Managing Multiple Record Entries
Part II

Lincoln Stoller, Ph.D.
Braided Matrix, Inc.

Contents of Part I

Contents of Part II

III. Introduction to Part II

In part I of this article I described four techniques of managing read/write access to several records simultaneously. These were global transactions, hierarchical access, setting flags, and multiple record locking. In this final part of the article I compare these methods.

IV. Pro's and Con's

These methods offer different degrees of control and require different sacrifices in performance. The structure of the database and the needs of the user should determine which method is appropriate in a given circumstance. The main characteristics of these methods will be reviewed on the criteria of simplicity, speed, robustness and impact on the database.

A. Simplicity

1.Global transactions

Global transactions require little more programming than starting and finishing a transaction. The machines of other database users are frozen until the transaction is complete. If the local user finds that some of the records to which they need write access are locked, the transaction can be canceled.

2. Hierarchical access

If the structure is hierarchical, most of this access management scheme is already in place; the programmer only needs to monitor the status of the parent record.

4D can be made to handle record access by allowing related-many input only through included layouts in the parents input screen. When entering a layout for a locked parent record, both the parent's fields and the included layout will be non-enterable.

3. Semaphores

Semaphores are the simplest of flagging methods (there are only 2 commands for semaphores) and they require no special database structure. In version 2 of 4th Dimensions (4D v2) there is only one kind of semaphore, while in 4D v3 there are local semaphores (those that start with a "$" and exist only for processes local to a given user) and global semaphores (those that exist for all processes.)

When the SEMAPHORE command returns false it has executed two actions: checking for a semaphore and setting a semaphore. This is noteworthy in situations where there may be contention among processes for record access. The point is that once the Semaphore command has determined that a semaphore is not set it has the exclusive ability to set the semaphore before anyone else.

4. Flag records

Using flag records requires database files especially for this purpose and additional code to set and remove the flags. However if the flags are used generically flag management can be encapsulated in a single procedure as shown below in Figure 5.

```
If (SetFlagSuccessful("Updating_AFile_sequence_num"))
   .
```

```
    .  { do tasks }
    .
    UNLOAD RECORD([FlagFile])`Flag is now unlocked.
End if
```

Figure 5.

The *SetFlagSuccessful* procedure loads a flag record and returns "true" if the record is unlocked

```
`Procedure SetFlagSuccessful                $1=flag file record name,
`   to set the flag                         $0=record status: True if unlocked.
    SEARCH( [FlagFile]; [FlagFile]Marker = $1)
    If (Records in selection( [FlagFile]) = 0)
        CREATE RECORD( [FlagFile])          `Create record if none exists.
        [FlagFile]Marker := $1
        SAVE RECORD( [FlagFile])
        $0 :=True
    Else
        READ WRITE( [FlagFile])
        LOAD RECORD( [FlagFile])
        If (Not(Locked( [FlagFile])))
            $0 :=True
        Else
            $0 :=False
        End if
    End if
    READ ONLY( [FlagFile])                  `Leave record loaded
                                            `until assignment complete.
```

Figure 6.

5. Pushing and popping locked records

Using the stack to preserve a record's locked status requires addition of:

- Loops to test and to push the records,
- Special input or dialog layouts for the user to enter modifications,
- Arrays to store temporary field changes, and
- Loops to pop, assign field values and save the records.

Stack management is less amenable to encapsulation since each file's record structure, and hence temporary storage requirements, are likely to differ.

Dealing with several records and a large number of fields requires putting greater effort into writing code to handle the many storage arrays. The cost of using this method is more code, more complex code and code that's harder to modify; the payoff is faster code execution and more efficient disk access.

6. Saving records during a multi-transaction

This method requires little more code than would be in place for a singe user application. Once the multi-transaction is started, write status is verified, and the needed records are locked, execution

can proceed as if there were no other users on the network. Regular input layouts and direct field assignments can be used to effect changes.

B. Speed

While record locking is not time intensive in itself, these methods involve varying amounts of supporting code and disk access.

The following times were clocked over a LocalTalk network. System 7 file sharing was used for the 4D v2 tests. The server was a Quadra 700 with 4D running version 2.2.3 or 4D Server in a 4.5 meg partition; the client was a Mac SE/30 running version 2.2.3 and a local structure file or 4D Client in a 3.5 meg partition. Compiled versions of the structure file were used, no other applications were running and the only INITs loaded on either machine were those for file sharing.

The file used for searching had 100 records with 4 fields one of which was indexed and the searches that were done located all 100 records. The time taken to find a single record is defined to be 1/100 of the time to locate the selection of 100 records.

