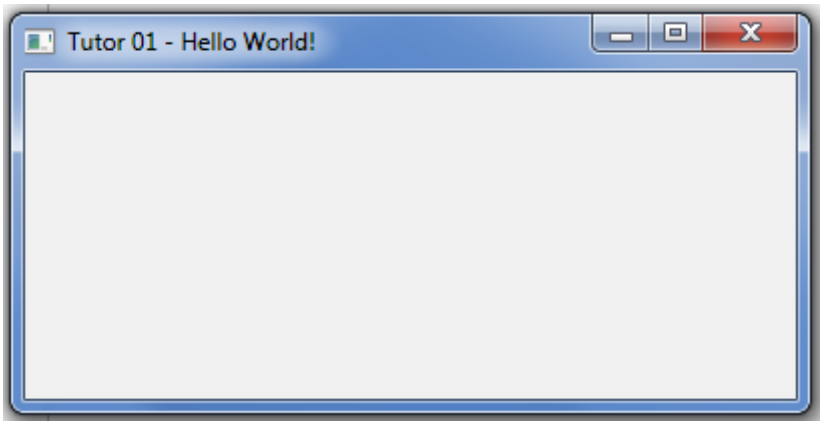
HMG TUTORIAL

Sample Files: \hmg\samples\tutorial ***version 3.0.28***

Your First HMG Program (tut1)

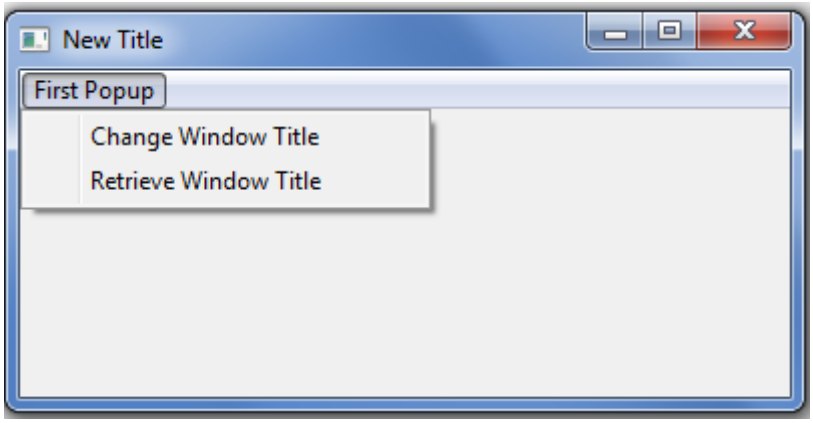


I'll not be original, so this program will display a 'Hello World' message :)

```
#include "hmg.ch"

Function Main

   DEFINE WINDOW Win_1 ;
       AT 0,0 ;
       WIDTH 400 ;
       HEIGHT 200 ;
       TITLE 'Tutor 01 - Hello World!' ;
       MAIN

   END WINDOW

   ACTIVATE WINDOW Win_1

Return
```

- DEFINE WINDOW command: Will create the main window for the program.
- Win_1: Is the name of the window.
- AT 0,0: Indicates the window position (row=0,col=0)
- WIDTH 400: Means that the window will have 400 pixels width.
- HEIGHT 200: Means that the window will have 200 pixels height.
- TITLE 'Hello World!': Indicates the text in the window title bar.
- MAIN: Indicates that we are defining the main application window (a main window is required for all HMG applications)
- ACTIVATE WINDOW Form_1: Will show the window and start the event loop.

Adding The Main Menu (tut2)

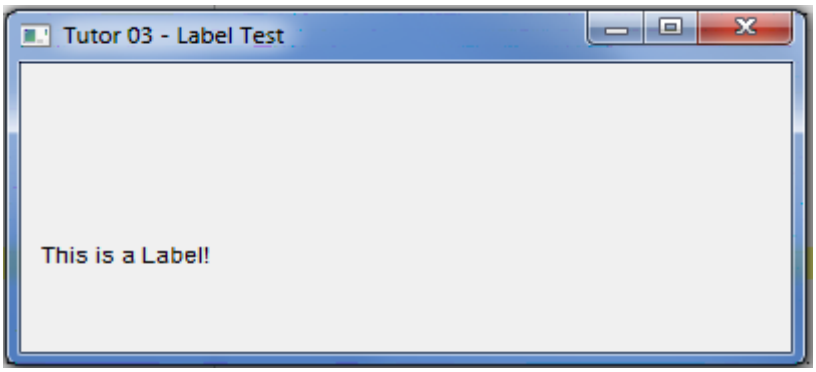


We add a main menu to the program now:

```
#include "hmg.ch"

Function Main

   DEFINE WINDOW Win_1 ;
       AT 0,0 ;
       WIDTH 400 ;
       HEIGHT 200 ;
       TITLE 'Tutor 02 - Property Test' ;
       MAIN
       DEFINE MAIN MENU
          POPUP "First Popup"
            ITEM 'Change Window Title' ACTION Win_1.Title := 'New Title'
            ITEM 'Retrieve Window Title' ACTION MsgInfo ( Win_1.Title )
          END POPUP
       END MENU
   END WINDOW
   ACTIVATE WINDOW Win_1
Return
```

As you can see it is pretty easy and intuitive.
All the main menu stuff will be between DEFINE MAIN MENU / END MENU statements.
Each individual menu popup will be between POPUP / END POPUP statements.
Each individual menu item will be coded via the ITEM statement.
You can use as popups as you need and you can nest it without any limit.

Adding a Label (tut3)



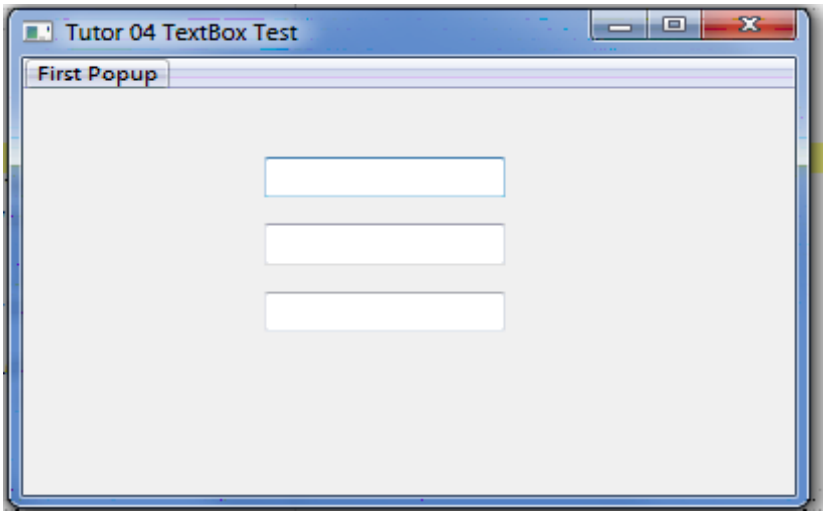
The label control allows you to show text and it is very easy to use too.

```
   @ 100,10 LABEL Label_1 VALUE 'This is a Label!'
```

@ 100,10 means that the text will be showed at row 100 , column 10 (remember that the measurement unit is the pixel)
Label_1 is the name of the control (we will identificate by this name)
VALUE clause indicates the initial value of the control when created.

```
#include "hmg.ch"

Function Main

   DEFINE WINDOW Win_1 ;
       AT 0,0 ;
       WIDTH 400 ;
       HEIGHT 200 ;
       TITLE 'Tutor 03 - Label Test' ;
       MAIN
       @ 100,10 LABEL Label_1 VALUE 'This is a Label!'
   END WINDOW

   ACTIVATE WINDOW Win_1

Return
```

