Mirroring with 4D 2004.3

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

Technical Note 05-38

Introduction

Mirroring used to be built into 4D Backup before 4D 2004. When 2004 was released, the backup operation was integrated into the 4D application. In the initial release of 4D 2004 mirroring was not implemented. 4D 2004.3 has introduced two new language commands that permit creating a mirroring backup system in your 4D 2004 server. This technical note will discuss these new commands and how to set up a mirroring system.

The purpose of this technical note is to introduce the new commands to enable you to implement a mirror. It is not an attempt to write the perfect solution for your mirroring needs. You will need to modify the code presented here to suit your own specific requirements.

In addition to the basic purpose of the technical note I've had to include some other interesting techniques just to properly demonstrate mirroring. Techniques like creating and maintaining preference settings in an XML file. And modifying preference settings on the server from a 4D Client, which will require some custom Client-Server communications.

What is mirroring?

Mirroring is the concept of having a second identical database running in parallel to the database in active use. Its purpose is to provide the quickest possible solution in the event that the main database become inoperative for any reason. Because mirroring keeps the database synchronized if the main database fails, the mirror database will be up to date as of the last synchronization. If the log file that contains everything since the last synchronization on the main server is recoverable, all that needs to happen is to synchronize those last few entries and you can be up and running again using the mirrored server as the primary server.

Since the mirrored server is already a server and is already running the whole process can literally be complete in a few minutes and the database will be fully functional again. No restoring from backups. Mirroring is the fastest way to get a backup server acting as the primary server possible.

Does the mirrored database have any special requirements?

Prior to 2004 the answer was yes. The machines involved really needed to be almost identical. Same hard drive partitions, same operating system, literally 'mirrors' of one another. In the 4D 2004 implementation, this is no longer true. In 4D 2004.3 mirroring can even take place cross platform. The only requirement of the mirror machine is that it absolutely can not modify data on its own. So you will need to be sure that your On Startup doesn't modify any data automatically. And also be very sure that nobody accidentally logs into the database and does something.

Log files

At the heart of both the old mirroring system and the 2004 mirroring system are log files. To set up a mirror you simply backup the database and begin a log file. Shut down the server and then transfer all the files to a mirror machine. To create a mirror all the files must be identical. Then both the mirrored server and the original server are restarted.

To maintain the mirror, periodically on the original server a new log files is created and the one just closed is sent to the mirror server where it is integrated. This process can be continued indefinitely maintaining both the original and complete duplicate copy on the mirrored server.

Log file numbering scheme

Each backup and log from a backup gets a unique number. If you are familiar with using Backup on 4D Server you should easily recognize them. Ex: MyDatabase[0739].4BK and MyDatabase[0738].4BL.

When a mirroring system is involved each transfer of the log to the mirrored server creates an additional segmented log file. In order to identify these logs they are given an additional segment number in the name of the file. MyDatabase[0739-0001].4BL; MyDatabase[0739-0002].4BL; MyDatabase[0739-0003].4BL; etc. It is these closed segmented log files that are sent to the mirrored machine for integration.

New log file function

The **New log file** function closes the current log file, renames it, and creates a new log file at the same location as the original log file. This is the same action that takes for a log file during a normal BACKUP command. The difference is that a backup does not take place. Only the new log file.

The new log file function returns the full Path Name of the log file that has just been closed and renamed. This name contains the last backup number and segment number as described above.

The function only executes on a 4D Server. If the command is not executed on 4D Server an error code of 1412 is generated and nothing else happens. The function also requires that a log file be active. If there is no log file active and error of 1403 is returned.

While the **New log file** function is operating it stops all other processes on the 4D Server.

Transfer the log file segment

Once a log file segment is created and closed it is transferred to the mirrored server. How the transfer is accomplished is up to the developer. It could be transferred manually/or programmatically using shared network volumes. Using the 4D Internet commands you could create an ftp file transfer mechanism, then use 4D Open to tell the mirrored database to integrate the ftp'd file. In the example database provided, I have chosen to use 4D SOAP call to transfer and integrate log files. I chose 4D SOAP as it was the simplest solution and provided an easy way to communicate success or failure of the log integration back to the main database.

Backup and log from a backup gets a unique number. If you are familiar with using Backup on 4D Server you should easily recognize them. Ex: MyDatabase[0739].4BK and MyDatabase[0738].4BL.
during a normal BACKUP command. The difference is that a backup does not take place, only the new log file.

INTEGRATE LOG FILE

The INTEGRATE LOG FILE command starts the integration of the log file given to the command. The command closes the existing log file and reads the entries from the log file and merges them into the datafile. It then makes the log file segment just read its current log file. Mirroring is now complete and the two databases are synchronized.

However, several problems might occur along the way. Firs, the command will only work on a 4D Server. If it is not running on a 4D Server the file is not integrated and an error of 1412 is generated. Like its counterpart, there must be a current activated log file. If there is no current log file a 1403 error is generated.

These two problems are easily overcome. But the bigger problem is if the files got out of sync. Internal to both the backup and log files are a set of numbers that for lack of a better term I will call sequence verification numbers. For successful integration to take place both the backup an logfile segment internal sequence verification numbers must be correct. You can't change these internal numbers simply by renaming the file. These numbers are imbedded into the files themselves. These numbers insure that the log file being integrated is the correct 'next' log file to go with the appropriate 'backup'. If anything happens to disrupt the correct sequence of database events in the log file, the integration of the log file will not occur and an error code of 1410 is generated which in sort says: "Wrong log file. The log file exists, but does not correspond to the open database."

ERROR HANDLING

The process of mirroring without problems is essential to those attempting to set up a mirror. Problems can sometimes occur. Some harmless, and mirroring should continue at the next available scheduled time. Some errors are fatal. No matter what the error, it is essential that proper error handling be observed. If you are not using an **ON ERROR CALL** routine any time you use one of the mirroring commands and an error occurs, the process will abort without completing the task and also without doing anything or even notifying you about the potential problem. In addition if the process is a self delaying process, since it aborted, it will not run the next time it should. It is absolutely essential to create and use an **ON ERROR CALL** routine when using any of the mirroring commands.

The example database

The example database is designed to demonstrate a simple mirroring system. It requires two 4D servers running 4D 2004.2 or higher. It contains some simple preference settings for scheduling mirroring, transferring and merging the log files, error handling, and disaster recovery.

Creating the example database mirror

In order to do mirroring, you must first have log files established. Then you transfer the database to a mirrored machine. Establishment of a mirror requires that you follow several exact steps.

1) Launch the database. (This database uses preference settings, these need to be selected when the database is first launched. For this database choose the 'Main DB' setting.)
2) Perform a backup of the data. (Note: It would be a good idea to include the plugins folder in the backup)
3) Go to the backup preferences and create a new log file.

