FlagShip

Object Oriented Database Development System

Cross-Compatible to Unix, Linux and MS-Windows

MULTISOFT

Release 8.1

Section OBJ

The whole FlagShip 8 manual consist of following sections:

Section	Content
GEN	General information: License agreement & warranty, installation and de-installation, registration and support
LNG	FlagShip language: Specification, database, files, language elements, multiuser, multitasking, FlagShip extensions and differences
FSC	Compiler & Tools: Compiling, linking, libraries, make, run-time requirements, debugging, tools and utilities
CMD	Commands and statements: Alphabetical reference of FlagShip commands, declarators and statements
FUN	Standard functions: Alphabetical reference of FlagShip functions
OBJ	Objects and classes: Standard classes for Get, Tbrowse, Error, Application, GUI, as well as other standard classes
RDD	Replaceable Database Drivers
EXT	C-API: FlagShip connection to the C language, Extend C System, Inline C programs, Open C API, Modifying the intermediate C code
FS2	Alphabetical reference of FS2 Toolbox functions
QRF	Quick reference: Overview of commands, functions and environment
PRE	Preprocessor, includes, directives
SYS	System info, porting: System differences to DOS, porting hints, data transfer, terminals and mapping, distributable files
REL	Release notes: Operating system dependent information, predefined terminals
APP	Appendix: Inkey values, control keys, ASCII-ISO table, error codes, dBase and FoxPro notes, forms
IDX	Index of all sections
fsman	The on-line manual “f**sman**” contains all above sections, search function, and additionally last changes and extensions

multisoft Datentechnik, Germany

Object Oriented Database Development System,
Cross-Compatible to Unix, Linux and MS-Windows

Section OBJ

Manual release: 8.1

For the current program release see your Activation Card,
or check on-line by issuing *FlagShip -version*

Note: the on-line manual is updated more frequently.

Copyright

Made in Germany. Printed in Germany.

Trademarks

Headquarter Address

multisoft Datentechnik
Schönaustr. 7
84036 Landshut
Germany

Phone: *(*+49) 0871-3300237

E-mail: support@flagship.de
support@multisoft.de
sales@multisoft.de

Web: http://www.fship.com

OBJ: Objects and Classes

1. Overview

FlagShip fully supports Clipper's and VO's implementation of OOP (object oriented programming) classes. FlagShip also provides facilities for defining and manipulating user-defined objects as a data type. In this section, only the predefined (standard) classes (and their compatibility to CA-Clipper and VisualObjects) are described. See section LNG.11 for a general description of OOP (object oriented programming).

The Visual FlagShip (FlagShip release 5 and later) is heavily based on OOP classes. In fact there are three different classes in the FlagShip library for each specific i/o operation. The decision which class should be taken is done either by the compiler when the -io=g/t/b switch was used, or at run-time from the system environment or via command-line switch. The run-time setup is available in the source <FlagShip_dir>/system/initio.prg. See also section LNG.1.2

1.1 Objects

Objects in FlagShip are complex data structures with predefined instance variables and methods to access them. The **object variable** has some similarity to an array variable, whereby the object elements contains both data and code. The data element is named **instance**, and the code element is a **method**.

The objects themselves are passive. They never initiate an action, process a user keystroke or overtake program control. Instead, the application controls the action by sending messages to the object, usually by assigning values to object instances or invoking a method function.

Since the objects are stored in regular FlagShip variables, there may coexist as many objects simultaneously, as required. The objects have the same life time, as the variable scope storing the object.

1.2 Classes

A class is a declaration of the object structure. It contains encapsulated data and code from the rest of your application. You may establish a tree- like hierarchy among the classes by using inheritance, see also section LNG.2.11 and CMD.CLASS.

There are many standard, predefined classes of objects, many of them are backward compatible to Clipper 5.x or VO:

Class	Used for	Backward compatible to
Applic }	Application class, instantiated in initio.prg	
AppWindow }		
CheckBox	Check box widget handling @..GET CHECKBOX	CL5.3
Color	Color basic class	
ColorPair	Color basic class	
ComboBox	Combo box widget handling @..GET COMBOBOX, LISTBOX	
DbServer	Database RDD	CL5.3
Error	Information on run-time errors	FS4,CL5.2
ErrorBox	Error box widget handling Alert()	
Font	Font basic class handling SET FONT	
Get	GET/READ system, user modifiable, handling @..GET	FS4,CL5.2
InfoBox	Info box widget handling InfoBox()	
ListBox	List box widget handling Achoice(), @..GET LISTBOX	CL5.3,VO
MenuItem	Subclass of menu structure	CL5.3
MessageBox	Message box widget	
Mouse	Mouse basic class handling Mcol(), Mrow()	
Point	Point basic class	VO
PopUp	Subclass of menu structure	CL5.3
Printer	Class handling printers	
Prompt	Class handling @..PROMPT, MENU TO	
PushButton	Push button widget handling @..GET PUSHBUTTON	CL5.3,VO
RadioButton	Radio button widget, handling @..GET RADIOBUTTON	CL5.3,VO
RadioGroup	Radio group widget, handling @..GET RADIOGROUP	CL5.3,VO
Rectangle	Rectangle basic class	VO
Size	Size basic class	VO
StatusBar	Status bar class handling StatBarMsg(), StatusMessage()	
TBColumn	Column definitions for TBROWSE	FS4,CL5.2
TBrowse	Browsing table-oriented data	FS4,CL5.2

TextBox	Text box widget	
TopBar	Subclass of menu structure	CL5.3
WarningBox	Warning box widget	

In addition to the above generic and i/o classes, FlagShip provides you with predefined database server classes (RDD), generally compatible to CA/VO (VisualObjects):

Class, RDD	Used for	Supports	Note
DataServer	inherited by other RDDs	RDD specific files	
DbServer	standard FlagShip's RDD	.dbf, .dbt, .idx	
DbfIdx	standard FlagShip's RDD	.dbf, .dbt, .idx	
AsciRdd	RDD for ASCII files	text file, import	
Cb4cdx	RDD for FoxBase, FoxPro	.dbf, .fpt, .cdx	*
Cb4ntx	RDD for Clipper	.dbf, .dbt, .ntx	*
Cb4ndx	RDD for dBASE III	.dbf, .dbt, .ndx	*
Cb4mdx	RDD for dBASE IV	.dbf, .dbt, .mdx	*

* Note: The additional replaceable database drivers (RDD) comprise an interface for other Xbase systems on heterogeneous networks. It is available for experimental purposes in source code in <FlagShip_dir>/ system/RDDcb4.tar.Z, and additionally require the CodeBase package. You may acquire Multiplatform CodeBase from Sequiter Software Inc, or other sources.

1.3 FlagShip Classes Sorted By Groups

Application class

Applic	}	Application class, instantiated in initio.prg
AppWindow	}	

Basic classes

Color	Color basic class
ColorPair	Color basic class
Dimension	Dimension basic class
Font	Font basic class handling SET FONT
Mouse	Mouse basic class handling Mcol(), Mrow()
Point	Point basic class
Rectangle	Rectangle basic class
Size	Size basic class

Error class

Error	Information on run-time errors

Basic input/output

CheckBox	Check box widget handling @..GET CHECKBOX
PushButton	Push button widget handling @..GET OUSHBUTTON
RadioButton	Radio button widget handling @..GET RADIOBUTTON

RadioGroup	Radio group widget handling @..GET RADIOGROUP
RadioButton	Radio button widget handling @..GET RADIOBUTTON
RadioGroup	Radio group widget handling @..GET RADIOGROUP

Extended input/output

Get	GET/READ system, user modifiable, handling @..GET
Prompt	Class handling @..PROMPT, MENU TO
TBrowse	Browsing table-oriented data
TBColumn	Column definitions for TBROWSE

Select widgets

ComboBox	Combo box widget handling @..GET COMBOBOX, LISTBOX
ListBox	List box widget handling Achoice(), @..GET LISTBOX

Pop-up messages

ErrorBox	Error box widget handling Alert()
InfoBox	Info box widget handling InfoBox()
MessageBox	Message box widget
TextBox	Text box widget
WarningBox	Warning box widget

Menu bar, Status bar

TopBar	Subclass of menu structure
PopUp	Subclass of menu structure
MenuItem	Subclass of menu structure
StatusBar	Status bar class handling StatBarMsg(), StatusMessage()

Printer

Printer	Class handling printers

Database drivers

AsciRdd	RDD for ASCII files
DataServer	inherited by other RDDs
DbServer	standard FlagShip's RDD
DbfIdx	standard FlagShip's RDD
Cb4cdx	RDD for FoxBase, FoxPro
Cb4ntx	RDD for Clipper
Cb4ndx	RDD for dBASE III
Cb4mdx	RDD for dBASE IV

1.4 FlagShip extensions

FlagShip provides you also with the facility for

- defining your own classes,
- manipulating the predefined classes by inheriting it into own class, and
- handling the user defined objects as a usual data type,

known as OOP (object oriented programming).

To create an object, or instantiate a class, you name the class followed by the instantiation operators in braces { } , or alternatively invoke the (predefined or by compiler automatically generated) creator function named className**NEW** followed by parameters in parentheses ( ) :

```
<oVar> := <className> { }                          -or-
<oVar> := <className> {<argumentList>}             -or-
<oVar> := <className>New ( )                       -or-
<oVar> := <className>New (<argumentList>)
```

The use of the instantiation operator { } is compatible to CA/VO, whereas creating the object via the <className>NEW( ) function is compatible to CA/Clipper 5.x.

When executing the creator function, or instantiating a class, the required amount of space is allocated and assigned to a FlagShip variable, called an object variable. Thereafter, the class and its objects are addressed by using the object variable, a send operator ":" and the associated method or instance variable name, called a selector. See detailed description in section LNG.2.11 and CMD.CLASS.

1.5 Instance Variables

Each instance variable has a defined place in the object structure and holds the internal object data. The name of the instance variable is used to access its contents by means of the "send" operator. The use of the instance variable is similar to a predefined element number of an array when using a #define manifest.

At object creation time, the instance variables are predefined to default values (mostly NIL, see description of each object).

There are several visibility modes of instances, specified during class declaration. Since this section describes the **predefined classes** only, following references are given for the visible, external instances only.

1.6 Methods

Methods are predefined functions to perform an action on the object. They too are accessed by name via the send operator, and executed using the optionally given arguments. **Access** and **Assign** are special cases of methods, which refer to non-exported instances, mostly used as read-only and/or write-only "virtual instance variable" with optional validation checking. They are accessed in the same way, as the instance variables.

1.7 Notation

Each instance variable or object method is referenced using the object (variable), a send operator, and the selector, which specifies a predefined name of the instance variable or a method. The capitalization (upper/lower case) is not significant.

Send operator: The ":" operator sends or receives messages to and from a selector of the specified object. Such messages access a variable or perform a special object action. The general syntax is

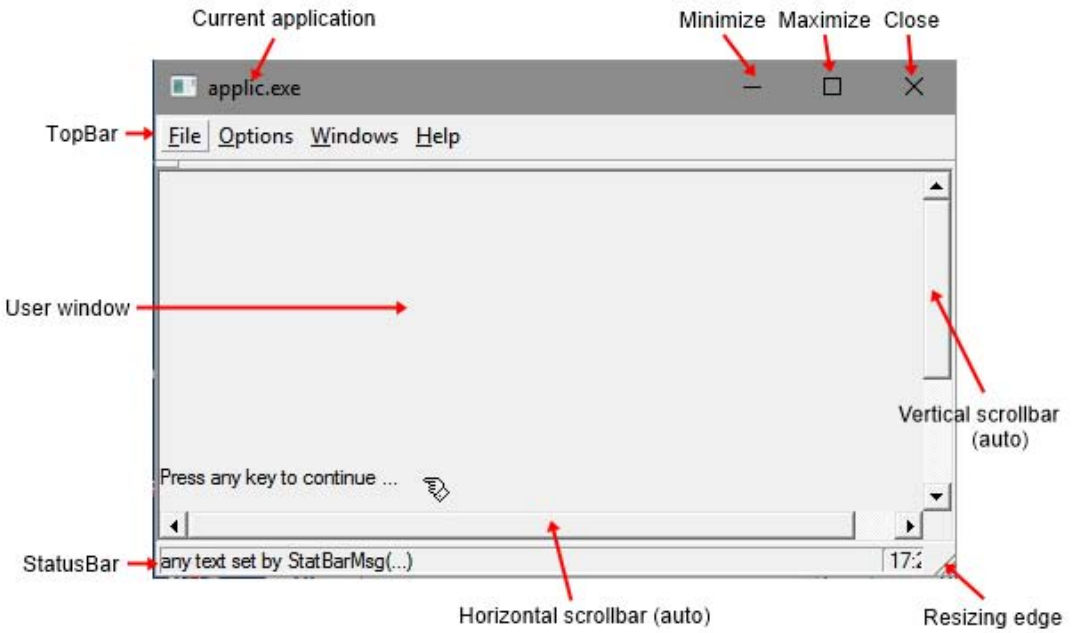
```
<object>:<assignable instance>  :=  <expression>

[<value>  := ]  <object>:<instance>

[<value>  := ]  <object>:<method> ([<argumentList>])
```

FlagShip checks the availability of the instance variables or methods both at compile-time as well as at run-time. The compile-time check is possible only if class prototyping is used (specified e.g. in the stdclass.fh file, which may be invoked from std.fh). Refer also to sections LNG.2.11.1 and CMD.PROTOTYPE. If the instance or method is unknown at compile-time, the slower run-time addressing is used. If the instance or method is unknown, NoiVarGet(), NoiVarPut() or NoMethod() will be invoked if available, or otherwise a run-time error occur.

Naming convention: the names of instances (and access/assign methods) are significant up to the first 10 characters, names of methods are significant in the full length. The capitalization is not significant. See also LNG.2.11.4 for details.

Application Class

Definitions



Application Basic Class

This class provides the basic application functionality and is supported mainly for VO compatibility, as a superset of the App class. In FlagShip, you may instantiate _gAppWindow (or _tAppWindow or _bAppWindow) directly instead, see also <FlagShip_dir>/system/initio.prg.

There are three different application classes: _gApp for GUI (graphical i/o), _tApp for terminal (curses oriented) i/o and _bApp for basic i/o. Before using any of the GUI classes, you need to instantiate the application by _gApp{} or _gAppWindow{} first.

Note, you should use this App class in special cases only, since it will not initialize the GUI window at all and allows basic i/o only. In the most cases, you will need additionally (or only) instantiate the AppWindow class as well.

Application Basic Class Index

Class App = _gApp, _tApp, _bApp
Inherits from: no ancestor
Inherited by: AppWindow
Class prototype: appclass.fh

AppType	ACC	Returns the type of the application
Col2Pixel()	METH	Re-calculate column coordinates to pixels
ColSize	ACC/ASS	Returns or set the pixel size of one column
ColSizeDef()	METH	calculates the def size of 1 column in pixel
DesktopHeight	ACC	Returns the height of the used desktop in pixel
DesktopHeight()	METH	Set/get the height of the used desktop in units
DesktopWidth	ACC	Returns the width of the used desktop in pixel
DesktopWidth()	METH	Set/get the width of the used desktop in units
DesktopSize()	METH	Returns the height and width of desktop
DesktopXDpi	ACC	Returns desktop horiz. DPI (dot/pixel per inch)
DesktopYDpi	ACC	Returns desktop vertical DPI (dot/pixel per inch)
DesktopSizeAvail()	METH	Returns the height and width of available desktop
Font	ACC/ASS	Returns or set the default application font obj
FontWindow	ACC/ASS	Returns or set the default window font object
Init()	Creator	For internal purposes only
Pixel2Col()	METH	Re-calculate given pixels to column coordinates
Pixel2Row()	METH	Re-calculate given pixels to row coordinates
PrgArgs()	METH	Returns an array containing the given arguments
Row2Pixel()	METH	Re-calculate given row coordinates to pixels
RowSize	ACC/ASS	Returns or set pixel size (height) of one row
RowSizeDef()	METH	calculates the default size of 1 row in pixel

Application Basic Class Properties

_gApp { } –> oApp ***CREATOR***
_bApp { } –> oApp ***CREATOR***
_tApp { } –> oApp ***CREATOR***

_gAppNew ( ) –> oApp ***CREATOR, alternative syntax***
_bAppNew ( ) –> oApp ***CREATOR, alternative syntax***
_tAppNew ( ) –> oApp ***CREATOR, alternative syntax***

Instantiates the basic GUI application functionality. Note, you may instantiate an application only once, so use either _gApp{} for GUI or _bApp{} for basic i/o or _tApp{} for terminal i/o, you may determine the currently used environment via the standard IsGuiMode() function. See example in AppWindow class.

oApp:AppType —> cType *ACCESS*

Returns or type of this application object: G = GUI, B = basic, T=terminal i/o, or "-" on error.

oApp:PrgArgs( ) —> aArgs

Returns an array containing the name of the executable and the given arguments/parameters at startup. The command line parameters are splitted in elements of the array, all of type character. If no additional parameters were given, the array contains one element with the name of the executable only. To determine the number of given parameters, use: len(oApp:PrgArgs())-1, see also example in AppWindow class.

oApp:ColSize —> iSize *ACCESS*
oApp:ColSize := iSize *ASSIGN*

Returns or set the pixel size (width) of one column. This value is automatically calculated at the time of gAppWindow instantiation or reset when a new font is assigned. To be able to handle the whole character set size, the value is set to the maximal char width of the current font, see also gFont:Width() for details. You may re- calculate the required char size by e.g.

```
oApp:Font    := gFont { "Times", 12 }
? "current default column size = ", oApp:ColSize
oApp:ColSize := oApp:Font:WidthMaxChar("45890_AEGMQSTWXZ")
?? " using = ", oApp:ColSize
```

which sets the default font to "Times" and its largest alphanumeric character width to be used as an average column size, instead if the largest font character used per default, which mostly is larger. See also examples in the AppWindow and Font class.

oApp:DesktopHeight —> iVertPixelSize *ACCESS*

Returns the height (vertical size) of the used desktop in pixel, i.e. 1200 if the current display mode is 1600x1200. This is a shortcut for the equivalent iVertPixelSize := DesktopHeight(UNIT_PIXEL)

oApp:DesktopHeight([unit],[userSize]) —> nSize

Returns the height (vertical size) of the used desktop size in units, and/or sets the user specified desktop height.

<**unit**> is either numeric value (UNIT_ROWCOL, UNIT_PIXEL, UNIT_MM, UNIT_CM, UNIT_INCH) or logical (.T. = pixel, .F. = rows). If not specified, current SET COORD UNIT is used (default is UNIT_ROWCOL).

<**userSize**> is optional new user-defined value in <units>. You may set different desktop height values for rows, pixel and mm. The cm and inch values are related to mm and supported for your convenience.

Returns : current desktop height in <units> (before new setting, if <userSize> was specified). Zero or -1 is returned on error.

The desktop height in pixel, mm and rows are used also internally for conversion between different units (pixels, rows, mm, cm and inches). These values are determined by system call, but in some operating systems or virtual machines, some of these values may be inaccurate or not available at all (the returned value is <= 0). If required, you may fix/set it manually by corresponding assignment of <userSize>. The new value(s) is/are then taken for the returned value and corresponding unit conversion. Zero or negative <userSize> value disables the user setting and triggers again system call at next invocation.

Valid only for GUI environment, i.e. when the Xserver (on Unix) is already running, or when the application is started in MS-Windows environment.

oApp:DesktopWidth —> iHorizPixelSize *ACCESS*

Returns the width (horizontal size) of the used desktop in pixel, i.e. 1600 if the current display mode is 1600x1200 Valid only for GUI environment, i.e. when the Xserver (on Unix) is running or when invoked in MS-Windows environment.

oApp:DesktopSize([lPixel]) —> aRowCol

Returns an array with two numeric values containing the height and width of desktop. If <lPixel> if true(.T.), the returned row and column data are in pixel, i.e. in the most cases equivalent to oApp:DesktopHeight and :DesktopWidth. If <lPixel> is false (.F.), the data are in row/col coordinates, otherwise the current SET PIXEL is used.

oApp:DesktopSizeAvail([lPixel]) —> aRowCol

Returns an array with two numeric values containing the height and width of the available desktop size. In some operating systems, where determinable, this size may be smaller than :DesktopSize() considering the reduced physical desktop size by the window taskbar etc. If <lPixel> is true(.T.), the returned row and column data are in pixel, if <lPixel> is false (.F.), the data are in row/col coordinates, otherwise the current SET PIXEL is used.

oApp:DesktopXDpi —> iPixel *ACCESS*

Returns the horizontal resolution of the desktop device, in dots per inch (in fact, in pixel per inch), that is used when computing font sizes width and for recalculation of pixel coordinates to LPI for printer output. Apply for GUI mode only. For terminal and basic i/o, the returned value is 0.

oApp:DesktopYDpi —> iPixel *ACCESS*

Returns the vertical resolution of the desktop device, in dots per inch (in fact, in pixel per inch), that is used when computing font sizes height and for recalculation of pixel coordinates to LPI for printer output. Apply for GUI mode only. For terminal and basic i/o, the returned value is 0.

oApp:DesktopXmm —> nSizeXmm *ACCESS*

Returns the horizontal size of the desktop device in mm. Apply for GUI mode only. For terminal and basic i/o, the returned value is 0. Note that this value is returned from the system API self, and may be inaccurate on some systems, or with generic desktop driver, or with VM (virtual machine).

oApp:DesktopYmm —> nSizeYmm *ACCESS*

Returns the vertical size of the desktop device in mm. Apply for GUI mode only. For terminal and basic i/o, the returned value is 0. Note that this value is returned from the system API self, and may be inaccurate on some systems, or with generic desktop driver, or with VM (virtual machine).

oApp:Font —> oFont *ACCESS*
oApp:Font := oFont *ASSIGN*

Returns or set the default application font object. See details in the gFont class description. When a new font is assigned, the ColSize and RowSize data are re-calculated, but not the window size which may be set by oAppWindow:DefSizes() method thereafter.

oApp:FontWindow —> oFont *ACCESS*
oApp:FontWindow := oFont *ASSIGN*

Returns or set the default application font object, used mainly for window frames and for message boxes (like Achoice(), InfoBox() and other widgets) without specified font.

oApp:RowSize —> iSize *ACCESS*
oApp:RowSize := iSize *ASSIGN*

Returns or set the pixel size (height) of one row. This value is automatically calculated at the time of gAppWindow instantiation or reset when a new font is assigned corresponding to the gFont:LineHeight, see details in the Font class description.

oApp:col2pixel(expN1) —> iColPix
oApp:row2pixel(expN1) —> iRowPix

These methods are for your convenience and are equivalent to standard functions Col2pixel(9 and Row2pixel(). They re-calculate the given row/column coordinates (also a fraction of) to pixels, independent on the SET PIXEL setting. Argument:

<**expN1**> the column or row coordinates respectively which should be converted to pixels.

Returns : the <expN1> multiplied by oAppWindow:ColSize or oAppWindow: RowSize and rounded to integer.

oApp:pixel2col(expN1) —> nCol
oApp:pixel2row(expN1) —> nRow

These methods are for your convenience and are equivalent to standard functions Pixel2col() and Pixel2row(). They re-calculate the given pixels to row/column coordinates, independent on the SET PIXEL setting. Argument:

<**expN1**> the pixel value which should be converted to column or row respectively.

Returns : the <expN1> divided by oAppWindow:ColSize or oAppWindow: RowSize and rounded to three decimal places.

Application Window Class

The Application Window is usually the top-most class in FlagShip and defines the main window of the application. It is per default instantiated automatically, and assigned to global constants **oApplic** and **oAppWindow**, see the user modifiable functions _<g|b|t>InitIo() in <FlagShip_dir>/system/initio.prg. You may freely use (access) both of them in your application.

Compatibility: The AppWindow class is a superset of AppWindow and TopAppWindow classes of VO. As opposite to VO where the AppWindow is a virtual class and the TopAppWindow inherits it, is the AppWindow in FlagShip a real class. The TopAppWindow is supported in FlagShip too, but for compatibility purposes only.

Application Window Class Index

Class AppWindow = _gAppWindow, _gAppWindow, _gAppWindow

Inherits from:	App
Inherited by:	TopAppWindow
Class prototype:	appclass.fh
Constants, manifests:	applic.fh

ApplicFont	ACC	Default window font object
AppType	ACC	Returns the type of the application
Attrib	ACC/ASS	Returns or sets the mode of the applic window
Caption	ACC/ASS	Returns/set the text displayed in the title bar
ColorBackground	ACC/ASS	Returns/set the standard window background color
ColorRgbBackground	ACC/ASS	Returns/set the standard window background color
ColSize	ACC/ASS	Returns or set the pixel size of one column
Col2Pixel()	METH	Re-calculate column coordinates to pixels
CurrSize()	METH	Returns the current application window sizes
CurrWinID()	METH	Returns current screenID
DefSize()	METH	Sets/returns the default applic window sizes
DesktopHeight	ACC	Returns the height of the used desktop in pixel
DesktopWidth	ACC	Returns the width of the used desktop in pixel
Display()	METH	(Re-) Displays the application window
EnableHorizontalScroll()	METH	Enable/disable horizontal scroll bar
EnableVerticalScroll()	METH	Enable/disable vertical scroll bar
ErrorMessage()	METH	Display error box, equivalent to ALERT()
Font	ACC/ASS	Returns/set the default application font object
Font()	METH	Same as the Font access/assign

FontApply()	METH	Set the current ::font as application font
Handle()	METH	Returns the handle of the application window
Hide()	METH	Hide this window so it is not visible
IsMdi()	METH	Returns .T. for MDI application, .F. for SDI
KeyboardFocus()	METH	get current or restore keyboard focus
Mcol()	METH	Get mouse position on user window
McolApp()	METH	Get mouse position on applic window
MouseAppTrap()	METH	Set mouse trapping on/off
MouseTrap()	METH	Set mouse trapping on/off
Move()	METH	Moves the application window to new position
Mrow()	METH	Get mouse position on user window
MrowApp()	METH	Get mouse position on applic window
NotifyAll	ACC/ASS	user code block, executed for all events
NotifyClose	ACC/ASS	user code block, executed on closing the applic
NotifyMdiClose	ACC/ASS	user code block, executed on closing the MDI
NotifyMove	ACC/ASS	user code block, executed on window movement
NotifyResize	ACC/ASS	user code block, executed on window resizing
Pixel2Col()	METH	Re-calculate given pixels to column coordinates
Pixel2Row()	METH	Re-calculate given pixels to row coordinates
PrgArgs()	METH	Returns an array containing the given arguments
ProcessEvents()	METH	Process pending events for a given time
Resize()	METH	Resizes the application window
Row2Pixel()	METH	Re-calculate given row coordinates to pixels
RowSize	ACC/ASS	Returns or set pixel size (height) of one row
RowSizeDef()	METH	Calculates the default size of 1 row in pixel
SetFixSize()	METH	Set fix sized application window
Show()	METH	Show minimized/maximized/normal
StatusBar	ACC	Returns the StatusBar object
Style	ACC/ASS	Returns or set the common Look and Feel
WinData()	METH	Returns an array containing window data

Application Window Class Properties

_gAppWindow { [exp1...exp7] } –> oAppWindow ***CREATOR***
_bAppWindow { [exp1...exp7] } –> oAppWindow ***CREATOR***
_tAppWindow { [exp1...exp7] } –> oAppWindow ***CREATOR***

_gAppWindowNew ([exp1...exp7]) –> oAppWindow ***CREATOR, altern syntax***
_bAppWindowNew ([exp1...exp7]) –> oAppWindow ***CREATOR, altern syntax***
_tAppWindowNew ([exp1...exp7]) –> oAppWindow ***CREATOR, altern syntax***

Instantiates the GUI application functionality, which includes the application window (including scroll bars, menu's etc.) and the user window in SDI or MDI mode at given coordinates and with the given or default window size. This class is for GUI (graphical interface) environment only. For other environment, use either _bAppWindow{} for basic i/o or _tAppWindow{} for terminal i/o. You may determine the currently used environment via the standard IsGuiMode() function.

Note, you may instantiate an application only once, so if you have already instantiated gApp class, you need to pass the used gApp object as the 1st parameter. This class is usually instantiated and assigned to a global constant oAppWindow in the user modifiable InitIo() function, which is called automatically at start-up of the FlagShip application just before other INIT functions or procedures or the main .prg module is invoked. The source code of InitIo() is available in the <FlagShip_dir>/system/ initio.prg file.

Arguments (all optional):

<**expO1**> : Owner (parent) object of the application. If you have already instantiated gApp class, you need to pass the used gApp object in this parameter. If the gAppWindow is instantiated alone as a top- most object, pass NIL (or no entry) instead.

<**expN2**> : horizontal coordinate (row) of the application window (top left edge) in either pixel coordinates or rows, depending on current SET PIXEL setting. Note, the default SET PIXEL is OFF for a backward compatibility, so the default entry is the row number, also as a decimal fraction. The horizontal pixel position is determined from the default window font. If not given, 0 is the default. May be changed by oAppWindow:DefSize()

<**expN3**> : vertical coordinate (column) of the application window (top left edge) in either pixel coordinates or columns, depending on SET PIXEL setting. If not given, 0 is the default. May be changed by oAppWindow:DefSize()

<**expN4**> : vertical size of the application window (pixels or rows, see above). If not given, the size is calculated to fit 25 rows when using the default font. May be changed by oAppWindow:DefSize()

<**expN5**> : horizontal size of the application window (pixels or rows, see above). If not given, the size is calculated to fit 80 columns when using the default font. May be changed by oAppWindow:DefSize()

<**expC6**> : string with the title of the application, displayed at the top bar. If not given, the name of the application is used. May be retrieved or changed by oAppWindow:Caption later.

<**expN7**> : attributes specifying the appearance of the application window. The attributes are specified in the applic.fh header file, the default setting is APP_SDI + APP_ALLBARS + APP_SBAR_AUTO which means: SDI (single document interface) behavior, menu bar, status bar, tool bar and tool tips are enabled, automatic horizontal and vertical scroll bars. These settings (except the SDI or MDI mode) can also be changed later.

The instantiation does not display the window yet, to be able to customize your application window by invoking of oAppWindow methods. To display the window, use the ::Display() or ::Show() methods. Life time of the object and application: when the last instantiated object of the _?AppWindow class is destroyed, the application will automatically terminate. This is usually not the case when setting it as public constant in InitIo(), where the application terminates at the usual QUIT command or RETURN from the main .prg module.

Example 1: instantiation of gAppWindow (done in initio.prg):

```
oAppWindow := _gAppWindow { }
oAppWindow:Display()
```

Example 2: instantiation of gApp and then gAppWindow:

```
if ! IsGuiMode()
   ? "-- sorry, cannot start GUI application, invoke startx first"
   quit
endif

oApp := gApp { }                    // instantiate gApp class first
? "desktop size = ", Ltrim(oApp:DesktopHeight), "x", ;
   Ltrim(oApp:DesktopWidth)
? "start-up parameters (including the executable name): "
aeval(oApp:PrgArgs(), {|x| qqout("'" + x + "' ") } )
if len(oApp:PrgArgs() ) <= 2
   ? "sorry, I need at least the two first command line parameters"
   ?
   quit
endif
oAppWindow := _gAppWindow { oApp }   // now, instantiate the class
oAppWindow:Display()
```

oAppWindow:Attrib —> iUsedMode *ACCESS*
oAppWindow:Attrib := iUsedMode *ASSIGN*

Returns or sets the binary or-ed mode of the application window (GUI mode), set during the instantiation or by oAppWindow:Enable*Scroll(). For setting the attributes, you may add or BinOR() the constants from applic.fh, the new attributes overrides the old one. When redefining the SDI to MDI and vice versa, the old user window(s) are closed and destroyed. You can determine the single settings by BinAND() the returning value with the attributes from the applic.fh header file, e.g.

```
local iMode := oApp:Attrib
? "this is a", if( BinAND(iMode, APP_MDI), "MDI", "SDI"), ;
   "based application"
? "the toolbar is", if ( BinAND(iMode, APP_TOOLBAR), "", "not "), ;
   "enabled"
```

ACCESS

oAppWindow:Caption —> cTitle
oAppWindow:Caption := cTitle *ASSIGN*

Returns or set the caption (title), i.e. the text displayed in the title bar of the application window (GUI mode). The default is the name of the executable.

oAppWindow:ColorBackground —> cColor *ACCESS*
oAppWindow:ColorBackground := cColor *ASSIGN*

Returns or sets the standard background color. The default value is set at application start and corresponds to main window background color in GUI mode, or "N" otherwise. This value is used when "?" is available in the color specification.

<**cColor**> is character string in SET COLOR notification, i.e. either combination of "N,W,R,G,B,+" symbols or as RGB string "#RRGGBB" where RR, GG and BB are hexadecimal values (00 to FF) specifying the triplet color.

oAppWindow:ColorRgbBackground —> aRgbValue *ACCESS*
oAppWindow:ColorRgbBackground := aRgbValue *ASSIGN*

Returns or sets the standard background color. The default value is set at application start and corresponds to main window background color in GUI mode, or {0,0,0} otherwise. This value is used when "?" is available in the color specification. This property corresponds to :ColorBackground and is provided for convenience to set or return color via RGB triplet.

<**aRgbValue**> is an array of three numeric elements in the range of 0 to 255 specifying each color triplet {red,green,blue}

oAppWindow:CurrSize(expN1, [expL2]) —> nValue

Retrieves the current application window sizes (GUI mode). Arguments:

<**expN1**> : is a constant (specified in the applic.fh header file) representing the request mode:

APP_Y_TOP = return top edge of the window in rows coordinates or a vertical pixel size (see also <expL2>)
APP_X_TOP = return left edge of the window in columns coordinates or a horizontal pixel size
APP_Y_SIZE = return vertical size of the application window (pixels or rows)
APP_X_SIZE = return horizontal size of the application window (pixels or columns)
APP_Y_USER = on visible window only: return vertical size of the inner part of the application window (pixels or rows)
APP_X_USER = on visible window only: return horizontal size of the inner part of the applic window (pixels or columns)

The *_SIZE is the outlined box size, the *_USER values represents the inner canvas of the application window, used for the MDI or SDI window. The _USER value is computed from the outer size *_SIZE and the window attributes (enabled/disabled menu bar, tool bar, scroll bar, status bar etc.).

<**expL2**> : (optional) if not specified or is NIL, the returned values represents the row/cols or pixels, depending on the current state of SET PIXEL on/OFF. If specified .T., the returned values are in pixels. If specified .F., the returned values are in row/col coord.

Returns the requested value according to <expN1> and <expL2>, or -1 on error. On program start-up, the windows coordinates are set to defaults or to by oAppWindow:DefSize() specified values, so e.g. oAppWindow:CurrSize (APP_Y_TOP) == oAppWindow:DefSize() [APP_Y_TOP]. When the user resizes or moves the window by a mouse, the current coordinates and size changes, whereby the current values can be determined by this method and may differ from the program defaults.

See also oAppWindow:DefSize(), oAppWindow:NotifyMove,

oAppWindow: NotifyResize, oAppWindow:Attrib, oAppWindow:WinData()

oAppWindow:DefSize([expA1], [expL2]) —> aOldSize

Sets and/or returns the default application window sizes (GUI mode). The optional <expA1> argument is an array of numeric (or NIL) elements, representing

[1 = APP_Y_TOP]	default top edge of the window in rows or vertical pixel coordinates (see also <expL2>)
[2 = APP_X_TOP]	default left edge of the window in columns or horizontal pixel coordinates
[3 = APP_Y_SIZE]	default vertical size of the application window (pixels or rows)
[4 = APP_X_SIZE]	default horizontal overall size of the application window (pixels or columns)
[5 = APP_Y_MIN]	minimal allowed vertical size of the application window (pixels or rows).
[6 = APP_X_MIN]	minimal allowed horizontal size of the application window (pixels or columns)
[7 = APP_Y_MAX]	maximal allowed vertical size of the application window (pixels or rows)
[8 = APP_X_MAX]	maximal allowed horizontal size of the application window (pixels or columns)

The constants representing the array elements are specified in the applic.fh header file.

The *_TOP and *_SIZE values are used when oAppWindow:Display() is executed for the first time. When the user resizes or moves the window by a mouse, the current coordinates and size will not match to these defaults; the current values can be determined by oAppWindow:Curr- Size(). You may programmatically re-move or re-size the window to these defaults by using e.g.

```
oAppWindow:Move(oAppWindow:DefSize()[1], oAppWindow:DefSize()[2] )
```

and/or

```
oAppWindow:Resize (...)
```

methods at any time.

The *_MIN and *_MAX values restricts both the user and the program not to re-size the window below or beyond these values. On attempt to under/oversize the window, its size is corrected automatically to the allowed minimum or maximum. If the <expA1> argument is not specified or is NIL, only the current default values are returned. If the array element of <expA1> is not numeric type, out of range, or if the array size is shorter, the corresponding element remains unchanged.

<**expL2**>: is an optional logical value. If not specified or is NIL, the given and returned values represents the row/cols or pixels, depending on the current state of SET PIXEL on/OFF. If specified .T., the given and returning values are in pixels. If specified .F., the given and returning values are in row/col coordinates.

Returns: an array of 8 numeric elements in the same order of <expA1> with the default settings (determined at the time of the method entry) in either row/column or pixel values, depending on the current SET PIXEL setting or the <expL2> value. If the coordinates or sizes were changed by <expA1>, the application window is moved or resized accordingly.

In Terminal i/o mode (-io=t), the values cannot be translated to row/col, therefore elements 1...4 are always in pixel.

Example 1: set the topmost window position and minimal horizontal size in pixel:

```
oAppWindow:DefSize( {20, 50, NIL, NIL, NIL, ;
                     oAppWindow:DesktopWidth-200}, .T.)
oAppWindow:Display()
```

Example 2: set the max available window height to desktop size:

```
#include "applic.fh"
local aDefa := array(APP_ROWCOL_ARR)
aDefa[APP_ROWS_MAX  ] := m->oApplic:DesktopHeight -120
aDefa[APP_COLS_MAX  ] := min(m->oApplic:DesktopWidth, ;
                             Col2Pixel(80) + 40)
aDefa[APP_ROWS_AVAIL] := aDefa[APP_ROWS_MAX]
aDefa[APP_COLS_AVAIL] := aDefa[APP_COLS_MAX]

@ aDefa[APP_ROWS_MAX], 0 say " " PIXEL
m->oApplic:DefSize(adefa, .T.)
```

Example 3: see also oAppWindow:Resize() and oAppWindow:SetFixSize()

oAppWindow:Display() —> self

(Re-) Displays the application window and all used sub-windows (i.e. user windows) in GUI mode. Ignored when the window is minimized or hidden.

oAppWindow:EnableHorizontalScroll([expNL1]) —> lnEnabled
oAppWindow:EnableVerticalScroll ([expNL1]) —> lnEnabled

Enable/disable/check horizontal or vertical scroll bar in an application window (GUI mode). Note: You may use either this method, or the corresponding flag in the <expN7> parameter during the gAppWindow instantiation.

<**expNL1**> is either optional logical value signaling on or off, or optional numeric value (APP_HSBAR_ON, APP_HSBAR_OFF, APP_HSBAR_AUTO or APP_VSBAR_ON, APP_VSBAR_OFF, APP_VSBAR_AUTO) whereby the constants are defined in the applic.fh file. If the parameter is not specified or is invalid, only current setting is returned. The return value <lnEnabled> depends on the <expNL1> parameter: if logical, the current on/off status is returned as logical value (.T. signals "always enabled" scrollbar), otherwise numeric value APP_HSBAR_* or APP_VSBAR_* signals the current state.

oAppWindow:ErrorMessage(expC1, [expN2], [expNCA3], [...]) —> nChoice

Display a message in an error box. This method is equivalent to the ALERT(expC1, expA2) standard function. The arguments are equivalent to those of MessageBox class:

<**expC1**> : The description text to be displayed in the error box.

<**expN2**> : Type of the message box. One of the constants MBOX_INFO, MBOX_WARNING, MBOX_ERROR, MBOX_QUEST or MBOX_NONE from dialog.fh, specifying the type of the box and the used icon. If omitted, MBOX_WARNING is the default here.

<**expNCA3**> : Type and caption of the used push button(s). Either numeric constant(s) or a string or an array with numeric or character elements specifying the response buttons. If not specified, a single "OK" option is presented.

<**expC4**>...<expO6> according to the MessageBox class are supported as well.

Returns: a numeric value indicating which option was chosen, same as MessageBox:Exec() .

oAppWindow:Font([expO1]) —> oFont

Returns or set the default application font object (GUI mode). This method is for your convenience only and is equivalent to oAppWindow:Font Access/Assign.

<**expO1**>: If specified, this font object will be set as the font of the current window.

Returns: The font object used in the current window.

oAppWindow:Hide() —> self

Hides the application window so it is not visible (GUI mode). To re-display, use oAppWindow:Display()

oAppWindow:IsMDI() —> lMdiMode

Returns .T. for MDI application, .F. for SDI (default). The MDI mode for Multiple Document Interface let you open additional user windows within the same application. Available in GUI mode only. Initialized by the -mdi compiler switch. When the -mdi switch was not specified during the link phase, or with non-GUI applications, the common SDI (for Single Document Interface) application mode is created.

oAppWindow:KeyboardFocus( ) —> ret

Get current keyboard focus address if any. Applicable for GUI mode, ignored otherwise. The "focus" is the current widget with input, e.g. the GET field, Prompt, Pushbutton, Radiobutton, Listbox, Memoedit(), Achoice(), Tbrowse() etc. and is assigned to variable <ret> if given. You may restore this focus later by passing the <ret> variable as parameter to oAppWindow:KeyboardFocus(@ret), see below.

oAppWindow:KeyboardFocus(@exp1) —> retL

Restore keyboard focus. Applicable for GUI mode, ignored otherwise.

<**exp1**> is the previously stored keyboard focus address. Pass this variable by reference. The <exp1> variable is reset to NIL upon return, hence you can use the return value from oAppWindow:KeyboardFocus( ) only once (simply repeat it on needs); this is for security reason, see note.

<**retL**> signals .T. on success, and .F. on failure

Note: use this method with care, the focus may NOT be restored for already closed or deleted widgets, otherwise segfault/protection fault may occur!

Example:

```
local saveFocus as usual
saveFocus := m->oApplic:KeyboardFocus( )       // save focus
... set focus to anything else
ok := m->oApplic:KeyboardFocus( @saveFocus )   // restore focus
```

oAppWindow:Move([expN1], [expN2], [expL3]) —> self

Moves the application window to the default or a new position. Arguments (all optional):

<**expN1**> : the new row number (also decimal fraction) or a vertical position of the top left edge in pixel (depending on the current SET PIXEL setting and the <expL3> argument). If not given, the default value from the object instantiation or from oAppWindow: DefSizes() invocation is used.

<**expN2**> : the new column number (also decimal fraction) or a horizontal position of the top left edge in pixel (depending on the current SET PIXEL setting and/or <expL3>). If not given, the default setting from the object instantiation or from oAppWindow:DefSizes() is used.

<**expL3**>: an optional logical value. If not specified or is NIL, the given values represent the row/cols or pixels, depending on the current state of SET PIXEL on/OFF. If specified .T., the given values are in pixels. If specified .F., the given values are in row/col coordinates.

oAppWindow:Move(...) is designed mainly for GUI mode. It however moves also Terminal i/o (MS-Windows only) similar to ConsoleSize(). In Linux, use xterm settings by newfswin.