Getting data from the user (The TextBox Control) (tut4)

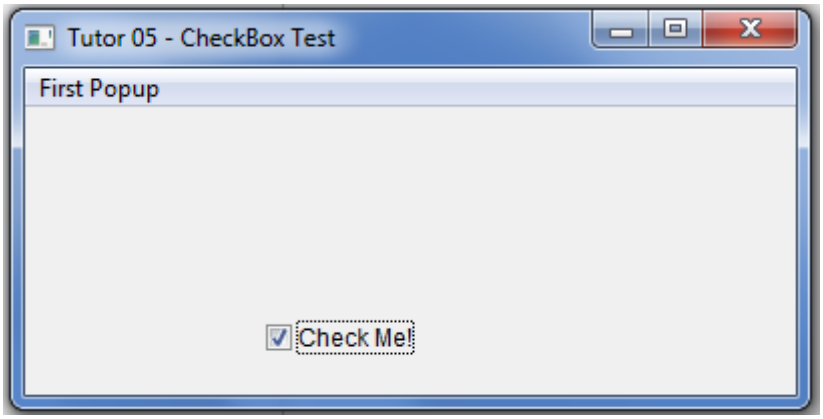


The TextBox control is the main way to obtain data from the user.

```
#include "hmg.ch"

Function Main

   DEFINE WINDOW Win_1 ;
       AT 0,0 ;
       WIDTH 400 ;
       HEIGHT 300 ;
       TITLE 'Tutor 04 TextBox Test' ;
       MAIN
       DEFINE MAIN MENU
          POPUP "First Popup"
```

```
            ITEM 'Change TextBox Content' ACTION Win_1.Text_1.Value := 'New TextBox
           Value'
            ITEM 'Retrieve TextBox Content' ACTION MsgInfo ( Win_1.Text_1.Value)
            SEPARATOR
            ITEM 'Change Numeric TextBox Content' ACTION Win_1.Text_2.Value := 100
            ITEM 'Retrieve Numeric TextBox Content' ACTION MsgInfo (
           Str(Win_1.Text_2.Value))
            SEPARATOR
            ITEM 'Change Numeric (InputMask) TextBox Content'
             ACTION Win_1.Text_3.Value := 1234.12
            ITEM 'Retrieve Numeric (InputMask) TextBox Content' ACTION MsgInfo (
           Str(Win_1.Text_3.Value))

         END POPUP
      END MENU
      @ 40 , 120 TEXTBOX Text_1
      @ 80 , 120 TEXTBOX Text_2 NUMERIC
      @ 120 , 120 TEXTBOX Text_3 NUMERIC INPUTMASK '9999.99'
   END WINDOW

   ACTIVATE WINDOW Win_1

Return
```

Getting Logical (tut5)



Sometimes, you need to get logical data. from the user. The easiest way to do that, is using a checkbox control.

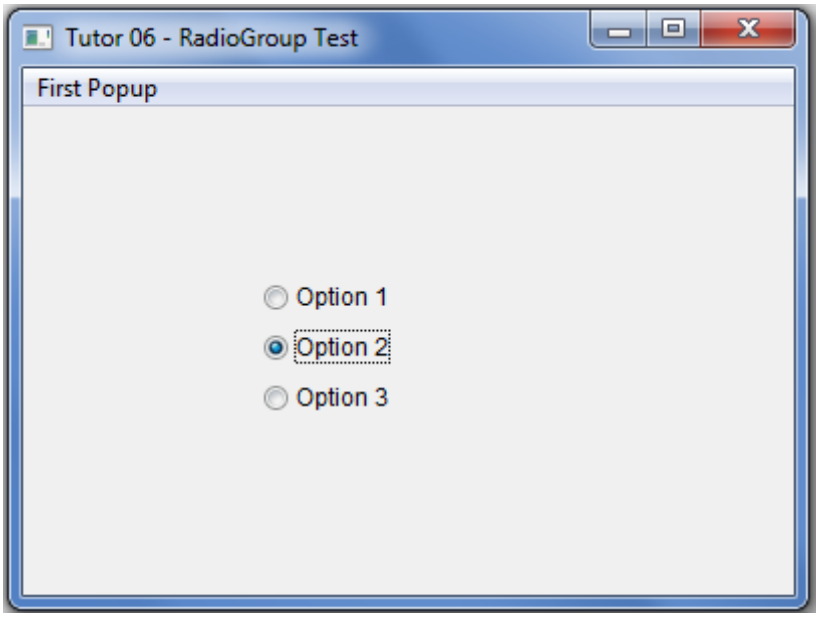
```
@ 180, 120 CHECKBOX Check_1
```

We add it to the program, along with new menu options to set or retrieve its value

```
#include "hmg.ch"

Function Main

   DEFINE WINDOW Win_1 ;
      AT 0,0 ;
      WIDTH 400 ;
      HEIGHT 300 ;
      TITLE 'Tutor 05 - CheckBox Test' ;
      MAIN
      DEFINE MAIN MENU
         POPUP "First Popup"
           ITEM 'Change CheckBox Value' ACTION Win_1.Check_1.Value := .T.
```

```
            ITEM 'Retrieve CheckBox Value' ACTION MsgInfo ( if(Win_1.Check_1.
            Value,'.T.','.F.'))

         END POPUP
      END MENU
          @ 100, 120 CHECKBOX Check_1 CAPTION 'Check Me!'
   END WINDOW

   ACTIVATE WINDOW Win_1

Return
```

Making a Choice (tut6)

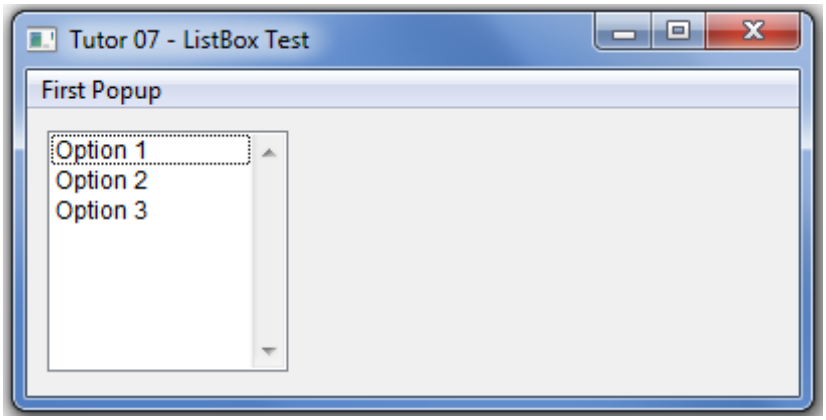


Sometimes, you need to get a choice from users, between a small group of options that are known at design time.
The best way to go is such cases is the RadioGroup control.

```
#include "hmg.ch"

Function Main

   DEFINE WINDOW Win_1 ;
      AT 0,0 ;
      WIDTH 400 ;
      HEIGHT 200 ;
      TITLE 'Tutor 06 - RadioGroup Test' ;
      MAIN
      DEFINE MAIN MENU
         POPUP "First Popup"
           ITEM 'Change RadioGroup Value' ACTION Win_1.Radio_1.Value := 2
           ITEM 'Retrieve RadioGroup Value' ACTION MsgInfo ( Str(Win_1.Ra
           dio_1.Value))

         END POPUP
      END MENU
          @ 80, 120 RADIOGROUP Radio_1 OPTIONS {'Option 1','Option 2','Option 3'}
```