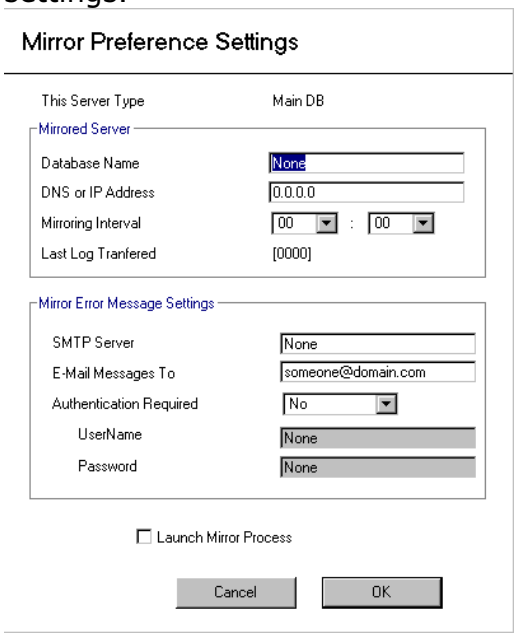
4) Shut down the database.
5) Transfer everything, structure, data files, backups, log files, etc. to a second server machine.
6) In the Preferences folder locate the Mirror folder and trash the folder.
7) Launch the mirror database on a 4D Server and this time choose the 'Mirror' setting. The mirror is now ready to go.
8) Launch the 'Main DB' on a 4D Server.

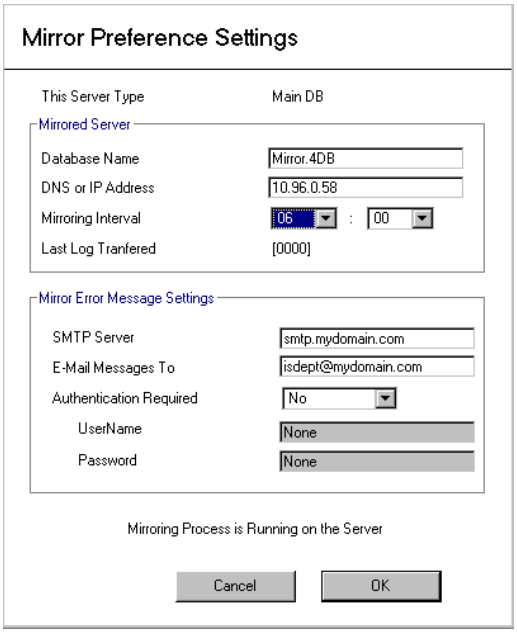
Any time the mirror is broken, you must go back and recreate the mirror from scratch, beginning with the backup of the data. The example database uses preference settings which may or may not be present in what you create. So those steps might be skipped in your database.

The mirror preference settings

I felt it was important to distinguish one database from the other. Since you can NOT use data to tell the databases apart, that would break the mirror, I chose an external preference file. It is a simple XML file that stores the settings. For the mirrored server the only setting used is the fact that it is the Mirror server. Each time the mirrored server is recreated simply trash the preferences and choose the 'Mirror' setting and that is all the mirror database needs.

On the other hand, the Main server does all the work. It needs far more settings.





For security purposes the example database requires the entry of the mirrored database name. I use this internal name to make sure that the mirrored database is running, and that the one you are trying to send the log

to is the correct database, before I even call the **New log file** function. I don't want to create a new log file that has nowhere to go.

The second setting necessary is the IP address of the mirrored server. If you prefer you could use a host name available with a DNS lookup.

Next is the Mirroring Interval. This is in hours and minutes. If the time you select is 00 Hours and 00 Minutes the mirroring of the server will stop. In a mirroring system you need some sort of timing mechanism. Perhaps, you would rather build one that works on certain times of the day, like the scheduler for the backup system available in 4D Server. If, so, I'll let you build it on your own.

The Last Log Transferred is for informational purposes only. It can not be changed. It indicates the last log successfully integrated into the mirror server. This will aid in disaster recovery if necessary.

The next section of settings are for an SMTP server and e-mail address where you would like to send error messages. If you leave this to the default settings of 'None' no messages will be sent.

Finally, at the bottom of the dialog is a status indicator for the mirroring process on the server. If it is not running you can select the checkbox to launch the mirroring process on the server. It will launch, provided that you have entered a mirroring interval. If the process is already running it will be updated with your revised values.

Mirror preference settings – Server side

The mirror preference settings are saved and used on the server. Since 4D already creates a preferences folder for Backup and other settings the database will use the same architecture to create its own preferences and store them in an XML file.

Determining the location of the file: This is done in a single method. If you want to change the location, the string found here is simply modified to the location where you would like the file saved. Don't worry that the values appears to be a windows format. The code will convert Windows to Mac if needed.

```
If (False)
    ` Method: Mirror_tMirrorPath
    ` 4D Technote on Mirroring a 2004 database
    ` Created by: Kent Wilbur
    ` Date: 9/27/2005

    ` Purpose: Contains the location of the Mirror path
```

```
   <>f_Version2004x2:=True
   <>fK_Wilbur:=True

End if

  ` Declare parameters
C_TEXT($0)

$0:="Preferences\\Mirror\\Settings.xml"
  `End of method
```

Loading/Creating the preferences file: When the 4D Server launches the Mirroring process which attempts to load the preferences file using the *Mirror_HandleMirrorPreferences* method. All of the manipulation of the preference file itself occurs in the single recursive method *Mirror_HandleMirrorPreferences*, so it might be a little confusing to read at first. Try first following the code in the method below assuming that the preference file and folders do not even exist using the portion of the case statement where $tAction= "Load" located on page 9.

Following through the code with using 'Load' will soon call the method again for 'create', simply jump back to the top of the method and follow it through the second time with $tAction="Create".

```
If (False)
     ` Method: Mirror_HandleMirrorPreferences
     ` 4D Technote on Mirroring a 2004 database
     ` Created by: Kent Wilbur
     ` Date: 9/27/2005

     ` Purpose: Creates a preferences file for the mirror

     ` Parameters
     `   $1   -   Action

   <>f_Version2004x2:=True
   <>fK_Wilbur:=True

End if

  ` Declare parameters
C_TEXT($1;$tAction)

  ` Declare local variables
C_BOOLEAN($fAbort)
C_STRING(16;$sXML_Reference)
C_STRING(16;$sXML_ElementReference)
C_LONGINT($LProcessID)
C_TEXT($tMirror_TimeInterval)
C_TEXT($tPreferencesPath)
C_TEXT($tSettingsFolderPath)
C_TEXT($tXML_ElementValue)

  ` Reassign for readability
$tAction:=$1

  ` Declare default values
$fAbort:=False
```

```
$tPreferencesPath:=GEN_tFormatPathname (Mirror_tMirrorPath )
```

After getting the path name from *Mirror_tMirrorPath*, the *GEN_tFormatPathname* method simply modifies the path name for the appropriate platform. (Macintosh or Windows.)

```
Case of
   : ($tAction="Create")
      CONFIRM("Choose the type for this server.";"Main DB";"Mirror")
      If (OK=1)
         <>tMirror_ServerType:="Main DB"
      Else
         <>tMirror_ServerType:="Mirror"
      End if
      <>tMirror_DatabaseName:="None"
      <>tMirror_ServerIPAddress:="0.0.0.0"
      $tMirror_TimeInterval:="00:00:00"
      <>tMirror_LastLogNumber:="[0000]"
      <>tMirror_SMTPServer:=<>tMirror_DatabaseName
      <>tMirror_ErrorEMailAccount:="someone@domain.com"
      <>tMirror_AuthenticationRequired:="No"
      <>tMirror_AuthenticationUserName:=<>tMirror_DatabaseName
      <>tMirror_AuthenticationPassword:=<>tMirror_DatabaseName

   : (Count parameters#1)
      $fAbort:=True  ` Not enough parameters
   Else
      $tMirror_TimeInterval:=String(<>hMirror_TimeInterval;HH MM SS )
End case


Case of
   : ($fAbort)
   : ($tAction="Load")
      $tSettingsFolderPath:=GEN_tGetFolderPathnames ($tPreferencesPath)
```

GEN_tFolderPathnames method separates the name of the path folders from the name of the file itself.

```
      If (Test path name($tSettingsFolderPath)#0)  ` See if the directory exists
         GEN_PreparePath ($tSettingsFolderPath)
      End if
```

The **Test path name** function will return a 0 if the value being checked is a folder path. Since we are testing for a folder, if any other value is returned, one or more of the folders do not exist. *GEN_PreparePath* is a method that will create any missing folders in the path.

