

```

////////////////////////////////////
//
// ALERTBOX.PRG
//
// Copyright:
//   Alaska Software, (c) 1997-2001. All rights reserved.
//
// Contents:
//   Function AlertBox()
//
//   This function can be used to replace MsgBox() and ConfirmBox().
//   It works similar to the text mode Alert() function but only for
//   GUI applications. The AlertBox() window is modal.
//
// Syntax:
//   AlertBox( [<oOwner>] , [<cText>] , ;
//             [<aButton>], [<nSysIcon>], ;
//             [<cTitle>]          ) -> nSelectedButton
//
// Parameters:
//   <oOwner> - Is the window on which the AlertBox() is centered.
//             It defaults to SetAppWindow()
//
//   <cText> - Message text like in Alert()
//            Semicolon ; is used for line break
//
//   <aButton> - Array with strings for pushbutton captions
//              It defaults to { "Ok" }
//
//   <nSysIcon> - #define constant XBPSTATIC_SYSICON_* to display an icon
//
//   <cTitle> - Title for Alert box
//
// Return:
//   The function returns the ordinal number of the pressed pushbutton
//   or zero if no selection is made (ESC key or Close window)
//
// Remarks:
//   If no icon is displayed, the message text is centered in the alert
//   box. If an icon is displayed, the text is left aligned next to the
//   icon. Pushbuttons are centered below the message text.
//
////////////////////////////////////

#include "Appevent.ch"
#include "Common.ch"
#include "Font.ch"
#include "Gra.ch"
#include "Xbp.ch"

/*
 * Max. number of pushbuttons displayed in the Alert box
 */
#define MAX_BUTTONS      4

```

```
#define BUTTON_FONT    FONT_HELV_SMALL
#define TEXT_FONT      FONT_TIMES_MEDIUM+FONT_STYLE_BOLD
#define TEXT_COLOR     GRA_CLR_DARKBLUE
```

```
*****
PROCEDURE X_ALERT
*****
```

RETURN

```
/*
 * This function works like the text mode Alert() box but for GUI applications.
 * It displays centered on the specified window.
 */
```

```
FUNCTION AlertBox( oOwner, cText, aCaption, nSysIcon, cTitle )
  LOCAL nEvent , mp1 , mp2, oXbp , lExit
  LOCAL oParent , oDlg, oPS, oFont, oFocus, oDesktop
  LOCAL aPushBtn , aPbSize, nPbWidth, nPbDist, nSelect
  LOCAL oIcon , aIconSize
  LOCAL aTextSize, aText
  LOCAL bSelect , bKeyboard
  LOCAL i, imax , nPos , aPos , aPos1 , aSize , nXsize, nYsize
```

```
*****
PUBLIC oDlg_alert, nSelect_alert
*****
```

```
DEFAULT oOwner TO SetAppWindow(), ;
  cText TO "" ;
  aCaption TO { "~Ok" } , ;
  cTitle TO ""
```

```
oDeskTop := AppDesktop()
```

```
/*
 * Create the Alert dialog window
 */
oDlg_alert := XbpDialog():new( oDesktop, oOwner, , { 100, 100 }, , .F. )
oDlg_alert:minButton := .F.
oDlg_alert:maxButton := .F.
oDlg_alert:title := cTitle
oDlg_alert:border := XBPDLG_DLGBORDER
oDlg_alert:close := { |mp1,mp2,obj| obj:hide(), lExit := .T. }
oDlg_alert:create()
oDlg_alert:drawingArea:SetFontCompoundName( BUTTON_FONT )
oDlg_alert:drawingArea:paint := { || Repaint( aText ) }
```

```
/*
 * Create icon if requested
 */
```

```

IF nSysIcon <> NIL
    oIcon      := XbpStatic():new( oDlg_alert:drawingArea, , {0,0}, {32,32} )
    oIcon:caption := nSysIcon
    oIcon:type    := XBPSTATIC_TYPE_ICON
    oIcon:create()
    aIconSize    := oIcon:currentSize()
ELSE
    aIconSize    := {0,0}
ENDIF

/*
* Create a presentation space with font to calculate pushbutton
* size from font metrics
*/
oPS  := XbpPresSpace():new():create( oDlg_alert:drawingArea:winDevice() )
oFont := XbpFont():new( oPS ):create( BUTTON_FONT )
oPS:setFont( oFont )

/*
* Find longest button caption
*/
imax := Min( MAX_BUTTONS, Len( aCaption ) )
npos := 0
nXsize := 0

FOR i:=1 TO imax
    IF Len( aCaption[i] ) > nXsize
        nXsize := Len( aCaption[i] )
        nPos   := i
    ENDIF
NEXT

/*
* Determine size for pushbuttons
*/
aPbSize := GraQueryTextBox( oPS, aCaption[nPos] )
aPbSize := { aPbSize[3,1] - aPbSize[2,1] ;
             , aPbSize[3,2] - aPbSize[2,2] }
aPbSize[1] *= 1.5
aPbSize[2] *= 2
nPbDist := aPbSize[1] / nXsize
nPbWidth := imax * ( aPbSize[1] + nPbDist )

/*
* Prepare the string for display and determine display size
*/
oFont := XbpFont():new( oPS ):create( TEXT_FONT )
oPS:setFont( oFont )
aTextSize := PrepareText( oPS, @aText, cText )

/*
* Calculate frame size for the Alert box (XbpDialog window)