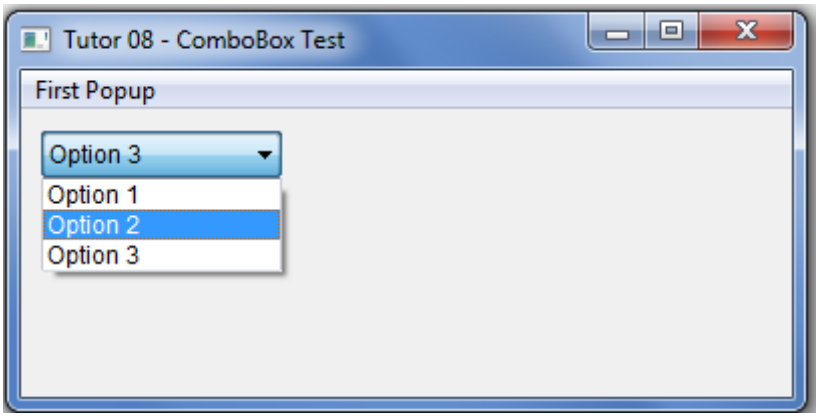
```
   END WINDOW

   ACTIVATE WINDOW Win_1

Return
```

More Choices (tut7)




```
There is various alternatives to get an user's choice besides RadioGroup.
One of them is the ListBox Control
@ 10, 10 LISTBOX List_1 ITEMS {'Option 1','Option 2','Option 3'}
Using a Listbox, you can add, change or remove items at runtime.

#include "hmg.ch"

Function Main

   DEFINE WINDOW Win_1 ;
       AT 0,0 ;
       WIDTH 400 ;
       HEIGHT 200 ;
       TITLE 'Tutor 07 - ListBox Test' ;
       MAIN
       DEFINE MAIN MENU
          POPUP "First Popup"
            ITEM 'Change ListBox Value' ACTION Win_1.List_1.Value := 2
            ITEM 'Retrieve ListBox Value' ACTION MsgInfo ( Str(Win_1.List_1.Value))
            SEPARATOR
            ITEM 'Add List Item' ACTION Win_1.List_1.AddItem ('New List Item')
            ITEM 'Remove List Item' ACTION Win_1.List_1.DeleteItem (2)
            ITEM 'Change List Item' ACTION Win_1.List_1.Item (1) := 'New Item Text'
            ITEM 'Get List Item Count' ACTION MsgInfo (Str(Win_1.List_1.ItemCount))
          END POPUP
       END MENU
           @ 10, 10 LISTBOX List_1 ITEMS {'Option 1','Option 2','Option 3'}
   END WINDOW

   ACTIVATE WINDOW Win_1

Return
```

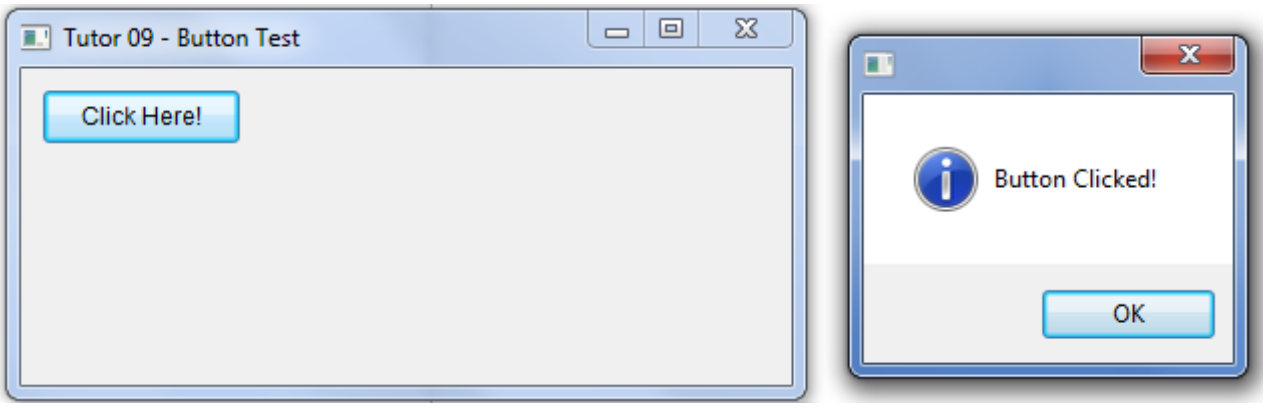
More Choices II (tut8)



Another alternative to get an user's choice is the COMBOBOX.
Using a Combobox, is similar to ListBox.

```
#include "hmg.ch"

Function Main

   DEFINE WINDOW Win_1 ;
       AT 0,0 ;
       WIDTH 400 ;
       HEIGHT 200 ;
       TITLE 'Tutor 08 - ComboBox Test' ;
       MAIN
       DEFINE MAIN MENU
          POPUP "First Popup"
            ITEM 'Change ComboBox Value' ACTION Win_1.Combo_1.Value := 2
            ITEM 'Retrieve ComboBox Value' ACTION MsgInfo ( Str(Win_1.Combo_1.Val
                 ue))
            SEPARATOR
            ITEM 'Add Combo Item' ACTION Win_1.Combo_1.AddItem ('New List Item')
            ITEM 'Remove Combo Item' ACTION Win_1.Combo_1.DeleteItem (2)
            ITEM 'Change Combo Item' ACTION Win_1.Combo_1.Item (1) := 'New Item
           Text'
            ITEM 'Get Combo Item Count' ACTION MsgInfo (Str(Win_1.Combo_1.Item
           Count))
          END POPUP
       END MENU
           @ 10, 10 COMBOBOX Combo_1 ITEMS {'Option 1','Option 2','Option 3'}
   END WINDOW

   ACTIVATE WINDOW Win_1
Return
```

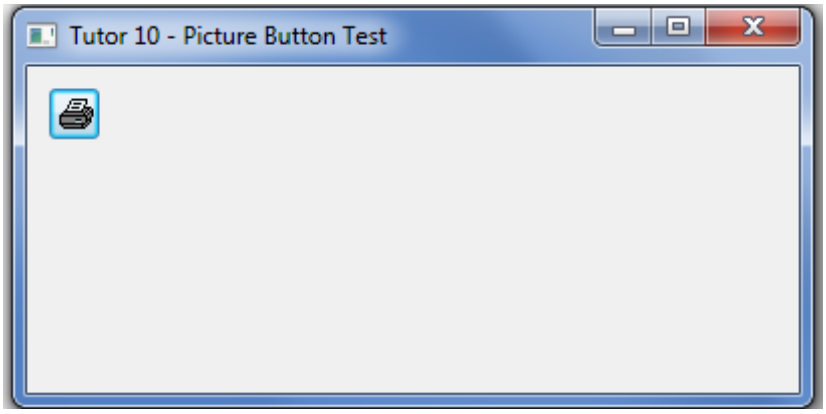
Pushing Actions (Standard Buttons) (tut9)