The records in the flag file had one indexed alpha field and the file contained a single record. There was only one semaphore in the external flags file used by version 2. Each result represents the average of several hundred trials after the longer times of the initial trials were discarded.

1. Global transactions

Multi-user access was 3 to 10 times faster during a global transaction in the version 2 database and took the same amount of time running version 3 on 4D Server.

SE/30 client:	**4D v2**			**4D v3**		
number of seconds to:	normal	global transact.	multi-transact.	normal	global transact.	multi-transact.
search for a record	.006	.001	.01	.002	.002	.002
search for, modify and save a record	.33	.10	0.65	.15	.15	.15
load and unload a record	.15	.04	0.24	.09	.09	.09

2. Setting a semaphore

Judging from the performance shown below, the time required to set a semaphore is 25% less than that required to query the database in 4D v2 and 50% less in 4D v3. Semaphores are the fastest way of setting a global flag.

3. Locking a flag's file record

The quickest flagging records scheme has a user loading an unlocked flag record before beginning a critical process and unloading the record when finished (as shown in the sample code in §IV.A.4). This takes between 30% to 100% longer than setting a semaphore as is shown below.

By maintaining a flags file with as few records as possible, and whose key fields are small and indexed, the time required to search the file of flag records will remain approximately constant as the database grows.

4. Creating or modifying a flag's file record

In this strategy the flag record file is queried with a SEARCH command and if no record is found with the given name a flag record is created and saved. An alternative strategy would be to SEARCH for a preexisting flag record, modify it and save the changes.

A record used in this manner can convey more information than a semaphore or a record used as a semaphore. However the need to search for and then save a record increases the time required to complete the operation.

SE/30 client:	**4D v2**	**4D v3**
number of seconds to:		
set a local semaphore	–	.001
set a global semaphore	.25	.08
search for an existing flag file record	.33	.19
search for, modify and save an existing flag file record	.33	.25

5. Locking a set of records

Both record locking methods, using the stack and saving during a multi-transaction, take comparable amounts of time to load and lock-up the selection. Starting a multi-transaction takes a negligible amount of time but disk access and related overhead cause record management during a multi-transaction to take 60% to 100% longer in 4D v2. On the other hand, multi-transactions actually speed things up in 4D v3, but rarely by more than 1%.

There is, however, a significant difference between locking records with multi-transactions as opposed to using the stack in the amount of data saved to disk when modifying an entry. If you're creating a new multi-record entry then both methods will save the same amount of data as all records and their effects on other files will have to be recorded. But if you're modifying a preexisting entry this may not be so. If the number of affected records is small while the number of associated records is large then the stacking method, that allows for resaving only the affected few, will execute much faster. Whether the stack method is worth the time and trouble will be determined by the nature of the task.

C. Robustness

The degree to which the database remains intact and operational after the machine processing an entry crashes off the network is a measure of database robustness. As a general rule, any multi-record locking method becomes more robust if you insure that there is no user input while data is being saved to the server. User input should be complete before you begin saving interrelated records and the

CPU should not be executing the instructions of another program. In other words the server should be devoted to the database application.

Methods are also more robust for 4D v2 when access is keyed to record status. AppleTalk updates the server's list of active nodes approximately every two seconds and when a user assigned a locked record disappears from the network that record is quickly unlocked. 4D v3 handles all multiuser activity internally and is aware of each user and each process. Because of this all of these methods appear to be equally robust in version 3.

The following observations are based on the behavior of a sample multi-user database after the remote node was crashed by removing the AppleTalk connector. My comments only apply to the crash of a node before it has unlocked its records or deactivated its flags; I have not tested what happens to the database if the remote node crashes while it's in the process of sending information to the server.

1. Global transaction

Running 4D v2, crashing the remote user off the network during a global transaction locked other users out of the database for about 90 seconds. After that all records were unlocked and the database operated normally. The flags file still needed to be discarded in order to reset the counter that assigns user numbers.

Running 4D v3 the crashed user was logged out of the database after a little over 2 minutes at which point all their locked records were unlocked and their semaphores cleared.

2. Setting Semaphores

In 4D v2 semaphores, like data, remain until they are cleared. The semaphores set by a crashed user remain in the flags file. They can be cleared by another user if they know the name of the semaphores set in error.

Retrieving the database from the limbo caused by a dangling semaphore can be accomplished by discarding the flags file. Assuming the database administrator has traced the database problem to a dangling semaphore they will have to log all users off the database in order to throw away this file.

In 4D v3 the semaphores are linked to the processes that created them and are discarded when the processed ends. When a client crashes off 4D Server the semaphores that they set are released in about 2 minutes.

