Más clases en Clipper 5.3

Por David Díaz

Por fin ya tenemos las tan esperadas clases de *CheckBox, ListItem, PushButton, RadioButton, RadioGroup.*

Para explicar el aspecto visual que queda en pantalla podría hacerlo a través de la clase *Get* y sus mandatos, además, esta tarea sería muy fácil, ya que la clase get te lo da todo hecho mediante el código de *GetSys*. De cualquier forma, no voy a explicar la clase Get, pues no es ésa mi intención para este artículo, lo que realmente me interesa, al menos de momento, es explicar las clases citadas anteriormente, con las cuales tendremos la oportunidad de aclarar el funcionamiento de los métodos y la utilización de las variables de instancia; para ello voy a desglosar cada una de estas clases explicando para qué sirven: con ejemplos, comentarios y alguna que otra pantalla.

CheckBox

```
CheckBox   Retorna un objeto de la clase CheckBox
oCheckBox:=  CheckBox( <nRow>,      ;
                       <nColumn>,   ;
                       [,<cCaption>];
                     )
```

Variables de instancia:

Buffer	Valor lógico que indica si ha sido o no chequeado, no se le puede asignar el valor directamente.
CapCol	Indica la columna para la posición del texto introducido en la variable *caption*.
CapRow	Indica la fila para la posición del texto introducido en la variable *caption*.
Caption	Texto descriptivo que aparece al visualizar el objeto *CheckBox*.
Cargo	Esta variable de instancia es ignorada por la clase.
Col	Columna donde irá posicionado los delimitadores de la clase para el chequeo de ésta.
ColorSpec	Color para la clase, irá en forma de cadena de caracteres.
Fblock	CodeBlock que será ejecutado cada vez que gane o pierda el foco.
HasFocus	Valor lógico que indica si está o no activado el foco.
Message	Texto para el mensaje.
Row	Número que indica la fila de los delimitadores.
Sblock	CodeBlock, que se evalúa cada vez que cambie el valor de la variable *buffer*.
Style	Es tipo carácter e indica los delimitadores que serán visualizados.
TypeOut	Siempre contendrá un valor *False*, esta variable no es utilizada por la clase, se ha colocado simplemente para compatibilidad con el *GetSys* que maneja la clase *Get*.

Métodos de la clase:

Display()	Muestra el objeto *ChekcBox* y el texto de *caption*.
KillFocus()	Quita el foco al objeto.
NewState()	Determina cuando *CheckBox* debe de ser chequeado.
SetFocus()	Da el foco al objeto.
Select()	Para activar o desactivar el objeto *CheckBox*.

El método *Select()* al ejecutarse, cambiará el valor de la variable de instancia *buffer* como si fuese un *swich*, es decir, si contiene un *false* le introduce un *true* y al revés, si contiene un *true* le introduce un fa*l*se.

Otra forma de que funcione el método *Select(),* consiste en pasarle un valor lógico controlando en cada momento el valor de *buffer.*

El *CodeBlock* asignado a la variable *sBlock* se ejecuta cuando cambia de valor la variable *buffer* (que no puede ser cambiada directamente), para ello utilizaremos el método *Select()* ya comentado, con o sin parámetro, según lo que queramos hacer, controlar el proceso de cambio de valor de la variable o no.

Si no recibe el foco se comportara igual, pero la opción se verá de forma no resaltada.

Si está seleccionado el objeto *CheckBox* y se le quita el foco, automáticamente se selecciona la opción. Por defecto, las variables de *ColPos* y *RowPos* toman valores para situarse a la izquierda de los delimitadores.

Cuando *buffer* vale *true* aparece el carácter que, por defecto, será el correspondiente código ASCII.

Veamos ahora el fuente 1.

```
// ---Fuente 1 ---------------------------------------------------------

FUNCTION ChkBox()

   LOCAL oChk, lVar
   CLEAR

   oChk := CheckBox( 10, 10, "&Ok" )
   oChk:sBlock := { || MiFunSelect( oChk ) }
   oChk:FBlock := { || MiFunFocus() }

   while .t.

      oChk:Display()
      oChk:Select()
      Inkey(0)

      do case
      case LastKey() == 27
          exit
      case LastKey() == 32
          // De esta forma podremos hacer la prueba de quitar y poner el foco
          if oChk:HasFocus
             oChk:KillFocus()
          else
             oChk:SetFocus()
          endif
      endcase

   enddo

RETURN NIL


// ---------------------------------------------------------------------
FUNCTION MiFunSelect()

   // Contador de cuantas veces se ha variado la variable buffer
   STATIC nCont := 0
```

```
   nCont := nCont + 1
   @ 12, 10 Say "sSelec "
   @ 12, 20 SAY nCont

RETURN NIL

// ------------------------------------------------------------------------


// ------------------------------------------------------------------------

FUNCTION MiFunFocus( )

   // Contador de cuantas veces se ha cambiado de foco
   STATIC nCont := 0
   nCont := nCont + 1
   @ 13, 10 SAY "fFocus "
   @ 13, 20 SAY nCont

RETURN NIL

// ------------------------------------------------------------------------
```

La Función *MiFunFocus()* será ejecutada cada vez que el control gane o pierda el foco. Tiene un contador que se implemeta mediante la variable estática interna *nCont*, de esta manera sabremos cuántas veces se llama a esta función.

