddress;->$atMirrors;",")

$LSizeOfArray:=Size of array($atMirrors)

Case of
   : ($tMirrorAction="VerifyPresent")
      For ($i;1;$LSizeOfArray)
         $LError:=Mirror_proxy_SOAP_LHandleEvents ($tMirrorDatabaseName;$tMirrorAction;$atMirrors{$i})
         If (($LError=0) & (MirrorSOAP_LResult#1))  ` Mirror is present
            MirrorSOAP_LErrorNumber:=0
            MirrorSOAP_tErrorMessage:=""
            $i:=$LSizeOfArray
         End if
      End for

   : ($tMirrorAction="IntegrateLog@")
      $Mirror_fBackupIncluded:=($tMirrorAction="IntegrateLog&Backup")
      If ($LSizeOfArray>1)  ` Save the log file to the text files
         For ($i;1;$LSizeOfArray)
            Mirror_HandleMultipleMirrorsXML ("AddLog";$atMirrors{$i};->$tLastLogFile)
         End for

         For ($i;1;$LSizeOfArray)
            Mirror_HandleMultipleMirrorsXML ("Load";$atMirrors{$i};->$tLogFileNames)
            ARRAY LONGINT($aLErrorNumber;0)
            ARRAY TEXT($atErrorMessages;0)
            ARRAY TEXT($atLogFiles;0)
            Mirror_Text2Array ($tLogFileNames;->$atLogFiles;",")

            For ($j;1;Size of array($atLogFiles))
               Case of
                     ` If there are more than one log file to be integrated and the backup is taking place place
                     `backup only when the last log file is sent
                  : ($Mirror_fBackupIncluded & ($j>1) & ($j=Size of array($atLogFiles)))
                     $tMirrorAction:="IntegrateLog&Backup"
                  : ($Mirror_fBackupIncluded & ($j<Size of array($atLogFiles)))
                     $tMirrorAction:="IntegrateLog"
                  Else   ` The scheduled current action is correct
               End case
```

```
            DOCUMENT TO BLOB($atLogFiles{$j};$Mirror_oLogFile)
            If (BLOB size($Mirror_oLogFile)>10000)
               ` If larger than10000 bytes compress the blob to reduce network bandwidth usage
               COMPRESS BLOB($Mirror_oLogFile)
            End if

            $tLogFilePath:=Mirror_tGetFolderPathnames ($atLogFiles{$j})
            If ($tLogFilePath#$tLastLogFile) ` Strips off the full pathname if present
               $tLastLogFile:=(Substring($atLogFiles{$j};Length($tLogFilePath)+1))
            End if
            $LError:=Mirror_proxy_SOAP_LHandleEvents
($tMirrorDatabaseName;$tMirrorAction;$atMirrors{$i};->$Mirror_oLogFile;$tLastLogFile)
                  ` Send the log file to the server
            If ($LError=0) ` Success delete the file from the array
               $atLogFiles{$j}:=""
            Else
               APPEND TO ARRAY($aLErrorNumber;$LError) ` Add any error numbers to the error array
               If (Length(MirrorSOAP_tErrorMessage)=0)
                  Mirror_SOAP_ErrorHandling ($LError)
               End if
               APPEND TO ARRAY($atErrorMessages;$atMirrors{$i}+" - "+MirrorSOAP_tErrorMessage)
                  ` Save the error message along with the server that generated the error
               MirrorSOAP_tErrorMessage:=""
               $j:=Size of array($atLogFiles) ` There was an error we must abort trying this backup
            End if

         End for

         $tLogFileNames:=""
         For ($j;1;Size of array($atLogFiles)) ` See if there were any log files NOT sent and integrated
            If (Length($atLogFiles{$j})>0) ` If so add them so they get sent next time
               If (Length($tLogFileNames)>0)
                  $tLogFileNames:=$tLogFileNames+","
               End if
               $tLogFileNames:=$tLogFileNames+$atLogFiles{$j}
            End if
         End for
          ` Save the log segments still to be sent or wipe and wipe out
          `from the preferences the file that were sucessfully sent
         Mirror_HandleMultipleMirrorsXML ("Save";$atMirrors{$i};->$tLogFileNames)

      End for

      $LSizeOfArray:=Size of array($aLErrorNumber) ` Check for any errors that may have occurred
      If ($LSizeOfArray>0)
         MirrorSOAP_tErrorMessage:=""
         For ($i;1;$LSizeOfArray)
            $LError:=$aLErrorNumber{$i} ` Any error number will do
            MirrorSOAP_tErrorMessage:=MirrorSOAP_tErrorMessage+$atErrorMessages{$i}
         +Char(Carriage return ) ` Combine all the error messages
         End for
      End if

   Else
      DOCUMENT TO BLOB($tLastLogFile;$Mirror_oLogFile)
      If (BLOB size($Mirror_oLogFile)>10000)
       ` If larger than10000 bytes compress the blob to reduce network bandwidth usage
         COMPRESS BLOB($Mirror_oLogFile)
      End if