Another way for let the users to take an action (besides menus) are buttons.

```
#include "hmg.ch"

Function Main

   DEFINE WINDOW Win_1 ;
       AT 0,0 ;
       WIDTH 400 ;
       HEIGHT 200 ;
       TITLE 'Tutor 09 - Button Test' ;
       MAIN
       @ 10,10 BUTTON Button_1 ;
           CAPTION 'Click Here!' ;
                   ACTION MsgInfo('Button Clicked!')
   END WINDOW

   ACTIVATE WINDOW Win_1

Return
```

Being More Graphical (Picture Buttons) (tut10)

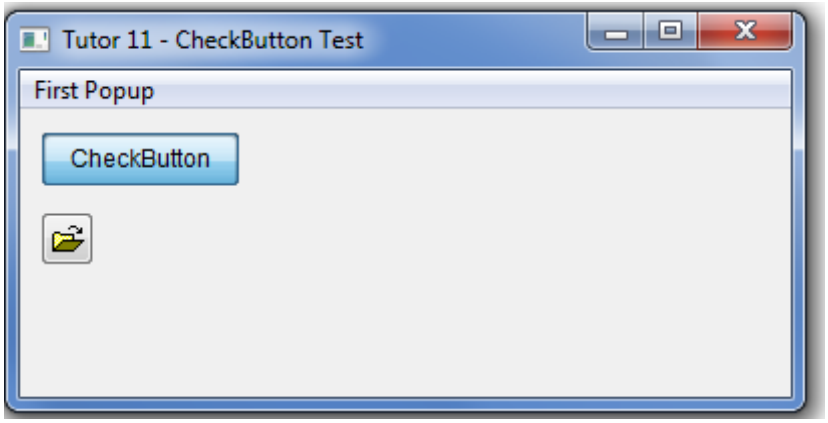


Instead a text caption you can use a picture.

```
#include "hmg.ch"

Function Main

   DEFINE WINDOW Win_1 ;
       AT 0,0 ;
       WIDTH 400 ;
       HEIGHT 200 ;
       TITLE 'Tutor 10 - Picture Button Test' ;
       MAIN
           @ 10,10 BUTTON PictureButton_1 ;
               PICTURE 'button.bmp' ;
               ACTION MsgInfo('Picture Button Clicked!!') ;
               WIDTH 27 ;
               HEIGHT 27 ;
               TOOLTIP 'Picture Button Tooltip'
   END WINDOW

   ACTIVATE WINDOW Win_1

Return
```

The optional tooltip clause, causes that a small window with an explanatory text be displayed when the mouse pointer stays over the control for a few seconds. You can use this clause with most MiniGUI controls.

Button + CheckBox = CheckButton (tut11)

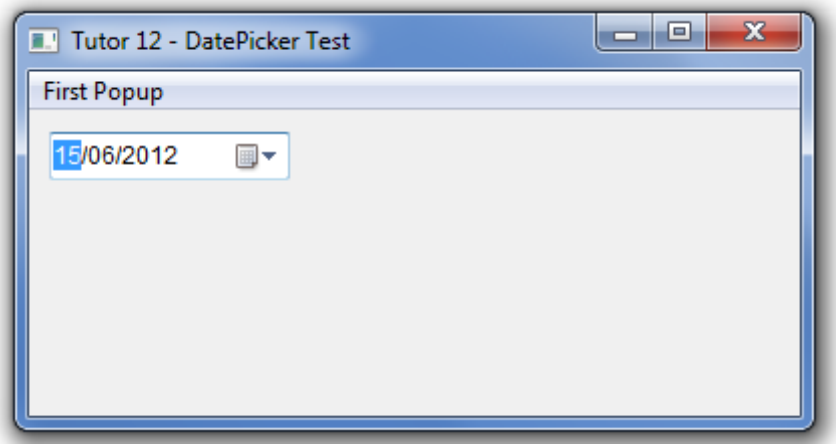


The CheckButton control, acts like a checkbox, but looks like a button. Like buttons, It comes in two flavors: Text and Graphical.

```
#include "hmg.ch"

Function Main

   DEFINE WINDOW Win_1 ;
       AT 0,0 ;
       WIDTH 400 ;
       HEIGHT 200 ;
       TITLE 'Tutor 11 - CheckButton Test' ;
       MAIN
       DEFINE MAIN MENU
          POPUP "First Popup"
            ITEM 'Change Text CheckButton Value' ACTION Win_1.CheckButton_1.Value
```

```
            := .T.
            ITEM 'Retrieve Text CheckButton Value' ACTION MsgInfo ( if(Win_1.Check
            Button_1.Value,'.T.','.F.'))
            SEPARATOR
            ITEM 'Change Picture CheckButton Value' ACTION Win_1.CheckButton_2.
            Value := .T.
            ITEM 'Retrieve Picture CheckButton Value' ACTION MsgInfo (
            if(Win_1.CheckButton_2.Value,'.T.','.F.'))
         END POPUP
      END MENU
          @ 10,10 CHECKBUTTON CheckButton_1 ;
              CAPTION 'CheckButton' ;
              VALUE .F.

          @ 50,10 CHECKBUTTON CheckButton_2 ;
              PICTURE 'Open.Bmp' ;
              WIDTH 27 ;
              HEIGHT 27 ;
              VALUE .F. ;
              TOOLTIP 'Graphical CheckButton'

   END WINDOW

   ACTIVATE WINDOW Win_1
Return
```

Getting Dates (The DatePicker Control) (tut12)

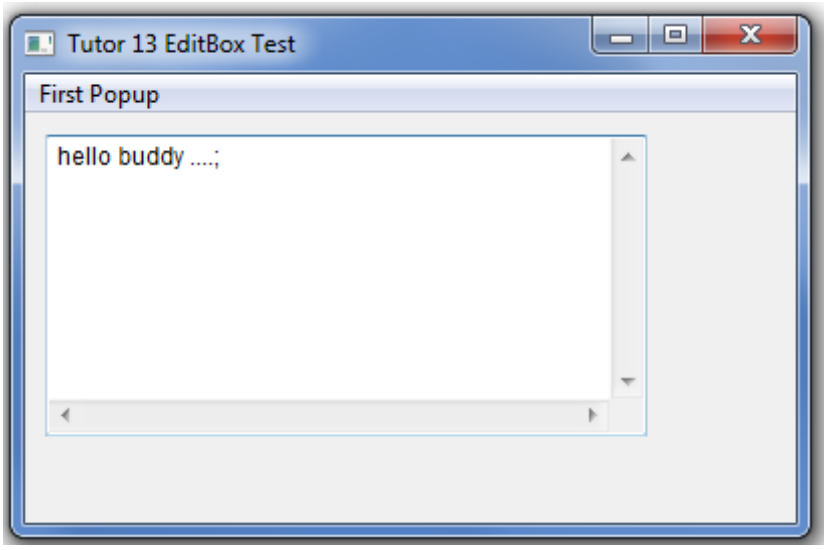


The easiest way to get dates from user is the datepicker control.

```
#include "hmg.ch"

Function Main

   DEFINE WINDOW Win_1 ;
      AT 0,0 ;
      WIDTH 400 ;
      HEIGHT 200 ;
      TITLE 'Tutor 12 - DatePicker Test' ;
      MAIN
      DEFINE MAIN MENU
         POPUP "First Popup"
           ITEM 'Change DatePicker Value' ACTION Win_1.date_1.Value := Date()
```

```
            ITEM 'Retrieve DatePicker Value' ACTION MsgInfo (
dtoc(Win_1.Date_1.Value))
          END POPUP
       END MENU
       @ 10,10 DATEPICKER Date_1
   END WINDOW

   ACTIVATE WINDOW Win_1

Return
```