La Función *MiFunSelect()* es ejecutada cada vez que se cambia el valor de *buffer*, esto ocurre, cuando se cambia la selección, es decir, cuando se ejecuta *Select()* .

Empezamos mostrando la figura1

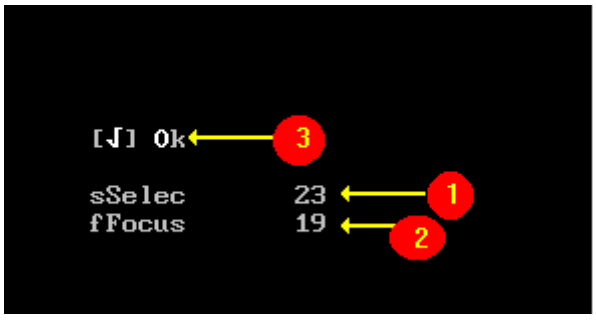

Figura 1: CheckBox y valores de métodos

(1) Valores visualizados por la función *MiFunSelect()* .
(2) Valores visualizados por la función *MiFunFocus()* .
(3) Cuando *buffer* vale *true* y tiene el foco.

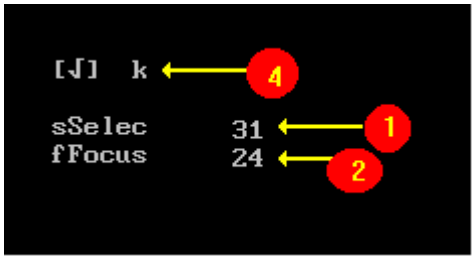

Figura 2: Valores de *buffer*

(4) Cuando *buffer* vale *true* y no tiene el foco.

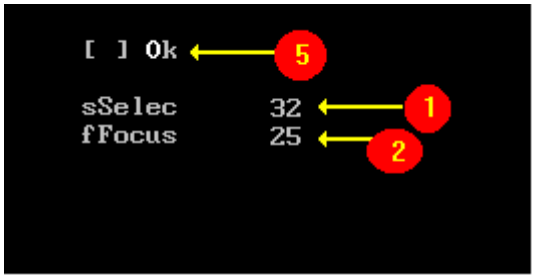

Figura 3: Valores de *buffer*
(5) Cuando *buffer* vale *false* (indistintamente del foco).

ListBox

ListBox Retorna un objeto de la clase *ListBox* .

```
oListBox:=  ListBox( <nTop>, ;
                     <nLeft>, ;
                     <nBottom>, ;
                     <nRight> ;
                     [, <lDropDown>]
                   )
```

Variables de instancia:

Bottom	Fila inferior de la ventana.
Buffer	Valor numérico indicando la posición del ítem activo.
CapCol	Columna del texto descriptivo *caption*.
CapRow	Fila del texto descriptivo *caption*.
Caption	Texto que se visualizar al mostrar el objeto, si no se ha introducido valor en *CapRow* o *CapCol* se mostrará sobre el recuadro que éste dibuja, y si *caption* es omitido lo visualizado será el texto del ítem activo.
Cargo	variable ignorada por la clase.
ColdBox	Caracteres para el recuadro cuando no tiene el foco.
ColorSpec	Caracteres indicando el color.
DropDown	Valor lógico que al ponerlo a *true* provocará que en la visualización coloque un carácter indicando que hay una lista.
Fblock	CodeBlock que se ejecuta cada vez que obtiene o pierde el foco.
HasFocus	Valor lógico que indica si el objeto tiene el foco.
HotBox	Carácter de recuadro que se visualizará al tener el foco.
IsOpen	Valor lógico que indica si esta visible la lista.
ItemCount	Total de ítems que contiene el objeto.
Left	Columna izquierda de la lista.
Message	Texto descriptivo para sacar mensajes.
Right	Columna derecha de la lista.
SBlock	CodeBlock opcional que se ejecuta cuando se cambia de ítem, tanto cuando se realiza con el método *Select()* o con los métodos *PrevItem()* o *NextItem()*.
Top	Fila superior de la lista.
TopItem	Ítem que esta en la fila superior de la lista.
TypeOut	Valor lógico que indica si hay algún ítem en la lista.
vScroll	Objeto *ScrollBar*, será opcional. *ScrollBar* es una barra *gráfica* que indica en que posición nos encontramos.

