

The Slot Machine Game (SLOTS.FSL)

Use the Slot Machine game to bet on the chance of dollar signs (\$) appearing after you spin. To play, follow these steps:

1. Choose the amount to bet in the Bet field.
2. Click the Spin button. The display in the top-left corner of the form shows the outcome of the spin.

The Payoff field shows how much money you won on your last spin. You can click the Payoffs button to see how payoffs are calculated.

The Pot field displays the total amount of money you have; it is recalculated after each spin.

You can borrow money when your bet exceeds the amount of money in the pot. When you do not have enough money to cover your bet, the Spin button is replaced with the Borrow button and the Money|Borrow command is enabled. To borrow money:

1. Press the Borrow button or choose the Money|Borrow command. A dialog box appears either giving you a loan or stating that a loan cannot be authorized at this time.
2. If a loan cannot be authorized, choose the Retry button until the loan is made.

Reference

For information on the programming techniques used to create the Slot Machine game, see [Slot Machine Programmer's Help](#).

Slot Machine Programmer's Help

The Slot Machine game uses both TCursors and tables, shows how to build a custom menu, and demonstrates the techniques you can use to develop a full-scale, data entry application. A **TCursor** is a pointer to the data in a table, stored in memory. See the *ObjectPAL Reference* for details.

Before any code was written, two tables and a form were created. This Help system contains a brief description of the procedure used to create the underlying structure of the SLOTS.FSL application.

Reference

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

[Slots Form object](#)

[slotsBox object](#)

[spinButton button](#)

[Pot field object](#)

[payOut object](#)

[HelpButton object](#)

[PayButton object](#)

Underlying Structure of SLOTS.FSL

This game uses two tables, STAKE.DB and SYMBOLS.DB, and one form, SLOTS.FSL. The Stake table has the following structure: Bet, \$; Pot, \$. This table has only one record. After the table was created, the first record was edited to assign the Bet field a value of 5 and the Pot field a value of 100. The value of the Pot field represents the amount of money you have, and the value of the Bet field represents the amount of money you are betting.

The Symbols table has the following structure: Symbol (A1) and SymColor (A12). There is no limit to the number of records this table can contain. The more records used, the longer the wheels in the slot machine spin.

The Symbol field in each record can contain any single character; these characters appear in the windows of the slot machine when you spin it. The dollar signs (\$) in this field are used to signal a payoff. The more dollar signs (\$), the better the odds of winning.

The SymColor field assigns a color to each symbol. Valid color names include Red, Yellow, DarkGreen, and Blue; they are listed online. To display the list, open an ObjectPAL Editor window and choose Language|Constants. Then, from the Types of Constants column, choose Colors. The constants appear in the Constants column.

The form is bound to the Stake table, and also uses a **TCursor** opened on the Symbols table. Menus enable you to borrow money, start a new game, or quit. Since the SpeedBar is not needed in this application, the Form type procedure *hideSpeedBar* is used to hide it.

Custom code prevents you from betting more money than is in the pot and also prevents you from altering the amount in the pot. Code is attached to the following objects:

<u>Slots Form object</u>	the <i>Slots</i> form itself
<u>slotsBox</u>	the box that contains the other objects
<u>spinButton</u>	a button
<u>Pot</u>	a field object
<u>payOut</u>	a field object
<u>HelpButton object</u>	a button
<u>PayButton object</u>	a button

The box that frames the symbols is for cosmetic purposes only. Because the default name is used, referring to it as part of the containership path is unnecessary. For information about default names and the containership path, see the *ObjectPAL Reference*.