      $tLogFilePath:=Mirror_tGetFolderPathnames ($tLastLogFile)
```

```
        If ($tLogFilePath#$tLastLogFile)  ` Strip off the full path name if present
           $tLastLogFile:=(Substring($tLastLogFile;Length($tLogFilePath)+1))
        End if
        $LError:=Mirror_proxy_SOAP_LHandleEvents ($tMirrorDatabaseName;$tMirrorAction;$atMirrors{1};-
>$Mirror_oLogFile;$tLastLogFile)  ` Send the log file to the server
        If ($LError#0)
           Mirror_SOAP_ErrorHandling ($LError)
        Else
           Repeat
              $LError:=Mirror_proxy_SOAP_LHandleEvents
($tMirrorDatabaseName;"Statusintegration";$atMirrors{1})
           ` test is integration is over
              DELAY PROCESS(Current process;500)
           Until (($LError#0) | (MirrorSOAP_LResult#2))
        End if
     End if

End case

$0:=$LError
  ` End of method
```

The log is transmitted and the status is checked through a SOAP request. If the status equals 2, the log is being integrated and the process is delayed until an error is returned or 'MirrorSOAP_LResult=1'. This allows us to handle very large logs without encountering timeout problems with the SOAP connection. If the creation of the new log file is successful, the path to the just closed log file segment is passed to the method. Thinking about the problem of multiple mirrors, each log file segment must be integrated in order to each mirror. It is impossible to manage each mirror separately because there is no way to create a log file beginning at a specified point in the log file. Yet, if one of the mirrors is temporarily offline. It still must receive the log file segments in correct order.

My solution was to create an XML preferences file for each mirror machine. The only thing in the preferences file is a list of all the logs that need to be integrated to that specific server. So the first step in integrating the log files into the mirrored servers is to append to each servers XML preferences file the full path to the just closed log file segment.

Then the code will loop through each mirror machine reading the XML preferences for that machine and send in order all log file segments found in the preferences file. Deleting their name from the file once the file has been successfully sent. Normally, the XML preferences has only one entry, the most recent log segment. But if a mirror machine has been off line there might be several segments to send to be integrated. Once the mirror comes back online. Each segment will be sent in order, and the mirror is successfully brought up to date. Note: In the event there are more than one log segment to integrate and the task is to both integrate and backup. The backup will only occur when the last segment has been integrated.

The method Mirror_HandleMultipleMirrorsXML is a simple method similar to the other XML parsing method that was listed earlier in this technical note. Please feel free to glance at it if you are curious about exactly how it works.

Each log file segment is stuffed into a BLOB and sent via SOAP to the mirrored server. (Note: As I mentioned before this uses SOAP, but other mechanisms could be built instead.) Feel free to study the code in the Mirror_proxy_SOAPHandleEvents method, but it was automatically generated by the Web Services Wizard and not modified significantly other than to use a parameter for the location of the mirror server.

```
: ($tMirrorAction="IntegrateLog@")
    $Mirror_fBackupIncluded:=($tMirrorAction="IntegrateLog&Backup")
    If ($LSizeOfArray>1)  ` Save the log file to the text files
        For ($i;1;$LSizeOfArray)
            Mirror_HandleMultipleMirrorsXML ("AddLog";$atMirrors{$i};->$tLastLogFile)
        End for

        For ($i;1;$LSizeOfArray)
            Mirror_HandleMultipleMirrorsXML ("Load";$atMirrors{$i};->$tLogFileNames)
            ARRAY LONGINT($aLErrorNumber;0)
            ARRAY TEXT($atErrorMessages;0)
            ARRAY TEXT($atLogFiles;0)
            Mirror_Text2Array ($tLogFileNames;->$atLogFiles;",")

            For ($j;1;Size of array($atLogFiles))
                Case of   ` If there are more than one log file to be integrated and the backup is
                                                taking place backup only when the last log file is sent
                    : ($Mirror_fBackupIncluded & ($j>1) & ($j=Size of array($atLogFiles)))
                        $tMirrorAction:="IntegrateLog&Backup"
                    : ($Mirror_fBackupIncluded & ($j<Size of array($atLogFiles)))
                        $tMirrorAction:="IntegrateLog"
                    Else   ` The scheduled current action is correct
                End case

                DOCUMENT TO BLOB($atLogFiles{$j};$Mirror_oLogFile)
                If (BLOB size($Mirror_oLogFile)>10000)  ` If larger than10000 bytes compress the
                                                            blob to reduce network bandwidth usage
                    COMPRESS BLOB($Mirror_oLogFile)
                End if

                $tLogFilePath:=Mirror_tGetFolderPathnames ($atLogFiles{$j})
                If ($tLogFilePath#$tLastLogFile)  ` Strip off the full path name if present
                    $tLastLogFile:=(Substring($atLogFiles{$j};Length($tLogFilePath)+1))
                End if
                $LError:=Mirror_proxy_SOAP_LHandleEvents ($tMirrorDatabaseName;
                            $tMirrorAction;$atMirrors{$i};->$Mirror_oLogFile;$tLastLogFile)
                                        ` Send the log file to the server
                If ($LError=0)  ` Success delete the file from the array
                    $atLogFiles{$j}:=""
                Else
                    APPEND TO ARRAY($aLErrorNumber;$LError)  ` Add any error numbers to
                                                                the error array
                    If (Length(MirrorSOAP_tErrorMessage)=0)
                        Mirror_SOAP_ErrorHandling ($LError)
                    End if
                        APPEND TO ARRAY($atErrorMessages;$atMirrors{$i}+
                                    " - "+MirrorSOAP_tErrorMessage)
                  ` Save the error message along with the server that generated the error
```