Métodos de la clase:

AddItem()	Añade nuevos ítems, el ítem tendrá un texto asociado y opcionalmente una expresión.
Close()	Restaura la pantalla que hay debajo de la lista.
DelItem()	Borra un ítem, pasándole por parámetro la posición a borrar.
Display()	Visualiza la lista y el contenido de *caption*.
FindText()	Localiza un texto entre todos los ítems, a este método se le pasará el texto como primer parámetro y un segundo parámetro opcional, para indicar a partir de qué ítem se empieza a buscar.
GetData()	Consigue el dato o expresión del ítem, indicándole la posición de la lista.
GetItem()	Devuelve el ítem, indicándole la posición de la lista.
GetText()	Retorna el texto, y al igual que en *GetItem()* y *GetData().* se tendrá que indicar la posición de la lista, si el valor es mayor que 0 el ratón está fuera de la lista.
HitTest()	Devuelve un valor numérico, indicando la relación entre el ratón y la lista, para ello se le tendrá que pasar la fila y columna donde se encuentra el ratón.
InsItem()	Inserta un nuevo ítem en la posición indicada, y al igual que en *AddItem* se le pasará un texto y una expresión opcional.
KillFocus()	Quita el foco, ejecutando, si hubiese, el CodeBlok de la variable *fBlock*.
NextItem()	Desactiva el ítem actual y activa el siguiente ítem.
Open()	Salva la pantalla donde se va a colocar la lista y la visualiza.
PrevItem()	Desactiva el ítem actual y activa el ítem previo.
Scroll()	Se encarga de hacer el scroll. Para ello se le pasa un parámetro numérico, *nMethod*, que indica la forma de hacer el scroll (no es necesario ejecutar este método).
Select()	Selecciona un ítem, para ello se le pasará como parámetro un valor numérico.
SetData()	Cambia el dato asociado al ítem, para indicar cuál es el ítem se le pasará como segundo parámetro la posición.
SetFocus()	Da el foco al objeto.
SetItem()	Cambia el ítem de la lista, para ello se le pasará un array con dos elementos, el texto y opcionalmente una expresión.
SetText()	Cambia el texto asociado al ítem, pasándole como segundo parámetro la posición del ítem.

Para seleccionar un ítem podemos hacerlo de varias formas, una de ellas es llevando un contador, y con el método *Select()* nos moveremos al sitio correspondiente, (cuando se intenta seleccionar un ítem que no existe no se produce ningún cambio), y la otra forma de seleccionar el ítem es con los métodos *PrevItem()* y *NextItem(),* al usar cualquiera de estos métodos será visualizado automáticamente, no hará falta ejecutar el método *Display().*

Si se le da el foco al objeto *ListBox,* antes de añadir los ítems, dará un error:

```
"Base/1132 Bound error:array access"
```

Para buscar un texto usaremos *FindText(),* en caso de no encontrar nada devolverá un 0. Veamos el fuente 2

```
// --- Fuente 2 ---------------------------------------------------------

#include "inkey.ch"
#include "box.ch"

FUNCTION LstBox()
   LOCAL nKey := 0, cVar := "4"
   LOCAL aLis := { ;
                 { "1", "Uno"   }, { "2", " Dos"  }, { "3", "Tres" },;
                 { "4", "Cuatro"}, { "5", "Cinco" }, { "6", "Seis" },;
```

```
                { "7", "Siete" }, { "8", "Ocho"  }, { "9", "Nueve" };
                }
   LOCAL oListBox, oScr

   CLEAR

   oListBox := ListBox( 10, 10, 18, 20 )
   oScr := ScrollBar( 12, 17, 20 )
   oScr:SBlock := { | | SblockScr() }
   oScr:ColorSpec := "GR+/B"

*  oListBox:caption := "&Capti¢n"

   aEval( aLis, {| elem | oListBox:AddItem( elem[1], elem[2] ) } )

   // cVar ha sido inicializado a 4, por lo tanto se colocara en el ítem 4
   nPos := oListBox:FindText( cVar,,.f. )

   if nPos = 0
      nPos := 1
endif
   oListBox:ColorSpec := ;
            ListbDefColors( "BG/n,GR+/n,BG/n,GR+/R,GR+/n,GR+/n,GR+/n" )
   oListBox:Open()

   // Le damos el foco
   oListBox:SetFocus()

   oListBox:SBlock := { || SblockLst() }
   oListBox:DropDown := .t.
   oListBox:vScroll := oScr
   oListBox:Select( nPos )
   oListBox:Display()


   while nKey != K_ESC .and. nKey != K_ENTER

      nKey := Inkey(0)

      do case
      case nKey == K_UP
          oListBox:PrevItem()
      case nKey == K_DOWN
          oListBox:NextItem()
      case nKey == K_DEL
          oListBox:DelItem( oListBox:Buffer )
          oListBox:Display()
      case nKey == K_INS
          oListBox:InsItem( oListBox:Buffer, "Ins", "Insertado" )
      endcase
   enddo

   oListBox:Close()


RETURN nil


// ---------------------------------------------------------------------------
STATIC FUNCTION SblockLst()
   STATIC nCont := 0
   nCont ++
   @ 22, 5 SAY " Lst nCont " ; ?? nCont

RETURN nil


// ---------------------------------------------------------------------------
STATIC FUNCTION SblockScr()
   STATIC nCont := 0
```

```
   nCont ++
   @ 22, 30 SAY " Scr nCont " ; ?? nCont

RETURN nil
// ---------------------------------------------------------------------------
```