Form Methods

Code attached to the *S/ots* 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 following 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 not eventInfo.isPreFilter() then
    if not isFile("slots.fsl") then
      msgInfo("Startup Error!", "The ObjectPAL example files must
              be in the working directory.")
      disableDefault
      close()
    else
      message("Welcome to SLOTS!  Choose Spin to try your luck.")
    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:slots.hlp")
    endif
endif
endmethod
```

***slotsBox* Methods and Procedures**

Three built-in methods affect the behavior of *slotsBox*: **open**, **menuAction** and **close**. In addition, three custom procedures are attached to *slotsBox*: **newGame**, **quitGame**, and **borrow**. All variables for this application are declared in *slotsBox*'s Var window.

All of the code is attached either to *slotsBox*, or to an object that *slotsBox* contains. The application is modular: if you want, you can easily copy *slotsBox* (and the objects it contains), and paste the whole application in a new form. The code runs without modification.

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

[Var window](#)

[open method](#)

[close method](#)

[menu methods and procedures](#)

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

[Slots Form object](#)

[spinButton button](#)

[Pot field object](#)

[payOut object](#)

[HelpButton object](#)

[PayButton object](#)

***slotsBox* Var Window**

Variables declared in *slotsBox*'s Var window (shown in the following code) are available to all objects *slotsBox* contains.

```
var
  theChoice  String      ; holds your menu choice
  goodCredit Logical     ; your credit rating
  symbolsTC  TCursor     ; used to get values from the Symbols table
  p1, p2     PopUpMenu   ; used to build the application menu
  m          Menu        ; the application menu
  nr         LongInt      ; the number of records in the Symbols table
endVar
```


slotsBox open Method

When *slotsBox* opens, the **open** method hides the SpeedBar, opens a **TCursor** onto SYMBOLS.DB, initializes the text boxes, and builds and displays a custom menu.

```
method open(var eventInfo Event)
    hideSpeedBar()

    if not symbolsTC.open("symbols.db") then ; open the table of symbols
        msgStop("Stop", "Couldn't open the Symbols table.")
        return
    endif

    nr = symbolsTC.nRecords() ; do this now, instead of when you spin,
                               ; so you only have to do it once

    edit() ; puts Stake.db (Bet and Pot) into Edit mode

    w1.text = "$" ; display $ in each text box
    w2.text = "$"
    w3.text = "$"

    if Pot < 1 then
        spinButton.labelText = "Borrow"
    else
        spinButton.labelText = "Spin"
    endif

    p1.addText("&New game") ; build a pop-up menu
    p1.addText("&Quit") ; the & specifies a key to press
    m.addPopUp("&Game", p1) ; add the pop-up to the menu

    p2.addText("&Borrow") ; build another pop-up menu
    m.addPopUp("&Money", p2) ; add it to the menu

    m.show() ; display the menu
endmethod
```

***slotsBox* close Method**

The **close** method does some routine housekeeping and restores the SpeedBar.

```
method close(var eventInfo Event)
    removeMenu() ; restore built-in menus
    endEdit()

    errorTrapOnWarnings(True)
    try
        symbolsTC.close()
    onFail
        errorTrapOnWarnings(False)
    endTry

    showSpeedBar()
endmethod
```

***slotsBox* Menu Methods and Procedures**

slotsBox uses the built-in **menuAction** method to handle your menu choices. **menuAction** calls one of the custom procedures **newGame**, **quitGame**, or **borrow** to handle the menu selection.

Select one of the following methods or procedures for a description of the code attached to the *slotsBox* object:

menuAction method

newGame procedure

quitGame procedure

borrow procedure

slotsBox menuAction Method

The **menuAction** method for *slotsBox* uses **eventInfo.menuChoice** and a **switch...endSwitch** block to handle menu choices. When you choose a menu item, one of the three custom procedures **newGame**, **quitGame**, or **borrow** executes.

Ampersands (&) in menu items provide keyboard access to the menu, so they must be included in the text strings for the case statements. For example, &New game lets you press *Alt+N* to choose the item. For more information about working with menus, see the Menu and PopUpMenu sections in the *ObjectPAL Reference*. **menuAction** stores the menu choice, then either lets you choose a menu item or provides the default choice.