Example: center application on desktop:

```
#include "applic.fh"
if AppIoMode() == "G"
   local xDesktop := oApplic:DesktopWidth
   local yDesktop := oApplic:DesktopHeight
   local xApplic  := oApplic:CurrSize(APP_X_SIZE, .T.)
   local yApplic  := oApplic:CurrSize(APP_Y_SIZE, .T.)
```

```
   oApplic:Move((yDesktop - yApplic) /2, ;
                (xDesktop - xApplic) /2, .T.)
endif
```

oAppWindow:NotifyAll –> cBlock *ACCESS*
oAppWindow:NotifyAll := cBlock *ASSIGN*

Event handler callback (GUI mode). Returns or set a user-supplied code block which should handle all events not handled previously by the oAppWindow:NotifyClose, oAppWindow:NotifyMove or oAppWindow:Notify- Resize event handler. The code block receives three arguments: the event number (APP_EV_* according to applic.fh), window-ID number (usually 0), and a string specifying the event origin (e.g. "Screen" or "EventHandler" or "Tbrowse" or "WindowAbort" etc.). You may get the event name in clear text (string) by Event2str(event_number). If the code block returns .T. the event should be processed further by the default event handler. When the code block return .F., the default event handler will ignore further processing of this event. Assigning a NIL value resets this event handler.

Example:

```
// print events on terminal screen (or to stderr file)
oAppWindow:NotifyAll := ;
     { |iEvent,iWin,cWhere| myEventNotif(iEvent,iWin,cWhere) }
...
FUNCTION myEventNotif(eventNo,iWin,cWhere)
??## time(1), "Event#" + ltrim(iEvent)+ "=" + event2str(iEvent), ;
     "occured in window#" + ltrim(iWin), "from", cWhere
return .T.    // process it
```

oAppWindow:NotifyClose –> cBlock *ACCESS*
oAppWindow:NotifyClose := cBlock *ASSIGN*

Event handler callback (GUI mode). Returns or set a user-supplied code block which handles the Applic.Window close event, sent by a mouse click on the (X) icon top right in main window. This event is sent just before the window (hence also the application) is closed. As with all other Notify* handlers, the code block receives the event number (here APP_EV_CLOSE) as a parameter. The code block itself or an UDF invoked from the code block may e.g. display an alert or message window asking the user if the application should really be closed. If the code block return .T., the event is accepted and the application closed. Otherwise the attempt for closing the window is rejected. Assigning a NIL value resets this event handler, further APP_EV_CLOSE events are sent to the general oAppWindow:NotifyAll handler, if any. If not so, the application is silently closed.

Example:

```
oAppWindow:NotifyClose := { |iEvent| myCloseHandle(iEvent) }
...
FUNCTION myCloseHandle(eventNo)
if ALERT("Do you want to close application?", {"No", "Yes"}) == 2
   CLOSE ALL
   return .T.
endif
return .F.
```

oAppWindow:NotifyMove –> cBlock *ACCESS*
oAppWindow:NotifyMove := cBlock *ASSIGN*

Event handler callback (GUI mode). Returns or set a user-supplied code block which handles the move event. This event is sent just before the window is moved by mouse. As with all other Notify* handlers, the code block receives the event number (here APP_EV_MOVE) as a parameter. If the code clock return .T., the event is accepted and the application window moved. Otherwise the attempt for moving the window is rejected. A NIL value resets this event handler, further APP_EV_MOVE events are sent to the general oAppWindow:NotifyAll handler, if any.

oAppWindow:NotifyResize –> cBlock *ACCESS*
oAppWindow:NotifyResize := cBlock *ASSIGN*

Event handler callback (GUI mode). Returns or set a user-supplied code block which handles the resizing event. This event is sent just before the window is resized by mouse. As with all other Notify* handlers, the code block receives the event number (here APP_EV_RESIZE) as a parameter. If the code clock return .T., the event is accepted and the application window resized, otherwise the attempt is rejected. A NIL value resets this event handler, further APP_EV_RESIZE events are sent to the general oAppWindow:NotifyAll handler, if any.

oAppWindow:ProcessEvents([expN1]) –> self

Process pending events for a given time in milliseconds or until there are no more events to process. You will usually not need to call this method from your .prg code (although you can), since it is invoked periodically by FlagShip run-time. You only will need to frequently call it (at least within 1-3 seconds) from a large or a time consuming C source (which is not handled automatically by the run-time) to allow the event manager to handle all the user events done in the meantime, like window resizing, movement, refresh, close, key and mouse trapping etc. Argument (optional):

<**expN1**> : Time period or mode. If specified 0 (zero) or not given, process all pending events but for max. 3000 milliseconds (3 sec). If > 0, process all pending events but max for the given time period <expN1> in milliseconds. If < 0, waits for one event to process it.

For your convenience, there is a function named ProcessEvents() which perform the same task. For source parts written in C, corresponding C callable function 'int ProcessEvents()' and 'int ProcesEventsClock (int milliSec)' are available as well, where the first checks only for pending output, and the second checks for output and displays clock. The milliSec parameter is usually 3000.

oAppWindow:Resize([expN1],[expN2],[expL3],[expL4],[expL5]) —> self

Resizes the application window to the default or a new size. See also oAppWindow:Size for an alternative setting. Arguments (all optional):

<**expN1**> : the new size in rows (also decimal fraction) or a vertical size of the application window in pixel (depending on the current SET PIXEL setting and the <expL3> value). If not given, the default setting from the object instantiation or from oAppWindow:defSizes() is used.

<**expN2**> : the new size in columns (also decimal fraction) or a horizontal size of the application window in pixel (depending on the current SET PIXEL setting and the <expL3> value). If not given, the default setting from the object instantiation or from oAppWindow:DefSizes() is used.

<**expL3**> : an optional logical value = pixels. If not specified or is NIL, the given values represents the row/cols or pixels, depending on the current state of SET PIXEL on/OFF. If specified .T., the given values are in pixels. If specified .F., the given values are in row/col coordinates. Applicable for GUI mode only.

<**expL4**> : an optional logical value = auto-resize. If not given or is true (.T.), the Resize() method will increase the application window automatically so, that at least the number of given rows and/ or columns are available and visible. With auto-resizing, the <expN1> and <expN2> coordinates are the inner window frame. The MaxRow() and MaxCol() will be set accordingly. Note: the auto-resizing may produce slightly inaccurate results in some windows manager. If <expL4> is false (.F.), the coordinates are outer application window frame and the programmer will control the visibility manually, see example 2 and 4 below. Best also to set MaxRow() and MaxCol().

<**expL5**> : an optional logical value = resize also status bar. If not given or is .T., the status bar items are resized accordingly to the size of the application window. False (.F.) value permits resizing of the status bar items. Applicable for GUI mode only.

oAppWindow:Resize(...) is designed mainly for GUI mode. It however resizes also Terminal i/o (MS-Windows only) similar to ConsoleSize(), i.e. it may decrease current window, but cannot increase it. In Linux, use xterm settings by newfswin.

Example 1: At application start, the window is sized to 25 rows by 80 columns, where the size of the largest character is used (which is usually large letter like M, Q, X etc, or semi-graphic character like chr(215)). If you wish to have smaller window, you may resize the applic window e.g. to an average of an upper/lower letter width by:

```
m->oApplic:Resize(25, StrLen2Col(replicate("Xx",40)) )
```

Example 2: To set screen height of the whole, current desktop less 40 pixel at bottom, and a width of 80-times the "X" character, use:

```
nRows := int( Pixel2row(m->oApplic:DesktopHeight - 40 - 80) )
m->oApplic:Resize(nRows, StrLen2Col(replicate("X",80)) )
```

or ditto manually:

```
iSizeX := StrLen2pix(replicate("X",80)) + 15 // add frame
iSizeY := m->oApplic:DesktopHeight - 40
MaxCol(.T., StrLen2pix(replicate("X",80)))
MaxRow(.T., iSizeY - 80)                    // subtract frame
m->oApplic:Resize(iSizeY, iSizeX, .T., .F.)
```

where both work fine for either proportional or fixed fonts. With fixed font set, see also the next example.

Example 3: set fixed font, resize to 25x80 rows/columns corresponding to the used font

```
SET FONT "courier" SIZE 12

m->oApplic:Resize(25, 80)     // auto-resize

// display test grid
line80 := "0....:....1....:....2....:....3....:...." + ;
          "4....:....5....:....6....:....7....:...."
for ii := 0 to 24
   @ ii,0 say line80
   @ ii,0 say ltrim(ii)
next
m->oApplic:WinData(.T.)       // display windows data on stderr
setpos(0,0)
wait "before exit..."
```

Example 4: same as example 3, but the programmer controls the visibility and have specified also the max resizable area. Note that the window coordinates specify here outer frame, so add frame displacement (may vary in dependence on used window manager).

```
#include "applic.fh"

#define DISPL_Y 80
#define DISPL_X 35

m->oApplic:Font:FontName("courier")
m->oApplic:Font:SizePoint(12)

// set max app.window sizes
aDefa := array(APP_ROWCOL_ARR)
aDefa[APP_ROWS_MAX  ] := m->oApplic:DesktopHeight -50  // screen
size
* aDefa[APP_ROWS_MAX] := Row2Pixel(25) + DISPL_Y       // or 25
rows
```

```
aDefa[APP_COLS_MAX  ] := min(m->oApplic:DesktopWidth, ;
                             Col2Pixel(80) + DISPL_X)

@ aDefa[APP_ROWS_MAX], 0 say " " PIXEL       // ensure auto-resize
m->oApplic:DefSize(adefa, .T.)               // set defaults

// resize to 25x80
m->oApplic:Resize(Row2pixel(25) + DISPL_Y, ;
                  Col2pixel(80) + DISPL_X, ;
                  .T., .F.)                  // manual resize
m->oApplic:display()

// display test grid 25x80
setpos(0,0)
wait "before say..."
line80 := "0....:....1....:....2....:....3....:...." + ;
          "4....:....5....:....6....:....7....:...."
for ii := 0 to 24
   @ ii,0 say line80
   @ ii,0 say ltrim(ii)
next
setpos(0,0)
wait "before exit..."
```

oAppWindow:SetFixSize(expN1, expN2, [expL3]) —> lOk

Resizes the application window to fix size. Applicable in GUI mode only, ignored otherwise. See also oAppWindow:Resize for an alternative setting.

<**expN1**> : the new size in rows (also decimal fraction) or a vertical size of the application window in pixel (depending on the current SET PIXEL setting and/or the <expL3> value).

<**expN2**> : the new size in columns (also decimal fraction) or a horizontal size of the application window in pixel (depending on the current SET PIXEL setting and/or the <expL3> value).

<**expL3**> : an optional logical value = pixels. If not specified or is NIL, the given values represents the row/cols or pixels, depending on the current state of SET PIXEL on/OFF. If specified .T., the given values are in pixels. If specified .F., the given values are in row/col coordinates. Applicable for GUI mode only.

<**lOk**> : returns .T. on success and .F. on failure

oAppWindow:SetFixSize(...) resizes the application window to size specified by <expN1> and <expN2>. The window cannot be resized thereafter by mouse, but only programatically by oAppWindow:Resize(). Also the maximize [O] and minimize [o] button at the window frame does not maximizes nor minimizes it anymore, this however depends on the system window manager. Note that oAppWindow:SetFixSize() do not apply for Wopen() nor MdiOpen(), these sub-windows are not user-resizeable.

Example: At application start, the window is sized to 25 rows by 80 columns, where the size of the largest character of current font is used (which is usually broad letter

like M, Q, X etc, or semi-graphic character like chr(215)). If you wish to have a window of fix size, which cannot be resized by user/mouse, you may use e.g.:

```
SET FONT "courier", 12
m->oApplic:SetFixSize(25, 80)
```

oAppWindow:Show([expN1]) —> nStatus

Set and/or get the window visibility (GUI mode). Argument (optional):

<**expN1**> a constant defined in applic.fh that represents how the window is shown:

APP_NORMSIZE (= SHOWNORMAL) : Shows the window on its owner (usually a desktop), in a size before minimizing or maximizing

APP_MINSIZE (= SHOWICONIZED) : Iconize, i.e. minimize the window.

APP_MAXSIZE (= SHOWZOOMED) : Shows the window at the maximum size allowed by its owner. The Application window occupies the whole desktop.

The window visibility will only be changed when a valid argument was passed. Note that also the user may iconize (minimize), maximize or reset the normal window visibility by a mouse click on the corresponding icon in the top right corner of the window.

Returns: a constant (see <expN1>) representing the current status (at the time of entering the method) set either by the application via this Show() method, or set by the user via mouse click.

oAppWindow:Style —> nStyle *ACCESS*
oAppWindow:Style := nStyle *ASSIGN*

Returns or set the appearance style, i.e. the common Look and Feel of a GUI application. The <nStyle> constant is APP_STYLE_MOTIF (Motif alike style), APP_STYLE_CDE (Common Desktop Environment alike) or APP_STYLE_WINDOWS (MS-Windows alike), all defined in applic.fh file. Per default, the style corresponds to the used environment, i.e. APP_STYLE_MOTIF in Unix and APP_STYLE_WINDOWS in MS-Windows.

oAppWindow:WinData([expL1]) —> aStatus

Determine and optionally print significant application window (GUI mode) data as an extract from other oAppWindow methods, e.g. :Desktop*(), :Font*(), :CurrSize() etc. Argument (optional):

<**expL1**> is a logical value. If set .T., the <aStatus> data are printed in human readable form to stderr device.

Returns: <aStatus> is three-dimensional array describing important application window data. The array elements corresponds to APP_WINDATA_* constants defined in applic.fh and the sub-array in each element is a textual description, row/height and column/width in pixel, in that order.

Example 1:

```
#include "applic.fh"
aData := m->oAppWindow:WinData()
yUserPix := aData[APP_WINDATA_USERSIZE,2]             // pixel
xUserCol := Pixel2col(aData[APP_WINDATA_USERSIZE,3])  // columns
? aData[APP_WINDATA_USERSIZE,1], "=", ;
  ltrim(yUserPix),"(in pixel) *", ltrim(xUserCol),"(in cols)"
```

Example 2: display data on screen

```
aeval(m->oAppWindow:WinData(), ;
      {|x| Qout(x[1], "=", ltrim(x[2]), "/", ltrim(x[3])) })
```

Example 3: print data to stderr

```
m->oAppWindow:WinData(.T.)
```

Basic Classes

These basic classes are often used to carry information for other classes.

The basic classes are:

- Color Class
- ColorPair Class
- Dimension Class
- Font Class
- Mouse Class
- Point Class
- Rectangle Class
- Size Class

Color Class

creates a Color object, which is used to describe color settings via RGB values. It can contain either the foreground or background color specification.

Class Color
Inherits from: - (none)
Inherited by: ColorPair
Class prototype: basclass.fh
Defines: color.fh

Properties of the Color Class

Color { [expNA1], [expN2], [expN3] } —> oColor ***CREATOR***
ColorNew ([expN1], [expN2], [expN3]) —> oColor ***CREATOR, altern syntax***

Instantiates a Color object by the RGB triplet values, or by RGB(0,0,0) = black. The triplet specifies your own mix of red, green, and blue components, each can be a value between 0 (lowest intensity) and 255 (highest intensity). Arguments (all optional):

<**expA1**> : An array of three elements {red,green,blue}. If the 1st argument is detected as array, the <expN2> and <expN3> arguments will be ignored.
<**expN1**> : The red component of the color in range of 0 to 255
<**expN2**> : The green component of the color in range of 0 to 255
<**expN3**> : The blue component of the color in range of 0 to 255

oColor:Blue —> iVal ***ACCESS***
oColor:Blue := iVal ***ASSIGN***

Access or assign the blue triplet of the color as a value from 0 to 255

oColor:Green —> iVal ***ACCESS***
oColor:Green := iVal ***ASSIGN***

Access or assign the green triplet of the color as a value from 0 to 255

oColor:Red —> iVal ***ACCESS***
oColor:Red := iVal ***ASSIGN***

Access or assign the red triplet of the color as a value from 0 to 255

oColor:Rgb ([aTrippl]) —> aTrippl ***METHOD***
oColor:Rgb ([iRed], [iGreen], [iBlue]) —> aTrippl ***METHOD***

Returns or set the corresponding color. If no arguments are given, only the array of 2 numeric elements (red, green, blue) with values in the range 0 to 255 is returned.

ColorPair Class

creates a ColorPair object, containing two Color objects as a pair for foreground and background.

Class ColorPair
Inherits from: Color
Inherited by: - (none)
Class prototype: basclass.fh
Defines: color.fh

Properties of the ColorPair Class

ColorPair { [expO1], [expO2] } —> oColorPair ***CREATOR***
ColorPairNew ([expO1], [expO2]) —> oColorPair ***CREATOR, altern syntax***

Instantiates a ColorPair objects.

<**expO1**> is the foreground Color object. If not given, the default is black = Color{ 0, 0, 0}

<**expO2**> is the background Color object. If not given, the default is white = Color{255,255,255}

oColorPair:Background —> oColor ***ACCESS***
oColorPair:Background := oColor ***ASSIGN***

Set or return the background color part of the object. Example:

```
? oPair:Background:Red       // 127  }
? oPair:Background:Green     // 127  }  a gray color
? oPair:Background:Blue      // 127  }
```

oColorPair:Foreground —> oColor ***ACCESS***
oColorPair:Foreground := oColor ***ASSIGN***

Set or return the foreground color part of the object.

Dimension Class

This class is provided for compatibility purposes and is equivalent to the Size class.

Class Dimension
Inherits from: Size
Inherited by: - (none)
Class prototype: basclass.fh
Defines: - (none)

oDimension := Dimension {[expN1], [expN2], [expL3]} ***CREATOR***
oDimension := DimensionNew ([expN1], [expN2], [expL3]) ***CREATOR alter.syntax***

See description and properties of the Size class

Mouse Class

This class is used to hold the mouse information. It is available in all i/o modes, but a meaningful information is given in the GUI mode only.

The mouse class is instantiated automatically in the InitIo() start-up function (see <FlagShip_dir>/system/initio.prg) to a global constant named "**_oMouse**" and should not be instantiated extra.

Class Mouse = _gMouse, _bMouse, _tMouse
Inherits from: - (none)
Inherited by: - (none)
Class prototype: mouseclass.fh
Defines: - (none)

Mcol()	METHOD	Determine the mouse cursor's screen column position
Mhide()	METHOD	Hide the mouse pointer
MLeftDown()	METHOD	Determine the press status of the left mouse button
MPresent()	METHOD	Determine if a mouse is present
MRestState()	METHOD	Re-establish the previous state of a mouse
MRightDown()	METHOD	Determine the status of the right mouse button
Mrow()	METHOD	Determine the mouse cursor's screen row position
MSaveState()	METHOD	Save the current state of a mouse
MSetBounds()	METHOD	Define an inclusion region (*)
MSetClip()	METHOD	Define an inclusion region (*)
MSetCursor()	METHOD	Determine a mouse's visibility
MSetPos()	METHOD	Set a new position for the mouse cursor
MShow()	METHOD	Display the mouse pointer
MState()	METHOD	Return the current mouse state

Point Class

This class is used to hold information about specific coordinate.

Class Point
Inherits from: -
Inherited by: -
Class prototype: basclass.fh
Defines: -

Properties of the Point Class

Point { [expN1], [expN2], [expL3] } —> oPoint ***CREATOR***
PointNew ([expN1], [expN2], [expL3]) —> oPoint ***CREATOR, altern. syntax***

Instantiates a point object. Arguments (all optional):

<**expN1**> : The x (column) coordinate of the widget. If not given, 0 is the default. See also oPoint:x and oPoint:x()

<**expN2**> : The y (row) coordinate of the widget. If not given, 0 is the default. See also oPoint:y and oPoint:y()

<**expL3**> : an optional logical value. If not specified or is NIL, the given values represent the row/cols or pixels, depending on the current state of SET PIXEL on/OFF. If specified .T., the given values are in pixels. If specified .F., the given values are in row/col coordinates.

oPoint:X —> iColPixel ***ACCESS***
oPoint:X := iColPixel ***ASSIGN***

The x (column) coordinate of the widget in pixels. See also oPoint:X( ) for an alternative syntax.

oPoint:X([expN1], [expL2]) —> nColumn

Set and/or return the x (column) coordinate of the widget. Arguments (optional):

<**expN1**> : The x (column) coordinate of the widget. If not given or is NIL, the X value remain unchanged and only the current size is returned.

<**expL2**> : an optional logical value. If not specified or is NIL, the <expN1> and <returnN> represents a value either in row/cols or in pixels, depending on the current state of SET PIXEL on/OFF. If specified .T., the given and returned value is in pixels (and thus equivalent to oPoint:X acc/ass). If specified .F., the given and returned value is in row/col coordinates.

<**returnN**> : The x (column) coordinate of the widget at the time of entering this method, either in pixel or col/row coordinates, depending on <expL2> argument.

oPoint:Y –> iRowPixel *ACCESS*
oPoint:Y := iRowPixel *ASSIGN*

The y (row) coordinate of the widget in pixels. See also oPoint:Y( ) for an alternative syntax.

oPoint:Y([expN1], [expL2]) –> nRow

Set and/or return the y (row) coordinate of the widget. Arguments (optional):

<**expN1**> : The y (row) coordinate of the widget. If not given or is NIL, the Y value remain unchanged and only the current size is returned.

<**expL2**> : an optional logical value: If not specified or is NIL, the <expN1> and <returnN> represents a value either in row/cols or in pixels, depending on the current state of SET PIXEL on/OFF. If specified .T., the given and returned value is in pixels (and thus equivalent to oPoint:Y acc/ass). If specified .F., the given and returned value is in row/col coordinates.

<**returnN**> : The x (column) coordinate of the widget at the time of entering this method, either in pixel or col/row coordinates, depending on <expL2> argument.

Rectangle Class

This class is used to hold information about specific coordinate.

Class Rectangle
Inherits from: -
Inherited by: -
Class prototype: basclass.fh
Defines: -

Bottom	ACC/ASS	The bottom coordinate in pixel
Bottom()	METHOD	The bottom coordinate in pixel or rows
Height	ACC/ASS	The height size in pixel
Height()	METHOD	The height size in pixel or rows
Left	ACC/ASS	The left coordinate in pixel
Left()	METHOD	The left coordinate in pixel or columns
Right	ACC/ASS	The right coordinate in pixel
Right()	METHOD	The right coordinate in pixel or columns
Top	ACC/ASS	The top coordinate in pixel
Top()	METHOD	The top coordinate in pixel or rows
Width	ACC/ASS	The width size in pixel
Width()	METHOD	The width size in pixel or rows
X	ACC/ASS	The left coordinate in pixel
X()	METHOD	The left coordinate in pixel or columns
Y	ACC/ASS	The top coordinate in pixel
Y()	METHOD	The top coordinate in pixel or rows

oRect := Rectangle { [expN1],[expN2],[expN3],[expN4],[expL5] } ***CREATOR***
oRect := RectangleNew ([expN1],[expN2],[expN3],[expN4],[expL5]) ***CREATOR***

Instantiate Rectangle object.

<**expN1**>...<expN4> are top, left, bottom, right coordinates in that order. If not specified, 0/0 is used for top/left and maxrow()/ maxcol() for bottom/right. If <expN3> is negative, it specifies the height instead of bottom. If <expN4> is negative, it specifies the width instead of right.

<**expL5**> an optional logical value. If not specified or is NIL, the given values represent the row/cols or pixels, depending on the current state of SET PIXEL on/OFF. If specified .T., the given values are in pixels. If specified .F., the given values are in row/col coordinates.

Size Class

Specifies a size of an object

Class Size
Inherits from: -
Inherited by: Dimension
Class prototype: basclass.fh
Defines: -

oSize := Size { [expN1], [expN2], [expL3] } ***CREATOR***
oSize := SizeNew ([expN1], [expN2], [expL3]) ***CREATOR alter.syntax***

Instantiates a Size (or Dimension) object. Arguments (all optional):

<**expN1**> : The width (x size) of the widget. If not given, 0 is the default. See also oSize:Width and oSize:Width()

<**expN2**> : The height (y size) of the widget. If not given, 0 is the default. See also oSize:Height and oSize:Height()

<**expL3**> : an optional logical value. If not specified or is NIL, the given values represent the row/cols or pixels, depending on the current state of SET PIXEL on/OFF. If specified .T., the given values are in pixels. If specified .F., the given values are in row/col coordinates.

oSize:Height —> iHeightPixel ***ACCESS***
oSize:Height := iHeightPixel ***ASSIGN***

The height (y size) of the widget in pixels. See also oSize:Height() method for an alternative syntax.

oSize:Height([expN1], [expL2]) —> nHeight

Set and/or return the height (y size) of the widget. Arguments (optional):

<**expN1**> : The height of the widget. If not given or is NIL, the value remains unchanged and only the current size is returned.

<**expL2**> : an optional logical value. If not specified or is NIL, the <expN1> and <returnN> represents a value either in row/cols or in pixels, depending on the current state of SET PIXEL on/OFF. If specified .T., the given and returned value is in pixels (and thus equivalent to oSize:Height acc/ass). If specified .F., the given and returned value is in row/col coordinates.

<**returnN**> : The y (height) coordinate of the widget at the time of entering this method, either in pixel or col/row coordinates, depending on <expL2> argument.

oSize:Width —> iWidthPixel *ACCESS*
oSize:Width := iWidthPixel *ASSIGN*

The width (x size) of the widget in pixels. See also oSize:Width( ) method for an alternative syntax.

oSize:Width([expN1], [expL2]) —> nWidth

Set and/or return the width (x size) of the widget. Arguments (optional):

<**expN1**> : The width of the widget. If not given or is NIL, the X value remain unchanged and only the current size is returned.

<**expL2**> : an optional logical value. If not specified or is NIL, the <expN1> and <returnN> represents a value either in row/cols or in pixels, depending on the current state of SET PIXEL on/OFF. If specified .T., the given and returned value is in pixels (and thus equivalent to oSize:Width acc/ass). If specified .F., the given and returned value is in row/col coordinates.

<**returnN**> : The x (width) coordinate of the widget at the time of entering this method, either in pixel or col/row coordinates, depending on <expL2> argument.

CheckBox Class

Create check boxes, which are widgets (controls) that can be toggled on or off by a user.

A check box allows the user to choose between 2 or 3 states. A 2-state check box allows a choice between a checked (ON) and unchecked (OFF) state. The choice is reflected in CheckBox:Value access, which may be TRUE or FALSE. A 3-state check box adds a third (UNDETERMINED) state, in which the box is dimmed. The third state is indicated by CheckBox:Value being TRUE and by CheckBox:ValueChanged (which is normally TRUE) being FALSE. Compatibility note: Clipper supports 2-state mode only.

The CheckBox class has been designed to be easily integrated into the standard FlagShip GET/READ system in addition to providing the necessary functionality to be utilized on its own.

The following code creates a check box with a caption "Check me"

```
oChkBox := CheckBox{30,50, "Check me", .T.}
oChkBox:Show()
```

FlagShip also support the use of check boxes via the common @..GET / READ interface

```
lChecked := .F.
@ 5,10 GET lChecked CHECKBOX CAPTION "Check me"
READ
```

As with other GUI classes in FlagShip, the general CheckBox class is internally inherited by three different sub-classes: _gCheckBox for GUI based application, _tCheckBox for terminal/text based mode, and _bCheckBox for basic i/o mode, all defined in the boxclass.fh header file. The proper class, corresponding to the used i/o mode, is set either at compile time with the compiler switch "-io=g|t|b", or latest at run-time depending on the currently used environment.

Note: in the basic i/o mode, only a rough check box functionality is simulated by the sequential in/output.

CheckBox Class Index

Class CheckBox
Inherits from: -
Inherited by: -
Class prototype: boxclass.fh
Defines: button.fh, set.fh

Bitmaps	ACC/ASS	Available for compatibility to Clipper only
Buffer	ACCESS	Indicates whether the check box is checked or not
CapCol	ACC/ASS	Screen column of the check box's caption
CapCol()	METHOD	Screen column of the check box's caption
CapRow	ACC/ASS	Screen row of the check box's caption
CapRow()	METHOD	Screen row of the check box's caption
Caption	ACC/ASS	String that describes the check box caption
Cargo	ACC/ASS	A user value of any type
Checked	ACC/ASS	Indicates whether the check box is checked
ClassName	METHOD	For compatibility to Clipper's getsys.prg only
Col	ACC/ASS	Screen column where the check box is displayed
Col()	METHOD	Screen column where the check box is displayed
ColdBox	ACC/ASS	Frame of check box without focus
ColorSpec	ACC/ASS	Color attributes
Destroy()	METHOD	Destroys the CheckBox object
Display()	METHOD	Show the check box and its caption on the screen
Fblock	ACC/ASS	Code block evaluated at receiving/losing focus
Handler	ACC/ASS	User defined keyboard handler
HandlerSelect	ACC/ASS	Process Select() in handler or auto
HasFocus	ACC	Indicates whether the object has input focus
Height	ACC/ASS	The height of the check box
Height()	METHOD	The height of the check box (incl. pixel setting)
HitTest()	METHOD	Determines if the mouse cursor is within the box
HotBox	ACC/ASS	Frame of check box with focus
KillFocus()	METHOD	Take input focus away from a CheckBox object
Message	ACC/ASS	String displayed in the windows status bar
Modified	ACC/ASS	Indicates whether the button is clicked
Row	ACC/ASS	Screen row where the check box is displayed
Row()	METHOD	Screen row where the check box is displayed
Sblock	ACC/ASS	Code block evaluated at user selection
Select()	METHOD	Set/clear the check box checked status
SetFocus()	METHOD	Set input focus to a CheckBox object
Show()	METHOD	Activates the default or user's input handler
Style	ACC/ASS	Delimiter and status display characters
ToolTip	ACC/ASS	Short pop-up info message
TypeOut	ACC	Always .F.
Value	ACC/ASS	Indicates whether the check box is checked or not
ValueChanged	ACC/ASS	State of the 3-state check box
Width	ACC/ASS	The width of the check box
Width()	METHOD	The width of the check box (incl. pixel setting)

CheckBox Class Instantiation

oCheckBox := [_g|_t|_b]CheckBox { [nR], [nC], [cText], [lPixel] } *[1]*
oCheckBox := [_g|_t|_b]CheckBoxNew ([nR], [nC], [cText], [lPixel]) *[2]*

oCheckBox := CheckBox ([nR], [nC], [cText], [lPixel]) *[3]*
oCheckBox := CheckBox { [oOwn], [nId], [oPoint], [oDim], [cText] } *[4]*

Any of the above syntax instantiate new check box object. Syntax [1], [2] and [3] are standard FlagShip and should be preferred. Syntax [4] is supported for compatibility to VO.

The widget (control) remains invisible until you invoke oCheckBox:Show() or oCheckBox:Display(). This allows the program to set up the control correctly (with the correct size, position, and any other parameters), while avoiding the "visual noise" of changing controls. Arguments:

<**nR**> row in coordinates or pixel, optional. If not specified, 0 is the default. See additional details in the oCheckBox:Row description.

<**nC**> column in coordinates or pixel, optional. If not specified, 0 is the default. See additional details in the :Col description.

<**cText**> caption text, optional. If not redefined by :CapCol and/or :CapRow, the text is displayed in the <nR> row and <nC> + 4 column.

<**lPixel**> if true(.T.), the row and column data are in pixel; if false (.F.), data are in row/col coordinates, otherwise the current SET PIXEL is used.

<**oOwn**> owner object of the check box, optional. Default is the oApplic object.

<**nId**> an unique ID between 1 and 8000 of the check box, optional. If not specified, internal ID is used.

<**oPoint**> the origin of the check box, in canvas coordinates

<**oDim**> the dimension of the check box, in canvas coordinates

Example 1: This example creates two check boxes and process different handling:

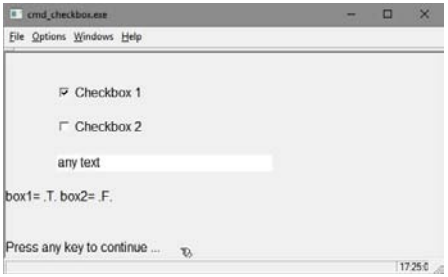
```
oBox1 := CheckBox{10,5,"Male"}
oBox2 := CheckBox(11,5)
oBox2:Caption := "Married"

// process all the handling automatically
oBox1:Show()

// handle this box manually same as in Clipper
oBox2:Display()
oBox2:SetFocus()
key := inkey()
```

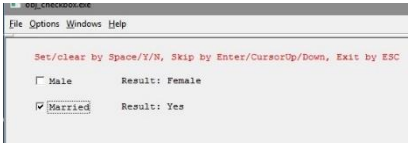
```
do case
case key == K_SPACE
   oBox2:Select(!oBox:Buffer)   // toggle on/off
case chr(key) $ "+yYtT"
   oBox2:Select(.T.)
case chr(key) $ "-nNfF"
   oBox2:Select(.F.)
endcase
oBox2:KillFocus()

// display the data
? if(oBox1:Checked,"Male","Female"), ;
  if(oBox2:Buffer, "", "not ") + "married"
```

Example 2: This example creates and integrates a check box within a GetList and activates it by performing a READ:

```
LOCAL cName := SPACE(25)
LOCAL lMarried := .T., lMale := .F.
LOCAL cAddress := space(25)

@ 5,10 SAY "Name     " GET cName
@ 7,10 SAY "Male     " GET lMale    CHECKBOX
@ 7,20 SAY "Married  " GET lMarried CHECKBOX
@ 9,10 SAY "Address  " GET cAddress
READ
@ 10,0 SAY trim(cName) + " is " + if(lMarried, "", "not ") + ;
           "married and is", if(lMale, "male", "female")
```

Example 3: see additional examples in FUN.CheckBox() and CMD.@...GET CHECKBOX





Compatibility: Available also in CL53 (syntax 3) and VO (syntax 4). See also: oCheckBox:Destroy()

CheckBox Class Properties

oCheckBox:Bitmaps —> aFile ***ACCESS***
oCheckBox:Bitmaps := aFile ***ASSIGN***

This property is available for compatibility to Clipper (in semi- graphical mode) only and is not used by FlagShip.

Compatibility: Available also in CL53.

oCheckBox:Buffer —> lChecked ***ACCESS***

<**lChecked**> is a logical value that indicates whether the check box is checked or unchecked. A value of true (.T.) indicates that it is checked and a value of false (.F.) indicates that it is not checked. Equivalent to oCheckBox:Checked instance.

Compatibility: Available also in CL53.

See also: oCheckBox:Checked, oCheckBox:Select()

oCheckBox:CapCol —> nCol ***ACCESS***
oCheckBox:CapCol := nCol ***ASSIGN***
oCheckBox:CapCol([nCol], [lPixel]) —> nCol

<**nCol**> is a numeric value that indicates the screen column where the check box's caption is displayed. The input and output value is either in coordinates or in pixels, depending on the current SET PIXEL setting. The default setting is oCheckBox:Col + 4 columns at instantiation time.

<**lPixel**> is optional value indicating if the passed and returned value is in coordinates or pixels. If true(.T.), the column data are in pixel; if false (.F.), data are in coordinates, otherwise the current SET PIXEL status is used.

Compatibility: Access/assign is Available also in CL53.

See also: oCheckBox:CapRow, oCheckBox:Caption

oCheckBox:CapRow —> nRow ***ACCESS***
oCheckBox:CapRow := nRow ***ASSIGN***
oCheckBox:CapRow ([nRow], [lPixel]) —> nRow

<**nRow**> is a numeric value that indicates the screen row where the check box's caption is displayed. The input and output value is either in coordinates or in pixels, depending on the current SET PIXEL setting. The default setting is taken from oCheckBox:Row at instantiation time.

<**lPixel**> is optional value indicating if the passed and returned value is in coordinates or pixels. If true(.T.), the column data are in pixel; if false (.F.), data are in coordinates, otherwise the current SET PIXEL status is used.

Compatibility: Access/assign is Available also in CL53.

See also: oCheckBox:CapCol, oCheckBox:Caption

oCheckBox:Caption –> cText *ACCESS*
oCheckBox:Caption := cText *ASSIGN*

<**cText**> is a string that describes the check box caption. If not redefined by :CapCol and/or :CapRow, the text is displayed at the :Row and :Col + 4 position set at instantiation time. You may specify an accelerator key: the character immediately following an ampersand (&) is treated as accelerator key. This accelerator key provides a quick and convenient mechanism for the user, to move input focus to specific check box. The user performs the selection by pressing the Alt key in combination with an accelerator key. The case of an accelerator key is ignored.

Compatibility: Available also in CL53 and VO.

See also: CheckBox:CapCol, oCheckBox:Caption

oCheckBox:Cargo –> exp *ACCESS*
oCheckBox:Cargo := exp *ASSIGN*

<**exp**> is a value of any type. The CheckBox:Cargo slot holds any user- definable data which can be retrieved later. This property is not used by the CheckBox object itself.

Compatibility: Available also in CL53.

oCheckBox:Checked –> lChecked *ACCESS*
oCheckBox:Checked := lChecked *ASSIGN*

<**lChecked**> is a logical value that indicates whether the check box is checked or unchecked. A value of true (.T.) indicates that it is checked and a value of false (.F.) indicates that it is not checked. Equivalent to oCheckBox:Buffer instance. In 3-state modus, a NIL value indicates the third UNDETERMINED state. The assign also changes CheckBox:Value and CheckBox:TextValue. Also, if the CheckBox:Value is changed, CheckBox:ValueChanged is set to TRUE.

Compatibility: Available also in VO.

See also: CheckBox:Buffer, CheckBox:Select(), CheckBox:Value

oCheckBox:ClassName() —> cText

For compatibility to Clipper's getsys.prg only. Returns fix "CHECKBOX" regardless the subclass. In FlagShip, you may also use IsObjClass() which provides you with more detailed information.

Compatibility: Available but undocumented in CL53

See also: IsObjClass() and IsObjProperty() functions

oCheckBox:Col —> nCol *ACCESS*
oCheckBox:Col := nCol *ASSIGN*
oCheckBox:Col([nCol], [lPixel]) —> nCol

<**nCol**> is a numeric value that indicates the screen column where the check box is displayed. With Access/assign, the value is either in coordinates or pixels according to the current SET PIXEL status.

<**lPixel**> is optional value indicating if the passed and returned value is in coordinates or pixels. If true(.T.), the column data are in pixel; if false (.F.), data are in coordinates, otherwise the current SET PIXEL status is used.

With terminal i/o, the <nCol> value specifies the column where the first character of oCheckBox:Stype (or :ColdBox, :HotBox) is displayed, i.e. where the left square bracket [X] of the check box representation display. The whole check box occupies 3 columns.

With GUI i/o, the check box is displayed as a widget (control) and <nCol> is the leftmost widget coordinate. To ensure the same look and feel to an application running in textual mode, and to display the widget at approx. the same screen position, the given <nCol> coordinate is automatically adapted by adding a pixel value taken from the global array element `_aGlobSetting[GSET_G_N_CHBOX_COL]` and `[GSET_G_N_CHBOX_WIDTH]` (see source in initio.prg) which may be positive or negative and are modifiable by the application.

Compatibility: Access/assign is Available also in CL53.

See also: CheckBox:Row, CheckBox{} instantiation

oCheckBox:ColdBox —> cBox *ACCESS*
oCheckBox:ColdBox := cBox *ASSIGN*

<**cBox**> is an optional string that specifies the characters to use when drawing a box around the check box when it does not have input focus. Its default value is a single line box, or the value specified in the global array `_aGlobSetting [GSET_T_C_COLDBOX]` respectively. Predefined <cBox> constants are in the box.fh file:

Constant	Description
B_SINGLE	Single line box
B_DOUBLE	Double line box
B_SINGLE_DOUBLE	Single line top/bottom, double line sides
B_DOUBLE_SINGLE	Double line top/bottom, single line sides
B_PLAIN	Use ASCII chars only

The ColdBox apply only if you assign null-string "" to oCheckBox:Style which is preferred otherwise.

Compatibility: Available also in FS5 only. This property is considered in terminal mode only and ignored otherwise.

See also: CheckBox:HotBox, CheckBox:SetFocus(), CheckBox:Style

oCheckBox:ColorSpec —> cAttrib *ACCESS*
oCheckBox:ColorSpec := cAttrib *ASSIGN*

<**cAttrib**> is a character string specifying the color attributes that are used by the display() and show() method. The string must contain four color specifiers.

Position in <cAttrib>	Applies To	Default value used from current SET COLOR
1	Check box without input focus	Unselected
2	Check box with input focus	Enhanced
3	The check box's caption	Standard
4	The check box caption's accelerator key	Background

Compatibility: Available also in CL53, This property is considered in terminal mode only and ignored otherwise.

See also: CheckBox:HasFocus, SET COLOR, SET()

oCheckBox:Destroy() —> NIL

Destroys the CheckBox object and restores the previous screen content. This method can be used when a CheckBox object is no longer needed. oCheckBox:Destroy() de-instantiates the CheckBox object and allows you to close and free any resources that were opened or created by the object, without waiting for the garbage collector. This method calls internally oCheckBox:Axit() which is the equivalence for :Destroy()

Compatibility: Available also in VO

See also: CheckBox{} instantiation

oCheckBox:Display() —> self

Show the check box, it frame/box and caption on the screen. The check box widget (control) remains invisible until you invoke oCheckBox: Display() or oCheckBox:Show(). This allows the program to set up the widget (control) correctly (with the correct size, position, and any other parameters), while avoiding the "visual noise" of changing controls. oCheckBox:Display() uses the values of the following instance variables to correctly show the list in its current context, in addition to providing maximum flexibility in the manner a check box appears on the screen: Buffer, Caption, CapCol, CapRow, Col, ColdBox or HotBox, ColorSpec, HasFocus, Row, and Style.

Compatibility: Available also in CL53

See also: CheckBox:Show()

oCheckBox:Fblock —> bBlock *ACCESS*
oCheckBox:Fblock := bBlock *ASSIGN*

<**bBlock**> is a code block or NIL. The code block callback, when present, is evaluated each time the CheckBox object receives or loses input focus. The code block receives two arguments: the object self and the current :HasFocus status, which indicates whether the check box is receiving (.T.) or losing (.F.) input focus. In GUI, the object receives focus every time the user clicks (or activates) the check box widget and looses focus when other widget is selected.

Compatibility: Available also in CL53, but Clipper does not pass any arguments to the code block, and hence cannot use generalized but object specific code blocks which needs to check the current oCheckBox:HasFocus status by itself.

See also: CheckBox:HasFocus, CheckBox:SetFocus(), CheckBox:KillFocus(), CheckBox:Sblock

oCheckBox:Handler —> bHandler *ACCESS*
oCheckBox:Handler := bHandler *ASSIGN*

<**bHandler**> is a code block or NIL. The code block, when present, is invoked from the oCheckBox:Show() method and replaces the default check box handler Available also in the <FlagShip_dir>/system/checkboxhand.prg source file. The code block receives one argument, the object self.

Compatibility: Available also in FS5 only.

See also: CheckBox:Show()

oCheckBox:HandlerSelect —> lOn *ACCESS*
oCheckBox:HandlerSelect := lOn *ASSIGN*

<**lOn**> is a logical value signaling that the checkbox switches the state either in the handler by :Select() if .T. or automatically if .F. (default). Apply for GUI mode only.

oCheckBox:HasFocus —> lFocus *ACCESS*

<**lFocus**> is a logical value indicating whether the object has input focus (TRUE) or not. In GUI, the object receives focus every time the user clicks (or activates) the widget and looses the focus when other widget is selected.

Compatibility: Available also in CL53

See also: CheckBox:KillFocus, CheckBox:SetFocus(), CheckBox:Fblock

oCheckBox:Height —> nRow *ACCESS*
oCheckBox:Height := nRow *ASSIGN*
oCheckBox:Height ([nRow], [lPixel]) —> nRow

<**nCol**> is a numeric value that indicates the height of the check box. With access and assign, the value is either in coordinates or pixels according to the current SET PIXEL status.

<**lPixel**> is optional value indicating if the passed and returned value is in coordinates or pixels. If true(.T.), the column data are in pixel; if false (.F.), data are in coordinates, otherwise the current SET PIXEL status is used.

Compatibility: Available also in FS5, apply for GUI mode only

oCheckBox:HitTest(nMouseRow, nMouseCol, [lPixel]) —> nStatus

Determines if the mouse cursor is within the region of the screen that the check box occupies.

<**nRow**> Numeric value representing the current or tested screen row position of the mouse cursor.