```
                    MirrorSOAP_tErrorMessage:=""
                $j:=Size of array($atLogFiles)` There was an error we must abort trying
                                                                this backup
            End if

        End for

        $tLogFileNames:=""
        For ($j;1;Size of array ($atLogFiles))  ` See if there were any log files NOT sent
                                                                and integrated
            If (Length ($atLogFiles{$j})>0)  ` If so add them so they get sent next time
                If (Length($tLogFileNames)>0)
                    $tLogFileNames:=$tLogFileNames+","
                End if
                $tLogFileNames:=$tLogFileNames+$atLogFiles{$j}
            End if
        End for
         ` Save the log segments still to be sent or wipe and wipe out from the preferences the
                                                        file that were sucessfully sent
        Mirror_HandleMultipleMirrorsXML ("Save";$atMirrors{$i};->$tLogFileNames)

    End for

    $LSizeOfArray:=Size of array($aLErrorNumber)   ` Check for any errors that may have
                                                        occurred
    If ($LSizeOfArray>0)
        MirrorSOAP_tErrorMessage:=""
        For ($i;1;$LSizeOfArray)
            $LError:=$aLErrorNumber{$i}  ` Any error number will do
            MirrorSOAP_tErrorMessage:=MirrorSOAP_tErrorMessage+$atErrorMessages{$i}+
                    Char(Carriage return )  ` Combine all the error messages
        End for
    End if
```

If there is only one mirror machine. (Probably normal for most uses of mirroring) the log segment is sent to the mirroring server and no XML peferences for each server are needed.

```
    Else
        DOCUMENT TO BLOB($tLastLogFile;$Mirror_oLogFile)
        If (BLOB size($Mirror_oLogFile)>10000)  ` If larger than10000 bytes compress the blob to
                                                        reduce network bandwidth usage
            COMPRESS BLOB($Mirror_oLogFile)
        End if

        $tLogFilePath:=Mirror_tGetFolderPathnames ($tLastLogFile)
        If ($tLogFilePath#$tLastLogFile)  ` Strip off the full path name if present
            $tLastLogFile:=(Substring($tLastLogFile;Length($tLogFilePath)+1))
        End if
        $LError:=Mirror_proxy_SOAP_LHandleEvents ($tMirrorDatabaseName;$tMirrorAction;
                $atMirrors{1};->$Mirror_oLogFile;$tLastLogFile)  ` Send the log file to the server
        If ($LError#0)
            Mirror_SOAP_ErrorHandling ($LError)
        End if
    End if

End case

$0:=$LError
   ` End of method
```

## The Mirroring process – Mirrored Server side

The mirrored server does basically nothing except sit and wait. It isn't even running any processes. If your database has stored procedures that run on the server, you should probably disable them by checking to see if the server is a mirrored server before beginning the stored procedures.

When a SOAP call from the main server hits the SOAP entry method *MIRROR_SOAP_MirrorHandleEvents* the mirrored server goes to work. It returns a 0 if the action succeeded or an error code if the action failed.

It has two main actions. First verify that the mirrored database is present and correctly named. The second is to integrate the log with the option to also backup the database.

```
If (False)
      ` Method: SOAP_MirrorHandleEvents
      ` 4D Technote on Mirroring a 2004.4 database
      ` Created by: Kent Wilbur
      ` Date: 5/26/2006

      ` Purpose: Handles SOAP requests for the Mirroring process

   <>Mirror_f_Version2004x4:=True
   <>Mirror_fK_Wilbur:=True

End if

   ` Declare variables for SOAP Types
C_BLOB(MirrorSOAP_oBLOB)
C_LONGINT(MirrorSOAP_LResult)
C_TEXT(MirrorSOAP_tAction)
C_TEXT(MirrorSOAP_tDatabaseName)
C_TEXT(MirrorSOAP_tLogFile)

   ` Declare local variables
C_LONGINT($LCompressed)
C_LONGINT($LErrorCode)
C_LONGINT($LPid)
C_LONGINT($LPosition)
C_TEXT($tDatabaseName)
C_TEXT($tLogFileBackupID)
C_TEXT($Mirror_tLastBackupID)
C_TEXT($Mirror_tBackupFolderPath)
C_TEXT($Mirror_tStructureFolderPath)
C_TEXT($Mirror_tNumberBackupsValue)

SOAP DECLARATION(MirrorSOAP_tDatabaseName;Is Text ;SOAP Input ;"MirrorSOAP_tDatabaseName")
SOAP DECLARATION(MirrorSOAP_tAction;Is Text ;SOAP Input ;"MirrorSOAP_tAction")
SOAP DECLARATION(MirrorSOAP_oBLOB;Is BLOB ;SOAP Input ;"MirrorSOAP_oBLOB")
SOAP DECLARATION(MirrorSOAP_tLogFile;Is Text ;SOAP Input ;"MirrorSOAP_tLogFile")
SOAP DECLARATION(MirrorSOAP_LResult;Is LongInt ;SOAP Output ;"MirrorSOAP_LResult")