Below we again check for the file using the **Test path name** function It will return a 1 if the value being checked is a file. By checking the full path, file name included, if anything other than a 1 is returned the file doesn't exist.

```
      If (Test path name($tPreferencesPath)#1)  ` Check to see if a valid file exists
```

So we call the Mirror_HandleMirrorPreferences method again to create the preference file.

```
         Mirror_HandleMirrorPreferences ("Create")  ` If not create a preferences file
```

Else

If the file exists, we simply load the preference settings.

```
   $sXML_Reference:=DOM Parse XML source($tPreferencesPath)
   $sXML_ElementReference:=DOM Find XML element($sXML_Reference;"/Mirror/
         PreferenceSetting/Mirror_ServerType")
   DOM GET XML ELEMENT VALUE($sXML_ElementReference;<>tMirror_ServerType)
   $sXML_ElementReference:=DOM Find XML element($sXML_Reference;"/Mirror/
         PreferenceSetting/Mirror_ServerName")
   DOM GET XML ELEMENT VALUE($sXML_ElementReference;<>tMirror_DatabaseName)
   $sXML_ElementReference:=DOM Find XML element($sXML_Reference;"/Mirror/
         PreferenceSetting/Mirror_ServerIPAddress")
   DOM GET XML ELEMENT VALUE($sXML_ElementReference;<>tMirror_ServerIPAddress)
   $sXML_ElementReference:=DOM Find XML element($sXML_Reference;"/Mirror/
         PreferenceSetting/Mirror_TimeInterval")
   DOM GET XML ELEMENT VALUE($sXML_ElementReference;$tMirror_TimeInterval)
   $sXML_ElementReference:=DOM Find XML element($sXML_Reference;"/Mirror/
         PreferenceSetting/Mirror_LastBackupNumber")
   DOM GET XML ELEMENT VALUE($sXML_ElementReference;<>tMirror_LastLogNumber)
   $sXML_ElementReference:=DOM Find XML element($sXML_Reference;"/Mirror/
         PreferenceSetting/Mirror_ErrorSMTPServer")
   DOM GET XML ELEMENT VALUE($sXML_ElementReference;<>tMirror_SMTPServer)
   $sXML_ElementReference:=DOM Find XML element($sXML_Reference;"/Mirror/
         PreferenceSetting/Mirror_ErrorEMail")
   DOM GET XML ELEMENT VALUE($sXML_ElementReference;<>tMirror_ErrorEMailAccount)
   $sXML_ElementReference:=DOM Find XML element($sXML_Reference;"/Mirror/
         PreferenceSetting/Mirror_ErrorEMailAuthentication")
   DOM GET XML ELEMENT VALUE($sXML_ElementReference;<>tMirror_AuthenticationRequired)
   $sXML_ElementReference:=DOM Find XML element($sXML_Reference;"/Mirror/
         PreferenceSetting/Mirror_ErrorEMailUsername")
   DOM GET XML ELEMENT VALUE($sXML_ElementReference;<>tMirror_AuthenticationUserName)
   $sXML_ElementReference:=DOM Find XML element($sXML_Reference;"/Mirror/
         PreferenceSetting/Mirror_ErrorEMailPassword")
   DOM GET XML ELEMENT VALUE($sXML_ElementReference;<>tMirror_AuthenticationPassword)
   DOM CLOSE XML($sXML_Reference)
   <>hMirror_TimeInterval:=Time($tMirror_TimeInterval)
End if

: ($tAction="Save")
   $sXML_Reference:=DOM Parse XML source($tPreferencesPath)
   $sXML_ElementReference:=DOM Find XML element($sXML_Reference;"/Mirror/
         PreferenceSetting/Mirror_ServerType")
   DOM SET XML ELEMENT VALUE($sXML_ElementReference;<>tMirror_ServerType)
   $sXML_ElementReference:=DOM Find XML element($sXML_Reference;"/Mirror/
         PreferenceSetting/Mirror_ServerName")
   DOM SET XML ELEMENT VALUE($sXML_ElementReference;<>tMirror_DatabaseName)
   $sXML_ElementReference:=DOM Find XML element($sXML_Reference;"/Mirror/
         PreferenceSetting/Mirror_ServerIPAddress")
   DOM SET XML ELEMENT VALUE($sXML_ElementReference;<>tMirror_ServerIPAddress)
   $sXML_ElementReference:=DOM Find XML element($sXML_Reference;"/Mirror/
         PreferenceSetting/Mirror_TimeInterval")
   DOM SET XML ELEMENT VALUE($sXML_ElementReference;$tMirror_TimeInterval)
   $sXML_ElementReference:=DOM Find XML element($sXML_Reference;"/Mirror/
         PreferenceSetting/Mirror_LastBackupNumber")
   DOM SET XML ELEMENT VALUE($sXML_ElementReference;<>tMirror_LastLogNumber)
   $sXML_ElementReference:=DOM Find XML element($sXML_Reference;"/Mirror/
         PreferenceSetting/Mirror_ErrorSMTPServer")
   DOM SET XML ELEMENT VALUE($sXML_ElementReference;<>tMirror_SMTPServer)
   $sXML_ElementReference:=DOM Find XML element($sXML_Reference;"/Mirror/
         PreferenceSetting/Mirror_ErrorEMail")
```

```
      DOM SET XML ELEMENT VALUE($sXML_ElementReference;<>tMirror_ErrorEMailAccount)
      $sXML_ElementReference:=DOM Find XML element($sXML_Reference;"/Mirror/
            PreferenceSetting/Mirror_ErrorEMailAuthentication")
      DOM SET XML ELEMENT VALUE($sXML_ElementReference;<>tMirror_AuthenticationRequired)
      $sXML_ElementReference:=DOM Find XML element($sXML_Reference;"/Mirror/
            PreferenceSetting/Mirror_ErrorEMailUsername")
      DOM SET XML ELEMENT VALUE($sXML_ElementReference;<>tMirror_AuthenticationUserName)
      $sXML_ElementReference:=DOM Find XML element($sXML_Reference;"/Mirror/
            PreferenceSetting/Mirror_ErrorEMailPassword")
      DOM SET XML ELEMENT VALUE($sXML_ElementReference;<>tMirror_AuthenticationPassword)
      DOM EXPORT TO FILE($sXML_Reference;$tPreferencesPath)
      DOM CLOSE XML($sXML_Reference)

   : ($tAction="SaveFromClient") | ($tAction="Save&LaunchFromClient")
      Mirror_HandleMirrorPreferences ("Save")
      Case of
         : ($tAction="Save&LaunchFromClient")
            $LProcessID:=New process("P_MirrorProcess";32000;"MirroringProcess";*)
         Else
            $LProcessID:=Process number("MirroringProcess";*)
            If ($LProcessID>0)
               DELAY PROCESS($LProcessID;0)  ` Wake up the process to update the parameters
            End if
      End case

   : ($tAction="Create")
      $sXML_Reference:=DOM Create XML Ref("Mirror")
      $tXML_ElementValue:="/Mirror/PreferenceSetting/Mirror_ServerType"
      $sXML_ElementReference:=DOM Create XML element($sXML_Reference;$tXML_ElementValue)
      DOM SET XML ELEMENT VALUE($sXML_ElementReference;<>tMirror_ServerType)
      $tXML_ElementValue:="/Mirror/PreferenceSetting/Mirror_ServerName"
      $sXML_ElementReference:=DOM Create XML element($sXML_Reference;$tXML_ElementValue)
      DOM SET XML ELEMENT VALUE($sXML_ElementReference;<>tMirror_DatabaseName)
      $tXML_ElementValue:="/Mirror/PreferenceSetting/Mirror_ServerIPAddress"
      $sXML_ElementReference:=DOM Create XML element($sXML_Reference;$tXML_ElementValue)
      DOM SET XML ELEMENT VALUE($sXML_ElementReference;<>tMirror_ServerIPAddress)
      $tXML_ElementValue:="/Mirror/PreferenceSetting/Mirror_TimeInterval"
      $sXML_ElementReference:=DOM Create XML element($sXML_Reference;$tXML_ElementValue)
      DOM SET XML ELEMENT VALUE($sXML_ElementReference;$tMirror_TimeInterval)
      $tXML_ElementValue:="/Mirror/PreferenceSetting/Mirror_LastBackupNumber"
      $sXML_ElementReference:=DOM Create XML element($sXML_Reference;$tXML_ElementValue)
      DOM SET XML ELEMENT VALUE($sXML_ElementReference;<>tMirror_LastLogNumber)
      $tXML_ElementValue:="/Mirror/PreferenceSetting/Mirror_ErrorSMTPServer"
      $sXML_ElementReference:=DOM Create XML element($sXML_Reference;$tXML_ElementValue)
      DOM SET XML ELEMENT VALUE($sXML_ElementReference;<>tMirror_SMTPServer)
      $tXML_ElementValue:="/Mirror/PreferenceSetting/Mirror_ErrorEMail"
      $sXML_ElementReference:=DOM Create XML element($sXML_Reference;$tXML_ElementValue)
      DOM SET XML ELEMENT VALUE($sXML_ElementReference;<>tMirror_ErrorEMailAccount)
      $tXML_ElementValue:="/Mirror/PreferenceSetting/Mirror_ErrorEMailAuthentication"
      $sXML_ElementReference:=DOM Create XML element($sXML_Reference;$tXML_ElementValue)
      DOM SET XML ELEMENT VALUE($sXML_ElementReference;<>tMirror_AuthenticationRequired)
      $tXML_ElementValue:="/Mirror/PreferenceSetting/Mirror_ErrorEMailUsername"
      $sXML_ElementReference:=DOM Create XML element($sXML_Reference;$tXML_ElementValue)
      DOM SET XML ELEMENT VALUE($sXML_ElementReference;<>tMirror_AuthenticationUserName)
      $tXML_ElementValue:="/Mirror/PreferenceSetting/Mirror_ErrorEMailPassword"
      $sXML_ElementReference:=DOM Create XML element($sXML_Reference;$tXML_ElementValue)
      DOM SET XML ELEMENT VALUE($sXML_ElementReference;<>tMirror_AuthenticationPassword)
      DOM EXPORT TO FILE($sXML_Reference;$tPreferencesPath)
      DOM CLOSE XML($sXML_Reference)
      <>hMirror_TimeInterval:=Time($tMirror_TimeInterval)
End case
  `End of method
```