```

```

*/
nXsize := Max( nPbWidth + 20, aTextSize[1] + 20 ) + aIconSize[1] * 1.5
nYsize := aTextSize[2] + 2 * aPbSize[2] + 10
aSize := oDlg_alert:calcFrameRect( { 0, 0, nXsize, nYsize } )

oDlg_alert:setSize( { aSize[3], aSize[4] } )
MoveToOwner( oDlg_alert )
aSize := oDlg_alert:drawingArea:currentSize()

IF nSysIcon == NIL
  /*
  * No icon - Center text in window
  */
  aPos := CenterPos( aTextSize, aSize )
ELSE
  /*
  * With icon - Align text next to icon
  */
  aPos := { 0, 0 }
  aPos[1] := 1.75 * aIconSize[1]
ENDIF
aPos[2] := 2.25 * aPbSize[2]
AdjustPos( aPos, aTextSize, aText, nSysIcon )

/*
* Position the icon to the upper left of the text
*/
IF oIcon <> NIL
  aPos[1] := aIconSize[1] / 3
  aPos[2] := aSize[2] - 1.5 * aIconSize[2]
  oIcon:setPos( aPos )
ENDIF

IF nSysIcon == NIL
  /*
  * No icon - Center pushbutton in window
  */
  aPos := CenterPos( {nPbWidth, 2 * aPbSize[2]}, aSize )
  aPos[1] += ( nPbDist / 2 )

ELSEIF nPbWidth < aTextSize[1]
  /*
  * Width of all pushbuttons is less than text width.
  * Center pushbutton below text
  */
  aPos := CenterPos( {nPbWidth, 2 * aPbSize[2]}, aTextSize )
  aPos[1] += ( 1.5 * aIconSize[1] + ( nPbDist / 2 ) )

ELSE
  /*
  * Align pushbuttons with text
  */
  aPos[1] := 1.75 * aIconSize[1]
ENDIF

```

```

aPushBtn := Array( imax )
aPos[2] := aPbSize[2] / 2
nSelect_alert := 0
bSelect := { |mp1, mp2, obj| nSelect_alert := obj:cargo, lExit := .T. }
bKeyBoard := { |nKey, mp2, obj| KeyHandler( nKey, obj, aPushBtn ) }

/*
* Create the pushbuttons
*/
FOR i:=1 TO imax
  aPushBtn[i] := XbpPushButton():new( oDlg_alert:drawingArea,, aPos, aPbSize )
  aPushBtn[i]:caption := aCaption[i]
  aPushBtn[i]:cargo := i
  aPushBtn[i]:tabstop := .T.
  aPushBtn[i]:activate := bSelect
  aPushBtn[i]:keyboard := bKeyboard

  IF i == 1
    aPushBtn[i]:group := XBP_BEGIN_GROUP
  ELSEIF i == imax
    aPushBtn[i]:group := XBP_END_GROUP
  ELSE
    aPushBtn[i]:group := XBP_WITHIN_GROUP
  ENDIF
ENDIF

  aPushBtn[i]:create()
  aPos[1] += ( aPbSize[1] + nPbDist )
NEXT

oDlg_alert:setModalState( XBP_DISP_APPMODAL )
oDlg_alert:show()
oFocus := SetAppFocus( aPushBtn[1] )

lExit := .F.
DO WHILE ! lExit
  nEvent := AppEvent( @mp1, @mp2, @oXbp )
  oXbp:handleEvent( nEvent, mp1, mp2 )
ENDDO

/*
* Cleanup: Reset modality, release graphical segment,
* dialog and set focus back.
*/
oDlg_alert:setModalState( XBP_DISP_MODELESS )
GraSegDestroy( oPS, aText[2] )
oDlg_alert:destroy()
SetAppFocus( oFocus )

RETURN nSelect_alert