Veamos en la figura 4 cómo queda.

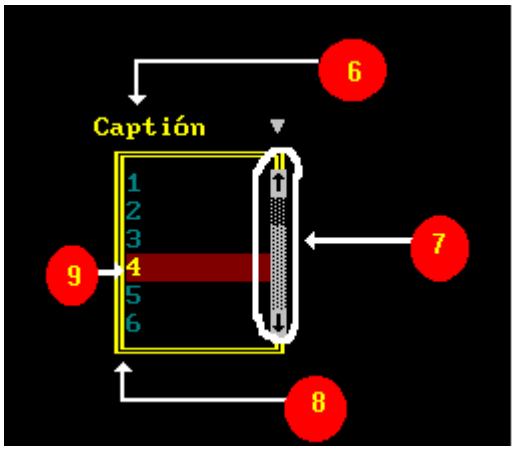

Figura 4: Barra de desplazamiento en ListBox

(6) Texto que se asignó a la variable *Caption.*

(7) Un objeto *ScrollBar*, que fue asignado a la variable de instancia *vScroll*, perteneciente al objeto *oListBox*.

(8) Caracteres de recuadro situados en la variable de instancia *HotBox* .

(9) En este caso el valor de *buffer* es 4.

Se puede no colocar ningún texto en la variable *Caption*, quedando como texto el texto del ítem activo. Esto se observa en el punto (10) de la figura 5

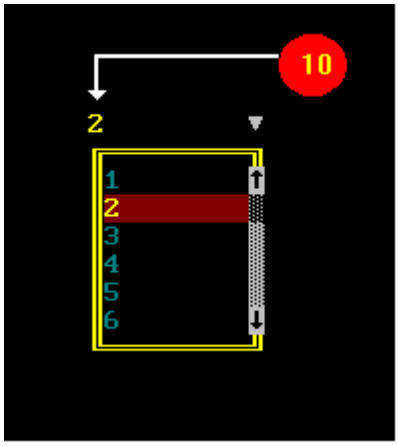

Figura 5: ListBox sobre números

PushButton

PushButton crea un objeto *PushButton*

```
```

```
oPB:= PushButton( <nRow>,     ;
                  <nColumn>,  ;
                  [<cCaption>];
                )
```

Variables de instancia:

Buffer	Valor lógico que indica si el botón ha sido pulsado, esta variable no es asignable.
Caption	Texto descriptivo para el botón.
Cargo	Variable de propósito general que no es usada por la clase.
Col	Columna donde irá visualizado el *PushButton*.
ColorSpec	Cadena de caracteres para colocar el color.
Fblock	En esta variable se podrá colocar un CodeBlock que será ejecutado siempre y cuando se gane o pierda el foco.
HasFocus	Valor lógico que indica si el objeto *PushButton* ha obtenido el foco.
Message	Texto guardado para el mensaje.
Row	Fila donde se visualizará el objeto.
Sblock	CodeBlock opcional que se ejecutará cada vez que se ejecte el método *Select()* del objeto.
Style	Caracteres que indican la forma del botón. Cuando la cadena esté vacía querrá decir que no existirán delimitadores, cuando se introduzcan dos caracteres se hará referencia a los delimitadores de derecha e izquierda, y cuando se coloquen ocho se hará referencia a una caja. El ancho del botón no podrá ser superior a tres líneas.
TypeOut	Valor lógico que nunca se usa, es colocado para compatibilidad con la clase *Get*.

Métodos de la clase:

Display()	Muestra el botón.
HitTest()	Indica si se ha pinchado con el ratón sobre el botón, para ello le pasaremos por parámetro la fila y columna de éste, si el método retorna 0 significa que se ha pulsado fuera del botón.
KillFocus()	Quita el foco al botón.
Select()	Activa el botón, cuando se ejecute este método el objeto revisará si tiene un CodeBlock en *sBlock,* y si lo tiene lo ejecutará.
SetFocus()	Da el foco al botón.

