Table of Contents

Introduction

Properties [a..z]

Events

Methods

Misc. Commands

Print System

HMG

xBase Development System For Windows

Systems can be built in 32\/64-bit and ANSI\/Unicode © 2002-2016 Roberto Lopez

All rights reserved

HMG Development Team: 2002-2012 Roberto Lopez (mail.box.hmg@gmail.com)

http://sites.google.com\/site\/hmgweb

2012-2016 S. Rathinagiri (srathinagiri@gmail.com)

http://www.rathinagiri.in

Dr. Claudio Soto (srvet@adinet.com.uy)

http://srvet.blogspot.com

Contributors for this HMG Doc:

- Roberto Lopez
- Dr. Claudio Soto
- Bicahi Esgici
- Pablo César
- S. Rathinagiri

HMG Help - Basics

HMG and Alternate Syntax:

All over this help document you can find that there are two types of syntax available for the definition of various controls. One is traditional **'@ row, col CONTROLTYPE controlname'** syntax and the alternate syntax with **'define....end'** statements. Each synatx is having its own merits and demerits.

@ nRow, nCol CONTROLTYPE controlname

While '@ nRow, nCol CONTROLTYPE controlname' syntax is a single line simple syntax it requires the sequence of various properties of that control as it is. For example the following statement is right:

```
@ 10,10 textbox t1 height 40 width 100
```

Where as the following is wrong

```
@ 10,10 textbox t1 width 100 height 40
```

In other words, you have to memorize the sequentail order of the various properties also.

Alternate Syntax: DEFINE CONTROLTYPE controlname:

In the case of alternate syntax, the control definition is between DEFINE....END statements and the property values can be in any order. That is the advantage of alternate syntax though it requires more lines of coding.

For example, both the following are right:

```
define textbox t1
   row 10
   col 10
   width 100
   height 30
end textbox
```

```
define textbox t1
   row 10
   col 10
   height 30
   width 100
end textbox
```

- BUILD.BAT

The easiest way is to use the Build.Bat file located at HMG root folder.

Usage:

Build [**/n**] [**/d**] [**/c**] <program.prg> | <project.hbp> [<hbmk2 params>]

[**/n**] : No run after build

[**/d**] : Enabled debugger

[**/c**] : Console mode

<program.prg> : A main text file containing your source code.

<filelist.hbp> : A text file with .hbp extension containing a source list.

<hbmk2 params> : A text file with configuration parameters as additional libs, include paths and lib paths.

This is also informed the .hbc file of your project.

For example:

> incpaths = incpath1 incpath2 ... incpathn
>
> libpaths = libpath1 libpath2 ... libpathn
>
> libs = lib1 lib2 ... libn
>
> mt = 'yes' (multi-threading support)
>
> See sample at: ..\SAMPLES\Basics\MULTIPRG)
>
> **Notes:**
>
> · Library names must not include 'lib' prefix nor '.a' extension.
>
> · Build.bat will create an error.log file in the app folder when build process ends with an error condition.
>
> · If in case of no parameter has been informed, any action is going to happen using the main Build.bat at HMG folder. Some Build.bat at C:\HMG..\SAMPLES will accept to compile even without any parameter because it will acts according all hbp files in the current folder. In this case, all files in the current folder are automatically be processed.
>
> · To build projects with long names option.

> Spaces at folders and prg files names normally is not required, but sometimes we would like to use in that way. Since on hmg 3.3.0 version, have been implemented for Build.bat and this works perfectly.
>
> Example at dot command (see below in red color example):
>
> **Build "My Project"** (must be in quoted)
>
> § This is regarding being possible for: hbp file, prg file or project folder names.
>
> § Not only used for your main prg, also for others prgs files with spaces.
>
> § But all files\/folders with long names, must be always in quoted and saved at hbp file.
>
> For hbp file: "My Project.hbp"
>
> For prg file: "My Main.prg"
>
> For project folder: "My Main"
>
> § For old projects will still keep it original names at least you want to change with spaces on it and re-build it with adding quotation marks.
>
> § Executable files with long names (spaces included) is created normally and also at project folder with long name.
>
> § This feature is not yet workable at IDE building because IDE needs to adjust for hbp creation with quotation marks.
>
> · To Build apps making executable file copy in a second place.
>
> When we compile our application we do it in an appropriate folder where we keep the source code of the project. But we often need to copy the executable that was generated for the right system folder (in use), i.e. where the data files are placed. So in that case we will need to use Windows Explorer to copy\/paste the executable in that one.
>
> To simplify this procedure we can automatize by edting your <Project_name>.hbc file and add this:
>
> **instpaths="c:\In other place\Sys\"** (with slash bar at end of full path)
>
> This is a hbc directives from Harbour to copy the target to <path>. If <path> is a directory, must end with path separator. This can be specified multiple times.

- HMG IDE

This programming environment allows visual form editing and application build.

Environment settings (HMG Folder, Program Editor, etc.) are stored in '.ini' files, located at IDE folder.

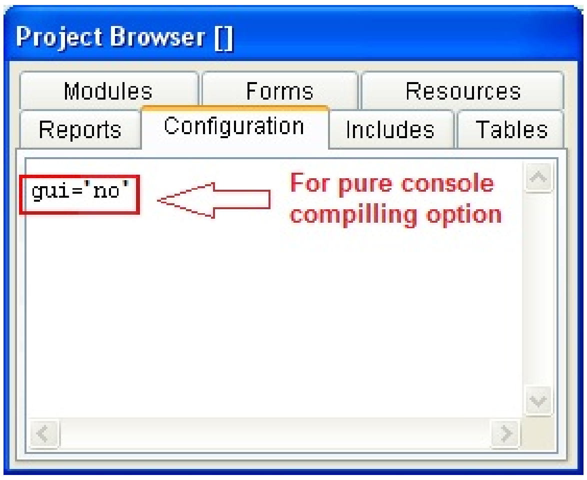
You can interactively change default settings (Main Menu->Tools->Preferences).

In IDE, all build customizations must be handled by the user from 'Configuration' (project browser) tab. It is direct edit of the project .hbc file.

Look at hbmk2 help for details. By default two gt drivers are ALWAYS linked:

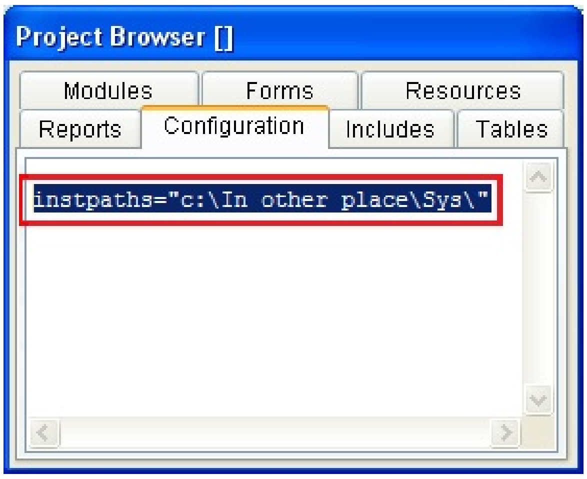
· GTGUI (as default) and GTWIN (Windows console)

· You can compile pure console adding **gui='no'** (in quotes)



```
gui='no'
```

· To Build apps making executable file copy in a second place at IDE, you will need write instpaths="¦ in Project Browser\/Configuration as follows:



```
instpaths="c:\In other place\Sys\"
```

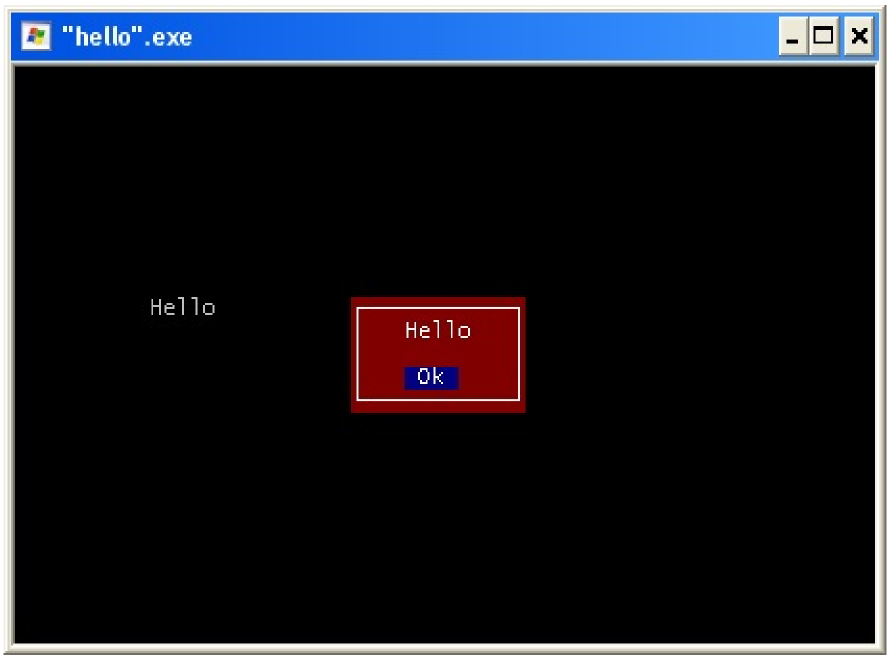
· To create apps console\/mixed mode, you only need to add this followinq command (red color):

REQUEST HB_GT_WIN_DEFAULT (at top of your main prg file).

See example:

```
REQUEST HB_GT_WIN_DEFAULT

Function Main()
SetMode(25,80)

Cls
?
@ 10,10 Say 'Hello'

Alert('Hello')
Return Nil
```

Screenshot



Tip:

When upgrading HMG, please clean previous build data from your projects using incremental building, use

the 'Reset project incremental data' option from the project menu.

Your First HMG Program

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
```

¼ **DEFINE WINDOW** command will create the main window for the program.

¼ **Win_1** is the name of the window.

¼ **AT 0,0** indicates the window position (row=0,col=0)

¼ **WIDTH 400** means that the window will have 400 pixels width.

¼ **HEIGHT 200** means that the window will have 200 pixels height.

¼ **TITLE 'Hello World!'** indicates the text in the window title bar.

¼ **MAIN** indicates that we are defining the main application window

(a main window is required for all HMG applications)

¼ **ACTIVATE WINDOW Form_1** will show the window and start the event loop.

Adding The Main Menu

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.

The MsgInfo() Function

This is a very useful function. It will show a little message window (with the system information icon) and a message (character type) passed as parameter.

You can optionally add a title (passed as the second parameter)

Adding a Label

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

```
 #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
```

@ 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 identify by this name)

VALUE clause indicates the initial value of the control when created.

Getting data from the user (The TextBox Control)

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

If you want to get numeric data, just add **NUMERIC** clause.

If you want to indicate an editing mask, you can use the **INPUTMASK** clause.

```
 #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.Va
lue := 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

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

We add **CHECKBOX** 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

Sometimes, you need to get a choice from users, between a small group of options that are known at design time.

We can make use of either one of the three:

- RadioGroup
- ListBox
- ComboBox

Making a Choice

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.Radio_1.Valu
e))
           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

There is various alternatives to get an user's choice besides RadioGroup.

One of them is the ListBox Control

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 06 - 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

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 06 - 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.Value)
 )
              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.ItemCount))
            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)

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 07 - 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)

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 8 - 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 HMG controls.

Button + CheckBox = CheckButton

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 9 - 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.CheckBu
tton_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.Chec
kButton_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)

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 10 - 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.Valu
e))
           END POPUP
        END MENU
        @ 10,10 DATEPICKER Date_1
    END WINDOW
    ACTIVATE WINDOW Win_1
Return
```

Getting Large Text (The EditBox Control)

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 11 - 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)

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 12 - 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

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 200 ;
        TITLE 'Tutor 13 - 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

Procedure DoTest()
Local i
    For i = 0 To 65535 Step 25
        Win_1.Progress_1.Value := i
    Next i
Return
```

Spinning Around

An alternate way to get numeric data is the spinner control. It consist of a textbox 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 14 - 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)

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 15 - 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)

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 16 - ToolBar Test' ;
        MAIN ;
        FONT 'Arial' SIZE 10
        DEFINE MAIN MENU
            POPUP '&File'
                ITEM '&Disable ToolBar Button' ACTION Win_1. ToolBar_1.Button_1.Enable
d := .F.
                ITEM '&Enable ToolBar Button' ACTION Win_1. ToolBar_1.Button_1.Enabled
 := .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
```

More Organization II (SPLITBOX Control)

Controls defined as part of this container can be arranged by users, using a gripperbar 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 17 - SplitBox Test' ;
        MAIN
        DEFINE SPLITBOX
            LISTBOX List_1 ;
                WIDTH 200 ;
                HEIGHT 400 ;
                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
```

Showing Status (Statusbar Control)

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 18 - Tab Test' ;
        MAIN
        DEFINE MAIN MENU
            POPUP '&StatusBar Test'
                ITEM 'Set Status Item 1' ACTION Win_1.StatusBar.Item(1) := "New value
1"
                ITEM 'Set Status Item 2' ACTION Win_1.StatusBar.Item(2) := "New value
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

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 19 - GRID Test' ;
        MAIN NOMAXIMIZE
        DEFINE MAIN MENU
            POPUP 'File'
                ITEM 'Set Grid RecNo' ACTION Win_1.Grid_1.Value := Val ( InputBox ('Se
t 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

The 'data-version' of these controls required the use of the following properties \/ methods to bound them to a database field:

Field Property: Stablishes the field that control is bounded 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 20 - 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
```

What is HMG ?

Well.. for the new people approaching HMG I can say that this is a mix between one of the best programming and data manipulation languages ever created (xBase) and the VB\/RapidQ GUI handling style.

All GUI object are public and can be created and managed with very simple code.

The basic components of HMG are:

· Harbour Compiler (generates C code from xBase code).

· Harbour HMG library (functions and preprocessor directives to handle GUI).

· MingW Compiler.

· HMG IDE (Optional tool for two-way visual design).

What are the main project goals ?

To keep the GUI handling as easy, elegant and clean as possible and the library core as compact, stable and reliable as we can.

It implies, that the changes to the core will be subject to very strict test prior tag a specific build as 'stable'.

Other of the main goals is to hide the complexities associated with the operating system internals, allowing to the programmer, focus on his application, instead OS technical things.

So, what is the place for experimentation and exciting new 'things' ?

The User components interface, of course.

I've created to let to any HMG user to add fully new GUI elements, or add properties and events to existing ones.

Where is the HMG site ?

http:\/\/www.hmgforum.com\/index.php

Where can I download HMG ?

http:\/\/sourceforge.net\/projects\/hmg\/files\/HMG3\/

http:\/\/www.hmgforum.com\/site\/

Why Roberto doesn't answer my messages ?

I have a very little free time for HMG right now. Anyway I'm reading and answering the bug area on this forum, almost daily, please use it.

Does HMG will be 'real' OOP some day in the future ?

Who knows

What about user contributions ?

I've noticed about comments like "Roberto does not accept code contributions anymore".

This is true, but need some clarification.

In 2005 I've decided to change the way I work on HMG.

From that moment I do all modifications personally, but I still receiving comments, suggestions ideas and wishes from the users.

The reason for this, is that I want to keep the control of the code included in HMG, to assure that it remains compact, fast, stable, reliable and backwards compatible, keeping and eye on general project goals (enumerated in the previous message).

An example of this, is the recent Activex addition.

I've noticed that this feature was incorporated to the Freewin project.

I've analyzed the code, learned how it works ant then rewrote it from the scratch to suit HMG needs.

This way, I'm not incorporating a 'black box' to the library that could be difficult or impossible to fix or modify. I'm incorporating code that is simple and easy to maintain.

I can make mistakes and write buggy code (of course) but there is a big possibility that I'm be able to fix it, since I've wrote it.

So, You can send to me (to HMG wish area on this forum) sample code with your proposed addition. I'll take a look at it.

What about other HMG based projects... ?

Why do not all HMGs guys unite and work together ?

I'll could add another question:

Why the humanity does not unite and work together?

Well... I appreciate and respect a lot to Grigory, Jacek and Janusz from HMG Extended and to Ciro and Vicente from OOHG. All of them are doing a great work, but simply we have different opinions about of what the project has to be. Thats all.

This is very good for users, since they have three alternatives to choice.

I feel very honored because two new projects emerged from my original work.

Why do not add to library a nice widget that do something... ?

It could be good for my application that needs it.

I could write now a long explanation about the need that the features added to the core library must be generic, designed to fill the needs of most common programming situations and not to solve a specific user's problem.

Fortunately you could get the Simpsons episode 7F16

In this episode Homer designs a car to suit his own needs. As you can imagine, the car was a monstrosity and (of course) a complete failure.

If the idea is not clear enough yet, here is a little explanation:

Adding lot of things, designed to suit specific (very unusual situations) to the library core, will give us a fat, slow, buggy and difficult to learn GUI system. I'll avoid it as much as I can.

Why do you don't release the HMG IDE source code ?

Which are your obscure intentions about it ?

IDE is a very big application. It has about 41000 lines of code.

To make it run faster it relies heavily on HMG and some C code to cover low level things not implemented on the library.

Since it relies on library internals, sometimes, modifications to library, obligate to me to make changes (usually extensive) to IDE code.

IMHO these (and other things) turn this project into a personal one, not suitable to make its code public.

Anyway, I not discard the possibility to make the code public in the future.

I can only assure that IDE will always be FREELY DISTRIBUTED as freeware.

There is no evil intentions behind this decision

Maybe there is a problem (not fully specified) with control XXX but I'm not sure.

Can you fix it please ?

My application is not working as expected.

I've attached to the message all app 5000 lines of code.

I guess that must be an HMG problem, please solve it.

Ok. I'll say once again

I cannot analyze a presunct bug based on a tale about it, or read a thousand lines app to attempt to find it. It is very time consuming and inefficient way to go.

Please, send me a simple sample showing the bug, to HMGForum bug report area. The sample must include all necessary files to compile and run it.

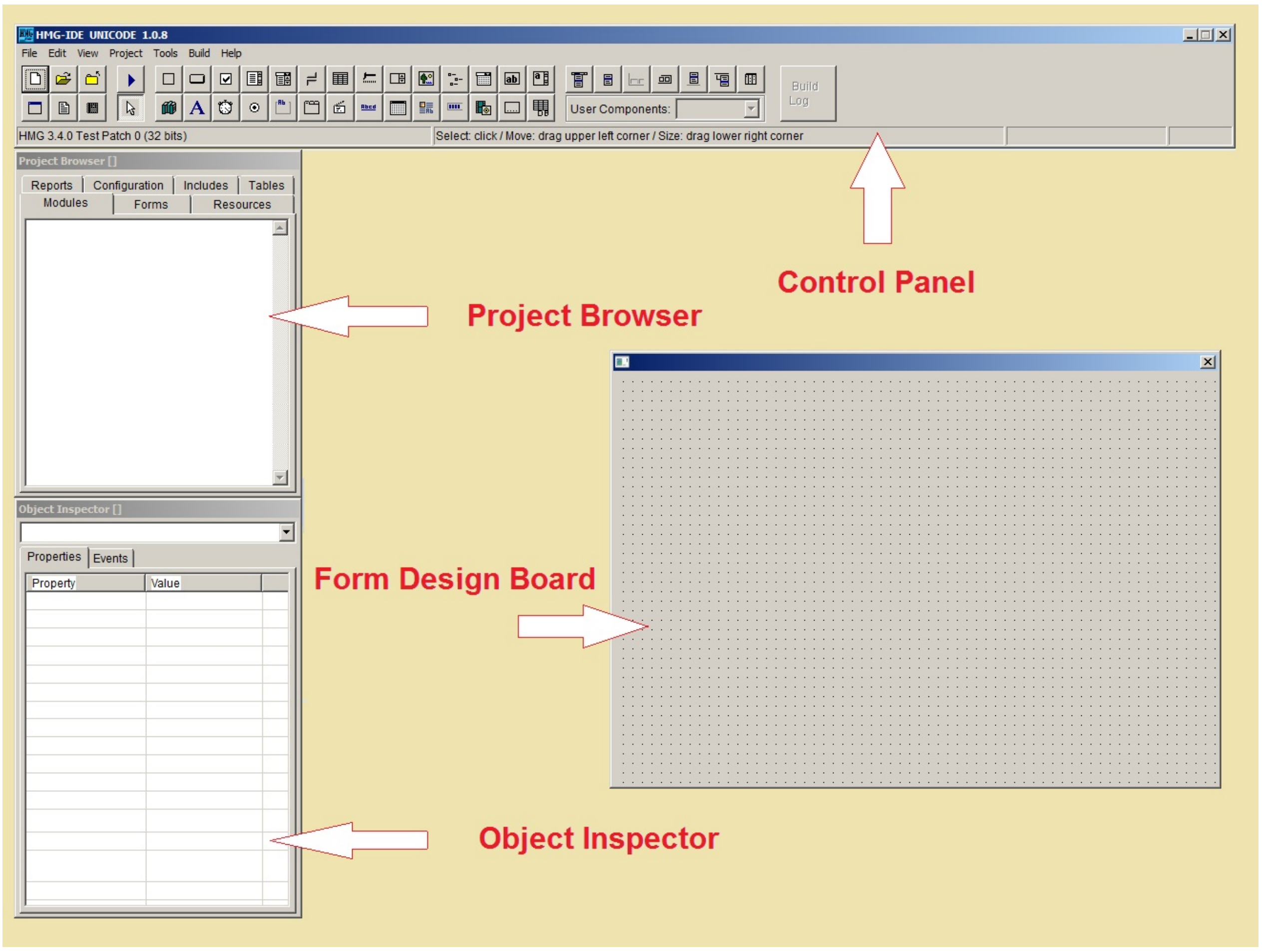
If you are not able to reproduce the problem in a small sample, is highly probable that be not an HMG problem, but a error in your app.

Roberto

HMG IDE

HMG Integrated **D**evelopment **E**nvironment is a comprehensive and highly sophisticated tool for projects management, form design in development and is extremely easy to use.

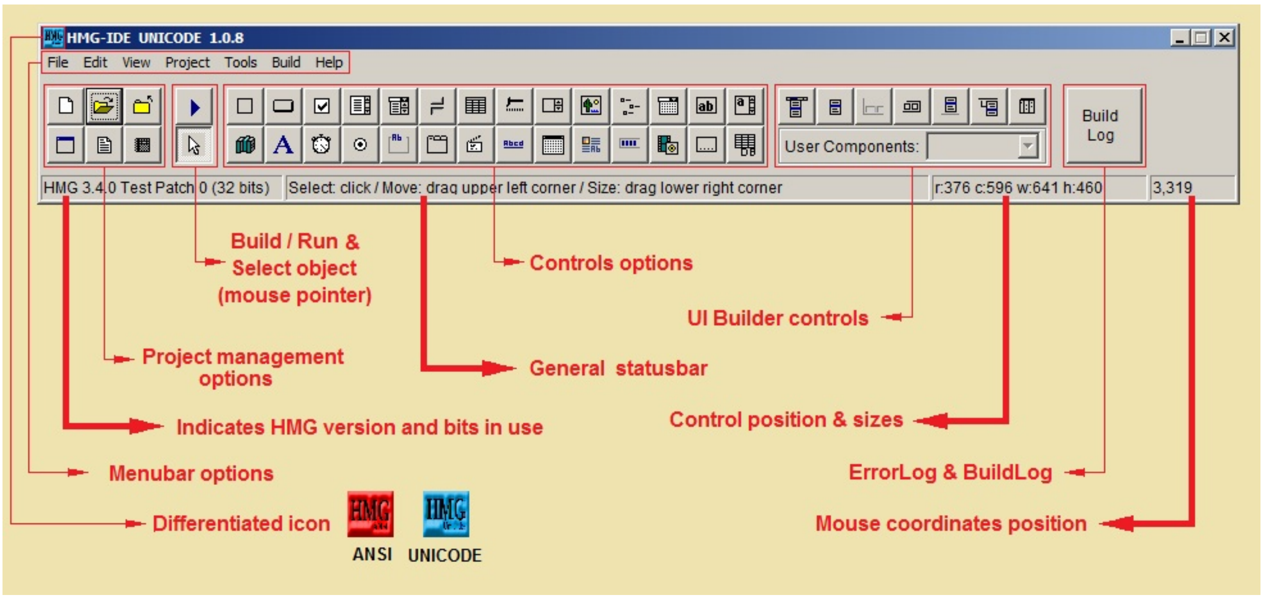
HMG IDE works with following 4 windows:



1. Control Panel (main window)
2. Project Browser
3. Object Inspector
4. Form Design

1.Control Panel (main window)

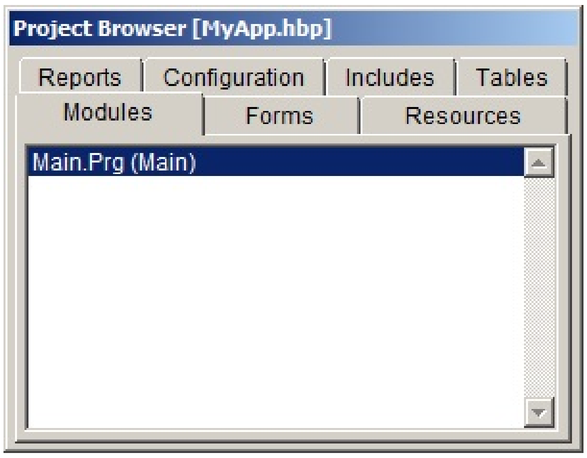
The main window is constituted on a menu bar and a tool box, having many command buttons with descriptive tool tips. This tool box may consider two sections: project management and form designer which this last one is divided into a "controls" and "builders".


HMG-IDE UNICODE 1.0.8
File Edit View Project Tools Build Help
Build Log
User Components:
HMG 3.4.0 Test Patch 0 (32 bits)
Select: click / Move: drag upper left corner / Size: drag lower right corner
r:376 c:596 w:641 h:460
3,319
Build / Run & Select object (mouse pointer)
Controls options
UI Builder controls
Project management options
General statusbar
Control position & sizes
Indicates HMG version and bits in use
Menubar options
ErrorLog & BuildLog
Differentiated icon
ANSI
UNICODE
Mouse coordinates position

2.Project Browser

The Project Browser window has four tabs:

1. Modules. All your sources files (.PRGs).
2. Forms. All your components for screens (.FRMs).
3. Resources. Embed images files (icons, mouse pointers, pictures) saved in .RC file.
4. Reports. Formatted print list files (.RPTs)
5. Configuration. Compilling settings (see hbmk2 options) saved in .HBC file.
6. Includes. Headers files (.ch).
7. Tables. All your databases files (.DBFs).



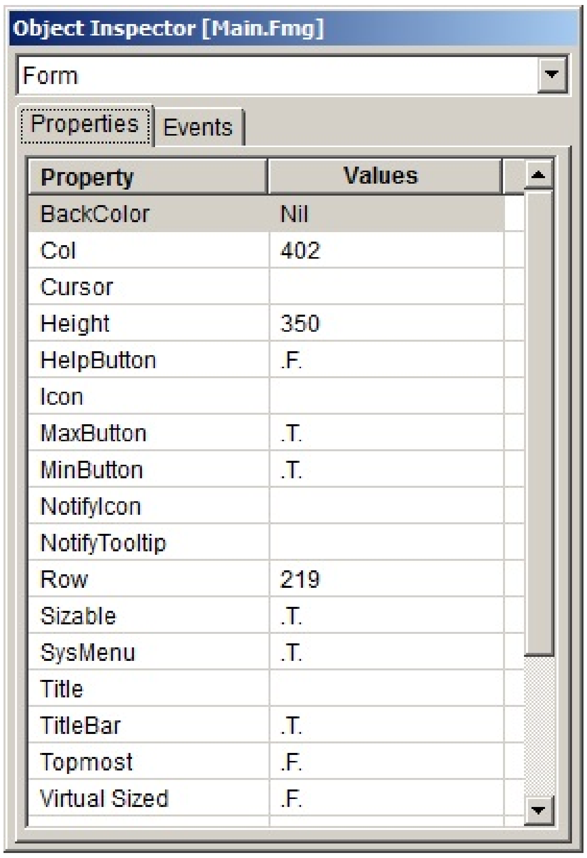
You can view, select and inspect all project elements in this window. Whenever you add or exclude any module (source files, forms, or reports), IDE automatically updates the project browser.

3.Object Inspector

The Object Inspector window is for view and change ***properties*** and ***events*** of GUI elements in your forms.

You can observe and modify properties and events value of graphical elements of your form in the Object Inspector window. **Some commonly listed properties:** BackColor, Col, Cursor, Height, Icon, MaxButton, MinButton, Row, Sizable, SysMenu, Title, TopMost, Visible, Width, WindowType

Common Events: OnInit, OnRelease, OnInteractiveClose, OnMouseClick, OnMouseDrag, OnMouseMove, OnSize, OnMaximize, OnMinimize, OnPaint, OnNotifyClick, OnGotFocus, OnLostFocus, OnScrollUp, OnScrollDown, OnScrollLeft, OnScrollRight, OnHScrollBox, OnVScrollBox, Action, OnChange, OnDisplayChange, OnSelect, OnEnter, OnVScroll, OnDblClick, OnHeadClick, OnQueryData



Property	Values
BackColor	Nil
Col	402
Cursor	
Height	350
HelpButton	.F.
Icon	
MaxButton	.T.
MinButton	.T.
NotifyIcon	
NotifyTooltip	
Row	219
Sizable	.T.
SysMenu	.T.
Title	
TitleBar	.T.
Topmost	.F.
Virtual Sized	.F.

4. Form Design

The form window is a chalkboard for designing forms and directing its graphical elements. New or existing, when you open a form, this windows also opened by IDE. With just a double-click you can easily place controls on your form: the first on desired button of control in form design"™s tool box and the second one is anywhere in form you like. After placed, you can resize and change its place by dragging.

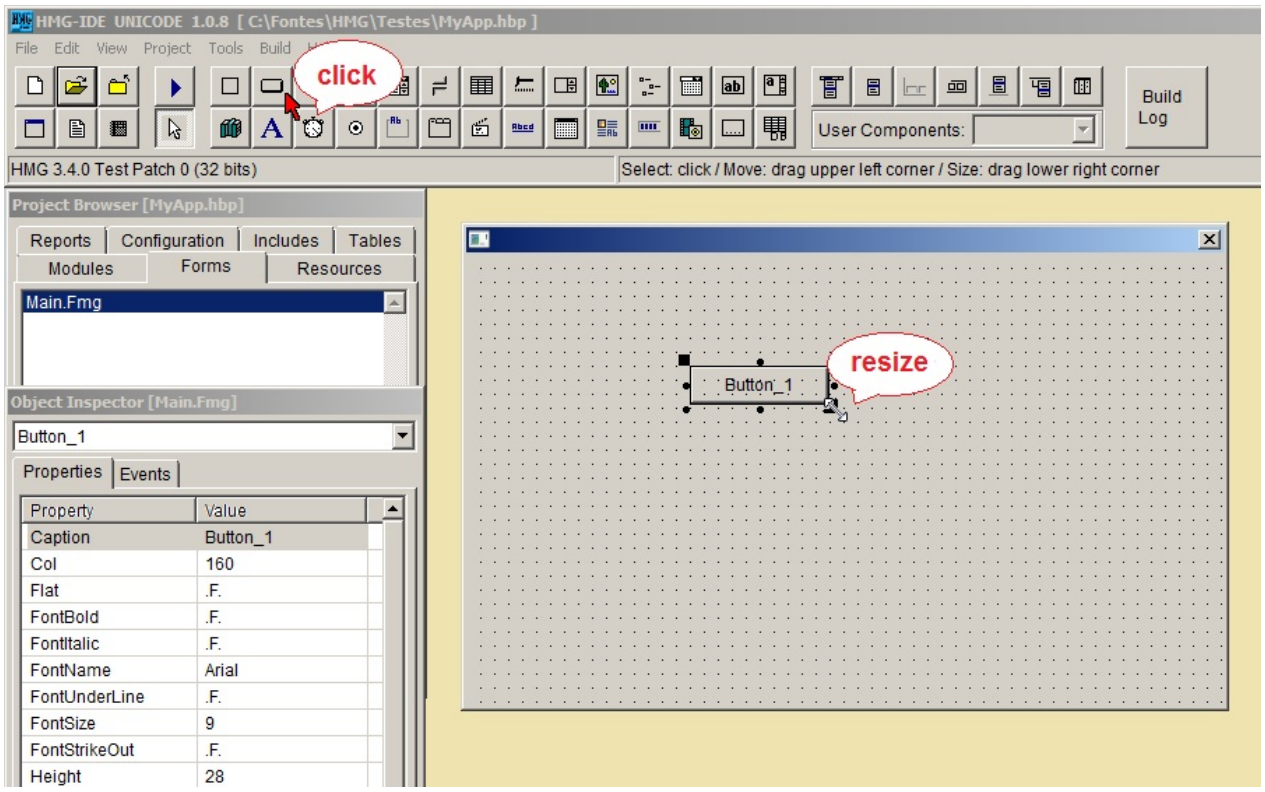
When you placed a control in your form, IDE assign default values to its properties and events. Whenever you change these values interactively on the form, IDE also updates them internally. You can observe and modify them in the Object Inspector window.

HMG forms are designed "two way" manner. Saved in a readable format; in fact they are pure HMG source codes, neither binary nor cryptic. You can separately open, inspect and modify it too. When opened by IDE will automatically converted to visual form.

Form design is somewhat WYSIWYG (what you see is what you get), i.e. it looks quite like its associated window, but a form always has a title bar, an active close box and minimize button, and a border, and are always resizable while the associated window may not have those features. These features are provided for forms to make them easy to design.

You can try to start with this sequence for form creation:

Define Window type and change all properties to your liking. · Click for any control at Control Panel and click inside de form design to be placed. · You can change positions, sizes and it properties as well events too. · When finished you must save it. This could be done at any time also during building process.



HMG-IDE UNICODE 1.0.8 [C:\Fontes\HMG\Testes\MyApp.hbp]

HMG 3.4.0 Test Patch 0 (32 bits)

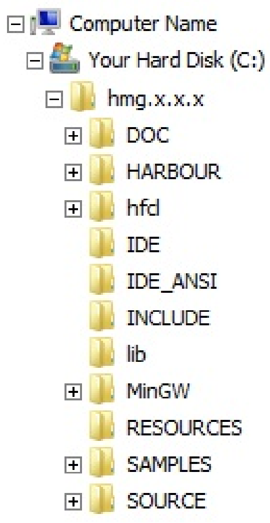
Select: click / Move: drag upper left corner / Size: drag lower right corner

Property	Value
Caption	Button_1
Col	160
Flat	.F.
FontBold	.F.
FontItalic	.F.
FontName	Arial
FontUnderLine	.F.
FontSize	9
FontStrikeOut	.F.
Height	28

Step 1

- Download latest release of HMG package and run HMG setup wizard for installation.
- Install HMG at normal and default path for location, this is the best practice.

Installer will create with following HMG folders structure:



```
Computer Name
  Your Hard Disk (C:)
    hmg.x.x.x
      DOC
      HARBOUR
      hfcl
      IDE
      IDE_ANSI
      INCLUDE
      lib
      MinGW
      RESOURCES
      SAMPLES
      SOURCE
```

At above picture where's **hmg.x.x.x** means that is the hmgroot folder. Which folder name is composed by HMG version installed.

Regarding **HARBOUR** and **MINGW** sub-folders, means you haven't need to download or install neither and have been included for use with more convenience and assurance of its versions that can work with each specific HMG version.

Regarding IDE, there two version: ANSI and UNICODE and are available to use in IDE folders.

Step 2

- To start edit your source code (PRGs files) you will need to install your favorite source code editor that you can feel more comfortable counting with advanced editing features of your preference.

Notepad++ is our most favorite source editor can be easy to indicate. Because is very powerful, supports several languages and is free editor.

You can download it from https:\/\/notepad-plus-plus.org\/download\/.

You will need setup your source code editor at menubar in Tools \ Preferences \ Module Editor.

0.0.1.

You also can also change your language settings for user interface messages.

Step 3

- To begin any project, is normal require to create a specific folder to contain your project, all source and all databases files to be contained.

0.0.2.

There are two ways to open a new project by our IDE:

1. By menubar File and select New Project
2. Or simple click to New Project button
3. Inform your project folder and give a name. In this case we can call the project as MyApp name.

You would use long file names but beware including spaces and special character in your full file name. Because our IDE is not supports special characters and spaces but in other hand Build.bat allows working with long names files.

- At this time, will be open and display two files:
- Main.prg file displayed at your favorite source code editor
- Main.fmg file displayed at Form Design board

Also are creating following files at your project folder:

1. MyApp.hbp
2. MyApp.hbc
3. MyApp.rc

You can see two files with Main name were created instead MyApp as it was defined for project name. In case you need to rename, you have two ways to do it:

1. Thru IDE

§ At menubar Project \/ Delete File and you can exclude both files by this way.

§ Now at menubar Project \/ New Module will creates a prg file with the right name given.

§ To create fmg file you click at menubar Project \/ New Form and will creates with right name

2. Outside of IDE

§ You will need to close project (not the IDE) by menubar File \/ Close Project.

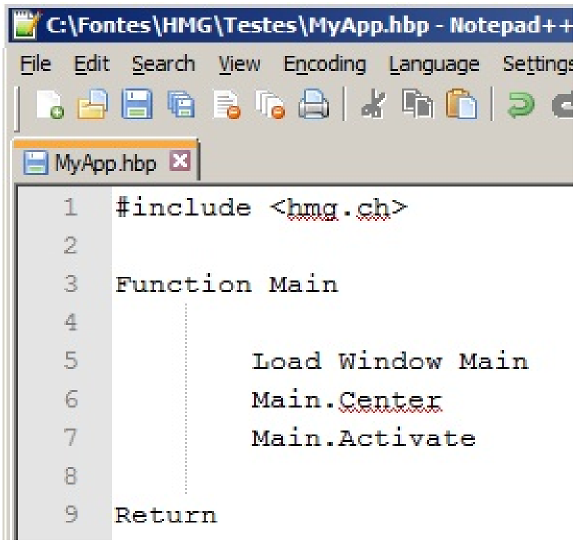
§ Thru Windows Explorer you can rename these files.

§ Will be necessary to re-edict MyApp.hbp and change Main name for MyApp and save the file.

§ Now you can open MyApp project

- From now you can edit your main prg file.

In this case MyApp.prg will appears with default source code, like this. This code is used for loading a previous fmg file done. If your project you make manually (without using fmg files) you can exclude following lines: **Load Window Main Main.Center Main.Activate** Then you write your own code. In case your fmg was previous renamed, you will need to replace Main by the new name.


C:\Fontes\HMG\Testes\MyApp.hbp - Notepad++
File Edit Search View Encoding Language Settings
MyApp.hbp

```
#include <hmg.ch>

Function Main

        Load Window Main
        Main.Center
        Main.Activate

Return
```

Step 4

- To add a new prgs just click at menubar **Project** \/ **New Module** will creates a prg file with the name given.
- In case you have others prgs files you can bring to actual project by clicking at menubar **Project** \/ **Import File**.

Step 5

- By double click on your main name in module tab of Project Browser, will open automatically for file editing.
- Then we can start to make MyApp demonstration. So at first line when we wish to work in GUI functions of HMG we have to include this as follows:

#include <hmg.ch>

This line is standard as GUI for all HMG modules. Unless you want build a project for console mode only, this line is not necessary.

Step 6

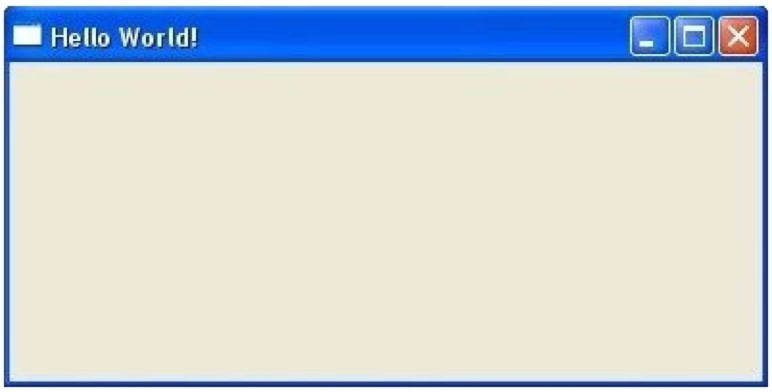
- Next, we'll add the **Main** function as follows:

```
Function Main()

DEFINE WINDOW Win_1 ;

AT 0,0 ;

WIDTH 400 ;

HEIGHT 200 ;

TITLE 'Hello World!' ;

MAIN

END WINDOW

ACTIVATE WINDOW Win_1

Return Nil
```

Step 7

- Save your program file and then build and run your project by selecting **Project** \ **Run** from menu or simply click "**Run**" button.

That's all !



0.0.3.

Analysing the source code demo:

Function Main statement should it be at every HMG project.

It must be with the name of "**Function Main**" or "**Procedure Main**" and could add parenthesis for parameters separated by comma.

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!' indicated text for window title at bar.

MAIN indicates that we are defining the main application window. Main window is required for all HMG applications.

ACTIVATE WINDOW Win_1 will show the window and start the event loop.

Return Nil will ends of function "main" return nothing.

This **";"** character indicates carriage return and used to facilitate to avoid code all in the same line. Breaking down lines helps for better understanding.

- none
  - none
    - 0.0.1.
    - 0.0.2.
    - 0.0.3.

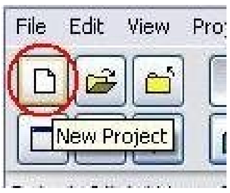
BROWSE is a control a few complicated than others. However HMG-IDE successfully support it.

In this sample we will build a Browse based application in a few steps using HMG-IDE.

Step - 1 : Make a folder for this sample.

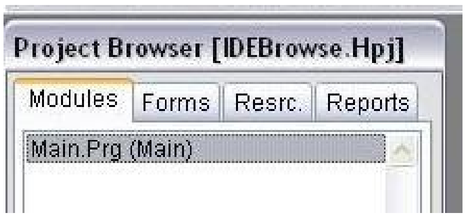
Step - 2 : Copy TEST.DBF and TEST.DBT from C:\hmg\SAMPLES\BROWSE_1 folder to your new folder.

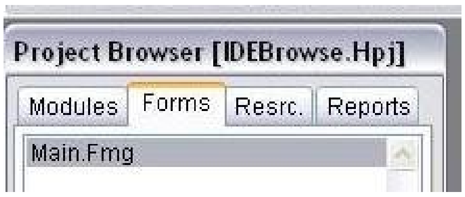
Step - 3 : Run HMG-IDE and chose "New Project".



IDE will ask you location and name for the new project; denote the folder you had build in the first step and give a name, say "IDEBrowse".

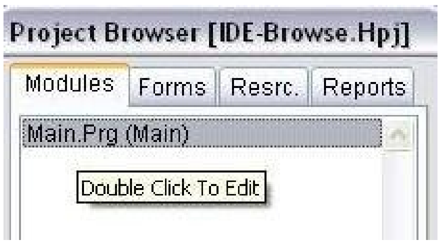
IDE will build a new main module named "Main.Prg" and a new main form named "Main.fmg"





Step - 4 :

Double click "Main.prg" in the "Modules" tab of "Project Browser" window for editing this "main" module:



IDE send this module (.prg) file to your programing editor. If you don't had denote an editor by IDE's "Tools\Preference" page, default editor is Window's NotePad.

You will see your main program written by IDE :

```
#include <hmg.ch>

Function Main

Load Window Main

Main.Center

Main.Activate

Return
```

Step - 5 :

Since BROWSE is a table based control, we must have a table and files opening\/closing rutins; these aren't duties of IDE.

So add this two little procedures to the main.prg for opening and closing table:

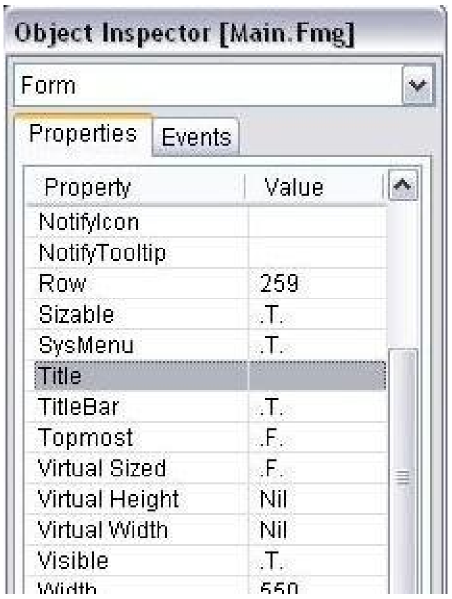
```
Procedure OpenTables()

Use Test Shared

Return Nil

Procedure CloseTables()