```
method menuAction(var eventInfo MenuEvent)
    theChoice = eventInfo.menuChoice() ; store the menu choice
    switch ; execute a custom method based on the menu choice
        case theChoice = "&New game" : newGame()
        case theChoice = "&Quit"      : quitGame()
        case theChoice = "&Borrow"    : borrow()
        otherwise                    : enableDefault
    endSwitch
endmethod
```

Three custom procedures act on menu choices: **newGame**, **quitGame**, and **borrow**. These methods display predefined dialog boxes using methods from the System type (for example, **msgYesNoCancel**).

***slotsBox* newGame Procedure**

newGame resets the amount of the bet and the amount in the pot.

```
proc newGame()  
  ; msgYesNoCancel (System type) displays a pre-defined dialog box  
  if msgYesNoCancel("New game", "Do you want to start a new game?  
The pot is $100.") = "Yes" then  
    pot = 100  
  endif  
endproc
```

***slotsBox* quitGame Procedure**

quitGame closes the form and restores the built-in menus. If you just want to switch to the Design window, press *F8* instead.

```
proc quitGame()  
  if msgYesNoCancel("Quit game", "Do you want to quit?") = "Yes" then  
    removeMenu() ; (System type) restores built-in menus  
    close()      ; close this form  
  endif  
endproc
```

***slotsBox* borrow Procedure**

borrow evaluates your credit rating and might or might not let you borrow money.

```
proc borrow()

    var
        Continue Logical
        tryAgain String
    endvar

    Continue = TRUE

    While Continue

        ; this is a rather arbitrary lending scheme
        if rand() > .4 then
            goodCredit = True
        else
            goodCredit = False
        endif
        ; msgInfo and msgRetryCancel display pre-defined dialog boxes
        if goodCredit = True then
            Continue = False
            msgInfo("Borrow", "Your credit is good.
You may borrow $10.") ; this message spans two lines

            pot = pot + 10

        else
            TryAgain = msgRetryCancel("Borrow", "Sorry. We can't authorize a
loan at this time. Try again later.") ; here, too, we force a line break
            If TryAgain = "Cancel" then
                Continue = False
            Endif
        endif
    endwhile
endproc
```

***spinButton* Methods and Procedures**

Select one of the following methods or procedures for a description of the code attached to the *spinButton* object:

spin procedure

open method

pushButton method

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

Slots Form object

slotsBox object

Pot field object

payOut object

HelpButton object

PayButton object

***spinButton* spin Procedure**

The spin is simulated in the procedure **spin**, which uses the **TCursor** *symbolsTC* to step through the records in the *Symbols* table, displays the value of the Symbol field in a text box, and sets the font color property of each text box.

The slot machine pays only if a dollar sign is displayed in a text box; the more dollar signs displayed, the more money is paid out.

```
var
    i, x SmallInt ; declare variables used by the custom proc spin
endVar

proc spin(uio UIObject)
    x = SmallInt(rand() * nr)+ 1 ; nr = symbolsTC.nRecords(),
                                ; set in slotsBox's open method
    for i from 1 to x           ; move to a record chosen at random
        symbolsTC.nextRecord()
        uio.text = symbolsTC.Symbol ; display the symbol in the text box
        uio.font.color = symbolsTC.SymColor ; set the font color property
    endFor
    symbolsTC.home()           ; return to the first record
endproc
```

***spinButton* open Method**

The **open** method attached to *spinButton* makes the button the active object when you start the application. It calls **moveTo** to move focus to *Self* (the button).

```
method pushButton(var eventInfo Event)
    self.moveTo()
endmethod
```

spinButton pushButton Method

The **pushButton** method attached to *spinButton* simulates the wheels that spin inside a slot machine. If you win, this method also determines how much money you receive.