If you are having trouble with all the XML stuff you might want to study a little bit about XML by searching the knowledgebase for technical notes on XML. This is using the DOM (Document Object Model) technique which allows you to build the structure completely in memory.

What follows is a crude analogy to some 4D concepts as to what is happening in each of the DOM commands used.

DOM Create XML Reference will create an XML reference in memory similar in concept to creating a 4D View document in memory.
DOM Create XML element will create a reference to an individual element in memory, similar to a pointer to a variable. But a hierarchical variable. Notice all the "/" in the Element Values.
DOM Parse XML Source loads the entire XML source item into memory and parses through the structure building the entire XML structure in memory.

DOM SET XML ELEMENT VALUE stores a specific value in the individual element reference. Think of this as $ptrVariable->:="MyValue"
DOM Find XML element locates the individual specified element in the XML structure loaded in memory.
DOM GET XML ELEMENT VALUE loads the value of an individual element into a 4D object. Think of this as myVariable/Field := $ptrVariable->
DOM EXPORT TO FILE writes the entire structure in the file.
DOM CLOSE XML frees the memory taken up by the XML DOM structure.

What results from all this is an XML document that looks like this:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<Mirror>

  <PreferenceSetting>
    <Mirror_ServerType>Main DB</Mirror_ServerType>
    <Mirror_ServerName>Mirror.4DB</Mirror_ServerName>
    <Mirror_ServerIPAddress>10.96.0.58</Mirror_ServerIPAddress>
    <Mirror_TimeInterval>06:00:00</Mirror_TimeInterval>
    <Mirror_LastBackupNumber>[0000]</Mirror_LastBackupNumber>
    <Mirror_ErrorSMTPServer>smtp.mydomain.com</Mirror_ErrorSMTPServer>
    <Mirror_ErrorEMail>isdept@mydomain.com</Mirror_ErrorEMail>
    <Mirror_ErrorEMailAuthentication>No</Mirror_ErrorEMailAuthentication>
    <Mirror_ErrorEMailUsername>None</Mirror_ErrorEMailUsername>
    <Mirror_ErrorEMailPassword>None</Mirror_ErrorEMailPassword>
  </PreferenceSetting>

</Mirror>
```

More on saving and modifying the XML next.

Mirror preference settings – Client side

Its been several years since any technical note dealt with controlling things on the server from a client machine. If you are familiar with how this is done, skip this portion and move on to the next.

The XML preference file resides on the Server machine in its preferences folder. It would be challenging to try to get a client machine to directly manipulate the file and modify its content. It is also awkward to have to go to a Client machine and **Execute on server** a method that would bring up a dialog on the Server machine, then run to that machine to enter the data.

Instead we will use some interprocess communication to accomplish the task by using the **GET PROCESS VARIABLE** and **SET PROCESS VARIABLE** commands. For both these commands you specify a process ID, and the name of the variable in the other process, as well as the local process variable that is to receive or currently contains the value. If the process ID is negative, you are talking about a process ID on the server, not the Client. Finally we can **Execute on server** to have the server update the file.

Everything needed takes place in the form method for the dialog that changes the preference settings

```
If (False)
    ` Form Method: Mirror_Preferences in the [zDialogs] table
    ` 4D Technote on Mirroring a 2004 database
    ` Created by: Kent Wilbur
    ` Date: 9/27/2005