El funcionamiento del método *Select()* es un tanto curioso, ya que si existe un CodeBlock en *sBlock* lo ejecuta siempre, y el valor de *buffer* siempre va a estar a false, un tanto curioso ¡No!. Pues bien, no lo es, ya que si revisamos el valor de *buffer* dentro de la función, podremos comprobar que efectivamente esta variable ha cambiado el valor a *true*.

Si al método *Select()* no se le pasa el parámetro numérico dará sensación de parpadeo muy rápido, revisará *sBlock* y nada más. Si se lo pasamos chequea para ver si la tecla pulsada coincide con el parámetro, de no ser así, el funcionamiento será idéntico, pero de coincidir quitará el vídeo inverso de caption y esperará la pulsación de cualquier otra tecla

Seguro que con un ejemplo nos entendemos mejor,

```
nKey = Inkey(0)
Select( K_ENTER )
```

Si la tecla pulsada fuera la barra espaciadora el efecto sería similar a ejecutar *Select(),* pero si la tecla pulsada fuera [Intro] quitaría el vídeo inverso de *caption* y esperaría a que fuera pulsada otra tecla distinta de [Intro], y así poder evaluar el contenido de *sBlock.*

Como se ha indicado anteriormente, dependiendo del valor que se introduzca en *Style,* la apariencia del boton cambia. En la figura 6 sale con el valor por defecto de la variable *Style.*

Figura 6: Valor por defecto de *Style*

En la figura 7 se ha introducido el valor "[]" en la variable *Style*

Figura 7: Dos corchetes en *Style*

En la figura 8 se ha introducido un valor de recuadro simple, a través de la constante preprocesada "B_SINGLE", a la variable *Style*

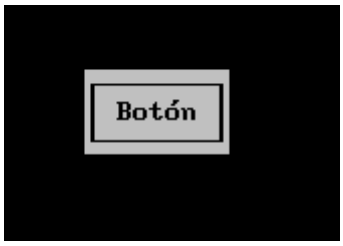

Figura 8: *B_SINGLE* en *Style*

Bien, ahora que ya se han solventado (más o menos) las posibles dudas, veamos un poquito de código. El del fuente 3 hace referencia a la figura 8.

```
// --- Fuente 3 ----------------------------------------------------------

#include "inkey.ch"
#include "box.ch"

FUNCTION PsButton()

   LOCAL nKey := 0, lSalir := .f.
   LOCAL oPsB
   CLEAR

   oPsB := PushButton( 4, 5, " Boton " )
   oPsB:sBlock := { || FsBlock( oPsB ) }
   oPsB:fBlock := { || FfBlock() }
   oPsB:Style := B_SINGLE
```

```
   oPsB:Message := " Mensaje para PushButton "

   oPsB:SetFocus()

   while !lSalir
      oPsB:Display()
      nKey := Inkey( 0 )
      do case
      case nKey = K_ESC
           lSalir := .t.
      case nKey = K_ENTER .or. nKey = K_SPACE
           oPsB:Select( K_ENTER )
      case nKey = K_TAB
           if oPsB:HasFocus
              oPsB:KillFocus()
           else
              PsB:SetFocus()
           endif
      endcase
   enddo

RETURN

/* ------------------------------------------------------------------------*


*-------------------------------------------------------------------------*/

FUNCTION FsBlock( oPsB )

   STATIC nCont := 0
   @ 22, 0 SAY "sBlock "
   @ 22, 10 SAY nCont++

RETURN

/* ------------------------------------------------------------------------*


*-------------------------------------------------------------------------*/

FUNCTION FfBlock()

   STATIC nCont := 0
   @ 21, 0 SAY "fBlock "
   @ 21, 10 SAY nCont++

RETURN

*-------------------------------------------------------------------------*/
```

RadioButton

RadioButto Crea un objeto *RadioButton*

```
oRB :=  RadioButton( <nRow>,     ;
                     <nColumn>,  ;
                     [<cCaption>];
                   )
```

Variables de instancia:

Buffer	Valor lógico que indica si el objeto ha sido seleccionado.
Caption	Caracteres que describen el objeto.
CapCol	Indica la columna de *caption*.
CapRow	Indica la fila de *caption.*
Cargo	Variable de uso general ignorada por la clase.
Col	Columna donde irán posicionados los caracteres de la variable de instancia *Style*..
ColorSpec	Cadena de caracteres para colocarle color.
Fblock	CodeBlock que se ejecuta cada vez que se gana o se pierde el foco.
HasFocus	Valor lógico que indica si tiene el foco.
Row	Fila donde se visualizarán los caracteres de la variable de instancia *Style*..
Sblock	CodeBlock que se ejecuta al ser seleccionado el objeto.
Style	Indica los caracteres usados en *RadioButton*.

