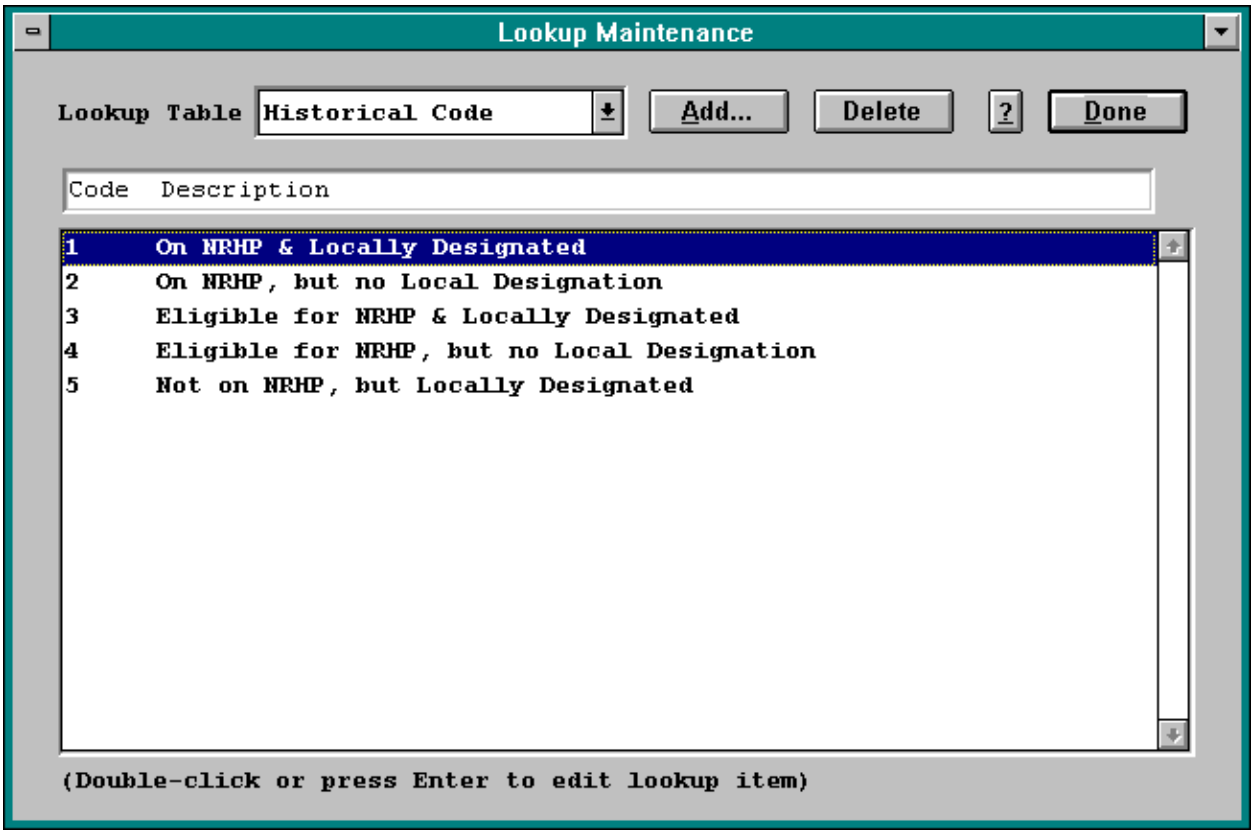
The Fox Hunt

By Whil Hentzen

Last month we investigated the use of a single table to hold all of our lookup tables, and promised that we'd address the maintenance screen for this table next. This topic bears some discussion because it's full of nasty little requirements that could trip up the unwary developer. Let's look at some of these points.

Our lookup maintenance screen will look like the following figure. The user can use the Lookup Table dropdown listbox to choose a table, maneuver through the items in a specific lookup table with the listbox, and use the Add and Delete pushbuttons to add and delete entries in the current lookup table. The user can also double-click or press Enter when an item is highlighted in the listbox in order to Edit that item. Adding or Editing an item causes the same dialog to appear, and we'll discuss how it works a little bit later.



You may wonder why there isn't a facility for adding additional lookup tables to our lookup system. This is because a lookup table is directly tied to a field in another field, via a foreign key and specific values. The only reason you'd need a new lookup table is because you - the developer - changed the table structure so that it requires an additional field, and accordingly, you'd need to add another lookup table as well. In other words, the user can't do anything to the system that would require them to be able to add a lookup table.

There are a couple of bits of slight-of-hand going on with the lookup table dropdown listbox. First, we're storing the names of the various lookup tables in the lookup table itself, and in addition, we're storing an "English" name (or, more accurately, a "user-friendly" name) in the lookup table as well. When we use the lookup values when doing screen validations, we'll be referring to the lookup table's cryptic name, but we'll display the friendly name in the dropdown for the user.