Use

Return Nil
```

Step - 6 :

Now, it's time for making some arrangements on the form:

Title :

- Click "Object Inspector [Main.Fmg]\Form\Properties\Title" :



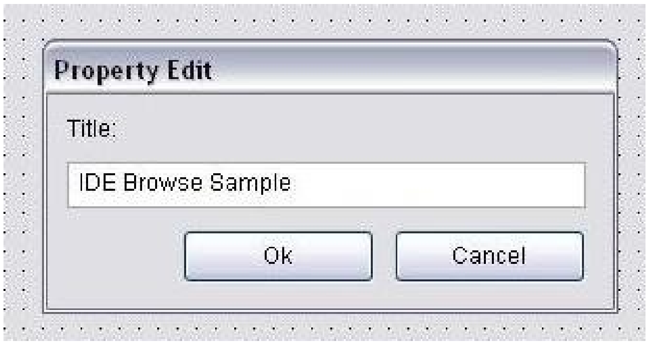
Object Inspector [Main.Fmg]

Form

Properties | Events

Property	Value
NotifyIcon	
NotifyTooltip	
Row	259
Sizable	.T.
SysMenu	.T.
Title	
TitleBar	.T.
Topmost	.F.
Virtual Sized	.F.
Virtual Height	Nil
Virtual Width	Nil
Visible	.T.
Width	550

- Give a "Title", say "**IDE Browse Sample**" to form :



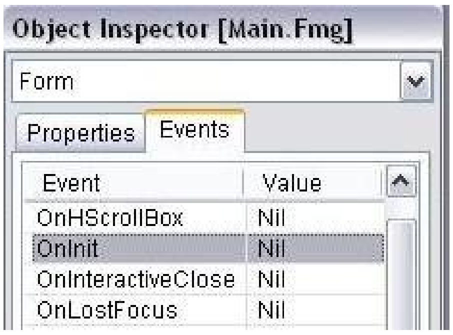
Property Edit

Title:

IDE Browse Sample

On Init Procedure :

- By clicking "Object Inspector [Main.Fmg]\Form\Events\OnInit" ;



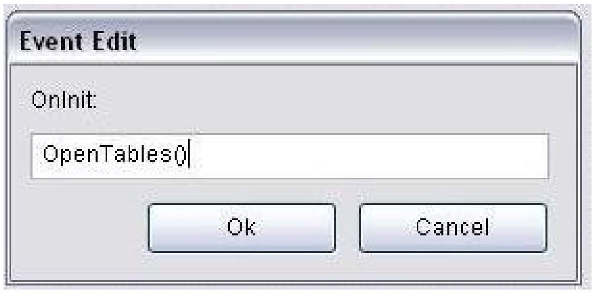
Object Inspector [Main.Fmg]

Form

Properties | Events

Event	Value
OnHScrollBox	Nil
OnInit	Nil
OnInteractiveClose	Nil
OnLostFocus	Nil

- Assign OpenTables() procedure name to this event:



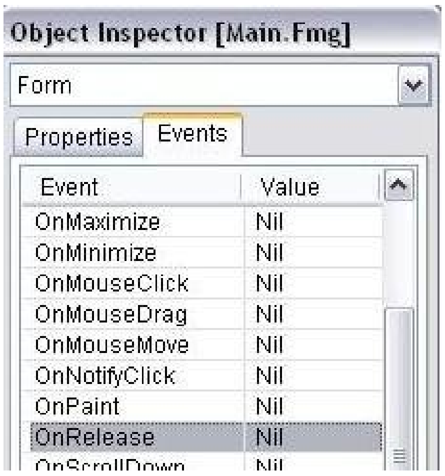
Event Edit

OnInit:

OpenTables()

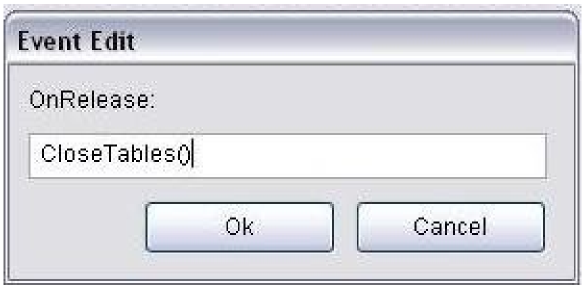
On Release Procedure :

- By clicking "Object Inspector [Main.Fmg]\Form\Events\OnRelease";



Event	Value
OnMaximize	Nil
OnMinimize	Nil
OnMouseClick	Nil
OnMouseDrag	Nil
OnMouseMove	Nil
OnNotifyClick	Nil
OnPaint	Nil
OnRelease	Nil

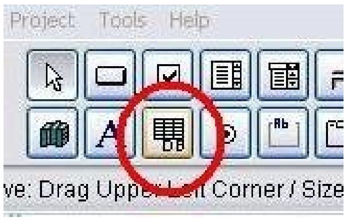
- Assign CloseTables() procedure name to this event:



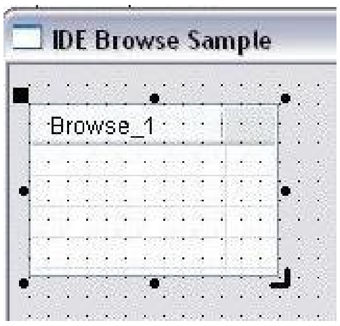
Step - 7 :

Now, we can place BROWSE control in our form:

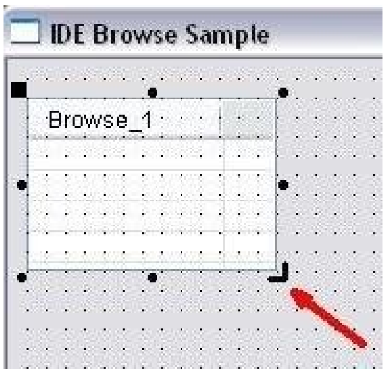
- Click BROWSE button in the Toolbox:



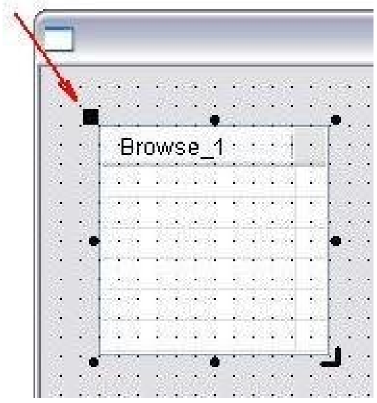
- Then click anywhere in the form :



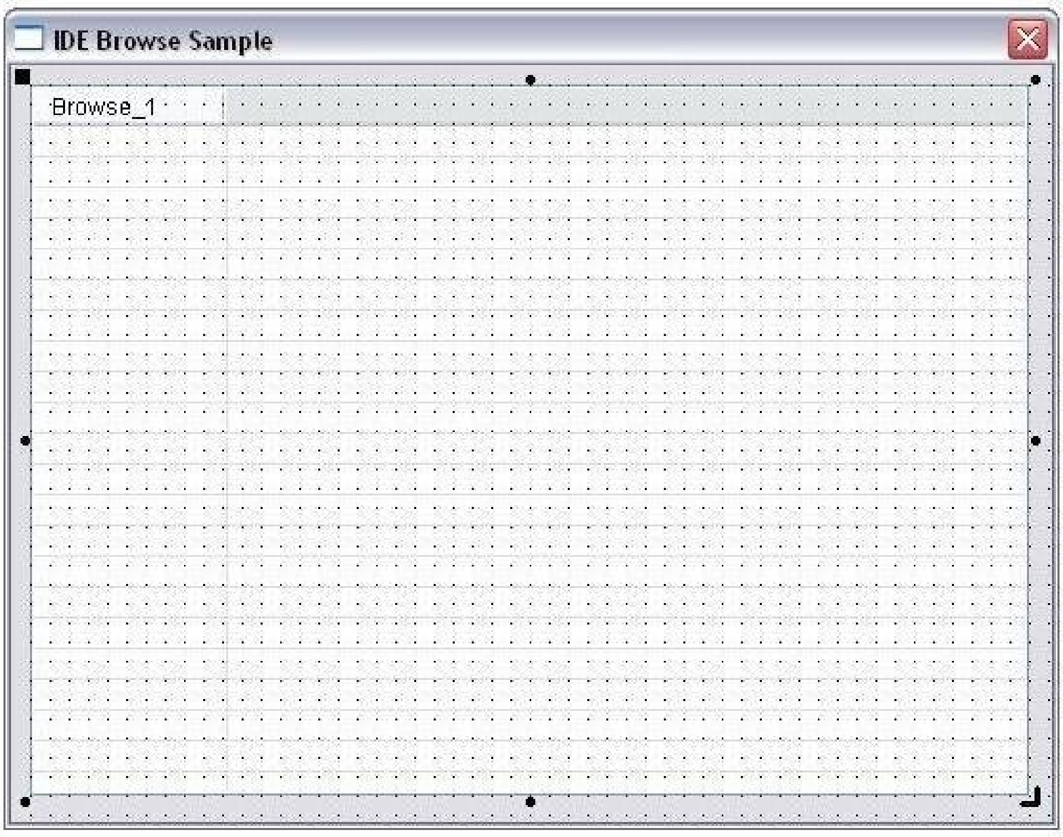
- Re-size control by dragging "resize" mark at the lower rigth corner :



- If required, change placement of control by using "replace" mark (at the upper-left corner)



- By using these two marks, you can place control in the form whatever you like:



Step - 8 :

And now, we will make some arrangements on the BROWSE control :

After activating (by clicking on it) BROWSE control;

By selecting "Object Inspector[Main.Fmg]\Browse_1\Properties" give these properties to Browse_1 control:

- "**Fields**" property :

{'Test->Code','Test->First','Test->Last','Test->Birth','Test->Married','Test->Bio'}

- "**Headers**" property :

{ 'Code', 'First Name', 'Last Name', 'Birth Date', 'Married', 'Biography' }

- "**Widths**" property :

  { 150, 150, 150, 150, 150, 150 }

- "**Work Area**" property :

Test

Note that "array" properties requires open and close curly-braces ('{}'), character properties doesn't requires quotation marks ("").

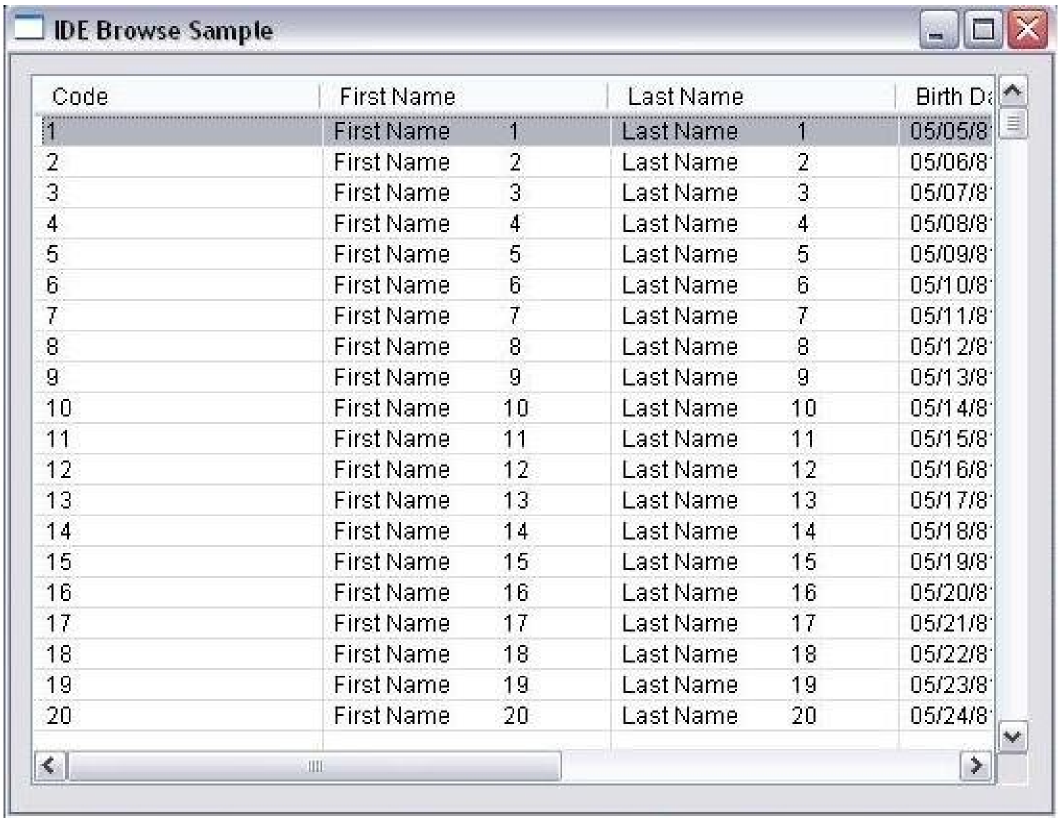
If you want allow editing abilities to user, you have change default values (.F.) of these properties to .T.

> ALLOWEDIT
>
> ALLOWAPPEND
>
> ALLOWDELETE

Last Step :

That is All !

Click "Run" button (or select "Run" from Project menu or simply press F5 key) for see results :



Code	First Name	Last Name	Birth Da
1	First Name 1	Last Name 1	05/05/8
2	First Name 2	Last Name 2	05/06/8
3	First Name 3	Last Name 3	05/07/8
4	First Name 4	Last Name 4	05/08/8
5	First Name 5	Last Name 5	05/09/8
6	First Name 6	Last Name 6	05/10/8
7	First Name 7	Last Name 7	05/11/8
8	First Name 8	Last Name 8	05/12/8
9	First Name 9	Last Name 9	05/13/8
10	First Name 10	Last Name 10	05/14/8
11	First Name 11	Last Name 11	05/15/8
12	First Name 12	Last Name 12	05/16/8
13	First Name 13	Last Name 13	05/17/8
14	First Name 14	Last Name 14	05/18/8
15	First Name 15	Last Name 15	05/19/8
16	First Name 16	Last Name 16	05/20/8
17	First Name 17	Last Name 17	05/21/8
18	First Name 18	Last Name 18	05/22/8
19	First Name 19	Last Name 19	05/23/8
20	First Name 20	Last Name 20	05/24/8

You have now a BROWSE based HMG application generated (almost) fully by IDE.

When you use "Run" command or attempt to closing your form by clicking "X" button, IDE ask you saving it (if any change you have made). You may also use File\Save Form menu command.

Since IDE is a "Two-Way Form Designer", you can edit .fmg files by double clicking its name in the "Forms" tab of "Project Browser" window.

And since .fmg files are pure HMG source code, you can open "main.fmg" by your editor for see form source code generated by IDE. Inspecting this code may be useful for better understanding and learning HMG.

You can also edit this file manually, but under your own risk ! Some modifications may not suitable to the standards of Form Designer.

Windows

DEFINE WINDOW

Creates a Window Definition

Standard Syntax (xBase Style):

```
DEFINE WINDOW <WindowName>
AT <nRow> ,<nCol>
WIDTH <nWindth>
HEIGHT <nHeight>
[ VIRTUAL WIDTH <nVirtualWindth> ]
[ VIRTUAL HEIGHT <nVirtualHeight> ]
[ TITLE <cTitle> ]
[ ICON <cIconName> ]
[ MAIN | CHILD | MODAL | SPLITCHILD | PANEL ]
[ NOSHOW ]
[ TOPMOST ]
[ NOAUTORELEASE ]
[ NOMINIMIZE ]
[ NOMAXIMIZE ]
[ NOSIZE ]
[ NOSYSMENU ]
[ NOCAPTION ]
[ CURSOR <CursorName> ]
[ ON INIT <InitProcedureName> | <bBlock> ]
[ ON RELEASE <ReleaseProcedureName> | <bBlock> ]
[ ON INTERACTIVECLOSE <InteractiveCloseProcedureName> | <bBlock> ]
[ ON MOUSECLICK <MouseClickProcedureName> | <bBlock> ]
[ ON MOUSEDRAG <MouseDragProcedureName> | <bBlock> ]
[ ON MOUSEMOVE <MouseMoveProcedureName> | <bBlock> ]
[ ON SIZE <WindowSizeProcedureName> | <bBlock> ]
[ ON MAXIMIZE <WindowMaximizeProcedureName> | <bBlock> ]
[ ON MINIMIZE <WindowMinimizeProcedureName> | <bBlock> ]
[ ON PAINT<WindowPaintProcedureName> | <bBlock> ]
[ BACKCOLOR <anBackColor> ]
[ FONT <cFontName> SIZE <nFontSize> ]
[ NOTIFYICON <cNotifyIconName> ]
[ NOTIFYTOOLTIP <cNotifyTooltip> ]
[ ON NOTIFYCLICK <NotifyClickProcedure> | <bBlock> ]
[ ON GOTFOCUS <ProcedureName> | <bBlock> ]
[ ON LOSTFOCUS <ProcedureName> | <bBlock> ]
[ ON SCROLLUP <ProcedureName> | <bBlock> ]
[ ON SCROLLDOWN <ProcedureName> | <bBlock> ]
[ ON SCROLLLEFT <ProcedureName> | <bBlock> ]
[ ON SCROLLRIGHT <ProcedureName> | <bBlock> ]
[ ON HSCROLLBOX <ProcedureName> | <bBlock> ]
[ ON VSCROLLBOX <ProcedureName> | <bBlock> ]
[ HELPBUTTON ]
[ GRIPPERTEXT <cGripperText> ]
[ BREAK ]
[ FOCUSED ]
... Control Definitions...
END WINDOW
```

Alternate Syntax:

```
DEFINE WINDOW <WindowName>
ROW <nRow>
COL <nCol>
WIDTH <nWindth>
HEIGHT <nHeight>
WINDOWTYPE MAIN | CHILD | MODAL | SPLITCHILD | STANDARD | PANEL
[ VIRTUALWIDTH <nVirtualWindth> ]
[ VIRTUALHEIGHT <nVirtualHeight> ]
[ TITLE <cTitle> ]
[ ICON <cIconName> ]
[ VISIBLE <lValue> ]
[ TOPMOST [<lValue>] ]
[ AUTORELEASE <lValue> ]
[ MINBUTTON <lvalue> ]
[ MAXBUTTON <lValue> ]
[ SIZABLE <lValue> ]
[ SYSMENU <lValue> ]
[ TITLEBAR <lValue> ]
[ CURSOR <CursorName> ]
[ ONINIT <InitProcedureName> | <bBlock> ]
[ ONRELEASE <ReleaseProcedureName> | <bBlock> ]
[ ONINTERACTIVECLOSE <InteractiveCloseProcedureName> | <bBlock> ]
[ ONMOUSECLICK <MouseClickProcedureName> | <bBlock> ]
[ ONMOUSEDRAG <MouseDragProcedureName> | <bBlock> ]
[ ONMOUSEMOVE <MouseMoveProcedureName> | <bBlock> ]
[ ONSIZE <WindowSizeProcedureName> | <bBlock> ]
[ ONMAXIMIZE <WindowMaximizeProcedureName> | <bBlock> ]
[ ONMINIMIZE <WindowMinimizeProcedureName> | <bBlock> ]
[ ONPAINT<WindowPaintProcedureName> | <bBlock> ]
[ BACKCOLOR <anBackColor> ]
[ FONTNAME <cFontName> FONTSIZE <nFontSize> ]
[ NOTIFYICON <cNotifyIconName> ]
[ NOTIFYTOOLTIP <cNotifyTooltip> ]
[ ONNOTIFYCLICK <NotifyClickProcedure> | <bBlock> ]
[ ONGOTFOCUS <ProcedureName> | <bBlock> ]
[ ONLOSTFOCUS <ProcedureName> | <bBlock> ]
[ ONSCROLLUP <ProcedureName> | <bBlock> ]
[ ONSCROLLDOWN <ProcedureName> | <bBlock> ]
[ ONSCROLLLEFT <ProcedureName> | <bBlock> ]
[ ONSCROLLRIGHT <ProcedureName> | <bBlock> ]
[ ONHSCROLLBOX <ProcedureName> | <bBlock> ]
[ ONVSCROLLBOX <ProcedureName> | <bBlock> ]
[ HELPBUTTON <lValue> ]
[ GRIPPERTEXT <cGripperText> ]
[ BREAK lValue ]
[ FOCUSED <lValue> ]

... Control Definitions...

END WINDOW
```

Focused, Break and GripperText properties are available only for 'SplitChild' type windows.

Topmost property is not available for modal and splitchild type windows.

Properties:

- Row
- Col
- Width
- Height
- Title
- NotifyIcon
- NotifyTooltip
- FocusedControl (R)
- Cursor (W)
- VirtualWidth (D)
- VirtualHeight (D)
- Icon (R)
- WindowType (D)
- Visible (D)
- AutoRelease (D)
- MinButton (D)
- MaxButton (D)
- Sizable (D)
- SysMenu (D)
- TitleBar (D)
- BackColor (D)
- FontName (D)
- FontSize (D)
- HelpButton (D)

- GripperText (D)
- Break (D)
- Focused (D)
- Topmost (D)
- Name (R)

(R) Read-Only (available after control definition via GetProperty function or semi-oop syntax)

(W) Write-Only (available after control definition via SetProperty function or semi-oop syntax)

(D) Available at definition only

Properties not tagged are available for read and write after control definition via SetProperty and GetProperty functions or semi-oop syntax.

Methods:

- Capture
- Print
- Show
- Hide
- Center
- Maximize
- Minimize
- Activate
- Restore
- Release
- SetFocus
- Redraw

Window Events:

- OnInit
- OnRelease
- OnInteractiveClose

- OnMouseClick
- OnMouseDrag
- OnMouseMove
- OnSize
- OnMaximize
- OnMinimize
- OnPaint
- OnNotifyClick
- OnGotFocus
- OnLostFocus
- OnScrollUp
- OnScrollDown
- OnScrollLeft
- OnScrollRight
- OnHscrollBox
- OnVscrollBox

New Properties:

```
ThisWindow|<FormName>.``Handle`` --> nFormHandle
ThisWindow|<FormName>.``Index`` --> nFormIndex
ThisWindow|<FormName>.``IsMinimized`` --> lBoolean
ThisWindow|<FormName>.``IsMaximized`` --> lBoolean
ThisWindow|<FormName>.``ClientAreaWidth`` --> nWidth
ThisWindow|<FormName>.``ClientAreaHeight`` --> nHeight
ThisWindow|<FormName>.``NoCaption`` [ := | --> ] lBoolean
ThisWindow|<FormName>.``NoMaximize`` [ := | --> ] lBoolean
ThisWindow|<FormName>.``NoMinimize`` [ := | --> ] lBoolean
ThisWindow|<FormName>.``NoClose`` [ := | --> ] lBoolean
ThisWindow|<FormName>.``NoSize`` [ := | --> ] lBoolean
ThisWindow|<FormName>.``NoSysMenu`` [ := | --> ] lBoolean
ThisWindow|<FormName>.``HScroll`` [ := | --> ] lBoolean
ThisWindow|<FormName>.``VScroll`` [ := | --> ] lBoolean
ThisWindow|<FormName>.``Enabled`` [ := | --> ] lBoolean
ThisWindow|<FormName>.``AlphaBlendTransparent`` := nAlphaBlend (0 to 255, Completely T
ransparent = 0, Opaque = 255)
ThisWindow|<FormName>.``BackColorTransparent`` := aRGBColor
ThisWindow|<FormName>.CenterWorkArea
ThisWindow|<FormName>.CenterIn ( FormName2 )
CENTER WINDOW <FormName> DESKTOP
CENTER WINDOW <FormName> IN <FormName2>
```

Visual Effects on Windows:

- **SET WINDOW** FormName **TRANSPARENT TO COLOR** aRGBColor
- **SET WINDOW** FormName **TRANSPARENT TO** nAlphaBlend --> nAlphaBlend = 0 to 255 (completely transparent = 0, opaque = 255)
- **SET WINDOW** FormName **[TRANSPARENT] TO OPAQUE**
- **FLASH WINDOW** FormName **CAPTION COUNT** nTimes **INTERVAL** nMilliseconds
- **FLASH WINDOW** FormName **TASKBAR COUNT** nTimes **INTERVAL** nMilliseconds
- **FLASH WINDOW** FormName **[ALL] COUNT** nTimes **INTERVAL** nMilliseconds
- **ANIMATE WINDOW** FormName **INTERVAL** nMilliseconds **MODE** nFlags
- **ANIMATE WINDOW** FormName **MODE** nFlags
- **MODE** <*nFlags*> (see the function AnimateWindow() in the API Window Reference), constants defined in **i_Window.ch**

#define AW_HOR_POSITIVE 0x00000001

#define AW_HOR_NEGATIVE 0x00000002

#define AW_VER_POSITIVE 0x00000004

#define AW_VER_NEGATIVE 0x00000008

#define AW_CENTER 0x00000010

#define AW_HIDE 0x00010000

#define AW_ACTIVATE 0x00020000

#define AW_SLIDE 0x00040000

#define AW_BLEND 0x00080000

Tips:

· SplitChild windows can be defined as part of a splitbox only.

· Toolbar's & SplitBox's parent window can't be a 'Virtual Dimensioned' window (use 'Virtual Dimensioned' splitchild's instead)

· NoAutoRelease Style: When this style is used, windows are hide instead released from memory when the user clicks in the close box.

· Using "Activate Window All" command at program startup will force "NoAutoRelease" style for all windows (excepting main).

· You must use "Show" and "Hide" methods to make a window visible or invisible.

· For virtual dimensioned Windows, two predefined controls are available.

· These controls are called vScrollBar and hScrollBar.

· Value property (read only) is available for these controls.

· ROW, COL, WIDTH and HEIGHT when any of those parameters are optional and when not been informed, default value as follows:

§ ROW and\/or COL: 0 (zero) position

§ WIDTH and\/or HEIGHT: maximum size according real desktop sizes.

Source code example:

```
#include "hmg.ch"
Function Main
SET AUTOSCROLL ON
DEFINE WINDOW Form_Main ;
   AT 0,0 ;
   WIDTH 640 ;
   HEIGHT 480 ;
   VIRTUAL WIDTH 1300 ;
   VIRTUAL HEIGHT 800 ;
   TITLE 'Virtual Dimensioned Window Demo' ;
   MAIN
   @ 100,10 BUTTON Button_1 ;
      CAPTION 'Vert. ScrollBar Value' ;
      ACTION MsgInfo( Str ( Form_Main.VScrollBar.Value ) ) ;
      WIDTH 150 HEIGHT 25
   @ 200,10 BUTTON Button_2 ;
      CAPTION 'Horiz. ScrollBar Value' ;
      ACTION MsgInfo( Str ( Form_Main.HScrollBar.Value ) ) ;
      WIDTH 150 HEIGHT 25
END WINDOW
ACTIVATE WINDOW Form_Main
Return Nil
```

ACTIVATE WINDOW ALL

Activates All Defined Windows

`ACTIVATE WINDOW ALL`

Using this command, all defined windows will be activated simultaneously.

All windows (excepting main) will be not visible at activation until show method be used to make them visible and NOAUTORELEASE style will be assumed for all non-main windows.

NoAutoRelease Style:

When this style is used, windows are hide instead released from memory when the user clicks in the close box.

You must use "Show" and "Hide" methods to make a window visible or invisible.

LOAD WINDOW

Load a window definition from a HMG window definition file

```
LOAD WINDOW <WindowDefinitionFileName> [ AS <WindowName> ]
```

If you omit the AS clause, the window definition file name (without extension) is assumed as the window name.

A HMG window definition file ('.fmg' file) is a text file containing a window definition, using standard HMG code, except the window name in the DEFINE WINDOW command (you must use TEMPLATE word instead).

DECLARE WINDOW

Declares a Window Name

```
DECLARE WINDOW <WindowName>
```

A window must be declared using this command, if you need to refer to it prior its definition in code, or when you refer to it in a different .prg file from the one it was defined using semi-oop syntax.

HMG Controls

@...ACTIVEX \/ DEFINE ACTIVEX

Creates an Activex container control

Standard Syntax (xBase Style):

```
@ <nRow>,<nCol> ACTIVEX <ControlName>
 [ OF | PARENT <ParentWindowName> ]
 WIDTH <nWidth>
 HEIGHT <nWidth>
 PROGID <cProgId>
```

Alternate Syntax:

```
DEFINE ACTIVEX <Name>
   PARENT <PropertyValue>
   ROW <PropertyValue>
   COL <PropertyValue>
   WIDTH <PropertyValue>
   HEIGHT <PropertyValue>
   OBJECT <PropertyValue>
END ACTIVEX
```

Properties:

- Row
- Col
- Width
- Height
- Object (R)
- Name (R)
- Parent (D)

  D: Available at control definition only

  R: Read-Only

@...ANIMATEBOX \/ DEFINE ANIMATEBOX

Creates an AnimateBox control

Standard Syntax (xBase Style):

```
@ <nRow> ,<nCol>
  ANIMATEBOX <ControlName>
  [ OF | PARENT <ParentWindowName> ]
  WIDTH <nWidth>
  HEIGHT <nJeight>
  [ FILE <cFileName> ]
  [ AUTOPLAY ]
  [ CENTER ]
  [ TRANSPARENT ]
  [ HELPID <nHelpId> ]
```

Alternate Syntax:

```
DEFINE ANIMATEBOX <ControlName>
   PARENT <xcValue>
   ROW <nValue>
   COL <nValue>
   WIDTH <nValue>
   HEIGHT <nValue>
   FILE <cValue>
   AUTOPLAY <lValue>
   CENTER <lValue>
   TRANSPARENT <lValue>
   HELPID <nValue>
   END ANIMATEBOX
```

Properties:

- Enabled
- Visible
- Row
- Width
- Height
- ToolTip
- Name
- Parent
- File
- AutoPlay
- Center
- Transparent

- HelpId D: Available at control definition only R: Read-Only

Methods:

```
  - Show
  - Hide
  - Open
  - Play
  - Seek
  - Stop
  - Close
  - Release
```

@...BROWSE \/ DEFINE BROWSE

Creates a browse control

Standard Syntax (xBase Style):

```
@ <nRow> ,<nCol>
 BROWSE <ControlName>
 [ OF | PARENT <ParentWindowName> ]
 WIDTH <nWidth>
 HEIGHT <nHeight>
 HEADERS <acHeaders>
 WIDTHS <anWidths>
 WORKAREA <WorkAreaName>
 FIELDS <acFields>
 [ VALUE <nValue> ]
 [ FONT <cFontname> SIZE <nFontsize> ]
 [ BOLD ] [ ITALIC ] [ UNDERLINE ] [ STRIKEOUT ]
 [ TOOLTIP <cToolTipText> ]
 [ DYNAMICBACKCOLOR <aDynamicBackColor> ]
 [ DYNAMICFORECOLOR <aDynamicBackColor> ]
 [ INPUTMASK <acInputMask> ]
 [ FORMAT <acFormat> ]
 [ INPUTITEMS <aInputItems> ]
 [ DISPLAYITEMS <aDisplayItems> ]
 [ BACKCOLOR <aBackColor> ]
 [ FONTCOLOR <aFontColor> ]
 [ ON GOTFOCUS <OnGotFocusProcedur> | <bBlock> ]
 [ ON CHANGE <OnChangeProcedure> | <bBlock> ]
 [ ON LOSTFOCUS <OnGotFocusProcedur> | <bBlock> ]
 [ [ ON DBLCLICK <OnDblClickProcedure> | <bBlock> ] | [ EDIT ] [ APPEND ] ]
 [ ON HEADCLICK <abBlock> ]
 [ WHEN <abBlock> ]
 [ VALID <abBlock> ]
 [ VALIDMESSAGES <acValidationMessages> ]
 [ READONLY <alReadOnlyFields> ]
 [ LOCK ]
 [ DELETE ]
 [ NOLINES ]
 [ IMAGE <acImageNames> ]
 [ JUSTIFY <anJustifyValue> ]
 [ NOVSCROLL ]
 [ HELPID <nHelpId> ]
 [ BREAK ]
 [ HEADERIMAGES <acHeaderImages> ]
```

Alternate Syntax:

```
DEFINE BROWSE <ControlName>
   PARENT <xcValue>
   ROW <nValue>
   COL <nValue>
   WIDTH <nValue>
   HEIGHT <nValue>
   HEADERS <acValue>
   WIDTHS <anValue>
   WORKAREA <xcValue>
   FIELDS <acValue>
   VALUE <nValue>
   FONTNAME <cValue>
   FONTSIZE <nValue>
   FONTBOLD <lValue>
   FONTITALIC <lValue>
   FONTUNDERLINE <lValue>
   FONTSTRIKEOUT <lValue>
   TOOLTIP <cValue>
   DYNAMICBACKCOLOR <abValue>
   DYNAMICFORECOLOR <abValue>
   INPUTMASK <acValue> ]
   FORMAT <acValues> ]
   INPUTITEMS <aaValues
   DISPLAYITEMS <aaValues>
   BACKCOLOR <anValue>
   FONTCOLOR <anValues
   ONGOTFOCUS <ActionProcedure>
   ONCHANGE <ActionProcedure
   ONLOSTFOCUS <ActionProcedure
   ONDBLCLICK <ActionProcedure>
   ALLWEDIT <lvalue>
   ALLWEAPPEND <lvalue>
   ON HEADCLICK <abBlock>
   WHEN <abBlock>
   VALID <abBlock>
   VALIDMESSAGES <acValues>
   READONLY <anValues>
   LOCK <lValue>
   ALLOWDELETE <lValue>
   LINES <lValue>
   IMAGE <acValue>
   JUSTIFY <anJustifyValue>
   VSCROLLBAR <lValue>
   HELPID <nValue>
   BREAK <lValue>
   HEADERIMAGES <acValue>
END BROWSE
```

Properties:

- AllowAppend

- AllowEdit
- AllowDelete
- Value
- Enabled
- Visible
- Row
- Col
- Width
- Height
- Header (nColumnIndex)
- HeaderImages (nColumnNumber)
- FontName
- FontSize
- FontBold
- FontItalic
- FontUnderline
- FontStrikeout
- ToolTip
- BackColor
- FontColor
- InputItems
- DisplayItems
- Name (R)
- Parent (D)
- Widths (D)
- Fields (D)
- WorkArea (D)
- Valid (D)
- ValidMessages (D)
- ReadOnlyFields (D)
- Lock (D)
- Lines (D)
- Image (D)
- Justify (D)
- VScrollBar (D)
- HelpId (D)
- When (D)
- DynamicBackColor (D)
- DynamicForeColor (D)
- InputMask (D)

- Format (D)

D: Available at control definition only R: Read-Only

Events:

- OnGotFocus
- OnChange
- OnLostFocus
- OnDblClick
- OnHeadClick

Methods:

- Show
- Hide
- SetFocus
- Refresh
- Release

Hints:

- Value property selects a record by its number (RecNo())
- Value property returns selected record number (recNo())
- Browse control does not change the active work area
- Browse control does not change the record pointer in any area (nor change selection when it changes) when SET BROWSESYNC is OFF (the default)
- You can programatically refresh it using refresh method.
- Variables called <MemVar>.<WorkAreaName>.<FieldName> are created for validation in browse editing window. You can use it in VALID array.
- Using APPEND clause you can add records to table associated with WORKAREA clause. The hotkey to add records is Alt+A. Append Clause Can't Be Used With Fields Not Belonging To Browse WorkArea
- Using DELETE clause allows to mark selected record for deletion pressing <Del> key
- The leftmost column in a browse control must be left aligned.
- When used in control definition, Header property must be loaded with a character array containing as elements as control columns.
- SET BROWSESYNC: When is set to ON, BROWSE control will move the record pointer in its workarea according to user selection or value property programatic setting.
- Setting 'Value' to reccount()+1 (EOF) will cause that browse windows get empty (no records will be showed).
- Vscrollbar can't be used with splitbox child browse.
- 'InputItems' property allows to control data input in the control. This property is an array (one element for each browse column). each element (if specified) must be a two

dimensional array. The first column in the array must contain the data to be shown to the user. the second column must contain the data to be stored in the table (ID) for each text row in the array.

- 'DisplayItems' property allows to control data display in the control. This property is an array (one element for each browse column). each element (if specified) must be a two dimensional array. the first column in the array must contain the text to be shown to the user. The second column must contain the ID for each array row. the array will be searched for a corresponding ID in the table to show the right text in each cell. If no correspondence is found, the cell will be blank.

@...BUTTON \/ DEFINE BUTTON

Creates a button control

Standard Syntax (xBase Style):

```
@ <nRow> ,<nCol>
BUTTON <ControlName>
[ OF | PARENT <ParentWindowName> ]
CAPTION <cCaption>
PICTURE <cPictureName> [ TOP | BOTTOM | LEFT | RIGHT ]
ACTION | ONCLICK | ON CLICK <ActionProcedureName> | <bBlock>
[ WIDTH <nWidth> HEIGHT <nHeight> ]
[ FONT <cFontName> SIZE <nFontSize> ]
[ BOLD ] [ ITALIC ] [ UNDERLINE ] [ STRIKEOUT ]
[ TOOLTIP <cToolTipText> ]
[ FLAT ]
[ NOTRANSPARENT ]
[ ON GOTFOCUS <OnGotFocusProcedur> | <bBlock> ]
[ ON LOSTFOCUS <OnLostFocusProcedure> | <bBlock> ]
[ NOTABSTOP ]
[ HELPID <nHelpId> ]
[ INVISIBLE ]
[ MULTILINE ]
```

Alternate Syntax:

```
DEFINE BUTTON <ButtonName>
   PARENT <ParentWindowName>
   ROW <nValue>
   COL <nValue>
   CAPTION <cValue>
   PICTURE <cValue>
   PICTALIGNMENT Top | Left | Right | Bottom
   ONCLICK <ActionProcedure>
   WIDTH <nValue>
   HEIGHT <nValue>
   FONTNAME <cValue>
   FONTSIZE <nValue>
   FONTBOLD <lValue>
   FONTITALIC <lValue>
   FONTUNDERLINE <lValue>
   FONTSTRIKEOUT <lValue>
   TOOLTIP <cValue>
   FLAT <lValue>
   TRANSPARENT <lValue>
   ONGOTFOCUS <ActionProcedure>
   ONLOSTFOCUS <ActionProcedure>
   TABSTOP <lValue>
   HELPID <nValue>
   VISIBLE <lValue>
   MULTILINE <lValue>
END BUTTON
```

Properties:

- Enabled
- Visible
- Row
- Col
- Width
- Height
- Caption
- FontName
- FontSize
- FontBold
- FontItalic
- FontUnderline
- FontStrikeout
- ToolTip
- Picture
- Name (R)
- Parent (D)

- Flat (D)
- TabStop (D)
- HelpId (D)
- Transparent (D)
- PictAlignment (D)
- MultiLine (D)

D: Available at control definition only R: Read-Only

Events:

- OnGotFocus
- OnLostFocus
- OnClick

Methods:

- Show
- Hide
- SetFocus
- Release

- Note: Transparence in picture buttons requires 256 or less color depth bitmaps.

@...CHECKBOX \/ DEFINE CHECKBOX

Creates a checkbox control

Standard Syntax (xBase Style):

```
@ <nRow> ,<nCol>
CHECKBOX <ControlName>
[ OF | PARENT <ParentWindowName> ]
CAPTION <cCaption>
[ WIDTH <nWidth>] [ HEIGHT <nHeight> ]
[ VALUE <lValue> ]
[ FIELD <FieldName> ]
[ FONT <cFontName> SIZE <nFontSize> ]
[ BOLD ] [ ITALIC ] [ UNDERLINE ] [ STRIKEOUT ]
[ TOOLTIP <cToolTipText> ]
[ BACKCOLOR <aBackColor> ]
[ FONTCOLOR <aFontColor> ]
[ ON GOTFOCUS <OnGotFocusProcedur> | <bBlock> ]
[ ON CHANGE <OnChangeProcedure> | <bBlock> ]
[ ON LOSTFOCUS <OnLostFocusProcedure> | <bBlock> ]
[ ON ENTER <OnEnterProcedure> | <bBlock> ]
[ TRANSPARENT ]
[ HELPID <nHelpId> ]
[ INVISIBLE ]
[ NOTABSTOP ]
```

Alternate Syntax:

```
DEFINE CHECKBOX <ControlName>
   PARENT <ParentWindowName>
   ROW <nValue>
   COL <nValue>
   CAPTION <cCaption>
   VALUE <lValue>
   FIELD <FieldName>
   WIDTH <nValue>
   HEIGHT <nValue>
   FONTNAME <cValue>
   FONTSIZE <nValue>
   FONTBOLD <lValue>
   FONTITALIC <lValue>
   FONTUNDERLINE <lValue>
   FONTSTRIKEOUT <lValue>
   TOOLTIP <cValue>
   BACKCOLOR <aBackColor>
   FONTCOLOR <aFontColor>
   ONGOTFOCUS <OnGotFocusProcedure>
   ONCHANGE <OnChangeProcedure>
   ONLOSTFOCUS <OnLostFocusProcedure>
   ONENTER <OnEnterProcedure>
   TRANSPARENT <lValue>
   HELPID <nHelpId>
   VISIBLE <lValue>
   TABSTOP <lValue>
END CHECKBOX
```

Properties:

- Value
- Enabled
- Visible
- Row
- Col
- Width
- Height
- Caption
- FontName
- FontSize
- FontBold
- FontItalic
- FontUnderline
- FontStrikeout
- ToolTip
- BackColor

- FontColor
- Name (R)
- Field (D)
- Parent (D)
- HelpId (D)
- TabStop (D)

D: Available at control definition only R: Read-Only

Events**:**

- OnGotFocus
- OnChange
- OnLostFocus
- OnEnter

Methods:

- Show
- Hide
- SetFocus
- Release
- Refresh
- Save

@...CHECKBUTTON \/ DEFINE CHECKBUTTON

Creates a CheckButton control

Standard Syntax (xBase Style):

```
@ <nRow> ,<nCol>
CHECKBUTTON<ControlName>
[ OF | PARENT <ParentWindowName> ]
CAPTION <cCaption> | PICTURE <cPictureName>
[ WIDTH <nWidth>] [ HEIGHT <nHeight> ]
[ VALUE <lValue> ]
[ FONT <cFontName> SIZE <nFontSize> ]
[ BOLD ] [ ITALIC ] [ UNDERLINE ] [ STRIKEOUT ]
[ TOOLTIP <cToolTipText> ]
[ ON GOTFOCUS <OnGotFocusProcedur> | <bBlock> ]
[ ON CHANGE <OnChangeProcedure> | <bBlock> ]
[ ON LOSTFOCUS <OnLostFocusProcedure> | <bBlock> ]
[ ON ENTER <OnEnterProcedure> | <bBlock> ]
[ HELPID <nHelpId> ]
[ INVISIBLE ]
[ NOTABSTOP ]
[ NOTRANSPARENT ]
```

Alternate Syntax:

```
DEFINE CHECKBUTTON <Controlname>
   PARENT <ParentWindowName>
   ROW <nValue>
   COL <nValue>
   CAPTION <cValue>
   WIDTH <nValue>
   HEIGHT <nValue>
   FONTNAME <cValue>
   FONTSIZE <nValue>
   FONTBOLD <lValue>
   FONTITALIC <lValue>
   FONTUNDERLINE <lValue>
   FONTSTRIKEOUT <lValue>
   TOOLTIP <cValue>
   ONGOTFOCUS <ActionProcedure>
   ONLOSTFOCUS <ActionProcedure>
   ONCHANGE <ActionProcedure>
   ON ENTER <OnEnterProcedure>
   TABSTOP <lValue>
   HELPID <nValue>
   VISIBLE <lValue>
   TRANSPARENT <lValue>
END CHECKBUTTON
```

Properties:

- Value
- Enabled
- Visible
- Row
- Col
- Width
- Height
- Caption
- FontName
- FontSize
- FontBold
- FontItalic
- FontUnderline
- FontStrikeout
- ToolTip
- Picture
- Name (R)
- Parent (D)
- HelpId (D)
- TabStop (D)

D: Available at control definition only R: Read-Only

Events:

- OnGotFocus
- OnChange
- OnLostFocus
- OnEnter

Methods:

- Show
- Hide
- SetFocus
- Release

Note: Transparence in picture checkbuttons requires 256 or less color depth bitmaps.

@...COMBOBOX \/ DEFINE COMBOBOX

Creates a ComboBox control

Standard Syntax (xBase Style):

```
@ <nRow> ,<nCol>
COMBOBOX <ControlName>
[ OF | PARENT <ParentWindowName> ]
[ ITEMS <caItems> ]
[ ITEMSOURCE <ItemSourceField1> [ , <ItemSourceField2> ] ]
[ VALUE <nValue> ]
[ VALUESOURCE <ValueSourceField> ]
[ DISPLAYEDIT ]
[ WIDTH <nWodth> ]
[ HEIGHT <nHeight>]
[ FONT <cFontName> SIZE <nFontSize> ]
[ BOLD ] [ ITALIC ] [ UNDERLINE ] [ STRIKEOUT ]
[ TOOLTIP <cToolTipText> ]
[ ON GOTFOCUS <OnGotFocusProcedur> | <bBlock> ]
[ ON CHANGE <OnChangeProcedure> | <bBlock> ]
[ ON LOSTFOCUS <OnLostFocusProcedure> | <bBlock> ]
[ ON ENTER <OnEnterProcedure> | <bBlock> ]
[ ON DISPLAYCHANGE <OnDisplayChangeProcedure> | <bBlock> ]
[ ON DROPDOWN <OnDropDownProcedure> | <bBlock> ]
[ ON CLOSEUP <OnCloseUpProcedure> | <bBlock> ]
[ NOTABSTOP ]
[ HELPID <nHelpId> ]
[ BREAK ]
[ GRIPPERTEXT <cGripperText> ]
[ INVISIBLE ]
[ SORT ]
[ IMAGE <acImageNames> ]
[ DROPPEDWIDTH <nDroppedWidth> ]
[ ON CANCEL <OnCancelProcedure> | <bBlock> ]
[ NOTRANSPARENT ]
```

Alternate Syntax:

```
DEFINE COMBOBOX <Controlname>
   PARENT <ParentWindowName>
   ROW <nValue>
   COL <nValue>
   WIDTH <nValue>
   HEIGHT <nValue>
   FONTNAME <cValue>
   FONTSIZE <nValue>
   FONTBOLD <lValue>
   FONTITALIC <lValue>
   FONTUNDERLINE <lValue>
   FONTSTRIKEOUT <lValue>
   TOOLTIP <cValue>
   ONGOTFOCUS <ActionProcedure>
   ONLOSTFOCUS <ActionProcedure>
   ONCHANGE <ActionProcedure>
   TABSTOP <lValue>
   HELPID <nValue>
   VISIBLE <lValue>
   ITEMS <caItems>
   ITEMSOURCE <ItemSourceField1> [ , <ItemSourceField2> ]
   VALUE <nValue>
   VALUESOURCE <ValueSourceField>
   DISPLAYEDIT <lValue>
   BREAK <lvalue>
   GRIPPERTEXT <cGripperText>
   SORT <lValue>
   IMAGE <acImageNames>
   DROPPEDWIDTH <nDroppedWidth>
   ONCANCEL <OnCancelProcedure>
   TRANSPARENT <lValue>
END COMBOBOX
```

Properties:

- Value
- Enabled
- Visible
- Item (nItemIndex)
- Items (D)
- ItemCount
- Row
- Col
- Width
- Height
- FontName
- FontSize

- FontBold
- FontItalic
- FontUnderline
- FontStrikeout
- ToolTip
- DisplayValue
- Name (R)
- ItemSource (D)
- Parent (D)
- HelpId (D)
- Sort (D)
- TabStop (D)
- DisplayEdit (D)
- Break (D)
- GripperText (D)
- Image (D)
- DroppedWidth (D)
- ValueSource (D)

D: Available at control definition only R: Read-Only

Events:

- OnGotFocus
- OnChange
- OnLostFocus
- OnDisplayChange
- OnEnter
- OnDropDown
- OnCloseUp
- OnCancel

Methods:

- Show
- Hide
- AddItem (cItemText)
- DeleteItem (nItemIndex)
- DeleteAllItems
- SetFocus
- Release

Hints:

- In a ComboBox the 'Height' clause refers to the total height (considering extended list height)
- When used in control definition, ITEM property must be a character array.
- When ITEMSOURCE property is set to a fieldname, 'Value' property uses the physical record number, as in browse.
- If you set the VALUESOURCE property to a fieldname, its content is returned instead the physical record number.
- 'DroppedWidth' property is used to set the dropdown list in a combobox control. 'DroppedWidth' can't be less that ComboBox width.
- OnDropDown Event will be executed when the user attempt to open dropdown list

Image Support:

- 'Image' Property specify a character array containing image file names or resource names.
- When you add an item, must specify the image array index number (zero based) and the text associated with it.
- When adding items at startup you must to use a two dimensional array.
- This array must have one row for each combo item and two columns.
- The first column must contain the image index and the second the text for the item.
- When using the additem or Item properties you must use a single array containing two elements. The first, the image index item and the second, the text for the item.
- When you retrieve the item, using the 'item' property, it will return a two elements array containing the image index and the text of the item.
- When 'Image' and 'ItemSource' properties are used simultaneously, 'ItemSource' must be specified as a list containing two field names.
- The first, the image index for the items, the second, the item text.
- 'Sort' and 'Image' can't be used simultaneously.

DEFINE CONTEXT MENU

Creates a Context Menu definition

Standard Syntax (xBase Style):

```
DEFINE CONTEXT MENU OF <ParentWindowName>
 MENUITEM <cItemCaption>
 ACTION <ActionProvedureName> | <bBlock>
 [ NAME <MenuItemName>]
 [ IMAGE <cImageName> ]
 [ CHECKED ]
 [ NOTRANSPARENT ]
 [ TOOLTIP <cToolTipText> ]

...
...
[ SEPARATOR ]
...
...
END MENU
```

Alternate Syntax:

```
DEFINE CONTEXTMENU PARENT <ParentWindowName>
   MENUITEM <cItemCaption>
   ONCLICK <ActionProvedure>
   [ NAME <MenuItemName>]
   [ IMAGE <cImageName> ]
   [ CHECKED <lValue> ]
   [ TRANSPARENT <lValue> ]
   [ TOOLTIP <cToolTipText> ]
 ...
 ...
   [ SEPARATOR ]
 ...
 ...
 END MENU
```

ContextMenu Properties:

- Parent (R)

R: Read-Only

MenuItem Properties:

- Name (R)
- Checked

- Enabled
- Image
- ToolTip

R: Read-Only

MenuItem Events:

- OnClick

You can **DEFINE\/RELEASE** Menu at runtime:

```
RELEASE CONTEXT MENU OF FormName
RELEASE CONTEXTMENU OF FormName
ReleaseContextMenu ( cFormName )
IsContextMenuDefined ( cFormName ) --> lBoolean
```

DEFINE CONTROL CONTEXT MENU

Add a Context Menu to a Control

Standard Syntax (xBase Style):

```
DEFINE CONTROL CONTEXT MENU <ControlName> [ OF <ParentWindowName> ]
MENUITEM <cItemCaption>
ACTION <ActionProvedureName> | <bBlock>
[ NAME <MenuItemName>]
[ IMAGE <cImageName> ]
[ CHECKED ]
[ NOTRANSPARENT ]
[ TOOLTIP <cToolTipText> ]
 ...
 ...
[ SEPARATOR ]
 ...
 ...
END MENU
```

Alternate Syntax:

```
DEFINE CONTROL CONTEXTMENU <ControlName> [ PARENT <ParentWindowName> ]
   MENUITEM <cItemCaption>
   ONCLICK <ActionProvedure>
   [ NAME <MenuItemName>]
   [ IMAGE <cImageName> ]
   [ CHECKED <lValue> ]
   [ TRANSPARENT <lValue> ]
   [ TOOLTIP <cToolTipText> ]
 ...
 ...
   [ SEPARATOR ]
 ...
 ...
END MENU
```

Control ContextMenu Properties:

- Parent (R)

  R: Read-Only

MenuItem Properties:

- Name (R)
- Checked
- Enabled

- Image
- ToolTip

R: Read-Only

MenuItem Events:

- OnClick

You can **DEFINE\/RELEASE** Menu at runtime:

```
RELEASE CONTROL CONTEXT MENU cControlName OF | PARENT cParentName
RELEASE CONTROL CONTEXTMENU cControlName OF | PARENT cParentName
ReleaseControlContextMenu ( cControlName, cParentForm )
IsControlContextMenuDefined ( cControlName, cParentForm ) --> Return lBoolean
```

Set On\/Off Control Context Menu:

```
SET CONTROL CONTEXTMENU [ ON | OFF ]
SET CONTROL CONTEXT MENU [ ON | OFF ]
```

@...DATEPICKER \/ DEFINE DATEPICKER

Creates a DatePicker control

Standard Syntax (xBase Style):

```
@ <nRow> ,<nCol>
DATEPICKER<ControlName>
[ OF | PARENT <cParentWindowName> ]
[ VALUE <dValue> ]
[ FIELD <FieldName> ]
[ WIDTH <nWidth> ]
[ HEIGHT <nHeight> ]
[ FONT <cFontName> SIZE <nFontSize> ]
[ BOLD ] [ ITALIC ] [ UNDERLINE ] [ STRIKEOUT ]
[ TOOLTIP <cToolTipText> ]
[ SHOWNONE ]
[ UPDOWN ]
[ RIGHTALIGN ]
[ ON GOTFOCUS <OnGotFocusProcedur> | <bBlock> ]
[ ON CHANGE <OnChangeProcedure> | <bBlock> ]
[ ON LOSTFOCUS <OnLostFocusProcedure> | <bBlock> ]
[ HELPID <nHelpId> ]
[ ON ENTER <OnEnterProcedure> | <bBlock> ]
[ INVISIBLE ]
[ NOTABSTOP ]
[ FORMAT <cDateFormat> ]
```

Alternate Syntax:

```
DEFINE DATEPICKER <ControlName>
   PARENT <ParentWindowName>
   ROW <nValue>
   COL <nValue>
   WIDTH <nValue>
   HEIGHT <nValue>
   FONTNAME <cValue>
   FONTSIZE <nValue>
   FONTBOLD <lValue>
   FONTITALIC <lValue>
   FONTUNDERLINE <lValue>
   FONTSTRIKEOUT <lValue>
   TOOLTIP <cValue>
   ONGOTFOCUS <ActionProcedure>
   ONLOSTFOCUS <ActionProcedure>
   ONCHANGE <ActionProcedure>
   TABSTOP <lValue>
   HELPID <nValue>
   VISIBLE <lValue>
   VALUE <dValue>
   SHOWNONE <lValue>
   UPDOWN <lValue>
   RIGHTALIGN <lValue>
   ONENTER <ActionProcedure>
   FORMAT <cDateFormat>
END DATEPICKER
```

Properties:

- Value
- Enabled
- Visible
- Row
- Col
- Width
- Height
- FontName
- FontSize
- FontBold
- FontItalic
- FontUnderline
- FontStrikeout
- ToolTip
- Name (R)
- Field (D)
- Parent (D)

- ShowNone (D)
- UpDown (D)
- RightAlign (D)
- HelpId (D)
- TabStop (D)

D: Available at control definition only R: Read-Only

Events:

- OnGotFocus
- OnChange
- OnLostFocus

Methods:

- Show
- Hide
- SetFocus
- Release
- Save
- Refresh

Note: description of the **cDateFormat** string

```
"d" The one- or two-digit day.
"dd" The two-digit day. Single-digit day values are preceded by a zero.
"ddd" The three-character weekday abbreviation.
"dddd" The full weekday name.
"M" The one- or two-digit month number.
"MM" The two-digit month number. Single-digit values are preceded by a zero.
"MMM" The three-character month abbreviation.
"MMMM" The full month name.
"yy" The last two digits of the year (that is, 1996 would be displayed as "96").
"yyyy" The full year (that is, 1996 would be displayed as "1996").
```

Example:

```
"d.MM.yyyy" will display date as 9.02.2012
"dd.MM.yyyy" will display date as 09.02.2012
"dd/MM/yyyy" will display date as 16/02/2012
"dd.MMM.yyyy" will display date as 16. FEB. 2012
```

- To make the information more readable, you can add body text to the format string by enclosing it in single quotes.
- Spaces and punctuation marks do not need to be quoted. Nonformat characters that are

not delimited by single quotes will result in unpredictable display by the DATEPICKER control.

- For example, to display the current date with the format "'Today is: 04:22:31 Tuesday Mar 23, 1996", the format string is "'Today is: 'hh':'m':'s dddd MMM dd', 'yyyy".
- To include a single quote in your body text, use two consecutive single quotes.
- For example, "'Don''t forget' MMM dd',' yyyy" produces output that looks like:
- Do not forget **Mar 23, 1996**. It is not necessary to use quotes with the comma, so "'Don''t forget' MMM dd, yyyy" is also valid, and produces the same output.

DEFINE DROPDOWN MENU

Creates a dropdown menu definition

Standard Syntax (xBase Style):

```
DEFINE DROPDOWN MENU BUTTON <ToolBarDropDownButtonName> [ OF <ParentWindowName> ]
   MENUITEM <cItemCaption>
   ACTION <ActionProvedureName> | <bBlock>
   [ NAME <MenuItemName>]
   [ IMAGE <cImageName> ]
   [ CHECKED ]
   [ NOTRANSPARENT ]
   [ TOOLTIP <cToolTipText> ]
   ...
   ...
      [ SEPARATOR ]
    ...
    ...
END MENU
```

Alternate Syntax:

```
DEFINE DROPDOWNMENU OWNERBUTTON <ParentToolBarButtonName> [ PARENT <ParentWindowName>
]
   MENUITEM <cItemCaption>
   ONCLICK <ActionProvedureName>
   [ NAME <MenuItemName>]
   [ IMAGE <cImageName> ]
   [ CHECKED <lValue> ]
   [ TRANSPARENT <lValue> ]
   [ TOOLTIP <cToolTipText> ]
   ...
   ...
   [ SEPARATOR ]
   ...
   ...
END MENU
```

DropDown Menu Properties:

- Parent (R)
- OwnerButton

R: Read-Only

MenuItem Properties:

- Name (R)

- Checked
- Enabled
- Image
- ToolTip

R: Read-Only

MenuItem Events:

- OnClick

You can **DEFINE\/RELEASE** Menu at runtime:

```
RELEASE DROPDOWN MENU BUTTON ButtonName OF FormName
RELEASE DROPDOWNMENU OWNERBUTTON ButtonName OF FormName
ReleaseDropDownMenu ( cButtonName, cFormName )
IsDropDownMenuDefined ( cButtonName, cFormName ) --> lBoolean
```

@...EDITBOX \/ DEFINE EDITBOX

Creates an EditBox control

Standard Syntax (xBase Style):