```
var
    i, x SmallInt ; declare variables used by the custom proc spin
endVar

proc spin(uio UIObject)
    x = SmallInt(rand() * nr) + 1
    ; nr = symbolsTC.nRecords(), set in slotBox's open
    for i from 1 to x ; move to a record chosen at random
        symbolsTC.nextRecord()
        uio.text = symbolsTC.Symbol ; display the symbol in the text box
        uio.font.color = symbolsTC.SymColor ; set the font color
    property
    endFor
    symbolsTC.home() ; return to the first record
endProc

method pushButton(var eventInfo Event)

doDefault
if Self.labelText = "Spin" then ; Don't let players bet more than they
have
    if Pot.value < Bet.value then
        msgStop("Hey!", "You don't have that much money.")
        return
    endif

    pot = pot - bet ; reduce the total money by the amount bet

    spin(w1) ; pass w1 to the spin procedure
    spin(w2)
    spin(w3)

    switch ; determine how much money to pay
        case (w1 = "$" and w2 = "$") and w3 = "$" ; player got three $'s
            : payOut = bet * 25
        case (w1 = "$" and w2 = "$") or (w1 = "$" and w3 = "$") or
            (w2 = "$" and w3 = "$") : payOut = bet * 10
        case w1 = "$" or w2 = "$" or w3 = "$" : payOut = bet * 2 ; player only got one $
        otherwise : payOut = 0
    endSwitch

    pot.value = pot.value + payOut.value ; add payOut to the pot
    Active.moveTo()

else

    Borrow() ; call custom routine

endif
```

```
If pot < 1 then
    Self.labelText = "Borrow"    ; player doesn't have any money
else
    Self.labelText = "Spin"      ; player has money, so allow spin
endif

endmethod
```

Pot Field Object Methods

Two methods are attached to the *Pot* field object: **canArrive** prevents you from altering the amount in the pot, and **changeValue** prevents you from borrowing money you don't need.

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

canArrive method

changeValue method

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

Slots Form object

slotsBox object

spinButton button

payOut object

HelpButton object

PayButton object

Pot canArrive Method

The **canArrive** method uses the Event type method **setErrorCode** to prevent the cursor from moving into the field. You can achieve the same effect by using ObjectPAL or Paradox for Windows interactively to turn off the field object's Tab Stop property. For more information about **setErrorCode** and **canArrive**, see the *ObjectPAL Reference*.

```
method canArrive(var eventInfo MoveEvent)
  if eventInfo.reason() = UserMove then ; if the user tries to move
  to the field
    eventInfo.setErrorCode(CanNotArrive)      ; block the move
  endIf
endmethod
```

Pot changeValue Method

The **changeValue** method sets the display attribute of the Money|Borrow menu item. If the pot has more than \$50, you can't borrow any money, and the item is gray (dimmed). Otherwise, the item is displayed normally, and you can borrow money if your credit is good.

```
method changeValue(var eventInfo ValueEvent)
    if self > 50 then
        setMenuChoiceAttribute("&Borrow", MenuGrayed + MenuDisabled)
    else
        setMenuChoiceAttribute("&Borrow", MenuEnabled)
    endIf
endMethod
```

***payOut* Field Object Methods**

The MoveEvent type method **setErrorCode** in the *payOut* field object's **canArrive** method is used to keep you from moving the cursor into the field object. You can achieve the same effect by using ObjectPAL or Paradox for Windows interactively to turn off the field object's Tab Stop property. For more information about **setErrorCode** and **canArrive**, see *ObjectPAL Reference*.

The *payOut* canArrive method

```
method canArrive(var eventInfo MoveEvent)
if eventInfo.reason() = UserMove then ; if the user tries to move to the
    field
        eventInfo.setErrorCode(CanNotArrive) ; block the move
    endIf
endmethod
```

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

[Slots Form object](#)

[slotsBox object](#)

[spinButton button](#)

[Pot field object](#)

[HelpButton object](#)

[PayButton object](#)

***HelpButton* Methods**

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

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

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

Slots Form object

slotsBox object

spinButton button

Pot field object

payOut object

PayButton object

PayButton Methods

Code attached to the built-in **pushButton** method opens a dialog box that lists the amount you win for each dollar sign.

```
method pushButton(var eventInfo Event)
msgInfo("PayOffs", "You win when a dollar sign ($) appears:
    1 $ = Bet * 2
    2 $ = Bet * 10
    3 $ = Bet * 25")
endmethod
```

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

Slots Form object

slotsBox object

spinButton button

Pot field object

payOut object

HelpButton object

