

One of the best design features of 4D is the division of the data file from the structure file. Crudely put, user work goes in the data file and developer stuff goes in the structure file. Contrast this to a system in which the programming resources are located in the same file as the data. When a developer is working on a system that already has user data in it he must perform extra tasks if the data is to be preserved or take the system down while it is reprogrammed and tested.

With a system such as 4D where the developer structures are kept in a different file from the data records, new work can be brought into the system in as little time it takes to reboot the server and swap files in a directory.

This works very well with 4D systems as long as one respects the proper location for information. This article will detail one situation where 4D systems can actually store user data in the structure file and by so doing, breach the integrity of one's own version upgrade. Most clients I have worked with expect their data to be intact after visits to their office. I hope to share with you a few techniques I have developed through experience.

The odd situation with user data being stored in the structure file can actually occur through the use of another great feature in 4D, Lists.

Lists are wonderful things. I like to think of them as a **STRING ARRAY** stored in the structure file. These lists can be used to populate arrays in the database, be used as choice lists or excluded

values for a field, or set the default values of an object upon creating a new record. Suffice to say there are many uses for lists in a database. One of the properties of a list is that users can add, modify, or delete items in the list if you as developer allow this to occur. Users can even sort the values in a list without making you do any additional programming.

But this is where the challenge lies. Consider the following: You maintain a vertical market application with 100 site licenses. There is one list in the database that you have set to user modifiable. Each site has made its own insertions, edits, and deletions to the default list you provide with an off-the-shelf copy of your application. Indeed, you have no idea what values are contained within the distributed 100 instances of your cherished structure file. Let us further suppose that the last update to your software was over a year ago and a significant amount of customization has been done by thousands of users across your various customer sites.

Given a situation like that you could not ethically distribute a new structure file that would simply replace what amounts to your customers work with what you deem appropriate as the “default”. In any case it would simply not be wise. You could ask the clients to make manual copies of their lists and then re-enter the lists after they install your new programming structure. But I would wager you would be swamped with technical support phone calls for many weeks.

One possible approach would be to write the lists to external files that would contain the users data. At startup time the database could read these files and reload any data contained within. There are several other issues that this method brings up, however. What do you do if the file is not in the right location during startup? What if the user(s) edit the file at the wrong time and do not maintain the format that the database expects to find?

I have implemented a much simpler solution. I store any user modifiable lists in the data file upon shutdown. Provided that the system is shutdown in a graceful way, the proper methods will execute and the data will be saved. Likewise, upon startup the database can load the users' lists from certain records in the data file.

Imagine one of the sites in the above vertical market application scenario. The client downloads a new copy of my structure file from my web site. They shutdown their server, at which time all of the user modifiable lists are stored in the data file. The site administrator swaps the old structure file with the new one and restarts the database application. The database upon startup gathers portions of the data file and uses the selected records to reconstruct the user's modified list exactly the way it was when the database was shut down.

With this system in place you won't receive any technical support inquiries, at least not on that issue.

The first step is defining the data structure to store the user's lists. As shown in figure 1 a simple table with three fields form the entire storage structure. The list name, the list element number, and the element itself contain all the information that you will need.

Illustrated below is the method that I use to save the user lists to records:

```
` srv_saveUserLists
` this saves a lists that I have allowed the users to modify
` at the time of server shutdown
` it saves the lists in the [userdataLists] table
` this will be called by "On server Shutdown"
` or by "On Exit" depending on what what type of 4D this is
` written by Dan LaSota February 2001, updated June 2001

` begin by clearing all the old records
READ WRITE([UserDataLists])
ALL RECORDS([UserDataLists])
DELETE SELECTION([UserDataLists])

C_LONGINT($arrayLoop)
C_STRING(30;$listName)
ARRAY STRING(80;$aTemp;0)

` this is the actual conversion of lists to records
$listName:="agreement_type"
LIST TO ARRAY($listName;$aTemp)
For ($arrayLoop;1;Size of array($aTemp))
  CREATE RECORD([UserDataLists])
  [UserDataLists]listName:=$listName
  [UserDataLists]value:=$aTemp{$arrayLoop}
  [UserDataLists]position:=$arrayLoop
  SAVE RECORD([UserDataLists])
```

```
UNLOAD RECORD([UserDataLists])
End for
```

If there are additional lists to save then you would want to duplicate the last block of code and change the variable \$listname to reflect which list you want to save. The reciprocal method, which loads the user data during startup, is listed below:

```
` srv_loadUsersLists
` this, on startup or on server startup, will load the records from UserListData
` written by Dan LaSota February 2001 updated June 2001

C_STRING(30;$listName)
ARRAY STRING(40;$aTemp;0)
ARRAY LONGINT($aTempPosition;0)

READ ONLY([UserDataLists])

$listName:="agreement_type"
QUERY([UserDataLists];[UserDataLists]listName=$listName)
SELECTION TO ARRAY([UserDataLists]value;$aTemp;[UserDataLists]position;$aTempPosition)
SORT ARRAY($aTempPosition;$aTemp)
ARRAY TO LIST($aTemp;$listName)
```

Additional lists can be added by duplicating the code block at the bottom of the method.

In order to implement this system it is critical that your methods run consistently when the database is shutdown, and started. The only way to ensure that an action occurs at these times is to use the Database Methods “On Server Startup”, “On Server Shutdown” for instances of 4D Server or “On Startup” and “On Exit” for instances of standalone 4D. You can use the function **Application Type** to determine whether you are running under 4D Client or standalone

4D if your application exists in both environments. Please consult the 4D Language Reference under the Database Methods Section.

In all likelihood you will not want these methods to run each time a client starts or exits your data system. Doing so would only add to the length of time it would take for them to begin or quit a session.

Please note that these methods will not work if you need to store hierarchical lists. If you need to store and retrieve hierarchical list data you will have to revamp the structure and methods to accommodate the more complex hierarchical list structure. But the same idea would apply.

Dan LaSota is a registered 4D Partner and has used 4D as a design tool since 1989. He can be reached at [dan@ae-data.com](mailto:dan@ae-data.com).