```
@ <nRow> ,<nCol>
EDITBOX<ControlName>
[ OF | PARENT <ParentWindowName> ]
WIDTH <nWidth>
HEIGHT <nHeight>
[ FIELD <FieldName> ]
[ VALUE <cValue> ]
[ READONLY ]
[ FONT <cFontName> SIZE <nFontSize> ]
[ BOLD ] [ ITALIC ] [ UNDERLINE ] [ STRIKEOUT ]
[ TOOLTIP <cToolTipText> ]
[ BACKCOLOR <aColor> ]
[ FONTCOLOR <aColor> ]
[ DISABLEDBACKCOLOR <aColor> ]
[ DISABLEDFONTCOLOR <aColor> ]
[ MAXLENGTH <nInputLength> ]
[ ON GOTFOCUS <OnGotFocusProcedur> | <bBlock> ]
[ ON CHANGE <OnChangeProcedure> | <bBlock> ]
[ ON LOSTFOCUS <OnLostFocusProcedure> | <bBlock> ]
[ HELPID <nHelpId> ]
[ BREAK ]
[ INVISIBLE ]
[ NOTABSTOP ]
[ NOVSCROLL ]
[ NOHSCROLL ]
```

Alternate Syntax:

```
DEFINE EDITBOX <ControlName>
   PARENT <ParentWindowName>
   ROW <nValue>
   COL <nValue>
   WIDTH <nValue>
   HEIGHT <nValue>
   FONTNAME <cValue>
   FONTSIZE <nValue>
   FONTBOLD <lValue>
   FONTITALIC <lValue>
   FONTUNDERLINE <lValue>
   FONTSTRIKEOUT <lValue>
   TOOLTIP <cValue>
   ONGOTFOCUS <ActionProcedure>
   ONLOSTFOCUS <ActionProcedure>
   ONCHANGE <ActionProcedure>
   TABSTOP <lValue>
   HELPID <nValue>
   VISIBLE <lValue>
   FIELD <FieldName>
   VALUE <cValue>
   READONLY <lValue>
   BREAK <lValue>
   VSCROLLBAR <lValue>
   HSCROLLBAR <lValue>
   BACKCOLOR <aColor>
   FONTCOLOR <aColor>
   DISABLEDBACKCOLOR <aColor>
   DISABLEDFONTCOLOR <aColor>
END EDITBOX
```

Properties:

- Value
- Enabled
- Visible
- Row
- Col
- Width
- Height
- FontName
- FontSize
- FontBold
- FontItalic
- FontUnderline
- FontStrikeout
- ToolTip

- BackColor
- FontColor
- DisabledBackColor
- DisabledFontColor
- CaretPos
- Name (R)
- Field (D)
- Parent (D)
- ReadOnly
- MaxLength (D)
- HelpId (D)
- Break (D)
- TabStop (D)
- VScrollBar (D)
- HScrollBar (D)
- GetTextLength

D: Available at control definition only R: Read-Only

Events:

- OnGotFocus
- OnChange
- OnLostFocus

Methods:

- Show
- Hide
- SetFocus
- Release
- Refresh
- Save

@...FRAME \/ DEFINE FRAME

Creates a frame control

Standard Syntax (xBase Style):

```
@ <nRow> ,<nCol>
FRAME<ControlName>
[ OF | PARENT <ParentWindowName> ]
[ CAPTION <cCaption> ]
WIDTH <nWidth>
HEIGHT <nHeight>
[ FONT <cFontName> ]
[ SIZE <nFontSize> ]
[ BOLD ]
[ ITALIC ]
[ UNDERLINE ]
[ STRIKEOUT ]
[ BACKCOLOR <aBackColor> ]
[ FONTCOLOR <aFontColor> ]
[ TRANSPARENT ]
```

Alternate Syntax:

```
DEFINE FRAME <ControlName>
   PARENT <ParentWindowName>
   ROW <nValue>
   COL <nValue>
   WIDTH <nValue>
   HEIGHT <nValue>
   FONTNAME <cValue>
   FONTSIZE <nValue>
   FONTBOLD <lValue>
   FONTITALIC <lValue>
   FONTUNDERLINE <lValue>
   FONTSTRIKEOUT <lValue>
   CAPTION <cCaption>
   BACKCOLOR <aBackColor>
   FONTCOLOR <aFontColor>
   TRANSPARENT <lValue>
END FRAME
```

Properties:

- Enabled
- Visible
- Row
- Col
- Width

- Height
- Caption
- FontName
- FontSize
- FontBold
- FontItalic
- FontUnderline
- FontStrikeout
- BackColor
- FontColor
- Name (R)
- Parent (D)
- Transparent (D)

D: Available at control definition only R: Read-Only

Methods:

- Show
- Hide
- Release

Note: FontColor property works only for 'Windows Clssic' \/ 'Windows Standard' themes.

@...GRID \/ DEFINE GRID

Creates a grid control

Standard Syntax (xBase Style):

```
  @ <nRow> ,<nCol>
  GRID <ControlName>
  [ OF | PARENT <ParentWindowName> ]
  WIDTH <nWidth>
  HEIGHT <nHeight>
  HEADERS <acHeaders>
  WIDTHS <anWidths>
  [ ITEMS <acItems> ]
  [ VALUE <nValue> ]
  [ FONT <cFontname> SIZE <nFontsize> ]
  [ BOLD ] [ ITALIC ] [ UNDERLINE ] [ STRIKEOUT ]
  [ TOOLTIP <cToolTipText> ]
  [ BACKCOLOR <aBackColor> ]
  [ FONTCOLOR <aFontColor> ]
  [ DYNAMICBACKCOLOR <aDynamicBackColor> ]
  [ DYNAMICFORECOLOR <aDynamicBackColor> ]
  [ ON GOTFOCUS <OnGotFocusProcedur> | <bBlock> ]
  [ ON CHANGE <OnChangeProcedure> | <bBlock> ]
  [ ON LOSTFOCUS <OnGotFocusProcedure> | <bBlock> ]
  [ [ ON DBLCLICK <OnDblClickProcedure> | <bBlock> ] |
  [ EDIT | ALLOWEDIT ] ]
  [ ON SAVE <OnSaveProcedure> | <bBlock> ]  [ COLUMNCONTROLS {aControlDef1,aControlDef2
 ,...aControlDefN}
  [ COLUMNVALID {bValid1,bValid2,...bValidN}
  [ COLUMNWHEN {bWhen1,bWhen2,...bWhenN}
  [ ON HEADCLICK <abBlock> ]
  [ VIRTUAL ]
  [ ITEMCOUNT <nItemCount> ]
  [ ON QUERYDATA <OnQueryDataProcedure> | <bBlock> ]
  [ MULTISELECT ]
  [ NOLINES ]
  [ NOHEADERS ]
  [ IMAGE <acImageNames> ]
  [ JUSTIFY <anJustifyValue> ]
  [ HELPID <nHelpId> ]
  [ BREAK ]
  [ HEADERIMAGES <acHeaderImages> ]
  [ CELLNAVIGATION ]
  [ ROWSOURCE <cRowSource>]
  [ COLUMNFIELDS <acColumnFields> ]
  [ ALLOWAPPEND ]
  [ ALLOWDELETE ]
  [ DYNAMICDISPLAY <abDynamicDisplay> ]
  [ LOCKCOLUMNS <nValue> ]
  [ ON CLICK <OnClickProcedure> ]
  [ ON KEY <OnKeyProcedure> ]
  [ ON CHECKBOXCLICKED <OnCheckBoxClickedProcedure> ]
  [ NOTRANSPARENT ]
  [ NOTRANSPARENTHEADER ]
  [ ON INPLACEEDITEVENT <OnInplaceEditEventProcedure> ]
  [ EDITOPTION <nEditOption> ]
```

Note: nEditOption --> GRID_EDIT_DEFAULT | GRID_EDIT_SELECTALL | GRID_EDIT_INSERTBLANK | GRID_EDIT_INSERTCHAR | GRID_EDIT_REPLACEALL

Alternate Syntax:

```
DEFINE GRID <ControlName>
   PARENT <ParentWindowName>
   ROW <nValue>
   COL <nValue>
   WIDTH <nValue>
   HEIGHT <nValue>
   FONTNAME <cValue>
   FONTSIZE <nValue>
   FONTBOLD <lValue>
   FONTITALIC <lValue>
   FONTUNDERLINE <lValue>
   FONTSTRIKEOUT <lValue>
   TOOLTIP <cValue>
   ONGOTFOCUS <ActionProcedure>
   ONLOSTFOCUS <ActionProcedure>
   ONSAVE <ActionProcedure> ONCHANGE <ActionProcedure>
   TABSTOP <lValue>
   HELPID <nValue>
   VISIBLE <lValue>
   ITEMS <caItems>
   HEADERS <acHeaders>
   WIDTHS <anWidths>
   VALUE <nValue>
   BACKCOLOR <aBackColor>
   FONTCOLOR <aFontColor>
   DYNAMICBACKCOLOR <aDynamicBackColor>
   DYNAMICFORECOLOR <aDynamicBackColor>
   ONDBLCLICK <OnDblClickProcedure>
   ALLOWEDIT <lValue>
   COLUMNCONTROLS {aControlDef1,aControlDef2,...aControlDefN}
   COLUMNVALID {bValid1,bValid2,...bValidN}
   COLUMNWHEN {bWhen1,bWhen2,...bWhenN}
   ONHEADCLICK <abBlock>
   VIRTUAL <lValue>
   ITEMCOUNT <nItemCount>
   ONQUERYDATA <OnQueryDataProcedure>
   MULTISELECT <lValue>
   LINES <lValue>
   SHOWHEADERS <lValue>
   IMAGE <acImageNames>
   JUSTIFY <anJustifyValue>
   HEADERIMAGES <acHeaderImages>
   CELLNAVIGATION <lValue>
   ROWSOURCE <cRowSource>]
   COLUMNFIELDS <acColumnFields>
   ALLOWAPPEND <lValue>
   ALLOWDELETE <lValue>
```

```
    DYNAMICDISPLAY <abDynamicDisplay>
    LOCKCOLUMNS <nValue>
    ONCLICK <OnClickProcedure>
    ONKEY <OnKeyProcedure>
    ONCHECKBOXCLICKED <OnCheckBoxClickedProcedure>
    ONINPLACEEDITEVENT <OnInplaceEditEventProcedure>
    EDITOPTION <nEditOption>
    TRANSPARENT <lValue>
    TRANSPARENTHEADER <lValue>
END GRID
```

Properties:

- RowSource
- ColumnFields
- AllowAppend
- AllowDelete
- DynamicDisplay
- RecNo
- ColumnWhen
- DynamicBackColor
- DynamicForeColor
- Cell (nRow , nCol)
- Value
- Enabled
- Visible
- Item (nItemIndex)
- ItemCount
- Row
- Col
- Width
- Height
- FontName
- FontSize
- FontBold
- FontItalic
- FontUnderline
- FontStrikeout
- ToolTip
- BackColor
- FontColor
- Header (nColumnNumber)
- HeaderImages (nColumnNumber)

- Name (R)
- Virtual (D)
- Parent (D)
- Widths
- MultiSelect (D)
- NoLines (D)
- Image
- Justify
- HelpId (D)
- Break (D)
- AllowEdit (D)
- ColumnControls
- ColumnValid
- Headers
- CellNavigation (D)
- Items (D)
- LockColumns (D)

D: Available at control definition only R: Read-Only

Properties Available For OnQueryData Procedure:

```
This.QueryData
This.QueryRowIndex
This.QueryColIndex
```

Properties Available For OnDblClick Procedure:

```
This.CellRowIndex
This.CellColIndex
This.CellRow
This.CellCol
This.CellWidth
This.CellHeight
```

- Note: These properties are not available when **OnDblClick** procedure is fired by <Enter> key pressing.

Properties Available For DynamicBackColor \/ DynamicForeColor \/ ColumnValid Processing:

```
This.CellRowIndex
This.CellColIndex
This.CellValue
```

Properties Available For OnSave Procedure:

```
This.AppendBuffer
This.EditBuffer
This.MarkBuffer
```

- This.EditBuffer: Array of one element per edited cell.

The elements has the following structure: { nLogicalRow , nLogicalCol , xValue , nRecNo }

- This.AppendBuffer: Array of one element per appended record. The elements has the following structure: { xFieldValue 1 , ... , xFieldValue n }
- This.MarkBuffer: Array of one element per record marked to be deleted or recalled. The elements has the following structure: { nLogicalRow , nRecNo , cMark ('D' or 'R') }

Properties Available For OnInplaceEditEvent Procedure:

```
This.IsInplaceEditEventInit --> .T. or .F.
This.IsInplaceEditEventRun --> .T. or .F.
This.IsInplaceEditEventFinish --> .T. or .F.
This.InplaceEditGridName --> eg. “Grid_1”
This.InplaceEditParentName --> eg. “Form_1”
This.InplaceEditControlHandle --> Handle of InplaceEdit ColumnControl, eg. Handle of T
EXTBOX, DATEPICKER, TIMEPICKER, COMBOBOX, SPINNER, CHECKBOX, etc.
This.InplaceEditControlIndex --> Return nControlIndex
```

Events***:***

- OnGotFocus
- OnChange
- OnLostFocus
- OnDblClick
- OnHeadClick
- OnQueryData
- OnSave
- OnClick
- OnKey
- OnCheckBoxClicked
- OnInplaceEditEvent

Methods:

- Show
- Append
- Save
- Refresh([lValue])
- Delete
- Recall
- Hide
- AddItem (acItemText)
- ClearBuffer
- DeleteItem (nItemIndex)
- DeleteAllItems
- SetFocus
- DisableUpdate
- EnableUpdate
- Release
- AddColumn ([nColIndex] , [cCaption] , [nWidth] , [nJustify])
- DeleteColumn (nColIndex)

- Hints:

- The leftmost column in a grid control must be left aligned.
- When used in control definition, Header property must be loaded with a character array containing as elements as control columns.
- When AddColumn \/ DeleteColumn methods are used, all items in grid (if any) will be lost.
- If MULTISELECT clause is used VALUE must be a numeric array, containing the index position of selected items.
- If EDIT clause is used, by doubleclicking an item, will open an editing window allowing to change the item content.
- EDIT and MULTISELECT clauses can't be used simultaneously.
- When RowSource is specified, the workarea specified, must be open at control definition.
- Hotkey when RowSource is specified:

  ALT + A --> Append

  ALT + D --> Delete

ALT + R --> ReCall

ALT + S --> Save (On Save)

ALT + U --> ClearBuffer

Control Definition Array:

TEXTBOX

```
{ cControlType , cDataType , cInputMask , cFormat }
cControlType = 'TEXTBOX' (Required)
cDataType = 'CHARACTER' , 'NUMERIC' , 'DATE' (Required)
cInputMask = cInputMask (Optional)
cFormat = cFormat (Optional)
```

DATEPICKER

```
{ cControlType , cControlStyle }
cControlType = 'DATEPICKER' (Required)
cControlStyle = 'DROPDOWN' , 'UPDOWN' (Required)
```

COMBOBOX

```
{ cControlType , acItems , axReturnValues }
cControlType 'COMBOBOX' (Required)
acItems (Required)
axReturnValues (Optional)
```

SPINNER

```
{ cControlType , nRangeMin , nRangeMax }
cControlType 'SPINNER' (Required)
nRangeMin (Required)
nRangeMax (Required)
```

CHECKBOX

```
{ cControlType , cCheckedLabel , cUnCheckedLabel }
cControlType 'CHECKBOX' (Required)
cCheckedLabel (Required)
cUnCheckedLabel (Required)
```

Note: Data type for each column will depend control specified.

```
NUMERIC TEXTBOX : NUMERIC
DATE TEXTBOX : DATE
CHARACTER TEXTBOX : CHARACTER
SPINNER : NUMERIC
COMBOBOX : NUMERIC (Any type if axReturnValues is specified)
CHECKBOX : LOGICAL
```

Sample:

```
@ 10,10 GRID Grid_1 ;
WIDTH 620 ;
HEIGHT 330 ;
HEADERS {'Column 1','Column 2','Column 3','Column 4',;
'Column 5'} ;
WIDTHS {140,140,140,140,140} ;
ITEMS aRows ;
EDIT ;
COLUMNCONTROLS { {'TEXTBOX','NUMERIC','$ 999,999.99'},;
{'DATEPICKER','DROPDOWN'} ,;
{'COMBOBOX',{'One','Two','Three'}} , ;
{ 'SPINNER' , 1 , 20 } , ;
{ 'CHECKBOX' , 'Yes' , 'No' } }
```

Justify constants:

```
GRID_JTFY_LEFT
GRID_JTFY_RIGHT
GRID_JTFY_CENTER
```

GRID Control improvement

New Features:

Grid Grouping:

```
<ParentWindowName>.<GridControlName>.GroupEnabled [ := | -->] lBoolean
Note: Grid Group is not available when application is running on Windows versions of 3
2-bits
You can check if your application is running on Win32 with the function HMG_IsRunAppIn
Win32()
<ParentWindowName>.<GridControlName>.GroupDeleteAll
<ParentWindowName>.<GridControlName>.GroupDelete ( nGroupID )
<ParentWindowName>.<GridControlName>.GroupDeleteAllItems ( nGroupID )
<ParentWindowName>.<GridControlName>.GroupExist ( nGroupID ) --> lBoolean
<ParentWindowName>.<GridControlName>.GroupCheckBoxAllItems ( nGroupID ) := lBoolean
<ParentWindowName>.<GridControlName>.GroupGetAllItemIndex ( nGroupID ) --> anItemIndex
<ParentWindowName>.<GridControlName>.GroupExpand ( nGroupID )
<ParentWindowName>.<GridControlName>.GroupCollapsed ( nGroupID )
<ParentWindowName>.<GridControlName>.GroupAdd ( nGroupID [, nPosition ] )
<ParentWindowName>.<GridControlName>.GroupInfo ( nGroupID ) [ := | -->] { [ cHeader ]
, [ nAlignHeader ] , [ cFooter ] , [ nAlingFooter ] , [ nState ] }
<ParentWindowName>.<GridControlName>.GroupItemID ( nItem ) [ := | -->] nGroupID

Note: nAlignHeader [ := | -->] GRID_GROUP_LEFT | GRID_GROUP_CENTER | GRID_GROUP_RIGHT
nAlingFooter [ := | -->] GRID_GROUP_LEFT | GRID_GROUP_CENTER | GRID_GROUP_RIGHT
nState [ := | -->] GRID_GROUP_NORMAL | GRID_GROUP_COLLAPSED
```

Gird CheckBox:

```
<ParentWindowName>.<GridControlName>.CheckBoxEnabled [ := | -->] lBoolean
<ParentWindowName>.<GridControlName>.CheckBoxItem ( nRow ) [ := | -->] lBoolean
<ParentWindowName>.<GridControlName>.CheckBoxAllItems := lBoolean
```

Grid Header Fonts, Images etc.,

```
<ParentWindowName>.<GridControlName>.PaintDoubleBuffer [ := | -->] lBoolean // Paints
via double-buffering, which reduces flicker
<ParentWindowName>.<GridControlName>.HeaderDYNAMICFONT ( nCol ) := {|| {cFontName, nFo
ntSize, [ lBold, lItalic, lUnderline, lStrikeOut ]} }
<ParentWindowName>.<GridControlName>.HeaderDYNAMICFORECOLOR ( nCol ) := {|| aColor }
<ParentWindowName>.<GridControlName>.HeaderDYNAMICBACKCOLOR ( nCol ) := {|| aColor }
<ParentWindowName>.<GridControlName>.Image ( lTransparent ) := { "image1.png", "image2
.bmp", ... }
<ParentWindowName>.<GridControlName>.ImageIndex ( nRow , nCol ) [ := | -->] nIndex
<ParentWindowName>.<GridControlName>.ImageList [ := | -->] hImageList
<ParentWindowName>.<GridControlName>.ColumnDYNAMICFONT ( nCol ) := {|| {cFontName, nFo
ntSize, [ lBold, lItalic, lUnderline, lStrikeOut ]} }
<ParentWindowName>.<GridControlName>.HeaderImageIndex ( nCol ) [ := | -->] nIndex
<ParentWindowName>.<GridControlName>.ChangeFontSize := nSize | NIL // Useful for use D
ynamic Font with more (less) Height than the size of font the Grid control
```

Dynamic Font

```
ARRAY FONT <cFontName> SIZE <nFontSize> [ BOLD ] [ ITALIC ] [ UNDERLINE ] [ STRIKEOUT
] --> { cFontName, nFontSize, lBold, lItalic, lUnderline, lStrikeout }
CREATE ARRAY FONT <cFontName> SIZE <nFontSize> [ BOLD <lBold> ] [ ITALIC <lIitalic> ]
[ UNDERLINE <lUnderline> ] [ STRIKEOUT <lStrikeout> ] --> { cFontName, nFontSize, lBol
d, lItalic, lUnderline, lStrikeout }
```

Additional Get Properties:

```
<ParentWindowName>.<GridControlName>.ColumnCOUNT --> nColumnCount
<ParentWindowName>.<GridControlName>.ColumnHEADER ( nColIndex ) --> cColumnHeader
<ParentWindowName>.<GridControlName>.ColumnWIDTH ( nColIndex ) --> nColumnWidth
<ParentWindowName>.<GridControlName>.ColumnJUSTIFY ( nColIndex ) --> nColumnJustify
<ParentWindowName>.<GridControlName>.ColumnCONTROL ( nColIndex ) --> aColumnControl
<ParentWindowName>.<GridControlName>.ColumnDYNAMICBACKCOLOR ( nColIndex ) --> bColumnD
ynamicBackColor
<ParentWindowName>.<GridControlName>.ColumnDYNAMICFORECOLOR ( nColIndex ) --> bColumnD
ynamicForeColor
<ParentWindowName>.<GridControlName>.ColumnVALID ( nColIndex ) --> bColumnValid
<ParentWindowName>.<GridControlName>.ColumnWHEN ( nColIndex ) --> bColumnWhen
<ParentWindowName>.<GridControlName>.ColumnONHEADCLICK ( nColIndex ) --> bColumnOnHead
Click
<ParentWindowName>.<GridControlName>.ColumnDISPLAYPOSITION ( nColIndex ) --> nColumnDi
splayPosition
<ParentWindowName>.<GridControlName>.CellEx ( nRowIndex, nColIndex ) --> xValue (very
more fast that .Cell)
<ParentWindowName>.<GridControlName>.CellRowFocused --> nCellRowIndex
<ParentWindowName>.<GridControlName>.CellColFocused --> nCellColIndex
<ParentWindowName>.<GridControlName>.CellRowClicked --> nCellRowIndex
<ParentWindowName>.<GridControlName>.CellColClicked --> nCellColIndex
<ParentWindowName>.<GridControlName>.CellNavigation --> lBoolean
<ParentWindowName>.<GridControlName>.EditOption --> GRID_EDIT_DEFAULT | GRID_EDIT_SELE
CTALL |
GRID_EDIT_INSERTBLANK | GRID_EDIT_INSERTCHAR |
GRID_EDIT_REPLACEALL
```

Additional Set Properties:

```
<ParentWindowName>.<GridControlName>.ColumnHEADER ( nColIndex ) := cColumnHeader
<ParentWindowName>.<GridControlName>.ColumnWIDTH ( nColIndex ) := [ nColumnWidth ] |
[ GRID_WIDTH_AUTOSIZE ] |
[ GRID_WIDTH_AUTOSIZEHEADER ]
<ParentWindowName>.<GridControlName>.ColumnJUSTIFY ( nColIndex ) := nColumnJustify
<ParentWindowName>.<GridControlName>.ColumnCONTROL ( nColIndex ) := aColumnControl
<ParentWindowName>.<GridControlName>.ColumnDYNAMICBACKCOLOR ( nColIndex ) := bColumnDy
namicBackColor
<ParentWindowName>.<GridControlName>.ColumnDYNAMICFORECOLOR ( nColIndex ) := bColumnDy
namicForeColor
<ParentWindowName>.<GridControlName>.ColumnVALID ( nColIndex ) := bColumnValid
<ParentWindowName>.<GridControlName>.ColumnWHEN ( nColIndex ) := bColumnWhen
<ParentWindowName>.<GridControlName>.ColumnONHEADCLICK ( nColIndex ) := bColumnOnHeadC
lick
<ParentWindowName>.<GridControlName>.ColumnDISPLAYPOSITION ( nColIndex ) := nColumnDis
playPosition
<ParentWindowName>.<GridControlName>.CellEx ( nRowIndex, nColIndex ) := xValue (very m
ore fast that .Cell)
<ParentWindowName>.<GridControlName>.BackGroundImage ( **nAction, cPicture, nRow, nCol
 )**
nAction = GRID_SETBKIMAGE_NONE | GRID_SETBKIMAGE_NORMAL | GRID_SETBKIMAGE_TILE | GRID_
SETBKIMAGE_WATERMARK
<ParentWindowName>.<GridControlName>.CellNavigation := lBoolean
<ParentWindowName>.<GridControlName>.EditOption := GRID_EDIT_DEFAULT | GRID_EDIT_SELEC
TALL | GRID_EDIT_INSERTBLANK | GRID_EDIT_INSERTCHAR | GRID_EDIT_REPLACEALL
```

New Methods:

```
<ParentWindowName>.<GridControlName>.AddColumnEx ( [ nColIndex ],[ cCaption ],[ nWidth
 ],[ nJustify ],[aColumnControl] )
<ParentWindowName>.<GridControlName>.AddItemEx ( aItem, nRow )
```

Set\/Get Grid New Properties\/Methods with equivalent syntax:

- **THIS.**XXX --> where XXX is the name of the new Grid Properties\/Methods
- AddColumn, AddColumnEx and DeleteColumn properties now NOT clean the Grid (NOT Delete all items),

  for compatibility with old behavior of ADDCOLUMN and DELETECOLUMN:
- **SET GridDeleteAllItems [TRUE|ON] | [FALSE|OFF]**
- **IsGridDeleteAllItems()** --> Return .T. or .F.
- Set colors and display mode in GRID cell navigation mode:

```
CellNavigationColor ( _SELECTEDCELL_FORECOLOR, aRGBcolor )
CellNavigationColor ( _SELECTEDCELL_BACKCOLOR, aRGBcolor )
CellNavigationColor ( _SELECTEDCELL_DISPLAYCOLOR, lBoolean )
CellNavigationColor ( _SELECTEDROW_FORECOLOR, aRGBcolor )
CellNavigationColor ( _SELECTEDROW_BACKCOLOR, aRGBcolor )
CellNavigationColor ( _SELECTEDROW_DISPLAYCOLOR, lBoolean )
```

- Get colors in GRID cell navigation mode:

```
aRGBcolor := CellNavigationColor ( _SELECTEDCELL_FORECOLOR )
aRGBcolor := CellNavigationColor ( _SELECTEDCELL_BACKCOLOR )
aRGBcolor := CellNavigationColor ( _SELECTEDROW_FORECOLOR )
aRGBcolor := CellNavigationColor ( _SELECTEDROW_BACKCOLOR )
```

@...HYPERLINK \/ DEFINE HYPERLINK

Creates an hyperlink control

Standard Syntax (xBase Style):

```
 @ <nRow>,<nCol> HYPERLINK <ControlName>
 [ OF <ParentWindowName> ]
 [ VALUE <cControlValue> ]
 [ ADDRESS <cLinkAddress>]
 [ WIDTH <nWidth> ]
 [ HEIGHT <nHeight> ]
 [ AUTOSIZE ]
 [ FONT <cFontName> ]
 [ SIZE <nFontSize> ]
 [ BOLD ]
 [ ITALIC ]
 [ TOOLTIP <cToolTipText> ]
 [ BACKCOLOR <anBackColor> ]
 [ FONTCOLOR <anFontColor> ]
 [ RIGHTALIGN ]
 [ CENTERALIGN ]
 [ HELPID <nHelpId> ]
 [ HANDCURSOR ]
 [ INVISIBLE ]
```

Alternate Syntax:

```
DEFINE HYPERLINK <ControlName>
   PARENT <ParentWindowName>
   ROW <nValue>
   COL <nValue>
   WIDTH <nValue>
   HEIGHT <nValue>
   FONTNAME <cValue>
   FONTSIZE <nValue>
   FONTBOLD <lValue>
   FONTITALIC <lValue>
   BACKCOLOR <anBackColor>
   FONTCOLOR <anFontColor>
   HELPID <nValue>
   VISIBLE <lValue>
   VALUE <cControlValue>
   ADDRESS <cLinkAddress>
   HANDCURSOR <lValue>
   AUTOSIZE <lValue>
   ALIGNMENT Left | Right | Center
   TOOLTIP <cToolTipText>
END HYPERLINK
```

Properties:

- Value
- Enabled
- Visible
- Row
- Col
- Width
- Height
- FontName
- FontSize
- FontBold
- FontItalic
- FontUnderline
- FontStrikeout
- AutoSize
- Address
- Name (R)
- Parent (D)
- BackColor
- FontColor
- HelpId (D)
- Alignment (D)
- Handcursor (D)

D: Available at control definition only R: Read-Only

Methods:

- Show
- Hide
- Release

@...IMAGE \/ DEFINE IMAGE

Creates an image control

Displays images in formats: BMP, GIF, JPG, TIF, WMF, EMF, CUR and PNG.

Standard Syntax (xBase Style):

```
@ <nRow> ,<nCol>
IMAGE <ControlName>
[ OF | PARENT <ParentWindowName> ]
[ ACTION | ONCLICK | ON CLICK <ActionProcedureName> | <bBlock> ]
PICTURE <cPictureName>
[ WIDTH <nWidth> ]
[ HEIGHT <nHeight> ]
[ STRETCH ]
[ HELPID <nHelpId> ]
[ INVISIBLE ]
[ TRANSPARENT ]
[ BACKGROUNDCOLOR anBackgroundColor ]
[ ADJUSTIMAGE ]
[ TRANSPARENTCOLOR anTransparentColor ]
[ TOOLTIP <cToolTipText> ]
```

Alternate Syntax:

```
DEFINE IMAGE <Controlname>
   PARENT <ParentWindowName>
   ROW <nRow>
   COL <nCol>
   ONCLICK <ActionProcedureName>
   PICTURE <cPictureName>
   WIDTH <nWidth>
   HEIGHT <nHeight>
   STRETCH <lValue>
   HELPID <nHelpId>
   VISIBLE <lValue>
   TRANSPARENT <lValue>
   BACKGROUNDCOLOR <anBackgroundColor>
   ADJUSTIMAGE <lValue>
   TRANSPARENTCOLOR <anTransparentColor>
   TOOLTIP <cToolTipText>
END IMAGE
```

Properties:

- Enabled
- Visible
- Picture

- Row
- Col
- Width
- Height
- Name (R)
- Parent (D)
- HelpId (D)
- Stretch (D)
- Transparent (D)
- BackgroundColor (D)
- AdjustImage (D)
- TransparentColor (D)
- hBitmap

D: Available at control definition only R: Read-Only

Note:

- If **WIDTH** is Zero the image is displayed with the Width of the original image, for default is zero.
- If **HEIGHT** is Zero the image is displayed with the Height of the original image, for default is zero.
- **STRETCH:** the Width\/Height of the image is adjusted exactly at the Width\/Height of the Image Control.
- **TRANSPARENT:** sets the color of the first pixel (col=0, row=0) of the image to treat as transparent in the image.
- **BACKGROUNDCOLOR:** sets the color of the Background image in ***anBackgroundColor***, for default is the color COLOR_BTNFACE.
- **ADJUSTIMAGE:** the WIDTH\/HEIGHT ratio of the Image Control is adjusted exactly at the Width\/Height ratio of the image.
- **TRANSPARENTCOLOR:** sets the color ***anTransparentColor*** to treat as transparent in the image, for default is the color of the first pixel of the image.

Events:

- OnClick

Methods:

- Show
- Hide
- Release
- Refresh

@...IPADDRESS \/ DEFINE IPADDRESS

Creates an IpAddress control

Standard Syntax (xBase Style):

```
@ <nRow> ,<nCol>
IPADDRESS <ControlName>
[ OF | PARENT <ParentWindowName> ]
[ HEIGHT <nHeight> ]
[ WIDTH <nWidth> ]
[ VALUE <anValue> ]
[ FONT <cFontName> SIZE <nFontSize> ]
[ BOLD ] [ ITALIC ] [ UNDERLINE ] [ STRIKEOUT ]
[ TOOLTIP <cToolTipText> ]
[ ON CHANGE <OnChangeProcedure> | <bBlock> ]
[ ON GOTFOCUS <OnGotFocusProcedur> | <bBlock> ]
[ ON LOSTFOCUS <OnGotFocusProcedur> | <bBlock> ]
[ HELPID <nHelpId> ]
[ INVISIBLE ]
[ NOTABSTOP ]
```

Alternate Syntax:

Properties:

```
DEFINE IPADDRESS
   PARENT <ParentWindowName>
   ROW <nValue>
   COL <nValue>
   HEIGHT <nHeight>
   WIDTH <nWidth>
   VALUE <anValue>
   FONTNAME <cValue>
   FONTSIZE <nValue>
   FONTBOLD <lValue>
   FONTITALIC <lValue>
   FONTUNDERLINE <lValue>
   FONTSTRIKEOUT <lValue>
   TOOLTIP <cToolTipText>
   ONCHANGE <OnChangeProcedure>
   ONGOTFOCUS <OnGotFocusProcedure>
   ONLOSTFOCUS <OnGotFocusProcedure>
   HELPID <nHelpId>
   VISIBLE <lValue>
   TABSTOP <lValue>
END IPADDRESS
```

- Value

- Enabled
- Visible
- Row
- Col
- Width
- Height
- FontName
- FontSize
- FontBold
- FontItalic
- FontUnderline
- FontStrikeout
- Name (R)
- HelpId (D)
- TabStop (D)

D: Available at control definition only R: Read-Only

Events:

- OnChange
- OnGotFocus
- OnLostFocus

Methods:

- Show
- Hide
- Release

@...LABEL \/ DEFINE LABEL

Creates a label control

Standard Syntax (xBase Style):

```
@ <nRow> ,<nCol>
LABEL <ControlName>
[ OF | PARENT <ParentWindowName> ]
VALUE <cValue>
[ ACTION | ONCLICK |
ON CLICK <ActionProcedureName> | <bBlock> ]
[ WIDTH <nWidth> ]
[ HEIGHT <nHeight> ]
[ AUTOSIZE ]
[ FONT <cFontname> SIZE <nFontsize> ]
[ BOLD ] [ ITALIC ] [ UNDERLINE ] [ STRIKEOUT ]
[ TOOLTIP <cToolTipText> ]
[ BACKCOLOR <anBackColor> ]
[ FONTCOLOR <anFontColor>]
[ TRANSPARENT ]
[ RIGHTALIGN | CENTERALIGN ]
[ HELPID <nHelpId> ]
[ INVISIBLE ]
[ ENDELLIPSES ]
[ NOPREFIX ]
```

Alternate Syntax:

```
DEFINE LABEL <ControlName>
   PARENT <ParentWindowName>
   ROW <nValue>
   COL <nValue>
   VALUE <cValue>
   ONCLICK <ActionProcedureName>
   WIDTH <nWidth>
   HEIGHT <nHeight>
   AUTOSIZE <lValue>
   FONTNAME <cValue>
   FONTSIZE <nValue>
   FONTBOLD <lValue>
   FONTITALIC <lValue>
   FONTUNDERLINE <lValue>
   FONTSTRIKEOUT <lValue>
   TOOLTIP <cToolTipText>
   BACKCOLOR <anBackColor>
   FONTCOLOR <anFontColor>
   TRANSPARENT <lValue>
   ALIGNMENT Left | Right | Center
   HELPID <nHelpId>
   VISIBLE <lValue>
   ENDELLIPSES <lValue>
   NOPREFIX <lValue>
END LABEL
```

Properties:

- Value
- Enabled
- Visible
- Row
- Col
- Width
- Height
- FontName
- FontSize
- FontBold
- FontItalic
- FontUnderline
- FontStrikeout
- AutoSize
- Name (R)
- Parent (D)
- BackColor
- FontColor

- HelpId (D)
- Alignment (D)
- EndEllipses (D)
- NoPrefix (D)

D: Available at control definition only R: Read-Only

Events:

- OnClick

Methods:

- Show
- Hide
- Release

Note:

ENDELLIPSES --> If the string does not fit in the size of the Label control, the string is truncated and ellipses (...) are added.

NOPREFIX --> Prevents interpretation of any ampersand (&) characters in the control's text as accelerator prefix characters. These are displayed with the ampersand removed and the next character in the string underlined.

@...LISTBOX \/ DEFINE LISTBOX

Creates a ListBox control

Standard Syntax (xBase Style):

```
@ <nRow> ,<nCol>
LISTBOX <ControlName>
[ OF | PARENT <ParentWindowName> ]
WIDTH <nWidth>
HEIGHT <nHeight>
[ ITEMS <acItems> ]
[ VALUE <nValue> ]
[ FONT <cFontName> SIZE <nFontSize> ]
[ BOLD ] [ ITALIC ] [ UNDERLINE ] [ STRIKEOUT ]
[ TOOLTIP <cToolTipText> ]
[ BACKCOLOR <aBackColor> ]
[ FONTCOLOR <aFontColor> ]
[ ON GOTFOCUS <OnGotFocusProcedur> | <bBlock> ]
[ ON CHANGE <OnChangeProcedure> | <bBlock> ]
[ ON LOSTFOCUS <OnLostFocusProcedur> | <bBlock> ]
[ ON DBLCLICK <OnDblClickProcedure> | bBlock> ]
[ MULTISELECT ]
[ HELPID <nHelpId> ]
[ BREAK ]
[ INVISIBLE ]
[ NOTABSTOP ]
[ SORT ]
[ DRAGITEMS ]
```

Alternate Syntax:

```
DEFINE LISTBOX
   PARENT <ParentWindowName>
   ROW <nValue>
   COL <nValue>
   WIDTH <nWidth>
   HEIGHT <nHeight>
   ITEMS <acItems>
   VALUE <nValue>
   FONTNAME <cValue>
   FONTSIZE <nValue>
   FONTBOLD <lValue>
   FONTITALIC <lValue>
   FONTUNDERLINE <lValue>
   FONTSTRIKEOUT <lValue>
   TOOLTIP <cToolTipText>
   BACKCOLOR <aBackColor>
   FONTCOLOR <aFontColor>
   ONGOTFOCUS <OnGotFocusProcedure>
   ONCHANGE <OnChangeProcedure>
   ONLOSTFOCUS <OnLostFocusProcedure>
   ONDBLCLICK <OnDblClickProcedure>
   MULTISELECT <lValue>
   HELPID <nHelpId>
   BREAK <lValue>
   VISIBLE <lValue>
   TABSTOP <lValue>
   SORT <lValue>
   DRAGITEMS <lValue>
END LISTBOX
```

Properties:

- Value
- Enabled
- Visible
- Item (nItemIndex)
- Items (D)
- ItemCount
- Row
- Col
- Width
- Height
- FontName
- FontSize
- FontBold
- FontItalic
- FontUnderline

- FontStrikeout
- ToolTip
- BackColor
- FontColor
- Name (R)
- Parent (D)
- MultiSelect (D)
- HelpId (D)
- Break (D)
- TabStop (D)
- Sort (D)
- DragItems (D)

D: Available at control definition only R: Read-Only

Events:

- OnGotFocus
- OnChange
- OnLostFocus
- OnDblClick

Methods:

- Show
- Hide
- AddItem (cItemText)
- DeleteItem (nItemIndex)
- DeleteAllItems
- SetFocus
- Release

Hints:

- If MULTISELECT clause is used VALUE must be a numeric array, containing the index position of selected items.
- When used in control definition, ITEM property must be a character array.

DEFINE MAIN MENU

Creates a Main Menu definition

Standard Syntax (xBase Style):

```
DEFINE MAIN MENU [ OF <ParentWindowName> ]
   DEFINE POPUP <cPopupCaption> [ NAME <PopupName> ]
      MENUITEM <cItemCaption>
         ACTION <ActionProvedureName> | <bBlock>
         [ NAME <MenuItemName>]
         [ IMAGE <cImageName> ]
         [ CHECKED ]
         [ NOTRANSPARENT ]
         [ TOOLTIP <cToolTipText> ]
         ...
         ...
         [ SEPARATOR ]
         ...
         ...
   END POPUP
   ...
   ...
END MENU
```

Alternate Syntax:

```
DEFINE MAINMENU [ PARENT <ParentWindowName> ]
   DEFINE POPUP <cPopupCaption> [ NAME <PopupName> ]
      MENUITEM <cItemCaption>
      ONCLICK <ActionProvedureName> | <bBlock>
      [ NAME <MenuItemName>]
      [ IMAGE <cImageName> ]
      [ CHECKED <lValue> ]
      [ TRANSPARENT <lValue> ]
      [ TOOLTIP <cToolTipText> ]
      ...
      ...
      [ SEPARATOR ]
      ...
      ...
   END POPUP
   ...
   ...
END MENU
```

MainMenu Properties:

- Parent (R)

R: Read-Only

MenuItem Properties:

- Name (R)
- Checked
- Enabled
- Image
- ToolTip

R: Read-Only

MenuItem Events:

- OnClick (R)

R: Read-Only

You can **DEFINE\RELEASE** Menu at runtime:

```
RELEASE MAIN MENU OF FormName
RELEASE MAINMENU OF FormName
ReleaseMainMenu ( cFormName )
IsMainMenuDefined ( cFormName ) --> lBoolean
```

@...MONTHCALENDAR \/ DEFINE MONTHCALENDAR

Creates a MonthCalendar control

Standard Syntax (xBase Style):

```
@ <row>,<col>
MONTHCALENDAR <ControlName>
[ OF | PARENT <ParentWindowName> ]
[ VALUE <dValue> ]
[ FONT <cFontName> ]
[ SIZE <nFontsize> ]
[ BOLD ] [ ITALIC ] [ UNDERLINE ] [ STRIKEOUT ]
[ TOOLTIP <cTooltip> ]
[ NOTODAY ]
[ NOTODAYCIRCLE ]
[ WEEKNUMBERS ]
[ INVISIBLE ]
[ NOTABSTOP ]
[ ON CHANGE <OnChangeProcedure> | <bBlock> ]
[ HELPID <nHelpId> ]
```

Alternate Syntax:

```
DEFINE MONTHCALENDAR <ControlName>
   PARENT <ParentWindowName>
   ROW <nValue>
   COL <nValue>
   VALUE <dValue>
   FONTNAME <cValue>
   FONTSIZE <nValue>
   FONTBOLD <lValue>
   FONTITALIC <lValue>
   FONTUNDERLINE <lValue>
   FONTSTRIKEOUT <lValue>
   TOOLTIP <cValue>
   TODAY <lValue>
   TODAYCIRCLE <lValue>
   WEEKNUMBERS <lValue>
   VISIBLE <lValue>
   TABSTOP <lValue>
   ONCHANGE <OnChangeProcedure>
   HELPID <nHelpId>
END MONTHCALENDAR
```

Properties:

- Value
- Enabled

- Visible
- Row
- Col
- Width
- Height
- FontName
- FontSize
- FontBold
- FontItalic
- FontUnderline
- FontStrikeout
- ToolTip
- Name (R)
- Parent (D)
- Today (D)
- TodayCircle (D)
- WeekNumbers (D)
- NoTabStop (D)
- HelpId (D)

D: Available at control definition only R: Read-Only

Events:

- OnChange

Methods:

- Show
- Hide
- SetFocus
- Refresh
- Release

DEFINE NOTIFY MENU

Creates a notify menu definition

Standard Syntax (xBase Style):

```
DEFINE NOTIFY MENU [ OF <cParentWindowName> ]
   MENUITEM <cItemCaption>
      ACTION <ActionProvedureName> | <bBlock>
      [ NAME <MenuItemName>]
      [ IMAGE <cImageName> ]
      [ CHECKED ]
      [ NOTRANSPARENT ]
      [ TOOLTIP <cToolTipText> ]
   ...
   ...
   [ SEPARATOR ]
   ...
   ...
END MENU
```

Alternate Syntax:

```
DEFINE NOTIFYMENU [ PARENT <cParentWindowName> ]
   MENUITEM <cItemCaption>
   ONCLICK <ActionProvedureName>
   [ NAME <MenuItemName>]
   [ IMAGE <cImageName> ]
   [ CHECKED <lValue> ]
   [ TRANSPARENT <lValue> ]
   [ TOOLTIP <cToolTipText> ]
   ...
   ...
   [ SEPARATOR ]
   ...
   ...
END MENU
```

Notify Menu Properties:

- Parent (R)

MenuItem Properties:

- Name (R)
- Checked
- Enabled
- Image
- ToolTip

R: Read-Only

MenuItem Events:

- OnClick (R)

You can **DEFINE\/RELEASE** Menu at runtime:

```
RELEASE NOTIFY MENU OF FormName
RELEASE NOTIFYMENU OF FormName
ReleaseNotifyMenu ( cFormName )
IsNotifyMenuDefined ( cFormName ) --> lBoolean
```

@...PLAYER \/ DEFINE PLAYER

Creates a Player control

Standard Syntax (xBase Style):

```
  @ <nRow> ,<nCol>
  PLAYER <ControlName>
  [ OF | PARENT <ParentWindowName> ]
  WIDTH <nWidth>
  HEIGHT <nHeight>
  FILE <cFileName>
  [ NOAUTOSIZEWINDOW ]
  [ NOAUTOSIZEMOVIE ]
  [ NOERRORDLG ]
  [ NOMENU ]
  [ NOOPEN ]
  [ NOPLAYBAR ]
  [ SHOWALL ]
  [ SHOWPOSITION ]
  [ HELPID <nHelpId> ]
```

Alternate Syntax:

```
DEFINE PLAYER <ControlName>
   PARENT <ParentWindowName>
   ROW <nRow>
   COL <nCol>
   WIDTH <nWidth>
   HEIGHT <nHeight>
   FILE <cFileName>
   AUTOSIZEWINDOW <lValue>
   AUTOSIZEMOVIE <lValue>
   ERRORDLG <lValue>
   MENU <lValue>
   OPEN <lValue>
   PLAYBAR <lValue>
   SHOWALL <lValue>
   SHOWPOSITION <lValue>
   HELPID <nHelpId>
END PLAYER
```

Properties:

- Enabled
- Visible
- Position
- Repeat
- Speed

- Volume
- Zoom
- Row
- Col
- Width
- Height
- Length
- Name (R)
- Parent (D)
- File (D)
- AutoSizeWindow (D)
- AutoSizeMovie (D)
- ErrorDlg (D)
- Menu (D)
- Open (D)
- PlayBar (D)
- ShowAll (D)
- ShowPosition (D)
- HelpId (D)

D: Available at control definition only R: Read-Only

Methods:

- Show
- Hide
- SetFocus
- Play
- PlayReverse
- Stop
- Pause
- Close
- Release
- Eject
- Open
- OpenDialog
- Resume

@...PROGRESSBAR \/ DEFINE PROGRESSBAR

Creates a ProgressBar control

Standard Syntax (xBase Style):

```
@ <nRow> ,<nCol>
PROGRESSBAR<ControlName>
[ OF | PARENT <ParentWindowName> ]
RANGE <nRangeMin> , <nRangeMax>
[ WIDTH <nWidth> ]
[ HEIGHT <nHeight> ]
[ TOOLTIP <cToolTipText> ]
[ VERTICAL ]
[ SMOOTH ]
[ HELPID <nHelpId> ]
[ BACKCOLOR <aBackColor> ]
[ FORECOLOR <aForeColor> ]
```

Alternate Syntax:

```
DEFINE PROGRESSBAR
   PARENT <ParentWindowName>
   ROW <nValue>
   COL <nValue>
   WIDTH <nValue>
   HEIGHT <nValue>
   RANGEMAX <nRangeMax>
   RANGEMIN <nRangeMin>
   TOOLTIP <cToolTipText>
   VERTICAL <lValue>
   SMOOTH <lValue>
   HELPID <nHelpId>
   BACKCOLOR <aBackColor>
   FORECOLOR <aForeColor>
END PROGRESSBAR
```

Properties:

- Value
- Enabled
- Visible
- Row
- Col
- Width
- RangeMax
- RangeMin

- BackColor
- ForeColor
- Name (R)
- Parent (D)
- Vertical (D)
- Smooth (D)
- HelpId (D)

D: Available at control definition only

R: Read-Only

Methods:

- Show
- Hide
- Release

Hints:

- The minimum value in the range can be from 0 to 65,535. Likewise, the maximum value can be from 0 to 65,535
- **SMOOTH, BACKCOLOR & FORECOLOR** properties works only for 'Windows Classic' \/ 'Windows Standard' themes.

# New command for ProgressBar Control

- Use this command when you do not know the amount of progress toward completion but wish to indicate that progress is being made.