<**nCol**> Numeric value representing the current or tested screen column position of the mouse cursor.

<**lPixel**> If specified TRUE, the mouse coordinates are assumed in pixel. If FALSE, the mouse parameters are assumed in current row/col coordinates. If this parameter is not specified (i.e. NIL), the kind of passed mouse coordinates is determined from the current SET PIXEL setting.

<**nStatus**> Returned numeric value indicating the relationship of the mouse cursor with the check box. The constants are specified in button.fh header file.

Value	Constant	Description
>= 0	n/a	The mouse is not located in the box region
-1025	HTCAPTION	The mouse cursor is on the box's caption
-2049	HTCLIENT	The mouse cursor is on the check box

Compatibility: Available also in CL53

See also: Mrow(), Mcol()

oCheckBox:HotBox –> cBox *ACCESS*
oCheckBox:HotBox := cBox *ASSIGN*

<**cBox**> is an optional string that specifies the characters to use when drawing a box around the check box when it has input focus. Its default value is a single line box, or the value specified in the global array `_aGlobSetting [GSET_T_C_HOTBOX]` respectively. Predefined <cBox> constants are in the box.fh file:

Constant	Description
B_SINGLE	Single line box
B_DOUBLE	Double line box
B_SINGLE_DOUBLE	Single line top/bottom, double line sides
B_DOUBLE_SINGLE	Double line top/bottom, single line sides
B_PLAIN	Use ASCII chars only

The HotBox apply only if you assign null-string "" to oCheckBox:Style which is preferred otherwise.

Compatibility: Available also in FS5 only. This property is considered in terminal mode only and ignored otherwise.

See also: CheckBox:ColdBox, CheckBox:HasFocus, CheckBox:SetFocus(), CheckBox:Style, @..BOX

oCheckBox:Init([par1]...[par5) –> self

This is an internal method invoked automatically at instantiation of the CheckBox object. It is not intended to be called by the application.

Compatibility: Available also in VO

See also: CheckBox{} instantiation

oCheckBox:KillFocus() –> self

Take input focus away from a CheckBox object. Upon receiving this message, the CheckBox object redisplays itself with the :ColdBox frame and, if present, evaluates

the code block specified by :Fblock. This message is meaningful only when the CheckBox object has input focus.

Compatibility: Available also in CL53. In Clipper, the box is not drawn automatically.

See also: CheckBox:HasFocus, CheckBox:SetFocus(), CheckBox:Fblock

oCheckBox:Message —> cText *ACCESS*
oCheckBox:Message := cText *ASSIGN*

<**cText**> is a character string displayed in the windows status bar (GUI), or in the screen line specified by SET MESSAGE (in terminal mode).

Compatibility: Available also in CL53.

See also: CheckBox:Tooltip(), SET MESSAGE, oApplic:StatusMessage()

oCheckBox:Modified —> lOk *ACCESS*
oCheckBox:Modified := lOk *ASSIGN*

<**lOk**> is a logical value that is set to TRUE when the user clicks on a button, and reset to FALSE when the mouse button is released.

Compatibility: Available also in VO. Apply in GUI mode only.

oCheckBox:Row —> nRow *ACCESS*
oCheckBox:Row := nRow *ASSIGN*
oCheckBox:Row([nRow], [lPixel]) —> nRow

<**nRow**> is a numeric value that indicates the screen row where the check box is displayed. With Access/assign, the value is either in coordinates or pixels according to the current SET PIXEL status.

<**lPixel**> is optional value indicating if the set/get value is in coordinates or pixels. If true(.T.), the row data are in pixel; if false (.F.), data are in coordinates, otherwise the current SET PIXEL status is used.

With terminal i/o, the <nRow> value specifies the column where the three characters of check box [X] display, see also :Style for details.

In GUI i/o mode, the check box is displayed as a widget (control) and <nRow> is the topmost widget coordinate when the row is specified in pixel. If the <nRow > is given in coordinates, the widget position is automatically adapted, to ensure the same look and feel to an application running in textual mode, and to display the widget at approx. the same screen position. The topmost widget position is then calculated from the given <nRow> coordinate minus the current line height plus a value taken from the global array element `_aGlobSetting [GSET_G_N_CHBOX_ROW]` and `_aGlobSetting [GSET_G_N_CHBOX_HEIGHT]` which is either positive or negative number of pixels.

Compatibility: Access/assign is Available also in CL53.

See also: CheckBox:Col, CheckBox{} instantiation

oCheckBox:Sblock —> bBlock *ACCESS*
oCheckBox:Sblock := bBlock *ASSIGN*

<**bBlock**> is an optional code block or NIL. The code block callback, when present, is evaluated each time the CheckBox object's state changes. The name "Sblock" refers to state block. The code block receives two arguments: 1) the object self, and 2) the check status, i.e. the content of oCheckBox:Buffer.

Compatibility: Available also in CL53, but Clipper does not pass any arguments to the code block; it hence cannot use generalized but object specific code blocks which must extract the required values from the known object by itself.

See also: CheckBox:Buffer, CheckBox:Fblock

oCheckBox:Select([lOnOff]) —> lOnOff

<**lOnOff**> is a logical value that indicates whether the check box should be checked or not. Set to true (.T.) to check the box or false to uncheck the box. If omitted, the check box state will toggle to its opposing state. Considered only if the box has input focus.

The check box state is typically changed when the space bar is pressed or the mouse's left button is pressed when its cursor is within the check box's region of the screen. FlagShip's default handler used in oCheckBox:Show() also accepts +,T,t,Y,y keys to set the status ON, and -,F,f,N,n keys to set the check box status OFF, and space or "x" key to toggle the status.

Compatibility: Available also in CL53

See also: CheckBox:Buffer

oCheckBox:SetFocus() —> self

Set input focus to a CheckBox object. Upon receiving this message, the CheckBox object redisplays itself with the :HotBox frame and, if present, evaluates the code block specified by :Fblock. This message is meaningful only when the CheckBox object does not have input focus. In GUI, the object receives focus also every time the user clicks (or activates) the widget.

Compatibility: Available also in CL53. In Clipper, the box is not drawn automatically.

See also: CheckBox:HasFocus, CheckBox:KillFocus(), CheckBox:Fblock, CheckBox:HotBox

oCheckBox:Show() —> self

This method activates either the default or user specific input handler. Is show the check box and its caption on the screen, activate focus, wait for user input and set the check box status in :Buffer accordingly, then kill the focus. The default handler is available also in the <FlagShip_dir>/system/checkboxhand.prg source file and is equivalent to a manual code sequence

```
oCheckBox:Display()
oCheckBox:SetFocus()
key := InkeyTrap(0)                      // considers SET KEY
do case
case chr(key) $ " xX"
   oCheckBox:Select(!oCheckBox:Buffer)   // toggle on/off
case chr(key) $ "+yYtT"
   oCheckBox:Select(.T.)
case chr(key) $ "-nNfF"
   oCheckBox:Select(.F.)
endcase
oCheckBox:KillFocus()
```

You may assign your own handler by the oCheckBox:Handler property.

Compatibility: Same named method is available also in VO which returns NIL

See also: CheckBox:Display(), CheckBox:Handler

oCheckBox:Style —> cStyle *ACCESS*
oCheckBox:Style := cStyle *ASSIGN*

<**cStyle**> is a character string that indicates the delimiter characters that are used by the check box's Display() method. The string must contain five characters. The first is the left delimiter, the 2nd is the checked indicator, the 3rd is the unchecked indicator, the 4th character is the right delimiter and the 5th character for the undetermined check box state. The default style is pre-defined in the global array element `_aGlobSetting[GSET_T_C_CHBOX_STYLE]` containing `"[X ]?"` at start-up; it may be re-defined by a simple assignment later.

When you assign null-string `""` to oCheckBox:Style, the oCheckBox:ColdBox and :HotBox is used as a frame around the check box region, and the 2nd, 3rd and 5th character in the global setting apply for the checked, unchecked, and undetermined indicator.

Compatibility: Considered in terminal mode only, ignored in GUI. Available also in CL53 whose default is a string of four characters containing `"[" + chr(251) + " ]"`. Neither the redefinition of defaults, nor the ColdBox and HotBox instances are available in Clipper.

See also: CheckBox:ColdBox, CheckBox:HotBox, CheckBox:Display(), CheckBox:Show()

oCheckBox:ToolTip —> cText *ACCESS*
oCheckBox:ToolTip := cText *ASSIGN*

<**cText**> is a string representing the displayed tool tip, i.e. a short info message which pop up's when the mouse is over the check box.

Compatibility: Available in FS5 only, apply for GUI, ignored otherwise

See also: CheckBox:Message

oCheckBox:TypeOut —> lVal *ACCESS*

<**lVal**> is a value always containing false (.F.). It is not used by the CheckBox object and is only provided for compatibility with the other GUI control classes.

Compatibility: Available also in CL53

oCheckBox:Value —> exp *ACCESS*
oCheckBox:Value := exp *ASSIGN*

<**exp**> contains TRUE (.T.) if the check box is in the checked (ON) state, FALSE (.F) if it is in the unchecked state (OFF) and NIL in the third UNDETERMINED state.

Compatibility: Available also in VO

oCheckBox:ValueChanged —> lStat *ACCESS*
oCheckBox:ValueChanged := lStat *ASSIGN*

<**lStat**> contains TRUE or FALSE, depending on the state of the check box. For 2-state check boxes, it is always TRUE. If a 3-state check box is in the third UNDETERMINED state, <lStat> is FALSE.

Compatibility: Available also in VO

See also: CheckBox:Value

oCheckBox:Width —> nCol *ACCESS*
oCheckBox:Width := nCol *ASSIGN*
oCheckBox:Width ([nCol], [lPixel]) —> nCol

<**nCol**> is a numeric value that indicates the width of the check box. With Access and assign, the value is either in coordinates or pixels according to the current SET PIXEL status.

<**lPixel**> is optional value indicating if the passed and returned value is in coordinates or pixels. If true(.T.), the column data are in pixel; if false (.F.), data are in coordinates, otherwise the current SET PIXEL status is used.

Compatibility: Available in FS5 only, apply for GUI, ignored otherwise

ComboBox Class

The ComboBox Class is a special case of the ListBox class. It creates and manages combo boxes.

Combo boxes display a list of items or choices to the user. The difference to the list box is, that only one list item is displayed at a time while the whole list is poped-up upon request.

As with other GUI classes in FlagShip, the general ComboBox class is internally inherited by three different sub-classes: _gComboBox for GUI based application, _tComboBox for terminal/text based mode, and _bComboBox for basic i/o mode, all defined in the boxclass.fh header file. The proper class, corresponding to the used i/o mode, is set either at compile time with the compiler switch "-io=g|t|b", or latest at run-time depending on the currently used environment.

Note: in the basic i/o mode, only a rough combo box functionality is simulated by the sequential in/output.

ComboBox Class Index

Class ComboBox

Inherits from:	ListBox
Inherited by:	- (none)
Class prototype:	boxclass.fh
Defines:	box.fh

AddItem()	METHOD	Add (append) a new item to a combo box
Bitmap	ACC/ASS	Display bitmap as combo box item
Bottom	ACC/ASS	Bottommost screen row of the box
Buffer	ACC	Position in the list of the selected item
CapCol	ACC/ASS	Screen column of the combo box's caption
CapRow	ACC/ASS	Screen row of the combo box's caption
Caption	ACC/ASS	String that describes the combo box caption
Cargo	ACC/ASS	A user value of any type
ChangeSelected()	METHOD	Change a range of items in a multiple selection
ClassName	METHOD	For compatibility to Clipper's getsys.prg only
Clear()	METHOD	Clear (delete) all items in a combo box
ClearSelection()	METHOD	Clear a multiple selection combo box
Close()	METHOD	Closes the combo box
ColdBox	ACC/ASS	Frame of combo box without focus
ColorSpec	ACC/ASS	Color attributes
CurrentItem	ACC/ASS	String representing the displayed ComboBox item
CurrItemNo	ACC/ASS	Numeric value indicating the selected item
CurrentText	ACC/ASS	Fix ""

DeleteItem()	METHOD	Remove an item from a combo box
DelItem(p1)	METHOD	Remove an item from a combo box
DeselectItem()	METHOD	Turn off the selection of a specified item
Destroy()	METHOD	Destroys the ComboBox object
Display()	METHOD	Show the combo box and its caption on the screen
DropDown	ACC	Always .T.
Exec()	METHOD	Process user input, same as :Show()
Fblock	ACC/ASS	Code block evaluated at receiving/loosing focus
FillUsing()	METHOD	Data server/dictionary driver
FindItem()	METHOD	Search a combo box for a specified item
FindText()	METHOD	Search a combo box for a specified string
FirstSelected()	METHOD	Position of the 1st item in a multiple selection
Font	ACC/ASS	Font object used to display the combo box items
GetData()	METHOD	Get the data portion of a combo box item
GetItem()	METHOD	Get the item property
GetItemValue()	METHOD	Same as GetData()
GetText(p1)	METHOD	Get the item text
HasFocus	ACC	Indicates whether the object has input focus
HitTest()	METHOD	Determines if the mouse cursor is within the box
HotBox	ACC/ASS	Frame of combo box with focus
InputBlock	ACC/ASS	CodeBlock for default/user keyboard handler
InsItem()	METHOD	Insert a new item to a combo box
IsOpen	ACC	Indicator whether the combo box widget is visible
ItemCount	ACC	Number of items in the list
KillFocus()	METHOD	Take input focus away from a ComboBox object
Left	ACC/ASS	Leftmost screen column of the box
ListFiles()	METHOD	Fill a combo box with the names of matching files
Message	ACC/ASS	String displayed in the windows status bar
Modified	ACC/ASS	Ignored.
NextItem()	METHOD	Skip to the next available item
NextSelected()	METHOD	Skip to the next selected item
Open()	METHOD	Opens the combo box (drop-down box)
PrevItem()	METHOD	Skip to the previous available item
Right	ACC/ASS	Rightmost screen column of the box
Sblock	ACC/ASS	Code block evaluated at user selection
Scroll()	METHOD	Scrolls the contents of a combo box up or down
Select()	METHOD	Change the selected item in a list
SelectedCount	ACC	Number of items selected in a multiple selection
SelectedFile	ACC	Selected file filled by :ListFiles()
SelectItem()	METHOD	Change the selected item in a list
SetData()	METHOD	Change the property of an available item
SetFocus()	METHOD	Set input focus to a ComboBox object
SetItem()	METHOD	Replaces the item property
SetText()	METHOD	Change/replace the displayed text of item
SetTop()	METHOD	Move a specified item to the top of the combo box
Show()	METHOD	Show the combo box and its caption on the screen
TextValue	ACC/ASS	String representing the displayed ComboBox item

ToolTip	ACC/ASS	Short pop-up info message
Top	ACC/ASS	Topmost screen row of the box
TopItem	ACC/ASS	Position of the first visible item
TypeOut	ACC/ASS	Indicator whether the list contains any items
Value	ACC/ASS	Any value associated with the specified item
ValueChanged	ACC/ASS	Indicator representing the status of :Value
Vscroll	ACC/ASS	Ignored in FlagShip

ComboBox Class Instantiation

oCmbBox := [_g|_t|_b]ComboBox { [nR1],[nC1], [nR2],[nC2], [lPixel] } *[1]*
oCmbBox := [_g|_t|_b]ComboBoxNew ([nR1],[nC1], [nR2],[nC2], [lPixel]) *[2]*

oCmbBox := ListBox ([nR1], [nC1], [nR2], [nC2], .T., [lPixel]) *[3]*
oCmbBox := ComboBox { [oOwn], [nResrc] } *[4]*
oCmbBox := ComboBox { [oOwn], [nId], [oPoint], [oDim], [nStyle] } *[5]*

Any of the above syntax instantiate new combo box object. Syntax [1] and [2] are standard FlagShip and should be preferred. Syntax [3] is supported for compatibility to Clipper 5.3, [4] and [5] is supported for compatibility to VO.

The combo box widget (control) remains invisible until you invoke oCmdBox:Show() or oCmdBox:Display(). This allows the program to set up the control correctly (with the correct size, position, and any other parameters), while avoiding the "visual noise" of changing controls. Arguments:

<**nR1**> topmost row in coordinates or pixel, optional. If not specified, 0 is the default

<**nC1**> leftmost column in coordinates or pixel, optional. If not specified, 0 is the default

<**nR2**> bottom row in coordinates or pixel, optional. If not specified, MaxRow() is default

<**nC2**> rightmost column in coordinates or pixel, optional. If not specified, MaxCol() is the default

<**lPixel**> if true(.T.), the row and column data are in pixel; if false (.F.), data are in row/col coordinates, otherwise the current SET PIXEL is used.

<**oOwn**> owner object of the combo box, optional. Default is the oApplic object.

<**nResrc**> resource ID of the combo box

<**nStyle**> style constant of the combo box according to box.fh, optional. If not given, LBS_STANDARD is the default.

Compatibility: Available also in CL53 (syntax 3) and VO (syntax 4 and 5).

See also: oListBox:Destroy()

ComboBox Class Properties

All properties of the ListBox class are available also for the ComboBox class. Refer to description of ListBox class for details.

Error Class

Generally ERROR objects contain information for the error handler routine. It can be compared to an array, however one with fixed elements. The ERROR object can also serve as an information carrier for the RECOVER part of a program interrupt structure. See BEGIN SEQUENCE.

The ERROR object contains only information (instances), no executable methods are available.

1. Error handling strategy

FlagShip offers different types of handling errors and exceptions:

- Program enquiry and correction of easily recoverable errors (e.g. through prior allocation with a default value) in an IF.. ..ELSE...ENDIF structure,
- Program generation and treatment of exceptional statuses and interrupts with the BEGIN SEQUENCE .. BREAK ... RECOVER ... END structure (see section CMD).
- Error handling at lower system level with ERROR blocks. These are generally used to handle errors in the FlagShip library, e.g. on wrong parameters, incorrect data types, input/output errors etc. The programmer can extend, modify, or even replace the default handling by using additional user-defined functions.

In the following chapter low-level error treatment is described. For a technical treatment of exception statuses, see sections LNG and CMD.

2. Error Blocks and Functions

When FlagShip asserts an error, it displays it on the screen, and a user action (Continue, Abort etc.) is prompted. See section FSC.4. The programmer can supervise the entire error action as all the relevant files are supplied in the source code.

1. If the library module determines an error, it generates the relevant error number (according to FSerrors.h) and a text error description.
2. This information is passed on to a low-level error function using parameters (contained in the file FSerror.prg).
3. The xxx_error function generates and occupies an ERROR object with information. Then either the standard or the user-defined high-level error handling routine is activated by the last generated error code block.
4. The error treatment routine evaluates the information from the ERROR object which has been passed on, and then performs the relevant treatment. Generally the error is first displayed on the screen, which enables the user to react appropriately (Cancel, Ignore, Callstack, Debugger). With more simple errors the program execution can be continued at the next source code instruction.

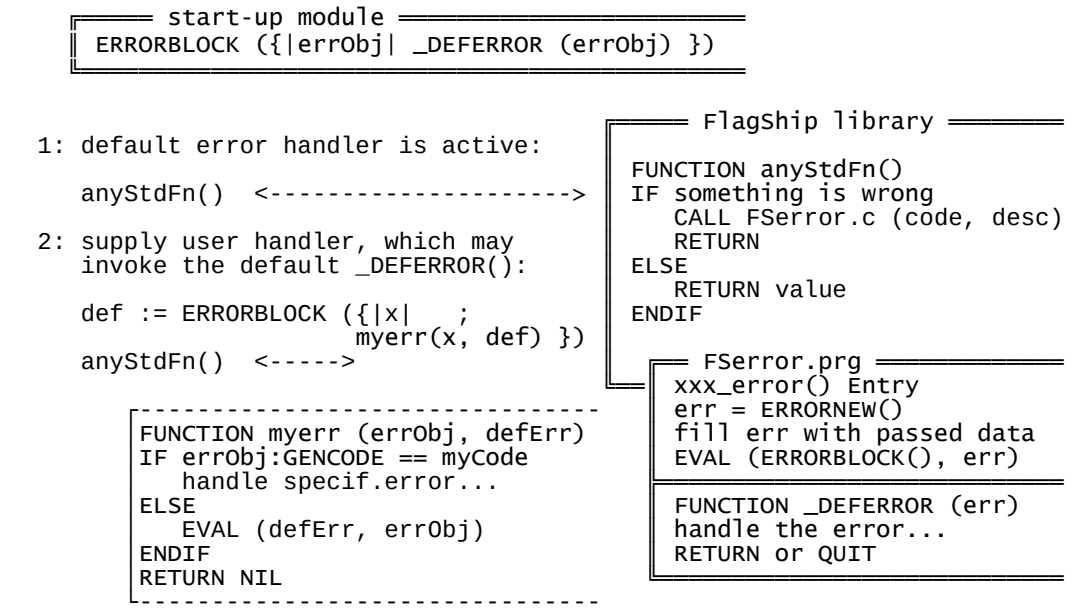
In order to be able to stagger the error handling routines as required, they are called via a special code block, not directly. With the function ERRORBLOCK() this code block is passed to the error system.

Before starting the program execution, FlagShip pre-generates a code block by calling the function _DEFERROR() with ERRORBLOCK(). This function executes the default error handling. If an additional or deviating error handling is to be implemented at any position in the application, the programmer can:

a. save the current error code block in a variable by calling the function ERRORBLOCK(),
b. pass a new code block as parameter to the function ERRORBLOCK(),
c. then restore the original error handling block.

If an error occurs in the sequence b...c, the user-specific function (more precisely: the contents of the code block) is executed, not the default handling. If the error is not the error, which requires specific handling, the UDF can have further treatment executed by the predefined error function.

The whole error handling can be represented as follows:



Technically a simple error handling for opening a database can look as follows:

```
STATIC defaulterr

USE nonexistent           // i/o error generated by default handler
defaulterr := ERRORBLOCK()         // retrieve the default handler
ERRORBLOCK( {|err| MyHandler (err, defaulterr) } )   // set new one

USE nonexistent                  // now, MyHandler() is called
ERRORBLOCK (defaulterr)          // re-activate the default handler

#include "error.fh"

FUNCTION MyHandler (errObj, oldHandler)
************************************
LOCAL getlist {}, file, dir, ii
IF errObj:GENCODE == EG_OPEN     .and.;  // see error.fh
   errObj:OSCODE  == 101         .and.;  // see FSerrors.h
   errObj:CANRETRY               .and.;  // is RETRY possible ?
   errObj:OPERATION == "MYUSE" .and.;    // see ERRORNEW() exampl
   LEN(errObj:FILENAME) > 0              // is file name supported?
   file := errObj:FILENAME               // full file name
   IF ALERT ("File " + file + " not found", ;
             "Abort", "Specify directory") != 2
      QUIT
   ENDIF
```

```
   ii   = RAT ("/", file)
   dir  = IF (ii > 0, SUBSTR (file, 1, ii), "")
   file = IF (ii > 0, SUBSTR (file, ii+1), file)
   dir  = PADR (dir, 250)
   @ 0,0 SAY "New directory:" GET dir PICTURE "@S50"
   READ
   IF LASTKEY() != 27
      dir := ALLTRIM (dir)
      IF (RIGHT (dir, 1) != "/"
         dir += "/"
      ENDIF
      errObj:FILENAME := dir + file       // new file specification
      RETURN .T.                          // RETRY
   ENDIF
ENDIF
RETURN EVAL (oldHandler, errObj)          // invoke default handler
```

Note: If you use the alternative, smaller error handler FSerror.c (i.e. link it), no error objects are generated by default. The error messages are displayed in a window that can be repositioned as required. Due to identity of names, the FSerror.prg should never be compiled in the directory <FlagShip_dir>/system. If you do this, the alternative C driver FSerror.c is overwritten.

When program writing, it is absolutely crucial to ensure that the error handling routine does not result in an endless loop by triggering the same error.

In the program <FlagShip_dir>/system/FSerror.prg a similar error object is generated on input/output error:

```
err:CARGO         := NIL
err:ARGS          := ""
err:CANDEFAULT    := .F.
err:CANRETRY      := .F.
err:CANSUBSTITUTE := .F.
err:DESCRIPTION   := "file write error;Write error;"
err:FILENAME      := ""
err:GENCODE       := 104
err:OPERATION     := ""
err:OSCODE        := 0
err:SEVERITY      := ES_IOERR
err:SUBCODE       := 0
err:SUBSYSTEM     := "BASE"
err:TRIES         := 0
```

Note: for even more comfortable programming, the instance variables will be even more precisely defined in the library of the next FlagShip version and many functions will support the full and half-automatic RETRY.

ErrorNew ()

Syntax 1:

obj = ERRORNEW ()

Syntax 2:

obj = ERROR { }

Purpose:

Creates a new, empty ERROR object.

Arguments:

none.

Returns:

<**obj**> is the new allocated ERROR object, usually assigned to a regular FlagShip variable.

Description:

ERRORNEW() creates a new object, which is used to carry information during program execution.

Example:

Checks if a file is available and traps to the error handler if not. The newly installed error handler also remains active during execution of the USE command. Prior to returning from the function, the previous error handler is restored.

```
#include "error.fh"

IF .not. myuse ("anydatabase")
   QUIT
ENDIF

FUNCTION myuse (file)
*********************
STATIC oldhandle                    // save current handle
LOCAL  myerr := ERRORNEW()          // create object
LOCAL  block := {|err| myhandler(err, oldhandle)}
oldhandle := ERRORBLOCK(block)      // install new handle

myerr:ARGS        := {file}
myerr:CANDEFAULT  := .F.
myerr:CANRETRY    := .T.
myerr:CANSUBSTIT  := .F.
myerr:CARGO       := NIL
myerr:DESCRIPTION := "File not found"
myerr:FILENAME    := file
myerr:GENCODE     := EG_OPEN
myerr:OPERATION   := "MYUSE"
myerr:OSCODE      := 101
myerr:SEVERITY    := ES_ERROR
myerr:SUBCODE     := 0
myerr:TRIES       := 0
```

```
IF ! FILE(file)
   DO WHILE .T.
      myerr:TRIES++
      IF EVAL (ERRORBLOCK(), myerr)   // trap error handler
         file := myerr:FILENAME
         IF FILE(file)
            EXIT
         ENDIF
      ELSE
         ERRORBLOCK (oldhandle)
         RETURN .F.
      ENDIF
   ENDDO
ENDIF

USE (file) SHARED                     // other errors poss.
ERRORBLOCK (oldhandle)                // restore old handle
RETURN .T.
```

Classification:

programming

Class:

ERROR class, prototyped in <FlagShip_dir>/include/errclass.fh

Compatibility:

Available in FS4, C5 and VO. The alternative syntax 2 and the possibility of inheriting it into an own subclass is available in FlagShip only.

Related:

ERRORBLOCK()

Error Class Properties

err:ARGS **Access/Assign**

Contains an array of the arguments supplied to an operator or function when an argument error occurs. For other types of error, err:ARGS contains a NIL value.

err:CANDEFAULT **Access/Assign**

Contains a logical value. TRUE indicates that the subsystem (operation, function) can perform default error recovery for the error condition. Availability of default handling and the actual default recovery strategy depends on the subsystem and the error condition. The minimum action in the error handle is simply to ignore the error condition and return FALSE which sets the default recovery. If err:CANDEFAULT is TRUE, err:CANSUBSTITUTE must be FALSE.

err:CANRETRY **Access/Assign**

Contains a logical value. TRUE indicates that the subsystem (operation, function) can retry the execution itself. Availability of retry depends on the subsystem and the error condition. To invoke the default retry, return TRUE from the error handler. If the automatic retry is not available, you may post the originate statement in a program loop, e.g. USE... while NETERR() USE... enddo while ! RLOCK() enddo If err:CANRETRY is TRUE, err:CANSUBSTITUTE must be FALSE.

err:CANSUBSTITUTE **Access/Assign**

Contains a logical value. TRUE indicates that the subsystem (operation, function) can substitute a new result when returning from the error handler. Argument errors and certain other simple errors allow the error handler to substitute a new result value for the failed operation. If err:CANSUBSTITUTE is TRUE, err:CANDEFAULT and err:CANRETRY must be FALSE.

err:CARGO **Access/Assign**

Contains a value of any data type supplied by the user. Not used by the Error system itself.

err:DESCRIPTION **Access/Assign**

Contains a character string that describes the error condition in textual form. A null-string "" indicates that the subsystem does not provide a printable description for the error.

err:FILENAME *Access/Assign*

Contains a character value representing the file name originally used to perform an input/output request. A null-string "" indicates that the subsystem does not provide information on the file name.

err:GENCODE *Access/Assign*

Contains a numeric value representing a generic error code according to the EG_xx manifests in the <FlagShip_dir>/include/error.fh (and the FSerrors.h) files. Zero indicates that the error condition is specific to the subsystem and does not correspond to any of the generic error codes.

err:OPERATION *Access/Assign*

Contains a character string representing the current operation or function name which caused the error. For undefined variables or functions, it contains the name of the variable or function. A null- string "" indicates that the subsystem does not provide information on the operation.

err:OSCODE *Access/Assign*

Contains a numeric value representing the operating system error code according to DOSERROR() and the EX_xx manifests in the <FlagShip_dir>/ include/error.fh file. Zero indicates that the error condition was not caused by system call.

err:SEVERITY *Access/Assign*

Contains a numeric value indicating the severity of the error condition. Following manifests are defined in the <FlagShip_dir>/ include/error.fh file:

ES_FTLERR	Fatal error, requires immediate termination of the application.
ES_IOERR	Input/output error; continuation may be possible, but the operation was not performed.
ES_RTERR	Run-time error; continuation may be possible, but the operation was not performed.
ES_INTERR	Internal error, probably caused by the continuation of previous i/o or run-time error.

err:SUBCODE *Access/Assign*

Contains a numeric value representing the subsystem-specific error code. May also be used for other purposes, such as a line number or additional information. Zero indicates that the subsystem does not provide the information.

err:SUBSYSTEM *Access/Assign*

Contains a character string representing the name of the subsystem generating the error. Errors indicated by FlagShip itself may set err:SUBSYSTEM to "BASE". For errors caused by the replaceable database driver (RDD), the name of the driver is set. May be used for other purposes, such as indicating an internal driver name. Null-string "" indicates that the subsystem does not provide the information.

err:TRIES *Access/Assign*

Contains a numeric value representing the number of times the failed operation has been attempted. When err:CANRETRY is TRUE, the value of err:TRIES can be used to limit the number of retry attempts. Zero indicates that the subsystem does not track the number of times the operation has been tried.

ErrorBox Class

FlagShip provides several GUI classes used for dialog communication. Apart from the general ALERT() function, there is available special MessageBox class and its sub-classes named TextBox, InfoBox, ErrorBox and WarningBox

ErrorBox Class Index

Class ErrorBox

Inherits from:	MessageBox
Inherited by:	- (none)
Class prototype:	dialogclass.fh
Defines:	dialog.fh, box.fh
Alternative:	standard function ALERT() or ERRBOX()

BoxText	ASSIGN	Sets or redefines the displayed text
BoxType	ASSIGN	Type of the message box, i.e. the used icon
Buttons	ACC/ASS	Type and caption of the used push buttons
Caption	ASSIGN	Caption (title) of the message box
ColorSpec	ACC/ASS	Color specification for terminal i/o
DefButton	ASSIGN	Assigns one button as default
Font	ACC/ASS	Sets or redefines the used font
GuiColor	ACC/ASS	Color specification for GUI
HotBox	ACC/ASS	Box frame for terminal i/o
Exec()	METHOD	Display the message box and wait for user action
Show()	METHOD	Equivalent to oBox:Exec()
Handle()	METHOD	for compatibility purposes only
TimeOut	ACC/ASS	time-out in seconds
Type	ACC/ASS	for compatibility purposes only

The ErrorBox is an usual MessageBox class with pre-defined properties

```
oBox:BoxType := MBOX_ERROR
oBox:BoxText := if(empty(cText), "Error !", cText)
oBox:Caption := "Error"
```

ErrorBox Class Instantiation

Syntax 1:

```
oBox := ErrorBox {[oOwner], [cText]}
```

Syntax 2:

```
oBox := ErrorBoxNew ([oOwner], [cText])
```

Syntax 3:

```
oBox := MessageBox {[cText], MBOX_ERROR, [ncaButt],
            [iDefBut], [cTitle], [nOwner],
            [oFont], .T. }
```

Any of the above syntax instantiate new Error box object, optionally with the given caption, text, push-button(s) and font. The default message box is modal. This means, the application is suspende until the user acknowledges the message. Arguments (all optional):

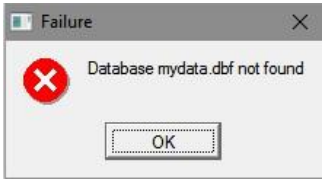
<**oOwner**> is supported for compatibility purposes to VO and is ignored.

<**cText**> the displayed text, i.e. the information to be printed in the message box. You may spilt the text in several lines by using either `";"` (semicolon) or LF=chr(10) character(s). If you need to print semicolon, use `"\;"`. To insert an empty line, use `"; ;"` or `";;"`. If the argument is omitted, no text is displayed. Assignable also by oBox:BoxText. See the oBox:BoxText property for details about alignment and HTML text formatting.

The box:Caption is pre-set to "Error", but may be re-defined by assigning any other text to :Caption.

Example:

```
oBox := ErrorBox{NIL, "Database " + myFile + " not found"}
oBox:Caption := "Failure"
oBox:Exec()
```



ErrorBox Class Properties

All the class properties are equivalent to MessageBox class; please refer there for further details.

InfoBox Class

FlagShip provides several GUI classes used for dialog communication. Apart from the general ALERT() function, there is available special MessageBox class and its sub-classes named TextBox, InfoBox, ErrorBox and WarningBox

As with other GUI classes in FlagShip, the general InfoBox class is internally inherited by three different sub-classes: _gInfoBox for GUI based application, _tInfoBox for terminal/text based mode, and _bInfoBox for basic i/o mode, all defined in the dialogclass.fh header file. The proper class, corresponding to the used i/o mode, is set either at compile time with the compiler switch "-io=g|t|b", or latest at run-time depending on the currently used environment.

InfoBox Class Index

Class InfoBox

Inherits from:	MessageBox
Inherited by:	- (none)
Class prototype:	dialogclass.fh
Defines:	dialog.fh, box.fh
Alternative:	standard function ALERT() or INFOBOX()

BoxText	ASSIGN	Sets or redefines the displayed text
BoxType	ASSIGN	Type of the message box, i.e. the used icon
Buttons	ACC/ASS	Type and caption of the used push buttons
Caption	ASSIGN	Caption (title) of the message box
ColorSpec	ACC/ASS	Color specification for terminal i/o
DefButton	ASSIGN	Assigns one button as default
Font	ACC/ASS	Sets or redefines the used font
GuiColor	ACC/ASS	Color specification for GUI
HotBox	ACC/ASS	Box frame for terminal i/o
Exec()	METHOD	Display the message box and wait for user action
Show()	METHOD	Equivalent to oBox:Exec()
Handle()	METHOD	for compatibility purposes only
TimeOut	ACC/ASS	time-out in seconds
Type	ACC/ASS	for compatibility purposes only

The InfoBox is an usual MessageBox class with pre-defined properties

```
oBox:BoxType := MBOX_INFO
oBox:Caption := if(empty(cTitle), "InfoBox", cTitle)
```

InfoBox Class Instantiation

Syntax 1:

```
oBox := InfoBox {[oOwner], [cTitle], [cText]}
```

Syntax 2:

```
oBox := InfoBoxNew ([oOwner], [cTitle], [cText])
```

Syntax 3:

```
oBox := MessageBox {[cText], MBOX_INFO, [ncaButt],
          [iDefBut], [cTitle], [nOwner],
          [oFont], .T. }
```

Syntax 4:

```
InfoBox (ctext, [cTitle]) -> nSelConstant
```

Any of the above syntax instantiate new info box object, optionally with the given caption, text, push-button(s) and font. The default message box is modal. This means, the application is suspended until the user acknowledges the message. The InfoBox() function is supported for Clipper compatibility - note the swapped parameters. Arguments (all optional):

<**oOwner**> is supported for compatibility purposes to VO and is ignored.

<**cText**> the displayed text, i.e. the information to be printed in the message box. You may split the text in several lines by using either ";" (semicolon) or LF=chr(10) character(s). If you need to print semicolon, use "\;". To insert an empty line, use "; ;" or ";;". If the argument is omitted, no text is displayed. Assignable also by oBox:BoxText. See the oBox:BoxText property for details about alignment and HTML text formatting.

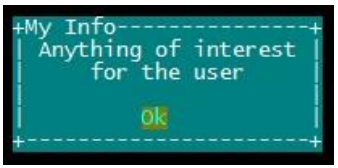
<**cTitle**> is a caption of the message box, i.e. text displayed in the title bar of the box. If omitted, the application name plus "Info/Warning/Error" text is used. Assignable also by oBox:Caption

Example 1:

```
oBox := InfoBox{NIL, "My Info","Anything of interest;for the user"}
oBox:Exec()
```

Example 2:

```
InfoBox("Anything of interest;for the user", "My Info")
```



InfoBox Class Properties

All the class properties are equivalent to MessageBox class, please refer there for further details.

The instantiation and the class properties are equivalent to MessageBox class, please refer there for further details.

MessageBox Class

FlagShip provides several GUI classes used for dialog communication. Apart from the usual @SAY..GET..READ, ACHOICE() etc., there is available special MessageBox class and its sub-classes named TextBox, InfoBox, ErrorBox and WarningBox. They extend the general ALERT() function for additional functionality.

MessageBox Class

A message box is a small window that displays a caption, a message, an icon (chosen from a predefined set of icons), and up to three push buttons (selected from a variety of predefined combinations). It provides an easy alternative to the dialog window when all you require from the user is a simple response. Message boxes require no sizing, positioning, or event handling. In addition, message boxes can be application modal (the process cannot continue until the user has acknowledged the message box), or it can be modal in relation to its owner window (process cannot continue in its own window until the user has acknowledged the message box).

The standard function ALERT() uses the MessageBox class in GUI mode. For your convenience, and for a backward VO compatibility, there are also sub-classes TextBox, InfoBox, ErrorBox and WarningBox available.

As with other GUI classes in FlagShip, the general MessageBox class is internally inherited by three different sub-classes: _gMessageBox for GUI based application, _tMessageBox for terminal/text based mode, and _bMessageBox for basic i/o mode, all defined in the dialogclass.fh header file. The proper class, corresponding to the used i/o mode, is set either at compile time with the compiler switch "-io=g|t|b", or latest at run-time depending on the currently used environment.

Note: in the basic i/o mode, only a rough MessageBox functionality is simulated by the sequential in/output.

MessageBox Class Index

Class MessageBox

Inherits from:	- (none)
Inherited by:	InfoBox, ErrorBox, TextBox, WarningBox
Class prototype:	dialogclass.fh
Defines:	dialog.fh, box.fh
Alternative:	standard function ALERT()

BoxText	ASSIGN	Sets or redefines the displayed text
BoxType	ASSIGN	Type of the message box, i.e. the used icon
Buttons	ACC/ASS	Type and caption of the used push buttons
Caption	ASSIGN	Caption (title) of the message box
ColorSpec	ACC/ASS	Color specification for terminal i/o
DefButton	ASSIGN	Assigns one button as default
Exec()	METHOD	Display the message box and wait for user action
Font	ACC/ASS	Sets or redefines the used font
GuiColor	ACC/ASS	Color specification for GUI
HotBox	ACC/ASS	Box frame for terminal i/o
Image	ACC/ASS	Set/get the file name of image pixmap
Show()	METHOD	Equivalent to oBox:Exec()
Handle()	METHOD	for compatibility purposes only
TimeOut	ACC/ASS	time-out in seconds
Type	ACC/ASS	for compatibility purposes only

MessageBox Class Instantiation

Syntax 1:

```
oBox := MessageBox ([cText], [nType], [ncaButt],
           [iDefBut], [cTitle], [nOwner], [oFont],
           [lModal] )
```

Syntax 2:

```
oBox := MessageBox {[cText], [nType], [ncaButt],
           [iDefBut], [cTitle], [nOwner], [oFont],
           [lModal] }
```

Any of the above syntax [1] and [2] instantiate new message box object, optionally with the given caption, text, push-button(s) and font. The default message box is modal. This means, the application is suspended until the user acknowledges the message. Arguments (all optional):

<**cText**> the displayed text, i.e. the information to be printed in the message box. You may split the text in several lines by using either ";" (semicolon) or "\n" or LF=chr(10) character(s). If you need to print semicolon, use "\;". To insert an empty line, use "; ;" or ";;". If the argument is omitted, no text is displayed. This value is also assignable by oBox:BoxText. See the oBox:BoxText description for details about alignment and HTML text formatting.

<**nType**> is a type of the message box. One of the constants MBOX_INFO, MBOX_WARNING, MBOX_ERROR, MBOX_QUEST, MBOX_NONE or MBOX_USER defined in dialog.fh, specifying the type of the box and the used icon. If omitted, MBOX_INFO is the default. Assignable also by oBox:BoxType. When MBOX_USER is specified, also oBox:Image assignment is required.

<**ncaButt**> is a type and caption of the used push buttons.

- Either a numeric constant MBOX_OK, MBOX_YES, MBOX_NO, MBOX_ABORT, MBOX_CANCEL, MBOX_RETRY, MBOX_IGNORE defined in dialog.fh,
- or up to three of these numeric constants added to each other,
- or a user defined string displayed in the push button. Note: the accelerator key character is escaped by an ampersand (&). If you need to display the ampersand itself, specify two ampersands (&&) in the string.
- or an array of numeric and/or string elements specifying each of the push buttons. Up to three buttons are considered, the rest (i.e. array element 4 and greater) is ignored.

If omitted or of an invalid type or value, MBOX_OK is the default. Assignable also by oBox:Buttons

<**iDefBut**> Default button (1 to 3) assigned to the RETURN key. If not specified or out of range, no default button is set (the default) so the user needs to press TAB

or cursor key before confirming by RETURN. Of course, this has no effect on the choice via mouse click. Assignable also by oBox:DefButton

<**cTitle**> is a caption of the message box, i.e. text displayed in the title bar of the box. If omitted, the application name plus "Info/Warning/Error" text is used. Assignable also by oBox:Caption

<**nOwner**> Owner of the message box. If omitted, 0 (zero) or when SDI mode is used, the parent of the message box is the application window. In MDI mode, giving a value grater one specifies the owning MDI window for which the message is modal.

<**oFont**> specifies the used font, when it should be different from the default, which is oApplic:FontWindow. See details in Font object. Assignable also by oBox:Font

<**lModal**> True (.T., the default) or when omitted declares the message box is modal. This means, the application is suspended until the user acknowledges the message. False (.F.) declares the message box non modal, which will not suspend the application or the MDI window.

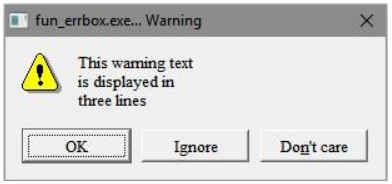
Example 1:

```
MessageBox{"Hello"}:Show()     // info box, OK button
```

Example 2: warning box with text and three buttons

```
    [ OK ] [ Ignore ] [ Don't care ]
```

```
#include "dialog.fh"
oBox := MessageBox{"This warning text;is displayed in" +;
           chr(10)+ "three lines", ;
           MBOX_WARNING, {MBOX_OK, MBOX_IGNORE, "Do&n't care"} }
oBox:Font := FontNew("Times", 10) // redefine the used font
nPressed  := oBox:Exec()          // display, wait for user action
if nPressed == 0
   ? "Esc key pressed"
elseif nPressed == 1
   ? "ok action..."
elseif nPressed >= 2
   ? "will ignore..."
endif
```



Example 3: see also FUN.Alert(), InfoBox(), TextBox()

MessageBox Class Properties

oBox:BoxText := cText *ASSIGN*

Set or redefine the displayed text, i.e. the information to be printed in the message box.