READ ONLY(*)  ` Set tables to read only for now
```

```
$LErrorCode:=0
MirrorSOAP_LErrorNumber:=0

$tDatabaseName:=Structure file
$Mirror_tStructureFolderPath:=Mirror_tGetFolderPathnames ($tDatabaseName)
$tDatabaseName:=(Substring($tDatabaseName;Length($Mirror_tStructureFolderPath)+1))
```

This database uses several of the backup preference settings (set by using 4D's built-in backup preferences. For example: you can specify where you want the log file segment and backup files to be saved by specifying the backup folder location. Note: I found it safer to simply use the backup file location rather than risk getting the mirrors out of sync by using the log file location which requires you to specify a log file

The method *Mirror_HandleBackupPreferences* loads the needed preferences. It is a read only method and does not attempt to modify 4D's backup preferences.

The log file segment has an 8 digit number representing the backup number and segment number. Since backups are not taking place on the main machine and backups may be taking place on the mirror machine the names of the file may not represent where they actually exist in regards to the backup number on the mirror machine. Therefore, the code will change the name of the file to represent the last backup number on the mirror machine so that the segments can correspond to the appropriate backup.

```
If (MirrorSOAP_tDatabaseName=$tDatabaseName)
   Mirror_HandleBackupPreferences ("Load";->$Mirror_tLastBackupID;->$Mirror_tNumberBackupsValue;-
>$Mirror_tBackupFolderPath)

   If (($Mirror_tBackupFolderPath="") | ($Mirror_tBackupFolderPath="/") | ($Mirror_tBackupFolderPath=":"))
      $Mirror_tBackupFolderPath:=$Mirror_tStructureFolderPath
   End if

   Case of
      : (MirrorSOAP_tAction="VerifyPresent")
         If (Test semaphore("$Mirror_BackupInProgress"))  ` Check to see if a backup is in progress
            $LErrorCode:=-17053
         Else
            MirrorSOAP_LResult:=1  ` The database is here
         End if

      : (MirrorSOAP_tAction="IntegrateLog@")
         If (Test semaphore("$Mirror_BackupInProgress"))  ` Check to see if a backup is in progress
            $LErrorCode:=-17053
         Else
            READ WRITE(*)  ` Set tables to read write for mirror integration
            BLOB PROPERTIES(MirrorSOAP_oBLOB;$LCompressed)
            If ($LCompressed#Is not compressed )
               EXPAND BLOB(MirrorSOAP_oBLOB)
            End if
            $LPosition:=Position("[";MirrorSOAP_tLogFile)
            $tLogFileBackupID:=Substring(MirrorSOAP_tLogFile;$LPosition+1;4)

            $len:=Length(MirrorSOAP_tLogFile)
            $LogFileID:=Num(Substring(MirrorSOAP_tLogFile;$len-8;4))
```

```
            Case of
               : ($tLogFileBackupID=$Mirror_tLastBackupID)  ` Do nothing everything is OK
               : (Num($tLogFileBackupID)<Num($Mirror_tLastBackupID))
                  ` Backups are occurring on the mirror machine. The main database can not do backups.
                  ` In order to avoid confusion the integrated log files need to have
                   `their names changed to represent the correct name
                  ` in correspondence to the backup file names
                  ` This will also assist later in the deletion of these files
                   `when the corresponding backup files are purged

   MirrorSOAP_tLogFile:=Substring(MirrorSOAP_tLogFile;1;$LPosition)+$Mirror_tLastBackupID+Substring(
MirrorSOAP_tLogFile;$LPosition+5)
            End case

            ON ERR CALL("Mirror_HandleMirrorError")
            BLOB TO DOCUMENT($Mirror_tBackupFolderPath+MirrorSOAP_tLogFile;MirrorSOAP_oBLOB)

            If (False)  `JYFH
                ` An integration can also be performed from the SOAP connection. However, if the log file
                ` is too big, the SOAP connection will time-out during integration
                ` This is why the integration is performed from a stored procedure
               INTEGRATE LOG FILE($Mirror_tBackupFolderPath+MirrorSOAP_tLogFile)
            Else
               <>doneintegre:=False
               $a:=New
process("Mirror_Integrate";256000;"Mirror_Integrate";$Mirror_tBackupFolderPath+MirrorSOAP_tLogFile)
               If ($a=0)
                  $0:=-1
                  MirrorSOAP_LResult:=0
               Else
                  MirrorSOAP_LResult:=2
               End if
            End if

            ON ERR CALL("")
            If (OK=1)
               $LErrorCode:=0
            Else
               $LErrorCode:=MirrorSOAP_LErrorNumber
            End if

            <>Mirror_tLastLogNumber:=MirrorSOAP_tLogFile
            Mirror_HandleMirrorPreferences ("Save")
            If (MirrorSOAP_tAction="IntegrateLog&Backup")
                  ` The backup needs to take place in it's own process so not to tie down the main server waiting
for a backup to complete
               $LPid:=New process("Mirror_Backup";<>DefaultStackSize;"Mirror_Backup")
            End if
         End if
      : (MirrorSOAP_tAction="Statusintegration")
         MirrorSOAP_LResult:=2
         $0:=0
         For ($i;1;30)
            If (Not(Test semaphore("integrationinprogress")))
               MirrorSOAP_LResult:=1
               $i:=31
            Else
               DELAY PROCESS(Current process;200)
            End if
         End for
   End case
```