```
SET PROGRESSBAR ControlName OF FormName ENABLE MARQUEE [ UPDATED MilliSeconds ]
SET PROGRESSBAR ControlName OF FormName DISABLE MARQUEE
```

@...RADIOGROUP \/ DEFINE RADIOGROUP

Creates a RadioGroup Control

Standard Syntax (xBase Style):

```
@ <nRow> ,<nCol>
RADIOGROUP <ControlName>
[ OF | PARENT <cParentWindowName> ]
OPTIONS <acOptions>
[ VALUE <nValue> ]
[ WIDTH <nWidth> ]
[ SPACING <nSpacing> ]
[ FONT <cFontName> SIZE <nFontSize> ]
[ BOLD ] [ ITALIC ] [ UNDERLINE ] [ STRIKEOUT ]
[ TOOLTIP <cToolTipText> ]
[ BACKCOLOR <aBackColor> ]
[ FONTCOLOR <aFontColor> ]
[ ON CHANGE <OnChangeProcedure> | <bBlock> ]
[ TRANSPARENT ]
[ HELPID <nHelpId> ]
[ INVISIBLE ]
[ NOTABSTOP ]
[ READONLY <alReadOnly> ]
[ HORIZONTAL ]
```

Alternate Syntax:

```
DEFINE RADIOGROUP <Controlname>
   PARENT <cParentWindowName>
   OPTIONS <acOptions>
   VALUE <nValue>
   WIDTH <nWidth>
   SPACING <nSpacing>
   FONTNAME <cValue>
   FONTSIZE <nValue>
   FONTBOLD <lValue>
   FONTITALIC <lValue>
   FONTUNDERLINE <lValue>
   FONTSTRIKEOUT <lValue>
   TOOLTIP <cToolTipText>
   BACKCOLOR <aBackColor>
   FONTCOLOR <aFontColor>
   ONCHANGE <OnChangeProcedure>
   TRANSPARENT <lValue>
   HELPID <nHelpId>
   VISIBLE <lValue>
   TABSTOP <lvalue>
   READONLY <alReadOnly>
   HORIZONTAL <lvalue>
END RADIOGROUP
```

Properties:

- Value
- Enabled
- Visible
- Row
- Col
- Width
- FontName
- FontSize
- FontBold
- FontItalic
- FontUnderline
- FontStrikeout
- ToolTip
- BackColor
- FontColor
- Caption (nItemNumber)
- ReadOnly
- Name (R)
- Parent (D)

- Spacing (D)
- HelpId (D)
- Horizontal (D)
- Options (D)

D: Available at control definition only

R: Read-Only

Events:

- OnChange

Methods:

- Show
- Hide
- SetFocus
- Release

@...RICHEDITBOX \/ DEFINE RICHEDITBOX

Creates a RichEditBox Control

Standard Syntax (xBase Style):

```
@ <nRow>,<nCol> RICHEDITBOX <ControlName>
[ OF | PARENT> <ParentWindowName> ]
[ WIDTH <nWidth> ]
[ HEIGHT <nHeight> ]
[ FIELD <Field> ]
[ VALUE <cValue> ]
[ READONLY ]
[ FONT <cFontName> ]
[ SIZE <nFontSize> ]
[ BOLD ]
[ ITALIC ]
[ UNDERLINE ]
[ STRIKEOUT ]
[ TOOLTIP <cToolTip> ]
[ BACKCOLOR <aBackColor> ]
[ MAXLENGTH <nMaxLength> ]
[ ON GOTFOCUS <OnGotFocusProcedure> | <bBlock> ]
[ ON CHANGE <OnChangeProcedure> | <bBlock> ]
[ ON LOSTFOCUS <OnLostFocusProcedur> | <bBlock> ]
[ HELPID <nHelpId> ]
[ INVISIBLE ]
[ NOTABSTOP ]
[ NOHSCROLL ]
[ NOVSCROLL ]
[ ON SELECT <OnSelectChangeProcedure> | <bBlock> ]
[ ON LINK <OnLinkProcedure> | <bBlock> ]
[ ON VSCROLL <OnVScrollProcedure> | <bBlock> ]
```

Alternate Syntax:

```
DEFINE RICHEDITBOX <ControlName>
   PARENT <ParentWindowName>
   ROW <nValue>
   COL <nValue>
   WIDTH <nWidth>
   HEIGHT <nHeight>
   FIELD <Field>
   VALUE <cValue>
   READONLY <lValue>
   FONTNAME <cFontName>
   FONTSIZE <nFontSize>
   FONTBOLD <lvalue>
   FONTITALIC <lValue>
   FONTUNDERLINE <lValue>
   FONTSTRIKEOUT <lValue>
   TOOLTIP <cToolTip>
   BACKCOLOR <aBackColor>
   MAXLENGTH <nMaxLength>
   ONGOTFOCUS <OnGotFocusProcedure>
   ONCHANGE <OnChangeProcedure>
   ONLOSTFOCUS <OnLostFocusProcedure>
   HELPID <nHelpId>
   VISIBLE <lValue>
   TABSTOP <lValue>
   HSCROLL <lValue>
   VSCROLL <lValue>
   ONSELECT <OnSelectChangeProcedure>
   ONLINK <OnLinkProcedure>
   ONVSCROLL <OnVScrollProcedure>
END RICHEDITBOX
```

Properties:

- Value
- Enabled
- Visible
- Row
- Col
- Width
- Height
- FontName
- FontSize
- FontBold
- FontItalic
- FontUnderline
- FontStrikeout
- ToolTip

- BackColor
- CaretPos
- Name (R)
- Field (D)
- Parent (D)
- ReadOnly
- MaxLength (D)
- HelpId (D)
- Break (D)
- TabStop (D)

D: Available at control definition only R: Read-Only

Events:

- OnGotFocus
- OnChange
- OnLostFocus
- OnSelect
- OnLink
- OnVScroll

Methods:

- Show
- Hide
- SetFocus
- Release
- Save
- Refresh

# RICHEDITBOX Control improvement

- New Properties:

```
<FormName>.<ControlName>.FontName --> cFontName
<FormName>.<ControlName>.FontSize --> nFontSize
<FormName>.<ControlName>.FontBold --> lBoolean
<FormName>.<ControlName>.FontItalic --> lBoolean
<FormName>.<ControlName>.FontUnderline --> lBoolean
<FormName>.<ControlName>.FontStrikeOut --> lBoolean
<FormName>.<ControlName>.FontColor --> aRGBcolor
<FormName>.<ControlName>.FontBackColor --> aRGBcolor
<FormName>.<ControlName>.FontScript --> RTF_SUBSCRIPT | RTF_SUPERSCRIPT | RTF_NORMALSC
RIPT
<FormName>.<ControlName>.Link --> lBoolean
```

```
<FormName>.<ControlName>.FontName := cFontName
<FormName>.<ControlName>.FontSize := nFontSize
<FormName>.<ControlName>.FontBold := lBoolean
<FormName>.<ControlName>.FontItalic := lBoolean
<FormName>.<ControlName>.FontUnderline := lBoolean
<FormName>.<ControlName>.FontStrikeOut := lBoolean
<FormName>.<ControlName>.FontColor := aRGBcolor | RTF_FONTAUTOCOLOR
<FormName>.<ControlName>.FontBackColor := aRGBcolor | RTF_FONTAUTOBACKCOLOR
<FormName>.<ControlName>.FontScript := RTF_SUBSCRIPT | RTF_SUPERSCRIPT | RTF_NORMALSCR
IPT
<FormName>.<ControlName>.Link := lBoolean
ThisRichEditBox.GetClickLinkRange --> { nMinPos, nMaxPos } // only valid into ON LINK
Procedure
ThisRichEditBox.GetClickLinkText --> cLinkText // only valid into ON LINK Procedure
<FormName>.<ControlName>.RTFTextMode --> lBoolean
<FormName>.<ControlName>.AutoURLDetect --> lBoolean
<FormName>.<ControlName>.Zoom --> nZoomPercentage
<FormName>.<ControlName>.SelectRange --> { nMinPos, nMaxPos }
<FormName>.<ControlName>.CaretPos --> nPos
<FormName>.<ControlName>.Value --> cText
<FormName>.<ControlName>.ViewRect --> { nLeft, nTop, nRight, nBottom }
<FormName>.<ControlName>.GetSelectText --> cSelectText
<FormName>.<ControlName>.GetTextLength --> nLengthText
<FormName>.<ControlName>.GetTextRange ( {nMinPos, nMaxPos} ) --> cTextRange
<FormName>.<ControlName>.GetPosChar ( nPos ) --> { nScreenRow, nScreenCol } or { -1, -
1 } if character is not displayed
<FormName>.<ControlName>.RTFTextMode := lBoolean
<FormName>.<ControlName>.AutoURLDetect := lBoolean
<FormName>.<ControlName>.Zoom := nZoomPercentage
<FormName>.<ControlName>.SelectRange := { nMinPos, nMaxPos }
<FormName>.<ControlName>.CaretPos := nPos
<FormName>.<ControlName>.Value := cText
<FormName>.<ControlName>.ViewRect := { nLeft, nTop, nRight, nBottom }
<FormName>.<ControlName>.AddText ( nPos ) := cText
<FormName>.<ControlName>.AddTextAndSelect ( nPos ) := cText
<FormName>.<ControlName>.CanPaste --> lBoolean
<FormName>.<ControlName>.CanUndo --> lBoolean
<FormName>.<ControlName>.CanRedo --> lBoolean
<FormName>.<ControlName>.BackGroundColor := aRGBcolor | RTF_AUTOBACKGROUNDCOLOR
<FormName>.<ControlName>.ParaAlignment --> RTF_LEFT | RTF_RIGHT | RTF_CENTER | RTF_JUS
TIFY
<FormName>.<ControlName>.ParaNumbering --> RTF_NOBULLETNUMBER | RTF_BULLET | RTF_ARABI
CNUMBER | RTF_LOWERCASELETTER | RTF_UPPERCASELETTER |
 RTF_LOWERCASEROMANNUMBER | RTF_UPPERCASEROMANNUMBER | RTF_CUSTOMCHARACTER
<FormName>.<ControlName>.ParaNumberingStyle --> RTF_NONE | RTF_PAREN | RTF_PARENS | RT
F_PERIOD | RTF_PLAIN | RTF_NONUMBER | RTF_NEWNUMBER
<FormName>.<ControlName>.ParaNumberingStart --> nNumberingStart
<FormName>.<ControlName>.ParaOffset --> nParagraphOffset // in millimeters
<FormName>.<ControlName>.ParaLineSpacing --> nInterLineSpacing
<FormName>.<ControlName>.ParaIndent --> nParagraphIndent // in millimeters
<FormName>.<ControlName>.ParaAlignment := RTF_LEFT | RTF_RIGHT | RTF_CENTER | RTF_JUST
IFY
<FormName>.<ControlName>.ParaNumbering := RTF_NOBULLETNUMBER | RTF_BULLET | RTF_ARABIC
```

```
NUMBER | RTF_LOWERCASELETTER | RTF_UPPERCASELETTER | RTF_LOWERCASEROMANNUMBER | RTF_UP
PERCASEROMANNUMBER | RTF_CUSTOMCHARACTER
<FormName>.<ControlName>.ParaNumberingStyle := RTF_NONE | RTF_PAREN | RTF_PARENS | RTF
_PERIOD | RTF_PLAIN | RTF_NONUMBER | RTF_NEWNUMBER
<FormName>.<ControlName>.ParaNumberingStart := nNumberingStart
<FormName>.<ControlName>.ParaOffset := nParagraphOffset // in millimeters
<FormName>.<ControlName>.ParaLineSpacing := nInterLineSpacing
<FormName>.<ControlName>.ParaIndent := nParagraphIndent // in millimeters
<FormName>.<ControlName>.FindText ( cFind, lDown, lMatchCase, lWholeWord, lSelectFindT
ext ) --> { nMinPos, nMaxPos }
<FormName>.<ControlName>.ReplaceText ( cFind, cReplace, lMatchCase, lWholeWord, lSelec
tFindText ) --> { nMinPos, nMaxPos }
<FormName>.<ControlName>.ReplaceAllText ( cFind, cReplace, lMatchCase, lWholeWord, lSe
lectFindText ) --> { nMinPos, nMaxPos }
```

Note:

- ***ParaNumbering***

```
RTF_NOBULLETNUMBER --> No paragraph numbering or bullets
RTF_BULLET --> Insert a bullet at the beginning of each selected paragraph
RTF_ARABICNUMBER --> Use Arabic numbers ( 0, 1, 2, ... )
RTF_LOWERCASELETTER --> Use lowercase letters ( a, b, c, ... )
RTF_UPPERCASELETTER --> Use lowercase Roman letters ( i, ii, iii, ... )
RTF_LOWERCASEROMANNUMBER --> Use uppercase letters ( A, B, C, ... )
RTF_UPPERCASEROMANNUMBER --> Use uppercase Roman letters ( I, II, III, ... )
RTF_CUSTOMCHARACTER --> Uses a sequence of characters beginning with the Unicode chara
cter specified by the ParaNumberingStart value
```

- ***ParaNumberingStyle***

```
RTF_NONE
RTF_PAREN --> Follows the number with a right parenthesis
RTF_PARENS --> Encloses the number in parentheses
RTF_PERIOD --> Follows the number with a period
RTF_PLAIN --> Displays only the number
RTF_NONUMBER --> Continues a numbered list without applying the next number or bullet
RTF_NEWNUMBER --> Starts a new number with ParaNumberingStart value
```

- New Methods:

```
<FormName>.<ControlName>.SelectAll
<FormName>.<ControlName>.UnSelectAll
<FormName>.<ControlName>.SelCopy
<FormName>.<ControlName>.SelPaste
<FormName>.<ControlName>.SelCut
<FormName>.<ControlName>.SelClear
<FormName>.<ControlName>.Undo
<FormName>.<ControlName>.Redo
<FormName>.<ControlName>.ClearUndoBuffer
<FormName>.<ControlName>.RTFLoadFile ( cFileName | cResouceName , lSelectText )
<FormName>.<ControlName>.RTFSaveFile ( cFileName , lSelectText )
<FormName>.<ControlName>.LoadFile ( cFileName | cResouceName , lSelectText , nFileType
 )
<FormName>.<ControlName>.SaveFile ( cFileName , lSelectText , nFileType )
```

Note*:* nFileType

```
RICHEDITFILE_TEXT
RICHEDITFILE_TEXTUTF8
RICHEDITFILE_TEXTUTF16
RICHEDITFILE_RTF
RICHEDITFILE_RTFUTF8
```

```
<FormName>.<ControlName>.RTFPrint ( { nMinPos, nMaxPos } , nLeft , nTop , nRight , nBo
ttom , PrintPageCodeBlock )
```

Note:

- Before invoking the RTFPrint() method you should call the SELECT PRINTER
- *nLeft* , *nTop* , *nRight* , *nBottom* are the margins of the paper expressed in millimeters
- *PrintPageCodeBlock* is a code block that is executed to send each page of the document to the printer,

  e.g. PrintPageCodeBlock := { || @ nRow , nCol PRINT "Pag. " + HB_NTOS(nPag++) CENTER }

See demo:

`\SAMPLES\Controls\RichEditBox`

@...SLIDER \/ DEFINE SLIDER

Creates a Slider Control

Standard Syntax (xBase Style):

```
@ <nRow> ,<nCol>
SLIDER <ControlName>
[ OF | PARENT <ParentWindowName> ]
RANGE <nRangeMin> , <nRangeMax>
[ VALUE <nValue> ]
[ WIDTH <nWidth> ]
[ HEIGHT <nHeight> ]
[ TOOLTIP <cToolTipText> ]
[ ON CHANGE <OnChangeProcedure> | <bBlock> ]
[ VERTICAL ]
[ NOTICKS ]
[ BOTH ]
[ TOP ]
[ LEFT ]
[ HELPID <nHelpId> ]
[ INVISIBLE ]
[ NOTABSTOP ]
```

Alternate Syntax:

```
DEFINE SLIDER <ControlName>
   PARENT <ParentWindowName>
   ROW <nValue>
   COL <nValue>
   RANGEMAX <nRangeMax>
   RANGEMIN <nRangeMin>
   VALUE <nValue>
   WIDTH <nWidth>
   HEIGHT <nHeight>
   TOOLTIP <cToolTipText>
   ONCHANGE <OnChangeProcedure>
   VERTICAL <lValue>
   TICKMARKS <lValue>
   BOTH <lValue>
   TOP <lValue>
   LEFT <lValue>
   HELPID <nHelpId>
   VISIBLE <lValue>
   TABSTOP <lValue>
END SLIDER
```

Properties:

- Value

- Enabled
- Visible
- Row
- Col
- Width
- Height
- ToolTip
- RangeMax
- RangeMin
- Name (R)
- Parent (D)
- Vertical (D)
- TickMarks (D)
- Both (D)
- Top (D)
- Left (D)
- HelpId (D)

D: Available at control definition only R: Read-Only

Events:

- OnChange

Methods:

- Show
- Hide
- SetFocus
- Release

Hints:

- The minimum value in the range can be from 0 to 65,535. Likewise, the maximum value can be from 0 to 65,535

@...SPINNER \/ DEFINE SPINNER

Creates a Spinner Control

Standard Syntax (xBase Style):

```
@ <nRow> ,<nCol>
SPINNER <ControlName>
[ OF | PARENT <ParentWindowName> ]
RANGE <nRangeMin> , <nRangeMax>
[ VALUE <nValue> ]
[ WIDTH <nWidth> ]
[ HEIGHT <nHeight> ]
[ FONT <cFontName> SIZE <nFontSize> ]
[ BOLD ] [ ITALIC ] [ UNDERLINE ] [ STRIKEOUT ]
[ TOOLTIP <cToolTipText> ]
[ BACKCOLOR <aBackColor> ]
[ FONTCOLOR <aFontColor> ]
[ ON GOTFOCUS <OnGotFocusProcedure> | <bBlock> ]
[ ON CHANGE <OnChangeProcedure> | <bBlock> ]
[ ON LOSTFOCUS <OnLostFocusProcedure> | <bBlock> ]
[ HELPID <nHelpId> ]
[ INVISIBLE ]
[ NOTABSTOP ]
[ WRAP ]
[ READONLY ]
[ INCREMENT <nIncrement> ]
```

Alternate Syntax:

```
DEFINE SPINNER <Controlname>
   PARENT <ParentWindowName>
   ROW <nRow>
   COL <nCol>
   RANGEMAX <nRangeMax>
   RANGEMIN <nRangeMin>
   VALUE <nValue>
   WIDTH <nWidth>
   HEIGHT <nHeight>
   FONTNAME <cValue>
   FONTSIZE <nValue>
   FONTBOLD <lValue>
   FONTITALIC <lValue>
   FONTUNDERLINE <lValue>
   FONTSTRIKEOUT <lValue>
   TOOLTIP <cToolTipText>
   BACKCOLOR <aBackColor>
   FONTCOLOR <aFontColor>
   ONGOTFOCUS <OnGotFocusProcedure>
   ONCHANGE <OnChangeProcedure>
   ONLOSTFOCUS <OnLostFocusProcedure>
   HELPID <nHelpId>
   VISIBLE <lValue>
   TABSTOP <lValue>
   WRAP <lValue>
   READONLY <lValue>
   INCREMENT <nIncrement>
END SPINNER
```

Properties:

- Value
- Enabled
- Visible
- Row
- Col
- Width
- Height
- FontName
- FontSize
- FontBold
- FontItalic
- FontUnderline
- FontStrikeout
- ToolTip
- BackColor

- FontColor
- RangeMax
- RangeMin
- Name (R)
- Wrap (D)
- ReadOnly (D)
- Increment (D)
- Parent (D)
- HelpId (D)

D: Available at control definition only R: Read-Only

Events**:**

- OnGotFocus
- OnChange
- OnLostFocus

Methods:

- Show
- Hide
- SetFocus
- Release

Hints:

- The minimum value in the range can be from –2,147,483,648 to 2,147,483,647. Likewise, the maximum value can be from –2,147,483,648 to 2,147,483,647 (signed 32-bit integer).

DEFINE SPLITBOX

Creates a SplitBox Control

Standard Syntax (xBase Style)

```
DEFINE SPLITBOX [ OF <ParentWindowName> ] [ BOTTOM ] [ HORIZONTAL ]
... Control / Window Definitions...
END SPLITBOX
```

Alternate Syntax:

```
DEFINE SPLITBOX
   [ PARENT <ParentWindowName> ]
   [ BOTTOM <lValue> ]
   [ HORIZONTAL <lValue> ]
 ... Control / Window Definitions...
END SPLITBOX
```

Properties:

- Bottom (D)
- Parent (D)
- Horizontal (D)

D: Available at control definition only

Controls \/ Windows defined as part of this container can be arranged

- by users, using a gripperbar located at control's left side. It can be used with listbox, grid, editbox, tree, browse, combobox, toolbar and 'SplitChild' windows. You must omit '@ <row>,<col>' in control definition.
- Since SplitBox is a container, you can specify when refer to properties or events of its child controls, ie:

```
Win1.SplitBox.Grid1.Value := 10
```

DEFINE STATUSBAR

Creates a StatusBar Control

Standard Syntax (xBase Style):

```
  DEFINE STATUSBAR
     [ OF | PARENT <ParentWindowName> ]
     [ FONT <cFontName> SIZE <nFontSize> ]
     [ TOP ]
     STATUSITEM <cItemCaption>
        [ WIDTH <nWidth>]
        [ ACTION <ActionProvedureName> | <bBlock> ]
        [ ICON <cIconName>]
        [ FLAT | RAISED ]
        [ TOOLTIP <cToolTipText>]
    [ DATE
       [ WIDTH <nWidth> ]
       [ ACTION <ActionProvedureName> | <bBlock> ]
       [ TOOLTIP <cToolTipText>] ]
    [ CLOCK
       [ WIDTH <nWidth> ]
       [ ACTION <ActionProvedureName> | <bBlock> ]
       [ TOOLTIP <cToolTipText> ]
       [ INTERVAL <nIntervalUpdate> ] ]
    [ KEYBOARD
       [ WIDTH <nWidth> ]
       [ ACTION <ActionProvedureName> | <bBlock> ]
       [ TOOLTIP <cToolTipText> ]
       [ INTERVAL <nIntervalUpdate> ] ]
     ...
     ...
 END STATUSBAR
```

Alternate Syntax:

```
DEFINE STATUSBAR
   [ PARENT <ParentWindowName> ]
   [ FONTNAME <cFontName> FONTSIZE <nFontSize> ]
   [ TOP <lValue> ]
   STATUSITEM <cItemCaption>
      [ WIDTH <nWidth>]
      [ ACTION <ActionProvedureName> | <bBlock> ]
      [ ICON <cIconName>]
      [ STYLE FLAT | RAISED ]
      [ TOOLTIP <cToolTipText>]
  [ DATE
      [ WIDTH <nWidth> ]
      [ ACTION <ActionProvedureName> | <bBlock> ]
      [ TOOLTIP <cToolTipText>] ]
  [ CLOCK
      [ WIDTH <nWidth> ]
      [ ACTION <ActionProvedureName> | <bBlock> ]
      [ TOOLTIP <cToolTipText> ]
      [ INTERVAL <nIntervalUpdate> ] ]
  [ KEYBOARD
      [ WIDTH <nWidth> ]
      [ ACTION <ActionProvedureName> | <bBlock> ]
      [ TOOLTIP <cToolTipText> ]
      [ INTERVAL <nIntervalUpdate> ] ]
  ...
   ...
END STATUSBAR
```

StatusBar Properties:

- Parent (D)
- FontName (D)
- FontSize (D)
- Top (D)

StatusItem Properties:

- Item (nItemIndex)
- Width (D)
- Icon (D)
- Style (D)
- ToolTip (D)

StatusItem Events:

- Action (D)

Hints:

- The tooltip for a part will only be displayed if the part has an icon and no text or if all of the text cannot be displayed inside the part.
- Since that can be only one StatusBar control per window, there is no need to give it a name (the name €˜StatusBar€™ is automatically assigned).

Sample:

```
#include "hmg.ch"
Function Main
   DEFINE WINDOW Form_1 ;
      AT 0,0 ;
      WIDTH 600 HEIGHT 400 ;
      TITLE 'HMG StatusBar Demo' ;
      MAIN
      DEFINE MAIN MENU
         POPUP '&StatusBar Test'
            ITEM 'Set StatusBar Item 1' ACTION Form_1.StatusBar.Item(1) := "New value
1"
            ITEM 'Set StatusBar Item 3' ACTION Form_1.StatusBar.Item(3) := "New value
3"
            ITEM 'Set StatusBar Item Icon' ACTION Form_1.StatusBar.Icon (3) := 'New.iC
O'
            ITEM 'Open Other Window...' ACTION Modal_Click()
         END POPUP
         POPUP '&Help'
            ITEM '&About' ACTION MsgInfo ("HMG StatusBar Demo")
         END POPUP
      END MENU
      DEFINE STATUSBAR
         STATUSITEM "Item 1" ACTION MsgInfo('Click! 1')
         STATUSITEM "Item 2" WIDTH 100 ACTION MsgInfo('Click! 2')
         STATUSITEM 'A Car!' WIDTH 100 ICON 'Car.Ico'
         CLOCK
         DATE
      END STATUSBAR
   END WINDOW
   CENTER WINDOW Form_1
   ACTIVATE WINDOW Form_1
Return Nil
```

STATUSBAR Control improvement

```
<ParentWindowName>.StatusBar.IconHandle (nItem) := hIcon
```

DEFINE TAB

Creates a Tab Control

Standard Syntax (xBase Style):

```
DEFINE TAB <ControlName>
   [ OF <ParentWindowName> ]
   AT <nRow> ,<nCol>
   WIDTH <nWidth>
   HEIGHT<nHeight>
   [ VALUE <nValue> ]
   [ FONT <cFontname> SIZE <nFonSize> ]
   [ BOLD ]
   [ ITALIC ]
   [ UNDERLINE ]
   [ STRIKEOUT ]
   [ TOOLTIP <cToolTipText> ]
   [ BUTTONS ]
   [ FLAT ]
   [ HOTTRACK ]
   [ VERTICAL ]
   [ VERTICAL ]
   [ MULTILINE ]
   [ ON CHANGE <OnChangeProcedure> | <bBlock> ]
   [ NOTRANSPARENT ]

   DEFINE PAGE <cPageCaption> [ IMAGE <cImageName> ]
       ... Control Definitions...
   END PAGE
   ...
END TAB
```

Alternate Syntax:

```
DEFINE TAB <ControlName>
   [ PARENT <ParentWindowName> ]
   ROW <nRow>
   COL <nCol>
   WIDTH <nWidth>
   HEIGHT <nHeight>
   [ VALUE <nValue> ]
   [ FONTNAME <cFontname> ]
   [ FONTSIZE <nFonSize> ]
   [ FONTBOLD <lValue> ]
   [ FONTITALIC <lValue> ]
   [ FONTUNDERLINE <lValue> ]
   [ FONTSTRIKEOUT <lValue> ]
   [ TOOLTIP <cToolTipText> ]
   [ BUTTONS <lValue> ]
   [ FLAT <lValue> ]
   [ HOTTRACK <lValue> ]
   [ VERTICAL <lValue> ]
   [ MULTILINE <lValue> ]
   [ ONCHANGE <OnChangeProcedure> | <bBlock> ]
   [ NOTRANSPARENT ]

   DEFINE PAGE <cPageCaption> [ IMAGE <cImageName> ]
      ... Control Definitions...
   END PAGE
   ...
END TAB
```

Properties:

- Value
- Enabled
- Visible
- Row
- Col
- Width
- Height
- Caption(nPageNumber)
- Fontname
- FontSize
- FontBold
- FontItalic
- FontUnderline
- FontStrikeout
- Name (R)
- Buttons (D)

- Flat (D)
- HotTrack (D)
- Vertical (D)
- Multiline (D)

D: Available at control definition only R: Read-Only

Events:

- OnChange

Methods:

- Show
- Hide
- Release
- AddPage (nPageNumber , cCaption [, cImageName])
- DeletePage (nPageNumber)
- AddControl (ControlName , nPagenumber , nRow , nCol)

Hints:

- VERTICAL style has no effect when XP or Vista themes are used.
- Since Tab is a container, you can specify when refer to properties or events of its child controls, including page number, ie: `Win1.Tab1(1).Grid1.Value := 10`

@...TEXTBOX \/ DEFINE TEXTBOX

Creates a TextBox Control

Standard Syntax (xBase Style):

Simple TEXTBOX:

```
@ <nRow>, <nCol>
 TEXTBOX <ControlName>
 [ OF | PARENT <ParentWindowName> ]
 [ HEIGHT <nHeight> ]
 [ WIDTH <nWidth> ]
 [ FIELD <FieldName> ]
 [ VALUE <cValue> ]
 [ READONLY ]
 [ FONT <cFontName> SIZE <nFontSize> ]
 [ BOLD ] [ ITALIC ] [ UNDERLINE ] [ STRIKEOUT ]
 [ TOOLTIP <ToolTipText> ]
 [ BACKCOLOR <aBackColor> ]
 [ FONTCOLOR <aFontColor> ]
 [ DISABLEDBACKCOLOR <aDisabledBackColor> ]
 [ DISABLEDFONTCOLOR <aDisabledFontColor> ]
 [ MAXLENGTH <nInputLength> ]
 [ UPPERCASE ]
 [ LOWERCASE ]
 [ NUMERIC ]
 [ PASSWORD ]
 [ ON CHANGE <OnChangeProcedure> ]
 [ ON GOTFOCUS <OnGotFocusProcedure> ]
 [ ON LOSTFOCUS <OnLostFocusProcedure> ]
 [ ON ENTER <OnEnterProcedure> ]
 [ RIGHTALIGN ]
 [ INVISIBLE ]
 [ NOTABSTOP ]
 [ HELPID <nHelpId> ]
```

TEXTBOX (Numeric InputMask):

```
@ <nRow>,<nCol>
 TEXTBOX <ControlName>
 [ OF | PARENT <ParentWindowName> ]
 [ HEIGHT <nHeight> ]
 [ WIDTH <nWidth> ]
 [ FIELD <FieldName> ]
 [ VALUE <nValue> ]
 [ READONLY ]
 [ FONT <cFontName> SIZE <nFontSize> ]
 [ BOLD ] [ ITALIC ] [ UNDERLINE ] [ STRIKEOUT ]
 [ TOOLTIP <ToolTipText> ]
 [ BACKCOLOR <aBackColor> ]
 [ FONTCOLOR <aFontColor> ]
 [ DISABLEDBACKCOLOR <aDisabledBackColor> ]
 [ DISABLEDFONTCOLOR <aDisabledFontColor> ]
 NUMERIC
 INPUTMASK <cMask>
 [ FORMAT <cFormat> ]
 [ ON CHANGE <OnChangeProcedure> ]
 [ ON GOTFOCUS <OnGotFocusProcedure> ]
 [ ON LOSTFOCUS <OnLostFocusProcedure> ]
 [ ON ENTER <OnEnterProcedure> ]
 [ RIGHTALIGN ]
 [ INVISIBLE ]
 [ NOTABSTOP ]
 [ HELPID <nHelpId> ]
```

TEXTBOX (Character InputMask):

```
@ <nRow>,<nCol>
 TEXTBOX <ControlName>
 [ OF | PARENT <ParentWindowName> ]
 [ HEIGHT <nHeight> ]
 [ WIDTH <nWidth> ]
 [ FIELD <FieldName> ]
 [ VALUE <cValue> ]
 [ READONLY ]
 [ FONT <cFontName> SIZE <nFontSize> ]
 [ BOLD ] [ ITALIC ] [ UNDERLINE ] [ STRIKEOUT ]
 [ TOOLTIP <ToolTipText> ]
 [ BACKCOLOR <aBackColor> ]
 [ FONTCOLOR <aFontColor> ]
 [ DISABLEDBACKCOLOR <aDisabledBackColor> ]
 [ DISABLEDFONTCOLOR <aDisabledFontColor> ]
 INPUTMASK <cMask>
 [ ON CHANGE <OnChangeProcedure> ]
 [ ON GOTFOCUS <OnGotFocusProcedure> ]
 [ ON LOSTFOCUS <OnLostFocusProcedure> ]
 [ ON ENTER <OnEnterProcedure> ]
 [ RIGHTALIGN ]
 [ INVISIBLE ]
 [ NOTABSTOP ]
 [ HELPID <nHelpId> ]
```

TEXTBOX (Date Type):

```
@ <nRow>, <nCol>
 TEXTBOX <ControlName>
 [ OF | PARENT <ParentWindowName> ]
 [ HEIGHT <nHeight> ]
 [ WIDTH <nWidth> ]
 [ FIELD <FieldName> ]
 [ VALUE <cValue> ]
 [ READONLY ]
 [ FONT <cFontName> SIZE <nFontSize> ]
 [ BOLD ] [ ITALIC ] [ UNDERLINE ] [ STRIKEOUT ]
 [ TOOLTIP <ToolTipText> ]
 [ BACKCOLOR <aBackColor> ]
 [ FONTCOLOR <aFontColor> ]
 [ DISABLEDBACKCOLOR <aDisabledBackColor> ]
 [ DISABLEDFONTCOLOR <aDisabledFontColor> ]
 < DATE >
 [ ON CHANGE <OnChangeProcedure> ]
 [ ON GOTFOCUS <OnGotFocusProcedure> ]
 [ ON LOSTFOCUS <OnLostFocusProcedure> ]
 [ ON ENTER <OnEnterProcedure> ]
 [ RIGHTALIGN ]
 [ INVISIBLE ]
 [ NOTABSTOP ]
 [ HELPID <nHelpId> ]
```

Alternate Syntax:

```
DEFINE TEXTBOX <Controlname>
   PARENT <ParentWindowName>
   ROW <nValue>
   COL <nValue>
   HEIGHT <nHeight>
   FIELD <FieldName>
   VALUE <nValue>
   READONLY <lValue>
   WIDTH <nWidth>
   INPUTMASK <cMask>
   FORMAT <cFormat>
   MAXLENGTH <nMaxLength>
   PASSWORD <lValue>
   FONTNAME <cValue>
   FONTSIZE <nValue>
   FONTBOLD <lValue>
   FONTITALIC <lValue>
   FONTUNDERLINE <lValue>
   FONTSTRIKEOUT <lValue>
   TOOLTIP <cToolTipText>
   BACKCOLOR <aBackColor>
   FONTCOLOR <aFontColor>
   DISABLEDBACKCOLOR <aDisabledBackColor>
   DISABLEDFONTCOLOR <aDisabledFontColor>
   DATATYPE CHARACTER | DATE | NUMERIC
   MAXLENGTH <nInputLength>
   CASECONVERT NONE | UPPER | LOWER
   ONGOTFOCUS <OnGotFocusProcedur>
   ONCHANGE <OnChangeProcedure>
   ONLOSTFOCUS <OnLostFocusProcedure>
   ONENTER <OnEnterProcedure>
   RIGHTALIGN <lValue>
   VISIBLE <lValue>
   TABSTOP <lValue>
   HELPID <nHelpId>
END TEXTBOX
```

Properties:

- Value
- Enabled
- Visible
- Row
- Col
- Width
- Height
- FontName
- FontSize

- FontBold
- FontItalic
- FontUnderline
- FontStrikeout
- ToolTip
- BackColor
- FontColor
- DisabledBackColor
- DisabledFontColor
- CaretPos
- ReadOnly
- Name (R)
- TabStop (D)
- Field (D)
- Parent (D)
- InputMask (D)
- Format (D)
- MaxLength (D)
- RightAlign (D)
- HelpId (D)
- CaseConvert (D)
- DataType (D)
- GetTextLength

D: Available at control definition only R: Read-Only

Events:

- OnGotFocus
- OnChange
- OnLostFocus
- OnEnter

Methods:

- Show
- Hide
- SetFocus
- Release
- Refresh
- Save

InputMask String (Numeric Textbox):

```
9 Displays digits
$ Displays a dollar sign in place of a leading space
* Displays an asterisk in place of a leading space
. Specifies a decimal point position
, Specifies a comma position
```

InputMask String (Non-Numeric Textbox):

```
9 Digits
A Alphabetic Characters
! Converts an alphabetic character to uppercase
```

(All other characters are included in text in the position indicated by the mask)

Format String (Allowed in Numeric Textbox Only):

```
C : Displays CR after positive numbers
X : Displays DB after negative numbers
( : Encloses negative numbers in parentheses
E : Displays numbers in British format
```

Hints:

- Inputmask and Maxlength clauses, can't be used simultaneously.

THIS OBJECT REFERENCE

Semi-OOP system Object

Syntax:

```
This.<Property/MethodName>
```

It provides a reference to the current control\/window in event code.

@...TIMEPICKER \/ DEFINE TIMEPICKER

Creates a TimePicker control

Standard Syntax (xBase Style):

```
@ <nRow> ,<nCol>
TIMEPICKER<ControlName>
[ OF | PARENT <cParentWindowName> ]
[ VALUE <cValue> ]
[ FIELD <FieldName> ]
[ FORMAT <cTimeFormat>]
[ WIDTH <nWidth> ]
[ HEIGHT <nValue> ]
[ FONT <cFontName> SIZE <nFontSize> ]
[ BOLD ] [ ITALIC ] [ UNDERLINE ] [ STRIKEOUT ]
[ TOOLTIP <cToolTipText> ]
[ SHOWNONE ]
[ ON GOTFOCUS <OnGotFocusProcedur> | <bBlock> ]
[ ON CHANGE <OnChangeProcedure> | <bBlock> ]
[ ON LOSTFOCUS <OnLostFocusProcedure> | <bBlock> ]
[ HELPID <nHelpId> ]
[ ON ENTER <OnEnterProcedure> | <bBlock> ]
[ INVISIBLE ]
[ NOTABSTOP ]
```

Alternate Syntax:

```
DEFINE TIMEPICKER <ControlName>
   PARENT <ParentWindowName>
   ROW <nValue>
   COL <nValue>
   WIDTH <nValue>
   HEIGHT <nValue>
   FIELD <FieldName>
   FONTNAME <cValue>
   FONTSIZE <nValue>
   FONTBOLD <lValue>
   FONTITALIC <lValue>
   FONTUNDERLINE <lValue>
   FONTSTRIKEOUT <lValue>
   TOOLTIP <cValue>
   ONGOTFOCUS <ActionProcedure>
   ONLOSTFOCUS <ActionProcedure>
   ONCHANGE <ActionProcedure>
   TABSTOP <lValue>
   HELPID <nValue>
   VISIBLE <lValue>
   VALUE <cValue>
   FORMAT <cTimeFormat>
   SHOWNONE <lValue>
   ONENTER <ActionProcedure>
END TIMEPICKER
```

Properties:

- Value
- Format
- Enabled
- Visible
- Row
- Col
- Width
- Height
- FontName
- FontSize
- FontBold
- FontItalic
- FontUnderline
- FontStrikeout
- ToolTip
- Name (R)
- Field (D)
- Parent (D)

- ShowNone (D)
- HelpId (D)
- TabStop (D)

D: Available at control definition only R: Read-Only

Events:

- OnGotFocus
- OnChange
- OnLostFocus
- OnEnter

Methods:

- Show
- Hide
- SetFocus
- Release
- Save
- Refresh

Related Functions:

```
HMG_TimeToValue ( cTime ) --> Return aTimeValue array
HMG_ValueToTime ( aTimeValue , cTimeFormat ) --> Return cTime string
HMG_TimeToTime ( cTime , cNewTimeFormat ) --> Returns cTime string with the time forma
t specified
HMG_IsTimeAMPM ( cTime ) --> Return .T. if cTime contains the substring "am" or "pm"
aTimeValue = { nHour, nMinute, nSecond } --> nHour is always in the 24-hour format
```

(nHour = 0..23, nMinute = 0..59, nSecond = 0..59)

- *cTimeFormat* | *cNewTimeFormat* = **_TIMELONG24H** | **_TIMELONG12H** | **_TIMESHORT24H** | **_TIMESHORT12H**

- *cTime* --> eg. "13:37:21" , "01:37:21 pm", "13:37" , "01:37 pm"

DEFINE TIMER

Creates a Timer Control

Syntax:

```
DEFINE TIMER <ControlName>
   OF | PARENT <ParentWindowName>
   INTERVAL <nInterval>
   ACTION <ActionProcedureName>
```

Properties:

- Parent (D)
- Interval (D)
- Value
- Enabled
- Name (R)

R: Read-Only

Events:

- Action

Methods:

- Release

DEFINE TOOLBAR

Creates a ToolBar Control

Standard Syntax (xBase Style):

```
DEFINE TOOLBAR <ControlName>
   [ OF | PARENT <ParentWindowName> ]
   [ BUTTONSIZE <nWidth> , <nHeight> ]
   [ IMAGESIZE <nWidth> , <nHeight> ]
   [ STRICTWIDTH ]
   [ FONT <cFontName> ]
   [ SIZE <nFontSize> ]
   [ BOLD ]
   [ ITALIC ]
   [ UNDERLINE ]
   [ STRIKEOUT ]
   [ TOOLTIP <cTooltipText> ]
   [ FLAT ]
   [ BOTTOM ]
   [ RIGHTTEXT ]
   [ GRIPPERTEXT ]
   [ BORDER ]
   [ BREAK ]

   BUTTON <Controlname>
      [ CAPTION <cCaption> ]
      [ PICTURE <cPictureName> ]
      [ ACTION | ON CLICK | ONCLICK <ActionProcedureName> | <bBlock> ]
      [ TOOLTIP <cToolTipText> ]
      [ SEPARATOR ]
      [ AUTOSIZE ]
      [ DROPDOWN ]
      [ WHOLEDROPDOWN ]
      [ CHECK ]
      [ GROUP ]
      [ NOTRANSPARENT ]
 ...
 END TOOLBAR
```

Alternate Syntax:

```
DEFINE TOOLBAR <ControlName>
   [ PARENT <ParentWindowName> ]
   [ BUTTONWIDTH <nWidth> ]
   [ BUTTONHEIGHT <nHeight> ]
   [ IMAGEWIDTH <nWidth> ]
   [ IMAGEHEIGHT <nHeight> ]
   [ STRICTWIDTH [lValue] ]
   [ FONTNAME <cFontName> ]
   [ FONTSIZE <nFontSize> ]
   [ FONTBOLD <lValue> ]
   [ FONTITALIC <lValue> ]
   [ FONTUNDERLINE <lValue> ]
   [ FONTSTRIKEOUT <lValue> ]
   [ TOOLTIP <cTooltipText> ]
   [ FLAT <lValue> ]
   [ BOTTOM <lValue> ]
   [ RIGHTTEXT <lValue> ]
   [ GRIPPERTEXT <lValue> ]
   [ BORDER <lValue> ]
   [ BREAK <lValue> ]

   TOOLBUTTON <Controlname>
      [ CAPTION <cCaption> ]
      [ PICTURE <cPictureName> ]
      [ ONCLICK <ActionProcedureName> | <bBlock> ]
      [ TOOLTIP <cToolTipText> ]
      [ SEPARATOR <lValue> ]
      [ AUTOSIZE <lValue> ]
      [ DROPDOWN <lValue> ]
      [ WHOLEDROPDOWN <lValue> ]
      [ CHECK <lValue> ]
      [ GROUP <lValue> ]
      [ NOTRANSPARENT <lValue> ]
    ...
END TOOLBAR
```

ToolBar Properties:

- Parent (D)
- ButtonWidth (D)
- ButtonHeight (D)
- ImageWidth (D)
- ImageHeight (D)
- StrictWidth (D)
- FontName (D)
- FontSize (D)
- FontBold (D)
- FontItalic (D)

- FontUnderLine (D)
- FontStrikeOut (D)
- ToolTip (D)
- Flat (D)
- Bottom (D)
- RightText (D)
- GripperText (D)
- Border (D)
- Break (D)

ToolBar Button Properties:

- Enabled
- Caption
- Picture (D)
- Separator (D)
- DropDown (D)
- WholeDropDown (D)
- Check (D)
- Group (D)
- ToolTip (D)
- AutoSize (D)
- NoTransparent (D)

D: Available at control definition only

ToolBar Button Events:

- OnClick

Hints:

- GRIPPERTEXT Property works only for toolbars defined inside splitbox.
- "Action" and "WholeDropDown" clauses can't be used simultaneously
- Since ToolBar is a container, you can specify when refer to properties or events of its child buttons, ie:

```
Win1.ToolBar1.Button1.Enabled := .F.
```

DEFINE TREE

Creates a Tree Control

Standard Syntax (xBase Style):

```
DEFINE TREE <ControlName>
   [ OF | PARENT <ParentWindowName> ]
   AT <nRow> ,<nCol>
   WIDTH <nWidth>
   HEIGHT <nHeight>
   [ VALUE <nValue> ]
   [ FONT <cFontname> SIZE <nFonSize> ]
   [ TOOLTIP <cToolTipText> ]
   [ ON GOTFOCUS <OnGotFocusProcedur> | <bBlock> ]
   [ ON CHANGE <OnChangeProcedure> | <bBlock> ]
   [ ON LOSTFOCUS <OnLostFocusProcedure> | <bBlock> ]
   [ ON DBLCLICK <OnDblClickProcedure> | <bBlock> ]
   [ ON EXPAND <OnExpandProcedure> | <bBlock> ]
   [ ON COLLAPSE <OnCollapseProcedure> | <bBlock> ]
   [ BREAK ]
   [ NODEIMAGES <aImgNode> [ ITEMIMAGES <aImgItem> ]
   [ NOROOTBUTTON ]
   [ ITEMIDS ]
   [ HELPID <nHelpId> ]
   [ NOTRANSPARENT ]

   [ NODE <cNodeCaption> ]
      [ IMAGES <aImage> ]
      [ ID <nItemID> ]
   ...
      [ TREEITEM <cTreeItemCaption>
         [ IMAGES <aImage> ]
         [ ID <nItemID> ] ]
      ...
   [ END NODE ]
   ...
   ...
END TREE
```

Alternate Syntax:

```
DEFINE TREE <ControlName>
   [ PARENT <ParentWindowName> ]
   ROW <nRow>
   COL <nCol>
   WIDTH <nWidth>
   HEIGHT <nHeight>
   [ VALUE <nValue> ]
   [ FONTNAME <cFontname> ]
   [ FONTSIZE <nFonSize> ]
   [ TOOLTIP <cToolTipText> ]
   [ ONGOTFOCUS <OnGotFocusProcedure> | <bBlock> ]
   [ ONCHANGE <OnChangeProcedure> | <bBlock> ]
   [ ONLOSTFOCUS <OnLostFocusProcedure> | <bBlock> ]
   [ ONDBLCLICK <OnDblClickProcedure> | <bBlock> ]
   [ ONEXPAND <OnExpandProcedure> | <bBlock> ]
   [ ONCOLLAPSE <OnCollapseProcedure> | <bBlock> ]
   [ BREAK <lValue> ]
   [ NODEIMAGES <aImgNode> ]
   [ ITEMIMAGES <aImgItem> ]
   [ ROOTBUTTON <lValue> ]
   [ ITEMIDS <lValue> ]
   [ HELPID <nHelpId> ]
   [ TRANSPARENT <lValue> ]

   [ NODE <cNodeCaption> ]
      [ IMAGES <aImage> ]
      [ ID <nItemID> ]
   ...
      [ TREEITEM <cTreeItemCaption>
         [ IMAGES <aImage> ]
         [ ID <nItemID> ] ]
      ...
   [ END NODE ]
   ...
   ...
END TREE
```

Properties:

- Value
- Enabled
- Visible
- Row
- Col
- Width
- Height
- Item (nItemIndex | nItemID)
- ItemCount

- FontName
- FontSize
- FontBold
- FontItalic
- FontUnderline
- FontStrikeout
- ToolTip
- Name (R)
- Break (D)
- NodeImages (D)
- ItemImages (D)
- HelpId (D)
- Parent (D)
- RootButton (D)

D: Available at control definition only R: Read-Only

Events:

- OnGotFocus
- OnChange
- OnLostFocus
- OnDblClick
- OnExpand (*)
- OnCollapse (*)

(*)***Note***: **Property available for** OnExpand **and** OnCollapse **events and** DynamicForeColor**,** DynamicBackColor **and** DynamicFont **properties**:**

```
This.TreeItemValue
```

Methods:

- Show
- Hide
- AddItem (cItemText , nParentItemIndex | nParentItemID)
- DeleteItem (nItemIndex | nItemID)
- DeleteAllItems
- Expand (nItemIndex | nItemID)
- Collapse (nItemIndex | nItemID)
- SetFocus
- Release

When **ITEMIDS** clause is specified, you can assign a numeric ID (ID clause) to tree items and nodes. That way, all tree properties and methods will work using these ID's instead item position.

# TREE Control improvement

Note:___ nValue = nItemIndex | nItemID_

- New Get Properties:

```
<ParentWindowName>.<TreeControlName>.AllValue --> anAllItemsValues | NIL
<ParentWindowName>.<TreeControlName>.RootValue --> nValue | NIL
<ParentWindowName>.<TreeControlName>.FirstItemValue --> nValue | NIL
<ParentWindowName>.<TreeControlName>.ParentValue ( nValue ) --> nValue | NIL
<ParentWindowName>.<TreeControlName>.ChildValue ( nValue ) --> anChildItemsValues
| NIL
<ParentWindowName>.<TreeControlName>.SiblingValue ( nValue ) --> anSiblingItemsVal
ues | NIL
<ParentWindowName>.<TreeControlName>.ItemText ( anItemsValues ) --> acItemsText |
NIL
<ParentWindowName>.<TreeControlName>.IsTrueNode ( nValue ) --> lBoolean
<ParentWindowName>.<TreeControlName>.NodeFlag ( nValue ) --> lBoolean
<ParentWindowName>.<TreeControlName>.ImageCount --> nImageCount
<ParentWindowName>.<TreeControlName>.ImageIndex ( nValue ) --> { iUnSelectItem , i
SelectItem }
<ParentWindowName>.<TreeControlName>.IsExpand ( nValue ) --> lBoolean
<ParentWindowName>.<TreeControlName>.ImageList --> hImageList
<ParentWindowName>.<TreeControlName>.HasLines --> lBoolean
<ParentWindowName>.<TreeControlName>.FullRowSelect --> lBoolean
<ParentWindowName>.<TreeControlName>.HasButton ( nValue ) --> lBoolean
<ParentWindowName>.<TreeControlName>.Cargo ( nValue ) --> xData
<ParentWindowName>.<TreeControlName>.CargoScan ( xData ) --> nValue | NIL
<ParentWindowName>.<TreeControlName>.GetPathValue ( nValue ) --> anPathValue | NIL
<ParentWindowName>.<TreeControlName>.GetPathName ( nValue ) --> acPathName | NIL
<ParentWindowName>.<TreeControlName>.GetDisplayLevel ( nValue ) --> nDisplayColumn
 | NIL
```

- New Set Properties:

```
<ParentWindowName>.<TreeControlName>.NodeFlag ( nValue ) := lBoolean
<ParentWindowName>.<TreeControlName>.ImageIndex ( nValue ) := { iUnSelectItem , iS
electItem }
<ParentWindowName>.<TreeControlName>.AddImage := cImageName
<ParentWindowName>.<TreeControlName>.TextColor := anRGBcolor
<ParentWindowName>.<TreeControlName>.BackColor := anRGBcolor
<ParentWindowName>.<TreeControlName>.LineColor := anRGBcolor
<ParentWindowName>.<TreeControlName>.DynamicForeColor := {|| anRGBcolor }
<ParentWindowName>.<TreeControlName>.DynamicBackColor := {|| anRGBcolor }
<ParentWindowName>.<TreeControlName>.DynamicFont := {|| {cFontName, nFontSize, [ l
Bold, lItalic, lUnderline, lStrikeOut ]} }
<ParentWindowName>.<TreeControlName>.ChangeFontSize := nSize | NIL // Useful for u
se Dynamic Font with more (less) Height than the size of font the Tree control
<ParentWindowName>.<TreeControlName>.ImageList := hImageList
<ParentWindowName>.<TreeControlName>.HasLines := lBoolean
<ParentWindowName>.<TreeControlName>.FullRowSelect := lBoolean
<ParentWindowName>.<TreeControlName>.HasButton ( nValue ) := lBoolean
<ParentWindowName>.<TreeControlName>.Cargo ( nValue ) := xData
```

- New Methods:

```
<ParentWindowName>.<TreeControlName>.Expand ( nValue , lRecursive )
<ParentWindowName>.<TreeControlName>.Collapse ( nValue , lRecursive )
<ParentWindowName>.<TreeControlName>.DisableUpdate
<ParentWindowName>.<TreeControlName>.EnableUpdate
<ParentWindowName>.<TreeControlName>.SetDefaultAllNodeFlag
<ParentWindowName>.<TreeControlName>.SetDefaultNodeFlag ( nValue )
<ParentWindowName>.<TreeControlName>.Sort ( nValue , lRecursive, lCaseSensitive, l
AscendingOrder, nNodePosition )
TREESORT ControlName OF ParentName
[ ITEM nValue ]
[ RECURSIVE lRecursive ]
[ CASESENSITIVE lCaseSensitive ]
[ ASCENDINGORDER lAscendingOrder ]
[ NODEPOSITION nNodePosition ]
nNodePosition = TREESORTNODE_FIRST | TREESORTNODE_LAST | TREESORTNODE_MIX
```

- ***Note:***

`<ParentWindowName>.<TreeControlName>.``IsTrueNode`` (nValue)` --> Only returns .T. if the item contain sub-items (child items).

`<ParentWindowName>.<TreeControlName>.``NodeFlag`` (nValue) := lBoolean` --> This flag allows you to force an Item to be Node (.T.) or not (.F.). This flag only affect the Nodes positions when run the Sort method.

For more information about of the new features see demo SortTreeDir in SAMPLES folder.

Dynamic Font:

```
ARRAY FONT <cFontName> SIZE <nFontSize> [ BOLD ] [ ITALIC ] [ UNDERLINE ] [ STRIKEOUT
] --> { cFontName, nFontSize, lBold, lItalic, lUnderline, lStrikeout }
CREATE ARRAY FONT <cFontName> SIZE <nFontSize> [ BOLD <lBold> ] [ ITALIC <lIitalic> ]
[ UNDERLINE <lUnderline> ] [ STRIKEOUT <lStrikeout> ] --> { cFontName, nFontSize, lBol
d, lItalic, lUnderline, lStrikeout }
```

Various Properties of Windows and Controls:

Address Property

Sets \/ Get the address in an HyperLink control

Syntax:

```
Address <aValue``> (Control Definition)
```

```
<WindowName>.<ControlName>.Address := cAddress
```

```
<WindowName>.<ControlName>.Address --> cAddress
```

Alignment Property

Specifies how a label or hyperlink control content should be aligned

Syntax:

```
Alignment Left | Right | Center
```

AllowAppend Property

Specifies whether new records can be added to a table within a data-bound Grid control. (Available at control definition)

Syntax:

```
AllowAppend <lValue>
```

AllowDelete Property

Specifies whether records can be deleted from a table within a browse control (Available at control definition)

Syntax:

```
AllowDelete <lValue>
```

AllowEdit Property

Specifies whether records can be edited from a table within a browse control (Available at control definition)

Syntax:

```
AllowEdit <lValue>
```

AutoPlay Property

Specifies whether an AnimateBox control starts playing automatically

Syntax:

```
AutoPlay <lValue>
```

AutoSize Property

Specifies whether a GUI object must be sized automatically

Syntax:

```
AutoSize <lValue>
```

AutoSizeMovie Property

Specifies whether a Player control will automatically resize its content

Syntax:

```
AutiSizeMovie <lValue>
```

Sample:

```
@ 200,0 PLAYER Player_1 ;
```

```
WIDTH 100 ;
```

```
HEIGHT 100 ;
```

```
FILE "sample.avi" ;
```

```
SHOWALL
```

AutoSizeWindow Property

Specifies whether a Player control will automatically resize its own window

Syntax:

```
AutoSizeWindow <lValue>
```

Sample:

```
@ 200,0 PLAYER Player_1 ;
```

```
WIDTH 100 ;
```

```
HEIGHT 100 ;
```

```
FILE "sample.avi" ;
```

```
SHOWALL NOAUTOSIZEWINDOW
```

BackColor Property

Specifies the background color for a GUI object

Syntax:

```
BackColor <anValue>
```

Must be specified as a three elements numeric array containing red, green and blue components

Both Property

Specifies whether ‘tickmarks’ will be shown on both sides of a slider control

Syntax:

```
Both <lValue>
```

Bottom Property

Specifies whether a ToolBar o SplitBox control should be shown at the bottom of a window

Syntax:

```
Bottom <lValue>
```

Break Property

Specifies when a splitchild GUI object should be displayed in a new ‘band’

Syntax:

```
Break <lValue>
```

ButtonHeight Property

Specifies the button height for a ToolBar control

Syntax:

```
ButtonHeight <nHeight>
```

ButtonWidth Property

Specifies the button width for a ToolBar control

Syntax:

```
ButtonWidth <nWidth>
```

Buttons Property

Specifies when buttons should be used instead tabs in a Tab control

Syntax:

```
Buttons <lValue>
```

Caption Property

Set \/ Gets caption of a gui object

Syntax:

```
Caption <cValue>``(Control Definition)
```

```
<ParentWindowName>.<ControlName>.Caption [ (nIndex) ] := cCaption
```

```
<ParentWindowName>.<ControlName>.Caption [ (nIndex) ] --> cCaption
```

`nIndex` is available only for **grid**, **radiogroup** and **tab** controls.

CaretPos Property

Set \/ Gets the caret position of a textbox control

Syntax:

```
CaretPos <nValue> (Control Definition)
```

```
<ParentWindowName>.<ControlName>.CaretPos := nCaretPosition
```

```
<ParentWindowName>.<ControlName>.CaretPos --> nCaretPosition
```

Note*:* if you set CaretPos to -1 moves the caret to the end of text and scrolls the text.

CaseConvert Property

Specifies whether a character TextBox input will be automatically converted to lowercase or uppercase

Syntax:

```
CaseConvert Upper | Lower | None
```

Cell Property

Specifies or retrieves the content of a Grid cell

Syntax:

```
<ParentWindowName>.<ControlName>.Cell(nRow,nCol) := CellContent
```

```
<ParentWindowName>.<ControlName>.Cell(nRow,nCol) --> CellContent
```

CellNavigation Property

Specifies whether individual cells can be selected in Grid control

Syntax:

CellNavigation <lValue>

Sample:

```
#include "hmg.ch"
Function Main
   Local aRows [10] [3]
   DEFINE WINDOW Form_1 ;
      AT 0,0 ;
      WIDTH 640 ;
      HEIGHT 400 ;
      TITLE 'Cell Navigation Grid Test' ;
      MAIN
   DEFINE MAIN MENU
      DEFINE POPUP 'File'
         MENUITEM 'Set Value To {5,2}' ACTION Form_1.Grid_1.Value := {5,2}
         MENUITEM 'Set Value To {0,0}' ACTION Form_1.Grid_1.Value := {0,0}
         MENUITEM 'Get Value' ACTION GetSelectionValue()
         MENUITEM 'Delete Item 5' ACTION Form_1.Grid_1.DeleteItem(5)
         MENUITEM 'AddItem' ACTION Form_1.Grid_1.AddItem({'X','X','X'})
         MENUITEM 'Delete All Items' ACTION Form_1.Grid_1.DeleteAllItems
      END POPUP
   END MENU
   aRows [1] := {'Simpson','Homer','555-5555'}
   aRows [2] := {'Mulder','Fox','324-6432'}
   aRows [3] := {'Smart','Max','432-5892'}
   aRows [4] := {'Grillo','Pepe','894-2332'}
   aRows [5] := {'Kirk','James','346-9873'}
   aRows [6] := {'Barriga','Carlos','394-9654'}
   aRows [7] := {'Flanders','Ned','435-3211'}
   aRows [8] := {'Smith','John','123-1234'}
   aRows [9] := {'Pedemonti','Flavio','000-0000'}
   aRows [10] := {'Gomez','Juan','583-4832'}
   DEFINE GRID Grid_1
      ROW 10
      COL 10
      WIDTH 500
      HEIGHT 322
      HEADERS {'Column 1','Column 2','Column 3'}
      WIDTHS {100,100,100}
      ITEMS aRows
      ALLOWEDIT .T.
      CELLNAVIGATION .T.
      ONCHANGE GridChange()
      VALUE { 3 , 2 }
   END GRID
END WINDOW
Form_1.Grid_1.setfocus
CENTER WINDOW Form_1
ACTIVATE WINDOW Form_1
Return

Function GridChange()
   Static OnChangeCounter := 0
   a := this.value
   OnChangeCounter++
   form_1.title := ' Row: ' + alltrim(str(a [1])) + ' Col: ' + alltrim(str(a [2])) + S
pace(10) + 'OnChange Count: ' + alltrim(str(OnChangeCounter))
return

Function GetSelectionValue
   Local a
   a := Form_1.Grid_1.Value
   MsgInfo( Str ( a [1] ) + ' ' + Str ( a [2] ) )
Return nil
```

Center Property

Specifies whether a AnimateBox control content should be aligned to center

Syntax:

`CENTER <lValue>`

CenterAlign Property

Specifies whether a label or hyperlink control content should be alignet to center

Syntax:

`CenterAlign <lValue>`

Check Property

Specifies whether a ToolBar button should work as a CheckButton

Syntax:

`Check <lValue>`

Sample:

```
DEFINE TOOLBAR ToolBar_1 BUTTONSIZE 85,85 FLAT BORDER
   BUTTON Button_1 ;
      CAPTION '&More ToolBars...' ;
      PICTURE 'button1.bmp' ;
      ACTION Modal_Click() TOOLTIP 'ONE'
   BUTTON Button_2 ;
      CAPTION '&Button 2' ;
      PICTURE 'button2.bmp' ;
      ACTION MsgInfo('Click! 2') TOOLTIP 'TWO'
   BUTTON Button_3 ;
      CAPTION 'Button &3' ;
      PICTURE 'button3.bmp' ;
      ACTION MsgInfo('Click! 3') TOOLTIP 'THREE' CHECK
END TOOLBAR
```

Checked Property

Set \/ Gets check state of a menu item

Syntax:

`Checked <lValue>`

`<ParentWindowName>.<MenuItemName>.Checked := lCheckState`

`<ParentWindowName>.<MenuItemName>.Checked --> lCheckState`

Col Property

Set \/ Gets column position of a gui object (window\/control)

Syntax:

`Col <nValue> (Control Definition)`

`<ParentWindowName>.<ControlName>.Col := nCol`

`<ParentWindowName>.<ControlName>.Col --> nCol`

`<ParentWindowName>.Col := nCol`

`<ParentWindowName>.Col --> nCol`

ColumnControls Property

Specifies controls to be used for each column in a Grid Control

Syntax:

`ColumnControls {aControlDef1,aControlDef2,...aControlDefN}`

Control Definition Array:

TEXTBOX

`{ cControlType , cDataType , cInputMask , cFormat }`

`cControlType = 'TEXTBOX' (Required)`

`cDataType = 'CHARACTER' , 'NUMERIC' , 'DATE' (Required)`

`cInputMask = cInputMask (Optional)`

`cFormat = cFormat (Optional)`

DATEPICKER

`{ cControlType , cControlStyle }`

`cControlType = 'DATEPICKER' (Required)`

`cControlStyle = 'DROPDOWN' , 'UPDOWN' (Required)`

TIMEPICKER

`{ cControlType , cTimeFormat }`

`cControlType = 'TIMEPICKER' (Required)`

`cTimeFormat = '', _TIMELONG24H, _TIMELONG12H, _TIMESHORT24H, _TIMESHORT12H (Required)`

COMBOBOX

`{ cControlType , acItems }`

`cControlType 'COMBOBOX' (Required)`

`acItems (Required)`

SPINNER

`{ cControlType , nRangeMin , nRangeMax }`

`cControlType 'SPINNER' (Required)`

`nRangeMin (Required)`

`nRangeMax (Required)`

CHECKBOX

`{ cControlType , cCheckedLabel , cUnCheckedLabel }`

`cControlType 'CHECKBOX' (Required)`

`cCheckedLabel (Required)`

`cUnCheckedLabel (Required)`

Data type for each column will depend control specified.

NUMERIC TEXTBOX : NUMERIC

DATE TEXTBOX : DATE

CHARACTER TEXTBOX : CHARACTER

SPINNER : NUMERIC

COMBOBOX : NUMERIC

CHECKBOX : LOGICAL

Sample:

```
@ 10,10 GRID Grid_1 ;
            WIDTH 620 ;
            HEIGHT 330 ;
            HEADERS {'Column 1','Column 2','Column 3','Column 4',;
                  'Column 5'} ;
            WIDTHS {140,140,140,140,140} ;
            ITEMS aRows ;
            EDIT ;
            COLUMNCONTROLS { {'TEXTBOX','NUMERIC','$ 999,999.99'},;
                  {'DATEPICKER','DROPDOWN'} ,;
                  {'COMBOBOX',{'One','Two','Three'}} , ;
                  { 'SPINNER' , 1 , 20 } , ;
                  { 'CHECKBOX' , 'Yes' , 'No' } }
```

ColumnValid Property

Codeblock array (one element per column) is evaluated at cell editing

Syntax:

`ColumnValid <abValues>`

This.CellValue variable are available at codeblock evaluation.

Sample:

```
COLUMNVALID { ;
  { || This.CellValue > 100 } , ;
  { || This.CellValue = Date() } , ;
  Nil , ;
  Nil , ;
  Nil ;
  }
```

ColumnWhen Property

Codeblock array (one element per column) that is evaluated at cell editing

Syntax:

`ColumnWhen <abValues>`

This.CellValue variable are available at codeblock evaluation.

Sample:

```
COLUMNWHEN { ;
 { || This.CellValue >= 'M' } , ;
 { || This.CellValue >= 'C' } , ;
 { || ! Empty ( This.CellValue ) } ;
 }
```

Cursor Property

Specifies a cursor file or resource for a Window

Syntax:

Cursor <lValue>

Sample:

```
#include "hmg.ch"
Function Main
    DEFINE WINDOW Win_1 ;
       AT 0,0 ;
       WIDTH 400 ;
       HEIGHT 400 ;
       TITLE 'Hello World!' ;
       MAIN ;
       CURSOR "Finger.cur"
 END WINDOW
 ACTIVATE WINDOW Win_1
Return
```

Date Property

Specifies whether a TextBox will be used for entering date data

Syntax:

`Date <lValue>`

DisabledBackColor Property

Specifies the background color for a disabled\/read-only GUI object

Syntax:

`DisabledBackColor <anColor>`

Must be specified as a three elements numeric array containing red, green and blue components

---

DisabledFontColor Property

Specifies the text color for a disabled\/read-only GUI object

Syntax:

```
DisabledFontColor <anColor> (Control Definition)
<WindowName>.<ControlName>.DisabledFontColor := anColor
<WindowName>.<ControlName>.DisabledFontColor --> anColor
```

This should be specified as a three element numeric array containing red, green and blue components.

DisplayEdit Property

Specified whether a ComboBox control can be edited

Syntax:

`DisplayEdit <lValue>`

Sample:

```
#include "hmg.ch"
Function Main
      DEFINE WINDOW Form_1 ;
            AT 0,0 ;
            WIDTH 400 ;
            HEIGHT 200 ;
            TITLE 'ComboBox Demo' ;
            MAIN
            DEFINE MAIN MENU
                  DEFINE POPUP 'Test'
                        MENUITEM 'Get Value' ACTION MsgInfo(Str(Form_1.Combo_1.Value))
                        MENUITEM 'Set Value' ACTION Form_1.Combo_1.Value := 1
                        MENUITEM 'Get DisplayValue' ACTION MsgInfo( Form_1.Combo_1.Dis
playValue )
                        MENUITEM 'Set DisplayValue' ACTION Form_1.Combo_1.DisplayValue
 := 'New Text'
                        MENUITEM 'Set Item' ACTION Form_1.Combo_1.Item (3) := 'New Tex
t'
                        MENUITEM 'Get Item' ACTION MsgInfo ( Form_1.Combo_1.Item (3) )
                  END POPUP
            END MENU
            @ 10,10 COMBOBOX Combo_1 ;
                  ITEMS { 'A' , 'B' , 'C' } ;
                  VALUE 1 ;
                  DISPLAYEDIT ;
                  ON DISPLAYCHANGE PlayBeep()
      END WINDOW
      CENTER WINDOW Form_1
      ACTIVATE WINDOW Form_1
Return
```

DisplayItems Property

Set data display in Browse Control

Syntax:

`DisplayItems <aaValues>`

This property allows to control data display in the control. It must be specified as an array \(one element for each browse column\). Each element \(if specified\) must be a two dimensional array. The first column in the array must contain the text to be shown to the user. The second column must contain the ID for each array row. The array will be searched for a corresponding ID in the table to show the right text in each cell. If no correspondence is found, the cell will be blank.

DisplayValue Property Specifies or retrieves the value for the editable part of a ComboBox control

Syntax:

```
DisplayValue <lValue> (Control Definition)
<ParentWindowName>.<ControlName>.DisplayValue := nCol
<ParentWindowName>.<ControlName>.DisplayValue --> nCol
```

DragItems Property

Specifies whether the items in a ListBox can be arranged by the user

Syntax:

`DragItems <lValue>`

DropDown Property

Specifies whether a Toolbar button will be used to open a dropdown menu

Syntax:

`DropDown <lValue>`

The button will show separate section containing a down arrow to open the menu.

```
      DEFINE TOOLBAR ToolBar_a BUTTONSIZE 30,30 FONT 'Arial' SIZE 8  FLAT
            BUTTON Button_1a ;
                  CAPTION '&Undo' ;
                  PICTURE 'button4.bmp' ;
                  ACTION MsgInfo('Undo')
            BUTTON Button_3a ;
                  CAPTION '&Close' ;
                  PICTURE 'button6.bmp' ;
                  ACTION MsgInfo('Close') ;
                  DROPDOWN
            DEFINE DROPDOWN MENU BUTTON Button_3a
                  ITEM 'Menu 1'     ACTION MsgInfo('Menu 1')
                  ITEM 'Menu 2'     ACTION MsgInfo('Menu 2')
            END MENU
      END TOOLBAR
```

DroppedWidth

Specifies the width of the list dropped when a ComboBox control arrow is clicked

Syntax:

`DroppedWidth <lValue>`

DynamicBackColor Property

Dynamically Set The Background Color For a Grid Or Browse Cell

Syntax:

`DynamicBackColor <abValues>`

Specifies a codeblock array (one element per column) evaluated for each cell at any Grid or Browse change to determine the background color.

This.CellRowIndex, This.CellColIndex and This.CellValue variables are available at codeblock evaluation.

Sample:

```
bColor := { || if ( This.CellRowIndex/2 == int(This.CellRowIndex/2) , ;
                   {128,128,128} , {192,192,192} ) }
DYNAMICBACKCOLOR { bColor , bColor, bColor, bColor, bColor, bColor }
```

DynamicDisplay Property

Specifies a code block array containing field dysplay-processing data for a data-bound Grid control.

Syntax:

`DynamicDisplay <abValues>`

DynamicForeColor Property

Dynamically Set The Foreground Color For a Grid Or Browse Cell

Syntax:

`DynamicForeColor <abValues>`

Specifies a codeblock array (one element per column) evaluated for each cell at any Grid or Browse change to determine the foreground color.

This.CellRowIndex, This.CellColIndex and This.CellValue variables are available at codeblock evaluation.

Sample:

```
bColor := { || if ( This.CellRowIndex/2 == int(This.CellRowIndex/2) , ;
                         {128,128,128} , {192,192,192} ) }
DYNAMICBACKCOLOR { bColor , bColor, bColor, bColor, bColor, bColor
```

Enabled Property

Set \/ Gets enabled state of a control

Syntax:

```
<WindowName>.<ControlName>.Enabled := lEnabledState
<WindowName>.<ControlName>.Enabled --> lEnabledState
```

ErrorDlg Property

Inhibits or allows displaying of player errors to users

Syntax:

`ErrorDlg <lValue>`

Sample:

```
@ 200,0 PLAYER Player_1 ;
   WIDTH 100 ;
   HEIGHT 100 ;
   FILE "sample.avi" ;
   SHOWALL NOERRORDLG
```

Field Property

Links controls (RICHEDITBOX, TEXTBOX, EDITBOX, CHECKBOX and DATEPICKER) to a table field.

Syntax:

```
Field <xcValue>
```

Fields Property

A character array (one element per column) specifying fields for a Browse control

Syntax:

```
Fields <acValues>
```

Sample:

```
DEFINE BROWSE Browse_1
 COL 10
 ROW 10
 WIDTH 610
 HEIGHT 390
 HEADERS { 'Code' , 'First Name' , 'Last Name' }
 WIDTHS { 150 , 150 , 150 }
 WORKAREA Test
 FIELDS { 'Test->Code' , 'Test->First' , 'Test->Last' }
 VALUE 1
END BROWSE
```

File Property

Specifies the file name (as a character string) for a Player or AnimateBox control

Syntax:

```
File <cValue>
```

Flat Property

Set 'Flat' Style For GUI Objects

Syntax:

```
Flat <lValue>
```

Specified whether ‘flat’ style will be used to show a StatusBar item, the buttons in a Tab control (‘buttons’ style) a ToolBar or a Button conttrol.

FocusedControl Property

Retrieves the focused control name of a Window

Syntax:

`<WindowName>.FocusedControl --> cFocusedControlName`

FontName Property

Set \/ Gets gui object's font name

Syntax:

```
FontName <anValue> (Control Definition)
<ParentWindowName>.<ControlName>.FontName := cFontName
<ParentWindowName>.<ControlName>.FontName --> cFontName
```

FontSize Property

Set \/ Gets gui object's font size

Syntax:

```
FontSize <nValue> (Control Definition)
<ParentWindowName>.<ControlName>.FontSize := nFontSize
<ParentWindowName>.<ControlName>.FontSize --> nFontSize
```

FontBold Property

Set \/ Gets gui object's font bold flag

Syntax:

```
FontBold <lValue> (Control Definition)
<ParentWindowName>.<ControlName>.FontBold := lFontBold
<ParentWindowName>.<ControlName>.FontBold --> lFontBold
```

FontColor Property

Specifies the text color for a GUI object

Syntax:

```
FontColor <anValue> (Control Definition)
<WindowName>.<ControlName>.FontColor := aFontColor
<WindowName>.<ControlName>.FontColor --> aFontColor
```

This should be specified as a three element numeric array containing red, green and blue components.

FontItalic Property

Set \/ Gets gui object's font italic flag

Syntax:

```
FontItalic <lValue> (Control Definition)
<ParentWindowName>.<ControlName>.FontItalic := lFontItalic
<ParentWindowName>.<ControlName>.FontItalic --> lFontItalic
```

FontUnderLine Property

Set \/ Gets gui object's font underline flag

Syntax:

```
 FontUnderLine <lValue> (Control Definition)
 <ParentWindowName>.<ControlName>.FontUnderline := lFontUnderline
 <ParentWindowName>.<ControlName>.FontUnderline --> lFontUnderline
```

FontStrikeOut Property

Set \/ Gets gui object's font strikeout flag

Syntax:

```
FontStrikeout <lValue> (Control Definition)
<ParentWindowName>.<ControlName>.FontStrikeOut := nFontStrikeOut
<ParentWindowName>.<ControlName>.FontStrikeOut --> nFontStrikeOut
```

Format Property

Specifies the edit format string for a TextBox|TimePicker control

Syntax:

`Format <cFormat>`

TextBox control

Format String (Allowed in Numeric Textbox Only):

C : Displays CR after positive numbers

X : Displays DB after negative numbers

(: Encloses negative numbers in parentheses

E : Displays numbers in British format

TimePicker control

_TIMELONG24H : 24 Hour Long Format

_TIMESHORT24H : 24 Hour Short Format

_TIMELONG12H : 12 Hour Long Format (am|pm)

_TIMESHORT12H : 12 Hour Short Format (am|pm)

GripperText Property

Specifies the text of the ‘Gripper’ for a SplitChild GUI object

Syntax:

```
GripperText <cText>
```

Group Property

Specifies whether toolbar buttons should be work as radio buttons

Syntax:

```
Group <lValue>
```

HandCursor Property

Specifies whether a hand cursor will be used in an Hyperlink control

Syntax:

`HandCursor <lValue>`

Header Property

Sets\/gets the header for a Grid or Browse control specific column

Syntax:

`<WindowName>.<ControlName>.Header(nColumn)`

It must be specified as a character string.

Sample:

```
#include "hmg.ch"
Function Main
   Local aRows [2] [3]
   aRows [1]   := {'Simpson','Homer','555-5555'}
   aRows [2]   := {'Mulder','Fox','324-6432'}
   DEFINE WINDOW Form_1 ;
          AT 0,0 ;
          WIDTH 640 HEIGHT 480 ;
          TITLE 'HMG Demo' ;
          MAIN
          @ 50,50 GRID Grid_1 ;
                             WIDTH 200 ;
                             HEIGHT 330 ;
                             HEADERS {'Last Name','First Name','Phone'} ;
                             WIDTHS {140,140,140};
                             ITEMS aRows ;
                             VALUE 1
      Form_1.Grid_1.Header(1) := ‘New Header’
   END WINDOW
   Form_1.Activate
Return
```

Headers Property

Set the headers for a Grid or Browse control

Syntax:

`Headers <acValues>`

It must be specified as a character string array.

Sample:

See sample from **Header Property** above.

HeaderImages Property

Specifies images for Grid and Browse headers

Syntax:

`HeaderImages (acValue)`

This is a character array containing image filenames or resourcenames (one for each column).

To change a header's image at runtime you must specify the column position as argument.

Sample (Control definition):

```
DEFINE GRID Grid_1
 ROW 10
 COL 10
 WIDTH 500
 HEIGHT 330
 HEADERS {'Last Name','First Name','Phone'}
 WIDTHS {140,140,140}
 ITEMS LoadItems()
 VALUE 1
 HEADERIMAGES { '00.bmp' , '01.bmp' , '02.bmp' }
 END GRID
```

Sample (Set image at runtime):

`Form_1.Grid_1.HeaderImages(1) := '03.bmp'`

Sample (Get Image at runtime):

`MsgInfo ( Form_1.Grid_1.HeaderImages(1) )`

Height Property

Set \/ Gets heigth of a gui object

Syntax:

```
Height <nValue> (Control Definition)
<ParentWindowName>.<ControlName>.Height := nHeight
<ParentWindowName>.<ControlName>.Height --> nHeight
```

HelpId Property

Allows to set the help id for context help

Syntax:

`HelpId <nValue>`

It may be accessed pressing F1 or using window help button.

Horizontal Property

Specifies whether gripper in a SplitBox is horizontal

Syntax:

`Horizontal <lValue>`

HotTrack Property

Specifies ‘Hot tracking’ of Tab pages

Syntax:

`Hottrack <lValue>`

HScrollBar Property

Specifies whether a horizontal scrollbar will be included in an Edit control

Syntax:

`HScrollBar <lValue>`

Sample (Control Definition):

```
DEFINE EDITBOX Edit_1
   ROW 60
   COL 230
   WIDTH 120
   HEIGHT 120
   VALUE ""
   HSCROLLBAR .T.
   VSCROLLBAR .T.
END EDITBOX
```

Icon Property

Specifies an icon for a StatusBar item or a Window

Syntax:

```
Icon <cValue>
```

Image Property

Specifies a character array containing image filenames or reourcenames to be used with Browse, ComboBox, Grid and Tree controls (a character string for MenuItems)

Syntax:

```
Image <acValues>
```

For MenuItems, it must be specified as a character string containing the image filename or resourcename

---

ImageSize, ImageWidth, ImageHeight Property

Specifies image width for toolbar buttons

Syntax:

```
ImageSize nWidth, nHeight
```

Alternative Syntax:

```
ImageWidth <nWidth>
```

```
ImageHeight <nHeight>
```

Sample:

```
DEFINE TOOLBAR ToolBar_b BUTTONSIZE 40,40 IMAEGESIZE 29, 20 FONT 'ARIAL' SIZE 8 FLAT
 BUTTON Button_1b ;
 CAPTION 'More ToolBars...' ;
 PICTURE 'button7.bmp' ;
 ACTION MsgInfo('Click! 2');
 BUTTON Button_2b ;
 CAPTION 'Button 2' ;
 PICTURE 'button7.bmp' ;
 ACTION MsgInfo('Click! 2');
 SEPARATOR
 BUTTON Button_3b ;
 CAPTION 'Button 3' ;
 PICTURE 'button7.bmp' ;
 ACTION MsgInfo('Click! 3')
 END TOOLBAR
```

Increment Property

Specifies numeric increment (when the user clicks up and down arrows) in a Spinner control

Syntax:

`Increment <nValue>`

Sample:

```
#include "hmg.ch"
Function Main
 DEFINE WINDOW Form_Main ;
 AT 0,0 ;
 WIDTH 640 HEIGHT 480 ;
 TITLE "Main Window" ;
 MAIN
 @ 150,250 SPINNER Spinner_2 ;
 RANGE 0,100 ;
 VALUE 5 ;
 WIDTH 100 ;
 TOOLTIP "Range 0,100 WRAP READONLY INCREMENT 5" ;
 WRAP INCREMENT 5
 END WINDOW
 ACTIVATE WINDOW Form_Main
Return Nil
```

InputItems Property

Allows to control data input in the Browse control

Syntax:

```
InputItems <aaValues>
```

This property is an array (one element for each browse column). Each element (if specified) must be a two dimensional array. The first column in the array must contain the data to be shown to the user. The second column must contain the data to be stored in the table (ID) for each text row in the array.

InputMask Property

Specifies the edit mask for a TextBox control

Syntax:

```
InputMask <cValue>
```

InputMask String (Numeric Textbox):

9 Displays digits

$ Displays a dollar sign in place of a leading space

* Displays an asterisk in place of a leading space

. Specifies a decimal point position

, Specifies a comma position

InputMask String (Non-Numeric Textbox):

9 Digits

A Alphabetic Characters

! Converts an alphabetic character to uppercase

(All other characters ar included in text in the position indicated by

the mask)

Interval Property

Specifies interval in milliseconds for timer control

Syntax:

```
Interval <nValue>
```

Item Property

Set \/ Get item text in a listbox, combobox, grid and Statusbar controls

Syntax:

```
<WindowName>.<ControlName>. Item (<nItem>) := ItemText
<WindowName>.<ControlName> . Item (<nItem>) --> ItemText
```

<ItemText> type is character for lists and combos, and array for grids

(in this case, the array length, is equal to column count)

Items Property

Set items for a ComboBox at control definition

Sy ntax:

`Items <acValue> (Control Definition)`

ItemCount Property

Get item count in a listbox, combobox or grid

Syntax:

`ItemCount <nValue>`

`<WindowName>.<ControlName>. ItemCount --> nItemCount`

ItemImages Property

Specifies Item Images For a Tree Item

Syntax:

`ItemImages <acValues>`

A character array (one or two elements) containing image filenames or resourcenames for tree items when unselected (first array item) and selected (optional second array item).

Sample:

```
DEFINE TREE Tree_1 AT 10,10 WIDTH 200 HEIGHT 400 VALUE 3;
NODEIMAGES { "doc_fl.bmp" };
ITEMIMAGES { "cl_fl.bmp", "op_fl.bmp" };
NOROOTBUTTON
    NODE 'Root' IMAGES {'world'}
       TREEITEM 'Item 1.1'
       TREEITEM 'Item 1.2'
       TREEITEM 'Item 1.3'
       NODE 'Docs'
          TREEITEM 'Docs 1' IMAGES {'varios'}
       END NODE
       NODE 'Notes' IMAGES {'varios'}
          TREEITEM 'Notes 1'
          TREEITEM 'Notes 2'
          TREEITEM 'Notes 3'
          TREEITEM 'Notes 4'
          TREEITEM 'Notes 5'
       END NODE
       NODE 'Books' IMAGES {'book'}
          TREEITEM 'Book 1' IMAGES {'doc'}
          TREEITEM 'Book 2' IMAGES {'doc'}
          NODE 'Book 3'IMAGES {'book'}
            TREEITEM 'Book 3.1'
            TREEITEM 'Book 3.2'
         END NODE
      END NODE
    END NODE
END TREE
```

ItemSource Property

Links a ComboBox control to a table field

Syntax:

`ItemSource <xcValue>`

When ITEMSOURCE property is set to a fieldname, ‘Value' property uses the physical record number, as in browse.

If you set the VALUESOURCE property to a fieldname, its containt is returned instead the physical record number.

Sample:

```
#Include "hmg.ch"
Function Main()
DEFINE WINDOW Form_1 ;
   AT 0,0 ;
   WIDTH 365 ;
   HEIGHT 120 ;
   TITLE "Exemplos ComboBox New" ;
   MAIN ;
   ON INIT OpenTables() ;
   ON RELEASE CloseTables()
   DEFINE MAIN MENU
      DEFINE POPUP '&Test'
        MENUITEM 'Get Value' ACTION MsgInfo( Str ( Form_1.Combo_1.Value ) )
        MENUITEM 'Set Value' ACTION Form_1.Combo_1.Value := 2
        MENUITEM 'Refresh' ACTION Form_1.Combo_1.Refresh
        MENUITEM 'DisplayValue' ACTION MsgInfo ( Form_1.Combo_1.DisplayValue )
      END POPUP
   END MENU
   @010,010 COMBOBOX Combo_1 ;
      ITEMSOURCE CIDADES->DESCRICAO ;
      VALUE 5 ;
      WIDTH 200 HEIGHT 100 ;
      FONT "Arial" SIZE 9 ;
      TOOLTIP "Combo Cidades"
END WINDOW
ACTIVATE WINDOW Form_1
Return

Procedure Opentables()
   Use Cidades Alias Cidades New
   Index On Descricao To Cidades1
Return

Procedure CloseTables()
   Use
Return
```

Justify Property

Numeric array (one element for each column) containing justification constants for Browse and Grid controls

Syntax:

`Justify (nValues)`

Sample (Browse):

```
DEFINE BROWSE Browse_1
 COL 10
 ROW 10
 WIDTH 610
 HEIGHT 390
 HEADERS { 'Code' , 'First Name' , 'Last Name' }
 WIDTHS { 150 , 150 , 150 }
 WORKAREA Test
 FIELDS { 'Test->Code' , 'Test->First' , 'Test->Last' }
 VALUE 1
 JUSTIFY { BROWSE_JTFY_LEFT,BROWSE_JTFY_CENTER, BROWSE_JTFY_CENTER }
END BROWSE
```

Sample (Grid):

```
 @ 10,10 GRID Grid_1 ;
 WIDTH 760 ;
 HEIGHT 240 ;
 HEADERS {'Last Name','First Name','Phone'} ;
 WIDTHS {140,140,140};
 ITEMS aRows ;
JUSTIFY { GRID_JTFY_LEFT,GRID_JTFY_RIGHT, GRID_JTFY_RIGHT }
```

Left Property

Specifies whether the content of a GUI object will be left aligned

Syntax:

`Left (lValue)`

Lines Property

Specifies whether lines are shown in a Grid or Browse control

Syntax:

`Lines <lValue>`

Lock Property

Specifies whether Browse control should lock records when saving changes made interactively by the user

Syntax:

`Lock <lValue>`

Sample:

```
 @ 10,10 BROWSE Browse_1 ;
 WIDTH 610 ;
 HEIGHT 390 ;
 HEADERS { 'Code' , 'First Name' , 'Last Name' } ;
 WIDTHS { 150 , 150 , 150 } ;
 WORKAREA Test ;
 FIELDS { 'Test->Code' , 'Test->First' , 'Test->Last' } ;
 DELETE ;
 LOCK ;
 EDIT ;
APPEND
```

LowerCase Property

Specifies whether a carácter TextBox input will be automatically converted to lowercase

Syntax:

`LowerCase <lValue>`

MaxLength Property

Specifies the maximun input length for a TextBox control

Syntax:

`MaxLength <nValue>`

Menu Property

Shows or Hides the Menu button from view on the Player control toolbar

Syntax:

`Menu <lValue>`

Sample:

```
@ 200,0 PLAYER Player_1 ;
WIDTH 100 ;
HEIGHT 100 ;
FILE "sample.avi" ;
SHOWALL NOMENU
```

MultiLine Property

For buttons, specifies whether the Button should wraps the text to multiple lines when the text is too long to fit on a single line. For Tab control, specified how the tabs are shown when not fit in the control width.

Syntax:

`MultiLine <lValue>`

MultiSelect Property

Specifies whether multiple selection is allowed in ListBox and Grid controls

Syntax:

`MultiSelect <lValue>`

Name Property

Retrieves the name of a GUI object

Syntax:

```
<Windowname> [. <ControlName>] . Name --> cObjectName
```

This property is read-only after control definiton.

NotifyIcon Property

Specifies a notification icon to be shown in the system tray

Syntax:

```
NotifyIcon <cValue>
```

NotifyToolTip Property

Specifies a tooltip text to be shown in the system tray

Syntax:

```
NotifyToolTip <cValue>
```

NodeImages Property

Specifies Node Images For a Tree Control

Syntax:

```
NodeImages <acImages>
```

A character array (one or two elements) containing image filenames or resourcenames for tree nodes when unselected (first array item) and selected (optional second array item).

Sample:

```
DEFINE TREE Tree_1 AT 10,10 WIDTH 200 HEIGHT 400 VALUE 3;
    NODEIMAGES { "doc_fl.bmp" };
    ITEMIMAGES { "cl_fl.bmp", "op_fl.bmp" };
    NOROOTBUTTON
       NODE 'Root' IMAGES {'world'}
          TREEITEM 'Item 1.1'
          TREEITEM 'Item 1.2'
          TREEITEM 'Item 1.3'
          NODE 'Docs'
             TREEITEM 'Docs 1' IMAGES {'varios'}
          END NODE
       NODE 'Notes' IMAGES {'varios'}
          TREEITEM 'Notes 1'
          TREEITEM 'Notes 2'
          TREEITEM 'Notes 3'
          TREEITEM 'Notes 4'
          TREEITEM 'Notes 5'
       END NODE
       NODE 'Books' IMAGES {'book'}
          TREEITEM 'Book 1' IMAGES {'doc'}
          TREEITEM 'Book 2' IMAGES {'doc'}
          NODE 'Book 3'IMAGES {'book'}
             TREEITEM 'Book 3.1'
             TREEITEM 'Book 3.2'
          END NODE
      END NODE
   END NODE
END TREE
```

Numeric Property

Specifies whether a TextBox will be used for entering numeric data

Syntax:

```
Numeric <lValue>
```

Parent Property

Specifies the parent window name for a GUI object

Syntax:

```
Parent <ParentWindowName>
```

PictAlignment Property

Specifies the image alignment in a Button control

Syntax:

```
PictAlignment Top | Left | Right | Bottom
```

You can specify picture alignment via TOP , LEFT , RIGHT and BOTTOM clauses (standard syntax) or setting the new 'PictAlignment' property when alternate syntax is used (the default aligment is 'Top').

This feature requires Windows XP or later version.

Picture Property

Set \/ Gets image name of a gui object

Syntax:

```
Picture <cImageName>
<ParentWindowName>.<ControlName>.Picture := cImageName
<ParentWindowName>.<ControlName>.Picture --> cImageName
```

PlayBar Property

Shows or Hides the Player toolbar

Syntax:

```
PlayBar <lValue>
```

Sample:

```
@ 200,0 PLAYER Player_1 ;
   WIDTH 100 ;
   HEIGHT 100 ;
   FILE "sample.avi" ;
   SHOWALL NOOPLAYBAR
```

RangeMax Property

Specifies the maximun value that can be entered in a Spinner control

Syntax:

```
RangeMax <nValue>
```

RangeMin Property

Specifies the minimun value that can be entered in a Spinner control

Syntax:

```
RangeMin <nValue>
```

ReadOnly Property

Specifies whether a GUI object is read-only

Syntax:

```
ReadOnly <lValue>
```

ReadOnlyFields Property

Specifies read-only columns on a Browse control as a logical array

Syntax:

```
ReadOnlyFields <alValues>
```

RecNo Property

Set \/ Gets selected item in a data-bound Grid control based on its physical record number.

Syntax:

```
<WindowName>.<ControlName>.RecNo := nRecNo
<WindowName>.<ControlName>.RecNo --> nRecNo
```

RightAlign Property

Specifies whether the content of a GUI object will be right aligned (especially useful for numeric textbox)

Syntax:

`RightAlign <lValue>`

RightText Property

Specifies whether the buttons in a ToolBar control should be right aligned

Syntax:

`RightText <lValue>`

Sample:

```
DEFINE TOOLBAR ToolBar_1 FLAT BUTTONSIZE 100,25 BOTTOM RIGHTTEXT IMAGESIZE 25,25
   BUTTON Button_1 ;
      CAPTION '&Undo' ;
      PICTURE 'button4.bmp' ;
      ACTION MsgInfo('UnDo Click!')
   BUTTON Button_2 ;
      CAPTION '&Save' ;
      PICTURE 'button5.bmp' ;
      ACTION MsgInfo('Save Click!')
   BUTTON Button_3 ;
      CAPTION '&Close' ;
      PICTURE 'button6.bmp' ;
      ACTION Form_2.Release
END TOOLBAR
```

RowSource Property

Specifies workarea name for a Grid COntrol

Syntax:

`RowSource <cValue>`

RootButton Property

Specifies whether a button should be shown for root element in a Tree control

Syntax:

`RootButton <lValue>`

[row]: **Row Property**

Set \/ Gets row position of a gui object

Syntax:

```
Row <nValue> (Control Definition)
<ParentWindowName>.<ControlName>.Row := nCol
<ParentWindowName>.<ControlName>.Row --> nCol
```

Separator Property

Specifies whether a ToolBar button includes a separator

Syntax:

```
Separator <lValue>
```

Sample:

```
DEFINE TOOLBAR ToolBar_1 BUTTONSIZE 85,85 FLAT BORDER
   BUTTON Button_1 ;
      CAPTION '&More ToolBars...' ;
      PICTURE 'button1.bmp' ;
      ACTION Modal_Click() TOOLTIP 'ONE'
   BUTTON Button_2 ;
      CAPTION '&Button 2' ;
      PICTURE 'button2.bmp' ;
      ACTION MsgInfo('Click! 2') TOOLTIP 'TWO' SEPARATOR
   BUTTON Button_3 ;
      CAPTION 'Button &3' ;
      PICTURE 'button3.bmp' ;
      ACTION MsgInfo('Click! 3') TOOLTIP 'THREE' CHECK
END TOOLBAR
```

ShowAll Property

Specifies whether all interface elements should be shown in a Player control

Syntax:

```
ShowAll <lValue>
```

Sample:

```
@ 275,110 PLAYER Player_2 ;
   WIDTH 100 ;
   HEIGHT 25 ;
   FILE "sample.mp3" ;
   SHOWALL
```

ShowHeaders Property

Specifies whether headers should be shown in a Grid control

Syntax:

```
ShowHeaders <lValue>
```

ShowNone Property

Specifies whether entering null date|time is allowed in a DatePicker|TimePicker control

Syntax:

```
ShowNone <lValue>
```

Sample:

```
@ 230,310 DATEPICKER Date_4 ;
   VALUE CTOD("01/01/2001") ;
   TOOLTIP "DatePicker Control ShowNone" ;
   SHOWNONE
```

If **SHOWNONE** clause is used and shownone checkbox is unchecked:

- **DATEPICKER** return an empty date
- **TIMEPICKER** return an empty string

ShowPosition Property

Determines whether the current position within the content of the Player control should be shown

Syntax:

```
ShowPosition <lValue>
```

Sizable Property

Specifies whether the border of a window allows size it

Syntax:

```
Sizable <lValue>
```

Smooth Property

Determines whether an altérnate style should be used to show the ProgressBar control

Syntax:

```
Smooth <lValue>
```

Sort Property

Specifies whether a ComboBox or ListBox should be automatically sorted

Syntax:

```
Sort <lValue>
```

Spacing Property

Sets the spacing between ítems in a RadioGroup control

Syntax:

```
Spacing <nValue>
```

Stretch Property

Specifies if the content of an Image control should be stretched to completely fill the control area

Syntax:

```
Stretch <lValue>
```

StrictWidth Property

Forces ToolBar control buttons width, preventing automatic width changes

Syntax:

```
StrictWidth <lValue>
```

Style Property

Specified Whether 'Flat' or 'Raised' Style Should be Used for StatusItem Objects

Syntax:

```
Style Flat | Raised
```

TabStop Property

Spevifies whether a GUI object should receive the focus when the user changes it using the [Tab] key

Syntax:

`TabStop <lValue>`

TickMarks Property

Specifies whether ‘TickMarks’ are shown in Slider control

Syntax:

`TickMarks <lValue>`

Title Property

Set \/ Gets title of a window

Syntax:

```
Title <cValue> (Control Definition)
<WindowName>.Title := cTitle
<WindowName>.Title --> cTitle
```

Today Property

Specifies whether the today date should be shown in a MonthCalendar control

Syntax:

`Today <lValue>`

Sample:

```
@ 10,10 MONTHCALENDAR Month_1 ;
   VALUE date() ;
   NOTODAY
```

TodayCircle Property

Specifies whether the today date should be circled in a MonthCalendar control

Syntax:

`TodayCircle <lValue>`

Sample:

```
@ 210,10 MONTHCALENDAR Month_2 ;
   VALUE CTOD("01/01/2001") ;
   FONT "Courier" SIZE 12 ;
   NOTODAYCIRCLE
```

ToolTip Property

Set \/ Gets gui object's tooltip text

Syntax:

```
ToolTip <cValue>
<ParentWindowName>.<ControlName>.ToolTip := cToolTipText
<ParentWindowName>.<ControlName>.ToolTip --> cToolTipText
```

Top Property

Specifies whether Slider control ‘TickMarks’ are located at the top of control

Syntax:

`Top <lValue>`

Transparent Property

Specifies whether the background of a GUI object should be transparent

Syntax:

`Transparent <lValue>`

UpDown Property

Specifies whether up and down arrows should be included in a DatePicker control

Syntax:

`UpDown <lValue>`

Sample:

```
@ 230,310 DATEPICKER Date_4 ;
   VALUE CTOD("01/01/2001") ;
   TOOLTIP "DatePicker Control UpDown" ;
   UPDOWN
```

UpperCase Property

Specifies whether a carácter TextBox input will be automatically converted to uppercase

Syntax:

`UpperCase <lValue>`

Valid Property

Codeblock array (one element per column) that is evaluated after field editing for Browse control

Syntax:

`Valid <abValues>`

Sample:

```
@ 10,10 BROWSE Browse_1 ;
   WIDTH 610 ;
   HEIGHT 390 ;
   HEADERS { 'Code' , 'First Name' , 'Last Name' } ;
   WIDTHS { 150 , 150 , 150 } ;
   WORKAREA Test ;
   FIELDS { 'Test->Code' , 'Test->First' , 'Test->Last' } ;
   VALUE 1 ;
   EDIT ;
   VALID { { || MemVar.Test.Code <= 1000 } , { || !Empty(MemVar.Test.First) } , { || !
Empty(MemVar.Test.Last) } } ;
   VALIDMESSAGES { 'Code Range: 0-1000', 'First Name Cannot Be Empty', Nil }
```

ValidMessages Property

Character array (one element per column) with messages to show when valid procedure returns .F.

Syntax:

`ValidMessages <lValue>`

Sample:

See above **Valid Property** Sample.

Value Property

Sets \/ Gets gui object's value

Syntax:

```
Value <xValue> (Control Definition)
<WindowName>.<ControlName>.Value := <Value>
<WindowName>.<ControlName>.Value -> <Value>
```

<Value> type depends upon the control type:

TIMEPICKER : Character.

DATEPICKER : Date.

MONTHCALENDAR: Date

TEXTBOX : Character.

TEXTBOX (Numeric) : Numeric.

TEXTBOX (Password) : Character.

RADIOGROUP : Numeric (Option Selected).

CHECKBOX : Logical.

COMBOBOX : Numeric (Row Selected).

LISTBOX : Numeric (Row Selected).

LISTBOX (Multiselect): Numeric Array (Rows Selected).

GRID (Standard): Numeric (Row Selected).

GRID (Multiselect): Numeric Array (Rows Selected).

GRID (CellNavigation): Numeric Array (Row and col selected).

EDITBOX : Character.

TAB : Numeric (Active Page).

SPINNER : Numeric.

SLIDER : Numeric.

BROWSE: Numeric (Selected Record Number (RecNo()))

TIMER : Numeric (Write Only)

PROGRESSBAR : Numeric (Write Only)

TOOLBAR BUTTON: Logical. .T. when button is selected (Check style) .

IPADDRESS: Numeric array (four elements).

TREE: Selected item ID.

LABEL: Character.

ValueSource Property

Setting VALUESOURCE property to a fieldname, its containt is returned instead the physical record number for ComboBox control

Syntax:

`ValueSource <xcValue>`

This property is used by ComboBox control linked to a table field via ‘ItemSource’ property.

Sample:

```
Function Main()
   DEFINE WINDOW Form_1 ;
      AT 0,0 ;
      WIDTH 365 ;
      HEIGHT 120 ;
      TITLE "Exemplos ComboBox New" ;
      MAIN ;
      ON INIT OpenTables() ;
      ON RELEASE CloseTables()
      DEFINE MAIN MENU
         DEFINE POPUP '&Test'
            MENUITEM 'Get Value' ACTION MsgInfo( Form_1.Combo_1.Value )
            MENUITEM 'Set Value' ACTION Form_1.Combo_1.Value := 2
            MENUITEM 'Refresh' ACTION Form_1.Combo_1.Refresh
         END POPUP
      END MENU
      @010,010 COMBOBOX Combo_1 ;
         ITEMSOURCE CIDADES->DESCRICAO ;
         VALUESOURCE CIDADES->DESCRICAO ;
         WIDTH 200 ;
         FONT "Arial" SIZE 9 ;
         TOOLTIP "Combo Cidades"
   END WINDOW
   CENTER WINDOW Form_1
   ACTIVATE WINDOW Form_1
Return

Procedure Opentables()
   Use Cidades Alias Cidades New
   Index On Descricao To Cidades1
Return

Procedure CloseTables()
  Use
Return
```

Vertical Property

Specifies vertical orientation for ProgressBar, Slider and Tab controls

Syntax:

```
Vertical <lValue>
```

Virtual Property

Specifies whether a Grid control allow virtual rows

Syntax:

```
Virtual <lValue>
```

This allows handling of millons rows without performance penalty.

When using virtual Grids you must avoid to use **Item property** and **additem method**. It can generate unexpected results.

Sample:

```
#include "hmg.ch"
Function Main
   DEFINE WINDOW Form_1 ;
      AT 0,0 ;
      WIDTH 450 ;
      HEIGHT 400 ;
      TITLE 'Hello World!' ;
      MAIN
      DEFINE MAIN MENU
         DEFINE POPUP 'File'
            MENUITEM 'Change ItemCount' ACTION Form_1.Grid_1.ItemCount := Val(InputBox
('New Value','Change ItemCount'))
         END POPUP
      END MENU
      @ 10,10 GRID Grid_1 ;
         WIDTH 400 ;
         HEIGHT 330 ;
         HEADERS {'Column 1','Column 2','Column 3'} ;
         WIDTHS {140,140,140};
         VIRTUAL ;
         ITEMCOUNT 100000000 ;
         ON QUERYDATA QueryTest()
  END WINDOW
  CENTER WINDOW Form_1
  ACTIVATE WINDOW Form_1
Return

Procedure QueryTest()
   This.QueryData := Str ( This.QueryRowIndex ) + ',' + Str ( This.QueryColIndex )
Return
```

Visible Property

Specifies whether a GUI object is visible

Syntax:

`Visible <lValue>`

VScrollBar Property

Specifies whether a vertical scrollbar will be included in an Edit or Browse control

Syntax:

`VScrollBar <lValue>`

Sample (Control Definition):

```
DEFINE BROWSE Browse_1
 ROW 50
 COL 100
 WIDTH 120
 HEIGHT 120
 VALUE 1
 WIDTHS {100,100}
 HEADERS {'Header 1',’Header 2’}
 WORKAREA Test
 FIELDS {'Test->Field1',’Test->Field2’}
 FONTNAME "Arial"
 FONTSIZE 9
 TOOLTIP "a browse!"
 VSCROLLBAR .T.
END BROWSE
```

WeekNumbers Property

Specifies whether week numbers are shown for MonthCalendar control

Syntax:

`WeekNumbers <lValue>`

Sample:

```
#include "hmg.ch"
Function Main
   DEFINE WINDOW Form_1 ;
      AT 0,0 ;
      WIDTH 640 HEIGHT 480 ;
      TITLE "Month Calendar Control Demo" ;
      MAIN
      @ 210,10 MONTHCALENDAR Month_2 ;
         VALUE CTOD("01/01/2001") ;
         FONT "Courier" SIZE 12 ;
         TOOLTIP "Month Calendar Control WeekNumbers" ;
         WEEKNUMBERS
   END WINDOW
   ACTIVATE WINDOW Form_1
Return Nil
```

When Property

Codeblock array (one element per column) that is evaluated prior to field editing for Browse control

Syntax:

`When <abValues>`

Sample:

```
@ 10,10 BROWSE Browse_1 ;
 WIDTH 610 ;
 HEIGHT 390 ;
 HEADERS { 'Code' , 'First Name' , 'Last Name' } ;
 WIDTHS { 150 , 150 , 150 } ;
 WORKAREA Test ;
 FIELDS { 'Test->Code' , 'Test->First' , 'Test->Last' } ;
 VALUE 0 ;
 EDIT ;
 WHEN { { || Test->Code == 10 } , , { || Test->Married == .t. } }
```

WholeDropDown Property

Specifies whether a Toolbar button will be used to open a dropdown menu

Syntax:

`WholeDropDown <lValue>`

Sample:

```
DEFINE TOOLBAR ToolBar_a BUTTONSIZE 30,30 FONT 'Arial' SIZE 8 FLAT
   BUTTON Button_1a ;
      CAPTION '&Undo' ;
      PICTURE 'button4.bmp' ;
      ACTION MsgInfo('Undo')
   BUTTON Button_2a ;
      CAPTION '&Save' ;
      PICTURE 'button5.bmp' ;
      WHOLEDROPDOWN
      DEFINE DROPDOWN MENU BUTTON Button_2a
         ITEM 'Exit' ACTION MsgInfo('Exit')
         ITEM 'About' ACTION MsgInfo ("HMG ToolBar Demo")
      END MENU
END TOOLBAR
```

Width Property

Set \/ Gets width of a gui object

Syntax:

```
Width <nValue> (Control Definition)
<ParentWindowName>.<ControlName>.Width := nWidth
<ParentWindowName>.<ControlName>.Width --> nWidth
```

Widths Property

Specifies column widths for Browse and Grid controls. It must be specified as a numeric array containing one element for each column in the control

Syntax:

Widths <anValues>

Sample:

```
@ 10,10 GRID Grid_1 ;
   WIDTH 620 ;
   HEIGHT 330 ;
   HEADERS {'Column 1','Column 2','Column 3'} ;
   WIDTHS {140,140,140} ;
   ITEMS { {‘1’,’2’,’3’} , {‘4’,’5’,’6’} }
```

WindowType Property

Set The Window Type

Syntax:

`WindowType Main | Child | Modal | SplitChild | Standard | Panel`

WorkArea Property

Specifies the workarea for a Browse control

Syntax:

`WorkArea <xcValue>`

Sample:

```
DEFINE BROWSE Browse_1
   COL 10
   ROW 10
   WIDTH 610
   HEIGHT 390
   HEADERS { 'Code' , 'First Name' , 'Last Name' }
   WIDTHS { 150 , 150 , 150 }
   WORKAREA Test
   FIELDS { 'Test->Code' , 'Test->First' , 'Test->Last' }
   VALUE 1
END BROWSE
```

Wrap Property

Specifies whether Spinner control content can be wrapped

Syntax:

`Wrap <lValue>`

Sample:

```
#include "hmg.ch"
Function Main
   DEFINE WINDOW Form_Main ;
      AT 0,0 ;
      WIDTH 640 HEIGHT 480 ;
      TITLE "Main Window" ;
      MAIN
      @ 150,250 SPINNER Spinner_2 ;
         RANGE 0,100 ;
         VALUE 5 ;
         WIDTH 100 ;
         WRAP
   END WINDOW
   CENTER WINDOW Form_Main
   ACTIVATE WINDOW Form_Main
Return Nil
```

Action Event

Occurs When The User Clicks On The Control

Syntax:

```
Action <ActionProcedure>
```

OnChange Event

Occurs When The Content Of a Control Changes

Syntax:

```
OnChange <ActionProcedure>
```

OnClick Event

Occurs when the user clicks on the control

Syntax:

```
OnCLick <ActionProcedure>
```

OnCloseUp Event

Occurs when the user closes the dropdown list of a ComboBox control

Syntax:

```
OnCloseUp <ActionProcedure>
```

OnDblClick Event

Occurs when the user double clicks on the control

Syntax:

```
OnDblClick <ActionProcedure>
```

The following data are available at ‘OnDblClick’ procedure:

- This.CellRowIndex
- This.CellColIndex

- This.CellRow
- This.CellCol
- This.CellWidth
- This.CellHeight
- Note: These properties are not available when OnDblClick procedure is fired by <Enter> key pressing.

OnDisplayChange Event

Occurs when the data in an editable ComboBox control is changed

Syntax:

```
OnDisplayChange <ActionProcedure>
```

OnDropDown Event

Occurs when the user open the dropdown list of a ComboBox control

Syntax:

```
OnDropDown <ActionProcedure>
```

OnEnter Event

Occurs when the user press the <Enter> key in a control with the input focus

Syntax:

```
OnEnter <ActionProcedure>
```

OnGotFocus Event

Occurs when a control or window gains the input focus

Syntax:

```
OnGotFocus <ActionProcedure>
```

OnHeadClick Event

Occurs when a Grid or Browse header is clicked

Syntax:

`OnHeadClick <ActionProcedure>`

OnHScrollBox Event

Occurs when the horizontal scrollbar button of a virtual dimensioned window is dragged

Syntax:

`OnHScrollBox <ActionProcedure>`

Position of scrollbars in a virtual dimensioned window can be retrieved via the ‘value’ property of ‘HscrollBar’ and ‘VscrollBar’ pseudo-controls:

```
WindowName.HscrollBar.value
WindowName.VscrollBar.value
```

OnInit Event

Occurs when a Window is initialized

Syntax:

`OnInit <ActionProcedure>`

OnInteractiveClose Event

Occurs when a Window is about to be closed by pressing the [X] button

Syntax:

`OnInteractiveClose <ActionProcedure>`

If the ‘InteractiveClose’ procedure returns .T., the window will be closed normally, otherwise it will not be closed.

OnKey

Occurs when the user press any key in Grids control with the input focus

Syntax:

```
OnKey <ActionProcedure>
```

You can also make use in your procedure function following HMG functions:

HMG_GetLastVirtualKeyDown() and *HMG_CleanLastVirtualKeyDown()*

Notes:

· Keyboard functions can be found in Advanced\/Read Keyboard and Mouse

· Keys values are defined in **\hmg...\INCLUDE\i_keybd.ch** file, same like as for HotKey definitions.

· **ON KEY** is used as a Grid property at Standard syntax (xBase style: **@... Grid**) and please do not confuse **ON KEY** for Keyboard Shortcuts (HotKeys) definition.

OnLostFocus Event

Occurs when a control or window had lost the input focus

Syntax:

```
OnLostFocus <ActionProcedure>
```

OnMaximize Event

Occurs when a Window is maximized

Syntax:

```
OnMaximize <ActionProcedure>
```

OnMinimize Event

Occurs when a Window is minimized

Syntax:

```
OnMinimize <ActionProcedure>
```

OnMouseClick Event

Occurs when the user click a Window

Syntax:

`OnMouseClick <ActionProcedure>`

The following data are available at ‘OnMouseClick’ procedure:

_HMG_SYSDATA [191] -> Mouse Row

_HMG_SYSDATA [192] -> Mouse Col

OnMouseDrag Event

Occurs when the user drag in a Window

Syntax:

`OnMouseDrag <ActionProcedure>`

The following data are available at ‘OnMouseClick’ procedure:

```
_HMG_SYSDATA [ 191 ] -> Mouse Row
_HMG_SYSDATA [ 192 ] -> Mouse Col
```

OnMouseMove Event

Occurs when the user moves the mouse over a Window

Syntax:

`OnMouseMove <ActionProcedure>`

The following data are available at ‘OnMouseClick’ procedure:

```
_HMG_SYSDATA [ 191 ] -> Mouse Row
_HMG_SYSDATA [ 192 ] -> Mouse Col
```

OnNotifyClick Event

Occurs when a notify icon is clicked

Syntax:

`OnNotifyClick <ActionProcedure>`

OnQueryData Event

Occurs when a virtual Grid control requires data to display a cell

Syntax:

```
OnQueryData <ActionProcedure>
```

The following data are available at ‘OnQueryData’ procedure:

- This.QueryData
- This.QueryRowIndex
- This.QueryColIndex

OnPaint Event

Occurs when a Window is painted by the system

Syntax:

```
OnPaint <ActionProcedure>
```

OnRelease Event

Occurs when a Window is released

Syntax:

```
OnRelease <ActionProcedure>
```

OnSave Event

Occurs when a data-bound Grid control is saved

Syntax:

```
OnSave <ActionProcedure>
```

Properties Available For OnSave Procedure:

- This.AppendBuffer
- This.EditBuffer

- This.MarkBuffer
- This.EditBuffer: Array of one element per edited cell. The elements has the following structure: { nLogicalRow , nLogicalCol , xValue , nRecNo }
- This.AppendBuffer: Array of one element per appended record. The elements has the following structure: { xFieldValue 1 , ... , xFieldValue n }
- This.MarkBuffer: Array of one element per record marked to be deleted or recalled. The elements has the following structure: { nLogicalRow , nRecNo , cMark ('D' or 'R') }

Note: When an action procedure is specified for this eventt, the default one is not executed.

OnScrollDown Event

Occurs when a virtual dimensioned window is scrolled down

Syntax:

`OnScrollDown <ActionProcedure>`

Position of scrollbars in a virtual dimensioned window can be retrieved via the ‘value’ property of ‘HscrollBar’ and ‘VscrollBar’ pseudo-controls:

```
WindowName.HscrollBar.value
WindowName.VscrollBar.value
```

OnScrollLeft Event

Occurs when a virtual dimensioned window is scrolled left

Syntax:

`OnScrollLeft <ActionProcedure>`

Position of scrollbars in a virtual dimensioned window can be retrieved via the ‘value’ property of ‘HscrollBar’ and ‘VscrollBar’ pseudo-controls:

```
WindowName.HscrollBar.value
WindowName.VscrollBar.value
```

OnScrollRIght Event

Occurs when a virtual dimensioned window is scrolled right

Syntax:

`OnScrollRight <ActionProcedure>`

Position of scrollbars in a virtual dimensioned window can be retrieved via the ‘value’ property of ‘HscrollBar’ and ‘VscrollBar’ pseudo-controls:

`WindowName.HscrollBar.value`

`WindowName.VscrollBar.value`

OnScrollUp Event

Occurs when a virtual dimensioned window is scrolled up

Syntax:

`OnScrollUp <ActionProcedure>`

Position of scrollbars in a virtual dimensioned window can be retrieved via the ‘value’ property of ‘HscrollBar’ and ‘VscrollBar’ pseudo-controls:

```
WindowName.HscrollBar.value
WindowName.VscrollBar.value
```

OnSize Event

Occurs when a Window is sized

Syntax:

`OnSize <ActionProcedure>`

OnVScrollBox Event

Occurs when the vertical scrollbar button of a virtual dimensioned window is dragged

Syntax:

`OnVScrollBox <ActionProcedure>`

Position of scrollbars in a virtual dimensioned window can be retrieved via the ‘value’ property of ‘HscrollBar’ and ‘VscrollBar’ pseudo-controls:

```
WindowName.HscrollBar.value
WindowName.VscrollBar.value
```

Activate Method

Activate a Window

Syntax:

```
<WindowName>.Activate
ACTIVATE WINDOW <WindowName1> ... [,<WindowNameN> ]
```

When you specify multiple windows, the maximun initially visible modal window count is 1. If you specify more than one modal windows you must use NOSHOW style to activate as not initially visible.

Append Method

Add a record to a data-bound Grid control.

Syntax:

`<WindowName>.<ControlName>.Append`

AddColumn Method

Add a New Column To a Grid Control

Syntax:

```
<ParentWindowName>.<ControlName>. AddColumn ( [ nColIndex ] , [ cCaption ] , [ nWidth
] , [ nJustify ] )
```

When this command \/ method is used all items in grid (if any) will be lost.

AddControl Method

Adds a control to a Tab control page

Syntax:

```
<WindowName>.<ControlName>.AddControl ( ControlName , nPagenumber , nRow , nCol )
```

AddItem Method

Adds a New Item To a ListBox, COmboBox Or Grid

Syntax:

```
<ParentWindowName>.<ControlName>.AddItem (<cItem> | <acItem> )
```

<Item> type must be character for lists and combos, and array for grids (in this case, the array lenght, must be equal to column count)

AddPage Method

Adds a new page to a Tab control

Syntax:

```
<WindowName>.<ControlName>.AddPage ( nPageNumber , cCaption [ , cImageName ] )
```

Capture Method

Capture a Window As a Bitmap File And Save It

Syntax:

```
 <WindowName>.Capture [ ( cFileName , nRow , nCol , nWidth , nHeight ) ]
```

Center Method

Center a Window

Syntax:

```
<WindowName>.CENTER
CENTER WINDOW <FormName> DESKTOP
CENTER WINDOW <FormName> IN <FormName2>
```

ClearBuffer Method

Empty the record buffer in a Grid control undoing all non-saved changes since the last call to 'Save' method.

Syntax:

```
Buffered <lValue>
```

Close Method

Closes a File In An AnimateBox Or a Player Control

Syntax:

```
 <ParentWindowName>.<ControlName>.Close
```

Delete Method

Marks for deletion selected record in a data-bound Grid control.

Syntax:

`<WindowName>.<ControlName>.Delete`

DeleteColumn Method

Deletes a Column From a Grid Control

Syntax:

```
 <ParentWindowName>.<ControlName>. DeleteColumn ( <nColIndex>)
```

When this command \/ method is used all items in grid (if any) will be lost.

DeleteItem Method

Deletes An Item From a ListBox, ComboBox Or Grid

Syntax:

```
<ParentWindowName>.<ControlName>. DeleteItem ( <nItemNumber>)
```

DeletePage Method

Deletes a page from a Tab control

Syntax:

```
<WindowName>.<Controlname>.DeletePage ( nPageNumber )
```

Eject Method

Ejects The Medium For a Player Control

Syntax:

```
<WindowName>.<ControlName>.Eject
```

Hide Method

Hide a GUI Object

Syntax:

```
<WindowName>.<ControlName>.Hide
<WindowName>.Hide
```

Maximize Method

Maximize a Window

Syntax:

```
<WindowName>.Maximize
```

Minimize Method

Minimize a Window

Syntax:

```
 <WindowName>.Minimize
```

Open Method

Opens a File On An AnimateBox Or Player Control

Syntax:

```
<WindowName>.<ControlName>.Open(cFileName)
```

Pause Method

Pauses Execution On a Player Control

Syntax:

```
<WindowName>.<ControlName>.Pause
```

Play Method

Start Playing on AnimateBox And Player Control*

Syntax:

```
<WindowName>.<ControlName>.Play
```

PlayReverse Method

Start Reverse Playing Content in a Player Control

Syntax:

```
<WindowName>.<ControlName>.PlayReverse
```

Print Method

Print The Content Of a Window

Syntax:

```
 <WindowName>.Print [ ( lPreview , ldialog , nRow , nCol , nWidth , nHeight ) ]
```

Recall Method

Recalls a current selected record in a data-bound Grid control.

Syntax:

```
<WindowName>.<ControlName>.Recall
```

Refresh Method

Refresh Content For a Data-Bound Control

Syntax:

```
<WindowName>.<ControlName>.Refresh
```

Data-bound Grid supports an optional logical parameter. When set to .t., the selected logical records is preserved.

Release Method

Release GUI Objects From Memory

Syntax:

```
<WindowName>.Release
<WindowName>.<ControlName>.Release
```

If ALL \/ MAIN clause is used (or main window is specified) all active windows are released and the program is terminated.

Resume Method

Resumes Playing In a Player Control

Syntax:

```
<WindowName>.<ControlName>.Resume
```

Restore Method

Restore a Window

Syntax:

```
 <WindowName>.Restore
```

Save Method

Saves The Content Of a Data-Bound Control

Syntax:

```
<WindowName>.<ControlName>.Save
```

Seek Method

Change Playing Position In An AnimateBox Control

Syntax:

```
<WindowName>.<ControlName>.Seek
```

SetFocus Method

Gives The Focus a Control Or a Window

Syntax:

```
<ParentWindowName>.<ControlName>. SetFocus
<WindowName>.SetFocus
```

Show Method

Show a GUI Object

Syntax:

```
<WindowName>. [<ControlName>].Show
```

Stop Method

Stop Execution On AnimateBox And Player Controls

Syntax:

```
<WindowName>.<ControlName>.Stop
```

COMPRESS

Creates a Zip File

Syntax:

```
COMPRESS [ FILES ] <afiles>
TO <cZipFile>
BLOCK <bBlock>
[ OVERWRITE ]
```

DECODE

Decodes Database Records Or Files

Syntax:

```
DECODE [FROM <(file)>] ON <key>
FIELDS <fields,...>
[ PASSWORD <password> ]
[ FOR <for> ]
[ WHILE <while> ]
[ ALL ]

DECODE <file1> TO <file2> [ PASSWORD <password>] [DELETE]
DECODE FILE <file> [ PASSWORD <password> ]
```

DISPLAY HELP

Display Windows Help

Syntax:

```
DISPLAY HELP
MAIN | CONTEXT <nTopic> | POPUP <nTopic>
```

DO EVENTS

Forces System Event Processing

Syntax:

`DO EVENTS`

DO REPORT

Creates a Report Based Upon Given Parameters

Syntax:

```
DO REPORT
TITLE <ctitle>
HEADERS <aheaders1> , <aheaders2>
FIELDS <aFields>
WIDTHS <aWidths>
[ TOTALS <aTotals> ]
[ NFORMATS <aformats> ]
WORKAREA <WorkArea>
LPP <nLinesPerPage>
CPL <nCahractersPerLine>
[ LMARGIN <nLeftmargin> ]
[ PAPERSIZE <nPaperSize> ]
[ NOFIXED ]
[ DOSMODE ]
[ PREVIEW ]
[ SELECT ]
[ IMAGE <cgraphic> AT <nfi> , <nci> TO <nff> , <ncf> ]
[ MULTIPLE ]
[ GROUPED BY <cGroupBy> ]
[ HEADRGRP <cHeaderGroup> ]
[ LANDSCAPE ]
```

DO REPORT FORM

Executes a Report From a Given Report Definition File

Syntax:

`DO REPORT FORM <cReportFormName[.rpt>`

DRAW \/ ERASE \/ PRINT GRAPH

Drawing Commands

Syntax:

Bar\/Lines\/Points Chart:

```
DRAW GARPH
   IN WINDOW <WindowName>
   AT <nRow>,<nCol>
   TO <nRow>,<nCol>
   TITLE <cTitle>
   TYPE [ BARS | LINES | POINTS ]
   SERIES <aSeries>
   YVALUES <aYValues>
   DEPTH <nDepth>
   BARWIDTH <nBarWidth>
   HVALUES <nHorizaontalValues>
   SERIENAMES <aSeriesNames>
   COLORS <anColors>
   [ 3DVIEW ]
   [ SHOWGRID ]
   [ SHOWXVALUES ]
   [ SHOWYVALUES ]
   [ SHOWLEGENDS ]
   [ LEGENDSWIDTH <nWidth> ]
```

Pie Chart:

```
 DRAW GRAPH IN WINDOW <window>
    AT <nT>,<nL>
    TO <nB>,<nR>
    TITLE <cTitle>
    TYPE PIE
    SERIES <aSer>
    DEPTH <nD>
    SERIENAMES <aName>
    COLORS <aColor>
    [ 3DVIEW ]
    [ SHOWXVALUES ]
    [ SHOWLEGENDS ]
```

Draw Line:

```
 DRAW LINE IN WINDOW <WindowName> AT <nRow>,<nCol>
    TO <nRow>,<nCol>
    [ PENCOLOR <anPenColor> ]
    [ PENWIDTH <nPenWidth> ]
```

Draw Rectangle:

```
DRAW RECTANGLE IN WINDOW <WindowName> AT <nRow>,<nCol>
   TO <nRow>,<nCol>
   [ PENCOLOR <anPenColor> ]
   [ PENWIDTH <nPenWidth> ]
   [ FILLCOLOR <anFillColor> ]
```

Draw RoundRectangle:

```
DRAW ROUNDRECTANGLE IN WINDOW <WindownNme>
   AT <nRow> , <nCol>
   TO <nRow> , <nCol>
   ROUNDWIDTH <nWidth>
   ROUNDHEIGHT <nHeight>
   [ PENCOLOR <anPenColor> ]
   [ PENWIDTH <nPenWidth> ]
   [ FILLCOLOR <anFillColor> ]
```

Draw Ellipse

```
DRAW ELLIPSE IN WINDOW <WindowName> AT <nRow>,<nCol>
   TO <nRow>,<nCol>
   [ PENCOLOR <anPenColor> ]
   [ PENWIDTH <nPenWidth> ]
   [ FILLCOLOR <anFillColor> ]
```

Draw Arc:

```
DRAW ARC IN WINDOW <WindowName> AT <nRow>,<nCol>
   TO <nRow>,<nCol>
   FROM RADIAL <nRow>,<nCol>
   TO RADIAL <nRow>,<nCol>
   [ PENCOLOR <anPenColor> ]
   [ PENWIDTH <anFillColor> ]
```

Draw Pie:

```
DRAW PIE IN WINDOW <windowname> AT <nRow>,<nCol>
   TO <nRow>,<nCol>
   FROM RADIAL <nRow>, <nCol>
   TO RADIAL <nRow>, <nCol>
   [ PENCOLOR <anPenColor> ]
   [ PENWIDTH <nPenWidth> ]
   [ FILLCOLOR <anFillColor> ]
```

Draw Polygon:

```
DRAW POLYGON IN WINDOW <WindowName> ;
   POINTS <anPoints>
   [ PENCOLOR <anPenColor> ]
   [ PENWIDTH <nPenWidth> ]
   [ FILLCOLOR <anFillColor> ]
```

Draw Polybezier:

```
DRAW POLYBEZIER IN WINDOW <WindowName>
   POINTS <anPoints>
   [ PENCOLOR <anPenColor> ]
   [ PENWIDTH <nPenWidth> ]
```

Erase:

```
ERASE [ IN ] WINDOW <WindowName>
```

Print Graph:

```
PRINT GRAPH [ OF ] <cWindowname> [ PREVIEW ] [ DIALOG ]
```

'Print Graph' command, prints BARS, POINTS, LINES or PIE graph previously drawn using DRAW GRAPH command in the specified window.

Alternatively, BosTaurus library which is exclusively for drawing and image manipulation can also be used.

EDIT

Edits Database Records In The Specified WorkArea

Syntax:

```
EDIT
WORKAREA <workarea>
[ TITLE <cTitle> ]
[ FIELDS <acFields> ]
[ READONLY <alReadOnlyFields> ]
[ SAVE <bSave> ]
[ SEARCH <bSearch> ]
```

```
EDIT EXTENDED
[ WORKAREA <cWorkArea> ]
[ TITLE <cTitle> ]
[ FIELDNAMES <acFieldNames> ]
[ FIELDMESSAGES <acFieldMessages> ]
[ FIELDENABLED <alFieldView> ]
[ TABLEVIEW <alTableView> ]
[ OPTIONS <aOptions> ]
[ ON SAVE <bSave> ]
[ ON FIND <bFind> ]
[ ON PRINT <bPrint> ]
```

ENCODE

Encodes Database Records Or Files

Syntax:

```
ENCODE [ FROM < file > ] ON < key >
FIELDS <fields,...>
[ PASSWORD <password> ]
[ FOR <ForExpression> ]
[ WHILE <WhileExpression> ]
[ ALL ]

ENCODE <file1> TO <file2> [ PASSWORD <password> ] [ DELETE ]
ENCODE FILE <file> [ PASSWORD <password> ]
```

EXECUTE

Opens Or Prints Specified Files

```
EXECUTE
[ OPERATION <operation> ]
[ FILE <file> ]
[ PARAMETERS <parameters> ]
[ DEFAULT <default> ]
[ MAXIMIZE | MINIMIZE | HIDE ]
```

```
EXECUTE
[ FILE <file> ]
[ MAXIMIZE | MINIMIZE | HIDE ]
WAIT
```

The file can be an executable file or a document file.

GRAPH BITMAP

Draw Graph in Bitmap Memory

```
GRAPH BITMAP PIE ;
    SIZE <nWidth>, <nHeight> ;
    SERIEVALUES <aSerieValues> ;
    SERIENAMES <aSerieNames> ;
    SERIECOLORS <aSerieColors> ;
    TITLE <cTitle> ;
    TITLECOLOR <aTitleColor>;
    DEPTH <nDepth> ;
    3DVIEW <l3DView> ;
    SHOWXVALUES <lShowXValues> ;
    SHOWLEGENDS <lShowLegends> ;
    NOBORDER <lNoBorder> ;
    STOREIN <hBitmapVar>
```

```
GRAPH BITMAP BARS|LINES|POINTS ;
   SIZE <nWidth>, <nHeight> ;
   SERIEVALUES <aSerieValues> ;
   SERIENAMES <aSerieNames> ;
   SERIECOLORS <aSerieColors> ;
   SERIEYNAMES <aSerieYNames> ;
   PICTURE <cPicture> ;
   TITLE <cTitle> ;
   TITLECOLOR <aTitleColor> ;
   HVALUES <nHValues> ;
   BARDEPTH <nBarDepth> ;
   BARWIDTH <nBarWidth> ;
   SEPARATION <nSeparation> ;
   LEGENDWIDTH <nLegendWindth> ;
   3DVIEW <l3DView> ;
   SHOWGRID <lShowGrid> ;
   SHOWXGRID <lShowXGrid> ;
   SHOWYGRID <lShowYGrid> ;
   SHOWVALUES <lShowValues> ;
   SHOWXVALUES <lShowXValues> ;
   SHOWYVALUES <lShowYValues> ;
   SHOWLEGENDS <lShowLegends> ;
   NOBORDER <lNoBorder> ;
   STOREIN <hBitmapVar>
```

Note:

- Assign a bitmap to an image control, e.g.: `Form_1.Image_1.HBITMAP := hBitmapVar`

- Save a bitmap to disk, e.g.: `BT_BitmapSaveFile( hBitmapVar, "Graph.PNG", BT_FILEFORMAT_PNG )`
- `See demo \SAMPLES\Controls\Graph\GRAPH_Bitmap`

ON KEY

Defines a Keyboard Shortcut

```
ON KEY <Key>
[ OF <ParentWindow> ]
ACTION <ActionProcedureName> | <bBlock>
```

<Key> must be one of the following:

Basic Keys

```
F1...F12
BACK,TAB,RETURN,ESCAPE,END,HOME,LEFT,UP,RIGHT,DOWN,INSERT,DELETE,PRIOR,NEXT
```

Alt Keys

```
ALT+A...ALT+Z
ALT+0...ALT+9
ALT+F1...ALT+F12
ALT+BACK,ALT+TAB,ALT+RETURN,ALT+ESCAPE,ALT+END,ALT+HOME,ALT+LEFT,ALT+UP,
ALT+RIGHT,ALT+DOWN,ALT+INSERT,ALT+DELETE,ALT+PRIOR,ALT+NEXT
```

Shift Keys

```
SHIFT+A...SHIFT+Z
SHIFT+0...SHIFT+9
SHIFT+F1...SHIFT+F12
SHIFT+BACK,SHIFT+TAB,SHIFT+RETURN,SHIFT+ESCAPE,SHIFT+END,SHIFT+HOME,
SHIFT+LEFT,SHIFT+UP,SHIFT+RIGHT,SHIFT+DOWN,SHIFT+INSERT,SHIFT+DELETE
SHIFT+PRIOR,SHIFT+NEXT
```

Control Keys

```
CONTROL+A...CONTROL+Z
CONTROL+1...CONTROL+9
CONTROL+F1...CONTROL+F12
CONTROL+BACK,CONTROL+TAB,CONTROL+RETURN,CONTROL+ESCAPE,CONTROL+END,
CONTROL+HOME,CONTROL+LEFT,CONTROL+UP,CONTROL+RIGHT,CONTROL+DOWN,
CONTROL+INSERT,CONTROL+DELETE,CONTROL+PRIOR,CONTROL+NEXT
```

PLAY WAVE

Plays a Wave From File Or Resource

```
 PLAY WAVE <cWaveName>
    [ FROM RESOURCE ]
    [ SYNC ]
    [ NOSTOP ]
    [ LOOP ]
    [ NODEFAULT ]
```

RELEASE KEY

Releases a Key Defined Via ON KEY Command

`RELEASE KEY <Key> OF <WindowName>`

Refer to ON KEY command for a list of available keys.

SET AUTOSCROLL

Configure AutoScroll Global Setting

`SET AUTOSCROLL ON | OFF`

When set to ON, virtual dimensioned windows scrollbars, are automatically adjusted to make a non visible control that had gained the focus, visible.

SET BROWSESYNC

Configures BrowseSync Globel Setting

This command affect Browse control behavior.

BROWSE is a compatibility (obsolete) control and therefore not recommended. It is superseded by the GRID control.

SET CODEPAGE

Sets The Current CodePage

`SET CODEPAGE TO UNICODE | ENGLISH | SPANISH | BULGARIAN | GERMAN | GREEK | HUNGARIAN | POLISH | PORTUGUESE | RUSSIAN | SERBIAN | SLOVENIAN`

SET COMMPATH

Sets The Common Path For Inter-Application Communication

`SET COMMPATH TO <cCommonPathName>`

SET EDGE

Set edges in the controls

```
SET CONTROL <ControlName> OF <FormName> CLIENTEDGE
SET CONTROL <ControlName> OF <FormName> STATICEDGE
SET CONTROL <ControlName> OF <FormName> NOTEDGE
```

SET FONT

Sets The Default Font For Controls

```
SET FONT TO <cFontName> , <nFontSize>
```

SET HELPFILE

Sets The Default Help File For a HMG Application

```
SET HELPFILE TO <cHelpFileName>
```

SET INTERACTIVECLOSE

Configure InteractiveClose Global Setting

```
SET INTERACTIVECLOSE ON | OFF | QUERY [MAIN]
```

When sets to OFF the windows can't be closed with ALT+F4 or Clicking [X]. When set to QUERY, the user is asked for confirmation. Using MAIN clause, user will be asked on close only for main program window.

SET LANGUAGE

Selects Language For Interface Messages

```
SET LANGUAGE TO  SPANISH | ENGLISH | FRENCH | PORTUGUESE | GERMAN | RUSSIAN | ITALIAN
| FINNISH | CROATIAN | BASQUE | POLISH | DUTCH | SLOVENIAN | CZECH
```

SET MULTIPLE

Impedes Attempts To Run Multiple Instances Of The Application

```
SET MULTIPLE ON | OFF [ WARNING ]
```

SET NAVIGATION

Configures Navigation Global Setting

```
SET NAVIGATION EXTENDED | STANDARD
```

When set to EXTENDED, pressing ENTER key when focus is in TEXTBOX COMBOBOX or DATEPICKER, acts like TAB key.

SET REGION

Sets The Shape Of a Window Based Upon Given Points Coordinates

```
  SET REGION OF <WindowName> RECTANGULAR <nRow>,<nCol>,<nWidth>,<nHeight>
  SET REGION OF <WindowName> ELLIPTIC <nRow>,<nCol>,<nWidth>,<nHeight>
  SET REGION OF <WindowName> POLYGONAL <aPoints> [ ALTERNATE | WINDING ]
  SET REGION OF <WindowName> RESET
```

SET SCROLL

Set scroll amount for horizontal and vertical scrollbars in virtual dimensioned windows

```
 SET SCROLLSTEP TO <nStep>
 SET SCROLLPAGE TO <nStep>
```

SET STATIONNAME

Sets The Station Name For Inter-Application Communication

`SET STATIONNAME TO <cStationname>`

SET TOOLTIPBACKCOLOR

Set ToolTip Background Color

`SET TOOLTIPBACKCOLOR <aColor>`

SET TOOLTIPFORECOLOR

Set Tooltip Foreground Color

`SET TOOLTIPFORECOLOR <aColor>`

SET TOOLTIPCUSTOMDRAW

Set Form\/Control ToolTip Custom Draw

```
SET TOOLTIPCUSTOMDRAW ON|OFF --> For Default ToolTip Custom Draw is OFF
SET TOOLTIPCUSTOMDRAW TO <lOn>
ToolTipCustomDrawIsActive() --> lBoolean

SET TOOLTIPCUSTOMDRAW FORM <FormName>
   [ BACKCOLOR <aBackColor> ]
   [ FORECOLOR <aForeColor> ]
   [ ARRAYFONT <aFont> ]
   [ BALLOON <lBalloon> ]
   [ TITLE <cTitle> ]
   [ ICON <xIcon> ]
SET TOOLTIPCUSTOMDRAW FORM <FormName> --> Remove tooltip custom draw of the FormName

SET TOOLTIPCUSTOMDRAW CONTROL <ControlName | MenuItemName> OF <ParentName>
   [ BACKCOLOR <aBackColor> ]
   [ FORECOLOR <aForeColor> ]
   [ ARRAYFONT <aFont> ]
   [ BALLOON <lBalloon> ]
   [ TITLE <cTitle> ]
   [ ICON <xIcon> ]

SET TOOLTIPCUSTOMDRAW CONTROL <ControlName | MenuItemName> OF <ParentName> --> Remove
tooltip custom draw of the ControlName/MenuItemName
```

Note:

```
aFont := ARRAY FONT <cFontName> SIZE <nFontSize> [ BOLD ] [ ITALIC ] [ UNDERLINE ] [ S
TRIKEOUT ] --> { cFontName, nFontSize, lBold, lItalic, lUnderline, lStrikeout }
aFont := CREATE ARRAY FONT <cFontName> SIZE <nFontSize> [ BOLD <lBold> ] [ ITALIC <lIt
alic> ] [ UNDERLINE <lUnderline> ] [ STRIKEOUT <lStrikeout> ] --> { cFontName, nFontSi
ze, lBold, lIitalic, lUnderline, lStrikeout }
xIcon := cIconFileName | TOOLTIPICON_NONE | TOOLTIPICON_INFO | TOOLTIPICON_WARNING | T
OOLTIPICON_ERROR | TOOLTIPICON_INFO_LARGE | TOOLTIPICON_WARNING_LARGE | TOOLTIPICON_ER
ROR_LARGE
```

- If the **Title** is not specified the **Icon** is not displayed.
- If the **Title** is specified the **ForeColor** property does not work.
- The **BackColor** property not work.

SET TOOLTIPSTYLE

Set Tooltip Style (Standard Or Balloon)

```
SET TOOLTIPSTYLE STANDARD | BALLOON
```

STORE KEY

Stores Action Block Define Via ON KEY Command To a Variable

```
STORE KEY <Key> OF <WindowName> TO <bVar>
```

SYSTEM OBJECT

Gives Access To Operating System Properties

System object will allow to read and write (when possible) various operating system properties.

Current implementation allows to access (read and write) the system clipboard and various system properties (read only).

Syntax:

```
System.Clipboard
System.Clipboard := <cString>
System.EmptyClipboard
System.DesktopWidth
System.DesktopHeight
 System.DefaultPrinter
 System.DesktopFolder
 System.MyDocumentsFolder
 System.ProgramFilesFolder
 System.SystemFolder
 System.TempFolder
 System.WindowsFolder
```

Samples:

```
 System.Clipboard := 'a string'
 <...>
 MsgInfo(System.Clipboard)
 <...>
 MsgInfo(System.DesktopWidth)
 <...>
 MsgInfo(System.DesktopHeight)
 <...>
 MsgInfo(System.DefaultPrinter)
```

UNCOMPRESS

Uncompress a Zip File

```
UNCOMPRESS [ FILE ] <cZipFile> [ BLOCK <bBlock> ]
```

WAIT WINDOW

Shows a Message Window

```
WAIT WINDOW <cMessage> [ NOWAIT ] | CLEAR
```

This command shows a window with a custom message and pauses program execution until a key is pressed or mouse is clicked.

When optional NOWAIT clause is specified program execution continue after message is displayed. To hide the message window WAIT CLEAR must be used.

This command must be used only after main window was activated.

Sample 1:

```
WAIT WINDOW "Press any key to continue"
```

Sample 2:

```
WAIT WINDOW "Processing" NOWAIT
 <...>
WAIT CLEAR
```

HMG Print System Commands and Functions Reference:

SELECT PRINTER

Selects a Printer

```
SELECT PRINTER <cPrinter> | DEFAULT
 [ TO <lSuccessVar> ]
 [ ORIENTATION <nOrientation> ]
 [ PAPERSIZE <nPaperSize> ]
 [ PAPERLENGTH <nPaperLength> ]
 [ PAPERWIDTH <nPaperWidth> ]
 [ COPIES <nCopies> ]
 [ DEFAULTSOURCE <nDefaultSource> ]
 [ QUALITY <nQuality> ]
 [ COLOR <nColor> ]
 [ DUPLEX <nDuplex> ]
 [ COLLATE <nCollate> ]
 [ PREVIEW ]
 [ NOSAVEBUTTON ]
 [ DIALOGFILENAME <cDialogFileName> ]
 [ SAVEAS <cFullFileName> ]
```

```
SELECT PRINTER DIALOG
 [ TO <lSuccessVar> ]
 [ PREVIEW ]
 [ NOSAVEBUTTON ]
 [ DIALOGFILENAME <cDialogFileName> ]
 [ SAVEAS <cSaveAsFileName> ]
```

Note:

```
cDialogFileName --> [ cPath\ ] cFileName [ +cExt ]
cSaveAsFileName --> [ cPath\ ] cFileName +cExt
cExt --> ".PDF" | ".BMP" | ".JPG" | ".GIF" | ".TIF" |".PNG" | ".EMF"
```

ORIENTATION: Specifies page orientation.

```
PRINTER_ORIENT_PORTRAIT
PRINTER_ORIENT_LANDSCAPE
```

PAPERSIZE: Specifies paper size.

```
PRINTER_PAPER_LETTER Letter, 8 1/2- by 11-inches
PRINTER_PAPER_LEGAL Legal, 8 1/2- by 14-inches
PRINTER_PAPER_A4 A4 Sheet, 210- by 297-millimeters
PRINTER_PAPER_CSHEET C Sheet, 17- by 22-inches
PRINTER_PAPER_DSHEET D Sheet, 22- by 34-inches
PRINTER_PAPER_ESHEET E Sheet, 34- by 44-inches
PRINTER_PAPER_LETTERSMALL Letter Small, 8 1/2- by 11-inches
PRINTER_PAPER_TABLOID Tabloid, 11- by 17-inches
PRINTER_PAPER_LEDGER Ledger, 17- by 11-inches
PRINTER_PAPER_STATEMENT Statement, 5 1/2- by 8 1/2-inches
PRINTER_PAPER_EXECUTIVE Executive, 7 1/4- by 10 1/2-inches
PRINTER_PAPER_A3 A3 sheet, 297- by 420-millimeters
PRINTER_PAPER_A4SMALL A4 small sheet, 210- by 297-millimeters
PRINTER_PAPER_A5 A5 sheet, 148- by 210-millimeters
PRINTER_PAPER_B4 B4 sheet, 250- by 354-millimeters
PRINTER_PAPER_B5 B5 sheet, 182- by 257-millimeter paper
PRINTER_PAPER_FOLIO Folio, 8 1/2- by 13-inch paper
PRINTER_PAPER_QUARTO Quarto, 215- by 275-millimeter paper
PRINTER_PAPER_10X14 10- by 14-inch sheet
PRINTER_PAPER_11X17 11- by 17-inch sheet
PRINTER_PAPER_NOTE Note, 8 1/2- by 11-inches
PRINTER_PAPER_ENV_9 #9 Envelope, 3 7/8- by 8 7/8-inches
PRINTER_PAPER_ENV_10 #10 Envelope, 4 1/8- by 9 1/2-inches
PRINTER_PAPER_ENV_11 #11 Envelope, 4 1/2- by 10 3/8-inches
PRINTER_PAPER_ENV_12 #12 Envelope, 4 3/4- by 11-inches
PRINTER_PAPER_ENV_14 #14 Envelope, 5- by 11 1/2-inches
PRINTER_PAPER_ENV_DL DL Envelope, 110- by 220-millimeters
PRINTER_PAPER_ENV_C5 C5 Envelope, 162- by 229-millimeters
PRINTER_PAPER_ENV_C3 C3 Envelope, 324- by 458-millimeters
PRINTER_PAPER_ENV_C4 C4 Envelope, 229- by 324-millimeters
PRINTER_PAPER_ENV_C6 C6 Envelope, 114- by 162-millimeters
PRINTER_PAPER_ENV_C65 C65 Envelope, 114- by 229-millimeters
PRINTER_PAPER_ENV_B4 B4 Envelope, 250- by 353-millimeters
PRINTER_PAPER_ENV_B5 B5 Envelope, 176- by 250-millimeters
PRINTER_PAPER_ENV_B6 B6 Envelope, 176- by 125-millimeters
PRINTER_PAPER_ENV_ITALY Italy Envelope, 110- by 230-millimeters
PRINTER_PAPER_ENV_MONARCH Monarch Envelope, 3 7/8- by 7 1/2-inches
PRINTER_PAPER_ENV_PERSONAL 6 3/4 Envelope, 3 5/8- by 6 1/2-inches
PRINTER_PAPER_FANFOLD_US US Std Fanfold, 14 7/8- by 11-inches
PRINTER_PAPER_FANFOLD_STD_GERMAN German Std Fanfold, 8 1/2- by 12-inches
PRINTER_PAPER_FANFOLD_LGL_GERMAN German Legal Fanfold, 8 1/2- by 13-inches
PRINTER_PAPER_USER User defined size paper
```

PAPERLENGTH \/ PAPERWIDTH: Sets a custom paper length\/width (millimeters).

To use custom paper size, you mus specify PRINTER_PAPER_USER as paper size.

COPIES: Set the number of copies to print.

This setting is ignored unless the printer driver indicates support for multiple-page copies.

DEFAULTSOURCE: Specifies the paper source.

```
PRINTER_BIN_ONLYONE
PRINTER_BIN_LOWER
PRINTER_BIN_MIDDLE
PRINTER_BIN_MANUAL
PRINTER_BIN_ENVELOPE
PRINTER_BIN_ENVMANUAL
PRINTER_BIN_AUTO
PRINTER_BIN_TRACTOR
PRINTER_BIN_SMALLFMT
PRINTER_BIN_LARGEFMT
PRINTER_BIN_LARGECAPACITY
PRINTER_BIN_CASSETTE
PRINTER_BIN_FORMSOURCE
```

QUALITY: Specifies the printer resolution.

```
PRINTER_RES_HIGH
PRINTER_RES_MEDIUM
PRINTER_RES_LOW
PRINTER_RES_DRAFT
```

COLOR: Switches between color and monochrome on color printers.

```
PRINTER_COLOR_COLOR
PRINTER_COLOR_MONOCHROME
```

DUPLEX: Selects duplex or double-sided printing.

```
PRINTER_DUP_SIMPLEX
PRINTER_DUP_HORIZONTAL
PRINTER_DUP_VERTICAL
```

This setting is ignored unless the printer driver indicates support for duplex printing.

COLLATE: Specifies whether collation should be used.

```
PRINTER_COLLATE_TRUE
PRINTER_COLLATE_FALSE
```

This setting is ignored unless the printer driver indicates support for collation.

PREVIEW: When specified, a print preview window is shown at print job end.

When selected printer driver copies setting is greater than 1, or collate is enabled, you'll not be able to change "copies" and "collate" values from preview's print dialog.

Active shortcuts during preview:

HOME: First Page.

PRIOR: Previous Page.

NEXT: Next page.

END: Last Page.

CONTROL+P: Open Print Dialog.

CONTROL+G: Open Go To Page Dialog.

MULTIPLY: Zoom In \/ Out.

ESCAPE \/ CONTROL+C ALT+F4: Close Preview.

CONTROL+S: Save Pages As EMF, BMP, JPG, GIF, TIFF or PNG Files.

CONTROL+T: Show \/ Hide Thumbnails Window.

Alternatively, you can click a thumnbail to change the active page.

New Functions:

```
OpenPrinterGetDC() --> hDC of the current Open Printer
OpenPrinterGetPageDC() --> hDC of the current Page being printed
IsPrintPageMetaFile() --> lBoolean
OpenPrinterGetPageWidth() --> Width (in millimeters) of the current Page being printed
OpenPrinterGetPageHeight() --> Height (in millimeters) of the current Page being print
ed
```

START PRINTDOC

Starts a Print Job

```
START PRINTDOC [ NAME <cPrintJobName> ] [ STOREJOBDATA <aJobData> ]
```

- *aJobData* memvar must be declared Public or Private,

aJobData must be passed as a parameter to the function:

```
HMG_PrintGetJobInfo ( aJobData )  --> {} or { nJobID, cPrinterName, cMachineName, cUse
rName,  cDocument, cDataType, cStatus, nStatus, nPriorityLevel, nPositionPrintQueue, n
TotalPages, nPagesPrinted, cLocalDate, cLocalTime }
```

cStatus --> This member should be checked prior to nStatus and, if cStatus is empty, the status is defined by the contents of the nStatus member.

nStatus --> The value of this member can be zero or a combination of one or more of the following values.

A value of zero indicates that the print queue was paused after the document finished spooling.

```
 JOB_STATUS_PAUSED
 JOB_STATUS_ERROR
 JOB_STATUS_DELETING
 JOB_STATUS_SPOOLING
 JOB_STATUS_PRINTING
 JOB_STATUS_OFFLINE
 JOB_STATUS_PAPEROUT
 JOB_STATUS_PRINTED
 JOB_STATUS_DELETED
 JOB_STATUS_BLOCKED_DEVQ
 JOB_STATUS_USER_INTERVENTION
 JOB_STATUS_RESTART
 JOB_STATUS_COMPLETE
```

END PRINTDOC

Ends a Print Job

`END PRINTDOC`

ABORT PRINTDOC

Stops The Current Print Job

This command erases everything drawn since the last call to the START PRINTDOC command.

```
ABORT PRINTDOC
```

START PRINTPAGE

Prepares The Printer Driver To Accept Data and initializes a new page.

```
START PRINTPAGE
```

END PRINTPAGE

Tells The Device That The Application Finished Writing To a Page

```
END PRINTPAGE
```

PRINT DATA

Prints Data

```
@ <Row> , <Col> PRINT [ DATA ] <xData>
 [ TO <nToRow> , <nToCol> ]
 [ FONT <cFontName> ]
 [ SIZE <nFontSize> ]
 [ BOLD ]
 [ ITALIC ]
 [ UNDERLINE ]
 [ STRIKEOUT ]
 [ COLOR <aColor> ]
 [ RIGHT | CENTER ]
 [ ANGLE <nAngleInDegrees> ]
```

- <xData> type can be character, numeric, date, logical or memo.
- Logical data is shown as 'Yes' or 'No' (translated according selected language).
- If <xData> is multi-line text (character or memo types), you must specify TO <nToRow> , <nToCol> optional parameters.
- When <xData> type is numeric, digits are printed begining exactly at specified position (without leading spaces).
- If optional **RIGHT** clause is used <Row> , <Col> indicates the right coordinates of <xData> (unless TO <nToRow> , <nToCol> is specified.
- If optional **CENTER** clause is used <Row> , <Col> indicates the center coordinates of <xData> (unless TO <nToRow> , <nToCol> is specified.

Units: <Row> , <Col> units are **millimeters**.

PRINT IMAGE

Prints an Image From File Or Resource

Print images in formats: BMP, GIF, JPG, TIF, WMF, EMF, CUR and PNG.

If **<STRETCH>** is not specified, the image will be automatically scaled to fit the specified rectangle.

```
@ <nRow> , <nCol> PRINT IMAGE <cImageFileName> | <cImageResourcename>
 WIDTH <nWidth>
 HEIGHT <nHeight>
 [ STRETCH ]
 [ TRANSPARENT ]
 [ TRANSPARENTCOLOR anTransparentColor ]
```

Units: <Row> , <Col> , <nWidth> , <nHeight> units are **millimeters**.

PRINT LINE

Prints a Line

```
@ <nRow> , <nCol> PRINT LINE
 TO <nToRow> , <nToCol>
 [ PENWIDTH <nPenWidth> ]
 [ COLOR <aColor> ]
```

Units: <Row> , <Col> , <nToRow> , <nToCol> , <nPenWidth> units are **millimeters**.

PRINT RECTANGLE

Prints a Rectangle

If ROUNDED clause is used, the rectangle will have rounded corners.

```
@ <nRow> , <nCol> PRINT RECTANGLE
 TO <nToRow> , <nToCol>
 [ PENWIDTH> <nPenWidth> ]
 [ COLOR <aColor> ]
 [ ROUNDED ]
 [ FILLED ]
```

Units: <Row> , <Col> , <nToRow> , <nToCol> , <nPenWidth> units are **millimeters**.

GetPrintableAreaWidth()

Returns The Current Printable Area Width in Milimeters

```
GetPrintableAreaWidth() -> nPrintableAreaWidth
```

GetPrintableAreaHeight()

Returns The Printable Area Height In Milimeters

```
GetPrintableAreaHeight() -> nPrintableAreaHeight
```

GetPrintableAreaHorizontalOffset()

Returns The Current Printable Area Horizontal Offset

The horizontal offset is the distance from paper edge to begining of printable area (it is returned in milimeters).

```
GetPrintableAreaHorizontalOffset() -> PrintableAreaHorizontalOffset
```

GetPrintableAreaVerticalOffset()

Returns The Current Printable Area Vertical Offset

The printable area vertical offset is the distance from paper edge to begining of printable area (it is returned in milimeters).

```
GetPrintableAreaVerticalOffset() -> nPrintableAreaVerticalOffset()
```

GetPrinter()

Returns The Selected Printer Name From a Selection Window

If [Cancel] button is pressed, an empty string is returned.

```
GetPrinter() --> cPrinterName
```

---

aPrinters()

Returns An Array With All Available Printers

```
aPrinters() --> aPrinterNames
```

GetDefaultPrinter()

Returns The Default Printer Name

`GetDefaultPrinter() --> cDefaultPrinterName`

HMG_PrinterGetStatus ()

`HMG_PrinterGetStatus ( [ cPrinterName ] ) --> nStatusPrinter`

nStatusPrinter --> The value of this member can be zero or a combination of one or more of the following values:

```
PRINTER_STATUS_OK
PRINTER_STATUS_PAUSED
PRINTER_STATUS_ERROR
PRINTER_STATUS_PENDING_DELETION
PRINTER_STATUS_PAPER_JAM
PRINTER_STATUS_PAPER_OUT
PRINTER_STATUS_MANUAL_FEED
PRINTER_STATUS_PAPER_PROBLEM
PRINTER_STATUS_OFFLINE
PRINTER_STATUS_IO_ACTIVE
PRINTER_STATUS_BUSY
PRINTER_STATUS_PRINTING
PRINTER_STATUS_OUTPUT_BIN_FULL
PRINTER_STATUS_NOT_AVAILABLE
PRINTER_STATUS_WAITING
PRINTER_STATUS_PROCESSING
PRINTER_STATUS_INITIALIZING
PRINTER_STATUS_WARMING_UP
PRINTER_STATUS_TONER_LOW
PRINTER_STATUS_NO_TONER
PRINTER_STATUS_PAGE_PUNT
PRINTER_STATUS_USER_INTERVENTION
PRINTER_STATUS_OUT_OF_MEMORY
PRINTER_STATUS_DOOR_OPEN
PRINTER_STATUS_SERVER_UNKNOWN
PRINTER_STATUS_POWER_SAVE
```

Bos Taurus

Graphic Library for HMG

Version 1.0.6

© Dr. Claudio Soto

srvet@adinet.com.uy

http:\/\/srvet.blogspot.com

Uruguay, December 2012

What is the graphics library Bos Taurus?

Bos Taurus is a set of graphics functions (free open code) written in C and Harbour based in the Windows GDI functions specially developed for easy programming of the **ON PAINT** events in **HMG**.

In the past, the ON PAINT event was little used in applications developed in HMG due to malfunction of this event, but recently since version 3.0.43 of **HMG Official** (2012\/08\/30) and version 2.1.5 of **HMG Extended** (2012\/09\/12) the bug was corrected.

The ON PAINT event runs in response to WM_PAINT message that the Windows System sent to a window for inform that have to paint the client area. In HMG the ON PAINT event allows paint directly on the client area of a window before paint the Controls (Button, Grid, Image, ComboBox, etc.) and before paint the Draw Commands (Draw Graph, Rectangle, Line, etc.).

The **BosTaurus_ChangeLog.TXT** contains the latest updates in the source files of the library. From version 1.0.3, **Bos Taurus** supports the **ANSI and UNICODE** character set.

Why use the graphics library Bos Taurus?

Because in HMG exist very few native functions to draw and manipulate graphic images.

In **Bos Taurus** the draw functions are called with a **Handle Device Context (hDC)** obtained from the selected output device (desktop, client area, bitmap, etc.) just like when programming in C for Windows. This allows the use of the same functions of drawing and the easy transfer of graphics from one device to another irrespective of the output device selected. For example if a drawing function we pass to a hDC of window, the function draw in the client area of the window, however, if we pass the hDC associated with a Bitmap, the function draw in the Bitmap image.

The system of use of **hDC** in the functions will help other people to create other drawing functions fully compatible with the graphics library **Bos Taurus**, the programmer only have to use the appropriate hDC that gives the graphics library **Bos Taurus**. In fact with this system can be implemented in Harbour virtually all graphics functions (GDI) of Windows thereby providing a great power of graphic manipulation to the programmer.

How to use the graphics library Bos Taurus?

It is very easy to use the graphics library **Bos Taurus** in HMG because it is part of the library of **HMG Official**

Prototype Example

This example is very simple. Loads a Bitmap to the start the application and puts the background image, before drawing the image paints a vertical colors gradient from white to black. When you close the application frees of the memory the handle of the Bitmap. **For more information see the demos that accompanying the Bos Taurus**.

