The Address Book Application (ADDRESS.FSL)

The ADDRESS.FSL form is an online address book. You can display information about a customer by using one of the following methods:

- Choose a letter from the list on the right side of the form; all customers whose names begin with that letter appear in the Name list at the bottom of the form.
  For example, choosing K retrieves all customers whose names begin with K.
- Press the Search by Customer Number button; the Locate Value dialog box appears.
  Enter the Customer number in the Value box.
  Make sure @ and .. is chosen and that Customer No appears in the Fields box, then click OK. Information for the specified customer appears in the top portion of the form.
  For example, entering 1221 in the Locate Value dialog box displays information about the Kauai Dive Shoppe.

You can also use Paradox's built-in tools to navigate and edit the table.

Reference

For information on the programming techniques used to create the Address Book application, see Address Book Programmer's Help.

Address Book Programmer's Help

The ADDRESS.FSL form is an online address book. With just a few small methods, this application provides quick and easy access to data. This application demonstrates one technique for showing the results of a query in a form.

If you want to create a similar application, you must know how to create tables and forms, and how to bind forms and table frames to tables, as explained in the *Paradox for Windows User's Guide*. This Help system contains a brief description of the underlying table structure of this application and the design objects placed on the ADDRESS.FSL form.

Reference

Select one of the following objects for a description of the code attached to it:

LETTERS table frame

foundTF record objects

searchCustNum button

Table Structure

The following tables are part of the underlying structure of the Address Book application:

- CUST.DB contains the customer information used in the address book.
- XXX.DB is an empty table with a single field: Temp (A1). It is a temporary table used to unbind *foundTF* from FOUND.DB during the query.
- FOUND.DB has one field: Name (A30). It displays the customer names that begin with the letter selected from the table frame.
- LETTERS.DB has one field: Letter (A1), containing 26 records (one for each letter of the alphabet). It contains the letters in the table frame.

See Also

Design objects

Design Objects

The following design objects are placed after binding the form to the tables:

- A multi-record object is bound to the *Cust* table; its record layout is set to 1 across and 1 down. Its name is CUST by default.
- A table frame is bound to the *Letters* table. Its name is LETTERS by default.
- A table frame is bound to the *Found* table, and its name is then changed to *foundTF*.
- A button named *searchCustNum*.

See Also

Table structure

Form Methods

Code attached to the Address form performs two tasks: it makes sure you're running the application from the working directory, and it invokes the Help application when you press F1.

Select one of the folllowing methods for a description of the code attached to the form:

open method

keyPhysical method

Form open Method

Code attached to the form's built-in **open** method makes sure you're running this application from the working directory. The call to **isFile** does the checking: when you give a file name without a path or an alias, Paradox looks in the working directory by default.

```
method open(var eventInfo Event)
  if eventInfo.isPreFilter() then
     ;code here executes for each object in form
  else
     if not eventInfo.isPreFilter() then
           if not isFile("address.fsl") then
                 msgInfo("Startup Error!", "The ObectPAL example files must
  be
                             in the working directory.")
                 close()
           endIf
     endIf
  endIf
endmethod
```

Form keyPhysical Method

Code attached to the form's built-in **keyPhysical** method invokes the Windows Help application to display context-sensitive help when you press F1. The call to **vChar** identifies each key as it's pressed. The call to **disableDefault** blocks Paradox's built-in response to F1.

```
method keyPhysical(var eventInfo KeyEvent)
  if eventInfo.isFirstTime() then
     if eventInfo.vChar() = "VK_F1" then
           disableDefault
           helpShowIndex(":WORK:address.hlp")
     endIf
  endIf
endmethod
```

The *LETTERS* Table Frame Methods

Like all table frames, *LETTERS* is a compound object. You can inspect the objects in a compound object just as you would any others, and you can attach code. Remember, code attached to the built-in methods for the single record object affects all records in the table frame.

Select one of the following methods or properties for a description of the code attached to the *LETTERS* table frame:

open method

setFocus method

TableName property

Select one of the following objects for a description of the code attached to it:

foundTF record objects

searchCustNum button

The Table Frame open Method

The following code is attached to the built-in **open** method for the record object. It uses the RowNo property to specify alternating colors for the rows in the table frame. The RowNo property returns the row number of the table frame, not the underlying table.