<**cText**> the displayed text, i.e. the information to be printed in the message box. You may split the text in several lines by using either ";" (semicolon) or "\n" or LF=chr(10) character(s). If you need to print semicolon, use "\;". To insert an empty line, use "; ;" or ";;". To clear the previous value, assign NIL or null-string "".

Alignment: the default alignment is left justified in GUI and centered in Terminal i/o mode. You may override this defaults by assigning

```
  _aGlobSetting[GSET_G_N_ALERT_ALIGN] := numValue  // default = 0
```

where <numValue> = 0: default alignment, 1: left justify, 2: center, 3: right justify.

Every single line can additionally be individually aligned by three special characters at the line begin (i.e. after the `";"` or `"\n"` line separator): `"<<!"` align left, `">>!"` align right, `"><!"` or `"<>!"` to center this line, e.g.

```
   oBox:cText := "><!centered;<<!left;>>!right;default"
```

Of course, these special markers are filtered out from the text.

If the line is too long, it is automatically wrapped. In GUI mode, you may specify the maximal text width and height (in pixel) by assigning

```
   _aGlobSetting[GSET_G_N_ALERT_WIDTH ] := nPix   // default = 0
   _aGlobSetting[GSET_G_N_ALERT_HEIGHT] := nPix   // default = 0
```

or <nPix> = 0: adjust/fit box to current window size, or <nPix> = -1: adjust/fit box to desktop size. In GUI mode, the default font oApplic:FontWindow is used, except you specify your own font object by assigning

```
   _aGlobSetting[GSET_G_O_ALERT_FONT] := oFont|NIL  // def = NIL
```

In Terminal i/o mode, the displayed box and wrapping is adjusted automatically according to MaxCol() and MaxRow().

HTML-Text: In GUI mode, you alternatively you may format the message text using HTML like tags (the tag itself is not case sensitive):

`<B>text_part</B>` = print "text_part" in bold

`<I>text_part</I>` = print "text_part" in italic

`<U>text_part</U>` = print "text_part" underlined

`<TT>text_part</TT>` = print "text_part" in fixed font

`<CENTER>text_part</CENTER>` = print "text_part" centered

`<PRE>...</PRE>` = preserve whitespaces in the "..." text part

<FONT color="#rrggbb">text_part</FONT> = print "text_part" in color, where rr=red, gg=green, bb=blue RGB fraction given in hexadecimal (00, 80, FF). Black text is "#000000", white "#ffffff", grey *"#808080"*, purple "#800080", red "#FF0000", blue "#0000FF" and so on. You also may use HTML color names like "yellow", "aqua" etc.

<FONT size=nn>text_part</FONT> = print "text_part" in another font size, nn is the logical size (1 to 7) of the font. The value may either be absolute, for example size=3, or relative like size=-2 or size=+1.

<FONT face=name>text_part</FONT> = print "text_part" in another font family of the font, for example face=times.

<HR> = draw horizontal line

 = new line

<P> or <P>...</P> = new paragraph <IMG src=file> = draw image file

Also simple <TABLE ...><TR><TD> colText </TD><TD> colText </TD></TR>... </TABLE> are supported. You may use following <table> tags: bgcolor, width, border, cellspacing, cellpadding. The <TR> tags are: bgcolor. The <TD> tags are: bgcolor, width, colspan, rowspan, align.

Same as in HTML documents, you may combine the tags, e.g. <B><U> underlined </U> <FONT color="red"> red </FONT> bold</B>

The HTML text formatting is recognized automatically by scanning the text whether there is something that looks like a tag before the first line break. To ensure HTML formatting, set <HTML> at beginning of the text. In HTML formatted text, the line breaks using ";" are not recognized, use
 or <P> tags instead. An example is available in memoedithand.prg

oBox:BoxType := nType ***ASSIGN***

Set or redefine the type of the message box, i.e. the used icon (none, info, warning, error).

<**nType**> is the same as argument <nType> in the MessageBox{...} instantiation, see description there. To clear the previous value, assign NIL or 0 (zero).

oBox:Buttons –> ncaButt ***ACCESS***
oBox:Buttons := ncaButt ***ASSIGN***

Set or redefine the type and caption of the used push buttons.

<**ncaButt**> is the same as argument <ncaButt> in the MessageBox{...} instantiation, see description there. To clear the previous value, assign NIL or 0 (zero).

oBox:Caption := cTitle *ASSIGN*

Sets or redefine the caption of the message box, i.e. text displayed in the title bar of the box.

<**cTitle**> is the same as argument <cTitle> in the MessageBox{...} instantiation, see description there. To clear the previous value, assign NIL or null-string ""

oBox:ColorSpec —> cColor *ACCESS*
oBox:ColorSpec := cColor *ASSIGN*

<**cColor**> is an optional color specification, considered in Terminal i/o mode. For GUI mode, see oBox:GuiColor. The default is pre-defined in the global/public array `_aGlobSetting[GSET_T_C_MSGBOXCOLOR]` and is `"W+/B,B/W"`, see initio.prg

oBox:DefButton := iDefBut *ASSIGN*

Set one button as default, i.e. the by RETURN key simulated button.

<**iDefBut**> is the same as argument <iDefBut> in the MessageBox{...} instantiation. Specify 0 to disable this feature. Value of 1 to 3 to sets the button number 1..3 as default. If the default button feature is disabled, you need to press TAB or Cursor key to reach the required button via keyboard. The default key assignment does not affect the choice via mouse click.

oBox:Exec([nTimeOut]) —> nSelected

Display the message box, wait for user action and return a numeric value representing the consecutive number of the pressed button, starting at 1 (one). Pressing the ESC key returns 0 (zero). Note, the order of keys is guaranteed only when using the array syntax in MessageBox{} instantiation or in the oBox:Buttons assignment. For constants added to each other, preferably use oBox:Show() instead. See example above. Arguments:

<**nTimeOut**> is a optional value specifying time-out in seconds. Zero = 0 which is the default let's forever until an user action.

<**nSelected**> is the consecutive number of the pressed button. 0 is returned on ESC or time out.

oBox:Font —> oFont *ACCESS*
oBox:Font := oFont *ASSIGN*

Set or redefine the used font.

<**oFont**> is the same as argument <oFont> in the MessageBox{...} instantiation, see description there. To clear the previous value, assign NIL which will then use the default oApplic:FontWindow font, or a font assigned globally for all text boxes by

```
_aGlobSetting[GSET_G_O_ALERT_FONT] := oFont|NIL  // def = NIL
```

oBox:GuiColor —> cColor *ACCESS*
oBox:GuiColor := caoColor *ASSIGN*

<**caoColor**> is color specification, considered in GUI mode. For Terminal i/o mode, see oBox:ColorSpec. <caoColor> corresponds to SET COLOR value and is either

- string with foreground and background color, e.g. "N+/RG+"
- or string as RGB tripplet, e.g. "#808080/#F8F4E0" or "N+/#F8F4E0"
- or an array of RGB values for foreground and background (e.g. {{128,128,128}, {248,244,224}}
- or an color object

<**cColor**> is current foreground and background color returned as RGB string tripplet, e.g. "#808080/#F8F4E0"

oBox:Handle() —> num

Supported for backward compatibility to VO only. Returns 0 (zero).

oBox:HotBox —> cBoxFrame *ACCESS*
oBox:HotBox := cBoxFrame *ASSIGN*

<**cBoxFrame**> is an optional string specified the frame displayed around the message box in Terminal i/o mode. The default is pre-defined in the global/public array `_aGlobSetting[GSET_T_C_ALERTBOX]` and is B_PLAIN or B_SINGLE when using #include "fspreset.fh"

oBox:Image —> cImageFile *ACCESS*
oBox:Image := cImageFile *ASSIGN*

<**cImageFile**> is a string specifying the name of user defined image used instead of the MBOX_INFO, MBOX_WARNING, MBOX_ERROR or MBOX_QUEST default images. If <cImageFile> is not empty(), MBOX_USER is assigned automatically to oBox:BoxType. The <cImageFile> may contain either the file name only to search in the current directory, or fully qualified name including path. Neither SET DEFAULT nor SET PATH or FS_SET("lower"|"upper") is considered. Any standard images of the type GIF, JPEG, PNG, BMP, XBM and XPM are supported. You will most probably prefer 32x32 to 50x50 pixel image with transparent background; the message text is displayed right of the image. To create the image, you may use Gimp, Photoshop or any other software. This property is considered in GUI mode, it is supported but ignored in Terminal i/o mode.

Example:

```
oMsgBox := MessageBox{"hallo;this is test;using my own image"}
oMsgBox:Image := "/home/data/common/my_img.gif"
oMsgBox:Exec()
```

oBox:Show() –> nSelConstant

Equivalent to oBox:Exec() but return a numeric constant, representing the pressed button. The constants are equivalent to <expN3> in the MessageBox{...} instantiation, i.e. MBOX_OK, MBOX_YES, MBOX_NO, MBOX_ABORT, MBOX_CANCEL, MBOX_RETRY or MBOX_IGNORE. On user-defined buttons given as string, MBOX_USER1, MBOX_USER2 or MBOX_USER3 constant is returned. Pressing the ESC key returns MBOX_USER0. All these constants are declared in dialog.fh

oBox:TimeOut –> nSeconds *ACCESS*
oBox:TimeOut := nSeconds *ASSIGN*

Get or set the time-out value in seconds. Zero = 0 is the default waits forever until an user action. Equivalent to <nTimeOut> parameter of Exec() method.

oBox:Type –> nType *ACCESS*
oBox:Type := mType *ASSIGN*

Supported for backward compatibility to VO, don't use for new development. A constant or combination of constants that indicates which push buttons and/or icons are displayed. See the BOX* and BUTT* constants in the dialog.fh header file.

TextBox Class

FlagShip provides several GUI classes used for dialog communication. Apart from the general ALERT() function, there is available special MessageBox class and its sub-classes named TextBox, InfoBox, ErrorBox and WarningBox

TextBox Class Index

Class TextBox

Inherits from:	MessageBox
Inherited by:	- (none)
Class prototype:	dialogclass.fh
Defines:	dialog.fh, box.fh
Alternative:	standard function ALERT() or TEXTBOX()

BoxText	ASSIGN	Sets or redefines the displayed text
BoxType	ASSIGN	Type of the message box, i.e. the used icon
Buttons	ACC/ASS	Type and caption of the used push buttons
Caption	ASSIGN	Caption (title) of the message box
ColorSpec	ACC/ASS	Color specification for terminal i/o
DefButton	ASSIGN	Assigns one button as default
Font	ACC/ASS	Sets or redefines the used font
GuiColor	ACC/ASS	Color specification for GUI
HotBox	ACC/ASS	Box frame for terminal i/o
Exec()	METHOD	Display the message box and wait for user action
Show()	METHOD	Equivalent to oBox:Exec()
Handle()	METHOD	for compatibility purpose only
TimeOut	ACC/ASS	time-out in seconds
Type	ACC/ASS	for compatibility purpose only

The TextBox is a usual MessageBox class with pre-defined properties

```
oBox:BoxType := MBOX_NONE
oBox:Caption := if(empty(cTitle), "TextBox", cTitle)
```

TextBox Class Instantiation

Syntax 1:

```
oBox := TextBox {[oOwner], [cTitle], [cText]}
```

Syntax 2:

```
oBox := TextBoxNew ([oOwner], [cTitle], [cText])
```

Syntax 3:

```
oBox := MessageBox {[cText], MBOX_NONE, [ncaButt],
            [iDefBut], [cTitle], [nOwner],
            [oFont], .T. }
```

Any of the above syntax instantiate new text box object, optionally with the given caption, text, push-button(s) and font. The default message box is modal. This means, the application is suspended until the user acknowledges the message. Arguments (all optional):

<**oOwner**> is supported for compatibility purposes to VO and is ignored.

<**cText**> the displayed text, i.e. the information to be printed in the message box. You may spilt the text in several lines by using either `";"` (semicolon) or LF=chr(10) character(s). If you need to print semicolon, use `"\;"`. To insert an empty line, use `"; ;"` or `";;"`. If the argument is omitted, no text is displayed. Assignable also by oBox:BoxText. See the oBox:BoxText property for details about alignment and HTML text formatting.

<**cTitle**> is a caption of the message box, i.e. text displayed in the title bar of the box. If omitted, the application name plus "Info/Warning/Error" text is used. Assignable also by oBox:Caption

Example 1:

```
oBox := TextBox{NIL, "","Hello;world!"}
oBox:Exec()
```

Example 2: see also FUN.Alert(), InfoBox(), TextBox()

TextBox Class Properties

All the class properties are equivalent to MessageBox class, please refer there for further details.

WarningBox Class

FlagShip provides several GUI classes used for dialog communication. Apart from the general ALERT() function, there is available special MessageBox class and its sub-classes named TextBox, InfoBox, ErrorBox and WarningBox

WarningBox Class Index

Class WarningBox

Inherits from:	MessageBox
Inherited by:	- (none)
Class prototype:	dialogclass.fh
Defines:	dialog.fh, box.fh
Alternative:	standard function ALERT() or WARNBOX()

BoxText	ASSIGN	Sets or redefines the displayed text
BoxType	ASSIGN	Type of the message box, i.e. the used icon
Buttons	ACC/ASS	Type and caption of the used push buttons
Caption	ASSIGN	Caption (title) of the message box
ColorSpec	ACC/ASS	Color specification for terminal i/o
DefButton	ASSIGN	Assigns one button as default
Font	ACC/ASS	Sets or redefines the used font
GuiColor	ACC/ASS	Color specification for GUI
HotBox	ACC/ASS	Box frame for terminal i/o
Exec()	METHOD	Display the message box and wait for user action
Show()	METHOD	Equivalent to oBox:Exec()
Handle()	METHOD	for compatibility purposes only
TimeOut	ACC/ASS	time-out in seconds
Type	ACC/ASS	for compatibility purposes only

The WarningBox is an usual MessageBox class with pre-defined properties

```
oBox:BoxType := MBOX_WARNING
oBox:BoxText := if(empty(cText ), "Warning ...", cText)
oBox:Caption := if(empty(cTitle), "Warning",     cTitle)
```

WarningBox Class Instantiation

Syntax 1:

```
oBox := WarningBox {[oOwner], [cTitle], [cText]}
```

Syntax 2:

```
oBox := WarningBoxNew ([oOwner], [cTitle], [cText])
```

Syntax 3:

```
oBox := MessageBox {[cText], MBOX_WARNING,
            [ncaButt], [iDefBut], [cTitle],
            [nOwner], [oFont], .T. }
```

Syntax 4:

```
Alert(cText, [aButt]) -> nSelected
```

Any of the above syntax instantiate new warning box object, optionally with the given caption, text, push-button(s) and font. The default message box is modal. This means, the application is suspended until the user acknowledges the message. Arguments (all optional):

<**oOwner**> is supported for compatibility purposes to VO and is ignored.

<**cText**> the displayed text, i.e. the information to be printed in the message box. You may spilt the text in several lines by using either `";"` (semicolon) or LF=chr(10) character(s). If you need to print semicolon, use `"\;"`. To insert an empty line, use `"; ;"` or `";;"`. If the argument is omitted, no text is displayed. Assignable also by oBox:BoxText. See the oBox:BoxText property for details about alignment and HTML text formatting.

<**cTitle**> is a caption of the message box, i.e. text displayed in the title bar of the box. If omitted, the application name plus "Info/Warning/Error" text is used. Assignable also by oBox:Caption

Example 1:

```
oBox := WarningBox{NIL, "My Warning","Something unusual happen"}
oBox:Exec()
```

Example 2: see also FUN.Alert(), InfoBox(), TextBox()

WarningBox Class Properties

All the class properties are equivalent to MessageBox class, please refer there for further details.

Font Class

This class is used to hold the font information. It is available in all i/o modes, but a meaningful information is given in the GUI mode only.

The Font Class is a collection of attributes of a font. When a text is drawn in GUI mode, it always use a specified or the default font. The most important attributes of a Font are FontFamily(), SizePoint() and Weight(). The used attributes, e.g. italic, underline, striked thru, bold etc. can either be set and retrieved at once by Attrib() or separately be the corresponding method.

As with other GUI classes in FlagShip, the general Fontclass is splitted into three different sub-classes: _gFont for GUI based application, _tFont for terminal/text based, and _bFont for basic i/o. Here, only the _gFont class makes sense and has the functionality described below. To enable a cross-compile-compatibility, the properties of _gFont are available also in the two other classes (tFont and bFont), but they usually have no action and return default value only.

The Font class is used in the Application Window for specifying the font appearance of the main window. For the user window (SDI or MDI), there are two fonts available, one for displaying of text (e.g. @..SAY..) and one for the data entry (e.g. @..GET..). These default windows are set automatically on start-up of the application, i.e. during the instantiation of the Application Window class. You may override these defaults either before creating of the Application Window (usually in the InitIo() modifiable function), or anytime later by using the oApplic:Font object.

Selecting a font is not a trivial operation. The font manager needs to search the system for installed fonts and theirs attributes. The font matching algorithm works as follows: First an available font family is searched for the given FontFamily or FontName. If the requested is not available, the style hint given as Attrib() is used to select a replacement family. If the style hint has not been set, "helvetica" will be used. The following attributes are then matched, in order of priority: character set, pitch, point size, weight, italic. If, for example, a font with the correct character set is found, but all other attributes do not match, this font is even though used, instead of a font with the wrong character set but with all other attributes correct.

Font Class Index

Class Font = _gFont, _bFont, _tFont
Inherits from: - (none)
Inherited by: - (none)
Class prototype: fontclass.fh
Defines: font.fh

Ascent	ACCESS	Distance from base line to the uppermost line
Attrib()	METHOD	Returns or set attributes of this font
AttribChar()	METHOD	Returns or set attributes of this font
Bold	ACC/ASS	Gets or sets the bold attribute for this font
CharSet()	METHOD	Sets and/or return the font character set
CharSetName()	METHOD	Returns the used charset as string "FONT_..."
CloneTo()	METHOD	Copies current font properties to other object
Dialog()	METHOD	Lets the user modify a font by a modal dialog
FontFamily	ACC/ASS	Returns or sets the desired font family name
FontFamily()	METHOD	alternative to the same named ACCESS or ASSIGN
FontName	ACC/ASS	Sets and/or return the desired font name
FontName()	METHOD	Sets and/or return the desired font name
FontPtr	ACCESS	Return the ptr to C++ font class
Height()	METHOD	Retrieves the highest char of the font in pixel
IsEqualTo()	METHOD	Compares Properties of current and other font
Italic	ACC/ASS	Gets, sets or clears italic attrib for this font
LineHeight()	METHOD	Retrieves the height of one row (line) in pixel
Lenght()	METHOD	Retrieves the size of passed text
Name	ACC/ASS	same as FontName Acc/Ass
Normal	ACC/ASS	Checks or sets the "normal" attribute
Pitch	ACC/ASS	Checks, sets or clears variable or fixed pitch
Size	ACC/ASS	Sets/returns the size of the font in points
SizePixel()	METHOD	Sets/returns the size of the font in pixels
SizePoint()	METHOD	Sets/returns the size of the font in points
StrikeThru	ACC/ASS	Checks, sets or clears the striked-thru attrib
Underline	ACC/ASS	Checks, sets or clears the underlined attribute
Width()	METHOD	Retrieves the width of the largest character
WidthChar()	METHOD	Returns the total width of the given string
WidthMaxChar()	METHOD	Return the width of the largest char in string

Font Class Instantiation

Font { [expC1], [expN2], [expC3] } —> oFont ***CREATOR***
FontNew ([expC1], [expN2], [expC3]) —> oFont ***CREATOR, altern. syntax***

Instantiates the Font object of a _gFont, _tFont or _bFont sub-class, in dependence on the compiler switches or latest at run-time. For size-and speed-sensitive applications, you may directly use the sub- class creators instead. Arguments (optional):

<**expC1**> : The desired font family name, see details in oFont:FontFamily. The given family name is case insensitive. If <expC1> is not given, NIL or empty, the default desktop font is used.

<**expN2**> : The desired font size in Points. You may specify or change the font size later by oFont:SizePoint( ) or oFont:SizePixel( ). If <expN2> is not given, or is NIL or 0 (zero), the default desktop size is used.

<**expC3**> : optional font attributes, any combination of letters "N" = normal, "B" = bold, "I" = italic, "U" = underline, "S" = striked thru. You may check or change the font attribute later by oFont:Normal, oFont:Bold, oFont:Italic, oFont:Underline and oFont:StrikeThru

The default character set in GUI mode (when assigning new font) is FONT_ISO8859_15 (=ISO-8859-15, Latin-9) which is nearly equivalent to FONT_ISO8859_1 (=ISO-8859-1, Latin-1) but contains also Euro sign, see details in http://en.wikipedia.org/wiki/Iso-8859-15 This default setting can be changed by assigning

```
  _aGlobSetting[GSET_G_C_FONTCHARSET] := "FONT_ISO8859_15" // def
```

Example 1: instantiation of oFont:

```
oFont1 := Font {"Times", 15}
oFont2 := Font { }
oFont3 := Font {"Arial", 12, "Bu" }
? "oFont1=", oFont1:FontFamily, "size=", oFont1:SizePixel()
? "oFont2=", oFont2:FontFamily, "size=", oFont2:SizePixel()
? "oFont3=", oFont3:FontFamily, "size=", oFont3:SizePixel(), ;
  "attrib=", oFont3:AttribChar(), "bold=", oFont3:Bold, ;
  "underline=", oFont3:Underline, "italic=", oFont3:Italic
```

Example 2: instantiation of gAppWindow and changing the default font of the main window:

```
#include "font.fh"
// oAppWindow := gAppWindow { } // done usually in the InitIo(),
//                              // not in the application !
oFont := oAppWindow:Font
oFont:SizePixel(10)             // == oAppWindow:Font:SizePixel(10)
oFont:Attrib(FONT_HELVETICA + FONT_NORMAL + ;
      FONT_UPRIGHT + FONT_VAR_PITCH)
oAppWindow:Display()
```

Font Class Properties

oFont:Ascent —> nPixel *ACCESS*

Returns the distance from the base line to the uppermost line where pixels may be draw. the return value is in pixels.

oFont:Attrib ([expN1]) —> iFontAtrib

Returns or sets the binary or-ed (or added) attribute of this font. Argument (optional): <expN1> is the font attribute to be set. Either a single constant or an addition of max. each one attribute from these groups:

```
Weight: FONT_LIGHT, FONT_NORMAL (default), FONT_DEMIBOLD,
  FONT_BOLD, FONT_BLACK, FONT_25PERCENT, FONT_50PERCENT,
  FONT_63PERCENT, FONT_75PERCENT, FONT_87PERCENT
Slant: FONT_UPRIGHT (default), FONT_ITALIC
Style: FONT_SANSSERIF, FONT_SERIF, FONT_TYPEWRITER, FONT_OLDENGLISH,
  FONT_SYSTEM, FONT_HELVETICA, FONT_ARIAL, FONT_SWISS, FONT_TIMES,
  FONT_ROMAN, FONT_COURIER, FONT_MODERN, FONT_DECORATIVE
Pitch: FONT_VAR_PITCH, FONT_FIX_PITCH
Special: FONT_STRIKED, FONT_STRIKED_OFF, FONT_UNDERL,
  FONT_UNDERL_OFF
```

Returns: binary or-ed attributes of the currently selected font (corresponding to ::FontFamily), before new attributes (if any) are set. You can determine the single attribute by BinAND() the return value with the attribute constant.

All the above constants are defined in the font.fh header file. Note that the font manager will try to set either your required attribute, or one close to, since most of the attributes are dependent on the available font properties.

Example: sets a new font and prints some of its attributes

```
local oFont := oFont { , 12 }
local iAttrib, iAttrOld
iAttrOld := oFont:Attrib( FONT_MODERN + FONT_UNDERL + ;
                          FONT_LATIN1 + FONT_BOLD )
? "Font ", oFont:FontName, "->", oFont:FontFamily, ;
  "size in pixel", ltrim(oFont:SizePixel))
iAttrib := oFont:Attrib()
if binAND(iAttrib, FONT_LIGHT) != 0
   ?? " light"
elseif binAND(iAttrib, FONT_NORMAL) != 0
   ?? " normal"
elseif binAND(iAttrib, FONT_DEMIBOLD) != 0
   ?? " demibold"
elseif binAND(iAttrib, FONT_BOLD) != 0
   ?? " bold"
else
   ?? " black"
endif
if binAND(iAttrib, FONT_UPRIGHT) != 0
```

```
   ?? " upright/roman "
elseif binAND(iAttrib, FONT_ITALIC) != 0
   ?? " italic "
endif
```

See also oFont:Bold, oFont:CharSet(), oFont:Italic, oFont:Normal, oFont:Pitch, oFont:StrikeThru, oFont:Underline

Supported in GUI mode only, other i/o modes does not set anything and returns 0 (zero) which signals that the requested attribute is not available.

oFont:AttribChar ([expC1]) —> cFontAtrib

Returns or sets attribute of this font given as mnemonic character(s).

Argument (optional):

<**expC1**> is the font attribute to be set. Any combination of letters "N" = normal, "B" = bold, "I" = italic, "U" = underline, "S" = striked thru. You may check or change the font attribute later by oFont:Normal, oFont:Bold, oFont:Italic, oFont:Underline and oFont:StrikeThru or explicit by oFont:Attrib(value)

Returns: a constant same as in <expC1>, determined at the time of entering this method.

oFont:Bold —> lStatus *ACCESS*
oFont:Bold := lStatus *ASSIGN*

Gets, sets or clears the bold attribute for this font. Fully equivalent to oFont:Attrib(FONT_BOLD), i.e.

```
isBold := binAND(oFont:Attrib(), FONT_BOLD) > 0
isBold := oFont:Bold
```

and

```
oFont:Bold := .T.
oFont:Attrib(FONT_BOLD)
```

See also oFont:Attrib(), oFont:Italic, oFont:Normal, oFont:Pitch, oFont:StrikeThru, oFont:Underline

Supported in GUI mode only, other i/o modes does not set anything and returns .F.

oFont:CharSet([expN1|expC1]) —> nAttrib

Sets and/or return the font character set (codec). Argument (optional):

<**expN1**> is a numeric constant from font.fh representing the font character set.

FONT_ISO8859_1 = Latin-1, common in much of Europe
FONT_ISO8859_2 = Latin-2, Central and Eastern European
FONT_ISO8859_3 = Latin-3, South European
FONT_ISO8859_4 = Latin-4, North European
FONT_ISO8859_5 = Latin/Cyrillic
FONT_ISO8859_6 = Latin/Arabic

FONT_ISO8859_7 = Latin/Greek with Euro sign
FONT_ISO8859_8 = Latin/Hebrew
FONT_ISO8859_9 = Latin-5, Turkish
FONT_ISO8859_10 = Latin-6, Nordic
FONT_ISO8859_11 = Latin/Thai alphabet
FONT_ISO8859_12 = Latin/Devanagari
FONT_ISO8859_13 = Latin-7, Baltic Rim
FONT_ISO8859_14 = Latin-8, Celtic
FONT_ISO8859_15 = Latin-9, same as Latin-1 but with Euro
FONT_ISO8859_16 = Latin-10, South-Eastern European
FONT_KOI8R = KOI8-R, Cyrillic - RFC 1489
FONT_KOI8U = KOI8-U, Cyrillic/Ukrainian - RFC 2319
FONT_SET_JA = font specific: Japanese
FONT_SET_KO = font specific: Korean
FONT_SET_TH_TH = font specific: Thai
FONT_SET_ZH = font specific: Chinese
FONT_SET_ZH_TW = font specific: traditional Chinese
FONT_SET_BIG5 = font specific: Chinese
FONT_GBK = font specific: simplified Chinese
FONT_CP1251 = Microsoft Cyrillic encoding
FONT_PT154 = Paratype Asian Cyrillic encoding
FONT_UNICODE = Unicode ISO-10646 (UTF-8 encoding)

<**expC1**> is a string, same as <expN1>, e.g. "FONT_ISO8859_1"

The default character set in GUI mode (when assigning new font) is FONT_ISO8859_15 (=ISO-8859-15, Latin-9) which is nearly equivalent to FONT_ISO8859_1 (=ISO-8859-1, Latin-1) but contains also Euro sign, see details in http://en.wikipedia.org/wiki/Iso-8859-15 This default setting can be changed by assigning

```
_aGlobSetting[GSET_G_C_FONTCHARSET] := "FONT_ISO8859_15" // def
```

Returns: a constant same as in <expN1>, determined at the time of entering this method. To get the name as string, use :CharSetName()

See also oFont:Attrib() and oFont:CharSetName()

Supported in GUI mode only, other i/o modes returns 0 (zero)

oFont:CharSetName( ) —> cCharset

Similar to CharSet() but returns the used charset as string "FONT_..."

oFont:CloneTo(expO1) —> oFont

Clones (copies) current font properties to a destination object. Argument :

<**expO1**> is the destination object of class Font, which properties should be overwritten by properties of the current object.

Returns: <expO1> self or NIL on error.

Note, you also may assign an object variable to another (e.g. oNewFont := oOldFont). But the assignment does not copy the object, it creates only a "link" to the target, so any change on one object reflects also to the other object. On the other hand, this oFont: CloneTo() method copies the current properties (all font attributes) to the target, whereby both the source and target objects remain independent from each other. This behavior is comparable to the standard function ACLONE() vs. an array assignment.

Example:

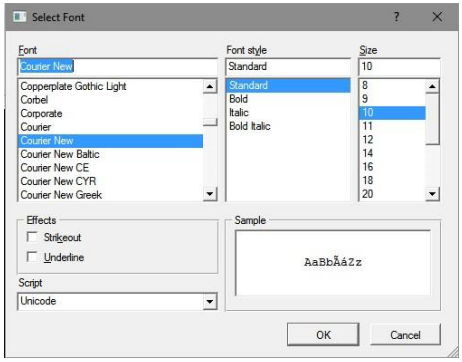
```
oFont1 := Font {"Courier"}
oFont2 := Font { }
? oFont1:FontFamily, oFont2:FontFamily    // Courier Helvetica
oFont1:CloneTo(oFont2)
? oFont1:FontFamily, oFont2:FontFamily    // Courier Courier
? oFont1:IsEqualTo(oFont2)                // .T.
```

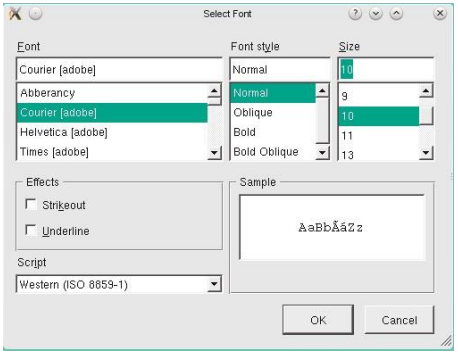
Supported in GUI mode only, other i/o modes returns NIL

oFont:Dialog( ) —> lChanged

Lets the user modify attributes of the current font object by a modal dialog: Supported in GUI mode only, other i/o modes do nothing and returns .F.

Returns: .T. and the modified font object if the user clicked "OK" button. Otherwise .F. and the original unmodified object if the user clicked "Cancel" or closed the dialog window.





Example:

```
oFont := oAppWindow:Font
? "current font:", oFont:FontFamily, ;
  "size pixel/pt:", ltrim(oFont:sizePixel), ltrim(oFont:sizePoint)
lChanged := oFont:Dialog()
? if (lChanged, "-changed-", "-unchanged-"), ;
  "new font:", oFont:FontFamily, ;
  "size pixel/pt:", ltrim(oFont:sizePixel), ltrim(oFont:sizePoint)
```

oFont:IsEqualTo(expO1) –> lEquiv

Compares the current font properties with other font object.

<**expO1**> is the second object of class gFont, which properties should be compared with properties of the current font object.

Returns: logical .T. if all properties and attributes of both objects are equivalent, .F. otherwise

oFont:FontFamily –> cName *ACCESS*
oFont:FontFamily := cName *ASSIGN*
oFont:FontFamily([cName]) –> cName *METHOD*

Sets a desired font, or return the truly selected font family name. Note, the font desired to be set may differ from the returned font family, depending on the installed fonts found by the font manager. To set a font name, you may specify

- either usual X11 or Win32 name, such as "Helvetica", "Arial", "Times", "Courier", "OldEnglish", "System", "AnyStyle", "SansSerif" (= Helvetica), "Serif" (= Times), "TypeWriter" (= Courier), "Decorative" (= OldEnglish), "Swiss" (= Helvetica) . The string with the family name is case insensitive.
- or a constant from the font.fh file specifying both the font and size: FONTMODERN8, FONTMODERN10, FONTMODERN12, FONTROMAN10, FONTROMAN12, FONTROMAN14, FONTROMAN18, FONTROMAN24, FONTSWISS8, FONTSWISS10, FONTSYSTEM8. These constants are for backward compatibility purposes to VO only. It is not recommended to use them for new applications.

The font manager will try to find the desired font or at least a font which is close to the given one. You may check the currently used font family by access oFont:FontFamily after assigning a font.

See also "Selecting a font" at the begin of this class description, oFont:FontName() and Style in oFont:Attrib()

Example:

```
oFont := oFont { "Courier" }
oFont:FontFamily := "Arial"
? oFont:FontFamily    // "Helvetica"
? oFont:FontName()    // "Arial"
```

Supported in GUI mode only, other i/o modes does not set anything and returns "" (null-string)

oFont:FontName –> cName *ACCESS*
oFont:FontName := cName *ASSIGN*
oFont:FontName([cName]) –> cName

Sets and/or return the desired font name. Note, the font desired to be set may differ from the really used font family (::FontFamily), depending on the installed fonts found by the font manager. This oFont:FontName() will return the last desired name, whilst the oFont:FontFamily() the really used font name/family. See also oFont:FontFamily for further details.

Supported in GUI mode only, other i/o modes does not set anything and returns "" (null-string)

oFont:Length(cString) –> nSize

Returns the number of characters of the given string. In GUI mode with Unicode font, each glyph is counted as 1 instead of used characters for.

Supported in GUI mode only, other i/o modes returns LEN(cString)

oFont:Height( ) –> nPixelSize

Retrieves the highest character of the currently used font in pixel. See also oFont:LineHeight() and example in oFont:Width()

Supported in GUI mode only, other i/o modes returns 1

oFont:Italic –> lStatus *ACCESS*
oFont:Italic := lStatus *ASSIGN*

Gets, sets or clears the italic attribute for this font. Fully equivalent to oFont:Attrib(FONT_ITALIC), i.e.

```
  isItalic := binAND(oFont:Attrib(), FONT_ITALIC) > 0
  isItalic := oFont:Italic
```

and

```
  oFont:Italic := .T.
  oFont:Attrib(FONT_ITALIC)
```

See also oFont:Attrib(), oFont:Bold, oFont:Normal, oFont:Pitch, oFont:StrikeThru, oFont:Underline

Supported in GUI mode only, other i/o modes does not set anything and returns .F.

oFont:LineHeight( ) –> nPixelSize

Retrieves the height of one row (line) in pixel. The return value is usually equivalent to oFont:Height() but may be slightly larger for some fonts. See example in oFont:Width()

Supported in GUI mode only, other i/o modes returns 1

oFont:Name –> cName *ACCESS*
oFont:Name := cName *ASSIGN*

Same as oFont:FontName Access/Assign

oFont:Normal –> lStatus *ACCESS*
oFont:Normal := lStatus *ASSIGN*

Checks, or sets the "normal" font attribute. Assigning .T. clears the Bold, Italic, Underline and StrikeThru flag, assigning .F. do not set or clear anything. Equivalent to oFont:Attrib (FONT_NORMAL), i.e.

```
isNormal := binAND(oFont:Attrib(), FONT_NORMAL) > 0
isNormal := oFont:Normal
```

and

```
oFont:Normal := .T.
oFont:Attrib(FONT_NORMAL)
```

See also oFont:Attrib(), oFont:Bold, oFont:Italic, oFont:Pitch, oFont:StrikeThru, oFont:Underline

Supported in GUI mode only, other i/o modes does not set anything and returns .T.

oFont:Pitch –> lStatus *ACCESS*
oFont:Pitch := lStatus *ASSIGN*

Checks, sets or clears the variable/proportional (.T.) or fixed (.F.) pitch attribute of this currently used font. Fully equivalent to oFont:Attrib(FONT_VAR_PITCH), i.e.

```
isFixPitch := binAND(oFont:Attrib(), FONT_VAR_PITCH) == 0
isVarPitch := binAND(oFont:Attrib(), FONT_VAR_PITCH) > 0
isVarPitch := oFont:Pitch
```

and

```
oFont:Pitch := .T.
oFont:Attrib(FONT_VAR_PITCH)
```

See also oFont:Attrib(), oFont:Bold, oFont:Italic, oFont:Normal, oFont:StrikeThru, oFont:Underline

Supported in GUI mode only, other i/o modes does not set anything and returns .F.

oFont:Size –> nPointSize *ACCESS*
oFont:Size := nPointSize *ASSIGN*

Equivalent to nPointSize := SizePoint() or SizePoint(nPointSize)

oFont:SizePixel([expN1]) —> nPixelSize

Sets and/or returns the size of the current font in pixels. Argument (optional):

<**expN1**> is the font size in pixel to be set. If not given, or is NIL or 0, the current size remain unchanged.

Returns: the current font size at the time of entering the method.

See example in oFont:Width()

Supported in GUI mode only, other i/o modes does not set anything and returns 1

oFont:SizePoint([expN1]) —> nPointSize

Sets and/or returns the size of the current font in points (1/72 inch). Argument (optional):

<**expN1**> is the font size in points to be set. If not given, or is NIL or 0, the current size remain unchanged.

Returns: the current font size at the time of entering the method.

See example in oFont:Width()

Supported in GUI mode only, other i/o modes does not set anything and returns 1

oFont:StrikeThru —> lStatus ***ACCESS***
oFont:StrikeThru := lStatus ***ASSIGN***

Checks, sets or clears the striked-thru attribute of this font. Fully equivalent to oFont:Attrib(FONT_STRIKED), i.e.

```
isStrikedThru := binAND(oFont:Attrib(), FONT_STRIKED) > 0
isStrikedThru := oFont:StrikeThru
```

and

```
oFont:StrikeThru := .T.
oFont:Attrib(FONT_STRIKED)
```

See also oFont:Attrib(), oFont:Bold, oFont:Italic, oFont:Normal, oFont:Pitch, oFont:Underline

Supported in GUI mode only, other i/o modes does not set anything and returns .F.

oFont:Underline –> lStatus *ACCESS*
oFont:Underline := lStatus *ASSIGN*

Checks, sets or clears the underlined attribute of this font. Fully equivalent to oFont:Attrib(FONT_UNDERL), i.e.

```
isUnderlined := binAND(oFont:Attrib(), FONT_UNDERL) > 0
isUnderlined := oFont:Underline
```

and

```
oFont:Underline := .T.
oFont:Attrib(FONT_UNDERL)
```

See also oFont:Attrib(), oFont:Bold, oFont:Italic, oFont:Normal, oFont:Pitch, oFont:StrikeThru

Supported in GUI mode only, other i/o modes does not set anything and returns .F.

oFont:Width( ) –> nPixelSize

Retrieves the width (in pixel) of the largest character in the current font. Example:

```
oFont := oFont { "SansSerif", 14 }
? "Font used     = ", oFont:FontFamily()          // helvetica
? "size in Point = ", oFont:SizePoint()           // 14.00
? "size in Pixel = ", oFont:SizePixel()           // 15
? "max h x w     = ", ltrim(oFont:Height()) + " x " + ;
                      ltrim(oFont:Width() )       // 17 x 15
? "max of 'aXMZ5'= ", oFont:WidthMaxChar("aXMZ5") // 13
? "width  'aXMZ5'= ", oFont:WidthChar("aXMZ5")    // 48
? "line height   = ", oFont:LineHeight()          // 17
```

Supported in GUI mode only, other i/o modes returns 1

oFont:WidthChar(cString) –> nPixelSize

Returns the total width (in pixel) of the given string. In GUI mode with Unicode font, the glyph width is counted instead of the size of used characters for. See example in oFont:Width()

Supported in GUI mode only, other i/o modes returns LEN(cString)

oFont:WidthMaxChar(cString) –> nPixelSize

Determine and return the width (in pixel) of the largest character in a given string. With variable font width, this value will usually be smaller for alpha-numeric chars than the value returned by oFont:Width(). See example in oFont:Width()

Supported in GUI mode only, other i/o modes returns 1

GET Class

The GET class provides a mechanism for interactive editing of database fields and variables. In FlagShip, the GET class is generally used to perform @...GET and READ commands. It also enables the creation of user- defined, screen-oriented input/output routines. The methods included make mechanisms for formatting and editing data, cursor navigation and data validation available.

The internal data of the GET object is stored in a normal FlagShip variable or an array element which is created by the GETNEW() function.

Normally, a GET object is associated with a particular input/output variable which stores the edited data. The GET object does not directly access this variable; instead, the variable is manipulated by evaluating a supplied code block. When a GET object is created using the standard @...GET command, an internal code block is **automatically** created which provides access to the variable named in the command. When the user assigns another code block to the BLOCK instance, it becomes the preferred.

Example:

```
LOCAL mystr := "any text" + space(100)
@ 5,10 GET mystr PICTURE "@!S20" COLOR "W+/B,N/W,,,R+/B" ;
           WHEN !EMPTY(mystr) VALID ISCHAR(mystr)
READ
```

is equivalent to:

```
LOCAL mystr := "any text" + space(100)
LOCAL getarr[1]
LOCAL myblock := {|par| IF(par==NIL, mystr, mystr := par)}

getarr[1] := GETNEW (5, 10, myblock, "MYSTR", ;
                     "@!S20", "W+/B,N/W,,,R+/B")
getarr[1]:PREBLOCK  := { || !EMPTY(mystr) }
getarr[1]:POSTBLOCK := { || ISCHAR(mystr) }
READMODAL (getarr)
getarr := {}
```

GETNEW()

Syntax 1:

```
obj = GETNEW ([expN1], [expN2], [expB3], [expC4],
             [expC5], [expC6], [expL7])
```

Syntax 2:

```
obj = GET {[expN1], [expN2], [expB3], [expC4],
           [expC5], [expC6], [expL7] }
```

Purpose:

Creates a new, empty GET object, optionally initialized by the supplied arguments.

Options:

<**expN1**> is the screen row where GET is displayed. This argument is equivalent to assigning the obj:ROW with the same value. The valid range is 0...MAXROW(), the default is zero.

<**expN2**> is the screen column where the GET is displayed. This argument is equivalent to assigning the obj:COLUMN with the same value, the valid range is 0...MAXCOL(). The default is zero.

<**expB3**> is the user supplied block which accesses and modifies the input variable or database field. This argument is equivalent to assigning the obj:BLOCK with the same data.

<**expC4**> is an optional name of the input variable or database field. This argument is equivalent to assigning the obj:NAME with the same string.

<**expC5**> is an optional picture specification used to format the input and output of the GET field. This argument is equivalent to assigning the obj:PICTURE with the same string.

<**expC6**> is an optional color specification used to display the GET field. This argument is equivalent to assigning the obj:COLORSPEC with the same string.

<**expL7**> is an optional logical value specifying whether the <expN1> and <expN2> parameters are given in coordinates or in pixels. When <expL7> is .T., both parameters are interpreted as pixel. If not given, NIL or .F., the interpretation depends on the current SET PIXEL status. This argument is set correspondingly by [NO]PIXEL clause of @..GET command.

Returns:

<**obj**> is the new allocated GET object, usually assigned to a regular FlagShip variable or to an array element.

Description:

GETNEW() creates a new, empty get object. If the optional arguments are supplied, the corresponding instance variables are filled with these values.

To perform a READ using the GET object, at least the first three arguments must be specified in GETNEW() or assigned using the instance variables. A READ for one GET field stored in a regular variable can be invoked using GETREADER(), while a READ for an array of GET objects can be invoked using the READMODAL() function.