```
#include "HMG.CH"
FUNCTION MAIN
   PRIVATE hBitmap := 0
   DEFINE WINDOW Win1;
      AT 0,0;
      WIDTH 700;
      HEIGHT 600;
      TITLE "Prototype Demo";
      MAIN;
      ON INIT Proc_ON_INIT ();
      ON RELEASE Proc_ON_RELEASE ();
      ON PAINT Proc_ON_PAINT ()
      // Definition of Application Controls
      @ 435, 280 BUTTON Button1 CAPTION "Click";
         ACTION MsgInfo ("Hello")
   END WINDOW
   CENTER WINDOW Win1
   ACTIVATE WINDOW Win1
RETURN

PROCEDURE Proc_ON_INIT
   hBitmap := BT_BitmapLoadFile ("HMG.bmp")
RETURN

PROCEDURE Proc_ON_RELEASE
   BT_BitmapRelease (hBitmap)
RETURN

PROCEDURE Proc_ON_PAINT
   LOCAL hDC, BTstruct
   hDC := BT_CreateDC ("Win1", BT_HDC_INVALIDCLIENTAREA, @BTstruct)
   BT_DrawGradientFillVertical (hDC,;
      0, 0,;
      BT_ClientAreaWidth ("Win1"),;
      BT_ClientAreaHeight("Win1"),;
      WHITE, BLACK)
   BT_DrawBitmap (hDC, 35, 200, 300, 250, BT_COPY, hBitmap)
   BT_DeleteDC (BTstruct)
RETURN
```

Information about the Library:

BT_InfoName ()

Return a string with the name of the library: "Bos Taurus"

Syntax:

`MsgInfo( BT_InfoName () )`

BT_InfoVersion ()

Return a string with the current version of the library, for example: "1.0.0"

Syntax:

```
MsgInfo( BT_InfoVersion() )
```

BT_InfoAuthor ()

Return a string with the name of the author: "© Dr. Claudio Soto (from Uruguay)"

Syntax:

`MsgInfo( BT_InfoAuthor() )`

Information about the Environment:

BT_GetDesktopHandle ()

Return a handle to the desktop (hWin). The desktop window covers the entire screen.

Syntax:

```
hWin := BT_GetDesktopHandle ()
```

BT_DesktopWidth ()

Return the width of the screen in pixels.

Syntax:

```
nWidth :=  BT_DesktopWidth ()
```

BT_DesktopHeight ()

Return the height of the screen in pixels.

Syntax:

```
nHeight := BT_DesktopHeight()
```

BT_WindowWidth (Win)

Return the width of the specified window in pixels.

Win: is the name (e.g. "Win1") or is the handle (e.g. hWin) of the window.

Syntax:

```
nWidth := BT_WindowWidth( 'form1' )
nWidth := BT_WindowWidth( hWin )
```

BT_WindowHeight ***(Win)***

Return the height of the specified window in pixels.

Win: is the name (e.g. "Win1") or is the handle (e.g. hWin) of the window.

BT_ClientAreaWidth(Win)

Return the width of the client area of the specified window in pixels. The client area includes the area of the Status Bar.

Win: is the name (e.g. "Win1") or is the handle (e.g. hWin) of the window.

BT_ClientAreaHeight (Win)

Return the height of the client area of the specified window in pixels. The client area includes the area of the Status Bar.

Win: is the name (e.g. "Win1") or is the handle (e.g. hWin) of the window.

BT_StatusBarWidth(cFormName)

Return the width of the Status Bar of the specified window in pixels. If not defined (not exist) the Status Bar, returns zero.

cFormName: is the name of the window(e.g. "Win1").

BT_StatusBarHeight (cFormName)

Return the height of the Status Bar of the specified window in pixels. If not defined (not exist) the Status Bar, returns zero.

cFormName: is the name of the window(e.g. "Win1").

Client Area Functions:

BT_ClientAreaInvalidateAll *(Win, lErase)*

Force to redraw the all of the client area of the specified window.

Win: is the name (e.g. "Win1") or is the handle (e.g. hWin) of the window.

lErase: specifies whether the background within the update region is to be erased when the update region is processed. If this parameter is TRUE (.T.), the background is erased. If this parameter is FALSE (.F.), the background remains unchanged. For default: *lErase* = .F.

BT_ClientAreaInvalidateRect *(Win, Row, Col, Width, Height, lErase)*

Force to redraw the specified rectangle of the client area of the specified window.

Win: is the name (e.g. "Win1") or is the handle (e.g. hWin) of the window.

Row, Col, Width, Height: specifies the dimensions in pixels of the rectangle to redraw. For default: *Row* = 0, *Col* = 0, *Width* = BT_ClientAreaWidth(Win), *Height* = BT_ClientAreaHeight(Win).

lErase: specifies whether the background within the update region is to be erased when the update region is processed. If this parameter is TRUE (.T.), the background is erased. If this parameter is FALSE (.F.), the background remains unchanged. For default: *lErase* = .F.

BT_BitmapLoadFile(cFileName)

Loads an image (BMP, JPG, GIF, TIF or PNG) from the disk or resources and returns a handle to bitmap format image (*hBitmap*).

cFileName: is the name of the file or of the resource that contains the image.

BT_BitmapLoadEMF(cFileName, aRGBcolor_Fill_Bk, NewWidht, NewHeight, Mode_Stretch)

Loads an EMF (Enhanced Meta File) image from the disk or resources and returns a handle to bitmap format image (*hBitmap*).

cFileName: is the name of the file or of the resource that contains the image.

aRGBcolor_Fill_Bk: array containing the RGB colors that paint the background of the bitmap. For default: *aRGBcolor_Fill_Bk* = {0,0,0} = BLACK.

New_Width, New_Height: is the new size of the bitmap image. For default this values are the original width and height of the EMF image.

Mode_Stretch: sets the mode as the image of origin is adjusts (is stretches or compresses) in the new size bitmap, it is one of the constants: BT_SCALE or BT_STRETCH (defined in BosTaurus.CH). For default *Mode_Stretch* = BT_STRETCH.

BT_BitmapSaveFile(hBitmap, cFileName, nTypePicture)

Save an image (BMP, JPG, GIF, TIF or PNG) in the disk.

hBitmap: is a handle to the bitmap image.

cFileName: is the name of the file to save.

nTypePicture: specifies the format in which you want to save the image, it is one of the constants: BT_FILEFORMAT_BMP, BT_FILEFORMAT_JPG, BT_FILEFORMAT_GIF, BT_FILEFORMAT_TIF or BT_FILEFORMAT_PNG (defined in BosTaurus.CH). For default *nTypePicture* = BT_FILEFORMAT_BMP.

BT_BitmapRelease (hBitmap)

Release a bitmap of the memory.

hBitmap: is a handle to bitmap.

Note: all of the handles of bitmap obtained from load, create, clone, copy, etc. a bitmap image is responsibility of the application for his release before close.

BT_BitmapWidth(hBitmap)

Return the width of the specified bitmap in pixels.

hBitmap: is a handle to bitmap.

BT_BitmapHeight (hBitmap)

Return the height of the specified bitmap in pixels.

hBitmap: is a handle to bitmap.

BT_BitmapBitsPerPixel (hBitmap)

Return the bits per pixel of the specified bitmap.

hBitmap: is a handle to bitmap.

BT_BitmapCreateNew(Width, Height, aRGBcolor_Fill_Bk)

Creates a bitmap in the memory and returns a handle to bitmap (*hBitmap*).

Width, Height: are the dimensions of the bitmap in pixels.

aRGBcolor_Fill_Bk: array containing the RGB colors that paint the background of the bitmap. For default: *aRGBcolor_Fill_Bk* = {0,0,0} = BLACK.

BT_BitmapClone(hBitmap, Row, Col, Width, Height)

Clones the specified bitmap and returns a handle to the cloned bitmap.

hBitmap: is a handle to the origin bitmap.

Row, Col, Width, Height: specifies the dimensions in pixels of the rectangle to clone. For default: *Row* = 0, *Col* = 0, *Width* = BT_BitmapWidth(hBitmap), *Height* = BT_BitmapHeight(hBitmap).

BT_BitmapCopyAndResize(hBitmap, New_Width, New_Height, Mode_Stretch, Algorithm)

Copy and resizes the specified bitmap and returns a handle to the bitmap with the new size.

hBitmap: is a handle to the origin bitmap.

New_Width, New_Height: is the new size of the bitmap image.

Mode_Stretch: sets the mode as the bitmap of origin is adjusts (is stretches or compresses) in the new size bitmap, it is one of the constants: BT_SCALE or BT_STRETCH (defined in BosTaurus.CH). For default *Mode_Stretch* = BT_STRETCH.

Algorithm: sets the mode as the bitmap is resized, it is one of the constants: BT_RESIZE_COLORONCOLOR, BT_RESIZE_HALFTONE or BT_RESIZE_BILINEAR (defined in BosTaurus.CH). For default *Algorithm* = BT_RESIZE_HALFTONE.

BT_BitmapPaste(hBitmap_D, Row_D, Col_D, Width_D, Height_D, Mode_Stretch, hBitmap_O)

Paste a bitmap (origin) into another bitmap (destination).

hBitmap_D: is a handle to the destination bitmap.

Row_D, Col_D, Width_D, Height_D: specifies the size of the rectangle in pixels in the destination bitmap where you will paste the origin bitmap. For default: *Row_D* = 0, *Col_D* = 0, *Width* = BT_BitmapWidth(hBitmap_D), *Height* = BT_BitmapHeight(hBitmap_D).

Mode_Stretch: sets the mode as the bitmap of origin is adjusts (is stretches or compresses) in the specified rectangle in the destination bitmap, it is one of the constants: BT_SCALE, BT_STRETCH or BT_COPY (defined in BosTaurus.CH).

hBitmap_O: is a handle to the origin bitmap.

BT_BitmapPasteTransparent(hBitmap_D, Row_D, Col_D, Width_D, Height_D, Mode_Stretch, hBitmap_O, aRGBcolor_transp)

Paste a transparent bitmap (origin) into another bitmap (destination).

hBitmap_D: is a handle to the destination bitmap.

Row_D, Col_D, Width_D, Height_D: specifies the size of the rectangle in pixels in the destination bitmap where you will paste the origin bitmap. For default: *Row_D* = 0, *Col_D* = 0, *Width* = BT_BitmapWidth(hBitmap_D), *Height* = BT_BitmapHeight(hBitmap_D).

Mode_Stretch: sets the mode as the bitmap of origin is adjusts (is stretches or compresses) in the specified rectangle in the destination bitmap, it is one of the constants: BT_SCALE, BT_STRETCH or BT_COPY (defined in BosTaurus.CH).

hBitmap_O: is a handle to the origin bitmap.

aRGBcolor_transp: array containing the RGB colors to treat as transparent in the origin bitmap. For default: *aRGBcolor_transp* = color of the first pixel of the origin bitmap.

BT_BitmapPasteAlphaBlend(hBitmap_D, Row_D, Col_D, Width_D, Height_D, Alpha, Mode_Stretch, hBitmap_O)

Paste a bitmap (origin) with Alpha Blend effect (that make the pixels to be transparent or semi transparent) into another bitmap (destination).

hBitmap_D: is a handle to the destination bitmap.

Row_D, Col_D, Width_D, Height_D: specifies the size of the rectangle in pixels in the destination bitmap where you will paste the origin bitmap. For default: *Row_D* = 0, *Col_D* = 0, *Width* = BT_BitmapWidth(hBitmap_D), *Height* = BT_BitmapHeight(hBitmap_D).

Alpha: is alpha blending value to be applied to the entire source bitmap (range 0 to 255, transparent = 0, opaque = 255).

Mode_Stretch: sets the mode as the bitmap of origin is adjusts (is stretches or compresses) in the specified rectangle in the destination bitmap, it is one of the constants: BT_SCALE, BT_STRETCH or BT_COPY (defined in BosTaurus.CH).

hBitmap_O: is a handle to the origin bitmap.

BT_BitmapInvert(hBitmap)

Invert the colors of an image (makes a negative of the original image).

hBitmap: is a handle to bitmap.

BT_BitmapGrayness(hBitmap, Gray_Level)

Set the grayness of the specified bitmap.

hBitmap: is a handle to bitmap.

Gray_Level: is the gray level in percentage (range 0 to 100%, none = 0, full = 100).

BT_BitmapBrightness(hBitmap, Light_Level)

Change the brightness of the specified bitmap.

hBitmap: is a handle to bitmap.

Light_Level is the light level (range -255 to +255, black = -255, white = +255, none = 0).

BT_BitmapContrast(hBitmap, ContrastAngle)

Increase the contrast of the colors in an image.

hBitmap: is a handle to bitmap.

ContrastAngle: is the angle of the slope (in radians) of the contrast transform.

BT_BitmapModifyColor(hBitmap, RedLevel, GreenLevel, BlueLevel)

Increases or decreases the colors channels of an image.

hBitmap: is a handle to bitmap.

RedLevel: is the Red color level to change (range -255 to +255, none = 0).

GreenLevel: is the Green color level to change (range -255 to +255, none = 0).

BlueLevel: is the Blue color level to change (range -255 to +255, none = 0).

BT_BitmapGammaCorrect(hBitmap, RedGamma, GreenGamma, BlueGamma)

Image displaying suffers from photometric distortions caused by the nonlinear response of display devices to lightness. The photometric response of a displaying device is known as the gamma response characteristic. Display monitors for different operating systems use different gammas. To compensate for these differences, the gamma of the image needs to be corrected. A gamma correction with gamma equal to 1.0 is an identity color transformation. Gamma less than 1.0 makes a picture look darker, and gamma larger than 1.0 makes a picture look lighter.

hBitmap: is a handle to bitmap.

RedGamma: is the Gamma Red color level to change (normally range between 0.2 and 5.0, none = 1.0).

GreenGamma: is the Gamma Green color level to change (normally range between 0.2 and 5.0, none = 1.0).

BlueGamma: is the Gamma Blue color level to change (normally range between 0.2 and 5.0, none = 1.0).

BT_BitmapConvolutionFilter3x3(hBitmap, aFilter)

The convolution is a matrix (or two-dimensional arrangement) applied to an image. The elements of this array are integers values. The result of this operation is a new image that has been filtered. The convolution basically modifies the color of a pixel based on the color of the neighboring pixels. For each color channel, the color value for each pixel is calculated on the original color and the color of the surrounding pixels.

hBitmap: is a handle to bitmap.

aFilter: is an array with the following values:

{ k1, k2, k3,

k4, k5, k6,

k7, k8, k9,

Divisor, Bias }

Where **k1 ... k9** are the values of the 3x3 matrix that multiply the pixels. The center cell value (k5) multiplies the pixel that is currently processing. **Divisor** determines the brightness of the final image and **Bias** increments (positive value) or decrements (negative value) the end color of the currently processed pixel. For more details see the demo10.

RULES OF THUMB TO CREATE USER DEFINED FILTERS

	Center Cell Value (k5)	Surrounding Cell Values (k1, k2, k3, k4, k6, k7, k8, k9)
Blur	POSITIVE	Symmetrical pattern of POSITIVE values
Sharpen	POSITIVE	Symmetrical pattern of NEGATIVE values
Edges	NEGATIVE	Symmetrical pattern of POSITIVE values
Emboss	POSITIVE	Symmetrical pattern of NEGATIVE values

Center Cell Value = k5

Surrounding Cell Values = k1, k2, k3, k4, k6, k7, k8, k9

Sum = (Center Cell Value) + (Surrounding Cell Values)

Sum = k1 + k2 + k3 + k4 + k5 + k6 + k7 + k8 + k9

If Sum \/ Divisor = 1 ---> Retain the brightness of the original image.

If Sum \/ Divisor > 1 ---> Increases the bright of the image

If Sum \/ Divisor < 1 ---> Darken the image

To reduce the effect of a filter (Blur, Sharpen, Edges, etc.) you must increase the value of the center cell (k5)

BT_BitmapTransform (hBitmap, Mode, Angle, aRGBColor_Fill_Bk)

Reflects and\/or rotates of the specified bitmap and returns a handle to the new transformed bitmap.

hBitmap: is a handle to the origin bitmap.

Mode *_is one or a combination (sum) of the constants BT_BITMAP_REFLECT_HORIZONTAL, BT_BITMAP_REFLECT_VERTICAL and BT_BITMAP_ROTATE* _(defined in BosTaurus.CH).

Angle: is the angle at which it is to rotate the image (range 0 to 360º). This parameter is considered only if BT_BITMAP_ROTATE is set in *Mode*, otherwise it is ignored.

aRGBColor_Fill_Bk: array containing the RGB colors to fill the empty spaces of the background of the image. This parameter is considered only if BT_BITMAP_ROTATE is set in *Mode*, otherwise it is ignored. For default: *aRGBcolor_Fill_Bk* = color of the first pixel of the origin bitmap.

BT_BitmapCaptureDesktop(Row, Col, Width, Height)

Captures the desktop and returns a handle to the image bitmap captured (hBitmap).

Row, Col, Width, Height: specifies the dimensions in pixels of the rectangle to capture. For default: *Row* = 0, *Col* = 0, *Width* = BT_DesktopWidth(), *Height* = BT_DesktopHeight().

BT_BitmapCaptureWindow(Win, Row, Col, Width, Height)

Captures a specified window and returns a handle to the image bitmap captured (hBitmap).

Win: is the name (e.g. "Win1") or is the handle (e.g. hWin) of the window.

Row, Col, Width, Height: specifies the dimensions in pixels of the rectangle to capture. For default: *Row* = 0, *Col* = 0, *Width* = BT_WindowWidth(Win), *Height* = BT_WindowHeight(Win).

BT_BitmapCaptureClientArea(Win, Row, Col, Width, Height)

Captures the client area of the specified window and returns a handle to the image bitmap captured (hBitmap).

Win: is the name (e.g. "Win1") or is the handle (e.g. hWin) of the window.

Row, Col, Width, Height: specifies the dimensions in pixels of the rectangle to capture. For default: *Row* = 0, *Col* = 0, *Width* = BT_ClientAreaWidth(Win), *Height* = BT_ClientAreaHeight(Win).

BT_BitmapClipboardGet(Win)

Gets a bitmap image from the clipboard and returns a handle to the bitmap (hBitmap). If the function fails or the clipboard is empty returns zero.

Win: is the name (e.g. "Win1") or is the handle (e.g. hWin) of the window associated to opening of the clipboard.

BT_BitmapClipboardPut(Win, hBitmap)

Put a bitmap image in the clipboard. If the function fails returns FALSE (.F.), otherwise returns TRUE (.T.).

Win: is the name (e.g. "Win1") or is the handle (e.g. hWin) of the window associated to opening of the clipboard.

hBitmap: is a handle to bitmap.

BT_BitmapClipboardClean(Win)

Empties the clipboard and frees all the handles of data in the clipboard (bitmap, text, metafiles, etc.). If the function fails or the clipboard is empty returns FALSE (.F.), otherwise returns TRUE (.T.).

Win: is the name (e.g. "Win1") or is the handle (e.g. hWin) of the window associated to opening of the clipboard.

BT_BitmapClipboardIsEmpty ()

Returns TRUE (.T.) if exist a bitmap image in the clipboard, otherwise returns FALSE (.F.).

Functions that Get\/Release Handle Device Context

BT_CreateDC(Win_or_hBitmap, Type, @BTstruct)

Returns the handle to the device context specified (hDC).

Type: specifies the type of the handle device context to get, can be one of the following constants: BT_HDC_DESKTOP (hDC to desktop), BT_HDC_WINDOW (hDC to all window), BT_HDC_ALLCLIENTAREA (hDC to the all client area),

BT_HDC_INVALIDCLIENTAREA (hDC to the region of the client area that is not valid: invalid client area, i.e. needs to be repainted) or BT_HDC_BITMAP (hDC to a bitmap) (defined in BosTaurus.CH).

Win_or_hBitmap: is the name (e.g. "Win1") or is the handle (e.g. hWin) to the window if *Type* is set as BT_HDC_WINDOW, BT_HDC_ALLCLIENTAREA or BT_HDC_INVALIDCLIENTAREA. *Win_or_hBitmap* is a handle to bitmap (hBitmap) if *Type* is set as BT_HDC_BITMAP. If *Type* is set as BT_HDC_DESKTOP, *Win_or_hBitmap* is ignored.

BTstruct: is a variable (array) passed by reference that stores information about the hDC obtained. This information is necessary to properly release the hDC obtained.

BT_DeleteDC(BTstruct)

Release from the memory a handle of device context.

BTstruct: is an array with information about a determined handle of device context returned by **BT_CreateDC** function.

Remarks

The ***Bos Taurus*** graphic library uses the *common* Device Contexts (DC). The common DCs are _Display _device contexts maintained in a special cache by the system. Because only a limited number of common device contexts exist, an application should release them after it has finished drawing. Because new display device contexts for the cache are allocated in the application's heap space, failure to release the device contexts eventually consumes all available heap space this causes an error when it cannot allocate space for the new device context. **This means that you should get the hDC, draw and then immediately release the hDC, do not store the hDC to process other events drawing later.**

The best way to process the ON PAINT event in HMG is get the handle of the Device Context (hDC) of the client area with *Type* = BT_HDC_INVALIDCLIENTAREA in the BT_CreateDC function. This way is fast for drawing and causes less display flicker that with

BT_HDC_ALLCLIENTAREA. The handle of the DC obtained with BT_HDC_ALLCLIENTAREA is indicated to paint the client area out of the ON PAINT event.

Functions for Drawing on a Device Context

BT_DrawGetPixel(hDC, Row, Col)

Gets RGB color value of the pixel at the specified coordinates to the Device Context (DC) specified and returns a array with the color = {R,G,B}.

hDC: is a handle to the device context.

Row, Col: specifies the coordinates in pixels in the DC.

BT_DrawSetPixel(hDC, Row, Col, aRGBcolor)

Sets RGB color value of the pixel at the specified coordinates to the Device Context (DC) specified.

hDC: is a handle to the device context.

Row, Col: specifies the coordinates in pixels in the DC.

aRGBcolor: is a array with the RGB color = {R,G,B}.

BT_DrawBitmap(hDC, Row, Col, Width, Height, Mode_Stretch, hBitmap)

Draws a bitmap in the Device Context (DC) specified.

hDC: is a handle to the device context.

Row, Col, Width, Height: specifies the size of the rectangle in pixels in the DC where you will draw the bitmap. For default: *Row* = 0, *Col* = 0, *Width* = BT_BitmapWidth(hBitmap), *Height* = BT_BitmapHeight(hBitmap).

Mode_Stretch: sets the mode as the bitmap is adjusts (is stretches or compresses) in the specified rectangle in the DC, it is one of the constants: BT_SCALE, BT_STRETCH or BT_COPY (defined in BosTaurus.CH).

hBitmap: is a handle to the bitmap.

BT_DrawBitmapTransparent(hDC, Row, Col, Width, Height, Mode_Stretch, hBitmap, aRGBcolor_transp)

Draws a transparent bitmap in the Device Context (DC) specified.

hDC: is a handle to the device context.

Row, Col, Width, Height: specifies the size of the rectangle in pixels in the DC where you will draw the bitmap. For default: *Row* = 0, *Col* = 0, *Width* = BT_BitmapWidth(hBitmap), *Height* = BT_BitmapHeight(hBitmap).

Mode_Stretch: sets the mode as the bitmap is adjusts (is stretches or compresses) in the specified rectangle in the DC, it is one of the constants: BT_SCALE, BT_STRETCH or BT_COPY (defined in BosTaurus.CH).

hBitmap: is a handle to the bitmap.

aRGBcolor_transp: array containing the RGB colors to treat as transparent in the bitmap. For default: *aRGBcolor_transp* = color of the first pixel of the origin bitmap.

BT_DrawBitmapAlphaBlend(hDC, Row, Col, Width, Height, Alpha, Mode_Stretch, hBitmap)

Draw a bitmap with Alpha Blend effect (that make the pixels to be transparent or semi transparent) in the Device Context (DC) specified.

hDC: is a handle to the device context.

Row, Col, Width, Height: specifies the size of the rectangle in pixels in the DC where you will draw the bitmap. For default: *Row* = 0, *Col* = 0, *Width* = BT_BitmapWidth(hBitmap), *Height* = BT_BitmapHeight(hBitmap).

Alpha: is alpha blending value to be applied to the entire source bitmap (range 0 to 255, transparent = 0, opaque = 255).

Mode_Stretch: sets the mode as the bitmap is adjusts (is stretches or compresses) in the specified rectangle in the DC, it is one of the constants: BT_SCALE, BT_STRETCH or BT_COPY (defined in BosTaurus.CH).

hBitmap: is a handle to the bitmap.

BT_DrawDCtoDC(hDC1, Row1, Col1, Width1, Height1, Mode_Stretch, hDC2, Row2, Col2, Width2, Height2)

Copy a rectangular area of a device context (origin) in another device context (destination).

hDC1: is a handle to the destination device context.

Row1, Col1, Width1, Height1: specifies the size of the rectangle in pixels in the destination DC where you will copy the origin DC.

Mode_Stretch: sets the mode as the rectangle in the origin DC is adjusts (is stretches or compresses) in the specified rectangle in the destination DC, it is one of the constants: BT_SCALE, BT_STRETCH or BT_COPY (defined in BosTaurus.CH).

hDC2: is a handle to the origin device context.

Row2, Col2, Width2, Height2: specifies the size of the rectangle in pixels in the origin DC to copied in the destination DC.

BT_DrawDCtoDCTransparent(hDC1, Row1, Col1, Width1, Height1, Mode_Stretch, hDC2, Row2, Col2, Width2, Height2, aRGBcolor_transp)

Copy a transparent rectangular area of a device context (origin) in another device context (destination).

hDC1: is a handle to the destination device context.

Row1, Col1, Width1, Height1: specifies the size of the rectangle in pixels in the destination DC where you will copy the origin DC.

Mode_Stretch: sets the mode as the rectangle in the origin DC is adjusts (is stretches or compresses) in the specified rectangle in the destination DC, it is one of the constants: BT_SCALE, BT_STRETCH or BT_COPY (defined in BosTaurus.CH).

hDC2: is a handle to the origin device context.

Row2, Col2, Width2, Height2: specifies the size of the rectangle in pixels in the origin DC to copied in the destination DC.

aRGBcolor_transp: array containing the RGB colors to treat as transparent in the origin DC. For default: *aRGBcolor_transp* = color of the first pixel of the origin DC.

BT_DrawDCtoDCAlphaBlend(hDC1, Row1, Col1, Width1, Height1, Alpha, Mode_Stretch, hDC2, Row2, Col2, Width2, Height2)

Copy a rectangular area of a device context (origin) with Alpha Blend effect (that make the pixels to be transparent or semi transparent) in another device context (destination).

hDC1: is a handle to the destination device context.

Row1, Col1, Width1, Height1: specifies the size of the rectangle in pixels in the destination DC where you will copy the origin DC.

Alpha: is alpha blending value to be applied to the entire source bitmap (range 0 to 255, transparent = 0, opaque = 255).

Mode_Stretch: sets the mode as the rectangle in the origin DC is adjusts (is stretches or compresses) in the specified rectangle in the destination DC, it is one of the constants: BT_SCALE, BT_STRETCH or BT_COPY (defined in BosTaurus.CH).

hDC2: is a handle to the origin device context.

Row2, Col2, Width2, Height2: specifies the size of the rectangle in pixels in the origin DC to copied in the destination DC.

BT_DrawGradientFillHorizontal(hDC, Row, Col, Width, Height, aColorRGBstart, aColorRGBend)

Fill a rectangular region from left to right (horizontal) in a Device Context (DC) with a background color that is interpolated from two colors specified.

hDC: is a handle to the device context.

Row, Col, Width, Height: specifies the size of the rectangle in pixels in the DC where it will be filled.

aColorRGBstart: array that contains the RGB colors with that will begin filling. For default: *aColorRGBstart* = {0,0,0} = BLACK

aColorRGBend: array that contains the RGB colors with that will end filling. For default: *aColorRGBend* = {255,255,255} = WHITE

BT_DrawGradientFillVertical(hDC, Row, Col, Width, Height, aColorRGBstart, aColorRGBend)

Fill a rectangular region from top to bottom (vertical) in a Device Context (DC) with a background color that is interpolated from two colors specified.

hDC: is a handle to the device context.

Row, Col, Width, Height: specifies the size of the rectangle in pixels in the DC where it will be filled.

aColorRGBstart: array that contains the RGB colors with that will begin filling. For default: *aColorRGBstart* = {255,255,255} = WHITE

aColorRGBend: array that contains the RGB colors with that will end filling. For default: *aColorRGBend* = {0,0,0} = BLACK

BT_DrawText(hDC, Row, Col, cText, cFontName, nFontSize, aFontColor, aBackColor, nTypeText, nAlingText, nOrientation)

Write a character string in the Device Context (DC) specified.

hDC: is a handle to the device context.

Row, Col: specifies the coordinates in pixels in the DC.

cText: is the text to draw.

cFontName: is the name of the font, e.g. “Times New Roman”

nFontSize: is the size of the font in logical units, e.g. 12.

aFontColor: array that contains the RGB colors that set the text color. For default *aFontColor* = {0,0,0} = BLACK

aBackColor: array that contains the RGB colors that set the background text color. For default *aBackColor* = {255,255,255} = WHITE

nTypeText: sets if background fills (opaque) or not (transparent) with the BackColor color before the text it draws, it is one of the constants: BT_TEXT_OPAQUE or BT_TEXT_TRANSPARENT (defined in BosTaurus.CH). The characteristic of the font is set with one of the following constants: BT_TEXT_BOLD, BT_TEXT_ITALIC, BT_TEXT_UNDERLINE, BT_TEXT_STRIKEOUT. For default: *nTypeText* = BT_TEXT_OPAQUE.

nAlingText: sets the horizontal and vertical alignment of the text with respect of specified coordinates in *Col* and *Row* respectively. The horizontal alignment is set with one of the following constants: BT_TEXT_LEFT, BT_TEXT_CENTER or BT_TEXT_RIGHT (defined in BosTaurus.CH). The vertical alignment is set with one of the following constants: BT_TEXT_TOP, BT_TEXT_BASELINE or BT_TEXT_BOTTOM (defined in BosTaurus.CH). For default *nAlingText* = BT_TEXT_LEFT + BT_TEXT_TOP

nOrientation: sets the orientation of the text with respect of specified coordinates in *Col* and *Row* (range -90 to +90º). In BosTaurus.CH is defines some constants: BT_TEXT_NORMAL_ORIENTATION, BT_TEXT_VERTICAL_ASCENDANT, BT_TEXT_VERTICAL_DESCENDANT, BT_TEXT_DIAGONAL_ASCENDANT, BT_TEXT_DIAGONAL_DESCENDANT. For default *nOrientation* = BT_TEXT_NORMAL_ORIENTATION.

BT_DrawTextEx(hDC, Row, Col, Width, Height, cText, cFontName, nFontSize, aFontColor, aBackColor, nTypeText, nAlignText, nOrientation)

Write a character string in a given rectangle in the Device Context (DC) specified.

hDC: is a handle to the device context.

Row, Col, Width, Height: specifies the size of the rectangle in pixels in the DC where it will be written.

cText: is the text to draw.

cFontName: is the name of the font, e.g. “Times New Roman”

nFontSize: is the size of the font in logical units, e.g. 12.

aFontColor: array that contains the RGB colors that set the text color. For default *aFontColor* = {0,0,0} = BLACK

aBackColor: array that contains the RGB colors that set the background text color. For default *aBackColor* = {255,255,255} = WHITE

nTypeText: sets if background fills (opaque) or not (transparent) with the BackColor color before the text it draws, it is one of the constants: BT_TEXT_OPAQUE or BT_TEXT_TRANSPARENT (defined in BosTaurus.CH). The characteristic of the font is set with one of the following constants: BT_TEXT_BOLD, BT_TEXT_ITALIC, BT_TEXT_UNDERLINE, BT_TEXT_STRIKEOUT. For default: *nTypeText* = BT_TEXT_OPAQUE.

nAlignText: sets the horizontal and vertical alignment of the text with respect of specified coordinates in *Col* and *Row* respectively. The horizontal alignment is set with one of the following constants: BT_TEXT_LEFT, BT_TEXT_CENTER or BT_TEXT_RIGHT (defined in BosTaurus.CH). The vertical alignment is set with one of the following constants: BT_TEXT_TOP, BT_TEXT_BASELINE or BT_TEXT_BOTTOM (defined in BosTaurus.CH). For default *nAlingText* = BT_TEXT_LEFT + BT_TEXT_TOP

nOrientation: sets the orientation of the text with respect of specified coordinates in *Col* and *Row* (range -90 to +90º). In BosTaurus.CH is defines some constants: BT_TEXT_NORMAL_ORIENTATION, BT_TEXT_VERTICAL_ASCENDANT, BT_TEXT_VERTICAL_DESCENDANT, BT_TEXT_DIAGONAL_ASCENDANT, BT_TEXT_DIAGONAL_DESCENDANT. For default *nOrientation* = BT_TEXT_NORMAL_ORIENTATION.

BT_DrawTextSize(hDC, cText, cFontName, nFontSize, nTypeText)

Gets the size in pixels of a string of characters for the Device Context (DC) specified and returns an array with six elements: ***{ nTextWidth, nTextHeight, A+B+C of first character, A, B, C }***.

Where: ***A*** spacing is the distance added to the current position before placing the glyph. ***B*** spacing is the width of the black part of the glyph. ***C*** spacing is the distance added to the current position to provide white space to the right of the glyph. The total advanced width is specified by ***A+B+C***.

hDC: is a handle to the device context.

cText: is the text to get size.

cFontName: is the name of the font, e.g. "Times New Roman"

nFontSize: is the size of the font in logical units, e.g. 12.

nTypeText: is the characteristic of the font (BT_TEXT_BOLD, BT_TEXT_ITALIC, BT_TEXT_UNDERLINE, BT_TEXT_STRIKEOUT).

BT_DrawPolyLine(hDC, aPointY, aPointX, aColorRGBLine, nWidthLine)

Draws a series of line segments by connecting the points the arrays specified.

hDC: is a handle to the device context.

aPointY: array containing the coordinates in pixels of the points on the y-axis.

aPointX: array containing the coordinates in pixels of the points on the x-axis.

aColorRGBLine: array that contains the RGB colors that set of the lines color.

nWidthLine: width in pixels of the lines. For default: *nWidthLine* = 1.

BT_DrawPolygon(hDC, aPointY, aPointX, aColorRGBLine, nWidthLine, aColorRGBFill)

Draws a polygon consisting of two or more vertices connected by straight lines.

hDC: is a handle to the device context.

aPointY: array containing the coordinates in pixels of the points on the y-axis.

aPointX: array containing the coordinates in pixels of the points on the x-axis.

aColorRGBLine: array that contains the RGB colors that set of the lines color.

nWidthLine: width in pixels of the lines. For default: *nWidthLine* = 1.

aColorRGBFill: array that contains the RGB colors that set of the fill color.

BT_DrawPolyBezier(hDC, aPointY, aPointX, aColorRGBLine, nWidthLine)

Draws one or more Bézier curves.

hDC: is a handle to the device context.

aPointY: array containing the coordinates in pixels of the points on the y-axis.

aPointX: array containing the coordinates in pixels of the points on the x-axis.

aColorRGBLine: array that contains the RGB colors that set of the lines color.

nWidthLine: width in pixels of the lines. For default: *nWidthLine* = 1.

BT_DrawArc(hDC, Row1, Col1, Row2, Col2, RowStartArc, ColStartArc, RowEndArc, ColEndArc, aColorRGBLine, nWidthLine)

Draws an elliptical arc.

hDC: is a handle to the device context.

Row1, Col1: specifies the upper-left corner of the bounding rectangle in pixels.

Row2, Col2: specifies the lower-right corner of the bounding rectangle in pixels.

_RowStartArc, ColStartArc: _specifies the ending point of the radial line defining the starting point of the arc in pixels.

RowEndArc, ColEndArc: specifies the ending point of the radial line defining the ending point of the arc in pixels.

aColorRGBLine: array that contains the RGB colors that set of the line color.

nWidthLine: width in pixels of the line. For default: *nWidthLine* = 1.

BT_DrawChord(hDC, Row1, Col1, Row2, Col2, RowStartArc, ColStartArc, RowEndArc, ColEndArc, aColorRGBLine, nWidthLine, aColorRGBFill)

Draws a chord (a region bounded by the intersection of an ellipse and a line segment, called a secant).

hDC: is a handle to the device context.

Row1, Col1: specifies the upper-left corner of the bounding rectangle in pixels.

Row2, Col2: specifies the lower-right corner of the bounding rectangle in pixels.

_RowStartArc, ColStartArc: _specifies the ending point of the radial line defining the starting point of the chord in pixels.

RowEndArc, ColEndArc: specifies the ending point of the radial line defining the ending point of the chord in pixels.

aColorRGBLine: array that contains the RGB colors that set of the line color.

nWidthLine: width in pixels of the line. For default: *nWidthLine* = 1.

aColorRGBFill: array that contains the RGB colors that set of the fill color.

BT_DrawPie(hDC, Row1, Col1, Row2, Col2, RowStartArc, ColStartArc, RowEndArc, ColEndArc, aColorRGBLine, nWidthLine, aColorRGBFill)

Draws a pie-shaped wedge bounded by the intersection of an ellipse and two radials.

hDC: is a handle to the device context.

Row1, Col1: specifies the upper-left corner of the bounding rectangle in pixels.

Row2, Col2: specifies the lower-right corner of the bounding rectangle in pixels.

_RowStartArc, ColStartArc: _specifies the endpoint of the first radial in pixels.

RowEndArc, ColEndArc: specifies the endpoint of the second radial in pixels.

aColorRGBLine: array that contains the RGB colors that set of the line color.

nWidthLine: width in pixels of the line. For default: *nWidthLine* = 1.

aColorRGBFill: array that contains the RGB colors that set of the fill color.

BT_DrawLine(hDC, Row1, Col1, Row2, Col2, aColorRGBLine, nWidthLine)

Draw a line by connecting two points specified.

hDC: is a handle to the device context.

Row1, Col1: are the coordinates in pixels of the starting point.

Row2, Col2: are the coordinates in pixels of the end point.

aColorRGBLine: array that contains the RGB colors that set of the line color.

nWidthLine: width in pixels of the line. For default: *nWidthLine* = 1.

BT_DrawRectangle(hDC, Row, Col, Width, Height, aColorRGBLine, nWidthLine)

Draw a rectangle with the specified dimensions.

hDC: is a handle to the device context.

Row, Col, Width, Height: specifies the dimensions in pixels of the rectangle to be draw.

aColorRGBLine: array that contains the RGB colors that set of the lines color.

nWidthLine: width in pixels of the lines. For default: *nWidthLine* = 1.

BT_DrawEllipse(hDC, Row1, Col1, Width, Height, aColorRGBLine, nWidthLine)

Draw an ellipse with the specified dimensions.

hDC: is a handle to the device context.

Row, Col, Width, Height: specifies the dimensions in pixels of the ellipse to be draw.

aColorRGBLine: array that contains the RGB colors that set of the line color.

nWidthLine: width in pixels of the line. For default: *nWidthLine* = 1.

BT_DrawFillRectangle(hDC, Row, Col, Width, Height, aColorRGBFill, aColorRGBLine, nWidthLine)

Draw a fill rectangle with the specified dimensions.

hDC: is a handle to the device context.

Row, Col, Width, Height: specifies the dimensions in pixels of the rectangle to be draw.

aColorRGBFill: array that contains the RGB colors that set of the fill color.

aColorRGBLine: array that contains the RGB colors that set of the lines color. For default: *aColorRGBLine = aColorRGBFill.*

nWidthLine: width in pixels of the lines. For default: *nWidthLine* = 1.

BT_DrawFillEllipse(hDC, Row, Col, Width, Height, aColorRGBFill, aColorRGBLine, nWidthLine)

Draw a fill ellipse with the specified dimensions.

hDC: is a handle to the device context.

Row, Col, Width, Height: specifies the dimensions in pixels of the ellipse to be draw.

aColorRGBFill: array that contains the RGB colors that set of the fill color.

aColorRGBLine: array that contains the RGB colors that set of the line color. For default: *aColorRGBLine = aColorRGBFill.*

nWidthLine: width in pixels of the line. For default: *nWidthLine* = 1.

BT_DrawFillRoundRect(hDC, Row, Col, Width, Height, RoundWidth, RoundHeight, aColorRGBFill, aColorRGBLine, nWidthLine)

Draw a fill rectangle with rounded corners.

hDC: is a handle to the device context.

Row, Col, Width, Height: specifies the dimensions in pixels of the rectangle to be draw.

RoundWidth: specifies the width in pixels of the ellipse used to draw the rounded corners.

RoundHeight: specifies the height in pixels of the ellipse used to draw the rounded corners.

aColorRGBFill: array that contains the RGB colors that set of the fill color.

aColorRGBLine: array that contains the RGB colors that set of the line color. For default: *aColorRGBLine = aColorRGBFill.*

nWidthLine: width in pixels of the line. For default: *nWidthLine* = 1.

BT_DrawFillFlood(hDC, Row, Col, aColorRGBFill)

Fill an area in all directions defined by the color of the point of the specified coordinates.

hDC: is a handle to the device context.

Row, Col: specifies the coordinates in pixels of the point where begin filling.

aColorRGBFill: array that contains the RGB colors that set of the fill color.

Functions of Connection Between HMG Controls and Bos Taurus

BT_HMGGetImage(cFormName, cControlName)

Returns the handle of the bitmap associated to an Image Control of HMG (@...IMAGE).

cFormName: is the name (e.g. "Win1") of the parent window.

cControlName: is the name (e.g. "Image1") of the image control.

BT_ HMGCloneImage(cFormName, cControlName)

Clones the bitmap associated to an Image Control of HMG (@...IMAGE) and returns a handle to the cloned bitmap.

cFormName: is the name (e.g. "Win1") of the parent window.

cControlName: is the name (e.g. "Image1") of the image control.

BT_HMGSetImage(cFormName, cControlName, hBitmap, lReleasePrevious)

Sets a specified bitmap into an Image Control of HMG (@...IMAGE) and automatically releases the handle of the bitmap previously associated to the Image Control.

cFormName: is the name (e.g. "Win1") of the parent window.

cControlName: is the name (e.g. "Image1") of the image control.

hBitmap: is a handle to the bitmap to set.

lReleasePrevious: releases of the hBitmap previous associate to Image Control. For default is .T.

Joke

You still have any doubt that Bos Taurus is the best graphics library for HMG?

HMG Advanced Functions

DISABLE\/ENABLE EVENTS

This commands\/function prevents re-entry while processing the events of a control or window allowing the use of other controls\/functions that generate messages of re-called of events

Syntax:

```
DISABLE [ CONTROL ] EVENT ControlName OF FormName
ENABLE [ CONTROL ] EVENT ControlName OF FormName
StopControlEventProcedure ( cControlName, cFormName, lStop )
DISABLE [ WINDOW ] EVENT OF FormName
ENABLE [ WINDOW ] EVENT OF FormName
StopWindowEventProcedure ( cFormName, lStop ) -
```

Complementary Functions:

```
GetLastActiveFormIndex () --> Return nFormIndex
GetLastActiveControlIndex () --> Return nControlIndex
GetFormNameByIndex ( nFormIndex ) ---> Return cFormName
GetControlNameByIndex ( nControlIndex ) ---> Return cControlName
```

Example:

```
#include "hmg.ch"
FUNCTION Main
   DEFINE WINDOW Form_1;
      AT 0,0;
      WIDTH 400;
      HEIGHT 300;
      ON GOTFOCUS Form_ONGOTFOCUS();
      MAIN
      @ 50, 50 BUTTON Button_1 CAPTION "Click" ACTION MsgInfo ("Hello")
      @ 100, 50 BUTTON Button_2 CAPTION "Minimize" ACTION Form_1.Minimize
      @ 150, 50 TIMEPICKER TimePicker_1 ON GOTFOCUS Control_ONGOTFOCUS ()
   END WINDOW
   CENTER WINDOW Form_1
   ACTIVATE WINDOW Form_1
RETURN

PROCEDURE Form_ONGOTFOCUS
   LOCAL i := GetLastActiveControlIndex ()
   DISABLE WINDOW EVENT OF Form_1
   MsgInfo ("ON GOTFOCUS: Form_1 " + IIF (i > 0," - Last Control Focused: "+ GetContro
lNameByIndex(i), ""))
   ENABLE WINDOW EVENT OF Form_1
RETURN

PROCEDURE Control_ONGOTFOCUS
   LOCAL i := GetLastActiveFormIndex ()
   DISABLE WINDOW EVENT OF Form_1 // --> StopWindowEventProcedure ("Form_1", .T.)
   DISABLE CONTROL EVENT TimePicker_1 OF Form_1 // --> StopControlEventProcedure ("Tim
ePicker_1", "Form_1", .T.)
   MsgInfo ("ON GOTFOCUS: TimePicker_1" + IIF (i > 0," - Last Form Focused: "+ GetForm
NameByIndex(i), ""))
   ENABLE CONTROL EVENT TimePicker_1 OF Form_1 // --> StopControlEventProcedure ("Time
Picker_1", "Form_1", .F.)
   ENABLE WINDOW EVENT OF Form_1 // --> StopWindowEventProcedure ("Form_1", .F.)
RETURN
```

CREATE EVENT

Commands and Functions for Manage Events

Syntax:

```
CREATE EVENT PROCNAME cProcName [ HWND hWnd ] [ MSG nMsg ] [ STOREINDEX nIndex ]
CREATE EVENT CODEBLOCK bCodeBlock [ HWND hWnd ] [ MSG nMsg ] [ STOREINDEX nIndex ]
nIndex := EventCreate ( cProcName | bCodeBlock, hWnd, nMsg )
EventRemove ( nIndex )
EventIsInProgress () --> lBoolean
EventIsKeyboardMessage () --> lBoolean
EventIsMouseMessage () --> lBoolean
EventIsHMGWindowsMessage () --> lBoolean
EventINDEX () --> nIndex
EventPROCNAME () --> cProcName | bCodeBlock
EventHWND () --> hWnd
EventMSG () --> nMsg
EventWPARAM () --> wParam
EventLPARAM () --> lParam
EventGetPROCNAME ( nIndex ) --> cProcName | bCodeBlock
EventGetHWND ( nIndex ) --> hWnd
EventGetMSG ( nIndex ) --> nMsg
EventSTOP ( nIndex, [ lStop ] ) --> lBoolean
EventProcessKeyboardMessage ( nIndex, [ lProcess ] ) --> lBoolean
EventProcessMouseMessage ( nIndex, [ lProcess ] ) --> lBoolean
EventProcessHMGWindowsMessage ( nIndex, [ lProcess ] ) --> lBoolean
EventProcessAllHookMessage ( nIndex, [ lProcess ] ) --> lBoolean
```

Complementary Functions:

```
GetSplitChildWindowHandle ( cFormName, cParentForm )
GetSplitBoxHandle ( cParentForm )
```

See demos:

```
\SAMPLES\Controls\EditBox\EditBoxOverwrite
\SAMPLES\Controls\Grid\GridIncrementalSearch
```

READ KEYBOARD AND MOUSE FUNCTIONS

Syntax:

```
HMG_GetLastVirtualKeyDown ( [ @hWnd ], [ @nMsg ], [ @wParam ], [ @lParam ] ) --> nVK_C
ode
HMG_GetLastVirtualKeyUp ( [ @hWnd ], [ @nMsg ], [ @wParam ], [ @lParam ] ) --> nVK_Cod
e
HMG_GetLastCharacter ( [ @hWnd ], [ @nMsg ], [ @wParam ], [ @lParam ] ) --> cCharacter
HMG_CleanLastVirtualKeyDown ( [ lCleanAll ] )
HMG_CleanLastVirtualKeyUp ( [ lCleanAll ] )
HMG_CleanLastCharacter ( [ lCleanAll ] )
HMG_GetLastVirtualKeyName ( [ lParam ] ) --> cVK_Name
HMG_VirtualKeyIsPressed ( VK_Code )
GetKeyState ( VK_Code ) --> return nKeyState
HMG_SendCharacter ( [ hWnd ], cText )

HMG_PressKey ( nVK1, nVK2, ... )
Note: simulates the pressure of a key or a combination of keys (where nVK is a virtual
 key code)

HMG_GetLastMouseMessage ( [ @hWnd ], [@nMsg], [@wParam], [@lParam] ) --> nMsg
HMG_CleanLastMouseMessage()
HMG_GetCursorPos ( [ hWnd ], @nRow, @nCol )
HMG_SetCursorPos ( [ hWnd ], nRow, nCol )
Note: if hWnd is a handle of window nRow and nCol is in reference to the coordinates o
f window,  otherwise the cursor position is in screen coordinates.