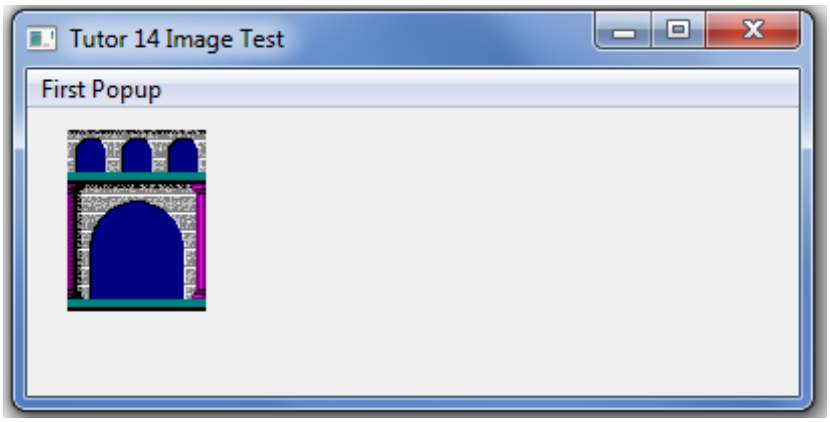
Getting Large Text (The EditBox Control) (tut13)



The EditBox control allows to handle large (multiline) text data.

```
#include "hmg.ch"
Function Main

   DEFINE WINDOW Win_1 ;
       AT 0,0 ;
       WIDTH 400 ;
       HEIGHT 300 ;
       TITLE 'Tutor 13 EditBox Test' ;
       MAIN
       DEFINE MAIN MENU
          POPUP "First Popup"
            ITEM 'Change EditBox Content' ACTION Win_1.Edit_1.Value := 'New EditBox
           Value'
            ITEM 'Retrieve EditBox Content' ACTION MsgInfo ( Win_1.Edit_1.Value)
          END POPUP
       END MENU
       @ 10,10 EDITBOX Edit_1 ;
           WIDTH 300 ;
           HEIGHT 150
   END WINDOW

   ACTIVATE WINDOW Win_1

Return
```

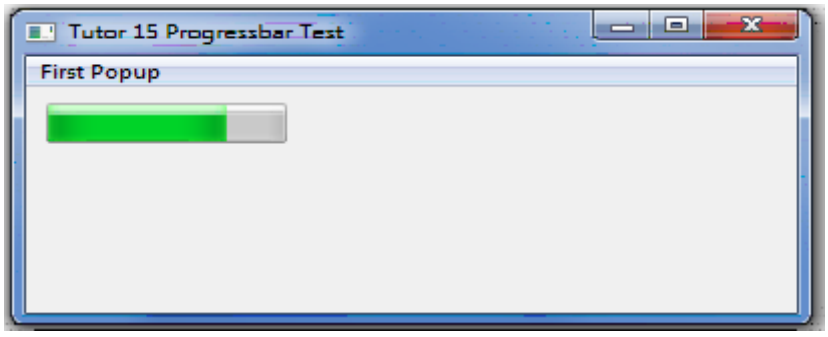
Displaying Images (The Image Control) (tut14)



The image control allows to show image files in your program.

```
#include "hmg.ch"

Function Main

   DEFINE WINDOW Win_1 ;
       AT 0,0 ;
       WIDTH 400 ;
       HEIGHT 200 ;
       TITLE 'Tutor 14 Image Test' ;
       MAIN
       DEFINE MAIN MENU
          POPUP "First Popup"
            ITEM 'Change Image Content' ACTION Win_1.Image_1.Picture := 'Open.Bmp'
            ITEM 'Retrieve Image Content' ACTION MsgInfo ( Win_1.Image_1.Picture)
          END POPUP
       END MENU
           @ 10,10 IMAGE Image_1 ;
           PICTURE 'Demo.Bmp' ;
           WIDTH 90 ;
           HEIGHT 90
   END WINDOW

   ACTIVATE WINDOW Win_1
Return
```

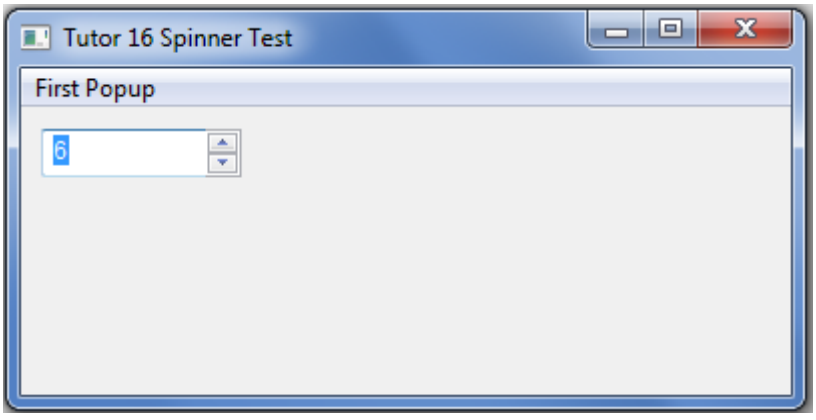
Showing Progress (tut15)



The progressbar control allows to show the completion status of an operation.

```
#include "hmg.ch"

Function Main

   DEFINE WINDOW Win_1 ;
       AT 0,0 ;
       WIDTH 400 ;
       HEIGHT 150 ;
       TITLE 'Tutor 15 Progressbar Test' ;
       MAIN
       DEFINE MAIN MENU
          POPUP "First Popup"
            ITEM 'ProgressBar Test' ACTION DoTest()
          END POPUP
       END MENU
       @ 10,10 PROGRESSBAR Progress_1 ;
           RANGE 0 , 65535
   END WINDOW

   ACTIVATE WINDOW Win_1

Return
Procedure DoTest()
Local i

   For i = 0 To 65535 Step 25
       Win_1.Progress_1.Value := i
   Next i

Return
```

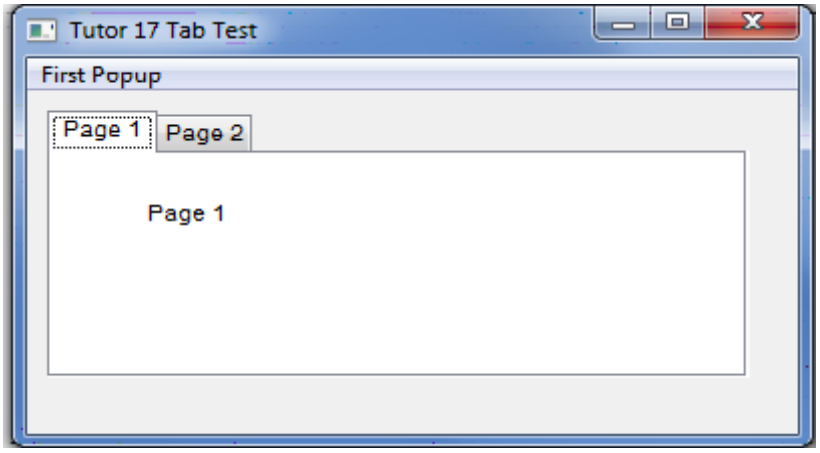
Spinning Around (tut16)