Some of the lookup tables are used internally in the system, or have values that shouldn't be modified by anyone but a developer. The records for these values have the field lAvail2Use set to False; the rest have a True stored in them. We use a SQL SELECT command to populate the dropdown's array, and select either all lookup tables or just those with the field lAvail2Use set to True. The following code goes in the setup snippet or in the screen's calling program:

```
* ZLUMAINT.PRG
*
* lookup maintenance program
*
* if we are a developer, we allow the tables
* where there are "hidden" lookup tables to be shown
```

```
* these tables are flagged via the lAvail2Use field
*
if upper(m.gcMethod) = "DEV"
 select distinct cNaTabEngl, cNaTable ;
  from ITLOOK ;
  where empt(cCdSubsys) ;
   or cCdSubsys = m.gcCdSubsys ;
  order by cNaTabEngl ;
  into array aAvailTab
else
 select distinct cNaTabEngl, cNaTable ;
  from ITLOOK ;
  where lAvail2Use ;
   and (empt(cCdSubsys) ;
   or cCdSubsys = m.gcCdSubsys) ;
  order by cNaTabEngl ;
  into array aAvailTab
endif

m.cAvailTab = aAvailTab[1,1]
if _tally = 0
 m.jnX = f_MsgBox("There are no lookup tables in this application")
 return .t.
endif

*
* initialize the array to populate the item listbox
* "Available Children"
*
declare aAvailCh[1]
aAvailCh[1] = ""
m.cAvailCh = aAvailCh[1]

*
* whether or not LU has user codes
*
* this is used to determine whether we show the UC field
* in the LU add screen
*
m.lLUHasUC = .f.

* initialize the header above the listbox in ZLUMAIN screen
m.cLUHeader = ""

* this initializes the var that we'll stuff into the
* record during adds
*
if upper(m.gcMethod) = "DEV"
 m.lAvail2Use = .t.
else
 m.lAvail2Use = .f.
endif

*
* populate the dropdown listbox's array
*
=v_cAvailTab()

do ZLUMAINT.SPR

return .t.
```

I use a global memory variable that is set either to "DEV" or to "CUST", depending on a number of conditions, including the permission level of the current user and an optional command line parameter. Note that in the ELSE section of this bit of code, we use the string

```
where lAvail2Use ;
```

without specifying what lAvail2Use is supposed to be equal to. Any expression must evaluate to a logical, and since the field itself is a logical field, we do not have to compare it to a value in order to create a logical expression.

Now that we've got the dropdown filled with the names of the available lookup tables, you've probably figured out that we're also populating the listbox with an array, and we're using a SQL SELECT command in the valid of the dropdown to fill that array each time the value of the dropdown changes. What isn't as obvious is the contents of the listbox. As we discussed last month, some lookup tables have a primary key and a description, while others have a primary key, a description, and a user-entered value. In the first case,

the lookup table would typically be used to populate a dropdown listbox from which the user can select a number of pre-defined values. In the second case, the lookup table would be used to validate a text entry that the user makes. Optionally, we may want to display the contents of the description field after the user enters the value, so that they can confirm that they entered the correct value.

This functionality presents a problem, in that we'll only be using the User Entered Value for some lookup tables. While we could kludge around, it would be much nicer if we just hid the User Entered Value field when appropriate. Thus, the listbox will display both fields if there is something in both, and just the description field if the User Entered Value field isn't used.

We'll accomplish this by counting how many records in the currently selected table contain something in the User Entered Value field, and if there are any, then we'll grab that field as well. The following code goes in the valid of the dropdown listbox (without the function name) or as a subroutine in the ZLUMAINT.PRG program file:

```
**************************************************************************
function v_cAvailTab
**************************************************************************
* populates the cAvailCh listbox for the selected table
* it may have a UC and a DE or just a DE
*

* determine whether we place the UC field in the array
declare aJunk[1]
aJunk[1] = 0
select count(*);
 from ITLOOK ;
 where !empty(cuev) ;
  and upper(cNaTable) = aAvailTab[ascan(aAvailTab, m.cAvailTab) +1] ;
 into array aJunk

* don't need to see if junk exists cuz a count will
* always return a value, even if it's 0

if aJunk[1] = 0
 * this table has no user codes
 * the third column is just for consistency sake
 select cDe, cCd, cuev ;
  from ITLOOK ;
  where upper(cNaTable) = aAvailTab[ascan(aAvailTab, m.cAvailTab) +1] ;
  order by cDe ;
  into array aAvailCh
 m.lLUHasUC = .f.
 m.cLUHeader = "Description"
else
 * this table has user codes
 select cuev + " " + cDe, cCd, cuev ;
  from ITLOOK ;
  where upper(cNaTable) = aAvailTab[ascan(aAvailTab, m.cAvailTab) +1] ;
  order by cuev ;
  into array aAvailCh
 m.lLUHasUC = .T.
 m.cLUHeader = "Code  Description"
endif
m.cAvailCh = aAvailCh[1]

select dist(lAvail2Use) ;
 from ITLOOK ;
 where upper(cNaTable) = aAvailTab[ascan(aAvailTab, m.cAvailTab) +1] ;
 into arra aJunk
if _tally = 0
 * if we've messed up the data in the table...
 wait window "Dev: The SELE DIST(lAvail2Use) returned 0"
 m.lAvail2Use = .t.
else
 m.lAvail2Use = aJunk[1]
endif

show gets
return .t.
```

This function will also store one of two strings to the variable m.cLUHeader. This memory variable appears on the screen above the listbox, and changes according to whether we have a User Entered Value (a code) or not.

We've now set up our lookup maintenance screen so that we can maneuver through the various lookup tables, and have added several enhancements. Next month, we'll cover Add, Edit and Delete functionality.