Example:

```
LOCAL myget := GETNEW(), myvar := SPACE(100)
LOCAL getarr[2], data1 := 1, data2 := 2

getarr[1] =GETNEW(0,0,{|par|IF(par==NIL,data1,data1:=par)})
getarr[2] =GETNEW(1,0,{|par|IF(par==NIL,data2,data2:=par)})
READMODAL (getarr)

myget:ROW     := 10
myget:COLUMN  := 0
myget:BLOCK   := {|par| IF(par==NIL, myvar, myvar:=par) }
myget:COLORSPEC := "W+/B,R/W,,,B/W"
myget:NAME    := "myvar"
myget:PICTURE := "@!S20"
GETREADER (myget)
```

Classification:

programming

Class:

GET class, prototyped in <FlagShip_dir>/include/getclass.fh

Source:

The user defined READ is available in <FlagShip_dir>/system/ getsys.prg, including the READMODAL() and GETREADER() functions.

Compatibility:

Available in FS4, C5 and VO. The alternative syntax 2 and the possibility of inheriting it into an own subclass is available in FlagShip only.

Related:

@..GET, READ, std.fh, VALTYPE(), GETACTIVE(), GETAPPLKEY(), GETDOSETKEY(), GETPOSTVAL(), GETPREVALID(), GETREADER(), READMODAL(), READINSERT(), READEXIT()

Get Class Index

Class Get
Inherits from: -
Class prototype: getclass.fh
Defines: getexit.fh, inkey.fh, set.fh

Assign()	METHOD	Assigns editing buffer to the GET variable
Backspace()	METHOD	Deletes character left of the cursor
Baddate	ACCESS	Does editing buffer contain valid date?
BadDate()	METHOD	internal
Block	ACC/ASS	Code block that associates object with variable
Buffer	ACC/ASS	Character editing buffer of the GET
Cargo	Export	Any user data
Changed	ACCESS	Has the get:BUFFER changed?
ClassName()	METHOD	Return "GET"
Clear	ACC/ASS	Clear buffer before editing?
Col	ACC/ASS	Screen column where the GET field starts
Col()	METHOD	Set/get screen column or pixel of the GET field
ColorDisp()	METHOD	Changes the color specification
ColorSpec	ACC/ASS	Color attributes for the GET object
Decpos	ACCESS	Decimal point position within the editing buffer
Copy()	METHOD	Copies marked text into cut-and-paste buffer
DelEnd()	METHOD	Deletes rest of the editing buffer
Delete()	METHOD	Deletes character under cursor
DelLeft()	METHOD	Deletes character left of cursor
DelRight()	METHOD	Deletes character right of cursor
DelWordLeft()	METHOD	Deletes word left of cursor
DelWordLef()	METHOD	same as DelWordLeft()
DelWordRight()	METHOD	Deletes word right of cursor
DelWordRig()	METHOD	same as DelWordRight()
Destroy()	METHOD	Destroys the GET object
DestroyOnAxit	ACC/ASS	Should get:Destroy() be called on Axit?
Display()	METHOD	Displays the GET object on the screen
EmptyDate	ACCESS	Is the date entry empty?
End()	METHOD	Moves cursor to the rightmost editable position
End2Char	ACC/ASS	Controlls the behavior of get:End()
Exec()	METHOD	Process user input and editing
ExitState	ACC/ASS	Controlls the action on GET exit
Font	ACC/ASS	Font object used for GET display and processing
GetEnabled	ACC/ASS	Enables/disables GET object from READ process
GuiColor	ACC/ASS	Corresponds to GUICOLOR clause in @..GET
GuiObj2var()	METHOD	similar to VarPut()
GuiVar2obj()	METHOD	similar to VarGet()
Handler	ACC/ASS	Code block specifying handler for get:Exec()

Hasfocus	ACCESS	Has the Get field input focus?
Height	ACC/ASS	Height of the GUI widget
Height()	METHOD	same as Height ACC/ASS
HitTest()	METHOD	Checks if the given coordinates are in GET
Home()	METHOD	Moves cursor to the leftmost editable position
Home2Char	ACC/ASS	Controlls the behavior of get:Home()
Insert()	METHOD	Inserts one or more character(s) into buffer
KillFocus()	METHOD	Removes input focus from the GET object
Left()	METHOD	Moves cursor left to nearest editable position
Message	ACC/ASS	String displayed in the SET MESSAGE line
Minus	ACC/ASS	Was minus sign entered?
Move()	METHOD	Move GUI widget to new position
Name	ACC/ASS	Name of the GET variable
OnClickAction	ACC/ASS	Action in READ triggered by code block
OnClickKeys	ACC/ASS	Simulates key press, triggered by code block
Original	ACCESS	Copy of the variable content at begin of edit
Overstrike()	METHOD	Puts one or more character(s) into buffer
Paste()	METHOD	Copy cut-and-paste buffer into editing buffer
Picture	ACC/ASS	String specifying the field formatting
Pos	ACCESS	Curr cursor position relative to buffer begin
PostBlock	ACC/ASS	Code block for post-validation (VALID clause)
PreBlock	ACC/ASS	Code block for pre-validation (WHEN clause)
Reader	ACC/ASS	Code block accessing user defined READ
Rejected	ACCESS	Was the edit character placed into buffer?
Reset()	METHOD	Resets internal status information
Right()	METHOD	Moves cursor right to nearest editable posit
Row	ACC/ASS	Screen row of the GET field
Row()	METHOD	Set/get screen row or pixel of the GET field
SetFocus()	METHOD	Sets input focus to the GET object
SetCursor()	METHOD	Sets the cursor type and color for GUI mode
Show()	METHOD	Same as get:Exec()
Subscript	ACC/ASS	Used if the GET variable is an array element
ToDecPos()	METHOD	Moves cursor right of the deci point position
ToolTip	ACC/ASS	String displayed in GUI mode
Type	ACCESS	Data type of the GET variable
Typeout	ACCESS	Accepted cursor movement?
Undo()	METHOD	Resets internal GET status information
Untransfor()	METHOD	same as Untransform()
Untransform()	METHOD	Converts get:BUFFER into variable date type
UpdateBuffer()	METHOD	Sets get:BUFFER to current value of GET variable
UpdateBuff()	METHOD	same as UpdateBuffer()
VarGet()	METHOD	Returns current value of the GET variable
VarPut()	METHOD	Sets the GET variable to specified value
Width	ACC/ASS	Width of the GUI widget
Width()	METHOD	same as Width ACC/ASS
WordLeft()	METHOD	Moves cursor one word to the left
WordRight()	METHOD	Moves cursor one word to the right

GET Instance Variables

get:BADDATE *Access*

Contains a logical value indicating that the editing buffer does not represent a valid date if the value is TRUE. When the date is valid, or the current GET is not a date, the value contains FALSE.

get:BLOCK *Access/Assign*

Contains the user supplied code block that associates the GET object with a variable. If the object is created by the @...GET command, an internal code block is used. When the user assigns another code block, the one stored in get:BLOCK will be preferred.

The code block takes an optional argument the value of which is assigned to the variable. If the argument is omitted, the code block returns the current value of the variable, e.g. when editing the "myvar" variable or "myfld" field:

```
myget:BLOCK = { |par| IF (par == NIL, myvar, myvar := par) }
myget:BLOCK = { |par| IF (PCOUNT()=0, FIELD->myfld, ;
                                FIELD->myfld := par) }
```

If the GET variable is an array element, you may use the get:BLOCK only for arrays with constant indices. When using the @...GET command the subscript(s) in the expression are stored internally. Setting and getting array elements may be done by using the get:VARGET() and get:VARPUT() methods, which is also the preferred method for simple variables.

get:BUFFER *Access/Assign*

Contains a character value which is the editing buffer used by the GET object. The value is meaningful only when the object has input focus. At other times, the value is mostly NIL or "", and all attempts to assign a new value do not affect the GET variable.

Note that in GUI i/o mode, the internal buffer data are stored and handled in ISO/ANSI character set, regardless the current SET GUITRANSL TEXT on/off translation mode. The buffer is set/translated in get:Display(), get:ASSIGN(), get:UNTRANSFORM(), get:UPDATEBUFFER() from/to the target variable or field by considering the current SET GUITRANSL TEXT status, or the equivalent SET SOURCE ASCII/ISO or SET (_SET_GUIASCII) flag. So if you check or update the buffer manually and SET (_SET_GUIASCII) is .T., access the buffer data using data := ANSI2OEM(get:BUFFER) or assign get:BUFFER := OEM2ANSI(data)

In textual/terminal i/o mode, the current TERM (or CodePage in Windows) is considered instead.

get:CARGO *Access/Assign*

Contains user data of any type, to store information retrieved later in the program. Not used by the GET system itself.

get:CHANGED *Access*

Contains a logical value indicating whether the get:BUFFER has changed since the GET has received input focus. It contains TRUE if the BUFFER has been changed by one the edit methods; otherwise it contains FALSE. Assigning a value to get:BUFFER or altering the GET variable will not change the state of get:CHANGED. Get:SETFOCUS() and get:KILLFOCUS() clears it to FALSE.

get:CLEAR *Access/Assign*

Contains a logical value indicating whether the editing buffer should be cleared before any values are entered. This instance is set TRUE when executing get:SETFOCUS() or get:UNDO(), and the get:PICTURE contains the "@K" (picture function) or the GET variable is a numeric type.

get:COL *Access/Assign*
get:COL([nCol], [lPixel]) *Method*

Contains a numeric value defining the screen column position where the GET field starts. Equivalent to the <column> argument of the @..GET command. In GUI mode the current SET PIXEL setting decides if the entry and return value is in coordinates or pixels. You may override this by using the second parameter of get:COL() method, where <lPixel> == .T. specify using pixel and <lPixel> == .F. specify using coordinates.

get:COLORSPEC *Access/Assign*

Contains a character string defining the display color attributes for the GET object, equivalent to the COLOR <color> argument of the @..GET command. The color string must contain at least both "enhanced" and "unselected" attributes. If this property is specified, it is always used in Terminal i/o mode. In GUI mode, it is used only if get:GUICOLOR is not set, and when SET GUICOLOR is ON. If get:COLORSPEC is not specified, get:Display() use the current SetColor() setting (always in Terminal i/o, and with SET GUICOLOR ON in GUI mode). For further color information, refer to (CMD) SET COLOR.

get:DECPOS

Contains a numeric value indicating the decimal point position within the editing buffer, meaningful only when editing a numeric variable and when the object has

input focus. If the variable and/or the picture has no decimals, this instance contains zero.

get:DESTROYONAXIT *Access/Assign*

Contains optional logical value, specifying that get:Destroy() should by called at exit of READ. Set by the CLEAR or DESTROY clause of @..GET or of READ. See get:Destroy() for details.

get:END2CHAR *Access/Assign*

Contains a logical value controlling the behavior of get:End(). If .T., get:End() moves behind the last character in buffer not a space, .F. moves to last editable character in buffer. The default is taken from a global variable `_aGlobSetting [GSET_L_GET_END2CHAR]`, which is .T. by default.

get:EXITSTATE *Access/Assign*

Contains a numeric value indicating the desired action, or the state when the GET object was exited and is used in the user-modifiable READ (see <FlagShip_dir>/ system/getsys.prg).

Val	getexit.fh	Description
0	GE_NOEXIT	No exit attempted, prepare GET for editing
1	GE_UP	Go to previous GET
2	GE_DOWN	Go to next GET
3	GE_TOP	Go to first GET
4	GE_BOTTOM	Go to last GET
5	GE_ENTER	Normal end of GET editing
6	GE_WRITE	Terminate READ, save GET
7	GE_ESCAPE	Terminate READ, do not save GET
7	GE_EXIT	same as GE_ESCAPE
8	GE_WHEN	WHEN clause unsatisfied
9	GE_MOUSE	Mouse button clicked

get:FONT *Access/Assign*

Contains font object used for the GET display and processing, or NIL if not applicable. If not specified, the default font oApplic:Font is used. Assign is considered before first get:Display(). As opposite to standard font, this font do not influence the row and colum coordinate. Applicable in GUI mode, ignored otherwise.

get:GETENABLED *Access/Assign*

Contains logical value which enables/disables the GET object from READ process (considered in getsys.prg), default is .T. When changing this property, you may need

to re-display this GET by get:Display(.T.) when color pair#7 in COLOR or GUICOLOR is available, for example

```
@...GET varname GUICOLOR...
atail(Getlist):GetEnabled := .F. ; atail(Getlist):Display(.T.)
```

get:GUICOLOR Export *Access/Assign*

Contains a character string defining the display color attributes for the GET object for executable running in GUI mode. The property is set by the GUICOLOR <color> clause of the @..GET command. The color string should contain at least both "enhanced" and "unselected" attributes and may contain also "disabled" and "unselectedWindow" pairs. If this property is specified, it is always used in GUI mode and is ignored in Terminal i/o, where get:COLORSPEC apply. If get:GUICOLOR is not specified, get:Display() use either get:COLORSPEC or SetColor(), in GUI mode only when SET GUICOLOR is ON. For further color information, refer to (CMD) SET COLOR.

get:HANDLER *Access/Assign*

Contains optional code block, used to handle keyboard input. get:Exec() passes the current object to the code block. The default setting is get:Handler := {|obj| GetReader(obj) } which triggers the GetReader() function available in getsys.prg source. See also get:READER.

get:HASFOCUS *Access*

Contains a logical value that indicates if the GET object has input focus, set by get:SETFOCUS(). If so, this instance contains TRUE, otherwise FALSE. See also the GETACTIVE() function in section FUN.

get:HEIGHT *Access/Assign*
get:HEIGHT([nSize], [lPixel]) *Method*

Contains numeric value specifying the height of the GUI widget. The default is a value of one row. The current SET PIXEL setting decides if the entry and the return value is in coordinates or pixels. You may override this behavior by using the second parameter of get:HEIGHT() method, where <lPixel> == .T. specify using pixel and <lPixel> == .F. specify using coordinates.

get:HOME2CHAR *Access/Assign*

Contains a logical value controlling the behavior of get:Home(). If .T., get:Home() moves to first character in buffer not a space, .F. moves to first editable character. The default is taken from a global variable `_aGlobSetting [GSET_L_GET_HOME2CHAR]`, which is .T. by default.

get:MESSAGE **Access/Assign**

Contains a character string displayed in the SET MESSAGE TO line (in Terminal i/o mode) or in the status bar of GUI mode. The message is displayed in READ, processed via getsys.prg when the get object receive and loose input focus. See also get:TOOLTIP and getsys.prg

get:MINUS **Access/Assign**

Contains a logical TRUE value when a minus sign has been added to the editing buffer. This is meaningful only when the object has input focus, during the editing of a numeric variable, if the current buffer is empty (zero) and the last change to the editing buffer was a minus sign. It is cleared to FALSE when any other change is made to the buffer.

get:NAME **Access/Assign**

Contains a character string representing the name of the GET variable. This value is optional and is only used by the GET methods to access a dynamically scoped variable, when neither the code block, nor the address of the variable (set by get:VARPUT()) was specified. When the object is created by the @..GET command, this instance contains the specified variable or field name.

get:OnClickAction **Access/Assign**

Contains either NIL or numeric value specifying next READ action (considered in getsys.prg handler). This request is usually set in get:Notify (or other) code block, and is same as get:ExitState property:

Val	getexit.fh	Description
0	GE_NOEXIT	No exit attempted, stay in GET
1	GE_UP	Go to previous GET
2	GE_DOWN	Go to next GET
3	GE_TOP	Go to first GET
4	GE_BOTTOM	Go to last GET
5	GE_ENTER	Normal end of GET editing
6	GE_WRITE	Terminate READ, save GET
7	GE_ESCAPE	Terminate READ, do not save GET
7	GE_EXIT	same as GE_ESCAPE

get:OnClickKeys **Access/Assign**

Contains either NIL or a string comparable to KEYBOARD, which keys are evaluated after exit from get:Notify (or other) code block. You may set in the code block e.g. obj:OnClickKeys := chr(K_UP, K_UP) to skip two fields up when this field is clicked. Considered in getsys.prg READ handler.

get:ORIGINAL **Access**

Contains a value of any data type that is a copy of the value in the GET variable at the time of get:SETFOCUS(), and is therefore meaningful only when the GET has input focus. It is used to restore the original by get:UNDO().

get:PICTURE **Access/Assign**

Contains a character value defining the PICTURE string that controls formatting and editing for the GET object. It is equivalent to the PICTURE clause of the @..GET command. On get:SETFOCUS(), this instance contents is validated and corrected when required.

get:POS

Contains a numeric value indicating the current cursor position relative to the beginning of the editing buffer, starting with zero. Meaningful only when the GET has input focus. Compatibility: Clipper's get:POS starts with 1.

get:POSTBLOCK **Access/Assign**

Contains an optional code block, which is used to validate the value of GET variable, and is executed when exiting the edit mode of the current GET. If specified, the code block should return TRUE to enable exiting the GET field when in READ, or FALSE to reenter the edit mode. When using a @..GET command, the code block body is build from the VALID and/or RANGE clauses. GET methods do not use this instance, but it is used by the user modifiable READ (see <FlagShip_dir>/system/ getsys.prg).

get:PREBLOCK **Access/Assign**

Contains an optional code block that validates GET object before entering. If specified, the code block should return TRUE to enable editing the object, or FALSE to skip the edit field being in READ. When using the @..GET command, the code block body is equivalent to the WHEN clause. The GET methods does not use this instance, but it is used by the user modifiable READ (see <FlagShip_dir>/system/ getsys.prg).

get:READER **Access/Assign**

Contains an optional code block to implement special READ behavior for any GET object. If specified, the standard READMODAL() function evaluates that block to READ the object, otherwise the default GETREADER() function (included in getsys.prg) is used. This property is available for compatibility purposes and is nearly equivalent to get: HANDLER, except that :READER has no default code block but get:HANDLER has.

get:REJECTED *Access*

Contains a logical value indicating whether the last character written to the buffer by get:INSERT() or get:OVERSTRIKE() was placed into the BUFFER. If so, the instance contains FALSE, or TRUE if the operation is rejected.

get:ROW *Access/Assign*
get:ROW([nCol], [lPixel]) *Method*

Contains a numeric value defining the screen row where the GET field is displayed. Equivalent to the <row> argument of the @..GET command. In GUI mode the current SET PIXEL setting decides if the entry and the return value is in coordinates or pixels.You may override this behavior by using the second parameter of get:ROW() method, where <lPixel> = .T. specify using pixel and <lPixel> == .F. specify using coordinates.

get:SUBSCRIPT *Access/Assign*

Contains an array of numeric values representing the dimensions of a GET array element, if such is used; or NIL if a regular variable is used. It is assigned by the standard @..GET command. For example, get:SUBSCRIPT contains {5,4,2} for @..GET xyz[5,4,2] or {15} when executing @..GET xyz[15].

get:TOOLTIP *Access/Assign*

Contains a character string displayed in GUI mode in small popup window when the mouse cursor is placed over the GET field. The get:TOOLTIP is ignored in other i/o modes. See also get:MESSAGE

get:TYPE *Access*

Contains a string specifying the data type of the GET variable, equivalent to the result of the TYPE() or VALTYPE() functions.

get:TYPEOUT *Access*

Contains TRUE, if the most recent method attempted to move the cursor out of the editing buffer, or if there are no editable positions in the buffer. FALSE indicates a correct cursor movement. The instance is reset by any cursor movement method.

get:WIDTH *Access/Assign*
get:WIDTH ([nSize], [lPixel]) *Method*

Contains numeric value specifying the width of the GUI widget. The default value is set from the GET field width in accordance to PICTURE specification. The current SET PIXEL setting decides if the entry and the return value is in coordinates or pixels.

You may override this behavior by using the second parameter of get:WIDTH() method, where <lPixel> == .T. specify using pixel and <lPixel> == .F. specify using coordinates.

GET Init & Status Methods

[get =] get:ASSIGN ()

Assigns the value in the editing buffer to the GET variable by evaluating get:BLOCK with the buffer contents supplied as its argument. Meaningful only when the object has input focus.

Note that in GUI i/o mode, the internal buffer data are stored and handled in ISO/ANSI character set. The buffer is set/translated here to the target variable or field by considering the current SET GUITRANSL TEXT status, or the equivalent SET SOURCE ASCII/ISO or SET(_SET_GUIASCII) flag.

In textual/terminal i/o mode, the current TERM (or CodePage in Windows) is considered instead.

[get =] get:COLORDISP (<expC>)

Changes the color specification of the GET object, similar to issuing get:COLORSPEC := <expC> ; get:DISPLAY().

[get =] get:DESTROY ()

Destroys the GET object and restores the screen. Called from READ via getsys.prg when the CLEAR clause of @..GET or READ was specified, or the get:DestroyOnAxit property was set otherwise.

[get =] get:DISPLAY ([<lForce>])

Displays the GET object on the screen. If the object has input focus, the get:BUFFER is displayed with the "selected" color attribute and the cursor is placed at the current editing position. If the object has no focus, the get:BLOCK is evaluated and the result is displayed using the "unselected" color attribute. Refer to SET COLOR for an explanation of color attributes. See also get:Exec()

Note: in GUI mode, Display() is highly optimized to avoid flickering. In some occurrences, this avoids re-displaying of the manually changed values. In such a case, invoke get:Display(.T.) which will force the re-display. The <lForce> parameter is accepted also in Text mode.

[get =] get:EXEC ()

Displays the GET object on the screen and process user input using the code block in get:Handler. It is similar to @..GET/READ command using a single Get field.

[get =] get:KILLFOCUS ()

Removes the input focus (set by get:SETFOCUS()) from the GET object, redisplays the editing buffer and discards internal state information. Executed only when the object has input focus, ignored elsewhere.

[get =] get:RESET ()

Resets the internal status information of the GET object to values, as when invoking get:SETFOCUS(). Executed only when the object has input focus, ignored elsewhere.

[aSetting =] get:SETCURSOR ([aOverwrite], [aInsert])

Sets the cursor mode and colors for READ in GUI mode.

<aOverwrite> is an array {<nMode>, <caoColor>} for overwrite mode
<aInsert> is an array {<nMode>, <caoColor>} for insert mode
<nMode> is numeric 0: cursor is default vertical bar
1: cursor is I-beam
2: cursor is box around the curr. Character
3: cursor is underline
<caoColor> is either foreground color character, or color object, or an array of RGB triplets {red,green,blue} ea 0..255

For example

```
oGet:SetCursor( {3,{0,255,0}}, {1,{0,0,0}} )
```

sets green underline in overwrite and black I-beam for insert mode. You may specify different modes/colors for each GET object. If not set, the default setting is used in READ. This default is user modifiable by assigning mode and colors for both overwrite and insert cursor mode

```
_aGlobSetting[GSET_A_READ_GUICURSOR] := {<aOverwrite>,<aInsert>}
```

whereby only array for <caoColor> is accepted. For example

```
_aGlobSetting[GSET_A_READ_GUICURSOR] :=
                                        {{2,{255,0,0}},{1,{0,0,255}}}
```

sets red box in overwrite and blue I-beam for insert mode. Default is

```
_aGlobSetting[GSET_A_READ_GUICURSOR] := {{0,{0,0,0}},{0,{0,0,0}}}
```

The method returns an array {<aOverwrite>,<aInsert>} of current setting or NIL for Terminal and Basic mode.

[get =] get:SETFOCUS ()

Sets the input focus to the GET object, initializes internal state information and the instances of get:BUFFER, get:POS, get:DECPOS, and get:ORIGINAL. Displays the buffer as does get:DISPLAY(), using the "selected" color attribute.

See also the GETACTIVE() function, which determines the currently focused GET object.

[get =] get:UNDO ()

Resets the internal status information of the GET object to values, as when invoking get:SETFOCUS(). Executing get:UNDO() is equivalent to copying get:ORIGINAL into the GET variable and then executing the get:RESET() method. Performed only when the object has input focus, ignored elsewhere.

retval = get:UNTRANSFORM ()

Converts the get:BUFFER into the date type of the original GET variable. get:ASSIGN() is similar to get:VARPUT(get:UNTRANSFORM()). Executed only when the object has input focus, ignored elsewhere.

[get =] get:UPDATEBUFFER ()

Sets the get:BUFFER to the current value of the GET variable and redisplays the edit buffer. Executed only when the object has input focus, ignored elsewhere.

retval = get:VARGET ()

Returns the current value of the GET variable. For simple variables, the <retval> corresponds to executing the statement retval := EVAL(get:BLOCK). When the GET variable is an array element, VARGET() is the only method to retrieve this element value. The content of the current value is returned "as is", without any translation.

[retval =] get:VARPUT (<exp>)

Sets the GET variable to the passed value of any data type. For simple variables, the get:VARPUT(exp) corresponds to the execution of EVAL(get:BLOCK,exp). When the GET variable is an array element, the VARPUT() method is the only one to assign this element value. The <exp> content is stored in the current variable "as is", i.e. without any translation.

GET Editing Methods

All the cursor movement and editing methods will be executed only when the GET object input focus is set by get:SETFOCUS(). Otherwise they are ignored.

[get =] get:BACKSPACE ()

Deletes the character to the left of the cursor moving the cursor one position to the left. Ignored, when the cursor is at the leftmost editable position.

[get =] get:COPY ()

Copies currently marked text into clipboard cut-and-paste buffer. Available for GUI mode only.

[get =] get:DELETE ()

Deletes the character under the cursor, moves the rest of the buffer one position left, when a string variable is edited.

[get =] get:DELEND ()

Deletes the rest of the editing buffer, starting at the current cursor position.

[get =] get:DELLEFT ()

Deletes the character to the left of the cursor.

[get =] get:DELRIGHT ()

Deletes the character to the right of the cursor.

[get =] get:DELWORDLEFT ()

Deletes the word to the left of the cursor.

[get =] get:DELWORDRIGHT ()

Deletes the word to the right of the cursor.

[get =] get:END ()

Moves the cursor to the rightmost editable position within the editing buffer, or to the last character in buffer (default). You may control the behavior individually for each GET object by get:End2Char or globally by assigning a logical value to

```
_aGlobSetting[GSET_L_GET_END2CHAR] := .F. // default is .T.
```

to set cursor to the last editable buffer position.

[get =] get:HOME ()

Moves the cursor to the leftmost editable position within the editing buffer, or to the first character in buffer (default). You may control the behavior individually for each GET object by get:Home2Char or globally by assigning a logical value to

```
_aGlobSetting[GSET_L_GET_HOME2CHAR] := .F. // default is .T.
```

to set cursor to the first editable buffer position..

[get =] get:INSERT (<char>)

Inserts one or more character(s) <char> into the editing buffer at the current cursor position. When editing character variables, the content of the editing buffer is shifted to the right. When editing numeric or date values, the existing content of the buffer is shifted to the left. See also get:OVERSRTIKE().

[get =] get:LEFT ()

Moves the cursor left to the nearest editable position within the editing buffer. If there is no editable position to the left, the cursor position remains unchanged.

[get =] get:OVERSTRIKE (<char>)

Puts one or more character(s) <char> into the editing buffer at the current cursor position, overwriting the current buffer character. The cursor is placed one position to the right.

Note that in GUI i/o mode, the internal buffer data are stored and handled in ISO/ANSI character set, regardless the current SET GUITRANSL TEXT on/off translation mode. So the passed <char> should be a part of the ISO/ANSI character set, passed e.g. directly from Inkey().

In textual/terminal i/o mode, the current TERM (or CodePage in Windows) is considered.

See also GETAPPLKEY() function, which applies a key value to the currently focused get:BUFFER.

[get =] get:PASTE ()

Copies (or inserts) content of clipboard cut-and-paste buffer into current GET field. Available in GUI mode only.

[get =] get:RIGHT ()

Moves the cursor right to the nearest editable position within the editing buffer. If there is no editable position to the right, the cursor position remains unchanged.

[get =] get:TODECPOS ()

Moves the cursor to the immediate right of the decimal point position in the editing buffer. Meaningful only when editing a numeric value and get:DECPOS is greater than zero.

[get =] get:WORDLEFT ()

Moves the cursor one word to the left within the editing buffer. It skips all characters within the current word and all leading spaces. The cursor remains on the first character of the previous word, or at the first editable buffer position.

[get =] get:WORDRIGHT ()

Moves the cursor one word to the right within the editing buffer. It skips all characters within the current word and all subsequent spaces. The cursor remains on the first character of the next word, or at the last editable buffer position.

ListBox Class

The ListBox Class creates and manages list boxes and combo boxes. The Achoice() function is based on ListBox class.

List boxes and combo boxes display a list of items or choices to the user. The list box methods will allow you to add, arrange, remove, and interrogate the list of items. When one of the items is selected, ListBox:CurrentItem, ListBox:CurrentItemNo, ListBox:TextValue, and ListBox: Value are updated. A list box may be bound to fields by oListBox:FillUsing().

As with other GUI classes in FlagShip, the general ListBox class is internally inherited by three different sub-classes: _gListBox for GUI based application, _tListBox for terminal/text based mode, and _bListBox for basic i/o mode, all defined in the boxclass.fh header file. The proper class, corresponding to the used i/o mode, is set either at compile time with the compiler switch `"-io=g|t|b"`, or latest at run-time depending on the currently used environment.

Note: in the basic i/o mode, only a rough list box functionality is simulated by the sequential in/output.

ListBox Class Index

Class ListBox

Inherits from:	- (none)
Inherited by:	ComboBox
Class prototype:	boxclass.fh
Defines:	box.fh

AddItem()	METHOD	Add (append) a new item to a list box
Bitmap	ACC/ASS	Display bitmap as list box item
Bottom	ACC/ASS	Bottommost screen row of the box
Buffer	ACC	Position in the list of the selected item
CapCol	ACC/ASS	Screen column of the list box's caption
CapRow	ACC/ASS	Screen row of the list box's caption
Caption	ACC/ASS	String that describes the list box caption
Cargo	ACC/ASS	A user value of any type
ChangeSelected()	METHOD	Change a range of items in a multiple selection
ClassName()	METHOD	For compatibility to Clipper's getsys.prg only
Clear()	METHOD	Clear (delete) all items in a list box
ClearSelection()	METHOD	Clear a multiple selection list box
Close()	METHOD	Closes the combo box ("drop-down list box")
ColdBox	ACC/ASS	Frame of list box without focus
ColorSpec	ACC/ASS	Color attributes for Terminal i/o
ColumnLeft	ACC/ASS	Number of the leftmost visible column
CurrentItem	ACC/ASS	String representing the displayed listbox item

CurrItemNo	ACC/ASS	Numeric value indicating the selected item
CurrentText	ACC/ASS	Fix ""
DeleteItem()	METHOD	Remove an item from a list box
DelItem(p1)	METHOD	Remove an item from a list box
DeselectItem()	METHOD	Turn off the selection of a specified item
Destroy()	METHOD	Destroys the ListBox object
Display()	METHOD	Show the list box and its caption on the screen
DropDown	ACC	Indicator of list box or combo box
Exec()	METHOD	Process user input, same as :Show()
Fblock	ACC/ASS	Code block evaluated at receiving/loosing focus
FillUsing()	METHOD	Data server/dictionary driver
FindItem()	METHOD	Search a list box for a specified item
FindText()	METHOD	Search a list box for a specified string
FirstSelected()	METHOD	Position of the 1st item in a multiple selection
Font	ACC/ASS	Font object used to display the list box items
GetData()	METHOD	Get the data portion of a list box item
GetItem()	METHOD	Get the item property
GetItemValue()	METHOD	Same as GetData()
GetText(p1)	METHOD	Get the item text
GuiColor	ACC/ASS	Color attributes for GUI mode
HasFocus	ACC	Indicates whether the object has input focus
HitTest()	METHOD	Determines if the mouse cursor is within the box
HotBox	ACC/ASS	Frame of list box with focus
InputBlock	ACC/ASS	CodeBlock for default/user keyboard handler
InsItem()	METHOD	Insert a new item to a list box
IsOpen	ACC	Indicator whether the combo box widget is visible
ItemCount	ACC	Number of items in the list
KillFocus()	METHOD	Take input focus away from a ListBox object
Left	ACC/ASS	Leftmost screen column of the box
ListFiles()	METHOD	Fill a list box with the names of matching files
Message	ACC/ASS	String displayed in the windows status bar
Modified	ACC/ASS	Ignored.
NextItem()	METHOD	Skip to the next available item
NextSelected()	METHOD	Skip to the next selected item
Open()	METHOD	Opens the combo box (drop-down box)
PrevItem()	METHOD	Skip to the previous available item
Right	ACC/ASS	Rightmost screen column of the box
Sblock	ACC/ASS	Code block evaluated at user selection
Scroll()	METHOD	Scrolls the contents of a list box up or down
Select()	METHOD	Change the selected item in a list
SelectBySingleClick	ACC/ASS	Allow selection by left mouse same as Enter
SelectBySpace	ACC/ASS	Allow selection by space key same as Enter
SelectedCount	ACC	Number of items selected in a multiple selection
SelectedFile	ACC	Selected file filled by :ListFiles()
SelectItem()	METHOD	Change the selected item in a list
SetData()	METHOD	Change the property of an available item
SetFocus()	METHOD	Set input focus to a ListBox object

SetItem()	METHOD	Replaces the item property
SetText()	METHOD	Change/replace the displayed text of item
SetTop()	METHOD	Move a specified item to the top of the list box
Show()	METHOD	Show the list box and its caption on the screen
TextValue	ACC/ASS	String representing the displayed listbox item
ToolTip	ACC/ASS	Short pop-up info message
Top	ACC/ASS	Topmost screen row of the box
TopItem	ACC/ASS	Position of the first visible item
TypeOut	ACC/ASS	Indicator whether the list contains any items
Value	ACC/ASS	Any value associated with the specified item
ValueChanged	ACC/ASS	Indicator representing the status of :Value
Vscroll	ACC/ASS	Ignored in FlagShip

ListBox Class Instantiation

oListBox := ListBox { [nR1],[nC1], [nR2],[nC2], [lCombo], [lPixel] } ***[1]***
oListBox := ListBoxNew ([nR1],[nC1], [nR2],[nC2], [lCombo], [lPixel]) ***[2]***

oListBox := ListBox ([nR1],[nC1], [nR2],[nC2], [lCombo], [lPixel]) ***[3]***

Any of the above syntax instantiate new list box (or combo box) object. Syntax [3] is also compatible to Clipper

The list box widget (control) remains invisible until you invoke oListBox:Show() or oListBox:Display(). This allows the program to set up the control correctly (with the correct size, position, and any other parameters), while avoiding the "visual noise" of changing controls. Arguments:

<**nR1**> topmost row in coordinates or pixel, optional. If not specified, 0 is the default

<**nC1**> leftmost column in coordinates or pixel, optional. If not specified, 0 is the default

<**nR2**> bottom row in coordinates or pixel, optional. If not specified, MaxRow() is default

<**nC2**> rightmost column in coordinates or pixel, optional. If not specified, MaxCol() is the default

<**lCombo**> if true (.T.), ComboBox (drop-down box in Clipper terminology) is used instead of ListBox. Optional, default is .F.

<**lPixel**> if true(.T.), the row and column data are in pixel; if false (.F.), data are in row/col coordinates, otherwise the current SET PIXEL is used. Apply for GUI mode only, ignored otherwise.

Tuning: The coordinates <nR1>...<nC2> usually specifies the outer box frame, common for both GUI and Terminal i/o mode. If you wish in GUI mode these coordinates specify the inner box, set

```
_aGlobSetting[GSET_G_L_LISTBOX_BOX] := .F.
```

If you don't wish to automatically adjust row/col in GUI mode, set

```
_aGlobSetting[GSET_G_L_LISTBOX_ADJ ] := .F.  // default = .T.
```

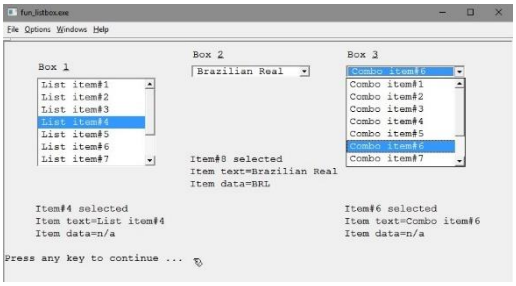
If the above adjustment is on (.T.), you may set the pixel values

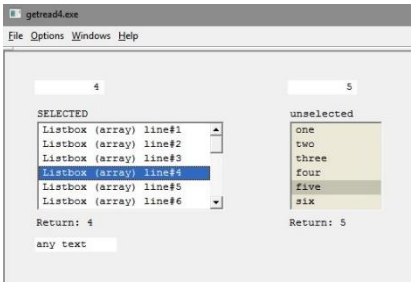
```
_aGlobSetting[GSET_G_N_LISTBOX_TOP ] := -2   // default
_aGlobSetting[GSET_G_N_LISTBOX_BOT ] := 2    // default
_aGlobSetting[GSET_G_N_LISTBOX_LEFT] := -7   // default
_aGlobSetting[GSET_G_N_LISTBOX_RIGH] := 6    // default
_aGlobSetting[GSET_G_N_COMBO_HEIGHT] := 4    // default
```

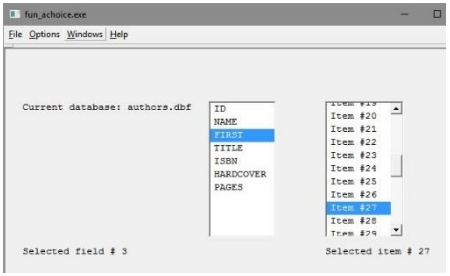
Example 1: This example creates and fills a list box with a list of animals:

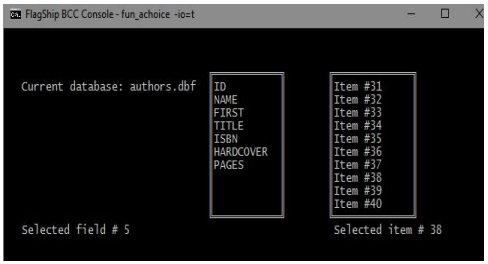
```
oLB := ListBox{5,1, 9,15, .F., .F.}
oLB:AddItem("Mouse")
oLB:AddItem("Cat")
oLB:AddItem("Dog")
iSelected := oLB:Show()    // or oLB:Exec()
```

Example 2: see further examples in FUN.ListBox(), ComboBox(), Achoice() and CMD.@...GET LISTBOX, @...GET COMBOBOX









Compatibility: Available also in Clipper5.3 (syntax 3 w/o <lPixel>). In VFS6 and VFS7 was <lPixel> in syntax 1 and 2 optional 5th parameter which conflicted with syntax 3.

See also: oListBox:Destroy()

ListBox Class Properties

oListBox:AddItem(cText, [nPos], [exp], [lSelect], [lBitmap]) —> nRet
oListBox:AddItem(cText, [exp]) —> nRet

Add (append) a new item to a list box at a specified position or at the list end. When present, the scroll bar is automatically updated to reflect the addition of the new item.

<**cText**> Character string of the item to be inserted/added and displayed in the list. You may specify hot-key for this item by prefacing the selectable character by "&" or "\&" or "\<". Otherwise the first character of <cText> is the hotkey.

<**nPos**> The position in the list box at which to insert the new item. Specify one of the following values (default is 0 = add):

0: In an unsorted list box, adds the new item at the end of the list; if sorted, inserts the new item at a position determined by the list box. This is the default setting
-1: Always adds the item at the list end
1: The first position in the list box
n The n-th position in the list box

<**exp**> Any value associated with the specified item, which enables to associate pertinent data with the text displayed in the list. The default is NIL

<**lSelect**> optional logical value specifying if the item is selectable (TRUE, the default) or not (FALSE). This can be re-defined by oListBox:DeselectItem()

<**nBitmap**> optional logical value specifying that the <cText> is a name of a bitmap which should be displayed instead of the text (TRUE), or if <cText> is a usual text value to be displayed as such (FALSE, the default).

<**nRet**> If the item was added, its position in the list box is returned (a value of 1 refers to the first position in the list box). If the item could not be added, 0 is returned.

Compatibility: Available also in CL53 and VO. In VO, the 1st format is used with max. three parameters. CL53 uses the 2nd format and the method return SELF instead.

See also: oListBox:FillUsing(), oListBox:InsItem(), oListBox:SetData(), oListBox:SetText(), oListBox:GetData(), oListBox:GetText()

oListBox:Bitmap —> cFile ***ACCESS***
oListBox:Bitmap := cFile ***ASSIGN***

<**cFile**> is a character string that indicates a bitmap file to be displayed as list box item. The type of the bitmap is determined from the file name extension, supported are currently .bmp, .gif, .jpeg, .jpg, .png, .ppm and .xpn. If no path is

given, the bitmap file must reside in the same directory as the application or in a directory specified by SET DEFAULT command. If no file is found, text "(bitmap)" will be displayed instead of bitmap. Apply only for GUI mode, otherwise "(bitmap)" text is displayed.

Compatibility: Available also in CL53.

not available yet

oListBox:Bottom —> nRow *ACCESS*
oListBox:Bottom := nRow *ASSIGN*

<**nRow**> is a numeric value that indicates the bottommost screen row where the list box is displayed. The input and output value is either in coordinates or in pixels, depending on the current SET PIXEL setting.

Compatibility: Available also in CL53.

See also: oListBox:Top, oListBox:CapCol

oListBox:Buffer —> nPos *ACCESS*

<**nPos**> is a numeric value that indicates the position in the list of the selected item.

Compatibility: Available also in CL53.

See also: oListBox:CurrItemNo

oListBox:CapCol —> nCol *ACCESS*
oListBox:CapCol := nCol *ASSIGN*

<**nCol**> is a numeric value that indicates the screen column where the list box's caption is displayed. The input and output value is either in coordinates or in pixels, depending on the current SET PIXEL setting.

Compatibility: Available also in CL53.

See also: oListBox:CapRow, oListBox:Caption

oListBox:CapRow —> nRow *ACCESS*
oListBox:CapRow := nRow *ASSIGN*

<**nRow**> is a numeric value that indicates the screen row where the list box's caption is displayed. The input and output value is either in coordinates or in pixels, depending on the current SET PIXEL setting.

Compatibility: Available also in CL53.

See also: oListBox:CapCol, oListBox:Caption

oListBox:Caption –> cText *ACCESS*
oListBox:Caption := cText *ASSIGN*

<**cText**> is a string that describes the list box caption. When present, the & character specifies that the character immediately following it in the caption is the list box's accelerator key. The accelerator key provides a quick and convenient mechanism for the user to move input focus from one data input control to a list box. The user performs the selection by pressing the Alt key in combination with an accelerator key. The case of an accelerator key is ignored.

Compatibility: Available also in CL53 and VO.

See also: oListBox:CapCol, oListBox:Caption

oListBox:Cargo –> exp *ACCESS*
oListBox:Cargo := exp *ASSIGN*

<**exp**> is a value of any type. The ListBox:Cargo slot holds any user- definable data which can be retrieved later. This property is not used by the standard ListBox object itself.

Compatibility: Available also in CL53.

oListBox:ChangeSelected(oRange, [lEnable]) –> lOk

Change a range of items in a multiple selection list box to a specified selection. This method is intended for use with a multiple selection list box (i.e., a list box created using the LBS_MULTIPLESEL style).

<**oRange**> The Range object representing the selected items.
<**lEnable**> The state of the selected items. If not specified, the default is TRUE.
<**lOk**> is TRUE (.T.) if successful; otherwise FALSE (.F.).

Compatibility: Available also in VO.

See also: oListBox:Select(), oListBox:ClearSelection()

oListBox:ClassName() –> "LISTBOX" or "COMBOBOX"

Returns fix "LISTBOX" or "COMBOBOX".

oListBox:Clear() –> NIL

Clear (delete) all items in a list box.

Compatibility: Available also in VO

See also: oListBox:DeleteItem(), oListBox:AddItem(), oListBox:FillUsing()

oListBox:ClearSelection() —> lOk

Clear a multiple selection list box of all selections. This method is intended for use with a multiple selection list box (i.e., a list box created using the LBOXMULTIPLESEL style).