An alternate way to get numeric data is the spinner control. It consist of a text-box with two arrows that allows to change control's value using the mouse.

```
#include "hmg.ch"

Function Main

   DEFINE WINDOW Win_1 ;
       AT 0,0 ;
       WIDTH 400 ;
       HEIGHT 200 ;
       TITLE 'Tutor 16 Spinner Test' ;
       MAIN
       DEFINE MAIN MENU
          POPUP "First Popup"
            ITEM 'Change Spinner Value' ACTION Win_1.Spinner_1.Value := 8
            ITEM 'Retrieve Spinner Value' ACTION MsgInfo ( Str(Win_1.Spinner_1.
           Value))
          END POPUP
       END MENU
       @ 10,10 SPINNER Spinner_1 ;
           RANGE 0,10 ;
           VALUE 5 ;
           WIDTH 100

   END WINDOW

   ACTIVATE WINDOW Win_1

Return
```

Getting Organized (TAB Control) (tut17)

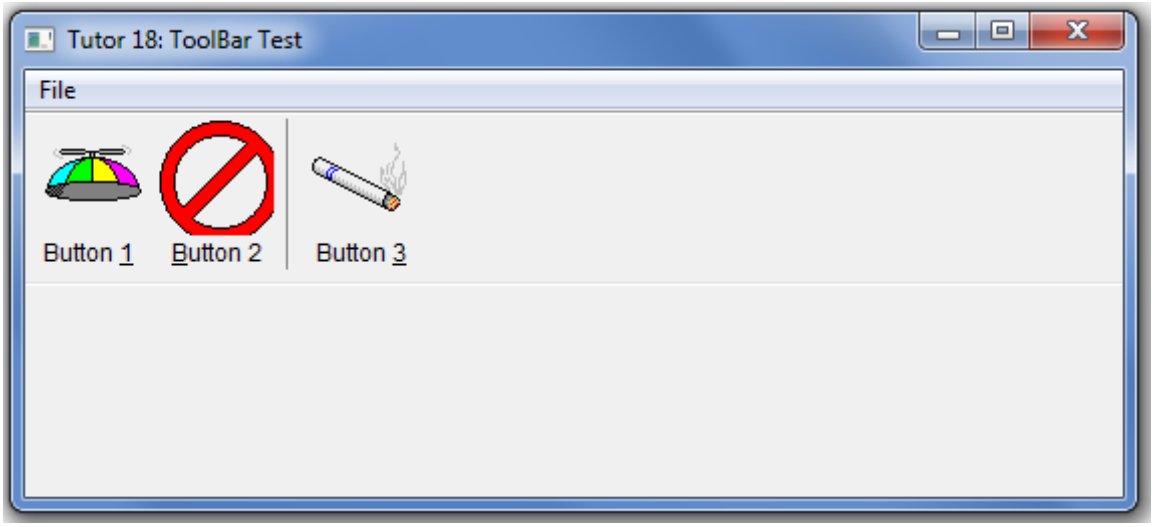


A TAB control lets organize controls and save form space, grouping the controls in folders.

```
#include "hmg.ch"

Function Main

   DEFINE WINDOW Win_1 ;
       AT 0,0 ;
       WIDTH 400 ;
       HEIGHT 250 ;
       TITLE 'Tutor 17 Tab Test' ;
```

```
      MAIN
      DEFINE MAIN MENU
         POPUP "First Popup"
           ITEM 'Change Tab Value' ACTION Win_1.Tab_1.Value := 2
           ITEM 'Retrieve Tab Value' ACTION MsgInfo ( Str(Win_1.Tab_1.Value))
         END POPUP
      END MENU
      DEFINE TAB Tab_1 ;
          AT 10,10 ;
          WIDTH 350 ;
          HEIGHT 150

          PAGE 'Page 1'
              @ 50,50 LABEL Label_1 VALUE 'This Is The Page 1'
          END PAGE

          PAGE 'Page 2'
              @ 50,50 LABEL Label_2 VALUE 'This Is The Page 2'
          END PAGE

      END TAB

   END WINDOW

   ACTIVATE WINDOW Win_1

Return
```

More Organization (TOOLBAR Control) (tut18)

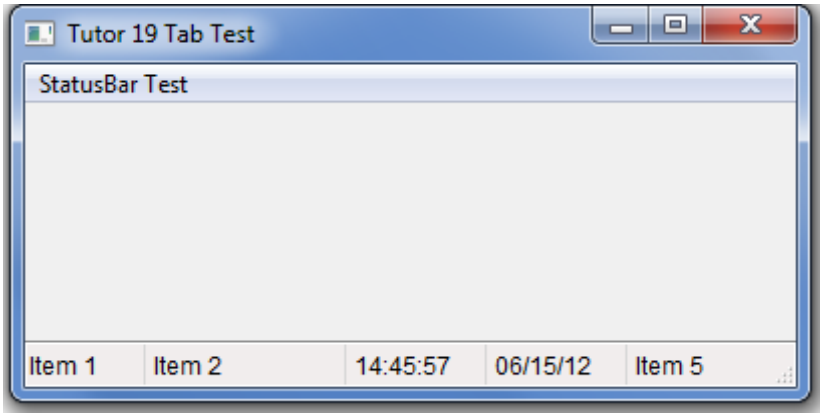


Toolbars are used to group command buttons in a bar located (usually) at window top (under main menu bar).

```
#include "hmg.ch"

Function Main

   DEFINE WINDOW Win_1 ;
       AT 0,0 ;
       WIDTH 640 HEIGHT 480 ;
```

```
        TITLE 'Tutor 18: ToolBar Test' ;
        MAIN ;
        FONT 'Arial' SIZE 10

        DEFINE MAIN MENU
            POPUP '&File'
                ITEM '&Disable ToolBar Button' ACTION Win_1. ToolBar_1.Button_1.En
                   abled := .F.
                ITEM '&Enable ToolBar Button' ACTION Win_1. ToolBar_1.Button_1.En
                   abled := .T.
                ITEM '&Exit' ACTION Win_1.Release
            END POPUP
        END MENU

        DEFINE TOOLBAR ToolBar_1 BUTTONSIZE 80,80 FLAT BORDER

            BUTTON Button_1 ;
                CAPTION '&Button &1' ;
                PICTURE 'button1.bmp' ;
                ACTION MsgInfo('Click! 1')

            BUTTON Button_2 ;
                CAPTION '&Button 2' ;
                PICTURE 'button2.bmp' ;
                ACTION MsgInfo('Click! 2') ;
                SEPARATOR

            BUTTON Button_3 ;
                CAPTION 'Button &3' ;
                PICTURE 'button3.bmp' ;
                ACTION MsgInfo('Click! 3')

        END TOOLBAR

    END WINDOW

    CENTER WINDOW Win_1

    ACTIVATE WINDOW Win_1

Return Nil
```

