The Checkbook Application (CHECKS.FSL)

Use the Checkbook application to enter expenditures and deposits. To enter a transaction, follow these steps:

1. Move to the empty row at the bottom of the form.
2. Enter a check number in the Num field if you are recording a check expenditure and move to the next field.
3. Enter a date in MM/DD/YY format in the Date field and move to the next field.
4. Enter a description of the transaction in the Memo field and move to the next field.
5. Enter the amount of the transaction in the Out field (if you are recording an expenditure) or the In field (if you are recording deposits).

   The amount of the transaction appears in the Amount field; expenditures are enclosed in parentheses. The Balance field is updated, reflecting the transaction. If the account is overdrawn, the Balance amount appears in red.
6. Your changes are saved when you close the form.

Reference

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

Checkbook Programmer's Help

The Checkbook application, CHECKS.FSL, is used to enter expenditures and deposits, which are stored in a Paradox table. The application uses a calculated field to calculate the balance after each entry, and displays the total in red characters when the account is overdrawn.

This application shows how to set data dependent properties, how to intercept keystrokes, and how to set the value of one field depending on the value of another field.

Before any code was written, the underlying table was created and a form was bound to it. This Help system contains a brief description of the procedure used to create the underlying table for the CHECKS.FSL application.

Reference

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

CHECKS table frame

Checks Form

CHECKS record object

CHECKS field objects

helpButton

The Underlying Table

Before any code was written, the underlying table, CHECKS.DB, was created and a form was bound to it. The CHECKS.DB table has the following structure:

Field name	Type
xNum	Short Number (Small Int)
xDate	Date
xMemo	Alpha20
xOut	Currency
xIn	Currency
xAmount	Currency

After the table was created, a form was bound to it by choosing File|New|Form. From the Design Layout dialog box, the tabular layout style was chosen to place a table frame in the form. By default, the table frame takes the name of the table it's bound to, so this table frame's name is *CHECKS*.

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

Checks Form
CHECKS record object
CHECKS field objects
helpButton

Form Methods

Code attached to the Checks 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

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

CHECKS table frame

CHECKS record object

CHECKS field objects

helpButton

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 not eventInfo.isPreFilter() then
     if not isFile("checks.fsl") then
           msgInfo("Startup Error!", "The ObjectPAL example files must be
  in the working directory.")
           close()
     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.

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

Table Frame Methods

A table frame is a compound object: when you use the Table tool to place a table frame, you're actually placing several design objects at once. By default, a table frame contains a header, a record object, and one or more field objects, depending on the structure of the underlying table.

You can inspect the objects in a compound object just as you would any others, and you can attach code. To display a diagram of its components, select the table frame, then choose Form|Object Tree. For more information about compound objects, see the *ObjectPAL Reference*.

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

open method

close method

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

Checks Form

CHECKS record object

CHECKS field objects

helpButton

Table Frame open Method

The built-in **open** method puts the table frame *CHECKS* into Edit mode so the user can make changes to the Checks table. The built-in **open** method also turns off the TabStop property for the field objects *Balance* and *xAmount*. This technique is useful when you want to let the user move to a field when certain conditions are met, and block the move otherwise. You can achieve the same effect using Paradox interactively by inspecting each object and unchecking its Tab Stop property.

```
method open(var eventInfo Event)
   self.edit()             ; put self (the table frame) into Edit mode
   Balance.TabStop = False ; prevent user from moving to this field
   xAmount.TabStop = False
endmethod
```

Table Frame close Method

The table frame's built-in **close** method takes the table frame out of Edit mode, committing the last changes to the Checks table, as shown in the following example:

```
method close(var eventInfo Event)
  self.endEdit()  ; take the table frame out of Edit mode, commit last
endmethod
```

Record Object Methods

Like every table frame, *CHECKS* contains a record object. Although there is no SpeedBar tool for creating a record object, you can select the record object that is built into a table frame, set its properties, and attach code to it.

Anything you do to this record object affects all records displayed in the table frame when you run the form. So, 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 for a description of the code attached to the *CHECKS* record object:

arrive method

depart method

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

CHECKS table frame

Checks Form

CHECKS field objects

helpButton

Record Object arrive Method

The following code is attached to the record object's built-in **arrive** method. It highlights the record by setting its color property to white. This methods assume the table frame color is set to gray in the Design window. (In other words, right-click CHECKS in the Design window, choose Color, then choose the gray panel.)

```
method arrive(var eventInfo MoveEvent)
   self.color = White
endmethod
```

Record Object depart Method

The following code is attached to the record object's built-in **depart** method. It turns off the highlight effect by changing the color property to gray.

```
method depart(var eventInfo MoveEvent)
   self.color = Gray
endmethod
```

In the Design window, you can select only one record object in the table frame. Anything you do to this record object in the Design window affects all records displayed in the table frame when you run the form. So, code attached to the built-in methods for the single record object affects all records in the table frame.

Field Object Methods

You attach methods to the field objects *xOut*, *xIn*, *Balance,* and *xAmount* to process keystrokes and trigger calculations. As with records, you can select only one instance of a field object in the Design window, but anything you do affects all instances displayed when you run the form. So, by attaching code to the single selectable instance of the *xOut* field object, you specify the behavior of all instances displayed in the table frame.

