The Fox Hunt

By Whil Hentzen

We're going to conclude our discussion of lookup tables by creating the functions that allow the user to add, edit, and delete items in a specific lookup table. These functions are fairly straightforward because of the way we've designed the lookup table to begin with.

The easiest function to handle, in some ways, is the Delete function. The user simply selects a lookup table, then selects an item in that lookup table, and presses Delete. Since every record in the lookup table has a unique key, we just delete the record with the key that matches the key of the currently highlighted record.

The array that populates the listbox that contains the lookup tables' items has three columns - the first contains the value that appears in the listbox - either the Description or the User Entered Value concatenated with the Description, the second contains the key for the item, and the third contains the User Entered Value by itself. Thus, the code attached to the Valid clause of the Delete pushbutton would look like this:

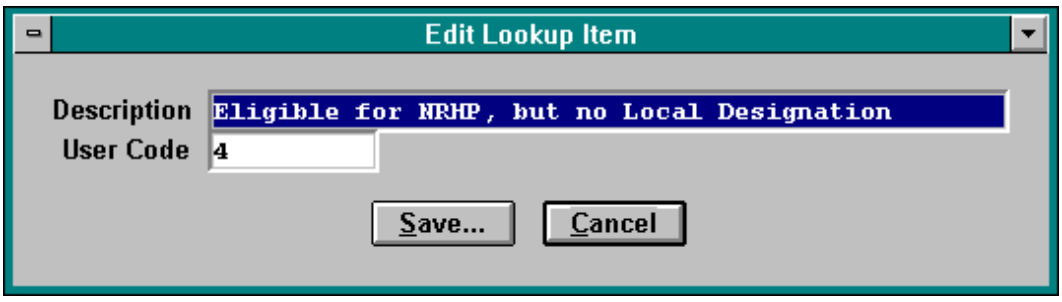
```
m.jnX = f_MsgBox("Delete this record?", "Watch Out!", 4, 32, 256 )
if m.jnX = 6
 delete for cCd = aAvailCh[ascan(aAvailCh, m.cAvailCh)+1]
 wait wind nowait "Record deleted"
else
 wait wind nowait "Delete cancelled"
endif
=v_cAvailTab()
```

The f_MsgBox function is simply a wrapper for the MsgBox function in FoxTools as discussed in earlier issues of the FoxPro Developer's Journal. If it returns the value 6, the user selected the "Yes" button in the dialog, and we delete the record. The ASCAN function looks for the listbox's currently highlighted item in the array, and then moves over one element to snatch the key value for that item. The v_cAvailTab function refreshes the items in the listbox.

Remember how I added the "in some ways" caveat up above? Well, the sharp-eyed among you may be thinking, "What if we've used this lookup value in a table somewhere? We've just destroyed any reference to it." You're absolutely correct. Typically, I only allow selected personnel to delete values in a lookup table - such as a system supervisor - or even restrict access to this function to developer level personnel only. Usually someone wants access to the Delete function so that they can clean up or revert mistakes that they just made.

You can, of course, write code that would cascade the lookup value delete function throughout the application, but that's an awful lot of work, and typically not used very often.

Now with Delete out of the way, let's take a whack at the Add and Edit functions. We're going to cover both at the same time, because they have a lot in common. The user accesses them in different ways - they press the Add button to add a new lookup value, but merely double-clicks or presses Enter when the desired item is highlighted. In both cases, we call the same function, f_cLUAdEd, albeit with an Add or Edit parameter. Furthermore, we use the same dialog, and simply change the title bar from "Add Lookup Item" to "Edit Lookup Item" depending on how the dialog was called.



Let's look at the function code:

```
*************************************************************************
function f_cLUAdEd
*************************************************************************
*
* useage: f_cLUAdEd("Add")
*
* called both from:
* - doubleclicking on an item in the listbox (pcLUMode = "Edit")
* - selecting the Add pb on the LUMAINT screen (pcLUMode = "Add")
para m.pcLUMode

m.cWinTitle = " " + m.pcLUMode + " Lookup Item "

*
* initialize memvars in case we add a new lookup item
* these memvars match up with fields in the ITLOOK table to
* identify which lookup table the item belongs to
*
m.cNaTable = aAvailTab[ascan(aAvailTab, m.cAvailTab)+1]
m.cNaTabEngl = m.cAvailTab
m.cCd = aAvailCh[ascan(aAvailCh, m.cAvailCh)+1]

*
* if the current lookup table has a user code, strip
* it out from the value in the listbox in order
* to find the description
*
if m.lLUHasUC
 m.cDe = subs(aAvailCh[ascan(aAvailCh, m.cAvailCh)], 7, 50)
else
 m.cDe = aAvailCh[ascan(aAvailCh, m.cAvailCh)]
endif
m.cuev = aAvailCh[ascan(aAvailCh, m.cAvailCh)+2]

* set a parm here indicating we are editing or adding
if upper(m.pcLUMode) = "ADD"
 *
 * assign empty values to memvars for the Add/Edit screen
 *
 m.lAddNotEd = .T.
 m.cCd = ""
 m.cDe = ""
 m.cuev = ""
else
 m.lAddNotEd = .F.
endif

* handles the f_when()
m.plCanEdit = .T.

do ZLUADD.SPR

=v_cAvailTab()
return .t.
```

The cWinTitle memory variable is used in the title of the ZLUADD screen that is called from this function. That way, we can sorta customize the screen and help the user remember what they're doing once they get to that screen. The next few lines are used to initialize memory variables that will be used when we add a new item to a lookup table. We already know what lookup table we're in, so we stuff that information into the appropriate fields in the new record.