    ` Purpose: Handles user interface for setting Mirroring Preferences

  <>f_Version2004x2:=True
  <>fK_Wilbur:=True

End if

  ` Declare local variables
C_LONGINT($LApplicationType)
C_LONGINT($LFormEvent)
C_LONGINT($LProcessID)
C_LONGINT($LMirrorProcessID)
C_TEXT($tTimeIncrement)

  ` Declare default
$LFormEvent:=Form event
$LApplicationType:=Application type

Case of
  : ($LFormEvent=On Load )
    ARRAY TEXT(atMirror_Authentication;2)
    atMirror_Authentication{1}:="No"
    atMirror_Authentication{2}:="Yes"

    Case of
```

```
      : ($LApplicationType=4D Client )
         GET PROCESS VARIABLE(LMirrorProcessID;<>tMirror_ServerType;<>tMirror_ServerType)
         GET PROCESS VARIABLE(LMirrorProcessID;<>tMirror_DatabaseName;
               <>tMirror_DatabaseName)
         GET PROCESS VARIABLE(LMirrorProcessID;<>tMirror_ServerIPAddress;
               <>tMirror_ServerIPAddress)
         GET PROCESS VARIABLE(LMirrorProcessID;<>hMirror_TimeInterval;<>hMirror_TimeInterval)
         GET PROCESS VARIABLE(LMirrorProcessID;<>tMirror_LastLogNumber;
               <>tMirror_LastLogNumber)
         GET PROCESS VARIABLE(LMirrorProcessID;<>tMirror_SMTPServer;<>tMirror_SMTPServer)
         GET PROCESS VARIABLE(LMirrorProcessID;<>tMirror_ErrorEMailAccount;
               <>tMirror_ErrorEMailAccount)
         GET PROCESS VARIABLE(LMirrorProcessID;<>tMirror_AuthenticationRequired;
               <>tMirror_AuthenticationRequired)
         GET PROCESS VARIABLE(LMirrorProcessID;<>tMirror_AuthenticationUserName;
               <>tMirror_AuthenticationUserName)
         GET PROCESS VARIABLE(LMirrorProcessID;<>tMirror_AuthenticationPassword;
               <>tMirror_AuthenticationPassword)

      : ($LApplicationType=4th Dimension )
         Mirror_HandleMirrorPreferences ("Load")

      Else
         CANCEL
   End case

   atMirror_Authentication:=Find in array(atMirror_Authentication;<>tMirror_AuthenticationRequired)
   If (atMirror_Authentication=2)  ` authentication required
      SET ENTERABLE(<>tMirror_AuthenticationUserName;True)
      SET ENTERABLE(<>tMirror_AuthenticationPassword;True)
   Else
      SET ENTERABLE(<>tMirror_AuthenticationUserName;False)
      SET ENTERABLE(<>tMirror_AuthenticationPassword;False)
   End if

   $tTimeIncrement:=Substring(String(<>hMirror_TimeInterval;HH MM );1;2)
   atMirror_Hr:=Find in array(atMirror_Hr;$tTimeIncrement)
   $tTimeIncrement:=Substring(String(<>hMirror_TimeInterval;HH MM );4;2)
   atMirror_Min:=Find in array(atMirror_Min;$tTimeIncrement)

` Set dialog defaults
   Mirror_ckLaunchProcess:=0
    SET VISIBLE(Mirror_ckLaunchProcess;False)
    SET VISIBLE(*;"MirrorStatus";True)
   Case of
      : (<>tMirror_ServerType="Mirror")
         tMessage:="This server is acting as the Mirrored Server"

      : (Application type#4D Client )
         tMessage:="Mirroring Can Not Run in Single User"

      : (LMirrorProcessID=-1)  ` There is no Mirroring process running on the Server
         SET VISIBLE(Mirror_ckLaunchProcess;True) )  ` Turn on the start process checkbox
         SET VISIBLE(*;"MirrorStatus";False)
         tMessage:=""
      Else
         tMessage:="Mirroring Process is Running on the Server"
   End case


: ($LFormEvent=On Unload )
   ARRAY TEXT(atMirror_Authentication;0)
```

```
      If (bOK=1)
         Case of
            : ($LApplicationType=4D Client )
               SET PROCESS VARIABLE(LMirrorProcessID;<>tMirror_DatabaseName;
                     <>tMirror_DatabaseName)
               SET PROCESS VARIABLE(LMirrorProcessID;<>tMirror_ServerIPAddress;
                     <>tMirror_ServerIPAddress)
               SET PROCESS VARIABLE(LMirrorProcessID;<>hMirror_TimeInterval;<>hMirror_TimeInterval)
               SET PROCESS VARIABLE(LMirrorProcessID;<>tMirror_SMTPServer;<>tMirror_SMTPServer)
               SET PROCESS VARIABLE(LMirrorProcessID;<>tMirror_ErrorEMailAccount;
                     <>tMirror_ErrorEMailAccount)
               SET PROCESS VARIABLE(LMirrorProcessID;<>tMirror_AuthenticationRequired;
                     <>tMirror_AuthenticationRequired)
               SET PROCESS VARIABLE(LMirrorProcessID;<>tMirror_AuthenticationUserName;
                     <>tMirror_AuthenticationUserName)
               SET PROCESS VARIABLE(LMirrorProcessID;<>tMirror_AuthenticationPassword;
                     <>tMirror_AuthenticationPassword)
               If (Mirror_ckLaunchProcess=1)
                  $LProcessID:=Execute on server("Mirror_HandleMirrorPreferences";16000;
                              "$UpdateMirrorPreferences";"Save&LaunchFromClient")
               Else
                  $LProcessID:=Execute on server("Mirror_HandleMirrorPreferences";16000;
                              "$UpdateMirrorPreferences";"SaveFromClient")
               End if

            : ($LApplicationType=4th Dimension )
               Mirror_HandleMirrorPreferences ("Save")
         End case
      End if
End case
 `End of form method
```

The Mirroring process – Main Server side

The responsibility of actually doing the mirroring falls upon the main server. Basically it is a process that sleeps until the time where it is scheduled to create a new log file and mirror the database. The method P_MirrorProcess is almost 100% involved in simply determining when the process should run. It handled 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 (False)
    ` Method: P_MirrorProcess
    ` 4D Technote on Mirroring a 2004 database
    ` Created by: Kent Wilbur
    ` Date: 9/27/2005

    ` Purpose: Begins the Mirroring Process

   <>f_Version2004x2:=True
   <>fK_Wilbur:=True

End if
```

```
  ` Declare local varaibles
C_STRING(6;$sVersion)
C_LONGINT($LDelayTicks)
C_LONGINT($LErrorCode)

C_TEXT($tClosedLogFileName)
C_TEXT($tDateTimeDelay)
C_TEXT($tLogFilePath)
C_TIME($hCurrentTime)
C_TIME($hDelayInterval)
C_TIME($hDelayUntil)
C_TIME($hTimeChange)

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

Case of
   : (Application type#4D Server )
      ALERT("Mirroring only works with 4D Server.")
   : (Num($sVersion)<802)
      ALERT("Mirroring requires version 2004.2 or higher.")
   Else
      If (<>hMirror_TimeInterval#?00:00:00?)  ` If a delay time has been defined
         $hDelayInterval:=<>hMirror_TimeInterval
         $hCurrentTime:=Current time  ` Set now for use later
         Repeat
            ` This section is necessary because other processes might call this process to resume.
                              But we don't want the process to activate too early
            If ($hDelayInterval<?00:15:00?)
               $hCurrentTime:=Current time  ` Less than 15 minutes, set to relative to time last mirroring
                                                     finished rather than  time begun
            End if
```

The next section of the code deals with midnight. If it is past mightnight the time needs to be revised.