<**lOk**> is TRUE (.T.) if successful; otherwise FALSE.

Compatibility: Available also in VO.

See also: oListBox:Select(), oListBox:ChangeSelected()

oListBox:Close() —> self

Closes the combo box ("drop-down list box" in Clipper terminology) and restores the screen previously visible in this area.

Compatibility: Available also in CL53.

See also: oListBox:Open()

oListBox:ColdBox —> cBox *ACCESS*
oListBox:ColdBox := cBox *ASSIGN*

<**cBox**> is an optional string that specifies the characters to use when drawing a box around the list box when it does not have input focus. Considered in Terminal mode only, ignored in GUI. Its default value is a single line box. Predefined <cBox> constants are in the box.fh file:

B_SINGLE	Single line box
B_DOUBLE	Double line box
B_SINGLE_DOUBLE	Single line top/bottom, double line sides
B_DOUBLE_SINGLE	Double line top/bottom, single line sides

Compatibility: Available also in CL53. This property is considered in terminal mode only and ignored otherwise.

See also: oListBox:HotBox, oListBox:SetFocus(), @..BOX

oListBox:ColorSpec —> cAttrib *ACCESS*
oListBox:ColorSpec := cAttrib *ASSIGN*

<**cAttrib**> is a character string specifying the color attributes that are used by the list box's display() method. If the list box is a combo box (drop-down list box in Clipper terminology), the string can contain eight color specifiers, otherwise it should contain at least seven color specifiers for a usual list box.

Position in <cAttrib>	Applies To	Default value used from curr SET COLOR
1	Unselected items, without input focus	1=Std
2	Selected item, without input focus	5=Unselected
3	Unselected items with input focus	1=Std
4	Selected item with input focus	2=Enhanced
5	The list box's border	3=Border
6	The list box's caption	1=Standard
7	The list box caption's accelerator key	4=Background
8	The list box's drop-down button	1=Standard
9	Disabled items	5=Unselected

You may change the assignment to SetColor() pair by

```
_aGlobSetting[GSET_A_ACHOICE_LBOX_COLOR] := {1,5,1,2,3,1,4,1,5}
```

Compatibility: Available also in CL53, This property is considered in terminal mode only and ignored otherwise. For GUI mode, see :GuiColor

See also: oListBox:HasFocus, oListBox:GuiColor, SET COLOR, SET()

oListBox:CurrentItem –> cText *ACCESS*
oListBox:CurrentItem := cText *ASSIGN*

<**cText**> is a string representing the displayed list box or combo box item selected. The ListBox:CurrentItem access also changes ListBox:CurrItemNo, ListBox:CurrentText, ListBox:TextValue, and ListBox:Value, if there is a match with the available display items.

Compatibility: Available also in VO

See also: oListBox:setText(), oListBox:getItem()

oListBox:CurrItemNo –> nPos *ACCESS*
oListBox:CurrItemNo := nPos *ASSIGN*

<**nPos**> is a numeric value, between 1 and the ListBox:ItemCount, indicating which item is currently selected. If no item is selected, it is 0.The ListBox: CurrentItemNo assign also changes ListBox:CurrentItem, ListBox:TextValue, and ListBox:Value. If the assigned <nPos> is zero, or if it exceeds the ListBox:ItemCount, then no item will be selected. If the ListBox: CurrItemNo assign represents a change, then ListBox: ValueChanged will be set to TRUE.

Compatibility: Available also in VO as ListBox:CurrentItemNo

See also: oListBox:CurrentItem, oListBox:getItem()

oListBox:CurrentText —> cText *ACCESS*
oListBox:CurrentText := cText *ASSIGN*

<**cText**> is set to the null string "" in ListBox and ComboBox, since there is no text editing for list boxes.

Compatibility: Available also in VO

See also: oListBox:CurrentItem

oListBox:DeleteItem([nPos]) —> lOk

Remove an item from a list box.

<**nPos**> The number of the item to be deleted. Valid values are 1 to oListBox:ItemCount or 0 (the default) specifying the currently selected item.

<**lOk**> returns TRUE (.T.) if successful, otherwise FALSE.

Compatibility: Available also in VO

See also: oListBox:DelItem(), oListBox:Clear()

oListBox:DelItem(nPos) —> self

This method is equivalent to oListBox:DeleteItem([nPos]) and is intended for backward CL53 compatibility.

Compatibility: Available also in CL53

See also: oListBox:DeleteItem(), oListBox:GetItem()

oListBox:DeselectItem([nPos]) —> lOk

Turn off the selection of a specified item in a list box. Normally, the user turns selections off, but this method enables the program to do so also. This method is intended for use with a multiple selection list box (i.e., a list box created using the LBOXMULTIPLESEL style).

<**nPos**> The position of the item (1 to oListBox:ItemCount or 0 for the currently selected item) to be deselected.

<**lOk**> returns TRUE if successful; otherwise, FALSE.

Compatibility: Available also in VO

See also: oListBox:Select(), oListBox:ChangeSelected()

oListBox:Destroy() —> NIL

Destroys the ListBox object and restores the previous screen content. This method can be used when a ListBox object is no longer needed. oListBox:Destroy() de-instantiates the ListBox object and allows you to close and free any resources that were opened or created by the object, without waiting for the garbage collector. This method calls internally oListBox:Axit() which is the equivalence for :Destroy()

Compatibility: Available also in VO

See also: ListBox{} instantiation

oListBox:Display() —> self

Show the list box and its caption on the screen. The list box widget (control) remains invisible until you invoke oListBox:Display() or oListBox:Show(). This allows the program to set up the control correctly (with the correct size, position, and any other parameters), while avoiding the "visual noise" of changing controls. oListBox:Display() uses the values of the following instance variables to correctly show the list in its current context, in addition to providing maximum flexibility in the manner a list box appears on the screen:

:Bottom, :CapCol, :CapRow, :Caption, :ColdBox, :ColorSpec, :HasFocus, :HotBox, : ItemCount, :Left, :Right, :Style, :Top, :TopItem, :vScroll, :CurrItemNo.

This method is similar to oListBox:Show(), but does not enter the event handler automatically, i.e. does not provide user input. The listbox page starting with :TopItem is displayed and the method returns. With ComboBox, only the closed box is displayed, except :Open() was called previously.

Compatibility: Available also in CL53

See also: oListBox:Show()

oListBox:DropDown —> lCombo *ACCESS*

<**lCombo**> is a logical value indicating whether the object is a combo box (TRUE), which is "drop-down list box" in Clipper terminology, or a usual list box (FALSE).

Compatibility: Available also in CL53

See also: ListBox{...} instantiation

oListBox:Exec([naComboOpen], [naComboClose]) —> nSelItem

This method is equivalent to oListBox:Show([naComboOpen],[naComboClose]). It shows the list box and its caption on the screen and process keyboard/ mouse input. In detail: It set input focus :SetFocus(), calls :Display(), enter and process the default or user's event/keyboard handler specified in :InputBlock, then clears the input focus by :KillFocus()

Compatibility: Available in FS only

See also: oListBox:Display, oListBox:SetFocus(), oListBox:HasFocus. oListBox:InputBlock

oListBox:Fblock —> bBlock *ACCESS*
oListBox:Fblock := bBlock *ASSIGN*

<**bBlock**> is a code block or NIL. The code block callback, when present, is evaluated each time the ListBox object receives or loses input focus. The code block receives two arguments: the object self and the current :HasFocus status, which indicates whether the list box is receiving (.T.) or losing (.F.) input focus. In GUI, the object receives focus every times the user clicks (or activates) the list box widget and looses focus when other widget is selected. You should not use Inkey() nor other input commands or functions in the callback UDF.

Compatibility: Available also in CL53, but Clipper does not pass any arguments to the code block, and hence cannot use generalized but object specific code blocks which needs to check the current oListBox:HasFocus status by itself.

See also: oListBox:HasFocus, oListBox:SetFocus(), oListBox:KillFocus(), oListBox:Sblock

oListBox:FillUsing([aText]) —> NIL
oListBox:FillUsing([oRdd], [field1], [field2], [field3], [field4]) —> NIL

Specify the set of values to be displayed in the list box, using an array or a data server. These values act as a constraint on the values that may be entered in the list box, and optionally as a translation between program values and display values.

<**aText**> An array containing the values to be placed in the list box. Either one- or multi-dimensional array may be used to define the text items shown in the list, values returned to the program when a list item is selected, and optional flags. Format of the <aText> array:

1. One-dimensional array containing strings to be displayed in the list. Other item properties are set to defaults, i.e. the returned value is NIL, the item is selectable and is not a bitmap
2. Multi-dimensional array of up to four elements each, containing [1] the string to be displayed in the list [2] optional: the corresponding value of any type returned to the program at selection (default is NIL),

[3] logical value indicating that the item is selectable (default is TRUE), and [4] logical value indicating whether the item is a bitmap whose file name is stored in the first element (default is FALSE).

<**oRdd**> The data server object that is to be used to provide the set of values. If not specified, the currently selected work area is used. You also may use the DbObject() function to provide the Rdd object.

<**field1**> The field name or it position in the record (corresponding to FieldPos() return value) that is to be used for the display values. If not specified, the values of the first field are used. The field needs to be CHAR type.

<**field2**> The field name or it position in the record that is to be used for the values that are returned to the program. The field can be of any type. If the field is not specified, NIL is used.

<**field3**> The name or position of a logical field that is to be used to indicate whether the item is selectable. If not specified or if the filed is not logical type, .T. is the default.

<**field4**> The name or position of a logical field that is to be used to indicate whether the first field is a name of bitmap file. If not specified, .F. is the default.

A list box shows the set of valid values for a field. Depending on what type of list box is used, the set of values may act as a constraint on the values that may be retrieved or only as a suggestion. Two sets of values may be specified, allowing for translation of values between the displayed, human-readable representation and the internal, programmatic value.

On database use, the current record is filled first and then the database is SKIP()ped forward filling the list box, until EOF() or until the end of scope is reached. On exit, the database is reset to its original state, i.e. same as on entering this method.

The :FillUsing() method provides a way of specifying the values to be included in the list all at once, instead of constructing the list item by item with the :AddItem() method. Note that this method add the items to the list, so you may freely combine several :FillUsing() and/or :AddItem() invocations as shown in the example.

Example:

Create a list box with different currencies, showing an explicit representation to the user but using a different representation internally. It also adds data from a database, selecting Asian currencies and using fields "CurrName" for the text as well as the 3rd field for the returned value.

```
oLBCurrency := ListBox{10, 10, 200, 400, , .T.}  // instantiate
oLBCurrency:FillUsing( {{"U.S. Dollars", "USD"},;
                        {"Can. Dollars", "CDN"},;
                        {"Mexican Pesos", "MEX"},;
                        {"Yen", "YEN"},;
                        {"British Pounds", "UK"},;
                        {"German Marks", "DM", .F.}, ;
                        {"Euro", "EUR"}} )           // defaults
```

```
oLbCurrency:AddItem("non selectable", -1, NIL, .F.)  // separator
USE currency INDEX currency SHARED NEW              / add from dbf
SET FILTER to upper(trim(CurrArea)) == "ASIA"
GO TOP                                          // find first matching
oLbCurrency:FillUsing(NIL, "CurrName", 3)   // add fields (1) and 3
oLbCurrency:Sblock := {|obj,pos,txt,val| ;
   alert("selected text =[" + txt + "];value =[" + ;
         transform(val) + "]") }             // action used
oLBCurrency:Show()                           // show and process
```

Compatibility: Available also in VO, which supports up to two-dimensional array or up to two fields.

See also: oListBox:AddItem(), oListBox:DeleteItem(), oListBox:Clear(), oListBox:SetData(), oListBox:SetText(), oListBox:GetData(), oListBox:GetText()

oListBox:FindItem(cText, [lWhole]) –> nPos

Search a list box for a specified string, and return the location of the first item in the list box that matches it. Note, this is a subset of the oListBox:FindText() method for VO compatibility and is equivalent to nPos := oListBox:FindText(cText, 1, .T., lWhole).

<**cText**> The text to search for.

<**lWhole**> Indicates how the search is to be performed. TRUE matches an exact <cText> string to a "whole" list box item text (for example, a "can" string does not match "scan"). FALSE finds a match for any list box prefixed by <cText> (for example, the string "cat" would match "catalog" in the list box). The default is TRUE.

<**nPos**> Returned numeric value indicating the position of the first item that contains the matching text, if a match is found (a value of 1 refers to the first position in the list box); otherwise 0 is returned if no match is found.

Compatibility: Available also in VO

See also: oListBox:FindText()

oListBox:FindText(cText, [nStart], [lCase], [lExact], [lShort], [lLeftTrim], [lOnlySel]) –> nPos

Search a list box for a specified string, and return the location of the first item in the list box that matches it.

<**cText**> The text to search for.

<**nStart**> Optional numeric value that indicates the starting position in the list of the search. The default is 1. The search starts from the <nStart> position to the end of the list and, when necessary, continues from the beginning of the list to <nStart> - 1

<**lCase**> Optional logical value that indicates whether the search should be case sensitive. TRUE (default) performs the search case sensitive, FALSE searches regardless the case.

<**lExact**> Optional logical value that indicates whether the search enforces an exact comparison including length and trailing characters. TRUE value indicates to search by an exact match using the == comparison by ignoring trailing spaces (i.e. "catalog" would match "catalog ", but not "my catalog "); a FALSE (the default) value compares only the <cText> size of the list text for equivalence (i.e. "cat" would match "catalog" in the list box).

<**lShort**> Optional logical value indicating whether "&" shortkey should be searched first. Default is .F.

<**lLeftTrim**> Optional logical value indicating whether the text should be left trimmed first. Default is .F.

<**lOnlySel**> Optional logical value indicating that only selectable items should be searched. Default is .F.

<**nPos**> Returned numeric value indicating the position of the first item that contains the matching text, if a match is found (a value of 1 refers to the first position in the list box); otherwise 0 is returned if no match is found.

Compatibility: Available also in CL53 (first 4 params only)

See also: oListBox:FindItem()

oListBox:FirstSelected() —> nPos

Returns the position of the first item selected in a multiple selection list box, or 0 if no item is selected. oListBox:FirstSelected() positions an imaginary cursor on the first item of the selection. This method is intended for use with a multiple selection list box (i.e., a list box created using the LBOXMULTIPLESEL style).

Compatibility: Available also in VO

See also: oListBox:CurrItemNo, oListBox:SelectedCount, oListBox:NextSelected(), oListBox:Select(), oListBox:ChangeSelected(), oListBox:DeselectItem(), oListBox:Clear()

oListBox:Font —> oFont *ACCESS*
oListBox:Font := oFont *ASSIGN*

<**oFont**> is a Font object (or NIL) used to display the list box items. If not set, the default application font oApplic:Font is used.

Compatibility: Available also in VO. Ignored in non-GUI mode.

See also: Font class, oApplic:Font

oListBox:GetData([nPos]) —> exp

Retrieves the data portion of a list box item associated with the item but not displayed in the list.

<**nPos**> numeric value that indicates the position within the list of the item whose data is being retrieved. 0 (zero, the default) specifies the currently selected item, 1 to :ItemCount is the requested item number otherwise.

<**exp**> Returned value of any type associated to the list box item by :AddItem(), :FillUsing(), :SetData() etc.

Compatibility: Available also in CL53 where the parameter is not optional and which does not support 0 for <nPos>.

See also: oListBox:GetText(), oListBx:GetItem(), oListBox:SetData(), oListBox:SetItem()

oListBox:GetItem([nPos]) —> aData

Retrieves the item property, i.e. the displayed text, associated data and additional flags returning these in one-dimensional array.

<**nPos**> numeric value that indicates the position within the list of the item whose data is being retrieved.0 (zero, the default) specifies the currently selected item, 1 to :ItemCount is the requested item number otherwise.

<**aData**> Returned one-dimensional array with four elements containing the item properties: Element [1] is the displayed text, [2] the associated data of any type, [3] a logical value specifying whether the item is selectable, [4] a logical value indicating whether the 1st element is a file name of a bitmap or a usual text.

Compatibility: Available also in CL53, which requires parameter but does not support 0 input, and return only the first two array elements. The same named VO method has different meaning and is equivalent to oListBox:GetText() in FS and CL53.

See also: oListBox:GetText(), oListBox:GetData(), oListBox:SetText(), oListBox:SetData(), oListBox:SetItem()

oListBox:GetItemValue([nPos]) —> exp

This is a VO equivalence for oListBox:GetData() method, and available for compatibility purpose.

Compatibility: Available also in VO.

See also: oListBox:GetData(), oListBox:GetText(), oListBox:GetItem()

oListBox:GetText([nPos]) –> cText

Retrieves the text portion of a list box item displayed in the list.

<**nPos**> numeric value that indicates the position within the list of the item whose data is being retrieved. 0 (zero, the default) specifies the currently selected item, 1 to :ItemCount is the requested item number otherwise.

<**cText**> Returned character value associated to the list box item by :AddItem(), :FillUsing(), :SetText() etc.

Compatibility: Available also in CL53 which does not support 0 input.

See also: oListBox:GetData(), oListBox:GetItem(), oListBox:SetText(), oListBox:SetItem()

oListBox:GuiColor –> cAttrib *ACCESS*
oListBox:GuiColor := cAttrib *ASSIGN*

<**cAttrib**> is a character string specifying the color attributes that are used by the list box's display() method in GUI mode. The string can contain four color specifiers in the SET COLOR syntax:

Position in <cAttrib>	Applies To	Default
1	Unselected items, without input focus	black/white
2	Selected item, without input focus	white/blue
3	Unselected items with input focus	black/white
4	Selected item with input focus	white/blue
5	Disabled items	#B0B0B0/white

To use default colors, skip it or specify N/N for the item or assign empty string "" to disable all, which is the default setting. Note that the standard background for selected item (with and without input focus) is usually set by the window manager and may hence differ according to the used platform. It is usually `W+/RGB(49,106,195) = W+/#316AC3` in Windows, and `W+/RGB(8,93,139) = W+/#085D8B` in Linux/KDE.

Example: display items in Listbox w/o focus on default background with grey bar, and items in focused Listbox using default colors

```
oLB := ListBox{5,1, 9,15}
oLB:AddItem(...)
oLB:InputBlock := ...
// either common for all platforms:
*  oLB:GuiColor := ",W+/#C0C0C0,,"    // or  ",W+/#C0C0C0"
// or platform specific:
#ifdef FS_WIN32
  oLB:GuiColor := "N/#ECE9D8,W+/#C0C0C0,N/W+,W+/#316AC3"
#else
  oLB:GuiColor := "N/#DEDEDE,W+/#C0C0C0,N/W+,W+/#085D8B"
#endif
item := oLB:Show()
```

Compatibility: This property is considered in GUI mode only and is ignored otherwise. For Terminal i/o mode, see :ColorSpec

See also: oListBox:HasFocus, oListBox:ColorSpec, SET COLOR, SET()

oListBox:HasFocus —> lFocus *ACCESS*

<**lFocus**> is a logical value indicating whether the object has input focus (TRUE) or not. In GUI, the object receives focus every times the user clicks (or activates) the widget and looses the focus when other widget is selected.

Compatibility: Available also in CL53

See also: oListBox:KillFocus, oListBox:SetFocus(), oListBox:Fblock

oListBox:HitTest(nMouseRow, nMouseCol, [lPixel]) —> nStatus

Determines if the mouse cursor is within the region of the screen that the list box occupies.

<**nRow**> Numeric value representing the current or tested screen row position of the mouse cursor.

<**nCol**> Numeric value representing the current or tested screen row position of the mouse cursor.

<**lPixel**> If specified TRUE, the mouse coordinates are assumed in pixel. If FALSE, the mouse parameters are assumed in current row/col coordinates. If this parameter is not specified (i.e. NIL), the kind of passed mouse coordinates is determined from the current SET PIXEL setting.

<**nStatus**> Returned numeric value indicating the relationship of the mouse cursor with the list box. The constants are specified in button.fh header file.

Value	Constant	Description
> 0	HTITEMS	The mouse is located on one of the list box items
0	HTNOWHERE	The mouse cursor is not within the region of the screen that the listbox or combobox occupies
-1	HTTOPLEFT	The mouse cursor is on the top left corner of the list box's border
-2	HTTOP	The mouse cursor is on the list box's top border
-3	HTTOPRIGHT	The mouse cursor is on the top right corner of the list box's border
-4	HTRIGHT	The mouse cursor is on the list box right border
-5	HTBOTTOMRIGHT	The mouse cursor is on the bottom right corner of the list box's border
-6	HTBOTTOM	The mouse cursor is on the list box bottom border
-7	HTBOTTOMLEFT	The mouse cursor is on the bottom left corner of the list box's border
-8	HTLEFT	The mouse cursor is on the list box's left border

-1000	HTSCROLLBAR	The mouse cursor is on listbox scrollbar
-1025	HTCAPTION	The mouse cursor is on the list box's caption
-4097	HTDROPBUTTON	The mouse cursor is on the ComboBox drop-down button
-5121	HTCELL	The mouse cursor is in ComboBox widget

Compatibility: Available also in CL53

See also: Mrow(), Mcol()

oListBox:HotBox –> cBox *ACCESS*
oListBox:HotBox := cBox *ASSIGN*

<**cBox**> is an optional string that specifies the characters to use when drawing a box around the list box when it has input focus. Its default value is a single line box. Predefined <cBox> constants are in the box.fh header file:

B_SINGLE	Single line box
B_DOUBLE	Double line box
B_SINGLE_DOUBLE	Single line top/bottom, double line sides
B_DOUBLE_SINGLE	Double line top/bottom, single line sides

Compatibility: Available also in CL53. This property is considered in terminal mode only and ignored otherwise.

See also: oListBox:ColdBox, oListBox:HasFocus, oListBox:SetFocus(), oListBox:ColorSpec, @..BOX

oListBox:Init([par1]...[par6]) –> self

This is an internal method invoked automatically at instantiation of the ListBox object. It is not intended to be called by the application.

Compatibility: Available also in VO

See also: ListBox{} instantiation

oListBox:InputBlock –> bHandler *ACCESS*
oListBox:InputBlock := bHandler *ASSIGN*

This property stores a code block, evaluated during :Show() or :Exec(). The code block usually calls the default or user specified keyboard and mouse handler, processing the listbox selection. The :Show() or :Exec() method passes four parameters to it: 1) the ListBox object self, 2) the pre-selected item number, 3) and 4) an inkey() value (or an array of numeric values) specifying the keys to open or close combo (drop-down) box, see oListBox:Show(). The code block should return numeric value specifying the selected item number (1 to :ItemCount) or 0 on selection abort. This return value is returned by :Show() or :Exec() and used in standard functions based on Listbox, for example in Achoice()

When :InputBlock was not instantiated yet or is NIL, the default input handler ListBoxHandler(), available in source in listboxhand.prg, is assigned to :InputBlock and called by :Show() or :Exec().

Alternatively, your application may call :Display() and invoke/process your input handler directly (w/o :Show(), same as Clipper do).

Example:

```
oLB := ListBox{5,1, 9,15}
oLB:AddItem("One")
oLB:AddItem("Two")
oLB:AddItem("Three")
oLB:InputBlock := {|obj,item,iOpen,iClose| ;
                       myListBoxHandler(obj,item,iOpen,iClose) }
item := oLB:Show()
setpos(10,0)
? "Selected item:", ltrim(item)
```

Compatibility: Available in VFS only

See also: oListBox:Display(), oListBox:Show(), oListBox:Exec()

oListBox:InsItem(nPos, cText, [exp], [lSelect], [lBitmap]) —> exp

Insert a new item to a list box at a specified position. This method is equivalent to the oListBox:AddItem(cText,nPos,[exp],[lSelect],[lBitmap]) method.

Compatibility: Available also in CL53 which support first three parameters.

See also: oListBox:AddItem(), oListBox:GetText(), oListBox:GetItem(), oListBox:SetData(), oListBox:SetItem(), oListBox:DeleteItem(), oListBox:Clear()

oListBox:IsOpen —> lStat *ACCESS*

<**lStat**> is a logical value indicating whether the combo box widget is fully visible (TRUE) or if the combo box shows the current value only (FALSE). With list box, oListBox:IsOpen always return TRUE.

Compatibility: Available also in CL53

See also: oListBox:Open(), oListBox:Close()

oListBox:ItemCount —> nCount *ACCESS*

<**nCount**> is a numeric value indicating the number of items in the list box.

Compatibility: Available also in CL53 and VO

See also: oListBox:AddItem(), oListBox:InsItem(), oListBox:FillUsing(), oListBox:DeleteItem(), oListBox:Clear()

oListBox:ItemSelectable(nPos, [lSet]) —> lRet

Set or check item to be selectable

<**nPos**> is a numeric value specifying the item position in the list

<**lSet**> is a logical value, .T. sets the item selectable, .F. disables it

<**lRet**> is a logical value reporting .T. when the item is selectable

See also: oListBox:AddItem(), oListBox:InsItem()

oListBox:KillFocus() —> self

Take input focus away from a ListBox object. Upon receiving this message, the ListBox object redisplays itself with the :ColdBox frame and, if present, evaluates the code block specified by :Fblock. This message is meaningful only when the ListBox object has input focus.

Compatibility: Available also in CL53.

See also: oListBox:HasFocus, oListBox:SetFocus(), oListBox:Fblock

oListBox:Left —> nCol *ACCESS*
oListBox:Left := nCol *ASSIGN*

<**nCol**> is a numeric value that indicates the leftmost screen column where the list box is displayed. The input and output value is either in coordinates or in pixels, depending on the current SET PIXEL setting.

Compatibility: Available also in CL53.

See also: oListBox:Right, oListBox:Top, oListBox:Bottom

oListBox:ListFiles([cDir], [oFixedText], [nFileType]) —> lOk

Fill a list box with the names of all files that match the specified path or file name. This method is unsupported by FlagShip and hence returns FALSE. You may use instead:

```
aDirList := directory(...)
oListBox:FillUsing(aDirList)
```

Compatibility: Available also in VO.

See also: oListBox:SelectedFile

oListBox:Message –> cText *ACCESS*
oListBox:Message := cText *ASSIGN*

<**cText**> is a character string displayed in the windows status bar (GUI), or in the screen line specified by SET MESSAGE (in terminal mode).

Compatibility: Available also in CL53.

See also: oListBox:Tooltip(), SET MESSAGE, oApplic:StatusMessage()

oListBox:Modified –> lOk *ACCESS*
oListBox:Modified := lOk *ASSIGN*

Ignored. This property include logical value that is always set to FALSE for a list box, since it does not contain text that can be edited.

Compatibility: Available also in VO.

oListBox:NextItem() –> self

Changes the selected item from the current item to the one immediately following it. If necessary, :NextItem() will call its :Display() or :Scroll() method to ensure that the newly selected item is visible. This message is meaningful only when the list box object has input focus. As opposite to the similar oListBox:NextSelected(), this method changes the "selected" item flag.

Compatibility: Available also in CL53.

See also: oListBox:NextSelected(), oListBox:PrevItem(), oListBox:FirstSelected(), oListBox:Select()

oListBox:NextSelected() –> nPos

After calling oListBox:FirstSelected(), this method is used to cycle through the remaining items selected in a multiple selection list box. As opposite to the similar oListBox:NextItem(), this method does not change the "selected" item flag. This method is intended for use with a multiple selection list box.

<**nPos**> is a position of the next selected item in the list box, or 0 if no item is selected or if there are no remaining items.

Compatibility: Available also in VO.

See also: oListBox:NextItem(), oListBox:FirstSelected()

oListBox:Open() —> self

Opens the combo box (drop-down list box in Clipper terminology) and saves the screen previously visible in this area.

To open the ComboBox, use the TAB, Space or # key, and shift-TAB or ^ key to close the box. These keys are user-modifiable by :Exec(...) or by assigning corresponding inkey-value(s) to

```
_aGlobSetting[GSET_A_COMBO_OPEN ]:= {K_DOWN, K_SPACE, 35} // default
_aGlobSetting[GSET_A_COMBO_CLOSE]:= {K_TAB, K_SH_TAB, 94} // default
```

before invoking the oComboBox:Display() or oComboBox:Exec().

Compatibility: Available also in CL53.

See also: oListBox:Close()

oListBox:PrevItem() —> self

Changes the selected item from the current item to the one immediately following it. If necessary, :NextItem() will call its :Display() or :Scroll() method to ensure that the newly selected item is visible. This message is meaningful only when the list box object has input focus. As opposite to the similar oListBox:NextSelected(), this method changes the "selected" item flag.

Compatibility: Available also in CL53.

See also: oListBox:NextItem(), oListBox:NextSelected(), oListBox:FirstSelected(), oListBox:Select()

oListBox:Right —> nCol *ACCESS*
oListBox:Right := nCol *ASSIGN*

<**nCol**> is a numeric value that indicates the rightmost screen column where the list box is displayed. The input and output value is either in coordinates or in pixels, depending on the current SET PIXEL setting.

Compatibility: Available also in CL53.

See also: oListBox:Left, oListBox:Top, oListBox:Bottom

oListBox:Sblock —> bBlock *ACCESS*
oListBox:Sblock := bBlock *ASSIGN*

<**bBlock**> is a code block or NIL. The code block callback, when present, is evaluated each time the user takes a selection in the ListBox object. Evaluated only when the ListBox (or ComboBox) has input focus.

The code block receives four arguments in this order: 1) the object self, 2) the ordinal position of the currently selected item in the array (i.e. :Buffer or :CurrItemNo), 3) the

currently selected item text (i.e. :TextValue :CurrentItem), and 4) the associated item value (:Value). If multiple selection is allowed and multiple items were selected, the 2nd, 3rd and 4th arguments are one-dimensional arrays in the size of :SelectedCount, containing the selected data. You should not use Inkey() nor other input commands or functions in the callback UDF.

Compatibility: Available also in CL53, but Clipper does not pass any arguments to the code block; it hence cannot use generalized but object specific code blocks which must extract the required values from the known object by itself.

See also: oListBox:HasFocus, oListBox:Buffer, oListBox:Fblock, oListBox:Text, oListBox:Value

Example: see oListBox:FillUsing()

oListBox:Scroll(nType) —> self

Scrolls the contents of a list box up or down.

<**nType**> Numeric value indicating the manner in which the scroll operation is carried out. The HTSCROLL* constants are available in button.fh header file.

Value	Constant	Performs
-3074	HTSCROLLUNITDEC	Scroll down one line
-3075	HTSCROLLUNITINC	Scroll up one line
-3076	HTSCROLLBLOCKDEC	Scroll down one window
-3077	HTSCROLLBLOCKINC	Scroll up one window.

Compatibility: Available also in CL53

oListBox:Select(nPos) —> self

Change the selected item in a list. On success, the number of selected items is set to 1. The selection state is typically changed by the user when one of the cursor keys is pressed or the mouse's left button is pressed when its cursor is within the ListBox object's screen region. This method allows programmable set or change the "selected" mode. If necessary, oListBox:Select() will call its :Display() or :Scroll() method to ensure that the newly selected item is visible.

<**nPos**> is a numeric value that indicates the position in the list of the item to select.

Compatibility: Available also in CL53

See also: oListBox:NextItem(), oListBox:FirstSelected(), oListBox:NextSelected(), oListBox:SelectItem()

oListBox:SelectBySingleClick –> lEnabled *ACCESS*
oListBox:SelectBySingleClick := lEnable *ASSIGN*

Allow selection by single click on left mouse button, i.e. same as mouse double click and handle it equivalent to Enter/Return. Considered/handled only in the keyboard handler in GUI mode.

oListBox:SelectBySpace –> lEnabled *ACCESS*
oListBox:SelectBySpace := lEnable *ASSIGN*

Allow current selection by space key, i.e. handle space equivalent to Enter/Return. If enabled (the default), searching for an item starting with space is disabled during the input. Considered/handled only in the keyboard handler.

oListBox:SelectedCount –> nNum *ACCESS*

<**nNum**> is a numeric value representing the total number of items that are currently selected in a multiple selection list box. This property is intended for use with a multiple selection list box.

Compatibility: Available also in VO.

See also: oListBox:FirstSelected(), oListBox:NextSelected(), oListBox:Select(), oListBox:SelectItem()

oListBox:SelectedFile –> cTxt *ACCESS*

<**cTxt**> is a string representing the selected file in a list box previously filled by the :ListFiles() method. In generally, the return value is equivalent to oListBox:CurrentItem and is supported for compatibility purposes only.

Compatibility: Available also in VO.

See also: oListBox:CurrentItem, oListBox:ListFiles(), oListBox:FirstSelected(), oListBox:NextSelected(), oListBox:Select(), oListBox:SelectItem()

oListBox:SelectItem(nPos) –> lOk

Change the selected item in a list and on success, reset :SelectedCount to 1. This method is fully equivalent to oListBox:Select() except the return value.

<**nPos**> is a numeric value that indicates the position in the list of the item to select.

Compatibility: Available also in VO

See also: oListBox:Select(), oListBox:FirstSelected(), oListBox:NextSelected(), oListBox:SelectedCount

oListBox:SetData(nPos, [exp], [lSelect], [lBitmap]) —> lOk

Change the property of an available item.

<**nPos**> The position in the list box at which to insert the new item; valid values are 1 to :ItemCount or 0 for the currently selected item.

<**exp**> Any value associated with the specified item, which enables to associate pertinent data with the text displayed in the list, default is NIL

<**lSelect**> optional logical value specifying if the item is selectable (TRUE, the default) or not (FALSE). This can be re-defined by oListBox:DeselectItem()

<**nBitmap**> optional logical value specifying that the <cText> is a name of a bitmap which should be displayed instead of the text (TRUE), or if <cText> is a usual text value to be displayed as such (FALSE, the default).

<**lOk**> TRUE if the item was changed, FALSE otherwise.

Compatibility: Available also in CL53 which supports two mandatory parameters and returns SELF.

See also: oListBox:SetText, oListBox:SetItem(), oListBox:AddItem(), oListBox:FillUsing(), oListBox:InsItem(), oListBox:GetData(), oListBox:GetText(), oListBox:GetItem()

oListBox:SetFocus() —> self

Set input focus to a ListBox object. Upon receiving this message, the ListBox object redisplays itself with the :HotBox frame and, if present, evaluates the code block specified by :Fblock. This message is meaningful only when the ListBox object does not have input focus. In GUI, the object receives focus also every times the user clicks (or activates) the widget.

Compatibility: Available also in CL53.

See also: oListBox:HasFocus, oListBox:KillFocus(), oListBox:Fblock

oListBox:SetItem(nPos, aData) —> lOk

Replaces the item property, i.e. the displayed text, associated data and additional flags providing these in one-dimensional array.

<**nPos**> numeric value that indicates the position within the list of the item whose data is being retrieved.0 (zero) specifies the currently selected item, 1 to :ItemCount is the requested item number.

<**aData**> is one-dimensional array with one to four elements containing the item properties: Element [1] is the displayed text, [2] the associated data of any type,

[3] a logical value specifying whether the item is selectable, [4] a logical value indicating whether the 1st element is a file name of a bitmap or a usual text.

<**lOk**> TRUE if the item was changed, FALSE otherwise.

Compatibility: Available also in CL53, which supports only the first two array elements.

See also: oListBox:AddItem(), oListBox:SetText(), oListBox:SetData(), oListBox:GetItem()

oListBox:SetText(nPos, cTxt) –> lOk

Change/replace the displayed text of an available item.

<**nPos**> The position in the list box at which to insert the new item; valid values are 1 to :ItemCount or 0 for the currently selected item.

<**cTxt**> Character string of the item to be changed and displayed in the list

<**lOk**> TRUE if the item was changed, FALSE otherwise.

Compatibility: Available also in CL53 which returns SELF.

See also: oListBox:SetData(), oListBox:SetItem(), oListBox:AddItem(), oListBox:FillUsing(), oListBox:InsItem(), oListBox:GetData(), oListBox:GetText()

oListBox:SetTop(nPos) –> self

Move a specified item to the top of the list box. This method is equivalent to oListBox:TopItem assign.

<**nPos**> The position in the list box which should be displayed at the top; valid values are 1 to :ItemCount or 0 for the currently selected item.

Compatibility: Available also in VO, which returns NIL

See also: oListBox:TopItem

oListBox:Show([naComboOpen], [naComboClose]) –> nSelItem

Show the list box and its caption on the screen and process keyboard/ mouse input. This method set input focus :SetFocus(), calls :Display(), enter and process the default or user's event/keyboard handler specified in :InputBlock, then clears the input focus by :KillFocus()

Alternatively, your application may call :Display() and invoke/process your input handler directly, w/o :Show() or :Exec(), same as Clipper do.

<**naComboOpen**> is optional numeric value or an array of numeric values specifying the key(s) used to open combo box (drop-down). The default setting is `{K_TAB, asc('#')}`

<**naComboClose**> is optional numeric value or an array of numeric values specifying the key(s) used to close combo box (drop-down). The default setting is `{K_SH_TAB, asc('^')}`

Both parameters are passed to the input handler (see :InputBlock) and are considered only for ComboBox, i.e. when the <lCombo> parameter in ListBox() instantiation is .T., or when the object was instantiated via ComboBox{} - in both cases the :DropDown instance returns .T.

The method returns the selected item# (1 to :ItemCount) or 0 when the user selection was aborted. The return value is passed from the default (or by :InputBlock user-provided) input handler.

Compatibility: Available also in VO (w/o parameters) which returns NIL
See also: oListBox:Display(), oListBox:Exec(), oListBox:InputBlock

oListBox:TextValue –> cText *ACCESS*
oListBox:TextValue := cText *ASSIGN*

<**cText**> is a string representing the displayed list box or combo box item selected. This property is equivalent to oListBox:CurrentItem access/assign.

Compatibility: Available also in VO
See also: oListBox:CurrentItem, oListBox:Value, oListBox:GetText(), oListBox:GetItem()

oListBox:ToolTip –> cText *ACCESS*
oListBox:ToolTip := cText *ASSIGN*

<**cText**> is a string representing the displayed tool tip, i.e. a short info message which pop up's when the mouse is over the list box.

Compatibility: Available also in FS5 only, apply for GUI, ignored otherwise
See also: oListBox:Message

oListBox:Top –> nRow *ACCESS*
oListBox:Top := nRow *ASSIGN*

<**nRow**> is a numeric value that indicates the topmost screen row where the list box is displayed. The input and output value is either in coordinates or in pixels, depending on the current SET PIXEL setting.

Compatibility: Available also in CL53
See also: oListBox:Bottom, oListBox:Left, oListBox:Right

oListBox:TopItem —> nPos *ACCESS*
oListBox:TopItem := nPos *ASSIGN*

<**nPos**> is a numeric value that indicates the position in the list box of the first visible item. Valid values are 1 to :ItemCount or 0 for the currently selected item. This property is equivalent to oListBox:SetTop(nPos)

Compatibility: Available also in CL53, which does not support the 0 value

See also: oListBox:SetTop()

oListBox:TypeOut —> lStat *ACCESS*

<**lStat**> A logical value that indicates whether the list contains any items. A TRUE value indicates that the list contains selectable items; otherwise, FALSE indicates that the list is empty. This property is equivalent to oListBox:ItemCount != 0.

Compatibility: Available also in CL53

See also: oListBox:ItemCount

oListBox:Value —> exp *ACCESS*
oListBox:Value := exp *ASSIGN*

<**exp**> Any value associated with the specified item, which enables to associate pertinent data with the text displayed in the list, default is NIL.

Compatibility: Available also in VO

See also: oListBox:GetData(), oListBox:GetItem(), oListBox:SetData(), oListBox:TextValue

oListBox:ValueChanged —> lStat *ACCESS*
oListBox:ValueChanged := lStat *ASSIGN*

<**lStat**> A logical value representing the status of oListBox:Value. It reports whether it has been changed from the previously selected item during the selection process. TRUE indicates that it has been changed from the prior choice, while FALSE indicates it has been not changed from the prior choice. The :Value may be changed by clicking on a different item, or via the oListBox:TextValue or oListBox:Value assigns.

Compatibility: Available also in VO

See also: oListBox:Value, oListBox:GetData(), oListBox:GetData(), oListBox:GetItem(), oListBox:TextValue

oListBox:Vscroll —> oScroll ***ACCESS***
oListBox:Vscroll := oScroll ***ASSIGN***

<**oScroll**> In CL53, contains ScrollBar object whose orientation must be vertical. Unsupported in FlagShip and hence the default return value is NIL.

Compatibility: Available also in CL53

MenuItem Class

The MenuItem Class is a property holder for TopBar and PopUp class.

In FlagShip, the main menu (TopBar class) with sub-menus (PopUp class) is created automatically for GUI mode at start-up of the application, called from initio.prg. The source of the main menu is available in <FlagShip_dir>/system/initiomenu.prg. You may modify the default menu any time later, see example in <FlagShip_dir>/examples/menu.prg

As with other GUI classes in FlagShip, the general MenuItem class is internally inherited by three different sub-classes: _gMenuItem for GUI based application, _tMenuItem for terminal/ text based mode, and _bMenuItem for basic i/o mode, all defined in the menuclass.fh header file. The proper class, corresponding to the used i/o mode, is set either at compile time with the compiler switch `"-io=g|t|b"`, or latest at run-time depending on the currently used environment. Note: in the basic i/o mode, only a rough MenuItem functionality is simulated by the sequential in/output.

MenuItem Class Index

Class MenuItem

Inherits from:	- (none)
Inherited by:	- (none)
Class prototype:	menuclass.fh
Defines:	button.fh, inkey.fh

Accelerator()	METHOD	accelerator key or text
Caption	ACC/ASS	item text or separator
Checked	ACC/ASS	check mark to the left?
ClassName()	METHOD	"MENUITEM" for Clipper compatibility
Column()	METHOD	set/get column
Data	ACC/ASS	CodeBlock or oPopUpMenu object or NIL
Enabled	ACC/ASS	can item be selected?
Font	ACC/ASS	the item's font
Id	ACC/ASS	unique identifier or 0
IsCodeblock()	METHOD	is MenuItem:Data a CodeBlock?
IsPopUp()	METHOD	is MenuItem:Data popup object?
IsSeparator()	METHOD	is MenuItem a separator?
Message	ACC/ASS	status bar message or ""
Row()	METHOD	set/get row
Select()	METHOD	FS5
Shortcut	ACC/ASS	accelerator in CL53 terminology
ShortKey	ACC/ASS	Inkey() lower value for quick access, def=0
ShortPos	ACCESS	index of Shortkey in Caption (def=0)
Style	ACC/ASS	delimiter, ignored by GUI
_setCurrent()	METHOD	for internal use

MenuItem Class Instantiation

oMenuItem := MenuItem (cCapt, expData, [nShort], [cMessage], [nID]) *[1]*
oMenuItem := MenuItem (cCapt, cnItemName) *[2]*
oMenuItem := MenuItem {cCapt, expData, [nShort], [cMessage], [nID]} *[3]*
oMenuItem := [_g|_t|_b]MenuItem {cCapt, expData, [nShort],[cMsg],[nID]} *[4]*

Any of the above syntax instantiate new MenuItem object. Syntax [1] is compatible to Clipper, syntax [2] is supported for compatibility purposes to FoxPro, syntax [3] is available in VO and FS5, syntax [4] in FS5 only. Arguments:

<**cCapt**> is a character string that contains either a text string (caption) that concisely describes the menu option or a menu separator specifier. Modifiable via oMenuItem:Caption property.

<**expData**> is a value that contains either a code block or a PopUpMenu object. This argument is ignored when <cCapt> contains a menu separator specifier. Modifiable via oMenuItem:Data property.

<**nShort**> is an optional numeric inkey value that indicates the shortcut key combination that selects and launches the menu selection. The default is 0. Modifiable via oMenuItem:Shortcut Constant values for various key combinations are defined in inkey.fh.

<**cMessage**> is an optional character string that indicates the text to display on the status bar when the menu item is selected. The default is an empty string. Modifiable via oMenuItem:Message

<**nID**> is an optional numeric value that uniquely identifies the menu item. The default is 0. Modifiable via oMenuItem:Id

<**cnItemName**> is a string specifying the menu name.