```
Else
   $LErrorCode:=-17051
End if

  `MirrorSOAP_LResult:=$LErrorCode
If ($LErrorCode#0)
   Mirror_SOAP_ErrorHandling ($LErrorCode)
End if
  `End of method
```

If an error occurred during integration or another error transpired, the error is interpreted into a error number and message and a SOAP fault is returned to the main database in the method *Mirror_SOAP_ErrorHandling*. Error numbers in the 1403 to 1420 range are 4D Backup error messages. Error numbers from -17050 to -17054 are error number assigned in this code. Please refer to the example database to examine all the errors and their messages

## Performing backups on the main server

In a word DON'T. If you do the mirror will be broken and you will need to start over. The code in the example database will warn you if you are about to break the mirror. The only time you should break the mirror is if you are creating a new mirror. Then you need to break the old one to create a new one.

## Performing backups on the mirroring server

One of the biggest improvements in the 2004.4 backup scheme is to allow backups on the mirroring server without breaking the mirroring synchronization. This provides for a variety of enhanced security measures including backing up the backups for removal to a safe storage place, all without having to shut down anything.

But like every other new feature you need to be careful how you use it. Lets face it, for very large databases, backup takes a while. You don't want to be sending files to the mirroring server while a backup is in progress. It simply will not get integrated. Fortunately, using a simple semaphore when backup begins and clearing it when it ends can also be used to notify the main server not to send a log file right now. Error -17053 from page 39 above. This is one of the main reasons that I recommend controlling the backups from the main database rather than the backup scheduler on the mirroring machine.

The *Mirror_OnBackupStartup* method should be called from the database method *On Backup Startup*.

```
If (False)
      ` Databse Method: Mirror_OnBackupStartup
      ` 4D Technote on Mirroring a 2004.4 database
      ` Created by: Kent Wilbur
```

```
      ` Date: 5/18/06

      ` Purpose:  Alerts the user that if they continue on the main machine the mirror will be broken and
                        returns an error code

    <>Mirror_f_Version2004x4:=True
    <>Mirror_fK_Wilbur:=True

End if

  ` Declare parameters
C_LONGINT($0)  ` Abort process if necessary

  ` Declare local variables
C_LONGINT($LProcessID)
C_LONGINT($LErrorCode)

$0:=0  ` Default allow backup to continue

Case of
    : (Undefined(<>Mirror_tServerType))  ` This can occur on launching a database with an unmatched log
                                                and backup.
        `Forcing a backup to occur before any code is run, therefore there is no defined variable in
            interprted mode

    : (<>Mirror_tServerType="Mirror")  ` OK to continue the Mirror machine is allowed to backup the
                                            datafile.
        If (Semaphore("$Mirror_BackupInProgress"))  ` Set the semaphore
            $0:=-17053
        End if

    : (Length(<>Mirror_tServerType)>0)  ` This database is part of a mirror
        CONFIRM("This database is part of a mirror. Continuing the backup will break the Mirror!";
            "Abort Backup";"Break the Mirror")
        If (OK=1)
            $0:=-17052
        End if

End case
  `End of method
```

Second, although 4D Backup will delete old backup files and the log files created by the backup, it will not delete the log segment files, it simply doesn't know about them. That is why I painstakingly rename them in the mirroring machines. So that I can delete them when the backup is finished and they are no longer needed because even a restore can't use them if the oldest backup available is newer than the log segments.

The *Mirror_OnBackupShutdown*method should be called from the database method *On Backup Shutdown*.

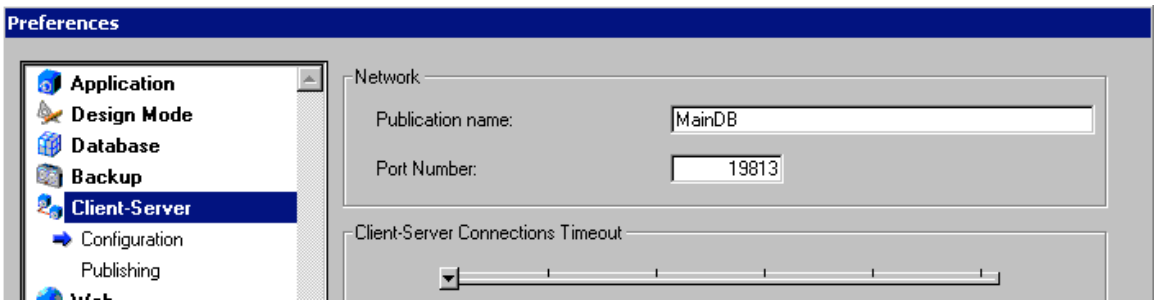
```
If (False)
      ` Method: Mirror_OnBackupShutdown
      ` 4D Technote on Mirroring a 2004.4 database
      ` Created by: Kent Wilbur
      ` Date: 5/18/06

      ` Purpose:  Handles deleting the  log files that have been integrated need to be purged
```

```
      <>Mirror_f_Version2004x4:=True
      <>Mirror_fK_Wilbur:=True

End if

  ` Declare local variables
C_LONGINT($i)
C_LONGINT($Mirror_LOldLogID)
C_LONGINT($Mirror_LThisLogID)
C_LONGINT($LPosition)
C_TEXT($Mirror_tLastBackupFolderPath)
C_TEXT($Mirror_tLastBackupID)
C_TEXT($Mirror_tLogFilePrefix)
C_TEXT($Mirror_tNumberBackupsValue)

Case of
      : (<>Mirror_tServerType="Mirror")
            CLEAR SEMAPHORE("$Mirror_BackupInProgress")