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

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

               If (Num(Substring($tDateTimeDelay;3;2))>(Day of(Current date)))
                                                      ` Next schedule after midnight
                  $LDelayTicks:=($hDelayUntil+?24:00:00?)-$hCurrentTime*60
               Else
                  $LDelayTicks:=$hDelayUntil-$hCurrentTime*60
               End if
               DELAY PROCESS(Current process;$LDelayTicks)

                ` It is possible that a Client machine could be changing the delay interval
                ` If this is true the time needs to be recalculated
               Case of
                  : ($hDelayInterval=<>hMirror_TimeInterval)  ` No changes necessary simply stay delaying

                  : ($hDelayInterval><>hMirror_TimeInterval)  ` Mirroring interval is being shortened
```

```
            $hTimeChange:=($hDelayInterval-<>hMirror_TimeInterval)
            $hDelayUntil:=Time(Substring($tDateTimeDelay;9;2)+":"+
                  Substring($tDateTimeDelay;11;2)+":"+Substring($tDateTimeDelay;13;2))
            If (Num(Substring($tDateTimeDelay;3;2))>(Day of(Current date)))  ` The scheduled time
                                                                                is tommorw
               $hDelayUntil:=$hDelayUntil+?24:00:00?
            End if

            Case of
               : (($hCurrentTime+$hTimeChange)<$hDelayUntil)  ` The new time is before the
                                                              scheduled time, simply reset the time
                  $hDelayUntil:=$hDelayUntil-$hTimeChange
                  If ($hDelayUntil>?24:00:00?)
                     $tDateTimeDelay:=GEN_tDateTimeStamp (Add to date(
                           Current date;0;0;1);$hDelayUntil-?24:00:00?)
                  Else
                     $tDateTimeDelay:=GEN_tDateTimeStamp (Current date;$hDelayUntil)
                  End if
               : (($hCurrentTime+$hTimeChange)>$hDelayUntil)
                     ` The old time is before the new time setting, do the old time as scheduled
            End case
            $hDelayInterval:=<>hMirror_TimeInterval  ` Reset to the current value

         Else   ` Mirroring interval is being lengthened
            $hDelayUntil:=Time(Substring($tDateTimeDelay;9;2)+":"+
                  Substring($tDateTimeDelay;11;2)+":"+Substring($tDateTimeDelay;13;2))
            If (Num(Substring($tDateTimeDelay;3;2))>(Day of(Current date)))  ` The scheduled time
                                                                                is tommorw
               $hDelayUntil:=$hDelayUntil+?24:00:00?
            End if
            $hDelayUntil:=$hDelayUntil+(<>hMirror_TimeInterval-$hDelayInterval)
            If ($hDelayUntil>?24:00:00?)
               $tDateTimeDelay:=GEN_tDateTimeStamp (Add to date(Current date;0;0;1);
                     $hDelayUntil-?24:00:00?)
            Else
               $tDateTimeDelay:=GEN_tDateTimeStamp (Current date;$hDelayUntil)
            End if

            $hDelayInterval:=<>hMirror_TimeInterval  ` Reset to the current value
      End case

   Until ((GEN_tDateTimeStamp (Current date;Current time)>=
         $tDateTimeDelay) | (<>hMirror_TimeInterval=?00:00:00?))

   If (<>hMirror_TimeInterval#?00:00:00?)  ` If we are not  turning off the mirroring
      $hCurrentTime:=Current time  ` Reset for next time interval
      $LErrorCode:=Mirror_LSendLogFile
      Case of
         : ($LErrorCode=1)  ` Mirroring Sucessful
         : ($LErrorCode=1411)  ` Mirroring did not occur because of critical operation try again next
                                                                  scheduled time
         : ($LErrorCode=-17050)  ` Mirroring database not available mirroring did not take place try
                                                                  again next scheduled time
         : ($LErrorCode=1403)  ` Fatal error no log file, stop mirroring
            <>hMirror_TimeInterval:=?00:00:00?  `stop mirroring without changing saved settings
         : ($LErrorCode=1410)  ` Fatal error log file out of sync, stop mirroring
            <>hMirror_TimeInterval:=?00:00:00?  `stop mirroring without changing saved settings
         : ($LErrorCode=1412)  ` Fatal error not running on Server
            <>hMirror_TimeInterval:=?00:00:00?  `stop mirroring without changing saved settings
         Else
            ` There might be other errors, leave other errors running
```

```
            End case
         End if

      Until ((<>fQuit) | (<>hMirror_TimeInterval=?00:00:00?))
   End if

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.

The Mirror_LSendLogFile method does most of the work. First it verifies that the other mirror is present. If the other mirror is not currently available, a new log file should not be created.

```
If (False)
    ` Method: Mirror_LSendLogFile
    ` 4D Technote on Mirroring a 2004 database
    ` Created by: Kent Wilbur
    ` Date: 9/29/2005

    ` Purpose: Handles segmenting and sending the log file to the mirror

  <>f_Version2004x2:=True
  <>fK_Wilbur:=True

End if

  ` Declare parameters
C_LONGINT($0)

  ` Declare local variables
C_LONGINT($LAbortCounter)
C_LONGINT($LError)
C_LONGINT($LLastSublogNumber)
C_LONGINT($LPosition)
C_LONGINT($LSMTPid)
C_TEXT($tLastLogFile)
C_TEXT($tLogFilePath)
C_TEXT($tIntegrateType)

Mirror_LError:=0
$LError:=0
$LAbortCounter:=0
$tLastLogFile:=""

If (Not(<>fQuit))
   Case of
      : (proxy_SOAP_LMirrorHandleEvents (<>tMirror_DatabaseName;"VerifyPresent";
                  <>tMirror_ServerIPAddress)#1)  ` Correct database is not available
         $LError:=-17050
         SOAP_tErrorMessage:="Mirror database is not available. Mirroring at "+String(Current time
                  ;HH MM )+" on "+String(Current date;Short )+" did not take place."

      Else
         ON ERR CALL("ERROR_HandleMirrorError")
         $tLastLogFile:=New log file  ` Get the full path name of the log file that has just been closed
         ON ERR CALL("")
```

```
   End case
```

Errors are trapped in the *ERROR_HandleMirrorError* method which simply assigns the Error variable to the Mirror_LError variable for handling and better understanding.

```
Case of
   : (Mirror_LError=1409) | (Mirror_LError=1411)  ` Transaction or other critical operation in progress
      $LAbortCounter:=$LAbortCounter+1
      If ($LAbortCounter>10)
         $LError:=Mirror_LError
         SOAP_tErrorMessage:="Transaction or other critical operation blocking mirroring. Error Code:
                  "+String(Mirror_LError)
      End if
```