3. Locking a flag's file record

Loading an unlocked flag record, like loading any unlocked record, activates byte range locking. When the remote node crashed before its write accessible records were unloaded, the records were unlocked within two seconds in version 2 and two minutes in version 3.

4. Creating or modifying a flag's file record

Creating special data flags suffers some of the drawback as setting semaphores in version 2, namely the problem of dangling flags. In this case the problem can be avoided if your messaging system requires both the existence of a flag record and that the flag record be locked. This insures that even though a crashed user's flags will remain in the data file, the record will be unlocked.

For example, if a user wants to modify the Jones record in the customer file they first search the file of flags for a "Jones_Modification" record. If the record is found and is locked then the user

knows to wait or cancel the entry. But if the record is found and is unlocked the user simply loads it, modifies the Jones record and then deletes (or possibly unloads) the flag record.

5. Pushing unlocked records onto the stack

When the stacking user crashed with write accessible records on their stack all these records were unlocked. The data file was unaffected because the stack was only in local memory.

6. Saving records during a multi-transaction

When a multi-transaction was started, records saved and then the node crashed off the network, these records remained locked for about 90 seconds in version 2 and two minutes in version 3.

D. Database impact

How can each of these methods be tailored so as to have minimal impact on other network users?

1. Global transaction

There is no flexibility in how a global transaction effects other users: once a user starts a global transaction all other database users are locked out of the database, their machine's frozen until the transaction is complete.

The initiator of the transaction has write access only to unlocked records; it isn't necessary to set multiple record locks since no other user has access to the database. However it is still necessary to check that needed records are unlocked.

If, for any reason, unlocked records are pushed onto the stack then these records will remain locked after the transaction is either validated or canceled. This is in contrast to multi-transactions which unlock all records when canceled or validated in 4D version 2 or 3.

If a series of SAVE RECORD commands are issued then all records will be unlocked if and when the transaction is canceled.

2. Hierarchical access

In theory hierarchical access schemes have a narrow focus affecting only the records involved in the multi-record entry. Unfortunately many structures are not purely hierarchical and even when they are, procedures may be required that violate the hierarchy. The impact of the method is then determined by the auxiliary record accessing schemes.

3. Setting Semaphores

A given semaphore carries a single bit of information and as such can't be very discriminating. Semaphores are suited to control generic, often repeated, procedures like the assigning and updating of sequence numbers.

If a simple flag is used to manage access to a lengthy and frequently used procedure there is the danger of creating a bottle neck. If performance is seriously degraded because of this it may be possible to divide the procedure into several parts and use a series of different semaphores.

For example, if you need to assign record I.D. numbers to four different files you could either set a single semaphore and process all assignments, or you could set a semaphore associated with the

first file, assign its ID, clear this semaphore and move on in sequence with different semaphores for the remaining files. This "pipelining" of access could significantly decrease the time spent processing backlogged users.

Semaphores retain an advantage over the creation of flag file records in that the SEMAPHORE command both checks and sets them in a single command. This assures that another user can not "steal" the semaphore after it has been found unset and before the user can set it.

4. Locking a flag's file record

Sending messages between nodes by testing the status of flagging records serves the same function as setting semaphores; both will have the same impact on the database. Choosing to use flagging records over semaphores gains security in exchange for speed.

5. Creating or modifying a flag's file record

Creating unique rather than generic flag records broadcasts more specific messages. Instead of setting a "Customer Modification" flag one could set a "Jones Modification" flag thereby interfering little with other users ability to modify customers. If many users need rapid access to the same process, the transaction processing is especially lengthy, or the transaction will be interrupted by user input then it may be advantageous to use highly specific flags.

It is worth noting that by using distinctive flag records any level of control can be reached: flags could be set to protect specific fields within individual records.

6. Stacking and Saving records during a multi-transaction

Techniques that test the status of the individual records offer a specific and low impact means of controlling network access. Because record status is managed automatically and is an integral part of the code, these methods may be more self-explanatory than using flags and as such may be easier to maintain.

IV. Overview

The following chart summarizes how each of the seven methods I've discussed rates in the major areas of simplicity, speed, robustness and impact.

Chart of multi-record access management techniques

	Simplicity of code	Speed of execution	Robustness with network	Impact on database
Global Transaction	trivial	fastest	strong	highest
Enforced Hierarchical Access	simple to involved	fast	strong	medium to low

Setting Semaphores	simple	fast	weak (4D v2) strong (4D v3)	medium
Locking Flag Records	simple	moderate	strong	medium
Creating Flag Records	simple to involved	slow	weak	medium to lowest
Stacking Unlocked Records	complex	moderate	strong	lowest
Saving During Multi-transactions	simple to involved	moderate to slow	strong	low