<**oMenuItem**> is the returned MenuItem object when all of the required arguments are present, or NIL on failure.

In FlagShip, the main menu (TopBar class) for GUI mode is created automatically at start-up of the application, called from initio.prg. The source is available in <FlagShip_dir>/system/initio.prg and initiomenu.prg. You may modify the default menu any time later, see example in <FlagShip_dir>/examples/menu.prg

If there is not special font specified via oMenuItem:Font or oTopBar:Font, the default window manager font is used.

Example: see <*FlagShip_dir>/system/initiomenu.prg* and <*FlagShip_dir>/examples/topmenu.prg*

Compatibility: Available also in CL53 and VO. See also: PopUp and TopBar classes

MenuItem Class Properties

oMenuItem:Accelerator([inkeyVal], [accText], [iMode]) —> iKey|cText

An accelerator is a keystroke sequence that is associated with a particular menu command. The accelerator is used to execute the menu command without requiring the application user to first display the menu and then choose the command. For example, if the File -> New command had an accelerator of Ctrl+N, you could simply press this key combination to open a new document, rather than having to choose the File menu and then the New command. Each window can be given its own accelerator. An accelerator generates events as though its associated menu command was actually selected. Note that an accelerators menu command does not even have to be visible on any menu - thus an accelerator can be seen as a direct keystroke sequence for generating a command event.

<**inkeyVal**> assign new inkey() value as accelerator key, 0 = disable, NIL = return current value

<**accText**> assign new accelerator text, "" = disable, NIL = return current value

<**iMode**> requested return mode mode : 1 = return current inkeyVal (default), 2= return current accText

oMenuItem:Caption —> cCapt ***ACCESS***
oMenuItem:Caption := cCapt ***ASSIGN***

Contains either a text string that concisely describes the menu option or a menu separator specifier. oMenuItem:caption is the text that appears in the actual menu.

A menu separator is a horizontal line in a pop-up menu that separates menu items into logical groups. Use the constant MENU_SEPARATOR in button.fh to assign the menu separator specifier to oMenuItem:caption.

When present, the & character specifies that the character immediately following it in the caption is the menu item's accelerator key. The accelerator key provides a quick and convenient mechanism for the user to select a menu item when the menu that it is contained within has input focus. When the menu is a member of a TopBar object, the user selects the menu item by pressing the Alt key in combination with the accelerator key. When the menu is a member of a PopUp object, the user selects the menu item by simply pressing the accelerator key. The accelerator key is not case sensitive.

oMenuItem:Cargo <—> anyValue ***EXPORT***

User definable value of any content. Not used by the MenuItem class self. The default is NIL.

oMenuItem:Checked –> lCheck *ACCESS*
oMenuItem:Checked := lCheck *ASSIGN*

Contains a logical value that indicates whether a check mark appears to the left of the menu item's caption. A value of true (.T.) indicates that a check mark should show; otherwise, a value of false (.F.) indicates that it should not.

oMenuItem:ClassName() –> "MENUITEM"

provided for Clipper compatibility purposes. Return fix "MENUITEM".

oMenuItem:Column([nCol], [lPixel]) –> nCol

Set/get column of the associated PopUp class.

<**nCol**> is a numeric value either in coordinates or in pixel. If NIL, only the current setting is returned.

<**lPixel**> If specified TRUE, the <nCol> input/output column coordinate is assumed in pixel. If FALSE, <nCol> is in row/col coordinates. If this parameter is not specified (i.e. NIL), the kind of passed and returned coordinates is determined from the current SET PIXEL setting.

oMenuItem:Data –> obData *ACCESS*
oMenuItem:Data := obData *ASSIGN*

<**obData**> contains either a code block or a PopUp object or NIL. When the menu item is selected, its code block, if present, is evaluated passing two parameters to the code block: the current MenuItem object, and menu-ID. If the codeblock returns .F., the TopBar or PopUp selection remains active, otherwise the current selection is terminated. If <obData> is PopUp object, the PopUp menu is opened on selection.

oMenuItem:Enabled –> lStatus *ACCESS*
oMenuItem:Enabled := lStatus *ASSIGN*

<**lStatus**> is a logical value that indicates whether the menu item can be selected or not. If true (.T.), it permit user access; if false (.F.) the user access is denied. When disabled, the item will be shown in its disabled color.

oMenuItem:Font –> oFont *ACCESS*
oMenuItem:Font := oFont *ASSIGN*

<**oFont**> is a Font object or NIL. Applicable for GUI mode only. If not specified, either the oTopBar:Font or the default window manager font is used. If specified, this item font has preference over the default oTopBar:Font

oMenuItem:Id —> nIdNum *ACCESS*
oMenuItem:Id := nIdNum *ASSIGN*

<**nIdNum**> is an optional numeric value that uniquely identifies the menu item. The default is 0 which sets an internal ID number automatically.

oMenuItem:IsCodeblock() —> lStatus

Returns true (.T.) when the MenuItem object contain code block, and false otherwise. This property is provided for your convenience and is equivalent to valtype(oMenuItem:Data) == "B"

oMenuItem:IsPopUp() —> lStatus

Returns true (.T.) when the MenuItem object contain PopUp object, and false otherwise. This property is provided for your convenience and is equivalent to valtype(oMenuItem:Data) == "O"

oMenuItem:IsSeparator() —> lStatus

Returns true (.T.) when the MenuItem object contain separator, and false otherwise. This property is provided for your convenience and is equivalent to oMenuItem:Data == NIL .or. oMenuItem:Data == MENU_CAPTION_SEPARATOR

oMenuItem:Message —> cbMsg *ACCESS*
oMenuItem:Message := cbNsg *ASSIGN*

<**cMsg**> is an optional string or code block evaluated to string that describes the menu item. The text appears on the screens status bar line if such (always in GUI mode), or in the SET MESSAGE TO line otherwise.

oMenuItem:Row([nRow], [lPixel]) —> nRow

Set/get row of the associated PopUp class.

<**nRow**> is a numeric value either in coordinates or in pixel. If NIL, only the current setting is returned.

<**lPixel**> If specified TRUE, the <nRow> input/output column coordinate is assumed in pixel. If FALSE, <nCol> is in row/col coordinates. If this parameter is not specified (i.e. NIL), the kind of passed and returned coordinates is determined from the current SET PIXEL setting.

oMenuItem:Select() —> self

Select the MenuItem, display it including all backward hierarchy

oMenuItem:Shortcut –> nInkeyVal *ACCESS*
oMenuItem:Shortcut := nInkeyVal *ASSIGN*

"Shortcut" is a Clipper terminology for an accelerator. It is fully equivalent to
ACCESS: `nInkeyVal := oMenuItem:Accelerator(NIL, NIL, 1)`
ASSIGN : `oMenuItem:Accelerator(nInkeyVal, NIL, 1)`

oMenuItem:ShortKey –> nInkeyVal *ACCESS*
oMenuItem:ShortKey := nInkeyVal *ASSIGN*

The ShortKey is defined using the & character in oMenuItem:Caption, set during the object instantiation or at oMenuItem:caption assign. As opposite to accelerator, the popup menu needs to be open for the ShortKey to be active.

<**nInkeyVal**> is the Inkey() equivalence of this character

oMenuItem:ShortPos –> nPos *ACCESS*

Returns an index of Shortkey in Caption, If no & character in the oMenuItem:Caption was specified, 0 is returned.

oMenuItem:Style –> cStyle *ACCESS*
oMenuItem:Style := cStyle *ASSIGN*

<**cStyle**> is a character string that indicates the delimiter characters that are used by the PopUp:Display() method. The string must contain two characters. The first is the character associated with the oMenuItem:checked property, its default value is the square root character. The second is the sub-menu indicator, its default is the right arrow character.

This property is considered in Terminal i/o mode only and ignored otherwise.

PopUp Class

Place and display items in pop-up menu.

In FlagShip, the main menu (TopBar class) with sub-menus (PopUp class) is created automatically for GUI mode at start-up of the application, called from initio.prg. The source of the main menu is available in <FlagShip_dir>/system/initiomenu.prg. You may modify the default menu any time later, see example in <FlagShip_dir>/examples/menu.prg

As with other GUI classes in FlagShip, the general PopUp class is internally inherited by three different sub-classes: _gPopUp for GUI based application, _tPopUp for terminal/text based mode, and _bPopUp for basic i/o mode, all defined in the menuclass.fh header file. The proper class, corresponding to the used i/o mode, is set either at compile time with the compiler switch "-io=g|t|b", or latest at run-time depending on the currently used environment.

Note: in GUI mode, the PopUp is handled only in the full menu bar context, i.e. as a sub-class (child) of TopBar or of other PopUp class. In basic i/o mode, only a rough PopUp functionality is simulated by the sequential in/ output.

PopUp Class Index

Class PopUp

Inherits from:	- (none)
Inherited by:	- (none)
Class prototype:	menuclass.fh
Defines:	button.fh, box.fh

AddItem()	METHOD	appending new item
Border	ACC/ASS	ignored in GUI
Bottom	ACC/ASS	bottommost screen row (pixel/row)
ClassName()	METHOD	"POPUPMENU" for Clipper compatibility
Close()	METHOD	deactivate pop-up menu
ColorSpec	ACC/ASS	term only, ignored in GUI
Current	ACC/ASS	selected item#
DelItem()	METHOD	remove an item
Display()	METHOD	show pop-up menu
GetAccel()	METHOD	item# corresp.to given accel
GetFirst()	METHOD	first selectable item
GetItem()	METHOD	returns the MenuItem object
GetLast()	METHOD	last selectable item
GetNext()	METHOD	next selectable item
GetPrev()	METHOD	previous selectable item
GetShortct()	METHOD	shortcut keystroke
HitTest()	METHOD	relationship of mouse and popup

InputBlock	ACC/ASS	user-supplied input
InsItem()	METHOD	insert new item at spec.position
IsOpen()	METHOD	popup open?
ItemCount	ACCESS	total number of items
ItemPos()	METHOD	find menu item
Left	ACC/ASS	leftmost screen column or NIL
MenuStruct()	METHOD	menu structure
Open()	METHOD	open popup
ResetAllItems()	METHOD	reset all MenuItem objects
Right	ACC/ASS	rightmost screen column or NIL
Top	ACC/ASS	topmost screen row or NIL
Select()	METHOD	select specif.item
SetItem()	METHOD	replace MenuItem object
SetAllItems()	METHOD	set tempor block to all MenuItem objects
SubmenuMark	ACC/ASS	mark for submenus (" ...")
Width	ACCESS	width required... (pixel/row)

PopUp Class Instantiation

oPopUp := PopUp ([nTop], [nLeft], [nBottom], [nRight], [lInPixel]) *[1]*
oPopUp := PopUp {[nTop], [nLeft], [nBottom], [nRight], [lInPixel]} *[2]*
oPopUp := [_g|_t|_b]PopUp {[nTop],[nLeft],[nBott],[nRight],[lInPix]} *[3]*

Any of the above syntax instantiate new PopUp object. Syntax [1] is compatible to Clipper, syntax [2] is available in VO and FS5, syntax [3] in FS5 only.

Note: in GUI mode, the PopUp is handled only in the full menu bar context, i.e. as a sub-class (child) of TopBar or of another PopUp class.

Arguments (all optional):

<**nTop**> is a numeric value that indicates the top screen row of the pop-up menu. If omitted, oPopUp:top is set to an appropriate value relative to <nBottom> that allows as many items as possible to show.

When the pop-up menu is a child of another menu, its top variable will be automatically set by the parent menu regardless of whether <nTop> is omitted.

<**nLeft**> is a numeric value that indicates the left screen column of the pop-up menu. If omitted, oPopUp:left is set to an appropriate value relative to <nRight> that allows as many menu columns as possible to show.

When the pop-up menu is a child of another menu, its left variable will be automatically set by the parent menu regardless of whether <nLeft> is omitted.

<**nBottom**> is a numeric value that indicates the bottom screen row of the pop-up menu. If omitted, oPopUp:bottom is set to an appropriate value relative to <nTop> that allows as many items as possible to show. If <nTop> is also omitted, oPopUp:bottom is set to center the menu vertically on the screen. The default value is determined the first time the pop-up menu is displayed.

When the pop-up menu is a child of another menu, its bottom variable will be automatically set by the parent menu regardless of whether <nBottom> is omitted.

<**nRight**> is a numeric value that indicates the right screen column of the pop-up menu. If omitted, oPopUp:right is set to an appropriate value relative to <nLeft> that allows as many menu columns as possible to show. If <nLeft> is also omitted, oPopUp:right is set to center the menu horizontally on the screen. The default value is determined the first time the pop-up menu is displayed.

When the pop-up menu is a child of another menu, its right variable will be automatically set by the parent menu regardless of whether <nRight> is omitted.

<**lPixel**> If specified TRUE, the input coordinates are assumed in pixel. If FALSE, the input are row/col coordinates. If this parameter is not specified (i.e. NIL), the kind of passed coordinates is determined from the current SET PIXEL setting.

<**oPopUp**> is a PopUp object when all of the required arguments are present, or NIL otherwise.

Example: see <*FlagShip_dir*>*/system/initiomenu.prg* and <*FlagShip_dir*>*/examples/topmenu.prg*

Compatibility: Available also in CL53.

See also: TopBar and MenuItem classes

PopUp Class Properties

oPopUp:AddItem(oMenuItem) —> self

Add a new item in the PopUp object. Arguments:

<**oMenuItem**> is a MenuItem object which is appended at the end of the PopUp items list.

See also: oPopUp:InsItem()

oPopUp:Border —> cBorder *ACCESS*
oPopUp:Border := cBorder *ASSIGN*

<**cBorder**> is an optional string that is used when drawing a border around the pop-up menu. Its default value is predefined in the global array element `_aGlobSetting[GSET_T_C_MENUBORDER]`, see initio.prg. It is usually `B_SINGLE + SEPARATOR_SINGLE`.

The string must contain either zero or exactly eleven characters. The first eight characters represent the border of the pop-up menu and the final three characters represent the left, middle, and right characters for the menu item separators. The eight characters which represent the pop-up menu border begin at the upper-left corner and rotate clockwise as follows: upper-left corner, top, upper-right corner, right, bottom, bottom-left corner, and left. This property apply for Terminal i/o mode only and is ignored in GUI.

oPopUp:Bottom —> nRow *ACCESS*
oPopUp:Bottom := nRow *ASSIGN*

<**nRow**> is a numeric value that indicates the bottommost screen row where the pop-up menu is displayed. If not specified when the PopUp object is instantiated, oPopUp:bottom contains NIL until the first time it is displayed. This property applies for Terminal i/o mode only and is ignored in GUI.

oPopUp:Cargo <—> anyValue *EXPORT*

User definable value of any content. Not used by the PopUp class self. The default is NIL.

oPopUp:ClassName() —> "POPUPMENU"

provided for Clipper compatibility purposes. Return fix "POPUPMENU".

oPopUp:Close([lChild]) —> self

<**lChild**> is a logical value that indicates whether oPopUp:close() should deactivate the pop-up menu in its selected item, which in turn deactivates the pop-up menu in its selected item and so on. This is useful for nested menus where multiple levels of choices are presented. A value of true (.T.) indicates that child pop-up menu items should be closed. A value of false (.F.) indicates that child pop-up menu items should not be closed. The default value is true.

oPopUp:Close() is used for deactivating of pop-up menu. When called, it performs three operations. First, if the value of <lChild> is not false, close() determines if its selected menu item contains a PopUp object. If so, it calls its selected menu item's close() method. Second, close() restores the previous contents of the region of the screen that it occupies. Third, close() sets its selected item to 0. When the pop-up menu is not open, no action is taken.

oPopUp:ColorSpec —> cColor *ACCESS*
oPopUp:ColorSpec := cColor *ASSIGN*

<**cColor**> is a character string that indicates the color attributes that are used by the pop-up menu's display() method. The string can contain up to seven color pairs:

Position in colorSpec	Applies To	Default value from system color setting
1	Not selected popup menu items	Standard
2	Selected popup menu item	Enhanced
3	Accelerator key for unselected items	Background
4	Accelerator key for the selected items	Enhanced
5	Disabled popup menu items	Unselected
6	Popup menu's border	Border
7	Statusbar message	Standard

This property applies for Terminal i/o mode and is ignored otherwise.

oPopUp:Current —> nPos *ACCESS*
oPopUp:Current := nPos *ASSIGN*

<**nPos**> is a numeric value that indicates which item is selected.

oPopUp:DelItem(nPos) —> self

Remove an item from a pop-up menu. Argument:

<**nPos**> is a numeric value that indicates the position in the pop-up menu of the item to be deleted.

oPopUp:Display() —> self

Shows a pop-up menu including its items on the screen. It checks all previously specified oPopUp object properties, calculates missing values if required, and displays the widget on the screen. Display() is also called automatically when the parent menu item in TopBar or PopUp is selected.

oPopUp:GetAccel(nInkeyVal) —> nInkeyVal

<**nInkeyVal**> is a numeric value that indicates the inkey() value to be checked. Returns a numeric value that indicates the position in the pop-up menu of the first item whose accelerator key matches that which is specified by <nInkeyVal>. The accelerator key is defined using the & character in oMenuItem:Caption.

oPopUp:GetFirst() —> nPos

Determine the position of the first selectable item in a pop-up menu. Selectable means a menu item that is enabled and whose caption is not a menu separator.

<**nPos**> is a numeric value that indicates the position within the pop-up menu of the first selectable item. Returns 0 if the pop-up menu does not contain a selectable item.

oPopUp:GetFirst() does not change the currently selected menu item. In order to change the currently selected pop-up menu item, you must call the oPopUp:Select() method.

oPopUp:GetItem(nPos) —> oMenuItem

Return the specified item in a pop-up menu, regardless if the item is selectable or not.

<**nPos**> is a numeric value that indicates the position in the pop-up menu of the item that is being retrieved.

<**oMenuItem**> is a MenuItem object at the position in the pop-up menu specified by <nPos> or NIL when <nPos> is invalid.

oPopUp:GetItem() does not change the currently selected menu item. In order to change the currently selected pop-up menu item, you must call the oPopUp:Select() method.

oPopUp:GetLast() —> nPos

Determine the position of the last selectable item in a pop-up menu. Selectable means a menu item that is enabled and whose caption is not a menu separator.

<**nPos**> is a numeric value that indicates the position within the pop-up menu of the last selectable item. Returns 0 if the pop-up menu does not contain a selectable item.

oPopUp:GetLast() does not change the currently selected menu item. In order to change the currently selected pop-up menu item, you must call the oPopUp:Select() method.

oPopUp:GetNext() —> nPos

Determine the position of the next selectable item in a pop-up menu. Selectable means a menu item that is enabled and whose caption is not a menu separator.

<**nPos**> is a numeric value that indicates the position within the pop-up menu of the next selectable item. Returns 0 if the pop-up menu does not contain next selectable item.

oPopUp:GetNext() does not change the currently selected menu item. In order to change the currently selected pop-up menu item, you must call the oPopUp:Select() method.

oPopUp:GetPrev() —> nPos

Determine the position of the previous selectable item in a pop-up menu. Selectable means a menu item that is enabled and whose caption is not a menu separator.

<**nPos**> is a numeric value that indicates the position within the pop-up menu of the previous selectable item. Returns 0 if the pop-up menu does not contain previous selectable item.

oPopUp:GetPrev() does not change the currently selected menu item. In order to change the currently selected pop-up menu item, you must call the oPopUp:Select() method.

oPopUp:GetShortct(nInkeyVal) —> nPos

<**nInkeyVal**> is a numeric value that indicates the inkey() value to be checked.

<**nPos**> is a numeric value that indicates the position in the pop-up menu of the first item whose shortcut key matches that which is specified by <nInkeyVal>. The shortcut key is defined using the oMenuItem:shortcut property.

oPopUp:HitTest(p1, p2) —> nStatus

Provided for backward compatibility purposes to Clipper only. <nStatus> is always 0.

oPopUp:InputBlock –> oBlock *ACCESS*
oPopUp:InputBlock := oBlock *ASSIGN*

<**oBlock**> is user-supplied code block, i.e. keyboard handler which should be used instead of the build-in one. If <oBlock> is NIL, the default handler will be (re)used. The code block is called in oPopUp:Display() and receive two arguments: the oPopUp object, and the pressed key as an Inkey() value. The code block should then perform the required action and return either

0 exit the PopUp processing
< 0 enter the current item
> 0 select the PopUp item specified by the return value

See also <FlagShip_dir>/examples/menu2.prg

Apply for Terminal i/o mode, ignored otherwise.

Source: example of the handler is available in <FlagShip_dir>/system/ topbarhandler.prg

oPopUp:InsItem(nPos, oMenuItem) –> self

Add a new item in the PopUp object at specified position. Arguments:

<**nPos**> is a numeric value specifying the position where the menu item should be inserted. A value greater than oPopUp:ItemCount perform the same action as oPopUp:AddItem()

<**oMenuItem**> is a MenuItem object which is inserted at the specified position in the PopUp items list.

See also: oPopUp:AddItem()

oPopUp:IsOpen() –> lStatus

<**lStatus**> is true (.T.) when the PopUp is open, or false otherwise

oPopUp:ItemCount –> nCount *ACCESS*

<**nCount**> is the total number of items in PopUp

oPopUp:ItemPos(oMenuItem) –> nPos

<**oMenuItem**> is a MenuItem object which should be searched in the PopUp item list

<**nPos**> is the found position of specified MenuItem object in the PopUp item list or 0 if not found.

oPopUp:Left —> nCol *ACCESS*
oPopUp:Left := nCol *ASSIGN*

<**nCol**> is the leftmost screen column. Apply for Terminal i/o, ignored otherwise.

oPopUp:MenuStruct([nDepth], [aStruct]) —> aStruct

Creates a multi-dimensional array containing the current menu structure. Arguments (all optional):

<**nDepth**> is the required depth. If not specified, all childs are determined.

<**aStruct**> if the array was passed as argument, the returned structure is appended, otherwise a new array is created. Any element contain a sub-array with {nDepth, nRelPos, oMenuItem, oMenuItem:Caption}

Example:

```
aStruct := oMyPopup:MenuStruct()
aeval(aStruct, {|x| qout(space(x[1] *2), ltrim(x[2]), x[4] )} )
```

oPopUp:Open() —> self

equivalent to oPopUp:Display()

oPopUp:ResetAllItems() —> NIL

Replace all PopUp items MenuItem's data with the by oPopUp:SetData() stored values.

oPopUp:Right —> nCol *ACCESS*
oPopUp:Right := nCol *ASSIGN*

<**nCol**> is the rightmost screen column. If not specified when the PopUp object is instantiated, oPopUp:Right contains NIL until the first time it is displayed. Apply for Terminal i/o, ignored otherwise.

oPopUp:Top —> nRow *ACCESS*
oPopUp:Top := nRow *ASSIGN*

<**nRow**> is a numeric value that indicates the topmost screen row where the pop-up menu is displayed. If not specified when the PopUp object is instantiated, oPopUp:bottom contains NIL until the first time it is displayed. This property applies for Terminal i/o mode only and is ignored in GUI.

oPopUp:Select(nPos) —> self

Set/select the specified item number. The method is typically called when one of the arrow keys is pressed.

<**nPos**> is a numeric value that indicates the position in the pop-up menu of the item to be selected.

Apply for Terminal i/o mode, ignored in GUI mode, where the action is performed automatically.

oPopUp:SetItem(nPos, oMenuItem) —> self

Replace specified MenuItem object in the PopUp list. Arguments:

<**nPos**> is the item position in range 1..oPopUp:ItemCount

<**oMenuItem**> is MenuItem object replacing the current pop up item.

oPopUp:SetAllItems(bBlock) —> NIL

Replace all PopUp items by the same code block <bBlock>, i.e. perform the same action on all menu items. The replacement has no affect on menu items that are separators or PopUp objects. You may restore the previous status any time later by oPopUp:ResetAllItems()

oPopUp:SubmenuMark —> cMark ***ACCESS***
oPopUp:SubmenuMark := cMark ***ASSIGN***

<**cMark**> is a character or string which should be displayed as a submenu indicator instead of the 2nd character of oMenuItem:Style.

This property is considered in Terminal i/o mode only and ignored otherwise.

oPopUp:Width —> nWidth ***ACCESS***

<**nWidth**> is the current width of the pop up window including the frame box. Available after the oPopUp:Display() was invoked. Apply for Terminal i/o mode only, ignored otherwise.

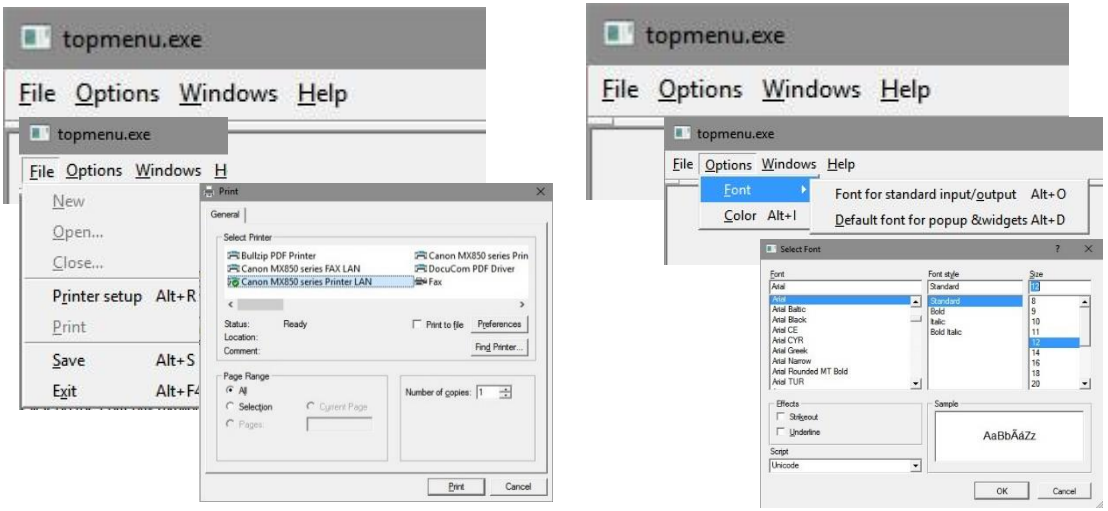
TopBar Class

Place and display items in TopBar menu. The TopBar is the main menu bar object in GUI mode.

In FlagShip, the main menu (TopBar class) with sub-menus (PopUp class) is created automatically for GUI mode at start-up of the application, called from initio.prg. A TopBar object is assigned there to constant variable "oTopBar". The source of the main menu is available in *<FlagShip_dir>/system/initiomenu.prg*. You may modify (or disable) the default menu any time later, see example in *<FlagShip_dir>/examples/topmenu.prg*

As with other GUI classes in FlagShip, the general TopBar class is internally inherited by three different sub-classes: _gTopBar for GUI based application, _tTopBar for terminal/text based mode, and _bTopBar for basic i/o mode, all defined in the menuclass.fh header file. The proper class, corresponding to the used i/o mode, is set either at compile time with the compiler switch `"-io=g|t|b"`, or latest at run-time depending on the currently used environment.

Note: in GUI mode, the TopBar is handled only in the full menu bar context, which handles automatically also all PopUp child objects, see **example** in *<FlagShip_dir>/examples/topmenu.prg*. In basic i/o mode, only a rough functionality is simulated by the sequential in/output.



As opposite to common procedural sequence, the GUI TopBar and it sub-items are **event oriented**, i.e. click on the main menu (or access it by Alt-? key) pop-up the corresponding item and executes the code block (and the stored procedure) independent on where your procedural application sequence is. TopBar menu is commonly used to execute small task and then continues in the interrupted application part. To drive the whole application via the main (TopBar) menu is possible, but slightly complicated compared to sequential/procedural @..PROMPT / MENU TO.

TopBar Class Index

Class TopBar
Inherits from: - (none)
Inherited by: - (none)
Class prototype: menuclass.fh
Defines: button.fh, box.fh

AddItem()	METHOD	append new item
ClassName()	METHOD	"TOPBARMENU" for Clipper compatibility
Clear()	METHOD	clear/delete all menu items
ColorSpec	ACC/ASS	color specification
Current	ACC/ASS	set/return selected item#
Delimiters	ACC/ASS	set/return delimiters
DelItem()	METHOD	remove specific item#
Display()	METHOD	show top bar menu & items
Exec()	METHOD	exec/process top bar menu & items
FindMenuProperty()	METHOD	find item by property
FindSelectedItem()	METHOD	find last selected item
Font	ACC/ASS	the font of whole menu structure
GetAccel()	METHOD	item# corresp.to given accel
GetFirst()	METHOD	first selectable item#
GetItem(p1)	METHOD	return oMenuItem at position#
GetLast()	METHOD	last selectable item#
GetNext()	METHOD	next selectable item#
GetPrev()	METHOD	previous selectable item#
HasFocus	ACCESS	Exec() in process ?
HitTest()	METHOD	relationship of mouse in TopBar
InputBlock	ACC/ASS	user-supplied input handler
InsItem()	METHOD	insert new item at position#
ItemCount	ACCESS	number of items in TopBar
ItemPos()	METHOD	find menu item
Left	ACC/ASS	leftmost column position
MenuStruct()	METHOD	menu structure
Right	ACC/ASS	rightmost column position
Row	ACC/ASS	row position
Select()	METHOD	change the selected item#
Separator	ACC/ASS	menu separator
SetItem()	METHOD	replace specific item

TopBar Class Instantiation

oTopBar := TopBar ([nRow], [nLeft], [nRight], [llnPixel]) *[1]*
oTopBar := TopBar {[nRow], [nLeft], [nRight], [llnPixel]} *[2]*
oTopBar := [_g|_t|_b]TopBar {[nRow],[nLeft],[nRight],[llnPix]} *[3]*

Any of the above syntax instantiate new TopBar object. Syntax [1] is compatible to Clipper, syntax [2] is available in VO and FS5, syntax [3] in FS5 only.

Note: in GUI mode, the TopBar is handled only in the full menu bar context, i.e. it process automatically also it sub-classes (childs) of the MenuItem and PopUp class. All the coordinates are ignored in GUI mode, since there is a fix position of the menu bar within the application menu. Arguments (all optional, considered in terminal i/o mode only):

<**nRow**> is a numeric value that indicates the screen row of the top bar menu. If omitted, 0 is the default. This value is modifiable by the oTopBar:Row property.

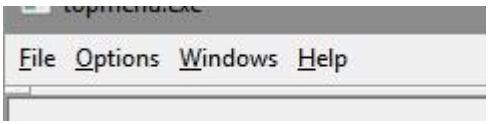
<**nLeft**> is a numeric value that indicates the left screen column of the top bar menu. The default is 0. Modifiable via the oTopBar:Left property.

<**nRight**> is a numeric value that indicates the right screen column of the top bar menu. Modifiable via the oTopBar:Right property.

<**lPixel**> If specified TRUE, the input coordinates are assumed in pixel. If FALSE, the input are row/col coordinates. If this parameter is not specified (i.e. NIL), the kind of passed coordinates is determined from the current SET PIXEL setting.

<**oTopBar**> is the instantiated TopBar object.

In FlagShip, the main menu bar (TopBar class) with sub-menus (PopUp class) is created automatically for GUI mode at start-up of the application, called from initio.prg and assigned to oTopBar global/public variable, see also *<FlagShip_dir>/system/ initiomenu.prg* and *<FlagShip_dir>/system/initio.prg*.



So instead of creating new TopBar object, you should use this m->oTopBar variable. The source of the main menu is available in *<FlagShip_dir>/system/initiomenu.prg*. You may modify or redefine the default menu any time later, see example in *<FlagShip_dir>/examples/ topmenu.prg*

Compatibility: Available also in CL53.

See also: PopUp and MenuItem classes, *<FlagShip_dir>/system/initiomenu.prg*

TopBar Class Properties

oTopBar:AddItem() —> self

Add a new item in the TopBar object. Arguments:

<**oMenuItem**> is a MenuItem object which is appended at the end of the TopBar items list.

See also: oTopBar:InsItem() and example <*FlagShip_dir*>*/examples/topmenu.prg*

oTopBar:Cargo <—> anyValue ***EXPORT***

User definable value of any content. Not used by the TopBar class self. The default is NIL.

oTopBar:ClassName() —> "TOPBARMENU"

provided for Clipper compatibility purposes. Return fix "TOPBARMENU".

oTopBar:Clear() —> self

Clears (deletes) all previously assigned TopBar items, without deleting the oTopBar object (constant) self. New TopBar items can be assigned thereafter, see also example <*FlagShip_dir*>*/examples/topmenu.prg*

After oTopBar:Clear() 1

oTopBar:ColorSpec —> cColor ***ACCESS***
oTopBar:ColorSpec := cColor ***ASSIGN***

<**cColor**> is a character string that indicates the color attributes that are used by the top bar menu's display() method. The string can contain up to seven color pairs:

Position in colorSpec	Applies To	Default value from system color setting
1	Not selected bar menu items	Standard
2	Selected top bar menu item	Enhanced
3	Accelerator key for unselected items	Background
4	Accelerator key for the selected items	Enhanced
5	Disabled top bar menu items	Unselected
6	Top bar menu's border	Border
7	The message	Standard

This property applies for Terminal i/o mode and is ignored otherwise.

oTopBar:Current —> nPos ***ACCESS***
oTopBar:Current := nPos ***ASSIGN***

<**nPos**> is a numeric value that indicates which item is selected.

oTopBar:Delimiters —> cDelim ***ACCESS***
oTopBar:Delimiters := cDelim ***ASSIGN***

<**cDelim**> is a string containing 0 or two characters specifying the left and right delimiter of the menu item, e.g. "[]". The default is an empty string "" specifying that no delimiters are used. Apply for Terminal i/o mode only, ignored otherwise.

oTopBar:DelItem(nPos) —> self

Remove an item from a top bar menu. Argument:

<**nPos**> is a numeric value that indicates the position in the top bar menu of the item to be deleted.

See also: oTopBar:Clear(), oTopBar:Append(), oTopBar:Insert()

oTopBar:Display() —> self

Shows a top bar menu on the screen. Display() checks all previously specified oPopUp object properties, calculates missing values if required, and displays the menu bar on the screen. To process menu bar entries automatically, use oTopBar:Exec()

oTopBar:Exec() —> nSelected

Process the top bar selection (and all childs) by using the build-in or user defined keyboard handler (specified by InputBlock).

<**nSelected**> is the selected menu ID number.

oTopBar:FindMenuProperty(bCompare) —> oMenuItem

Searches the whole menu structure (including all childs) for the requested menu item property.

<**bCompare**> is a code block specifying the search criteria. It receives one parameter, the oMenuItem object, and should return true (.T.) on match, or .F. otherwise.

<**oMenuItem**> is the found menu item object when the code block report .T., or NIL when the requested property was not found.

Example:

```
oMenuItem := m->oTopBar:FindMenuProperty( {|obj| obj:Id == 123 } )
oMenuItem := m->oTopBar:FindMenuProperty( ;
                  {|obj| upper(left(obj:Caption)) == "PRINT" } )
if valtype(oMenuItem) == "O"
   ? "MenuItem found ..."
endif
```

oTopBar:FindSelectedItem() —> oMenuItem

Searches the whole menu structure (including all childs) for the requested menu item property.

<**oMenuItem**> is the menu item currently selected, or NIL if none.

oTopBar:Font —> oFont *ACCESS*
oTopBar:Font := oFont *ASSIGN*

<**oFont**> is the used font object for the whole menu bar sequence (i.e. top bar including all childs). If not specified or is NIL, the default window manager font is used, except a menu item has own font assigned by oMenuItem:Font. Apply for GUI mode only, ignored otherwise.

Example:

```
m->oTopBar:Font := Font{"Courier",10}
m->oTopBar:Display()    // refresh topbar menu & submenus

if valtype(m->oTopBar:Font) == "O"
   ? "used TopBar font =", m->oTopBar:Font:FontFamily
else
   ? "default TopBar font =", m->oApplic:FontWindow:FontFamily
endif
```

oTopBar:GetAccel(nInkeyVal) —> nInkeyVal

<**nInkeyVal**> is a numeric value that indicates the inkey() value to be checked. Returns a numeric value that indicates the position in the top bar menu of the first item whose accelerator key matches that which is specified by <nInkeyVal>. The accelerator key is defined using the & character in oMenuItem:Caption.

oTopBar:GetFirst() —> nPos

Determine the position of the first selectable item in a top bar menu. Selectable means a menu item that is enabled and whose caption is not a menu separator.

<**nPos**> is a numeric value that indicates the position within the top bar menu of the first selectable item. Returns 0 if the top bar menu does not contain a selectable item.

oTopBar:GetFirst() does not change the currently selected menu item. In order to change the currently selected top bar menu item, you must call the oTopBar:Select() method.

oTopBar:GetItem(nPos|cPos) —> oMenuItem

Return the specified item in a top bar menu, regardless if the item is selectable or not.

<**nPos**> is a numeric value that indicates the position in the top bar menu of the item that is being retrieved.

<**cPos**> is the string specifying the menu name specified by the FoxPro compatible syntax during the MenuItem(cCapt, cMenuName) invocation.

<**oMenuItem**> is a MenuItem object at the position in the top bar menu specified by <nPos> or NIL when <nPos> is invalid.

oTopBar:GetItem() does not change the currently selected menu item. In order to change the currently selected top bar menu item, you must call the oTopBar:Select() method.

oTopBar:GetLast() —> nPos

Determine the position of the last selectable item in a top bar menu. Selectable means a menu item that is enabled and whose caption is not a menu separator.

<**nPos**> is a numeric value that indicates the position within the top bar menu of the last selectable item. Returns 0 if the top bar menu does not contain a selectable item.

oTopBar:GetLast() does not change the currently selected menu item. In order to change the currently selected top bar menu item, you must call the oTopBar:Select() method.

oTopBar:GetNext() —> nPos

Determine the position of the next selectable item in a top bar menu. Selectable means a menu item that is enabled and whose caption is not a menu separator.

<**nPos**> is a numeric value that indicates the position within the top bar menu of the next selectable item. Returns 0 if the top bar menu does not contain next selectable item.

oTopBar:GetNext() does not change the currently selected menu item. In order to change the currently selected top bar menu item, you must call the oTopBar:Select() method.

oTopBar:GetPrev() —> nPos

Determine the position of the previous selectable item in a top bar menu. Selectable means a menu item that is enabled and whose caption is not a menu separator.

<**nPos**> is a numeric value that indicates the position within the top bar menu of the previous selectable item. Returns 0 if the top bar menu does not contain previous selectable item.

oTopBar:GetPrev() does not change the currently selected menu item. In order to change the currently selected top bar menu item, you must call the oTopBar:Select() method.

oTopBar:HasFocus —> lStatus ***ACCESS***

<**lStatus**> is true (.T.) when the top bar has input focus and the Exec() is executed, or .F. otherwise.

oTopBar:HitTest(p1, p2) —> nStatus

Provided for backward compatibility purposes to Clipper only.

<**nStatus**> is always 0.

oTopBar:InputBlock —> oBlock ***ACCESS***
oTopBar:InputBlock := oBlock ***ASSIGN***

<**oBlock**> is user-supplied code block, i.e. keyboard handler which should be used instead of the build-in one. If <oBlock> is NIL, the default handler will be (re)used. The code block is called in oTopBar:Exec() and receive two arguments: the oTopBar object, and the pressed key as an Inkey() value. The code block should then perform the required action and return either

0 : exit the PopUp processing
<0 : enter the current item
>0 : select the PopUp item specified by the return value

See also <FlagShip_dir>/examples/menu2.prg
Apply for Terminal i/o mode, ignored otherwise.
Source: example of the handler is available in <FlagShip_dir>/system/topbarhandler.prg

oTopBar:InsItem(nPos, oMenuItem) —> self

Add a new item in the top bar object at specified position. Arguments:

<**nPos**> is a numeric value specifying the position where the menu item should be inserted. A value greater than oTopBar:ItemCount perform the same action as oTopBar:AddItem()

<**oMenuItem**> is a MenuItem object which is inserted at the specified position in the top bar items list.

See also: oTopBar:AddItem()

oTopBar:ItemCount —> nCount *ACCESS*

<**nCount**> is the total number of items in top bar

oTopBar:ItemPos(oMenuItem) —> nPos

<**oMenuItem**> is a MenuItem object which should be searched in the top bar item list

<**nPos**> is the found position of specified MenuItem object in the top bar item list or 0 if not found.

oTopBar:Left —> nCol *ACCESS*
oTopBar:Left := nCol *ASSIGN*

<**nCol**> is the leftmost screen column. Apply for Terminal i/o, ignored otherwise.

oTopBar:MenuStruct([nDepth], [aStruct]) —> aStruct

Creates a multi-dimensional array containing the current menu structure. Arguments (all optional):

<**nDepth**> is the required depth. If not specified, all childs are determined.

<**aStruct**> if the array was passed as argument, the returned structure is appended, otherwise a new array is created. Any element contain a sub-array with {nDepth, nRelPos, oMenuItem, oMenuItem:Caption}

Example:

```
aStruct := oTopBar:MenuStruct()
aeval(aStruct, {|x| qout(space(x[1] *2), ltrim(x[2]), x[4] )} )
```

oTopBar:Right —> nCol *ACCESS*
oTopBar:Right := nCol *ASSIGN*

<**nCol**> is the rightmost screen column. If not specified when the top bar object is instantiated, oTopBar:Right contains NIL until the first time it is displayed. Apply for Terminal i/o, ignored otherwise.

oTopBar:Row —> nRow *ACCESS*
oTopBar:Row := nRow *ASSIGN*

<**nRow**> is the screen row. If not specified, 0 is the default. Apply for Terminal i/o, ignored otherwise.

oTopBar:Select(nPos) —> self

Set/select the specified item number. The method is typically called when one of the arrow keys is pressed.

<**nPos**> is a numeric value that indicates the position in the top bar menu of the item to be selected.

Apply for Terminal i/o mode, ignored in GUI mode, where the action is performed automatically.

oTopBar:Row —> cSeparator *ACCESS*
oTopBar:Row := cSeparator *ASSIGN*

Set/return the TopBar separator character, e.g. "|". The default is empty string "". Apply for Terminal i/o mode, ignored otherwise.

oTopBar:SetItem(nPos, oMenuItem) —> self

Replace specified MenuItem object in the top bar list. Arguments:

<**nPos**> is the item position in range 1..oTopBar:ItemCount

<**oMenuItem**> is MenuItem object replacing the current top bar item.

Printer Class

The printer class handles the output to the standard or selected printer driver. In GUI, it provides also a dialog for printer setup, i.e. to select the required driver.

The Printer setup and Print is available in the default Menu->File. The Print option is disabled there as long as the printer spooler file is empty. See also LNG.5.1.6 and <FlagShip_dir>/system/initiomenu.prg for details.