Next, we determine the values for the Description, the User Entered Value (if there is one), and the item's primary key. If we're editing, we use the values from the current record, otherwise, we create memvars with empty values.

Finally, we set a pair of flags that we use in the Add/Edit screen itself. The m.plCanEdit flag is used in the When and Valid snippets of the Description and User Entered Value GETs in the ZLUADD screen. In order to provide a visual clue for the user as to whether they made a change to the contents of a field, the Save and Cancel buttons are originally dimmed. Each GET field contains a When() snippet that traps the original value of the field and a Valid() snippet that checks to see if the current value matches the original value. If the user did change the value, the Valid() contains code to enable the Save and Cancel pushbuttons.

Additionally, the When() snippet also has a flag that determines whether the user can edit at all. If the flag, m.plCanEdit, is set to false, we don't allow the user to edit the field by returning .F. from the When() snippet.

```
*************************************************************************
```

```
function f_when
*********************************************************************
*

if !m.plCanEdit
 return .f.
endif
*
* the variable m.guOldVal is global
*
m.guOldVal = eval( "m." + varr() )
return .t.

*********************************************************************
function f_valid
*********************************************************************
*

*
* this is the text string that represents the memvar on the screen
*
m.pcCurVar = "m." + varr()

*
* this is the value of the memvar
*
m.puCurVal = eval(m.pcCurVar)

if !(m.guOldVal == m.puCurVal)
 m.glHasChanged = .t.
 show get m.cSave enabled
 show get m.cCancel enabled
endif

*
* get rid of the old value in case the user makes another change
*
m.guOldVal = ""

return .t.
```

The other flag, m.lAddNotEd, is used in the Save routine of the Add/Edit screen. It controls whether a new record is added or new data is simply put into the current record.

```
*********************************************************************
func v_cLUASave
*********************************************************************
*
priv m.pcNewKey

*
* initialize audit fields
*
m.cLMod = m.gcNaUser
m.nLMod = f_ts()

if m.lAddNotEd
 * adding
 pcNewKey = f_incrid( "cCd" )
 if !empt( m.pcNewKey )
  if !m.lLUHasUC
   m.cuev = ""
  endif
   insert into ITLOOK ;
   (cNaTabEngl, cNaTable, cCd, cDe, cuev, lAvail2Use, cCdSubsys, cLMod, nLMod ) ;
   values ;
   (m.cNaTabEngl, m.cNaTable, pcNewKey, m.cDe, m.cuev, m.lAvail2Use, m.cCdSubsys, m.cLMod, m.nLMod)
 else
  m.jnX = f_MsgBox("Key not available; record not added")
 endif

else
 * editing
 * f_push saves current the current work area, index tag, and record position
 =f_push()
 sele ITLOOK
 set orde to cCd
```

```
 if seek(m.cCd)
  if !m.lLUHasUC
   m.cuev = ""
  endif
  gather memvar fields cDe, cuev, cLMod, nLMod
 else
  wait wind "DEV: why can't we find " + m.cCd + " when gathering (nothing saved)"
 endif
 * f_pop restores the current work area, index tag, and record position
 =f_pop()
endif

return .t.
```

The Save function contains two parts - one for saving a record if we're adding a new record and another when we're editing an existing record. We also handle the situation of whether or not there is a User Code in the current lookup table in each case.

Now that we're done setting up the screen, we can call the screen itself, and the screen has a couple of tricks. The first one is the magic changing title. In the setup snippet (and this can't go in a PRG that calls the screen), we use the command

```
#ITSEXPRESSION ~
```

and then in the screen's title box itself, we enter the string

```
~cWinTitle
```

When the screen is generated and run, it will interpret the ~ to mean "the string following this character is a memory variable, so evaluate it instead of placing the string (cWinTitle) into the title bar of the window." Thus, when we initialized m.cWinTitle to be either "Add Lookup Item" or "Edit Lookup Item", we were preparing this memory variable to be used as the window's title.

The second trick is to use the GENSCRNX directive *:IF in the screen itself. We placed it in the comment snippet of the User Entered Value field.

```
*:IF m.lLUHasUC
```

This will cause the User Entered Value field to appear only if the memvar m.lLUHasUC is true. That way, the user doesn't even see a User Entered Value GET on the screen if it's not appropriate.