Métodos de la clase:

Display()	Visualiza el *RadioButton* y la variable *caption*.
HitTest()	Determina si la fila y columna del ratón están dentro del *RadioButton*.
IsAccel()	Determina si la tecla pasada por parámetro está incluida como tecla aceleradora, para ello se le podrá pasar en forma carácter o como código ASCII.
KillFocus()	Borra el foco del objeto.
Select()	Selecciona el objeto y, de tener un CodeBlock en *sBlock* será ejecutado.
SetFocus()	Da el foco al objeto.

Está clase se utiliza en conjunto con la clase *RadioButton*, también se puede utilizar sola aunque carezca de sentido debido a la existencia de las clases *CheckBox* y *PushButton*. No obstante no es motivo para no enseñarla individualmente.

Podemos ver en la figura 9 un objeto *RadioButton,* con el valor *Buffer* a *falso* y en la 10 con valor *true*.

Figura 9: Objeto *RadioButton* no seleccionado

Figura 10: Objeto *RadioButton* seleccionado

Veamos en el fuente 4 el código necesario para utilizarla.

```
// --- Fuente 4 ----------------------------------------------------------

#include "inkey.ch"

FUNCTION RdButton()
```

```
  LOCAL oRdB, lSalir := .f., nKey := 0
  CLEAR

  oRdB := RadioButto( 5, 10, " &Opcion  " )
  oRdB:SetFocus()

  while !lSalir
     oRdB:Display()
     nKey = Inkey( 0 )
     do case
     case nKey = K_SPACE
         oRdB:Select()
     case nKey = K_ESC
         lSalir := .t.
     endcase
  enddo

RETURN

// ---------------------------------------------------------------------
```

RadioGroup

RadioGroup Retorna un objeto *RadioGroup*.

```
oRG:=  RadioGroup( <nTop>,   ;
                   <nLeft>,  ;
                   <nBottom>,;
                   <nRight>  ;
                 )
```

Variables de instancia:

Bottom	Fila inferior del objeto *RadioButton*.
Buffer	Valor numérico que indica el objeto *RadioButton* seleccionado.
*CapCo*l	Columna de *caption*.
CapRow	Fila de *caption*.
Caption	Texto que describe al objeto *RadioGroup*.
Cargo	Variable de uso general que no es usada por la clase.
ColBox	Columna de la caja.
ColorSpec	Cadena de caracteres que hacen referencia al color.
Fblock	CodeBlock que se ejecutará al ganar o perder el foco.
HasFocus	Valor lógico que indica si el objeto *RadioGroup* tiene el foco.
HotBox	Caracteres de recuadro que serán visualizados cuando el objeto obtenga el foco.
ItemCount	Número de objetos *RadioButton*, es decir, ítems que contiene dicho objeto.
Left	Columna izquierda del recuadro que se visualiza al ejecutar *Display(),*
Message	Texto descriptivo que se utiliza para el mensaje.
Right	Columna derecha del recuadro.
Top	Fila superior del recuadro.
TypeOut	Valor lógico que indica si *RadioGroup* contiene algún objeto *RadioButton*.

métodos de la clase:

AddItem()	Añade nuevos objetos *RadioButton*, para ello solo tendrá que pasarse por parámetro el nuevo objeto.
DelItem()	Borra un objeto *RadioButton*, indicándole la posición del objeto *RadioGroup*.
Display()	Visualiza los ítems *RadioButton*.

GetAccel() Se le pasará una tecla y retornará el primer objeto *RadioButton* que contiene la tecla aceleradora.
GetItem() Obtiene un objeto *RadioButton*, pasándole como parámetro la posición del objeto.
HitTest() Determina si el ratón ha sido pulsado dentro de la región del objeto *RadioGroup*.
InsItem() Inserta un ítem *RadioButton*, indicándole la posición como primer parámetro, y pasándole como segunda opción el objeto.
KillFocus() Quita el foco del objeto.
NextItem() Cambia y selecciona el siguiente objeto *RadioButton*.
PrevItem() Cambia y selecciona el ítem previo.
Select() Selecciona un objeto *RadioButton*, para ello se le pasará por parámetro el número de posición.
SetColor() Cadena de caracteres para colocarle color.
SetFocus() Da el foco al objeto *RadioGroup*.
SetStyle() Determina el *Style* de los objetos *RadioButton*.

Cuando me refiero a los ítems de *RadioGroup* hago mención a los objetos *RadioButton* que se le han ido añadiendo a *RadioGroup*.

Cuando nos movamos sobre un ítem este pasará a estar activo. *RadioGroup* activa y desactiva los ítems de *Radiobutton* automáticamente, con lo cual, a priori, no tiene sentido que nosotros lo controlemos. Por lo tanto el contenido de *sBlock* se ejecutará cada vez que se cambie de ítem.