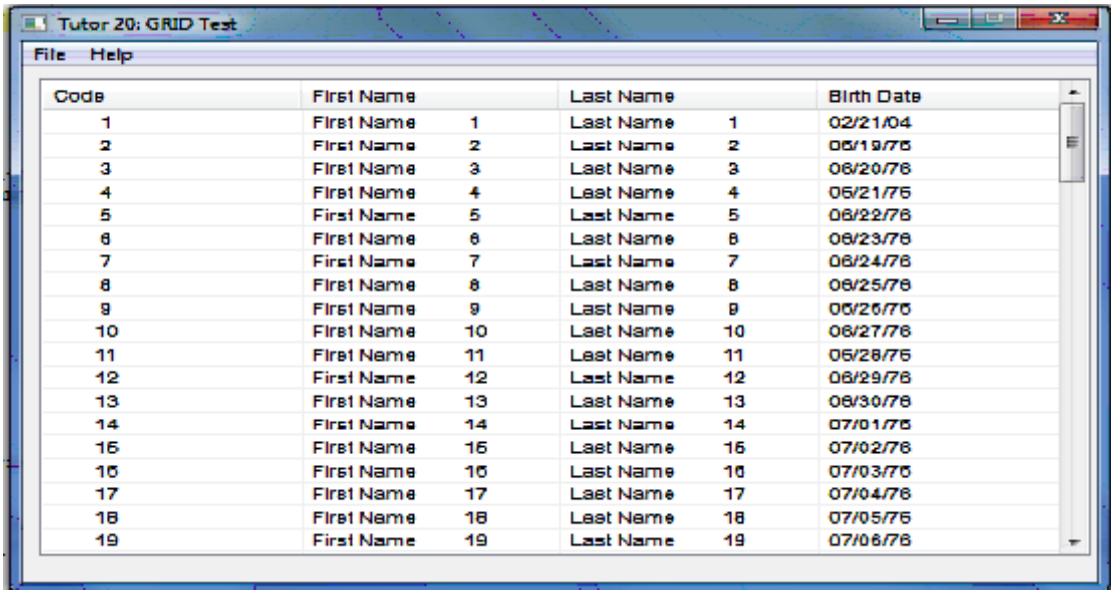
Showing Status (Statusbar Control) (tut19)



This control creates a bar at window's bottom, used to show information (usually status information)

```
#include "hmg.ch"

Function Main

   DEFINE WINDOW Win_1 ;
       AT 0,0 ;
       WIDTH 400 ;
       HEIGHT 200 ;
       TITLE 'Tutor 19 Tab Test' ;
       MAIN
       DEFINE MAIN MENU
           POPUP '&StatusBar Test'
               ITEM 'Set Status Item 1' ACTION Win_1.StatusBar.Item(1) := "New val-
ue 1"
               ITEM 'Set Status Item 2' ACTION Win_1.StatusBar.Item(2) := "New val-
ue 2"
           END POPUP
       END MENU
       DEFINE STATUSBAR
           STATUSITEM "Item 1" ACTION MsgInfo('Click! 1')
           STATUSITEM "Item 2" WIDTH 100 ACTION MsgInfo('Click! 2')
           CLOCK
           DATE
           STATUSITEM "Item 5" WIDTH 100
       END STATUSBAR

   END WINDOW

   ACTIVATE WINDOW Win_1

Return
```

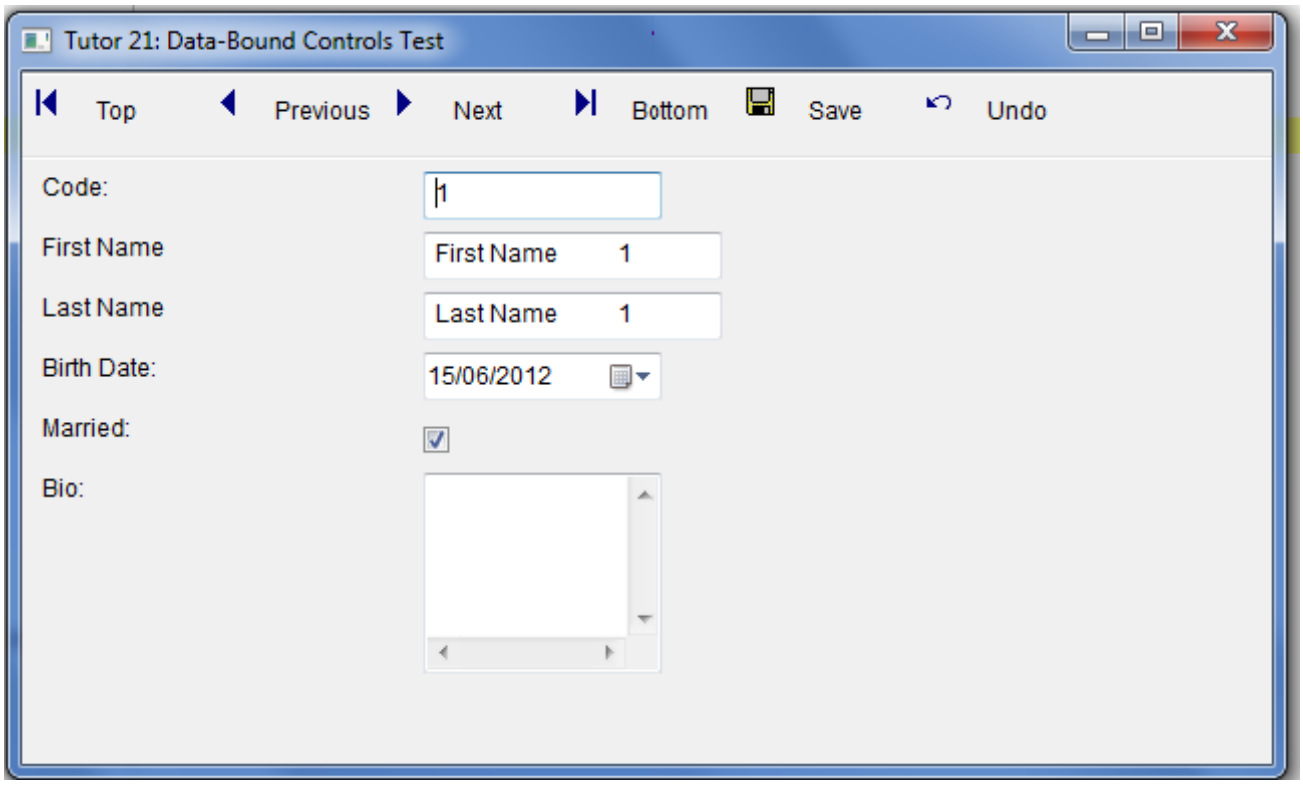
Data-Controls I: DATA-BOUND GRID (tut20)



Data controls are designed to handle data in .dbf files directly.

```
Grid control allows to show / edit database records in tabular format.
'RecNo' property is use to set / get the selected record number (recno()).

#include "hmg.ch"

Function Main

OpenTables()
   DEFINE WINDOW Win_1 ;
       AT 0,0 ;
       WIDTH 640 HEIGHT 480 ;
       TITLE 'Tutor 20: BROWSE Test' ;
       MAIN NOMAXIMIZE

       DEFINE MAIN MENU
           POPUP 'File'
               ITEM 'Set Grid RecNo' ACTION Win_1.Grid_1.Value := Val ( InputBox
               ('Set Grid RecNo','') )
               ITEM 'Get Grid RecNo' ACTION MsgInfo ( Str ( Win_1.Grid_1.RecNo ) )
               SEPARATOR
               ITEM 'Exit' ACTION Win_1.Release
           END POPUP
           POPUP 'Help'
               ITEM 'About' ACTION MsgInfo ("Tutor 20: GRID Test")
           END POPUP
       END MENU

       @ 10,10 GRID Grid_1 ;
           WIDTH 610 ;
           HEIGHT 390 ;
           HEADERS { 'Code' , 'First Name' , 'Last Name', 'Birth Date', 'Married' ,
            'Biography' } ;
           WIDTHS { 150 , 150 , 150 , 150 , 150 , 150 } ;
           ROWSOURCE "Test" ;
           COLUMNFIELDS { 'Code' , 'First' , 'Last' , 'Birth' , 'Married' , 'Bio' }
;
           ALLOWDELETE ;
           ALLOWEDIT

   END WINDOW

   CENTER WINDOW Win_1

   ACTIVATE WINDOW Win_1

Return Nil

Procedure OpenTables()
   Use Test
   Win_1.Browse_1.Value := RecNo()
Return Nil

Procedure CloseTables()
   Use
Return Nil
```