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

xOut

xIn

Balance

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

CHECKS table frame

Checks Form

CHECKS record object

helpButton

xOut Field Object Methods

The *xOut* field object is for transactions (like checks) where money flows out of the account. By adding code to its **changeValue** method, it is certain that the amount is subtracted. For simplicity, this method assumes you always enter positive amounts, but does nothing to prevent you from entering negative amounts.

Select one of the following methods for a description of the code attached to the *xOut* field object:

changeValue method

keyChar method

action method

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

xIn

Balance

xOut changeValue Method

A field object's built-in **changeValue** method executes when changes to a field object's value are about to be posted. As long as the cursor is in the field object, Paradox assumes you're still editing the field. When you move the cursor out of the field, Paradox assumes you are finished editing, so it calls the field object's **changeValue** method.

The **doDefault** call executes the built-in code for the **changeValue** method, committing the changed value and displaying it in the field object. This is an important step; the built-in code posts the changed value of the field object. Until it executes, Paradox works with the previous (unchanged) value of the field object.

```
method changeValue(var eventInfo ValueEvent)
   doDefault                          ; execute the built-in code to commit
  changes
   if self.value <> "" then           ; If the field is not empty,
      xAmount.value = -1 * self.value ; insert a negative amount, and
   endIf                              ; trigger xAmount's changeValue
  method.
endmethod
```

The following statement triggers the *xAmount* field object's **changeValue** method immediately; it does not wait for the current method to finish executing.

```
xAmount = -1 * self.value
```

In this application, a transaction can be either a check (entered in *xOut*) or a deposit (entered in *xIn*), but not both. When you type into *xOut*, this method erases the value in *xIn*.

xOut keyChar Method

The built-in **keyChar** method is called when an object gets a key event that isn't translated to an action. When you press a key, Paradox checks to see if it's meaningful (like *F1*, which invokes the Help system), in which case Paradox must take some action. Otherwise, Paradox passes the event to the target object.

The following code checks the value of the *xIn* field object. If *xIn* contains a value, this code blanks it out.

```
method keyChar(var eventInfo KeyEvent)
   if xIn.value <> "" then    ; If the other field is not empty,
      xIn.blank()             ; make it blank.
  endIf
endmethod
```

xOut action Method

The following code executes when you try to paste a value into the *xOut* field object. Like the code attached to this object's built-in **keyChar** method, this code checks the value of the *xIn* field object. If *xIn* contains a value, this code blanks it out.

```
method action(var eventInfo ActionEvent)
   if eventInfo.id() = EditPaste then
      if xIn.value <> "" then    ; If the other field is not empty,
         xIn.blank()             ; make it blank.
      endIf
   endIf
endmethod
```

xIn Field Object Methods

The *xIn* field object is for transactions (like deposits) where money flows into the account. As with the *xOut* field object, *xIn* expects positive amounts, but does nothing to prevent you from entering negative amounts.

Select one of the following methods for a description of the code attached to the *xIn* field object:

changeValue method

keyChar method

action method

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

xOut

Balance

xIn changeValue Method

```
method changeValue(var eventInfo ValueEvent)
   doDefault                  ; execute the built-in code now
   if self.value <> "" then
      xAmount = self          ; Put the value into the xAmount field,
   endIf                      ; and trigger xAmount's changeValue method.
endmethod
```

xIn keyChar Method

When you type into *xIn*, this method erases the value in *xOut*.

```
method keyChar(var eventInfo KeyEvent)
   if xOut.value <> "" then    ; If the other field is not empty,
      xOut.blank()             ; make it blank.
  endIf
endmethod
```

xIn action Method

The following code executes when you try to paste a value into the *xIn* field object. Like the code attached to this object's built-in **keyChar** method, this code checks the value of the *xOut* field object. If *xOut* contains a value, this code blanks it out.

```
method action(var eventInfo ActionEvent)
   if eventInfo.id() = EditPaste then
      if xOut.value <> "" then    ; If the other field is not empty,
         xOut.blank()             ; make it blank.
      endIf
   endIf
endmethod
```

Balance Field Object Methods

The *Balance* field object is an unbound field that displays the amount in your account. In Design mode, *Balance* is defined to be a calculated field that displays the sum of the values in the *xAmount* field of the Checks table. The *User's Guide* explains how to define calculated fields.

newValue

The attached code to *Balance's* **newValue** method makes the font color data-dependent: values less than zero display in red characters.

```
method newValue(var eventInfo Event)
   if self.value < 0 then
      self.font.color = Red    ; use red characters to display values < 0
   else
      self.font.color = Black  ; use black characters to display values > 0
   endIf
endmethod
```

The code is attached to **newValue** instead of **changeValue** because *Balance* is a calculated field. A calculated field only displays values, and does not write them to a table. Because Paradox never writes the data from *Balance* to a table, it never calls *Balance*'s **changeValue** method. However, it does call **newValue** each time *Balance* displays a new value.

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

xOut

xIn

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:checks.hlp")
   message("")
endmethod
```

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

CHECKS table frame

Checks Form

CHECKS record object

CHECKS field objects