SET CONTROL <ControlName> OF <FormName> ONKEYEVENT <FuncName> | NIL

Note: the following function is a complement to SET ONKEYEVENT
HMG_GetOnKeyControlIndex ( [ @nSubIndex ] ) --> nIndex

SET CONTROL <ControlName> OF <FormName> ONMOUSEEVENT <FuncName> | NIL

Note: the following function is a complement to SET ONMOUSEEVENT
HMG_GetOnMouseControlIndex ( [ @nSubIndex ] ) --> nIndex
```

Complementary Functions:

Gets Form\/Control Name and Form Parent Name by window handle (complement, e.g. to the read keyboard functions):

```
GetFormNameByHandle ( hWnd, @cFormName, @cFormParentName ) --> Return nFormIndex
GetControlNameByHandle ( hWnd, @cControlName, @cFormParentName ) --> Return nControlIn
dex
```

See demo:

`\SAMPLES\Controls\Grid\GridIncrementalSearch`

Memory, Processes and Threads

```
RELEASE MEMORY, release unused memory (leak memory), use for example:
ON KEY F5 ACTION ( RELEASE MEMORY )
DEFINE TIMER ... ACTION ( RELEASE MEMORY )
GetCurrentProcessID() --> return nProcessID
EnumProcessID () ---> return array { nProcessID1, nProcessID2, ... }
GetWindowThreadProcessID ( hWnd, @nThread, @nProcessID )
```

WoW64:

```
IsWow64Process ( [ nProcessID ] ) --> return lBoolean
- return TRUE if a 32-bit application is running under 64-bit Windows (WOW64)
- return FALSE if a 32-bit application is running under 32-bit Windows
- return FALSE if a 64-bit application is running under 64-bit Windows
Note: WOW64 (Win32 On Win64) is the x86 emulator that allows 32-bit Windows-based appl
ications to running on 64-bit Windows
```

Process Info:

```
GetProcessName ( [ nProcessID ] ) --> return cProcessName
GetProcessFullName ( [ nProcessID ] ) --> return cProcessFullName
GetProcessImageFileName ( [ nProcessID ] ) --> return cProcessImageFileName
TerminateProcess ( [ nProcessID ] , [ nExitCode ] )
EmptyWorkingSet( [ ProcessID ] ) ---> lBoolean release unused memory (leak memory)
```

GetProcessMemoryInfo()

```
GetProcessMemoryInfo ( [ ProcessID ] ) --> returns an array with 9 numbers with used m
emory info for a process
- PageFaultCount, // The number of page faults
- PeakWorkingSetSize,
- WorkingSetSize, // The current working set size, in bytes
- QuotaPeakPagedPoolUsage,
- QuotaPagedPoolUsage, // The current paged pool usage, in bytes
- QuotaPeakNonPagedPoolUsage,
- QuotaNonPagedPoolUsage, // The current nonpaged pool usage, in bytes
- PagefileUsage, // Total amount of memory that the memory manager has committed for t
he running this process, in bytes
- PeakPagefileUsage
```

GlobalMemoryStatusEx ()

```
GlobalMemoryStatusEx () --> returns an array with 7 numbers with used memory info for
the system
- MemoryLoad, // approximate percentage of physical memory that is in use (0 indicates
 no memory use and 100 indicates full memory use)
- TotalPhys, // amount of actual physical memory, in bytes
- AvailPhys, // amount of physical memory currently available, in bytes
- TotalPageFile, // current committed memory limit for the system or the current proce
ss, whichever is smaller, in bytes
- AvailPageFile, // maximum amount of memory the current process can commit, in bytes
- TotalVirtual, // size of the user-mode portion of the virtual address space of the c
alling process, in bytes
- AvailVirtual, // amount of unreserved and uncommitted memory currently in the user-m
ode portion of the virtual address space of the calling process, in bytes
```

HMG_GetObjectCount([nProcessId])

```
HMG_GetObjectCount( [ nProcessId ] ) --> returns an array of 3 items with information
about the number of system objects used for a process { nGDIObjects, nUserObjects, nKe
rnelObjects }
```

GDI Objects:\\___ \\_[_https:\/\/msdn.microsoft.com\/en-us\/library\/ms724291(v=vs.85).aspx_](https:\/\/msdn.microsoft.com\/en-us\/library\/ms724291%28v=vs.85%29.aspx)

Bitmap

Brush

DC

Enhanced metafile

Enhanced-metafile DC

Font

Memory DC

Metafile

Metafile DC

Palette

Pen and extended pen

Region

User Objects:\\___ \\_[_https:\/\/msdn.microsoft.com\/en-us\/library\/ms725486(v=vs.85).aspx_](https:\/\/msdn.microsoft.com\/en-us\/library\/ms725486%28v=vs.85%29.aspx)

Accelerator table

Caret

Cursor

DDE conversation

Hook

Icon

Menu

Window

Window position

Kernel Objects:\\___ \\_[_https:\/\/msdn.microsoft.com\/en-us\/library\/windows\/desktop\/ms724485(v=vs.85).aspx_](https:\/\/msdn.microsoft.com\/en-us\/library\/windows\/desktop\/ms724485%28v=vs.85%29.aspx)

Access token

Change notification

Communications device

Console input

Console screen buffer

Desktop

Event

Event log

File

File mapping

Find file

Heap

I\/O completion port

Job

Mailslot

Memory resource notification

Module

Mutex

Pipe

Process

Semaphore

Socket

Thread

Timer

Update resource

Window station

NOTE about 32\/64 bit processes, in conventional way:

- with apps of 32-bit is possible only get information about of 32-bit processes
- with apps of 32-bit is not possible get information about of 64-bit processes
- with apps of 64-bit is possible get information about of 32 and 64-bit processes

CHECK TYPE

Check at runtime the specified type of a variable or function\/procedure parameter

The checking the type of a variable or a function\/procedure parameter is useful for a ***defensive programming*** (minimization of input errors)

```
CHECK TYPE [ SOFT ] <VarName1> AS <VarType1> [ , < VarName2> AS < VarType2> [, < VarNa
meN> AS < VarTypeN> ] ]
```

Where:

```
<VarName> --> is the name of variable
<VarType> --> is one of the following variable types:
- ARRAY
- BLOCK
- CHARACTER
- DATE
- HASH
- LOGICAL
- NIL
- NUMERIC
- MEMO
- POINTER
- SYMBOL
- TIMESTAMP
- OBJECT
- USUAL (the variable can take any of the above types, in the practice the variable ty
pe is not checked)
```

SOFT --> when the SOFT clause is specified the variable can take the NIL value or the specified value, otherwise the variable can take only the value specified

Example:

```
Procedure MyDisplay( cName, nAge, aColor, xValue )
   LOCAL Today := DATE()
   CHECK TYPE cName AS CHARACTER ,;
   nAge AS NUMERIC ,;
   aColor AS ARRAY ,;
   xValue AS USUAL ,;
   Today AS DATE
 ...
Return
```

Note: If any of the variable does not have the specified type an error message is displayed and the program is aborted.

USER COMPONENTS INTERFACE

Standard Interface To Allow HMG Users Create Its Own Components

InstallEventHandler ()

```
InstallEventHandler ( <cEventHandlerFunctionName> )
```

Every time that an event is triggered by the system the specified function will be called.

This function will receive the following parameters: hWnd, nMsg, wParam, lParam

If your function process the message, it must return a not NIL value. If not, it must return NIL.

InstallMethodHandler()

```
InstallMethodHandler(<cMethodName>,<cMethodHandlerProcName>)
```

Every time that specified method is called, your procedure will be executed.

If your procedure process the method, you must set the 'user process flag' (_HMG_SYSDATA [63]) to .T. If not, you must

set it to .F.

InstallPropertyHandler()

```
InstallPropertyHandler(<cPropName>,<cSetFucName>,<cGetFuncName>)
```

When the specified property is set, <cSetFucName> is called with all the original parameters

When the specified property is value retrieved <cGetFucName> is called with all the original parameters.

In both cases if your functions process the property, you must set the 'user process flag' (_HMG_SYSDATA [63]) to .T. If not, you must set it to .F.

Other Considerations:

If you've created components that requires #command directives (ie. a control) you must add these definitions to i_UsrInit.ch file located at HMG's INCLUDE folder.

The definitions must include start and end markers that will be used to allow the creation of automated installation and maintenance procedures for components.

The format for start marker is:

```
 #define BASEDEF_<YourComponentTypeName>
```

The format for end marker is:

```
#undef BASEDEF_<YourComponentTypeName>
```

If you've created methods or properties with new names (not currently specified in i_windows.ch, DECLARE WINDOW command definition) you must add translation directives for them in **'i_UsrSOOP.ch**' file, located at HMG's INCLUDE folder.

The definitions must include start and end markers too and the translation directives must follow special rules.

The format for start marker is:

```
 #define SOOP_<YourComponentTypeName> ;;
```

The format for translation directives for methods is:

```
#xtranslate <Window> . \<Control\> . MethodName => Domethod ( <"Window">, \<"Control"\
> , "MethodName" ) ;;
```

The format for translation directives for properties is:

```
 #xtranslate <Window> . \<Control\> . PropName => GetProperty ( <"Window">, \<"Control
"\> , "PropName" ) ;;
 #xtranslate <Window> . \<Control\> . PropName := \<v\> => SetProperty ( <"Window">, \
<"Control"\> , "PropName" , \<v\> ) ;;
```

The format for end marker is:

```
#undef SOOP_<YourComponentTypeName> ;;
```

Sample:

`c:\hmg\samples\user_components`

HMG External Help\/Reference\/Guide:

HMG Online Documentation: http:\/\/www.hmgforum.com\/hmgdoc\/hmgdoc.htm

Harbour Reference Guide: http:\/\/harbour.github.io\/doc\/index.html

Harbour for Beginners: http:\/\/www.kresin.ru\/en\/hrbfaq_3.html

CA-Clipper 5.3
http:\/\/www.itlnet.net\/programming\/program\/Reference\/c53g01c\/menu.html

HMG Tutorial http:\/\/www.elektrosoft.it\/tutorials\/hmg\/hmg.asp

HMG Tutorial YouTube Channel
https:\/\/www.youtube.com\/channel\/UCcINdVHnB_76DRFmf1D88QA

HMG- Viva Cllipper http:\/\/vivaclipper.wordpress.com\/category\/hmg\/

Introduction to HFCL

Library collection

HMG Forum Component Library (HFCL) is a collection of small utilities, enhancements, add-on functions to HMG controls. It has been created by a group of volunteers who are members of HMG Forum under the valuable guidance of Mr. Roberto Lopez (the founder of HMG itself). As of now, HFCL has several components\/utilities, for more detail see the documentation and **demos in folder: SAMPLES\HFCL**

How to use HFCL:

It's simple, as currently the HFCL functions are linked automatically by the compiler with the library of HMG, therefore only need to include in their applications the **header file HFCL.ch**

Sample:

```
#include "HMG.ch"
#include "HFCL.ch"
FUNCTION Main()
 // Define windows, controls, call HFCL functions, etc.
RETURN NIL
```

AutoFill

Add auto filling feature to a **TextBox Control**.

- ***Usage***: Build an array for words to filling in TextBox entry
- Specify **AutoFill()** as ONCHANGE procedure of TextBox
- Specify **AFKeySet()** as ONGOTFOCUS procedure of TextBox
- Specify **AFKeyRls()** as ONLOSTFOCUS procedure of TextBox
- ***Example***:

```
aCountries := {"Afghanistan", "Albania", "Algeria", "Andorra", "Angola", ... }
ASORT( aCountries ) // This Array MUST be sorted
DEFINE TEXTBOX TextBoxCountries
   ROW 48
   COL 110
   ONCHANGE AutoFill( aCountries )
   ONGOTFOCUS AFKeySet( aCountries )
   ONLOSTFOCUS AFKeyRls( )
END TEXTBOX
```

Authors:

Started by B. Esgici <esgici@gmail.com>

Enhanced by Roberto Lopez <mail.box.hmg@gmail.com> and S. Rathinagiri <srathinagiri@gmail.com>

@...COMBOSEARCHBOX \/ DEFINE COMBOSEARCHBOX

Define a Combined Search Box Control

Syntax:

```
 @ <nRow> ,<nCol> COMBOSEARCHBOX <ControlName>
    [ OF | PARENT <ParentWindowName> ]
    [ HEIGHT <nHeight> ]
    [ WIDTH <nWidth> ]
    [ VALUE <nValue> ]
    [ FONT <cFontName> SIZE <nFontSize> ]
    [ BOLD ] [ ITALIC ] [ UNDERLINE ] [ STRIKEOUT ]
    [ TOOLTIP <cToolTipText> ]
    [ BACKCOLOR <aBackColor> ]
    [ FONTCOLOR <aFontColor> ]
    [ MAXLENGTH <nInputLength> ]
    [ UPPERCASE | LOWERCASE ]
    [ ON GOTFOCUS <OnGotFocusProcedur> | <bBlock> ]
    [ ON LOSTFOCUS <OnLostFocusProcedure> | <bBlock> ]
    [ ON ENTER <OnEnterProcedure> | <bBlock> ]
    [ RIGHTALIGN ]
    [ NOTABSTOP ]
    [ HELPID <nHelpId> ]
    [ ITEMS <aItems> ] // the aItems array MUST be sorted
    [ ANYWHERESEARCH ]
    [ DROPHEIGHT <nDropHeight> ]
    [ ADDITIVE <lAdditive> ]
    [ ROWOFFSET <nRowOffset> ]
    [ COLOFFSET <nColOffset> ]
```

Alternative Syntax:

```
DEFINE COMBOSEARCHBOX <Controlname>
   <PropertyName> <PropertyValue>
   <EventName> <EventProcedure> | <bBlock>
END COMBOSEARCHBOX
```

Properties:

- Row
- Col
- Height
- Width
- Value

- Items
- FontName
- FontSize
- FontItalic
- FontUnderline
- FontStrikeout
- ToolTip
- BackColor
- FontColor
- CaretPos
- Name (R)
- TabStop (D)
- Parent (D)
- Numeric (D)
- MaxLength (D)
- UpperCase (D)
- LowerCase (D)
- RightAlign (D)
- HelpId (D)
- AnyWhereSearch(D)
- DropHeight (D)
- Additive (D)
- RowOffset (D)
- ColOffset (D)

D: Available at control definition only

R: Read-Only

- **Events**:

  - OnGotFocus
  - OnChange
  - OnLostFocus
  - OnEnter
- **Methods**:
  - Show
  - Hide
  - SetFocus
  - Release
  - Refresh
  - Save
- **Caution**:
  - The user can't have '**on change**' procedure block for this box, since this event is handled internally.
  - Enabling **Anywheresearch** would search inside the string anywhere instead from the first character.
  - **DropHeight** specifies the maximum dropdown height of the ComboSearchBox.
  - The **aItems** array MUST be sorted.
  - Additive option can be used to retain the entered text by the user even if the item is not available in the aItems. This will be useful, if you want to add the item to the list. Default value is .f.
  - RowOffset and ColOffset values can be used to fix the ComboSearchBox placement in Panel Windows and windows with nocaptions or themed. RowOffset\/ColOffset can be positive or negative to move the combosearchbox to be shown. Please see the sample for details.

Authors:

Started by Bicahi Esgici <esgici@gmail.com>

Enhanced by S.Rathinagiri <srathinagiri@gmail.com>

Revised by Grigory Filatov <gfilatov@inbox.ru>

HMG GridPrint

Grid\/Array data Print interface

HMG GridPrint is an interface to HMG print facility to print the contents of a **Grid control or Array** along with some additional features.

For the **Grid** control it can be called by two ways, viz., method and command.

- **Method**:

`<WindowName>.<ControlName>.Print(aHeaders)`

- **Command**:
- **Grid Data**

  ```
  PRINT GRID <cGridName> ;
     OF <cParentName> ;
     [ FONT <cFontName> ] ;
     [ SIZE <nFontSize> ] ;
     [ ORIENTATION <cOrientation> ] ;
     [ HEADERS <aHeaders> ] ;
     [ SHOWWINDOW ] ;
     [ MERGEHEADERS <aMergeHeaders> ] ;
     [ COLUMNSUM <aColumnSum> ] ;
     [ WIDTHS <aColumnWidths> ] ;
     [ PAPERSIZE <nPaperSize>] ;
     [ PAPERWIDTH <nPaperWidth>] ;
     [ PAPERHEIGHT <nPaperHeight>] ;
     [ VERTICALLINES ] ;
     [ HORIZONTALLINES ] ;
     [ LEFT <nLeft>] ;
     [ TOP <nTop>] ;
     [ RIGHT <nRight>] ;
     [ BOTTOM <nBottom>]
  ```

- **Array Data**

```
PRINT ARRAY <aArrayData> ;
   [ FONT <cFontName> ] ;
   [ SIZE <nFontSize> ] ;
   [ ORIENTATION <cOrientation> ] ;
   [ HEADERS <aHeaders> ] ;
   [ SHOWWINDOW ] ;
   [ MERGEHEADERS <aMergeHeaders> ] ;
   [ COLUMNSUM <aColumnSum> ] ;
   [ ARRAYHEADERS <aArrayHeaders>] ;
   [ ARRAYJUSTIFY <aArrayJustify>] ;
   [ WIDTHS <aColumnWidths> ] ;
   [ PAPERSIZE <nPaperSize>] ;
   [ PAPERWIDTH <nPaperWidth>] ;
   [ PAPERHEIGHT <nPaperHeight>] ;
   [ VERTICALLINES ] ;
   [ HORIZONTALLINES ] ;
   [ LEFT <nLeft>] ;
   [ TOP <nTop>] ;
   [ RIGHT <nRight>] ;
   [ BOTTOM <nBottom>]
```

Parameters for these commands:

cGridName: Name of the grid control

cParentName: Name of the parent window of the grid control

cFontName: Name of the font as a character string

nFontSize: Size of the font in points

cOrientation: “P” for Portrait and “L” for Landscape

aHeaders: An array of strings for the custom headers for the report. (Maximum of three headers allowed)

ShowWindow: If omitted, the grid would be printed directly to the printer without showing the

GridPrint features window. Even though 'showwindow' is omitted, if the columns don't fit in

the page width, the GridPrint window will be shown.

aMergeHeaders: To merge two or more columns and have a common header. For using this feature,

a two dimensional array of details of merged headers shall be passed with each

element in the following format,

```
{{nStartCol1 , nEndCol1 , cMergedHeader1},{nStartCol2 , nEndCol2 , cMergedHeader2},...
}
```

aColumnSum: Page-wise summation of some columns of the grid control can be achieved using this parameter.

A two dimensional array with number of elements equal to the number of columns,

in the following format,

```
{{lSumReqdCol1,cTransformMask1},{lSumReqdCol2,cTransformMask2},{lSumReqdCol3,cTransfor
mMask3},...}
```

shall be passed.

aColumnWidths: An array of column widths in mm.

nPaperSize: HMG Paper Description

nPaperWidth: Paper width in mm (only for custom paper size)

nPaperHeight: Paper height in mm (only for custom paper size)

VERTICALLINES: To draw column separator. Default is off.

HORIZONTALLINES: To draw row separator. Default is off.

nLeft: Left Margin in mm

nTop: Top Margin in mm

nRight: Right Margin in mm

nBottom: Bottom Margin in mm

- **Sample**:

```
PRINT GRID grid_1;
   OF form_1;
   FONT "Arial";
   SIZE 12;
   ORIENTATION "P";
   SHOWWINDOW ;
   HEADERS {"Header1","Header2"} ;
   MERGEHEADERS mergedheaders ;
   COLUMNSUM summation ;
   WIDTHS {0,20,20,20,40} ;
   PAPERSIZE PRINTER_PAPER_USER ;
   PAPERWIDTH 200 ;
   PAPERHEIGHT 200;
   VERTICALLINES;
   HORIZONTALLINES;
   LEFT 15;
   TOP 15;
   RIGHT 15;
   BOTTOM 15
```

GridPrint Window:

This window has many features to configure the report which are all self explanatory. They are listed here:

- Selection of columns to report (You can remove a particular column of the grid in reporting)
    - The print size of the columns can be altered.
    - Headers and Footers can be added\/modified.
    - Page numbers are printed on each page. It can be printed either on top or bottom of the page. Or, if it is not needed that can be removed too.
- Font size of the report can be changed.
- Column wise, row wise grid lines can be printed\/removed.
- Report can be vertically centered to look neatly.
- Extra White space between the columns can be spread, so that the report is to fill the width of the paper excluding the margin.
- Orientation of the report can be selected.
- Printer can be selected from a list.
- Standard page sizes or custom page size can be selected, so are the margins.

- A blue print preview of the first page is shown before getting printed.

Author:

S. Rathinagiri <srathinagiri@gmail.com>

HMG MakeShortcut

Make a file, folder or internet shortcut

Make a File or Folder shortcut (.lnk file):

```
MakeFDirShortCut (cTargetFileDir ,; // Full File or Folder name to be targeted
cShortCutTitle ,; // Short-cut (.lnk File) Name
cShortCutTTip ,; // Text to display when mouse over the short cut
cShortCutFolder ,; // Folder Name for short-cut (.lnk file ) Default is Desktop.
cIcon ,; // Icon for the shortcut
cShortCutWrkdir ) // Working Directory
```

Make Internet shortcut (.url file):

```
MakeInternetShortCut ( cTargetURL ,; // Internet address to be targeted
cShortCutTitle ,; // Short-cut (.url File) Name
cShortCutFolder ) // Folder Name for short-cut (.lnk file ) Default is Desktop.
```

- ***Example***:

```
MakeFDirShortCut ( 'c:\hmg.3.1.3' , 'HMG 3.1.3' , 'Go to HMG 3.1.3 base directory')
MakeInternetShortCut ( 'http://www.hmgforum.com' , 'HMG Forum' )
```

Authors:

B.P Dave and B. Esgici <esgici@gmail.com>

HMGSQL

SQL Bridge for MySQL, PostgreSQL, SQLite

HMG SQL is a collection of SQL RDBMS bridges for MySQL, PostgreSQL and SQLite database.

The main advantages of HMGSQL are:

1. It eliminates the differences of connection procedures and query execution of each RDBMS package.
2. It reduces change in the HMG codes when we want to switch over from one RDBMS to another RDBMS package.
3. All SQL activities are consolidated to a minimum number of functions which in-turn reduces the number of lines of coding in your programs a lot.

Usage of HMGSQL:

The following are the functions used in all the three libraries (viz., HMGMySQL,HMGPGSQL, HMGSQLite) for the purposes mentioned.

Connect2DB() -> oDBO - This is used to connect to the RDBMS database. This function returns the DataBase Connection Object which can be used in all other SQL activities. Even though the function name is the same, parameters passed to this function differs according to the RDBMS package chosen. This is explained below:

- **HMGMySQL** – Syntax: `Connect2DB(cHost,cUser,cPassword,cDBname)`
  - Example:

    ```
    connect2db('localhost','user1','pass1','productdb')
    ```

- **HMGPGSQL** – Syntax: `Connect2DB(cHost,cUser,cPass,cDb,nPort)` . Default nPort is 5432.
  - Example:

    ```
    connect2db('localhost','user1','pass1','productdb',5432)
    ```

- **HMGSQLite** – Syntax: `Connect2DB(cDBname,lCreate)` .
  - Example:

    ```
    connect2db('c:\mydata\db.sqlite3',.t.)
    ```

SQL(oDBO,cQStr) –> aTable - This is used to execute only the “SELECT” type queries. This function returns a two dimensional array of the query result (even though the resultant number of rows and columns be 1). An empty array will be returned in case there are no rows returned.

Example:

```
aTable := SQL(dbo,”select * from products order by name”)
```

MiscSQL(oDBO,cQStr) -> lOk – This is used to execute all the other miscellaneous commands like “Insert, Delete, Update, Create, Alter, Grant” etc., except “Select”. This function returns a logical value. If the query execution was successful, .t. is returned. Otherwise .f. is returned.

Example:

```
lOk := MiscSQL(dbo,”insert into products values (1,'Product1')”)
```

C2SQL(xValue) -> xValue – This function is used to parse the values to be SQL safe which can be used in any query in-line. This function returns the parsed value of the value passed.

Example:

```
aTable := SQL(dbo,”select * from ledger where date <= “+c2sql(date()))
```

CloseDB(oDBO) -> Nil – This function is used only in the case of PostgreSQL database connection. This is used to close the database connection.

Example:

```
closedb(dbo)
```

Author:

S. Rathinagiri <srathinagiri@gmail.com>

HMG-UNICODE

Unicode Documentation

Since version HMG.3.1.0 (2012\/11\/25), **HMG at the same time supports ANSI and Unicode character set**, or only ANSI character set (for compatibility with previous versions) depending on the choice of compilation in the build of the library. By default HMG supports ANSI and Unicode character set (see INCLUDE\SET_COMPILE_HMG_UNICODE.CH).

Unicode is the current standard in character set, say *Microsoft* in your documentation:

“Unicode is a worldwide character encoding standard that provides a unique number to represent each character used in modern computing, including technical symbols and special characters used in publishing. Unicode is required by modern standards, such as XML and ECMAScript (JavaScript), and is the official mechanism for implementing ISO\/IEC 10646 (UCS: Universal Character Set). It is supported by many operating systems, all modern browsers, and many other products. ***New Windows applications should use Unicode*** *to avoid the inconsistencies of varied code pages and to aid in simplifying localization.”*

Thereby **HMG-Unicode** is the future in the xBase programming for Windows. Since version HMG.3.2.0 (2013\/12\/08), HMG-Unicode is considered stable.

HMG-Unicode required **set 'Encoding in UTF-8' in your text editor** for all the source code files which contain strings in languages using Unicode characters. See Main.UNI.Demo in SAMPLES folder for Unicode characters in Tamil language.

# General Functions\/Commands

- **HMG_SupportUnicode()** - Return .T. or .F.
- Return true only if HMG is compiled for support the ANSI and Unicode character set.
- **HMG_CharSetName()** - Return "UNICODE" --> if HMG is compiled for support the ANSI and Unicode character set.
- Return "ANSI" --> if HMG is compiled for support ONLY the ANSI character set.
- **SET CODEPAGE TO UNICODE** - Sets the character code page to UTF-8 (Unicode).
- If HMG is compiled for support ANSI\/Unicode character set: UTF-8 is default code page.

- **HMG_IsCurrentCodePageUnicode()** - Returns TRUE if current code page is UTF-8.
- **IF HMG SUPPORT UNICODE [RUN | STOP]** --> This is a security command for avoid error in program execution\/compilation for

programmers that used HMG library with and without support for the Unicode character set.

- **RUN** --> Only run the program if HMG library supports the Unicode character set.
- **STOP** --> Stop the execution of the program if HMG library supports the Unicode character set (useful when deactivate the COMPILE_HMG_UNICODE directive to build the HMG library).
- *Note*: The programs written entirely in ANSI can be compiled easily with HMG-UNICODE, adding to the beginning of the

  function MAIN() the appropriate ANSI code page, eg. SET CODEPAGE TO SPANISH, without need to disabling

  the COMPILE_HMG_UNICODE directive (in file INCLUDE\SET_COMPILE_HMG_UNICODE.CH) and rebuild the HMG library.

  The hybrid programs must alternate the appropriate ANSI code page with UTF-8 code page according to the needs.

- Remember: To develop applications that support the ANSI\/UNICODE character set, you should replace in your

programs ALL functions that ONLY support the ANSI character set, by ANSI\/UNICODE equivalent functions.

# Alternative string functions that support ANSI\/Unicode character set

ANSI\/UNICODE ANSI Only

- **HMG_LEN()** <=> LEN()
- **HMG_LOWER()** <=> LOWER()
- **HMG_UPPER()** <=> UPPER()
- **HMG_PADC()** <=> PADC()
- **HMG_PADL()** <=> PADL()
- **HMG_PADR()** <=> PADR()
- **HMG_ISALPHA()** <=> ISALPHA()
- **HMG_ISDIGIT()** <=> ISDIGIT()

- **HMG_ISLOWER()** <=> ISLOWER()
- **HMG_ISUPPER()** <=> ISUPPER()
- **HMG_ISALPHANUMERIC()** <=> RETURN (ISALPHA(c) .OR. ISDIGIT(c))
- (*) **HB_USUBSTR()** <=> SUBSTR()
- (*) **HB_ULEFT()** <=> LEFT()
- (*) **HB_URIGHT()** <=> RIGHT()
- (*) **HB_UAT()** <=> AT()
- (*) **HB_UTF8RAT()** <=> RAT()
- (*) **HB_UTF8STUFF()** <=> STUFF()

  (*) Harbour native functions

# Gets Unicode text value

- **HB_UCODE (** cUnicodeCharacter **)** --> Return nCode
- **HB_UCHAR (** nCode **)** --> Return cUnicodeCharacter
- **HMG_GetUnicodeValue (** cUnicodeText **)** --> Return array { nCode1, nCode2, ..., nCodeN }
- **HMG_GetUnicodeCharacter (** { nCode1, nCode2, ..., nCodeN } **)** --> Return cUnicodeText

# UTF8 functions

- **HMG_IsUTF8 (** cString **)** --> lBoolean
- **HMG_IsUTF8WithBOM (** cString **)** --> lBoolean
- **HMG_UTF8RemoveBOM (** cString **)** --> cString
- **HMG_UTF8InsertBOM (** cString **)** --> cString
- **HMG_UNICODE_TO_ANSI (** cTextUNICODE **)** --> cTextANSI
- **HMG_ANSI_TO_UNICODE (** cTextANSI **)** --> cTextUNICODE

# Unicode functions

- **HMG_StrCmp (** cText1 , cText2 , [lCaseSensitive] **)** --> CmpValue

HARBOUR string functions that support ANSI and UNICODE character set

ANSI\/Unicode(character) and Binary(byte) string functions:

```
HB_UCHAR( <nCode> ) -> <cText> // return string with U+nCode character in HVM CP encod
ing
HB_BCHAR( <nCode> ) -> <cText> // return 1 byte string with <nCode> value
HB_UCODE( <cText> ) -> <nCode> // return unicode value of 1-st character (not byte) in
 given string
HB_BCODE( <cText> ) -> <nCode> // return value of 1-st byte in given string
HB_ULEN( <cText> ) -> <nChars> // return string length in characters
HB_BLEN( <cText> ) -> <nBytes> // return string length in bytes
HB_UPEEK( <cText>, <n> ) -> <nCode> // return unicode value of <n>-th character in giv
en string
HB_BPEEK( <cText>, <n> ) -> <nCode> // return value of <n>-th byte in given string
HB_UPOKE( [@]<cText>, <n>, <nVal> ) -> <cText> // change <n>-th character in given str
ing to unicode <nVal> one and return modified text
HB_BPOKE( [@]<cText>, <n>, <nVal> ) -> <cText> // change <n>-th byte in given string t
o <nVal> and return modified text
HB_USUBSTR( <cString>, <nStart>, <nCount> ) -> <cSubstring>
HB_BSUBSTR( <cString>, <nStart>, <nCount> ) -> <cSubstring>
HB_ULEFT( <cString>, <nCount> ) -> <cSubstring>
HB_BLEFT( <cString>, <nCount> ) -> <cSubstring>
HB_URIGHT( <cString>, <nCount> ) -> <cSubstring>
HB_BRIGHT( <cString>, <nCount> ) -> <cSubstring>
HB_UAT( <cSubString>, <cString>, [<nFrom>], [<nTo>] ) -> <nAt>
HB_BAT( <cSubString>, <cString>, [<nFrom>], [<nTo>] ) -> <nAt>
```

ANSI\/Unicode (character) string functions:

```
HB_TOKENCOUNT()
HB_TOKENGET()
HB_ATOKENS()
HB_TOKENPTR() /* like HB_TOKENGET() but returns next token starting from passed positi
on (0 based) inside string, f.e.:
HB_TOKENPTR( cString, @nTokPos, Chr( 9 ) ) -> cToken */
MEMOREAD()
MEMOWRIT()
HB_MEMOREAD() // not limited to 64 KB as MEMOREAD()
HB_MEMOWRIT() // not limited to 64 KB as MEMOWRIT()
/* warning <nLineLength> is in bytes, <nLineLength> must be greater than the number of
 bytes of the
longest line of text in UTF-8 */
MEMOLINE( <cString>, [ <nLineLength>=79 ], [ <nLineNumber>=1 ], [ <nTabSize>=4 ], [ <l
Wrap>=.T. ], [<cEOL>|<acEOLs> ] ) -> <cLine>
MLCOUNT ( <cString>, [ <nLineLength>=79 ], [ <nTabSize>=4 ], [ <lWrap>=.T. ], [ <cEOL>
|<acEOLs> ] ) -> <nLines>
MLPOS ( <cString>, [ <nLineLength>=79 ], [ <nLineNumber>=1 ], [ <nTabSize>=4 ], [ <lWr
ap>=.T. ], [<cEOL>|<acEOLs> ] ) -> <nLinePos>
// MLCTOPOS() --> NOT support UTF-8
// MPOSTOLC() --> NOT support UTF-8
```

```
MEMOTRAN()
HB_STRDECODESCAPE ( <cEscSeqStr> ) -> <cStr> /* decode string with \ escape sequences
*/
HB_STRCDECODE ( <cStr> [, @<lCont> ] ) -> <cResult> | NIL /* decode string using C com
piler rules */
/* If second parameter <lCont> is passed by reference then it allows to decode multili
ne strings.
In such case <lCont> is set to .T. if string ends with unclosed "" quoting. Function r
eturns decoded string or NIL on syntax error. */
HB_WILDMATCH (cPattern, cValue [, lExact] ) /* compares two strings */
/* Compares cValue with cPattern.
cPattern * may contain wildcard characters (?*)
When lExact is TRUE then it will check if whole cValue is covered by cPattern else it
will check if cPattern is a prefix of cValue */
HB_WILDMATCHI (cPattern, cValue) /* compares two strings */
/* Compares cValue with cPattern
Check if whole cValue is covered by cPattern */
HB_FILEMATCH (cFileName, cPattern)
/* eg. HB_FILEMATCH ("picture.bmp", "*.bmp") ---> return TRUE if file exist */
/* eg. HB_FILEMATCH ("c:\image\picture.bmp", "picture.bmp") ---> return TRUE if file e
xist */
HB_STRTOEXP() /* convert string to valid macrocompiler expression */
STRTRAN()
LTRIM()
RTRIM()
TRIM() /* synonymn for RTRIM */
ALLTRIM()
/* operator $ */
<cSubStr> $ <cStr> /* return TRUE if <cSubStr> is contained in <cStr> */
HB_STRTOUTF8 (<cStr> [, <cCPID> ] ) -> <cUTF8Str>
HB_UTF8TOSTR (<cUTF8Str> [, <cCPID> ] ) -> <cStr>
* <cCPID> is Harbour codepage id, f.e.: "EN", "ES", "ESWIN", "PLISO", "PLMAZ", "PL852"
, "PLWIN", ...
* When not given then default HVM codepage (set by HB_SETCODEPAGE()) is used.
HB_TRANSLATE ( <cSrcText>, [<cPageFrom>], [<cPageTo>] ) --> cDstText /* is used usuall
y to convert between the Dos and the Windows code pages of the same language */
HB_UTF8CHR ()
HB_UTF8ASC ()
HB_UTF8AT ()
HB_UTF8RAT () /* NOTE: In HB_UTF8RAT we are still traversing from left to right, as it
 would be required anyway to determine the real string length */
HB_UTF8SUBSTR ()
HB_UTF8LEFT ()
HB_UTF8RIGHT ()
HB_UTF8PEEK ()
HB_UTF8POKE ()
HB_UTF8STUFF ()
HB_UTF8LEN ()
HB_UTF8STRTRAN() /* equal to STRTRAN() */
REPLICATE() /* returns n copies of given string */
```

Miscellaneous Functions

All functions STRINGS related to DATE and TIME

```
SPACE() /* returns n copies of a single space */
STR()
STRZERO ()
TYPE()
VAL()
HB_VALTOSTR() /* converts any data type to STR Value */
HB_VALTOEXP() /* converts any data type to STR Expression */
VALTYPE()
HB_ISSTRING()
HB_ISCHAR()
HB_ISMEMO()
```

HARBOUR string functions that support Only ANSI character set

All these functions work with CodePages using custom character indexes (ANSI CodePage), do not support UNICODE character set

```
LEN()
AT()
HB_AT()
HB_ATI()
MLCTOPOS()
MPOSTOLC()
PAD() /* synonymn for PADR */
PADC()
PADL()
PADR()
RAT()
HB_RAT()
LEFT()
RIGHT()
LOWER()
UPPER()
STUFF()
SUBSTR()
TRANSFORM()
HB_STRSHRINK()
ISALPHA()
ISDIGIT()
ISLOWER()
ISUPPER()
```

HMG 64-bits Documentation

Since version HMG.3.3.0 (2014\/04\/24) **HMG builds applications in 32 and 64 bits** depending if the choice of compilation is with **build.bat** or **build64.bat** respectively. When you compile with the **IDE** select the option “Build” in main menu.

Guidelines to make compatible code with Win32\/64-bits

# PRG level: at high level is not necessary any changes in your code, all Harbour functions are portables to 64-bits.

# C level: at low level are necessary some changes in your code because HMG stores pointers in integer numbers and the size of the pointers change with the architecture (32 or 64 bits), the other hand some functions of Windows that use HMG are not portables.

a) Values that can take 32 or 64 bits

HMG use many pointers and numbers type, and the size of the pointers and some these data types are of 32 or 64 bits according to the architecture chosen for compiling, for example:

- All Pointers: all GDI handles (HANDLE, HBITMAP, HBRUSH, HCURSOR, HDC, HENHMETAFILE, HFONT, HGDIOBJ, HGLOBAL, HICON, HINSTANCE, HKEY, HMENU, HMODULE, HPEN, HRGN, HWND, etc.), void*, int*, NULL, PVOID, LPVOID, PINT, LPINT, etc.
- Some data types such as: WPARAM, LPARAM, LRESULT, LONG_PTR, LONG_INT, size_t, time_t, etc.

b) Deprecated (obsolete) functions

Some API-functions are not portable to 64-bits and are considered obsolete such as **SetWindowLong**, **GetWindowLong**, **SetClassLong**, **GetClassLong**, etc. If you need these functions in your code, use the new equivalent functions (this list is not exhaustive): **SetWindowLongPtr, GetWindowLongPtr, SetClassLongPtr, GetClassLongPtr**, etc.

c) Create portable code

For example, in 32-bits is created a high level version of the function SendMessage() as:

```
HB_FUNC ( SENDMESSAGE )
{
   HWND hWnd = (HWND) hb_parnl (1);
   UINT Msg = (UINT) hb_parnl (2);
   WPARAM wParam = (WPARAM) hb_parnl (3);
   LPARAM lParam = (LPARAM) hb_parnl (4);
   LRESULT ret;
   ret = SendMessage (hWnd, Msg, wParam, lParam);
   hb_retnl ((LONG) ret);
}
```

But this code is not portable to 64-bits because:

1) HWND, WPARAM and LPARAM are 64-bit values and is assigned 32-bit values with hb_parnl()

2) is returned a LONG number (32-bit value) whereas LRESULT is a 64-bit value

If you desire write **portable codes in HMG** you must write as:

```
HB_FUNC ( SENDMESSAGE )
{
   HWND hWnd = (HWND) HMG_parnl (1);
   UINT Msg = (UINT) hb_parnl (2);
   WPARAM wParam = (WPARAM) HMG_parnl (3);
   LPARAM lParam = (LPARAM) HMG_parnl (4);
   LRESULT ret;
   ret = SendMessage (hWnd, Msg, wParam, lParam);
   HMG_retnl ((LONG_PTR) ret);
}
```

HMG_parnl, HMG_retnl and LONG_PTR take a 32 or 64 bit value according to the chosen architecture.

HMG defines a series of macros (INCLUDE\HMG_UNICODE.h) that take a value of 32 or 64 bit according to the architecture chosen for compiling:

- HMG_parnl

- HMG_parvnl

- HMG_retnl

- HMG_retnllen

- HMG_stornl

- HMG_storvnl

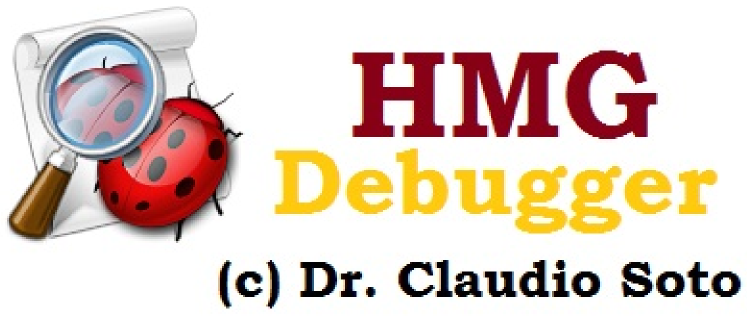
- HMG_arraySetNL

- HMG_arrayGetNL

Note: remember when you pass a value to these functions you should use **LONG_PTR or INT_PTR** for type conversion instead of LONG or INT. When the parameters or values returned are of 32-bits you continue using hb_parnl, hb_parni, hb_retnl, hb_retni, etc. and continue the conversion of type with LONG or INT as usually.

	WIN32	**WIN64**
char	8	8
short	16	16
int	32	32
long	32	32
long long	64	64
size_t	**32**	**64**
pointer	**32**	**64**
Others(*)	**32**	**64**
(*) LONG_PTR, INT_PTR, WPARAM, LPARAM, LRESULT, etc.		

HMG Debugger



srvet@adinet.com.uy

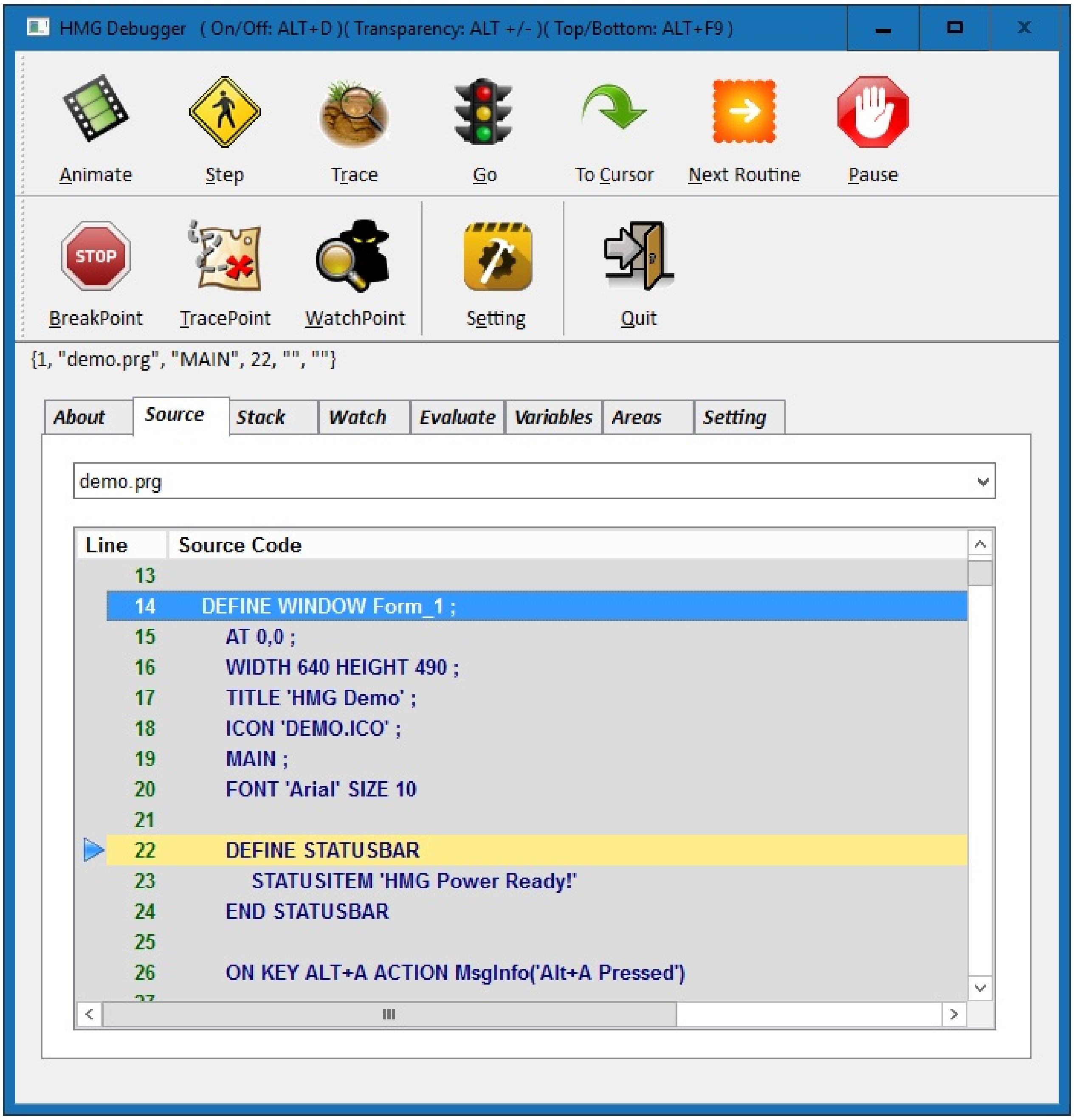
http:\/\/srvet.blogspot.com

Uruguay, September 2015

HMG Debugger is an embedded native GUI debugger in the HMG library that allows you to debug your source code of an application while your executable file is running. **HMG debugger** is invoked when compiled the application with the **\/d** option on the command line or with the option **Project->Debug** of the main menu of the IDE, eg.

Build.bat \/d <program.prg> | <project.hbp>

The debugger is activated automatically at startup the application.



{1, "demo.prg", "MAIN", 22, "", ""}

```
13
14    DEFINE WINDOW Form_1 ;
15        AT 0,0 ;
16        WIDTH 640 HEIGHT 490 ;
17        TITLE 'HMG Demo' ;
18        ICON 'DEMO.ICO' ;
19        MAIN ;
20        FONT 'Arial' SIZE 10
21
22        DEFINE STATUSBAR
23            STATUSITEM 'HMG Power Ready!'
24        END STATUSBAR
25
26        ON KEY ALT+A ACTION MsgInfo('Alt+A Pressed')
```

The debugger provides you with certain features that make it easier to isolate and identify errors. Among these features are the ability to:

- Trace source code line-by-line (single step mode)
- Watch a variable as it is updated (watchpoint)
- Monitor variables by storage class
- Inspect work areas and set values
- Change the value of a variable
- Execute procedures and user-defined functions linked into you application
- Stop program execution when a variable changes value (tracepoint)

- Stop program execution at a line of code or when a function is called (breakpoint)

Note that Harbour allows you place more than one program statement on a single line separated by semicolon. For example:

lNewPage := (nLineNo > 55); ReportPage (lNewPage)

In this example, the debugger does not allow you to step through the first and second statements independently nor does it allow you to set a breakpoint at either statement. The entire line is treated as single entity. The code in the above example should be broken up into two lines, as follows:

lNewPage := (nLineNo > 55)

ReportPage (lNewPage)

This make the debugging easier and also makes the code more readable.

1) Global Hotkey:

CTRL+H: hotkeys help.

ALT+D: enable\/disable debugger while the application is running. When the debugger is disabled, all breakpoints and tracepoints are ignored.

ALT +\/-: increment\/decrement the transparency of windows debugger.

ALT+F9: sets window debugger TopMost\/Bottom.

ALT+M: release unused memory (leak memory).

F11: switches between ToolBar and Main Menu.

ALT+X: exit the program.

Hotkeys for debugger commands:

F3: Animate

F8: Step

F10: Trace

F5: Go

F7: To Cursor

Ctrl+F5: Next Routine

Ctrl+F3: Pause

F9: BreakPoint

2) Run Mode:

The debugger provides several different ways to execute a program, called execution program.

Animate: Runs line-by-line automatically with a delay that is specified in the Setting Tab.

Go: Is very fast, runs the application in the same manner as it would in Animate mode with the difference that not send information to debugger until one finds a Breakpoint or Tracepoint.

To Cursor: Runs application in the same manner as Go mode but until the code line of the current cursor position.

Next Routine: Runs application in the same manner as Go mode but until start the next activation (that is, procedure, function, code block, or message send).

Step: Runs application line-by-line, it allows you to execute one line of program code at a time.

Trace: Runs the application in the same manner as it would in Step mode with the difference that Trace mode does not display the code for functions and procedures called by the current program nor does it trace codeblocks back to their definition. In the Trace mode the debugger does not display the code for called routines.

3) Point Mode:

Breakpoint: A breakpoint is a line of program code or a function or procedure call that, when encountered, causes the debugger to stops the execution of the program. In other words, a breakpoint defines a physical breaking point in a program. To set a breakpoint in the current program, move the cursor to the appropriate line in the Source code Tab and press Breakpoint Toolbar Button, the selected line change the color. To define a function or procedure as a breakpoint use the following method in the Evaluate Tab: HMG_Debugger():BreakPointAddFunc(“MyFuncName”). For delete a breakpoint first move the cursor to the appropriate line in the Source code Tab and press Breakpoint Toolbar Button, when a breakpoint is deleted, the line of code returns to its normal color.

Tracepoint: A tracepoint is an expression whose current value is displayed in the Watch monitor. The value of each tracepoint is updated as your application executes; however, whenever its value changes, program execution stops and control passes to the debugger. In this respect, a tracepoint is similar to a breakpoint, e.g. Sales->SaleAmount >= 10

Watchpoint: A watchpoint is an expression whose current value is displayed in the Watch monitor. The value of each watchpoint is updated as your application executes, this allows you monitoring the value of a variable or expression, e.g. FOUND(). The difference between watchpoint and tracepoint, is that watchpoint not stops the execution of the program when its value changes.

4) Tabs:

About: display the logo of debugger and information about of the author.

Source code: In this monitor is displayed the source code of the application. The debugger searches for source code files in the current directory, if not found the debugger searches in the PATH system. Only after these locations are exhausted will an error message be displayed indicating that the file could not be found. Inside the Source code window there is a highlight bar with an arrow called the execution bar which is positioned on the line of code about to be executed. The execution bar moves as execution continues. When the Source code is active, the cursor (a highlight row) appears in the grid to indicate your current position in the file being viewed. The cursor can be moved up and down using the direction keys.

Stack: The call Stack monitor contains the list of all pending activations, including procedures, functions, code blocks, and messages sends. The current activation (the current proc. level) is always at the top of the call stack. If you press Return in the highlighted proc. level the Source code monitor reflect the correspondent line of code.

Watch: This monitor displayed the number, type, expression and value of each tracepoint and watchpoint. The tracepoints and watchpoints are distinguished using the abbreviations “tp” and “wp”. Tracepoints and watchpoints are created using Tracepoint and Watchpoint Toolbar Buttons respectively. To remove or edit a point definition press key Delete or Return respectively when the point is highlighted in the grid watch.

Evaluate: This monitor allows you view and\/or change the value of a variable (e.g. cMsg:= cMsg + “Hello”), evaluate an expression (e.g. EOF()==.F. .AND. nCount == 10), execute a function or procedure (e.g. ATAIL(aArray)) or create a new variable (e.g. __MVPUBLIC(“MyVarName”)).

Variables: This monitor lets you control the display of public, private, static, and local variables, press key Return for inspect the contents of array, hash or object selected. The name of highlighted variable is automatically copied into the Evaluate monitor for facilitate the view or edit the value.

Areas: The work Areas monitor allows you to view database and other work area information. This monitor is divided into two panes. The first pane display the alias name for each open database, information regarding the currently file highlighted shown in the other pane.

Setting: Each time the debugger is invoked, it automatically searches for a script file with the name Init.dbg. If a file by this name is located in the current directory the debugger executes it as script file. This Tab allows you to load and save script files and debugger settings.

By default, option Trace Code Blocks is on, if you switch off code block tracing is disabled. If trace code block is enabled, when a code block is evaluated when executing a program the debugger moves the execution bar to the line of code where the block was created. This allows you to see the contents of the block (which is unavailable during normal inspection), and occurs regardless of whether the block was declared in the current routine.

HMG Classes

HMG_TString Class

This class selects automatically the correct ANSI or Unicode string function depending of current code page, the use of this class create a more portable your code, and is useful for a ***defensive programming*** (minimization of input errors)

Methods:

```
New() // Constructor
Chr( nCode ) // return string with nCode Unicode/ANSI character
Char( nCode ) // return string with nCode Unicode/ANSI character
Asc( cString ) // return Unicode/ANSI value of 1-st character (not byte) in given stri
ng
Code( cString ) // return Unicode/ANSI value of 1-st character (not byte) in given str
ing
Len( cString ) // return string length in characters
ByteLen( cString ) // return string length in bytes
Peek( cString, n ) // return unicode value of <n>-th character in given string
Poke( cString, n, nVal ) // change <n>-th character in given string to unicode <nVal>
one and return modified text
SubStr( cString, nStart, nCount )
Left( cString, nCount )
Right( cString, nCount )
At( cSubString, cString, nFrom, nTo )
Rat( cSearch, cTarget )
Stuff( cString, nStart, nDelete, cInsert )
LTrim( cString )
RTrim( cString )
AllTrim( cString )
StrTran( cString, cSearch, cReplace, nStart, nCount )
Replicate( cString, nCount )
Space( nCount )
StrToUTF8( cStr, cCPID )
UTF8ToStr( cUTF8Str, cCPID )
IsUTF8( cString )
EOL() // CR+LF
OsNewLine() // CR+LF
Lower( cString )
Upper( cString )
PadC( cString, nLength, cFillChar )
PadL( cString, nLength, cFillChar )
PadR( cString, nLength, cFillChar )
IsAlpha( cString )
IsDigit( cString )
IsLower( cString )
IsUpper( cString )
IsAlphaNumeric( cString ) // return (ISALPHA(c) .OR. ISDIGIT(c))
StrCmp( cString1, cString2, lCaseSensitive ) // return -1 (minor), 0 (equal) , +1 (gre
at)
```

Sample:

For default HMG create an instance of this class, to access this instance with pseudovariable **oString**, e.g.

```
oString:Len( cString )
oString:Upper( cString )
```

- You create a new instance of this class with pseudofunction:

```
oMyStr := HMG_TStringNew() // this pseudofunction call the constructor method HMG_TStr
ing():New()
```

and use, e.g.

```
oMyStr:Len ( cString )
oMyStr:Upper( cString )
```