The record at the top of the table frame is row 1, the next row is row 2, and so on for all the records displayed in the table frame. As each record object in the table frame opens, this code executes. Because *Self* refers to the object executing the code, the following statement assigns a different value to *rn* for each record:

```
rn = Self.RowNo
```

Here is the built-in **open** method code:

```
method open(var eventInfo Event)
  var
     rn LongInt
  endVar

  rn = self.rowNo
  if rn.mod(2) = 0 then
     self.color = Gray
  else
     self.color = White
  endIf
endmethod
```

The Table Frame setFocus Method

The following code is attached to the **setFocus** method built into the field object *Letter*. This code is attached to **setFocus** rather than **canArrive** because **setFocus** sets the highlight in a field object, making it easier to see which letter was chosen. This code (and this application) demonstrates a way to show the results of a query in a form. Refer to the *Paradox for Windows User's Guide* for more information about binding forms to queries.

```
method setFocus(var eventInfo Event)
   var
      ss string
      qq query
   endvar

   doDefault
   delayScreenUpdates(Yes)
   foundTF.visible = No              ; hide the table frame

   foundTF.TableName = "xxx.db"      ; bind to XXX.DB while we write to
  FOUND.DB
   DMRemoveTable("found.db")         ; unbind FOUND.DB so we can write
                                     ; the query results to it

   ss = self.value + ".."            ; assign a value to the variable
                                     ; used as tilde variable in query

   qq = query                        ; build a query statement

        cust.db | Customer No | Name      |
                | Check       | Check ~ss |

        endQuery

   executeQBE(qq, "found.db")        ; run the query, write results to
                                     ; FOUND.DB

   foundTF.TableName = "found.db"    ; add FOUND.DB to data model, bind it
  to
                                     ; foundTF
   foundTF.visible = Yes             ; display the updated table frame
  foundTF
   delayScreenUpdates(No)
endmethod
```

The Table Frame TableName Property

You use the TableName property twice: first to bind *foundTF* to XXX.DB, which unbinds FOUND.DB to run the query and store the results, and then to rebind *foundTF* to FOUND.DB to display the results. When you use the TableName property to bind a table frame to a table, it adds the table to the form's data model. The Form type procedure **DMRemoveTable** is called to unbind FOUND.DB and remove it from the form's data model while you run the query and write the results. If **DMRemoveTable** is not called, FOUND.DB would be bound to the form and opened, and the attempt to write the query results would fail.

The *foundTF* Record Object

The table frame *foundTF* is a compound object that contains a header, a record object, and some field objects. The record object displays the names and customer numbers found by the query. You can choose a name or number to display more information.

Select the following method for a description of the code attached to the *foundTF* record object:

canArrive method

Select one of the following objects for a description of the code attached to it:

LETTERS frame

searchCustNum button

The *foundTF* canArrive Method

Code attached to the record object's built-in **canArrive** method blocks a move to the blank record at the end of the table. **canArrive** effectively asks permission to move focus to an object. This code uses the BlankRecord property to see if the record you're trying to move to is the blank (empty) record at the end of the table.

```
method canArrive(var eventInfo MoveEvent)
  var
     destObj UIObject
     destVal AnyType
  endvar

  if self.blankRecord then
     eventInfo.setErrorCode(CanNotArrive)
  else
     eventInfo.getDestination(destObj)
     CUST.locate(destObj.name, destObj.Value)
  endIf
endmethod
```

Because the constant UserMove is used with **eventInfo.reason**, the custom code for this method executes only when **canArrive** is triggered by the user interacting with the form, but not when it's triggered by Paradox or an ObjectPAL statement. If the move is allowed, the following statement uses the Event type method **getDestination** to assign to *destObj* the UIObject that was the destination of the move.

```
eventInfo.getDestination(destObj)
```

Then, in the following statement, *destObj.name* returns the name of the destination object, *destObj.value* returns the object's value (name and value are UIObject properties), and this data is passed to the **locate** method, which searches the *CUST* multi-record object.

```
CUST.locate(destObj.name, destObj.Value)
```

The *searchCustNum* Button

This button's **pushButton** method displays Paradox's built-in Locate Value dialog box (described in the *Paradox for Windows User's Guide*). The Locate Value dialog box waits for you to enter a number, then searches the table. If the search succeeds, it displays the results; if it fails, it displays a message. Because the Locate Value dialog box is a built-in Paradox dialog box, you can't control it using ObjectPAL. It's up to the user to use it correctly.

```
method pushButton(var eventInfo Event)

;-----------------------------------------------------------------------
; This method attempts to locate a user-entered customer number, using
; the built-in Locate Value dialog box.
;-----------------------------------------------------------------------

  var
     retVal Logical
  endvar

  CUST.Customer_No.moveTo()         ; move to Customer No field
  retVal = CUST.action(DataSearch)   ; call dialog and perform search
  If not retVal then                ; If search failed...
     beep()                          ; beep and display a message
     msgInfo("Oops!", "Either you canceled the dialog box or I couldn't
                       find the number you entered.")
  endIf

endmethod
```

You can use the UIObject type method **action** with ObjectPAL constants like DataSearch to access many Paradox facilities. For more information about using **action,** see the *Object PAL Reference*.

Select one of the following objects for a description of the code attached to it:

LETTERS frame

foundTF record objects

Help Button Methods

Code attached to the built-in **pushButton** method of the button named *helpButton* invokes the Help application to display help for the Checks form.

```
method pushButton(var eventInfo Event)
  message("Loading HELP; one moment, please...")
     helpShowIndex(":WORK:address.hlp")
  message("")
endmethod
```