The Fox Hunt

By Whil Hentzen

In the past few columns, we've spent a lot of time talking about tools underlying our application. We're going to spend the next few columns discussing user interface issues. In each episode, we'll tend toward the use of data driven techniques instead of just using "Brute Force." This will have the dual advantages of reusability and reliability. If we can call the same function from multiple screens, or even multiple applications, and just customize it by passing a few parameters, we'll save time and make our application more robust. Additionally, since we'll eventually be using code that has been implemented over and over, we can be more comfortable knowing that it's been tested and it works.

Our topic this month is the creation of a browse that enables the user to navigate through the table. The user will press a pushbutton on the screen, and a browse that displays the records in the primary table will appear. The user will be able to scroll through the Browse window, and, when they press Enter, the record pointer will be positioned on that record.

This actually isn't too hard, and in fact, really doesn't require the use of any data-driven techniques, does it? All we need to do is scroll through the fields in the table and build a BROWSE FIELDS string with them. However, there are two special things we're going to do. This month, we're going to show you how to control which fields in the table will show up in the browse. We may not want to have fields like internal keys or timestamp fields appear. Next month, we'll provide an additional object on the screen - a checkbox - that will bring forward a list of the fields in this table and allow the user to select which fields from the list will appear in the browse - and in what order.

We'd like to be able to use this tool in screen after screen, and so we'll have to be able to find the list of fields automatically. We don't want to have to write code that includes those field names. To begin with, this would be time-consuming and laborious to do in screen after screen. Furthermore (and this is the worst part), it would be a real pain to maintain - each time you change the fields in the table, you'd have to change the program. This would become a drag very quickly.

Instead, we're going to keep the names of all of our tables and fields in the system in a table called SYSCOL, and then access that table when populating the browse. Here's what the structure of this table looks like.

Field	Field Name	Type	Width	Dec
1	TABLENAME	Character	10	
2	COLUMNNAME	Character	10	
3	PROMPTNAME	Character	40	
4	COLUMNPOS	Numeric	3	0

And this is what the contents of the table looks like:

Tablename	Columnname	Promptname	Columnpos
DOCTOR	CIDDOCT		0
DOCTOR	CNADOCT	Doctor Name	2
DOCTOR	CNODOCT	Doctor Number	1
DOCTOR	CADD1DOCT	Doctor Address Line 1	3
DOCTOR	CADD2DOCT	Doctor Address Line 2	4
DOCTOR	CCITY	Doctor City	5
PATIENT	CIDPAT		0
PATIENT	CNAPAT	Patient Name	2
PATIENT	CSSNPAT	Patient Social Security No	1

When the screen is called, the Setup snippet (or wrapper program if you desnippetize) does two important things. First, it selects the table that is the subject of the screen. For example, if we're in the Doctor maintenance screen, then the program performs a SELECT DOCTOR command in order to make sure that the current work area has the Doctor table. We also assign the alias name to a memory variable that we'll call m.jcTableName. If the user just presses the Browse pushbutton, the following code executes:

```
select PromptName, ColumnName ;
 from SYSCOL ;
 where !empty(promptname) ;
 and  TableName = m.jcTableName ;
 orde by columnpos ;
 into arra a2PossFields
```

This command pulls the names of all of the fields for the current table that have a data in the Promptname field. This provides two benefits. First, we can decide which fields will show up in the browse and which will not just by placing data in the Promptname field. Second, since a fieldname like "cnapat" is probably not going to be very helpful to the user, we can use the contents of this Promptname field as the heading for the column.

Immediately after the SELECT statement, we check to see if there was anything selected. If not, we warn and bail.

```
if _tally = 0
 wait wind "There are no English names in SYSCOL"
 retu .t.
endi
```

Next, we create a text string that contains the FIELDS clause for the BROWSE command.

```
m.jcBrowFlds = "field "
for i = 1 to alen(a2PossFields) step 2
 m.jcNextFld = ;
  + allt(a2PossFields[i + 1]) ;
  + ":H='" + allt(a2PossFields[i]) +"'" ;
  + iif(i+1 <> alen(a2PossFields), ", "," ")
 if len(m.jcBrowFlds) + len(m.jcNextFld) < 1900
  m.jcBrowFlds = m.jcBrowFlds + m.jcNextFld
 else
  * Remove trailing comma from previous field
  m.jcBrowFlds = substr(m.jcBrowFlds,1,len(m.jcBrowFlds)-2) + " "
  exit
 endif
endfor
```

There are a couple of notable items about this simple FOR...ENDFOR loop. FIrst, notice that we STEP 2 in the FOR command. This is because the a2PossFields array is two columns wide and we don't want to run the FOR loop for every element in the array, just every row. Next, you'll see that we have a test for the length of the string to be less than 1900 characters. This is because FoxPro has a line limit per command line and compiled line. We need to make sure we don't go over, and we need to leave room for the rest of the Browse command.

Finally, we call the browse. We'll remap the Enter key to act as the Ctrl-W key so that the user can press Enter to select a new record. If they have done so, we'll be sure to assign this record number to the memory variable that keeps track of this in the screen.

```
*
* the window w_browse is defined in the main program
*
 on key label ENTER keyboard "{CTRL-W}"
 browse ;
  noappend nodelete noedit nomenu ;
  &jcBrowFlds ;
  title "<Enter> to Choose" ;
  window w_browse ;
  color scheme 10
 on key label ENTER

*
* if the user moved the record pointer while in the browse, we're
* going to go to that record back in the window
*
m.jnCurRec = recn()
```

Next month, we'll explore the use of a "Mover" dialog that will enable us to allow the user to select the fields they want to see in the Browse window.