              ` On the mirror machine integrated log files may be piling up and need to be deleted
            $Mirror_tLastBackupID:=""
            $Mirror_tNumberBackupsValue:=""
            $Mirror_tLastBackupFolderPath:=""
            Mirror_HandleBackupPreferences ("Load";->$Mirror_tLastBackupID;
                  ->$Mirror_tNumberBackupsValue;->$Mirror_tLastBackupFolderPath)

            If (Num($Mirror_tLastBackupID)>Num($Mirror_tNumberBackupsValue))
                  $Mirror_LOldLogID:=Num($Mirror_tLastBackupID)-Num($Mirror_tNumberBackupsValue)
                  ARRAY TEXT($atDocuments;0)

                  DOCUMENT LIST($Mirror_tLastBackupFolderPath;$atDocuments)  ` Get the possible files

                  $LPosition:=Position("[";<>Mirror_tLastLogNumber)
                  $Mirror_tLogFilePrefix:=Substring(<>Mirror_tLastLogNumber;1;$LPosition)

                  For ($i;1;Size of array($atDocuments))

                        If ((Position($Mirror_tLogFilePrefix;$atDocuments{$i})>0) &
                              (Position(".4DL";$atDocuments{$i})>0))  `
                                    Is this a log file segment for this database?
                              $LPosition:=Position("[";$atDocuments{$i})
                              $Mirror_LThisLogID:=Num(Substring($atDocuments{$i};$LPosition+1;4))
                              If ($Mirror_LThisLogID<=$Mirror_LOldLogID)  ` This is an old log file that should be
                                                                                                    deleted
                                    DELETE DOCUMENT($Mirror_tLastBackupFolderPath+$atDocuments{$i})
                              End if
                        End if
                  End for

            End if

End case
  `End of method
```

## Keeping users out of the mirrored database

Both the main server and the mirrored server are online. It is quite possible that a user might accidentally log into the mirrored server. There are a couple of things that you can do to prevent this from happening.

If you are using a built application, you will need to be sure that you reassign the database publication name for the mirroring server(s). Otherwise the built application client could easily mistake the mirroring server for the real server on the network. However, this could make recovering from a disaster harder, unless you keep a copy of the built server structure for the real operational server on the mirroring machine as well, so that you don't have to rebuild the database to change the Publication name.




However, a simpler solution is to deny access to the mirror machine in code. Each connection to a server must go through one of two database methods to gain access to the database. On Server Open Connection and On Web Authenticaion. Use a simple test in each of these methods to see if this is the mirroring machine and deny access if it is.

```
If (False)
	` Database Method: On Server Open Connection
	` 4D Technote on Mirroring a 2004.4 database
	` Created by: Kent Wilbur
	` Date: 5/23/2006

	<>Mirror_f_Version2004x4:=True
	<>Mirror_fK_Wilbur:=True

End if

C_LONGINT($0)
$0:=Mirror_OnServerOpenConnection
  `End of method

If (False)
	` Method: Mirror_OnServerOpenConnection
	` 4D Technote on Mirroring a 2004.4 database
	` Created by: Kent Wilbur
	` Date:5/23/2006

	` Purpose: Doesn't allow connections into the 'Mirror' database

	<>Mirror_f_Version2004x4:=True
	<>Mirror_fK_Wilbur:=True

End if

  ` Declare parameters
C_LONGINT($0)
```

```
$0:=0  ` Default everything OK
If (<>Mirror_tServerType="Mirror")
      $0:=-17001  ` Don't allow connections to the mirrored server. Data might get accidentally changed
End if
  `End of method
```

The only thing I don't like about this solution is the error message on the client machine. Because it is 4D rejecting the connection at its lowest level and you can't modify 4D to understand your own error messages at this level. 4D Client interprets any error here as not enough client licenses.

The On Web Authentication is slightly more complex because as we are using SOAP we must allow through all calls to our method for log integration while rejecting everything else.

```
If (False)
        ` Database Method: On Web Authentication
        ` 4D Technote on Mirroring a 2004.4 database
        ` Created by: Kent Wilbur
        ` Date: 5/23/2006

      <>Mirror_f_Version2004x4:=True
      <>Mirror_fK_Wilbur:=True

End if

C_BOOLEAN($0)
$0:=Mirror_OnWebAuthentication
  `End of method


If (False)
        ` Method: Mirror_OnWebAuthentication
        ` 4D Technote on Mirroring a 2004.4 database
        ` Created by: Kent Wilbur
        ` Date: 5/23/06

        ` Purpose:  Rejects invalid web connections

      <>Mirror_f_Version2004x4:=True
      <>Mirror_fK_Wilbur:=True

End if

  ` Declare parameters
C_BOOLEAN($0)

Case of
      : (Not(Is SOAP request))
            $0:=False
      : (Get SOAP info(SOAP Method Name )#"Mirror_SOAP_HandleEvents")
            $0:=False
      Else
            $0:=True
End case
```

This has proven to work reliably. Although you might choose to intercept a Web connection differently and send a more user friendly error web page.

## What to do about those pesky transactions

Earlier we talked about transactions and how to set up the preferences to determine how long to wait for existing transactions to finish etc. Now lets look at code on either the Client side or server side in stored procedures, etc. First of all, never use Automatic transactions for anything in a mirrored server. Next, before you start a transaction look to see if a **New log file** command is in progress. If you remember we wrappered the creating of a new log file segment with a semaphore.

Anyplace you use the command START TRANSACTION you whould wrapper that code with a check for the semaphore. Her is an example of the form method for Customers. (OK, so a transaction really isn't needed in Customers, it's just an example database.)