Con *SetStyle* podremos asegurar, como ya he comentado antes, que todos los objetos tienen los mismos caracteres en *Style*, pero hay que tener cuidado en dos aspectos:

1.- Cambia a la hora de la visualización, y también son cambiados los valores de la variable *Style* de los objetos *RadioButton*.
2.- Sólo cambiará el juego de caracteres de los ítems añadidos, en caso de que los ítems sean incluidos después no serán alterados. En la figura 11 lo podemos ver un conjunto de *RadioButtons*.

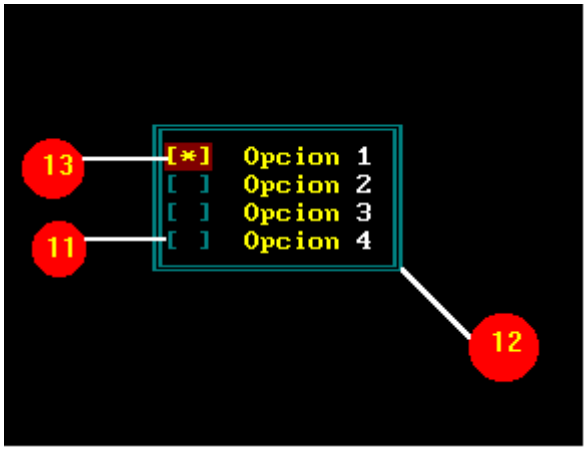

Figura 11: Aspecto de un *RadioGroup*

(11) Un ítem inactivo.
(12) Recuadro correspondiente a los caracteres de recuadro de la variable *HotBox.*
(13) Ítem activo. En este caso el valor de *buffer* será 1.

Veamos, en el fuente 5, qué código hay que trazar para poder manejar está clase

```
// --- Fuente 5 ----------------------------------------------------------

#include "inkey.ch"

FUNCTION RdGroup

   LOCAL oRdB, lSalir := .f., oRdG, nGet
   CLEAR

   oRdG := RadioGroup( 4, 9, 9, 24 )
   oRdG:ColorSpec := "BG/n,GR+/n,BG/n"
   oRdG:Message = " Mensaje para Radio Group "

   oRdB := RadioButto( 5, 10, " Opcion &1 " )
   oRdB:ColorSpec := "BG/n,GR+/n,BG/n,GR+/R,GR+/n,w+/n,w+/n"
   oRdB:fBlock := { || FfBlock( " FBlock -> Opcion 1 " ) }
   oRDB:sBlock := { || SfBlock( " SBlock -> Opcion 1 " ) }
   oRdG:AddItem( oRdB )

   oRdB := RadioButto( 6, 10, " Opcion &2 " )
   oRdB:ColorSpec := "BG/n,GR+/n,BG/n,GR+/R,GR+/n,w+/n,w+/n"
   oRdB:fBlock:=  { || FfBlock( " FBlock -> Opcion 2 " ) }
   oRDB:sBlock := { || SfBlock( " SBlock -> Opcion 2 " ) }
   oRdG:AddItem( oRdB )

   oRdB := RadioButto( 7, 10, " Opcion &3 " )
   oRdB:ColorSpec := "BG/n,GR+/n,BG/n,GR+/R,GR+/n,w+/n,w+/n"
   oRdB:fBlock := { || FfBlock( " FBlock -> Opcion 3 " ) }
   oRDB:sBlock := { || SfBlock( " SBlock -> Opcion 3 " ) }
   oRdG:AddItem( oRdB )

   oRdB := RadioButto( 8, 10, " Opcion &4 " )
   oRdB:ColorSpec := "BG/n,GR+/n,BG/n,GR+/R,GR+/n,w+/n,w+/n"
   oRdB:fBlock := { || FfBlock( " FBlock -> Opcion 4 " ) }
   oRDB:sBlock := { || SfBlock( " SBlock -> Opcion 4 " ) }
   oRdG:AddItem( oRdB )

   // Cambia todos los estilos de los objetos RadioButton introducidos
   // hasta el momento
   oRdG:SetStyle( "[* ]" )
   oRdG:SetFocus()
   ORdG:Select( 1 )
   while !lSalir
      oRdG:Display()
      nKey := Inkey( 0 )
      do case
      case nKey == K_UP
          oRdG:PrevItem()
      case nKey == K_DOWN
          oRdG:NextItem()
      case nKey == K_ESC
          lSalir := .t.
      otherwise
          // Dos formas de ver si hay una tecla aceleradora

          // Recorriendo todos los items
          for nFor := 1 to oRdg:ItemCount
              @ 15 + nFor , 50 SAY oRdg:GetItem( nFor ):IsAccel( Chr( nKey ) )
          next
          // o con el m,todo creado para este fin
          @ 12 , 50 SAY " CON GetAccel " ; ?? oRdG:GetAccel( nKey )
      endcase

   enddo
   oRdG:KillFocus()

RETURN
```

```
FUNCTION FfBlock( cMsg )

   STATIC nCont := 0
   @ 21, 0 SAY Space( 80 )
   @ 21, 6 SAY cMsg
   ?? nCont++

RETURN


FUNCTION SfBlock( cMsg )

   STATIC nCont := 0
   @ 22, 0 SAY Space( 80 )
   @ 22, 6 SAY cMsg
   ?? nCont++

RETURN
```