If the VerifyPresent routine passes, then a new log file is created. If the creation of the new log file is successful, the just closed 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 proxy_SOAPHandleEvents method, but it was automatically generated by the Web Services Wizard and not modified significantly other than to use the preferences settings for the location of the mirror server.

```
   : (Length($tLastLogFile)>0)
      $tIntegrateType:="IntegrateLog"
      $LAbortCounter:=0

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

      $tLogFilePath:=GEN_tGetFolderPathnames ($tLastLogFile)
      If ($tLogFilePath#$tLastLogFile)  ` Strip off the full path name if present
         $tLastLogFile:=(Substring($tLastLogFile;Length($tLogFilePath)+1))
      End if
      $LError:=proxy_SOAP_LMirrorHandleEvents (<>tMirror_DatabaseName;$tIntegrateType;
               <>tMirror_ServerIPAddress;->SOAP_oMirrorBLOB;$tLastLogFile)
                                                      ` Send the log file to the server
      If ($LError=1)
         <>tMirror_LastLogNumber:=$tLastLogFile
         Mirror_HandleMirrorPreferences ("Save")  ` This saves the name of the last log number
      End if
      SET BLOB SIZE(SOAP_oMirrorBLOB;0)  ` Clean up memory

   Else
      $LError:=Mirror_LError
End case

End if

$0:=$LError
```

Finally, if there are errors, error messages are sent via e-mail to the appropriate party.

```
Case of
   : ($LError=1)  ` Mirroring Sucessful nothing else to do
   : ($LError=0)  ` Mirroring did not occur, but nothing to do now
   : (Length(<>tMirror_SMTPServer)=0) & (<>tMirror_SMTPServer#"None")  ` Nothing to do no place to
                                                                          send error message
   : (Position(".";<>tMirror_SMTPServer)<1) | ((Position(".";<>tMirror_SMTPServer))=
                     (Length(<>tMirror_SMTPServer)))  `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

      $LError:=SMTP_New ($LSMTPid)
      $LError:=SMTP_Host ($LSMTPid;<>tMirror_SMTPServer)
      $LError:=SMTP_From ($LSMTPid;<>tMirror_ErrorEMailAccount)
      $LError:=SMTP_Subject ($LSMTPid;"Error with Mirroring for "+<>tMirror_DatabaseName)
      $LError:=SMTP_To ($LSMTPid;<>tMirror_ErrorEMailAccount)
      $LError:=SMTP_Body ($LSMTPid;SOAP_tErrorMessage)
      If (<>tMirror_AuthenticationRequired="Yes")
         $LError:=SMTP_Auth ($LSMTPid;<>tMirror_AuthenticationUserName;
                  <>tMirror_AuthenticationPassword)
      End if
      $LError:=SMTP_Send ($LSMTPid)
      $LError:=SMTP_Clear ($LSMTPid)

End case
SOAP_tErrorMessage:=""
  `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 *SOAP_MirrorHandleEvents* the mirrored server goes to work. It returns a 1 for the action succeeded or an error code if the action failed.

It has two actions. First verify that the mirrored database is present and correctly named. The second is to integrated the log.

```
If (False)
    ` Method: SOAP_MirrorHandleEvents
    ` 4D Technote on Mirroring a 2004 database
    ` Created by: Kent Wilbur
    ` Date: 9/29/2005

    ` Purpose: Handles SOAP requests for the Mirroring process

   <>f_Version2004x2:=True
```

```
   <>fK_Wilbur:=True

End if

  ` Declare variables for SOAP Types
C_BLOB(SOAP_oMirrorBLOB)
C_LONGINT(SOAP_LMirrorResult)
C_TEXT(SOAP_tMirrorAction)
C_TEXT(SOAP_tMirrorDatabaseName)
C_TEXT(SOAP_tMirrorLogFile)

  ` Declare local variables
C_LONGINT($LCompressed)
C_LONGINT($LErrorCode)
C_TEXT($tDatabasePath)
C_TEXT($tDatabaseName)

SOAP DECLARATION(SOAP_tMirrorDatabaseName;Is Text ;SOAP Input ;"SOAP_tMirrorDatabaseName")
SOAP DECLARATION(SOAP_tMirrorAction;Is Text ;SOAP Input ;"SOAP_tMirrorAction")
SOAP DECLARATION(SOAP_oMirrorBLOB;Is BLOB ;SOAP Input ;"SOAP_oMirrorBLOB")
SOAP DECLARATION(SOAP_tMirrorLogFile;Is Text ;SOAP Input ;"SOAP_tMirrorLogFile")
SOAP DECLARATION(SOAP_LMirrorResult;Is LongInt ;SOAP Output ;"SOAP_LMirrorResult")

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

$LErrorCode:=0
Mirror_LError:=0

$tDatabaseName:=Structure file
$tDatabasePath:=GEN_tGetFolderPathnames ($tDatabaseName)
$tDatabaseName:=(Substring($tDatabaseName;Length($tDatabasePath)+1))
If (SOAP_tMirrorDatabaseName=$tDatabaseName)
   Case of
      : (SOAP_tMirrorAction="VerifyPresent")
         SOAP_LMirrorResult:=1  ` The database is here

      : (SOAP_tMirrorAction="IntegrateLog@")
         READ WRITE(*)  ` Set tables to read write for mirror integration
         BLOB PROPERTIES(SOAP_oMirrorBLOB;$LCompressed)
         If ($LCompressed#Is not compressed )
            EXPAND BLOB(SOAP_oMirrorBLOB)
         End if
         ON ERR CALL("ERROR_HandleMirrorError")
         BLOB TO DOCUMENT($tDatabasePath+SOAP_tMirrorLogFile;SOAP_oMirrorBLOB)
         INTEGRATE LOG FILE($tDatabasePath+SOAP_tMirrorLogFile)
         ON ERR CALL("")
         If (OK=1)
            SOAP_LMirrorResult:=OK
         Else
            $LErrorCode:=Mirror_LError
         End if

   End case

Else
   $LErrorCode:=-17051
End if

If ($LErrorCode#0)
   SOAP_LMirrorResult:=$LErrorCode
   SOAP_ErrorHandling ($LErrorCode)
End if
```

```
  `End of method
```

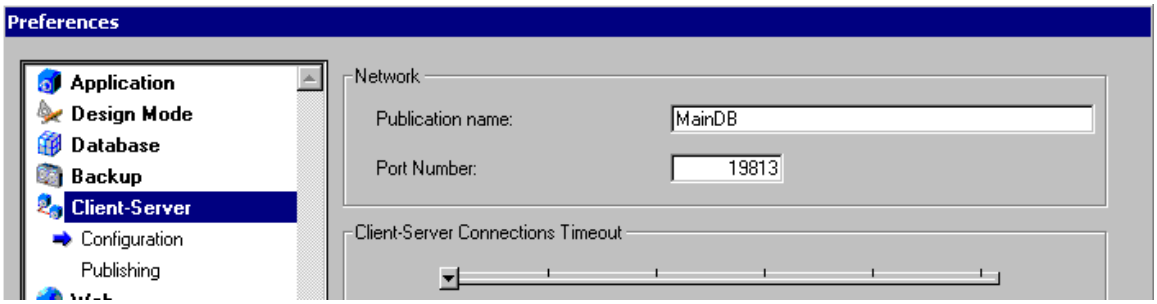
If an error occurred during integration or another error transpired, the error is interpreted into a error number and message and a SOAP fault is returned to the main database.