```
If (False)
      ` Form Method: [Customers];"Input"
      ` 4D Technote on Mirroring a 2004.4 database
      ` Created by: Kent Wilbur
      ` Date: 5/23/2006

   <>Mirror_f_Version2004x4:=True
   <>Mirror_fK_Wilbur:=True

End if

C_LONGINT($LFormEvent)
$LFormEvent:=Form event

Case of
   : ($LFormEvent=On Load )
      While (Test semaphore("MirroringInProgress"))  ` This is to hold starting transactions while the new
                                                          log file is being created
         MESSAGE("Mirroring In Progress. Please Wait.")
         Mirror_MyDelay (Current process;60)  ` Wait one second before trying again
      End while
      START TRANSACTION

   : ($LFormEvent=On Validate )
      VALIDATE TRANSACTION

   : ($LFormEvent=On Unload )
      If (In transaction)
         CANCEL TRANSACTION
      End if

End case
```

This example is crude and I'm sure you will want something more sophisticated but you get the idea.

## Recovering from a disaster - small

The mirroring system is designed for quick recovery from small disasters. For many, my definition of small would be a major disaster. By small disaster I mean that the main server becomes inoperative. All data is lost. Recovery is possible but will take hours to restore from the backups.

If you can access the hard drive for the main server and retrieve the last open log file that has not yet been integrated into the database, recovery is easy. The last log file needs to be integrated into the mirrored database. Below are the specific steps for the example database. Anything you create will need to include at least some of these steps.

1) Simply move the Log file to a 4D Client machine in a folder by itself.
   This is necessary because the last log file will need to be merged into the mirrored machine before it can be set up as the main server. The command **INTEGRATE LOG FILE** can only occur on the Server. Therefore, the 4D Client machine will temporarily tack the place of the crashed main 4D Server and send the last file to the mirrored server for integration.

2) Launch the 4D Client and log into the mirrored database.
   Depending on the security measures chosen above it may be necessary to change the port number of either the server or the 4D Client.

3) Go to the user environment and execute the method *E_RestoreDataFromMirrorLogs.*
   We will review this code in the next section.

4) Select the log file to be merged.

5) Go to the server and change the port number if still necessary.

6) Shut down the mirrored server.

7) Trash the mirror prefferences.

8) Launch the server and select the server to be the main database.
   You are back in business.

If you can't recover the last open log file you will have to do without that data. If this is the case you can skip steps 1 through 4 above and proceed with step 5.

## Recovering from a disaster - major

The worst possible scenario is that both the main server and the mirrored server data files are both damaged. In this case if you are doing periodic

backups on the mirroring machine you can use the most recent backup, otherwise, you must go back to the original backup and begin from there. (Hope you didn't throw away any of those log file segments because the drive got full.) In any even, the backups do not contain any of the log file segments created since the backup was created. It doesn't matter which database is used. Follow the steps below to recover from the major disaster.

1) Restore the database from your chosen backup. (original or hopefully more recent)
2) Move all the Log files created after the backup you are using to a 4D Client machine in a folder by themselves.
3) Launch the 4D Client and log into the server.
4) Go to the user environment and execute the method *E_RestoreDataFromMirrorLogs.*
5) Select one of the log files to be merged.
6) After merging, do a backup and begin everything again as you set up the mirror in the first place. You are back in business.

Note: It might be necessary to separately merge the final log file as you did in the steps above in the recovery from a minor disaster minor. As the code does not distinguish between active and closed log segments in the same pass.

```
If (False)
      ` Method: Mirror_E_RestoreDataFromLogs
      ` 4D Technote on Mirroring a 2004.4 database
      ` Created by: Kent Wilbur
      ` Date: 5/23/2006

      ` Purpose:  Integrates log files into a database restored from a backup
      ` Not the mirror

    <>Mirror_f_Version2004x4:=True
    <>Mirror_fK_Wilbur:=True

End if

  ` Declare local variables
ARRAY TEXT($atDocuments;0)
ARRAY TEXT($atLogFiles;0)
C_BLOB($Mirror_oLogFile)
C_LONGINT($i)
C_LONGINT($LApplicationType)
C_LONGINT($LError)
C_LONGINT($LPid)
C_LONGINT($LPosition)
C_TEXT($tDatabaseName)
C_TEXT($tDatabasePath)
C_TEXT($tFileName)
C_TEXT($tFolderPath)
C_TIME($hDocRef)

$LApplicationType:=Application type

Case of
    : ($LApplicationType=4D Client )
```

```
ALERT("Please locate a log file to be integrated.")

$hDocRef:=Open document("";"";Get Pathname )  ` show all files, but if a log file is not chosen
                                                this will not work

If (OK=1)
   CLOSE DOCUMENT($hDocRef)
   $tFolderPath:=Mirror_tGetFolderPathnames (Document)
   $tFileName:=Substring(Document;Length($tFolderPath)+1)  ` Get the name of the log file
                                                              chosen

    ` Get necessary information from the server
   MirrorSOAP_tDatabaseName:=""
   Mirror_tThisServerIPAddress:=""
```

Even though we are running on a 4D Client machine of the server it is easier to integrate the log file using the same SOAP methods used by the main Server. The integration code (SOAP) needs both the name of the database and IP address of the "mirrored" server. This data is NOT in the mirror preference settings. So we will use some inter-process communication between the Client and the Server and as the Server to look up the data for themselves. The code above reads the values using **GET PROCESS VARIABLE**.

