#19 Checkbox Indirection with Subfiles

Written by Dave Dell'Aquila October, 1987

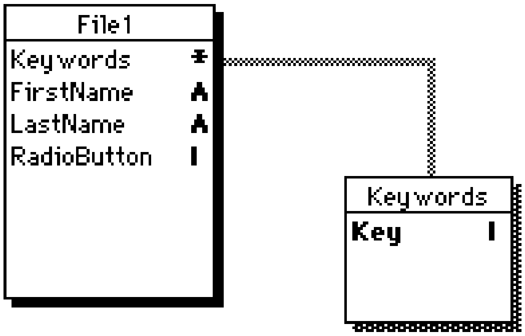
This tech note describes how to efficiently store the values of check boxes in your database.

The natural way to store check box values would involve storeing a boolean value (a 0 or a 1) for a check box in one of the 511 fields available in each file. Storing check boxes in this way would also rapidly use up the number of fields you can use in each file. A first analysis of the problem would lead one in the direction of using several packing algorithms to store the values of booleans in long integer fields, one boolean per bit, or perhaps packing the values of the booleans into a text string, then parsing the values in the string via the **Substring** command. These methods limit the number of booleans they can store (a long int would only store 32 booleans) and the text field would take considerable time in parsing the string for the correct values. Additionally, you can't perform indexed searches on either of these methods because you must have an algorithm to analyze the particular values to find the boolean you're interested in. A third approach would be to sum the values of the booleans raised to the power of their respective position, and then store the result. Again, an algorithm must be designed to pack and unpack the number to determine which booleans should be on or off.

None of the above methods allow the user to perform an indexed search on a boolean field because of the processing that must go on in order to extract the desired information.

There is however a solution which utilizes the subfile and indirection capabilities of 4th Dimension. This Tech Note will describe that process.

THEORY:
This technique takes advantage of 4th Dimension's subfiles and indirection to overcome the obstacles encountered in the other approaches. Basically, we want to save a subrecord which contains the numeric value for the booleans that have the value of 1, and make sure that there are no subrecords for the booleans that have the value of 0. As it turns out, 4th Dimension's subfile structure lends itself naturally to this approach. The structure would look like:



We will also need to associate a number with each check box so that we can store the appropriate integer in the subfile. In the input layout, you name the check boxes in sequential array order, as you would with scrollable array elements. For example, the first check box could be named cb1, the second cb2, etc. Once you've committed a name to a check box, it's committed - don't change it.

After the record has been accepted by the user, we use indirection to save the values of the check boxes to the subfile. We call a global procedure to do this and pass it the the number of check boxes to update which is stored in the zero element for the check box array which is passed to the $1 local variable. The global procedure *SaveVars* looks like:

```
  `$1=number of check boxes to save (controls number of iterations)

$i:=0   `set local counter to 0

While ($i<($1))   `loop while there are still check boxes to save...

 $i:=$i+1

 SEARCH SUBRECORDS([File1]Keywords;[File1]Keywords'Key=$i)   `search if value exists

 Case of    `two things are possible
    `check box is ON and no subrec
   : ((cb{$i}=1)&(Records in subselection([File1]Keywords)=0))
      CREATE SUBRECORD([File1]Keywords)
      [File1]Keywords'Key:=$i
    ` check box is OFF and there is a subrec
   : ((cb{$i}=0)&(Records in subselection([File1]Keywords)#0))
      DELETE SUBRECORD([File1]Keywords)
 End case

End while
```

Starting with the value 1, we search the subfile to see if there is a subrecord that contains the value "1". Depending on whether or not a subrecord is found, four things are possible, but we only concern ourselves with two:

1. The check box ending with the current count value is ON and there is no subrecord with the current count value. In this case we have to create a subrecord and store the current value of $i in the subfield.
2. The other possibility is when the check box is OFF and a subrecord for that check box exists. In this case we must delete it.

In the event that the check box is on and a corresponding subrecord exists we do nothing. If the check box is off and there is no subrecord, we do nothing.

The procedure walks through all the values up to the number of check boxes in the layout, and performs the above test to each one.

Now the appropriate subrecords are saved, we have to have a way to set the check box variables when the layout procedure is first executed. In the before phase, we call the following global *LoadVars*:

```
 `This procedure assigns values to the check boxes in the layout

ALL SUBRECORDS([File1]Keywords)

`search for all positive integers in subfile
SEARCH SUBRECORDS([File1]Keywords;[File1]Keywords'Key>0)

While (Not(End subselection([File1]Keywords)))
 cb{[File1]Keywords'Key}:=1
 NEXT SUBRECORD([File1]Keywords)
End while
```

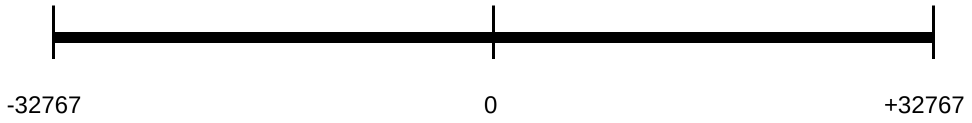
Whenever a layout is opened in 4th Dimension, all the check boxes are set to zero. This global procedure first searches for all positive subrecords (see Variations below). Then while there are subrecords remaining, assign the value of 1 to the check box using indirection. For example, if there are only two subrecords in the subfile, say 3 and 5, then we must "turn on" check boxes cb3 and cb5. The while loop will only execute twice, first time setting cb{3} to 1 then going to the next subrecord in the selection and setting it's check box to 1; cb{5}:=1.

When the procedure finishes, the correct check boxes are activated.

ADDING CHECK BOXES:
The process of adding check boxes is simplified due to the nature of this approach. All that needs to be modified is the addition of a check box with the next integer in the sequence. In other words, if you are adding an eleventh check box, it's variable name would be cb11. You will also update the zero element which contains the number of check boxes, so in this example cb0 would be assigned the value 11.

SUMMARY:
In the layout procedure, you call *LoadVars* in the **Before** phase. The user clicks on the check boxes, then either accepts or cancels the record. If the record is accepted, you call *SaveVars* in the **After** phase.

VARIATIONS:
The subfile structure within 4th Dimension is capable of storing up to 32,767 subrecords. So far this tech note has only made use of the positive integers. Graphically, we can think of it as a line representing the range of possible values we can use:



In a future tech note I will discuss how to use the negative portion of this spectrum to store array elements using indirection. It is also possible to modify the global procedures for storing and retrieving check boxes that are displayed in dialogs using the same subfile. This will also be addressed in a later tech note.