```
If (False)
      ` Method: SOAP_ErrorHandling
      ` 4D Technote on Mirroring a 2004 database
      ` Created by: Kent Wilbur
      ` Date: 9/29/2005

      ` Purpose: Key containing description of the errors

    <>f_Version2004x2:=True
    <>fK_Wilbur:=True

End if

  `Declare parameters
C_LONGINT($1;$LErrorCode)

  ` Reassign for readability
$LErrorCode:=$1

Case of
    : ($LErrorCode=-1403)
       SOAP_tErrorMessage:="No log file. The mirror database is not configured to have a log file. "+
                  "Integration can not take place."
    : ($LErrorCode=-1410)
       SOAP_tErrorMessage:="The log files are out of sync. Integration can not take place."+" You will need
                  to rebuild the mirror server from scratch."
    : ($LErrorCode=-1412)
       SOAP_tErrorMessage:="The mirror database is not running 4D Server.  Integration can only occur
                  when us"+"ing 4D Server."
    : ($LErrorCode=-17051)
       SOAP_tErrorMessage:="Invalid Database Name. Can not merge data into a database with this name.
                  "+"Check mirror preference settings."
End case

SEND SOAP FAULT(SOAP Client Fault ;String($LErrorCode)+" - "+SOAP_tErrorMessage)
  `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.

First, at the very least rename the Network Publication name for the mirrored server. This is done under the Preferences for Client-Server.



If the database is named something different at least it is unlikely that a user will accidentally log into the database.

A more secure solution is to change the port number for 4D Client. Then the normal 4D Client application can't accidentally log into the wrong database. (Don't forget to restore the Port number if you need to set up the mirror to act as the main database.

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 you must go back to the original backup and begin from there. However, the original backup does not contain any of the log file segments. It doesn't matter which database is used. Follow the steps below to recover from the major disaster.

1) Restore the database from the last backup.
2) Move all the Log files 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: E_RestoreDataFromMirrorLogs
    ` 4D Technote on Mirroring a 2004 database
    ` Created by: Kent Wilbur
    ` Date: 11/1/2005

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

  <>f_Version2004x2:=True
  <>fK_Wilbur:=True

End if
```

```
  ` Declare local variables
ARRAY TEXT($atDocuments;0)
ARRAY TEXT($atLogFiles;0)
C_BLOB($oBlob)
C_LONGINT($i)
C_LONGINT($LApplicationType)
C_LONGINT($LError)
C_LONGINT($LPid)
C_LONGINT($LPosition)
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:=GEN_tGetFolderPathnames (Document)
         $tFileName:=Substring(Document;Length($tFolderPath)+1)  ` Get the name of the log file chosen

           ` Get necessary information from the server
         SOAP_tMirrorDatabaseName:=""
         tMirror_ThisServerIPAddress:=""
```

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**.

```
         $LPid:=Execute on server("E_RestoreDataFromMirrorLogs";32000)
         Repeat
            PROCESS_MyDelay (Current process;2)  ` Wait for the server to get the information
            GET PROCESS VARIABLE($LPid;SOAP_tMirrorDatabaseName;SOAP_tMirrorDatabaseName)
            GET PROCESS VARIABLE($LPid;tMirror_ThisServerIPAddress;tMirror_ThisServerIPAddress)
         Until (Length(SOAP_tMirrorDatabaseName)>0)
```

Log segments 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
         $LError:=proxy_SOAP_LMirrorHandleEvents (SOAP_tMirrorDatabaseName;"IntegrateLog";
                     tMirror_ThisServerIPAddress;->Mirror_oLogFile;$atLogFiles{$i})
                                                   ` Send the log file to the server
         If ($LError#1)
            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
      $LError:=proxy_SOAP_LMirrorHandleEvents (SOAP_tMirrorDatabaseName;"IntegrateLog";
               tMirror_ThisServerIPAddress;->Mirror_oLogFile;$tFileName)
                                  ` Send the log file to the server

      If ($LError#1)
         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.

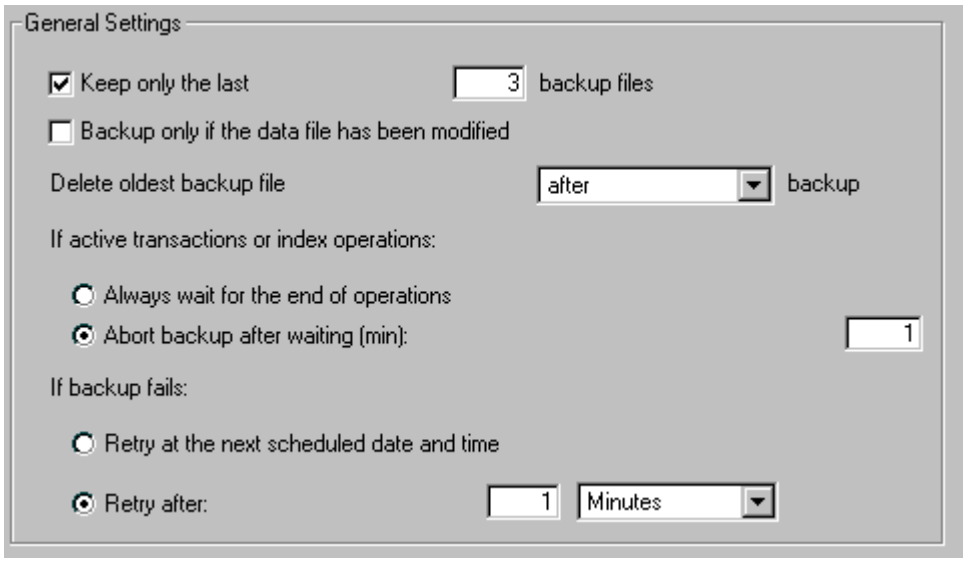
```
   : ($LApplicationType=4D Server )
      SOAP_tMirrorDatabaseName:=""
      $LError:=IT_MyTCPAddr (tMirror_ThisServerIPAddress;$tFileName)
      $tDatabaseName:=Structure file
      $tDatabasePath:=GEN_tGetFolderPathnames ($tDatabaseName)
      SOAP_tMirrorDatabaseName:=(Substring($tDatabaseName;Length($tDatabasePath)+1))
      PROCESS_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
```

New log file function and critical operations

The **New log file** function and certain critical operations such as transactions or indexing are mutually exclusive. For example you can not create a **New log file** while a client machine or stored procedure has an open transaction, nor can a new transaction be started while the **New log file** function is taking place.

If your database uses transactions, you will want to schedule your mirroring routine to run when there are no users are active in the database. If transactions are in progress during the New log file it is possible that the server might hang for a few minutes waiting for the transaction. You can minimize the wait by changing some default settings in the backup properties of the database.

In the General Settings be sure that the Abort backup after waiting (min.) radio button is selected. Also change the default time from 3 minutes to 1 minute.



DO NOT change the setting to 0 minutes. This has the effect of making the backup wait for the transaction to complete and never timeout. If necessary you can code for the success or failure of the New log file function by looking at the error code. If the error code is 1411, the New log file failed because of a critical operation. Usually a transaction was in progress. Your scheduler could try again until successful, or the beginning of the next business day. Sorry, code for this is not in the example database.

Summary

Creating a mirrored system is no longer a database administrator task. It requires the cooperation and implementation of 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 2004.3 which fixed a minor bug that this example code would not have even hit.

Most 4D databases don't need mirroring. But for those that do, I suggest you take a look at what can be done with the two new language commands.