Data-Controls II: TEXTBOX, DATEPICKER, CHECKBOX, EDITBOX (tut21)



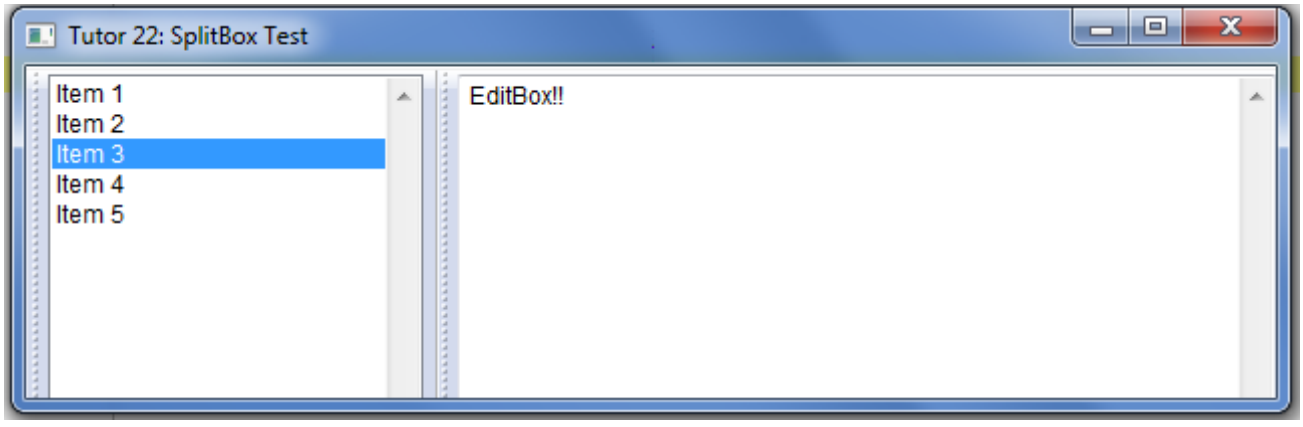
The 'data-version' of these controls require the use of the following properties / methods to link them to a database field:
   Field Property: Establishes the field that control is linked to.
   Refresh Method: Updates control content based on current field content.
   Save method: Updates database file according control content.

```
#include "hmg.ch"
Function Main

   DEFINE WINDOW Win_1 ;
       AT 0,0 ;
       WIDTH 640 ;
       HEIGHT 480 ;
       TITLE 'Tutor 21: Data-Bound Controls Test' ;
       MAIN ;
       ON INIT OpenTables() ;
       ON RELEASE CloseTables()

       DEFINE TOOLBAR ToolBar_1 BUTTONSIZE 100,30 FLAT RIGHTTEXT BORDER

           BUTTON TOP ;
               CAPTION '&Top' ;
               PICTURE 'primero.bmp' ;
               ACTION ( DbGoTop() , Refresh() )

           BUTTON PREVIOUS ;
               CAPTION '&Previous';
               PICTURE 'anterior.bmp' ;
               ACTION ( DbSkip(-1) , Refresh() )
```

```
        BUTTON NEXT ;
            CAPTION '&Next';
            PICTURE 'siguiente.bmp' ;
            ACTION ( DbSkip(1) , if ( eof() , DbGoBottom() , Nil ) , Refresh() )

        BUTTON BOTTOM ;
            CAPTION '&Bottom' ;
            PICTURE 'ultimo.bmp' ;
            ACTION ( DbGoBottom() , Refresh() )

        BUTTON SAVE ;
            CAPTION '&Save' ;
            PICTURE 'guardar.bmp' ;
            ACTION ( Save() , Refresh() )

        BUTTON UNDO ;
            CAPTION '&Undo' ;
            PICTURE 'deshacer.bmp' ;
            ACTION ( Refresh() )

    END TOOLBAR

    @ 50,10 LABEL LABEL_1 VALUE 'Code:'
    @ 80,10 LABEL LABEL_2 VALUE 'First Name'
    @ 110,10 LABEL LABEL_3 VALUE 'Last Name'
    @ 140,10 LABEL LABEL_4 VALUE 'Birth Date:'
    @ 170,10 LABEL LABEL_5 VALUE 'Married:'
    @ 200,10 LABEL LABEL_6 VALUE 'Bio:'

    @ 50,200 TEXTBOX TEXT_1;
        FIELD TEST->CODE ;
        NUMERIC ;
        MAXLENGTH 10

    @ 80,200 TEXTBOX TEXT_2;
        FIELD TEST->FIRST ;
        MAXLENGTH 30

    @ 110,200 TEXTBOX TEXT_3;
        FIELD TEST->LAST ;
        MAXLENGTH 30

    @ 140,200 DATEPICKER DATE_4 ;
        FIELD Test->Birth

    @ 170,200 CHECKBOX CHECK_5 ;
        CAPTION '' ;
        FIELD Test->Married

    @ 200,200 EDITBOX EDIT_6 ;
        FIELD Test->Bio ;
        HEIGHT 100

END WINDOW

Win_1.Text_1.SetFocus

ACTIVATE WINDOW Win_1
```

```
Return Nil

Procedure Refresh

   Win_1.Text_1.Refresh
   Win_1.Text_2.Refresh
   Win_1.Text_3.Refresh
   Win_1.Date_4.Refresh
   Win_1.Check_5.Refresh
   Win_1.Edit_6.Refresh
   Win_1.Text_1.SetFocus

Return

Procedure Save

   Win_1.Text_1.Save
   Win_1.Text_2.Save
   Win_1.Text_3.Save
   Win_1.Date_4.Save
   Win_1.Check_5.Save
   Win_1.Edit_6.Save

Return

Procedure OpenTables
   USE TEST
Return

Procedure CloseTables
   USE
Return
```

More Organization II (SPLITBOX Control) (tut22)



Controls defined as part of this container can be arranged by users, using a grip-perbar located at control's left side. You must omit '@ <row>,<col>' in control definition.

```
#include "hmg.ch"

Function Main
```

```
   DEFINE WINDOW Win_1 ;
       AT 0,0 ;
       WIDTH 640 HEIGHT 450 ;
       TITLE 'Tutor 22: SplitBox Test' ;
       MAIN

       DEFINE SPLITBOX

           LISTBOX List_1 ;
               WIDTH 200 ;
               HEIGHT 200 ;
               ITEMS {'Item 1','Item 2','Item 3','Item 4','Item 5'} ;
               VALUE 3 ;
               TOOLTIP 'ListBox 1'

           EDITBOX Edit_1 ;
               WIDTH 200 ;
               HEIGHT 400 ;
               VALUE 'EditBox!!' ;
               TOOLTIP 'EditBox' ;
               MAXLENGTH 255

       END SPLITBOX

   END WINDOW

   CENTER WINDOW Win_1

   ACTIVATE WINDOW Win_1

Return Nil
```

--
Roberto Lopez <mail.box.hmg@gmail.com>