Scrollbar

ScrollBar Retorna un objeto *ScrollBar*

```
oSBar := ScrollBar(  <nStart>,     ;
                     <nEnd>,       ;
                     <nOffset>,    ;
                     [<bSBlock>]   ;
                     [, <nOrient>] ;
                   )
```

Variables de instancia:

BarLength	Es el número de caracteres que hay en el área o zona existente entre las flechas.
Cargo	Variable de propósito general ignorada por la clase.
ColorSpec	Cadena de caracteres para darle color.
Current	Valor numérico que hace referencia al desplazamiento.
End	Posición en pantalla para último carácter. “Flecha abajo”.
Offset	Indica la columna o fila dependiendo del valor de *Orient*, donde irá situado el objeto *ScrollBar*.
Orient	Indica la orientación de la barra.
Sblock	CodeBlock que es ejecutado cada vez que el estado cambia.
Start	La primera fila o columna donde (dependiendo del valor de *Orient*) se empezará a visualizar el objeto *ScrollBar*.
Style	Caracteres usados para *ScrollBar*.
ThumbPos	La posición que ocupa dentro de la barra del objeto *ScrollBar*.
Total	Valor numérico, en caso de ir a un *ListBox* será el total de ítems que contiene el objeto *ListBox*, por defecto será 100.

Métodos de la clase:

Display()	Visualiza el objeto *ScrollBar*.
Update()	Refresca la posición del carácter que refleja el valor de *ThumbPos*, no visualiza todo el objeto *ScrollBar*.
HitTest()	Determina si el ratón ha sido pulsado dentro de la zona del objeto *ScrollBar*.

Para manejar esta clase, si es utilizada con algún objeto de la clase *ListBox(),* no hay por qué preocuparse, ya que el propio objeto maneja el *Scroll*, pero, si por el contrario, es manejada manualmente, tendremos que tener en cuenta la posición de este *Scroll,* aunque nos podemos desentender de la posición real (*ThumbPos*) de la barra y usar la relativa (*Current*).

Un ejemplo sencillo:

Sea una lista de 40 elementos, el *scroll* con *x* posiciones, la variable de instancia *total* por defecto tiene 100. La posición relativa la obtendríamos con la operación:

```
Current=Contador*(total/elem. a tratar)
```

Si la variable de instancia *Orient* contuviera un 1 la barra se vería vertical y las variables *Start* y *End* harían referencia a las filas, como se ve en la figura 12. Cuando contiene un 2 se ve horizontal y, por tanto, las variables hacen referencia a las columnas.

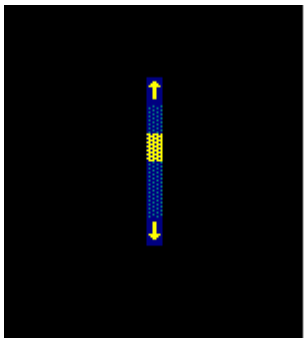

Figura 12: Barra de desplazamiento vertical

Para un mejor entendimiento pasemos al contenido del fuente 6

```
// --- Fuente 6 ----------------------------------------------------------

#include "inkey.ch"
#define  K_MAX 40
#define  K_MIN 0

FUNCTION SclBar()

LOCAL oScr, nCont := 0, nKey := 0

CLEAR

   oScr := ScrollBar( 10, 20, 5 )

   // Orient contiene el dato referente a la orientación
   // con las opciones
   //    1 o SCROLL_VERTICAL
   //    2 o SCROLL_HORIZONTAL
   oScr:Orient := 1
   oScr:offset := 20

   oScr:Display()
   while nKey != K_ESC
```

```
    oScr:UpDate()
    nKey := Inkey( 0 )
    // current contiene la posición relativa dentro de la barra
    oScr:Current :=  nCont * ( 100/( K_MAX - K_MIN ) )

    do case
    case nKey == K_UP
        if nCont > K_MIN
           nCont --
        endif
    case nKey == K_DOWN
        if nCont < K_MAX
           nCont ++
        endif

        Inkey(0)
    endcase

  enddo

RETURN nil
```