```

```

/*
 * Key handling for pushbuttons in alert box
 * The array containing the pushbuttons is not used here
 * (but may be useful for others...)
 */
STATIC PROCEDURE KeyHandler( nKey, oButton, aPushButtons )

DO CASE
CASE nKey == xbeK_ESC
    PostAppEvent( xbeP_Close,,, oButton:setParent():setParent() )
CASE nKey == xbeK_RETURN
    PostAppEvent( xbeP_Activate,,, oButton )
ENDCASE

RETURN

/*
 * Calculate entire display area for text to be displayed
 */
STATIC FUNCTION PrepareText( oPS, aText, cText )
    LOCAL aAttr, aTextBox, nMaxWidth, nMaxHeight, cLine, aLine

    aText      := {}
    nMaxWidth  := 0
    nMaxHeight := 0
    aAttr      := Array( GRA_AS_COUNT )

    aAttr[ GRA_AS_COLOR ] := GRA_CLR_BLUE
    oPS:setAttrString( aAttr )

    DO WHILE ! Empty( cText )
        cLine := CutStr( ",", @cText )
        IF Empty( cLine )
            cLine := " "
        ENDIF
        aLine := { {0,0}, cLine, 0, 0 }
        aTextBox := GraQueryTextBox( oPS, aLine[2] )
        aLine[3] := aTextBox[3,1] - aTextBox[2,1] // width of substring
        aLine[4] := aTextBox[3,2] - aTextBox[2,2] // height of substring
        nMaxWidth := Max( nMaxWidth, aLine[3] )
        nMaxHeight += ( 2 + aLine[4] )
        AAdd( aText, aLine )
    ENDDO

    aText := { oPS, 0, aText, aAttr }

RETURN { nMaxWidth, nMaxheight }

```

```

/*
 * Adjust GraStringAt() positions for centered display or left alignment
 */
STATIC PROCEDURE AdjustPos( aPos, aSize, aText, nSysIcon )
  LOCAL aLines := aText[3]
  LOCAL i, imax := Len( aLines )

  IF nSysIcon == NIL
    /*
     * No Icon - Display text centered
     */
    FOR i := 1 TO imax
      aLines[i,1,1] := aPos[1] + ( (aSize[1] - aLines[i,3]) / 2 )
      aLines[i,1,2] := aPos[2] + aSize[2] - i * ( 2 + aLines[i,4] )
    NEXT
  ELSE
    /*
     * With Icon - Display text left aligned
     */
    FOR i := 1 TO imax
      aLines[i,1,1] := aPos[1]
      aLines[i,1,2] := aPos[2] + aSize[2] - i * ( 2 + aLines[i,4] )
    NEXT
  ENDIF
RETURN

```

```

/*
 * Repaint text using a graphical segment
 */
STATIC PROCEDURE Repaint( aText )
  LOCAL oPS := aText[1]
  LOCAL nID := aText[2]
  LOCAL aLines := aText[3]
  LOCAL aAttr := aText[4]

  oPS:setAttrString( aAttr )

  IF nID == 0
    nID := GraSegOpen( oPS )
    AEval( aLines, { |a| GraStringAt( oPS, a[1], a[2] ) } )
    GraSegClose( oPS )
    aText[2] := nID
  ENDIF

  GraSegDraw( oPS, nID )

RETURN

```

```

/*
 * Move a window within its parent according to the origin of
 * its owner window. Default position is centered on the owner.
 */

```

```

STATIC PROCEDURE MoveToOwner( oDlg_alert, aPos )
  LOCAL oOwner := oDlg_alert:setOwner()
  LOCAL oParent := oDlg_alert:setParent()
  LOCAL aPos1, nWidth

  DEFAULT aPos TO CenterPos( oDlg_alert:currentSize(), oOwner:currentSize() )

  DO WHILE oOwner <> oParent
    aPos1 := oOwner:currentPos()

    aPos[1] += aPos1[1]
    aPos[2] += aPos1[2]

    IF oOwner:isDerivedFrom( "XbpDialog" )
      /*
      * Adjust for thickness of the owner window's frame
      */
      nWidth := ( oOwner:currentSize()[1] - oOwner:drawingArea:currentSize()[1] ) / 4
      aPos[1] += nWidth
      aPos[2] += nWidth
    ENDIF
    oOwner := oOwner:setParent()
  ENDDO

  oDlg_alert:setPos( aPos )
RETURN

/*
* Calculate the center position from size and reference size
*/
STATIC FUNCTION CenterPos( aSize, aRefSize )
RETURN { Int( (aRefSize[1] - aSize[1]) / 2 ) ;
        , Int( (aRefSize[2] - aSize[2]) / 2 ) }

/*
* Cut a string at a given delimiter and remove delimiter from string
*/
STATIC FUNCTION CutStr( cCut, cString )
  LOCAL cLeftPart, i := At( cCut, cString )

  IF i > 0
    cLeftPart := Left( cString, i-1 )
    cString := SubStr( cString, i+Len(cCut) )
  ELSE
    cLeftPart := cString
    cString := ""
  ENDIF

RETURN cLeftPart

```