```
$LPid:=Execute on server("Mirror_E_RestoreDataFromLogs";32000)
Repeat
   Mirror_MyDelay (Current process;2)  ` Wait for the server to get the information
   GET PROCESS VARIABLE($LPid;MirrorSOAP_tDatabaseName;
         MirrorSOAP_tDatabaseName)
   GET PROCESS VARIABLE($LPid;Mirror_tThisServerIPAddress;
         Mirror_tThisServerIPAddress)
Until (Length(MirrorSOAP_tDatabaseName)>0)
```

Log segements have a specific format [0000-0000]. If we are merging multiple segments we will look for that format and merge all files that match this pattern in their file name and have the correct main backup number.

```
$LPosition:=Position("[";$tFileName)
If ($LPosition+10=Position("]";$tFileName))  ` Make sure we are getting mirror log files for
                                               merging not regular log files
    ` Now we need to find all corresponding log files
   DOCUMENT LIST($tFolderPath;$atDocuments)

   For ($i;1;Size of array($atDocuments))

      Case of
         : (Length($tFileName)#Length($atDocuments{$i}))  ` Not the same length,
                                                            not a log file
         : (Substring($tFileName;1;$LPosition+5)#Substring($atDocuments{$i};
               1;$LPosition+5))  ` Not the same Datafile and backup number
         : (Substring($tFileName;$LPosition+10)#Substring($atDocuments{$i};
               $LPosition+10))  ` Not a log file
         Else
            APPEND TO ARRAY($atLogFiles;$atDocuments{$i})
      End case
   End for

   If (Size of array($atLogFiles)>0)  ` Integrate the log files
      SORT ARRAY($atLogFiles)
```

```
            For ($i;1;Size of array($atLogFiles))

                DOCUMENT TO BLOB($tFolderPath+$atLogFiles{$i};$Mirror_oLogFile)
                If (BLOB size($Mirror_oLogFile)<10000)  ` If larger than10000 bytes
                            compress the blob to reduce network bandwidth usage
                    COMPRESS BLOB($Mirror_oLogFile)
                End if
                $LError:=Mirror_proxy_SOAP_LHandleEvents (MirrorSOAP_tDatabaseName;
                    "IntegrateLog";Mirror_tThisServerIPAddress;->$Mirror_oLogFile;
                    $atLogFiles{$i})  ` Send the log file to the server
                If ($LError#0)
                    ALERT("Big Problem!")
                    $i:=Size of array($atLogFiles)
                End if
                SET BLOB SIZE($Mirror_oLogFile;0)  ` Clean up memory

            End for
        End if
```

Merging a single log file without the closed log format of [0000-0000] is handled here.

```
    Else   ` Merging single log file
        DOCUMENT TO BLOB($tFolderPath+$tFileName;$Mirror_oLogFile)
        If (BLOB size($Mirror_oLogFile)<10000)  ` If larger than10000 bytes compress the blob
                                        to reduce network bandwidth usage
            COMPRESS BLOB($Mirror_oLogFile)
        End if
        $LError:=Mirror_proxy_SOAP_LHandleEvents (MirrorSOAP_tDatabaseName;
            "IntegrateLog";Mirror_tThisServerIPAddress;->$Mirror_oLogFile;$tFileName)
                                            ` Send the log file to the server

        If ($LError#0)
            ALERT("Log file not merged!")
        End if
        SET BLOB SIZE($Mirror_oLogFile;0)  ` Clean up memory

    End if
End if
```

This is where the 4D Server looks up the information about itself and saves the information into some process variables. Once save the process delays for a few seconds then dies. During those few seconds the code above reads the contents of the process variables.

```
: ($LApplicationType=4D Server )
    MirrorSOAP_tDatabaseName:=""
    $LError:=IT_MyTCPAddr (Mirror_tThisServerIPAddress;$tFileName)
    $tDatabaseName:=Structure file
    $tDatabasePath:=Mirror_tGetFolderPathnames ($tDatabaseName)
    MirrorSOAP_tDatabaseName:=(Substring($tDatabaseName;Length($tDatabasePath)+1))
    Mirror_MyDelay (Current process;300)  ` Leave process alive long enough to get variable values
                                        to the client

Else
    ALERT("Log files can only be integrated from a 4D Client machine.")
End case
 ` End of method          ` Date: 11/1/2005
```

## Summary

Creating a mirrored system is no longer a database administrator task. It requires implementation by the developer. However, it isn't a difficult task to complete. Administrators of a 4D Database often had trouble establishing a mirror, so the developer was usually involved anyway.

The new mirroring commands make the task more flexible that in previous versions of 4D. And a user friendly interface can easily be created to administer the task.

Although mirroring is possible with 2004.2, I would recommend that you use at least 2004.4 which fixed several minor bugs and added some new features to get around some built-in limitations.

Most 4D databases don't need mirroring. But for those that do, take a look at the component include with this database. It might be just what you need.