The printer object is instantiated automatically in initio.prg and is stored in global constant named "oPrinter"

Class Printer

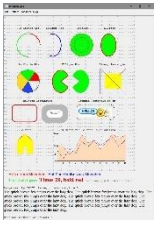
Inherits from:	- (none)
Inherited by:	- (none)
Class prototype:	printerclass.fh
Defines:	- (none)
Instantiated to	oPrinter and _oPrinter

Printer Class Index

Color	ACC/ASS	Set/return the printers color property
Display()	METHOD	Display the printer spooler file
DocName	ACC/ASS	Return/Set the print document for GUI
Driver	ACC/ASS	Return/Set the default driver name
DriverHldpi	ACCESS	Get the horizontal driver resolution in LDPI
DriverHmargin	ACCESS	Get the horizontal margin of the driver
DriverVldpi	ACCESS	Get the vertical driver resolution in LDPI
DriverVmargin	ACCESS	Get the horizontal margin of the driver
Exec()	METHOD	Invokes the printer-spooler
ExecBlock	ACC/ASS	User code block replacing Exec() behavior
ExecFormatted()	METHOD	Reformats file/output and print it
ExecPrintScreen()	METHOD	Prints the whole user-screen
Font	ACC/ASS	Optional font object, used by :ExecFormatted()
HeaderLeft	ACC/ASS	Left part of header for ExecFormatted()
HeaderMid	ACC/ASS	Center part of header for ExecFormatted()
HeaderRight	ACC/ASS	Right part of header for ExecFormatted()
InputFileName	ACC/ASS	Get/set the file name to print
InputUser()	ASSIGN	File name assigned by SET PRINTER TO <file>
MarginBottom	ACC/ASS	Set/get the bottom margin
MarginBottom()	METHOD	Set/get the bottom margin
MarginLeft	ACC/ASS	Set/get the left margin
MarginLeft()	METHOD	Set/get the left margin
MarginRight	ACC/ASS	Set/get the right margin
MarginRight()	METHOD	Set/get the right margin
MarginTop	ACC/ASS	Set/get the top margin

MarginTop()	METHOD	Set/get the top margin
NumCopies	ACC/ASS	Number of copies considered by :Exec*()
Orientation	ACC/ASS	Portrait or Landscape, for user ExecBlock
OutputFileName	ACC/ASS	Optional for user ExecBlock
PageAll	ACC/ASS	Optional for user ExecBlock
PageFrom	ACC/ASS	Optional for user ExecBlock
PageOrder	ACC/ASS	First or Last, for user ExecBlock
PageSize	ACC/ASS	Optional for user ExecBlock
PageTo	ACC/ASS	Optional for user ExecBlock
Print	ACC/ASS	Enable/disable printing to GUI driver
PrintExecutable	ACC/ASS	Set/get the default spooler executable
Setup()	METHOD	Dialog to select printer properties (GUI only)
SetupAborted	ACCESS	Was the SetUp() aborted by user?
Show()	METHOD	Equivalent to Display()
TabStop	ACC/ASS	Tabulator size for :ExeFormatted()
GUIabort()	METHOD	GUI only: abort output
GUIcol()	METHOD	GUI only: get/set current column
GUIcolWidth()	METHOD	GUI only: get column width
GUIdevOut()	METHOD	GUI only: print text same as DevOut()
GUIdrawBox()	METHOD	GUI only: draw box
GUIdrawLine()	METHOD	GUI only: draw line
GUIexec()	METHOD	GUI only: print, same as PrintGui()
GUIfixPage()	METHOD	GUI only: fix page size
GUImaxCol()	METHOD	GUI only: max columns on page
GUImaxRow()	METHOD	GUI only: max rows on page
GUInewLine()	METHOD	GUI only: line feed, same as Qout()
GUInewPage()	METHOD	GUI only: form feed, same as EJECT
GUIpageNum	ACC/ASS	GUI only: get/set page number
GUIrowHeight()	METHOD	GUI only: get row height
GUIrow()	METHOD	GUI only: get/set current row
GUIsetColor()	METHOD	GUI only: set default color
GUIsetFont()	METHOD	GUI only: set default font
GUIsetPos()	METHOD	GUI only: set new print position
GUIstart()	METHOD	GUI only: init printout
GUItestPage()	METHOD	GUI only: print test page
GUItextOut()	METHOD	GUI only: print text same as Qqout()

The **source** of the printer class is available in *<FlagShip_dir>/ system/o*printer*.prg* files. See also examples in *<FlagShip_dir>/examples/printer.prg* and *printergui.prg*



Printer Class Instantiation

The printer class is instantiated automatically in initio.prg and is stored in global constant named "oPrinter"

Printer Class Properties

oPrinter:Color —> isColor ***ACCESS***
oPrinter:Color := isColor ***ASSIGN***

Set/return the printers color property This is optional property, not used directly by :Exec() but my be used and considered in user-defined :ExecBlock The default is .T. or an user-set value from :SetUp()

oPrinter:Display ([nTop],[nLeft],[nBott],[nRight], [lPixel],[nLineSize],[cUdf]) —> lOk ***METHOD***

Displays the printer spooler file via MemoEdit(). The <nTop>, <nLeft>, <nBott> and <nRight> are optional MemoEdit() coordinates. When <lPixel> is .T., these coordinates are in pixel, .F. in row/cols, and NIL select default SET PIXEL. <nLineSize> determines the line length, which defaults to <nRight> - <nLeft>. If <nLineSize> is greater that the default, the window scrolls horizontally. <cUdf> is a string specifying the name of a user-defined function to control the editing process.

oPrinter:DocName —> cName ***ACCESS***
oPrinter:DocName := cName ***ASSIGN***

Set/return the document name in GUI mode for PrintGui(). The Assign must be set before SET GUIPRINT ON or PrintGui(.T.) or oPrinter:Setup() to be considered. If not set, the default name is ExecName()_YYYYMMDD.HHMMSS

oPrinter:Driver —> cPrintDriver ***ACCESS***
oPrinter:Driver := cPrintDriver ***ASSIGN***

Return/Set the default driver name (system dependant). The default driver is user-selected in oPrinter:SetUp() and is e.g. "Printer1" on Unix or driver name in MS-Windows. The driver name is used in oPrinter:Exec()

oPrinter:DriverHldpi —> nResol ***ACCESS***

Get the horizontal driver resolution in LDPI (logical dots per inch). It is not the real printer resolution, but the unit used for margins. For laser printers, the returned value is usually 72 x 72 dpi = 2.83 x 2.83 dots per mm. If the printer is not set yet, use default printer.

oPrinter:DriverHmargin —> nPixel *ACCESS*

Get the horizontal margin of the driver in pixel. If the printer is not set yet, use default printer

oPrinter:DriverVldpi —> nResol *ACCESS*

Get the vertical driver resolution in LDPI (logical dots per inch). It is not the real printer resolution, but the unit used for margins. For laser printers, the returned value is usually 72 x 72 dpi = 2.83 x 2.83 dots per mm. If the printer is not set yet, use default printer.

oPrinter:DriverVmargin —> nPixel *ACCESS*

Get the horizontal margin of the driver in pixel. If the printer is not set yet, use default printer

oPrinter:Exec ([nWait],[ncMode]) —> lOk *METHOD*

Invokes the printer-spooler.

<**nWait**> is optional waiting period in seconds after printing (to display messages), default = 5sec.

<**ncMode**> = 0 or NIL: in Linux, invoke lpr or lp or driver set by PrintExecutable. In Windows, invokes native WinSpool driver (the source is available in o_printer.prg). In GUI mode, check for Setup() is performed.

<**ncMode**> = 1: in Linux, use "cp <file> /dev/lp0", in Windows, use "copy <file> PRN:"

<**ncMode**> = string: used port/device, e.g. "LPT3:" in Windows, or "/dev/lp2" in Linux/Unix The printed file name is either set by oPrinter:InputFileName or is determined by FS_SET("print") otherwise.

Before calling oPrinter:Exec(), either

a) execute oPrinter:Setup(), otherwise it will be called automatically in GUI mode
b) or assign oPrinter:Driver := "My Printer Name" (but insecure)
c) or use oPrinter:Exec(,1)
d) or use oPrinter:Exec(,"device name")

otherwise simple copy <printerfile> to standard printer device is used, same as with oPrinter:Exec(,1). You also may set oPrinter:NumCopies, oPrinter:MarginTop and oPrinter:MarginLeft beforehand.

Note: The _oprinter:Exec() may also be called from main menu File->Print (see initiomenu.prg) via STATIC FUNCTION InitIoPrint(obj, menuID)

Example: see LNG.5.1.8.c and <FlagShip_dir>/examples/printer.prg

oPrinter:ExecBlock –> bCodeBlock *ACCESS*
oPrinter:ExecBlock := bCodeBlock or NIL *ASSIGN*

User code block replacing the Exec() behavior. If the code block is specified, oPrinter:Exec() invokes this block instead if the standard oPrinter:Exec() method.

oPrinter:ExecFormatted ([cFileName],[oFont]) –> lOk *METHOD*

Reads the <cFileName>, reformats output according to <oFont>, invokes the printer-spooler set by oPrinter:SetUp(). If <cFileName> is not specified, oPrinter:InputFile-Name or FS_SET("print") is used. The <oFont> is considered in GUI mode only and is optional font object specifying the printer font. It also may be set by oPrinter:Font.

Before executing ExecFormatted(), you may change it behavior by HeaderLeft, HeaderMid, HeaderRight, MarginTop, MarginBottom, MarginLeft, MarginRight, NumCopies, TabStop

In GUI mode, following attributes within the text are supported:

`<b> ... </b>` - Bold font style
`<i> ... </i>` - Italic font style
`<u> ... </u>` - Underlined font style

In Terminal i/o mode, oPrinter:ExecFormatted() behaves same as oPrinter:Exec().

Example: see LNG.5.1.8.c and <FlagShip_dir>/examples/printer.prg

oPrinter:ExecPrintScreen ([lAdapt]) –> lOk *METHOD*

Prints the whole user-screen to default spooler according to :SetUp() The oPrinter:PrintScreen() is also executed from main menu (see initiomenu.prg) either via STATIC FUNCTION InitIoScrPrint(obj, menuID) or user redefined

oPrinter:Font –> oFont *ACCESS*
oPrinter:Font := oFont *ASSIGN*

Optional font object, used by oPrinter:ExecFormatted()

oPrinter:HeaderLeft –> cText *ACCESS*
oPrinter:HeaderLeft := cText *ASSIGN*

Left part of header for ExecFormatted(). <cText> can be any string including special macros "<>" for page number, "<date>" for current date, "<time>" for current time, "<file>" for printer file name. The default setting is "Page <>"

oPrinter:HeaderMid —> cText *ACCESS*
oPrinter:HeaderMid := cText *ASSIGN*

The mid part of header for ExecFormatted(). <cText> can be any string including special macros "<>" for page number, "<date>" for current date, "<time>" for current time, "<file>" for printer file name. The default setting is "<date> <time>"

oPrinter:HeaderRight —> cText *ACCESS*
oPrinter:HeaderRight := cText *ASSIGN*

Right part of header for ExecFormatted(). <cText> can be any string including special macros "<>" for page number, "<date>" for current date, "<time>" for current time, "<file>" for printer file name. The default setting is "<file>"

oPrinter:InputFileName —> cFileName *ACCESS*
oPrinter:InputFileName := cFileName *ASSIGN*

Get/set the file name to print by oPrinter:Exec() or ExecFormatted(). If not specified, the default printer spooler file name is either the SET PRINTER TO <cFileName>, or is determined by the FS_SET("print") function.

oPrinter:InputUser(cFileName) *METHOD*

File name assigned by SET PRINTER TO <cFileName>. This method is for internal use only.

oPrinter:MarginBottom —> nValue *ACCESS*
oPrinter:MarginBottom := nValue *ASSIGN*

Set/get the bottom margin for ExecFormatted() and ExecPrintScreen() and PrintGui() in GUI mode. <nValue> is in LDPI (logical dots per inch). Defaults are defined in printerclass.fh, but best set it to

```
MarginLeft & MarginRight = oPrinter:DriverHmargin and
MarginTop & MarginBottom = oPrinter:DriverVmargin
```

i.e. corresponding to the currently used printer driver. Example: see <FlagShip_dir>/examples/printer.prg and printergui.prg

oPrinter:MarginBottom([unit],[newValue]) —> nValue *METHOD*

Set/get the bottom margin in GUI mode, same as :MarginBottom ACC/ASS but accepts conversion from/to units. If <unit> is not given, current SET PIXEL or SET COORD UNIT is used, default is row/col.

oPrinter:MarginLeft –> nValue *ACCESS*
oPrinter:MarginLeft := nValue *ASSIGN*

Set/get the left margin for ExecFormatted() and ExecPrintScreen() and ExecGUI() and PrintGui() in GUI mode. Analog to oPrinter:MarginBottom, see additional description there. This instance is set also by SET MARGIN TO ... command when `_aGlobSetting[GSET_L_SETPRINTER_MARGIN]` is set .T. (default is .F.)

oPrinter:MarginLeft([unit],[newValue]) –> nValue *METHOD*

Set/get the bottom margin in GUI mode, same as :MarginLeft ACC/ASS but accepts conversion from/to units. If <unit> is not given, current SET PIXEL or SET COORD UNIT is used, default is row/col.

oPrinter:MarginRight –> nValue *ACCESS*
oPrinter:MarginRight := nValue *ASSIGN*

Set/get the right margin for ExecFormatted() and ExecPrintScreen() in GUI mode. Analog to oPrinter:MarginBottom, see additional description there.

oPrinter:MarginRight([unit],[newValue]) –> nValue *METHOD*

Set/get the bottom margin in GUI mode, same as :MarginRight ACC/ASS but accepts conversion from/to units. If <unit> is not given, current SET PIXEL or SET COORD UNIT is used, default is row/col.

oPrinter:MarginTop –> nValue *ACCESS*
oPrinter:MarginTop := nValue *ASSIGN*

Set/get the top margin for ExecFormatted() and ExecPrintScreen() and PrintGui() in GUI mode. Analog to oPrinter:MarginBottom, see additional description there.

oPrinter:MarginTop([unit],[newValue]) –> nValue *METHOD*

Set/get the bottom margin in GUI mode, same as :MarginTop ACC/ASS but accepts conversion from/to units. If <unit> is not given, current SET PIXEL or SET COORD UNIT is used, default is row/col.

oPrinter:NumCopies –> nValue *ACCESS*
oPrinter:NumCopies := nValue *ASSIGN*

Set/get the number of copies for PrintGui(), Exec() or ExecFormatted(). This value is considered by :Setup() in GUI mode (invoked implicitly by the first PrintGui(.T.) if not done yet). You may set the value programatically by assigning required number of copies - but before invoking oPrinter:Setup() or PrintGui(.T.). The default value is 1. The user may however change it by *[Number of copies]* or *[Preferences]* or *[Property]* in the printer dialog.

oPrinter:Orientation —> cValue ***ACCESS***
oPrinter:Orientation := cValue ***ASSIGN***

Set/get printer orientation. <cValue> is either "Portrait" or "Landscape", input of "P" and "L" is accepted as well. This value is considered by :Setup() in GUI mode, invoked also implicitly by the first PrintGui(.T.) if required. The user may change it by *[Property]* or *[Preferences]* of the printer dialog. You may change the value programatically by assign and new invocation of oPrinter: Setup(). The default is "Portrait" or user-selected value from Setup().

oPrinter:OutputFileName —> cValue ***ACCESS***
oPrinter:OutputFileName := cValue ***ASSIGN***

Set/get output file name. <cValue> is any string. This value is considered by :Setup() in GUI mode, invoked also implicitly by the first PrintGui(.T.) if required. The user may change it by *[Property]* or *[Preferences]* of the printer dialog. You may change the value programatically by assign and new invocation of oPrinter: Setup(). The default is "" or user-selected value from Setup(). Note: this functionality is currently supported in Linux only.

oPrinter:PageAll —> lSet ***ACCESS***
oPrinter:PageAll := .T. ***ASSIGN***

Set/get status for printing of all pages. <lSet> is .T. when PageFrom and/or PageTo is 0, otherwise .F. ASSIGN accepts .T. only, and will reset both PageFrom and PageTo to 0. This value is considered by :Setup() in GUI mode, invoked also implicitly by the first PrintGui(.T.) if required. The user may change it by *[Property]* or *[Preferences]* of the printer dialog. You may change the value programatically by assign and new invocation of oPrinter:Setup().

oPrinter:PageFrom —> nPage ***ACCESS***
oPrinter:PageFrom := nPage ***ASSIGN***

Set/get the first printed page. If <nPage> is 0 (set by PageAll), all pages are printed; otherwise specify the 1st page number (1..n). This value is considered by :Setup() in GUI mode, invoked also implicitly by the first PrintGui(.T.) if required. The user may change it by *[Property]* or *[Preferences]* of the printer dialog. You may change the value programatically by assign and new invocation of oPrinter:Setup(). The default is 0 or user-selected value from Setup().

oPrinter:PageOrder —> cValue ***ACCESS***
oPrinter:PageOrder := cValue ***ASSIGN***

Set/get printer page order. <cValue> is either "First" or "Last", input of "F" and "L" is accepted as well. This value is considered by :Setup() in GUI mode, invoked also

implicitly by the first PrintGui(.T.) if required. The user may change it by *[Property]* or *[Preferences]* of the printer dialog. You may change the value programatically by assign and new invocation of oPrinter:Setup(). The default is "First" or an user-selected value from SetUp().

oPrinter:PageSize —> cValue *ACCESS*
oPrinter:PageSize := cValue *ASSIGN*

Set/get used page size. <cValue> is "A4" or an user-selected value from SetUp(), ASSIGN accept any char value w/o check. This value is considered by :Setup() in GUI mode, invoked also implicitly by the first PrintGui(.T.) if required. The user may change it by *[Property]* or *[Preferences]* of the printer dialog. You may change the value programatically by assign and new invocation of oPrinter: Setup(). The default is "A4" or user-selected value from SetUp().

oPrinter:PageTo —> nPage *ACCESS*
oPrinter:PageTo := nPage *ASSIGN*

Set/get the last printed page. If <nPage> is 0 (set by PageAll), all pages are printed; otherwise specify the last page number (1..n). This value is considered by :Setup() in GUI mode, invoked also implicitly by the first PrintGui(.T.) if required. The user may change it by *[Property]* or *[Preferences]* of the printer dialog. You may change the value programatically by assign and new invocation of oPrinter:Setup(). The default is 0 or user-selected value from SetUp().

oPrinter:Print —> lOn *ACCESS*
oPrinter:Print := lOn *ASSIGN*

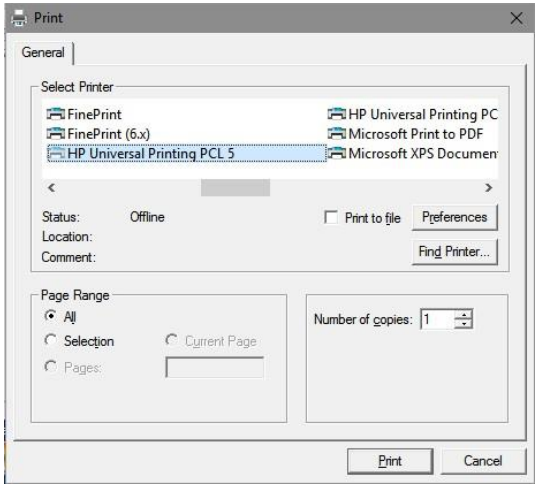
Enable/disable printing to GUI driver. The default is .T. (enabled)

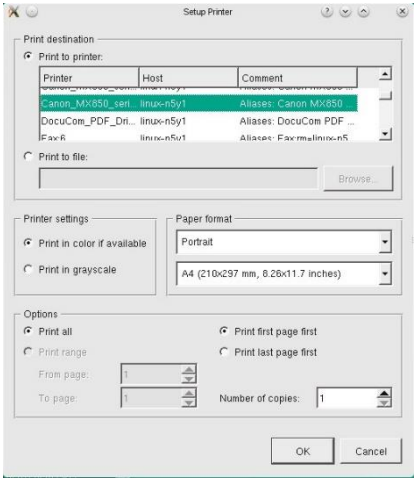
oPrinter:PrintExecutable —> cName *ACCESS*
oPrinter:PrintExecutable := cName *ASSIGN*

Set/get the default spooler executable (system dependant), user- selected in SetUp(). Used in Exec() for Linux. If not set manually, "lpr" or "lp" is returned in Unix/Linux, otherwise "".

oPrinter:Setup() *METHOD*

Pop-up dialog to select printer properties, considered in GUI mode only for Exec(), ExecFormatted() and ExecPrintScreen(). If not yet invoked explicitly, it is called by the Exec*() method or the first PrintGui(.T.) function. The Setup() is also executed from main menu (see initiomenu.prg) either via STATIC FUNCTION InitIoPrSet(obj, menuID) or user redefined.





oPrinter: SetupAborted —> lStatus *ACCESS*

Reports whether the SetUp() was aborted by user. <lStatus> returns .T. when the "Cancel" button was pressed in the Setup() pop-up dialog, and .F. otherwise. You may check this property to abort printing.

oPrinter:Show() *METHOD*

This method is equivalent to oPrinter:Display()

oPrinter:TabStop —> aValues *ACCESS*
oPrinter:TabStop := aValues or NIL *ASSIGN*

Set/get tabulator position (in LDPI pixels) for GUI ExecFormatted(), considered for \t = chr(9) character in the text line. <aValues> is one-dimensional array of numeric values. Every Tab position occupy one numeric element in the array, also unsorted, e.g.

```
oPrinter:TabStop := {10,20,50,120}
```

in LDPI units, see DriverHldpi. All tab values must be > 0, otherwise no tabs will be set. Reset by an empty array or NIL assignment.

Example: see <FlagShip_dir>/examples/printer.prg

oPrinter:GUIabort() —> lOk *METHOD*

Abort current printing, clear rendering buffer, SETs GUIPRINT OFF. Returns .T. on success and .F. on failure with developer's warning, i.e. if oPrinter:GUIstart() or SET GUIPRINT ON or PrintGui(.T.) was yet not invoked. Applicable in GUI mode only, ignored otherwise.

oPrinter:GUIcol([unit], [newPos]) —> nPos *METHOD*

Get/set current printer column.

<**unit**> is optional numeric value (see UNIT_* in set.fh). If not given, current SET COORD UNIT or SET PIXEL is used (default is UNIT_ROWCOL = row/col).

<**newPos**> is optional numeric value specifying new printer column in <units>.

Returns current (before setting) printer column in <unit>s, or 0 when GUI printout was yet not activated by oPrinter:GUIstart() or SET GUIPRINT ON or PrintGui(.T.). Applicable in GUI mode only, ignored otherwise.

oPrinter:GUIcolWidth([unit]) —> num *METHOD*

Get column width according the current font.

<**unit**> is optional numeric value (see UNIT_* in set.fh). If not given, current SET COORD UNIT or SET PIXEL is used (default is UNIT_ROWCOL = row/col).

Returns current printer column width in <unit>s, or 0 when GUI printout was yet not activated by oPrinter:GUIstart() or SET GUIPRINT ON or PrintGui(.T.). Applicable in GUI mode only, ignored otherwise.

oPrinter:GUIdevOut(cText,[coColor], [oFont]) —> NIL *METHOD*

Print text same as DevOut() or Qqout(...) functions and ?? command. This method is for your convenience and equivalent to ::GUItextOut()

<**cText**> is the string to be printed. SET GUITRANS is considered.

<**coColor**> is optional color as string or Color object.

<**oFont**> is optional font specification as Font object.

Applicable when GUI printout was activated by oPrinter:GUIstart() or SET GUIPRINT ON or PrintGui(.T.), ignored otherwise.

oPrinter:GUIdrawBox(p1,p2,p3,p4,p5,p6,p7,p8) —> NIL *METHOD*

Draw box similar to @..BOX command or DispBox() function.

<**p1**> optional top row position. If not given, current row is used.

<**p2**> optional left column position. If not given, current column is used.

<**p3**> optional bottom row position, otherwise GuiMaxRow() is used.

<**p4**> optional right column position, otherwise GuiMaxCol() is used.

<**p5**> optional line width in dots, default is 1.

<**p6**> optional rounding corners (0..99), zero draws angled corners

<**p7**> optional color as string or array or Color object.

<**p8**> optional unit value (see UNIT_* in set.fh).

Applicable when GUI printout was activated by oPrinter:GUIstart() or SET GUIPRINT ON or PrintGui(.T.), ignored otherwise.

oPrinter:GUIdrawLine(p1,p2,p3,p4,p5,p6,p7) —> NIL *METHOD*

Draw line similar to @..DRAW or @..TO.. command.

<**p1**> optional start row position. If not given, current row is used.

<**p2**> optional start column position. If not given, current column is used.

<**p3**> end row position in <p7> units.

<**p4**> end column position in <p7> units.

<**p5**> optional line width in dots, default is 1.

<**p6**> optional color as string or array or Color object.

<**p7**> optional unit value (see UNIT_* in set.fh), default is row/col.

Applicable when GUI printout was activated by oPrinter:GUIstart() or SET GUIPRINT ON or PrintGui(.T.), ignored otherwise.

oPrinter:GUIexec() —> lSuccess *METHOD*

Print GUI buffer created by oPrinter:GUI*() methods or by common @..SAY, ?, ??, @..DRAW, @..BOX commands or associated functions. Applicable after oPrinter:GUIstart() or PrintGui(.T.) or SET GUIPRINT ON. This method is equivalent to PrintGui() w/o parameter. Considered in GUI mode only, ignored otherwise.

oPrinter:GUIfixPage([aSize]) —> aSize *METHOD*

Adjust or report page adjustment size for currently by :Setup() selected printer. Such adjustment may be desirable to utilize whole printable page size, since the most standard drivers reports approximate values only.
<**aSize**> is an array specifying the printable page adjustment

aSize[1] = optional string containing printer driver name
aSize[2] = optional string containing page size
aSize[3] = optional string containing page orientation
aSize[4] = numeric value for unit sizes, one of UNIT_* in set.fh
aSize[5] = array of 4 numeric elements for 1st (and next) pages, all four values are in units according to aSize[4]
aSize[5,1] = non-printable margin left
aSize[5,2] = printable width
aSize[5,3] = non-printable margin top
aSize[5,4] = printable height
aSize[6] = optional array of 4 numeric elements for 2nd (and next) pages, same structure as aSize[5]. If not given, aSize[5] data are used also for second and next pages.

Returns <aSize> in above structure or an empty array if adjustment was not specified yet.

Example:

```
// Hint: for the first time, execute this example with disabled
// oPrinter:GUIfixPage(aSetup) statement, then adjust the aSetup
// values according to printout, enable oPrinter:GUIfixPage(aSetup)
// statement and check anew.
#ifdef FS_WIN32   // Windows
 aSetup := {"Canon MX850 series Printer", ;
            "A4", "Portrait", UNIT_MM, ;
            {3.4, 203.1, 5, 286.8}, {0, 203.1, 0, 286.8} }
#else             // Linux
 aSetup := {"Canon_MX850", "A4", "Portrait", UNIT_CM, ;
            {0.31, 20.34, 0.42, 28.76} }
#endif
oPrinter:Setup()                // select printer driver
if oPrinter:SetupAborted
  wait "sorry ..."
  quit
endif
oPrinter:GUIstart()             // start GUI rendering
// oPrinter:GUIfixPage(aSetup)  // optional: set adjustment
oPrinter:GUItestPage()          // create test page #1
oPrinter:GUInewPage()           // form feed
oPrinter:GUItestPage()          // create test page #2
oPrinter:GUIexec()              // end GUI rendering, print it
wait
```

oPrinter:GUImaxCol([unit]) —> num *METHOD*

Reports max available columns per page

<**unit**> is optional numeric value (see UNIT_* in set.fh). If not given, current SET COORD UNIT or SET PIXEL is used (default is UNIT_ROWCOL = row/col).

Returns max printer columns in <unit>s according to current font, or 0 when GUI printout was yet not activated by oPrinter:GUIstart() or SET GUIPRINT ON or PrintGui(.T.). Considered in GUI mode only, ignored otherwise.

oPrinter:GUImaxRow([unit]) —> num *METHOD*

Reports max available rows per page

<**unit**> is optional numeric value (see UNIT_* in set.fh). If not given, current SET COORD UNIT or SET PIXEL is used (default is UNIT_ROWCOL = row/col).

Returns max printer rows in <unit>s according to current font, or 0 when GUI printout was yet not activated by oPrinter:GUIstart() or SET GUIPRINT ON or PrintGui(.T.). Considered in GUI mode only, ignored otherwise.

oPrinter:GUInewLine() —> NIL *METHOD*

Execute line feed, sets printer position on 1st column in new line, same as ? or Qout(""). The row height is calculated from current font height plus global value from `_aGlobSetting[GSET_G_N_ROW_SPACING]`. Applicable when GUI printout was activated by oPrinter:GUIstart() or SET GUIPRINT ON or PrintGui(.T.), ignored otherwise.

oPrinter:GUInewPage() —> NIL *METHOD*

Execute form feed, sets printer position on 1st column in new page, increases page number (oPrinter:GUIpageNum), comparable to EJECT command. Applicable when GUI printout was activated by oPrinter: GUIstart() or SET GUIPRINT ON or PrintGui(.T.), ignored otherwise.

oPrinter:GUIpageNum —> num *ACCESS*
oPrinter:GUIpageNum := num *ASSIGN*

Get/set current page number starting at 1. 0 is reported when GUI printout was yet not activated by oPrinter:GUIstart() or SET GUIPRINT ON or PrintGui(.T.), and after oPrinter:GUIexec(). Considered in GUI mode only, ignored otherwise.

oPrinter:GUIrowHeight([unit]) —> num *METHOD*

Get row height according the current font.

<**unit**> is optional numeric value (see UNIT_* in set.fh). If not given, current SET COORD UNIT or SET PIXEL is used (default is UNIT_ROWCOL = row/col).

Returns current printer row height in <unit>s, or 0 when GUI printout was yet not activated by oPrinter:GUIstart() or by SET GUIPRINT ON or PrintGui(.T.). Considered in GUI mode only, ignored otherwise.

oPrinter:GUIrow([unit], [newPos]) —> nPos *METHOD*

Get/set current printer row.

<**unit**> is optional numeric value (see UNIT_* in set.fh). If not given, current SET COORD UNIT or SET PIXEL is used (default is UNIT_ROWCOL = row/col).

<**newPos**> is optional numeric value specifying new printer row in <units>.

Returns current (before setting) printer row in <unit>s, or 0 when GUI printout was yet not activated by oPrinter:GUIstart() or by SET GUIPRINT ON or PrintGui(.T.). Considered in GUI mode only, ignored otherwise.

oPrinter:GUIsetColor([coColor]) —> lSuccess (or cColor) *METHOD*

Set default printer text and drawing color. This can be temporarily overwritten by PRINTCOLOR clause of @..SAY, @..DRAW, ?, ?? etc. commands, see below. Considered in GUI mode only, ignored otherwise.

<**coColor**> is a string according to SET COLOR "foreground/background" or a Color object. Only the first (standard) color pair is used. Empty string (default) disables this setting, NIL or none parameter returns current setting as a string.

Returns .T. on success or .F. when GUI printout was yet not activated by oPrinter:GUIstart() or by SET GUIPRINT ON or PrintGui(.T.). When <coColor> is NIL, not given or is a empty string, the current color string is returned, when GUI printout was activated.

The used color for GUI printouts is determined in this sequence:

a) PRINTCOLOR clause of @..SAY, @..DRAW, ?, ?? etc. commands
b) oPrinter:GUIsetColor() setting, if not empty
c) GUICOLOR clause of @..SAY, @..DRAW, ?, ?? etc. commands
d) SetColor() when SET GUICOLOR is ON
e) otherwise "N/W+"

oPrinter:GUIsetFont([oFont]) –> oFont *METHOD*

Set or get default printer font (standard is set to Courier,10). This method is equivalent to oPrinter:Font := oFont assignment. The font can be temporarily overwritten by corresponding method parameter or the FONT command clause of @..SAY, ?, ?? etc. commands.

<**oFont**> is optional Font object.

Returns current font object (before setting) or NIL when GUI printout was yet not activated by oPrinter:GUIstart() or by SET GUIPRINT ON or PrintGui(.T.). Considered in GUI mode only, ignored otherwise.

oPrinter:GUIsetPos(row, col, [unit]) –> lSuccess *METHOD*

Set new printer's position. Alternative methods are oPrinter:GUIcol(), oPrinter:GUIrow(), oPrinter:GUInewLine() and oPrinter:GUInewPage().

<**row**> is numeric value specifying the row or y position in <unit>s.

<**col**> is numeric value specifying the col or y position in <unit>s.

<**unit**> is optional numeric value (see UNIT_* in set.fh). If not given, current SET COORD UNIT or SET PIXEL is used (default is UNIT_ROWCOL = row/col).

Returns .T. on success or .F. when GUI printout was yet not activated by oPrinter:GUIstart() or by SET GUIPRINT ON or PrintGui(.T.). Considered in GUI mode only, ignored otherwise.

oPrinter:GUIstart([oFont]) –> lSuccess *METHOD*

Init (or continue) GUI printout rendering. May be called subsequently; when the rendering is already initialized, nothing happens. This method is invoked (without parameter) also by SET GUIPRINT ON and PrintGui(.T.).

<**oFont**> is optional Font object, set as default.

Returns .T. on success or .F. when GUI printout was already activated Considered in GUI mode only, ignored otherwise.

oPrinter:GUItestPage() –> lSuccess *METHOD*

Print test page with driver/margin/page settings and box around the printable area. See example in oPrinter:GUIfixPage() above.

Returns .T. on success or .F. when GUI printout was yet not activated by oPrinter:GUIstart() or by SET GUIPRINT ON or PrintGui(.T.). Considered in GUI mode only, ignored otherwise.

oPrinter:GUItextOut(cText, [coColor], [oFont]) —> NIL *METHOD*

Print text similarly to ?? command or DevOut() or Qqout() functions.

<**cText**> is the string to be printed. SET GUITRANS is considered.

<**coColor**> is optional color as string or Color object.

<**oFont**> is optional font specification as Font object.

Applicable when GUI printout was activated by oPrinter:GUIstart() or SET GUIPRINT ON or PrintGui(.T.), ignored otherwise.

Push Button Class

The push button, also referred to as command button, is perhaps the most central widget in any graphical user interface: Push it to perform some associated action. Typical actions are Ok, Apply, Cancel, Close or Help by executing user defined function specified by oPush:Notify or :Sblock or :Fblock instances.

The following code creates a push button labeled "Press Me" and executes function ButUDF() when the button is pressed/clicked:

```
oButton := PushButton{17,50, "Press Me", , {|obj| ButUDF(obj)} }
oButton:Show()
// ...
FUNCTION ButUDF(oButt)
  @ 24,0 SAY "Button [" + oButt:Caption + "] pressed at " + time(1)
return
```

FlagShip also support the use of push buttons via the common @..GET / READ interface

```
local lButt := .F.
local cData := space(20)
@ 5,14 GET lButt PUSHBUTTON CAPTION "Any action" ;
                 NOTIFY {|oPush| ButUDF(oPush) }   // see above
@ 7, 5 SAY "Any data"
@ 7,14 GET cData
READ
```

The text can be changed anytime later with oPushBut:Caption. You can also define a pixmap with oPushBut:Pixmap(). The text/pixmap is manipulated as necessary to create "disabled" appearance according to the respective GUI style when the button is disabled. A command button can, in addition to the text or pixmap label, also display a little icon, see oPushButt:Bitmap and oPushButt:SetImage() for details.

When the push button is activated, either with the mouse, the spacebar or a keyboard accelerator, it calls a user code block, if such is supplied by oPushButt:Notify, oPushButt:FBlock and/or oPushButt:SBlock, which then process the required action.

Command buttons in dialogs are by default auto default buttons, i.e. they become the default push button automatically when they receive the keyboard input focus. A default button is a command button that is activated when the users hits the Enter or Return key in a dialog. Adjust this behavior with <expL4> during the instantiation, or later by oPushButt:Default(). The default buttons reserve a little extra space necessary to draw a default button indicator. If you do not want this space around your buttons, use `oPushButt:SetStyle(BUT_AUTO-BORDER,.F.)` or `oPushButt:SetStyle(SETSTYLE_FLAT)`

There are sometimes confusions when to use push or other buttons. As a general rule, use a push button when the application or dialog window performs an action when the user clicks on it (like Apply, Cancel, Close, Help, etc.) and when the widget is supposed to have a wide,

rectangular shape with a text label. See RadioButton, CheckButton and Menu classes for other GUI command buttons.

As with other i/o and GUI classes in FlagShip, the generic PushButton class stay either for _gPushButton for GUI based application, _tPushButton for terminal/text based, or _bPushButton for basic i/o. When the general PushButton class is instantiated, FlagShip will assign the proper i/o class either at compile-time or run-time.

Note: in the basic i/o mode, only a rough push button functionality is emulated by the sequential in/output.

PushButton Class Index

Class PushButton
Inherits from: - (none)
Inherited by: - (none)
Class prototype: buttonclass.fh
Defines: button.fh

AsString()	METHOD	Return an identifying label for the button
Bitmap	ACC/ASS	A bitmap file displayed as a icon
Buffer	ACC/ASS	Indicates that the push button has been pushed
Caption	ACC/ASS	The displayed text of the button
Cargo	ACC/ASS	User data of any type
ClassName()	METHOD	returns "PUSHBUTTON"
Col	ACC/ASS	Upper left push button coordinate
ColorSpec	ACC/ASS	Color attributes, ignored in GUI mode
CurrentText	ACC/ASS	Same as oPushButton:Caption
Display()	METHOD	Display the PushButton, same as Show()
Enable()	METHOD	Enable or disable the button availability
Exec()	METHOD	Display the PushButton widget and execute handler
ExitState	ACC/ASS	state of @..GET/READ
FBlock	ACC/ASS	Code block evaluated at receiving/losing focus
Font	ACC/ASS	The used font object
Font()	METHOD	The used font object
GuiColor	ACC/ASS	Color attributes for GUI mode
HasFocus	ACC	Status of the bush button input focus
Height()	METHOD	Height of the push button widget
Hide()	METHOD	Hide the PushButton
HitTest()	METHOD	Determining if the mouse is within the push button
KillFocus()	METHOD	Take input focus away from the PushButton
Message	ACC/ASS	Short help in the window status line
Notify	ACC/ASS	Code block evaluated when the push button is pressed
OnClickAction	ACC/ASS	Action in READ triggered by code block
OnClickKeys	ACC/ASS	Simulates key press, triggered by code block

Resize()	METHOD	Resize the push button widget to a new size
Row	ACC/ASS	Upper left push button coordinate
SBlock	ACC/ASS	Code block evaluated on button push and release
Select()	METHOD	Simulates the button press
SetFocus()	METHOD	Gives input focus to the PushButton object
SetImage()	MMETHOD	Assign an image to PushButton
SetStyle()	METHOD	Set/return the style of the Push button widget
Show()	METHOD	Display the PushButton, same as :Display()
SizeX	ACC/ASS	The horizontal size (width) of the widget
SizeY	ACC/ASS	The vertical size (height) of the widget
Style	ACC/ASS	For terminal mode only
ToolTip	ACC/ASS	Short pop-up info message
Visible	ACC	Report the push button visibility
Width()	METHOD	Set/return the width of the push button widget
X()	METHOD	Set/return the x coordinate of the button widget
Y()	METHOD	Set/return the y coordinate of the button widget

PushButton Class Instantiation

Syntax 1:

```
oPushButton := [_g|_t|_b]PushButton {[nR], [nC],
          [cText], [lDef], [bNotif], [lPix]}
```

Syntax 2:

```
oPushButton := [_g|_t|_b]PushButtonNew ([nR], [nC],
          [cText], [lDef], [bNotif], [lPix])
```

Syntax 3:

```
oPushButton := PushButton ([nR], [nC], [cText],
          [lDef], [bNotif], [lPix])
```

Syntax 4:

```
oPushButton := PushButton {[oOwn], [nId], [oaPos],
          [oaSize], [cText], [nSty]}
```

Any of the above syntax instantiate new push button object. Syntax [1], [2] and [3] are standard FlagShip and should be preferred. Syntax [4] is supported for compatibility to VO.

<**nR**> Vertical coordinate (row) of the upper left edge. Either in pixels or col/row coordinates, dependent on the current state of SET PIXEL on|OFF or the <lPix> parameter. Modifiable by oPushButt:Row or oPushButt:Y(nR,[lPix]).

<**nC**> Horizontal coordinate (column) of the upper left edge. Either in pixels or col/row coordinates, dependent on the current state of SET PIXEL on|OFF or the <lPix> parameter. Modifiable by oPushButt:Col or oPushButt:X(nC,[lPix]).

<**cText**> The informational text to be printed in the button. If not given, null-string "" (i.e. no text) is displayed. See additional details in oPushButt:Caption.

<**lDef**> Set this button as default, i.e. button that is activated when the users hits the Enter or Return key. If this argument is not specified or of other value than .T., the button can only be activated by mouse click. To perform the "click" by keyboard, use Tab or cursor keys in conjunction with Return or space key.

<**bNotif**> is an optional code block, equivalent to oPushButton:Notify assignment. To activate this codeblock, subsequent oPushBut:Show() or oPushBut:Display() is required.

<**lPixel**> if true(.T.), the row and column data are in pixel; if false (.F.), data are in row/col coordinates, otherwise the current SET PIXEL is used.

<**oOwn**> The window that owns the button. If <expO1> is not given or empty, the User Window object <oUserWindow> is used.

<**nId**> An unique ID between 1 and 32000 which identify the button. If not given or is out of range, a next free ID will automatically be determined by FlagShip.

<**oaPos**> Button coordinates (upper left edge). Either a Point object (which allows entry in pixels or col/row), or an array of two numeric elements specifying the coordinates {row, col} in pixel. If not given 0,0 is assumed.

<**oaSize**> Button size (height and width). Either a Dimension object (which allows entry in pixels or col/row), or an array of two numeric elements specifying the button size {height, width} in pixel. If not given, the default button size is calculated from the current font and <cText>.

<**nSty**> The style of the push button, see BS_* constants in the button.fh file. Modifiable by oPushButton:SetStyle()

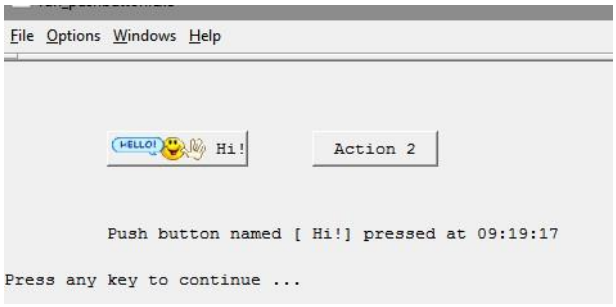
Compatibility: Available also in CL53 (syntax 3, first three params) and VO (syntax 4).

Example 1:

See also examples in FUN.PushButton() as well as above at the begin of Push Button Class description

Example 2:

```
local oPush1, oPush2
SET FONT "Courier", 10
oPush1 := PushButton(3, 10)
oPush1:Width(110,.T.)                    // width in pixel
oPush1:Height(28,.T.)                    // height in pixel
oPush1:Caption := " Hi!"                 // button caption
oPush1:Bitmap  := "hello.png"            // display image and/or text
oPush1:Notify  := {|obj| myNotif(obj)}   // action
oPush1:Show()                            // display

oPush2 := PushButton(3, 30, "Action 2", , {|obj| myNotif(obj)} )
oPush2:Bitmap := "action2.gif"           // display image if available
oPush2:Show()
setpos(8,0)
wait
return

FUNCTION myNotif(pushObj)
@ 7,10 say "Push button named [" + pushObj:Caption + "] pressed at " + ;
           time()
return
```



PushButton Class Properties

oPushButton:AsString() —> cLabel

Return an identifying label for the button of the form <ClassName>-<Caption>. For compatibility to VO only, don't use for new development.

oPushButton:Bitmap —> cFilename *ACCESS*
oPushButton:Bitmap := cFilename *ASSIGN*

Contains a character string that indicates a bitmap file to be displayed as a icon in addition to the text set by oPushButton:Caption. Specify full qualified name (including path) if the image file is not in the current or the SET PATH directory. FlagShip detects the image format automatically from the image data. Currently are supported following image file formats: .png, .bmp, .xbm, .xpm, .jpeg (.jpg) and .pnm in format PBM (P1 or P4), PGM (P2 or P5), PPM (P3 or P6). It also support .gif, but note: Unisys has changed its position regarding GIF. If you are in a country where Unisys holds a patent on LZW compression (Canada, France, Germany, Italy, Japan, UK, USA) Unisys may require you to license that technology. Therefore, GIF support may be removed completely in a future version of FlagShip. We recommend using the .png or .jpg format.

oPushButton:Bitmap Acc/Ass is a shortcut for `oPushButton:SetImage(cFile-Name, .T., .T.)` which crops the image to default or given size.

oPushButton:Buffer —> lValue *ACCESS*

Contains a logical value indicating that the push button has been pushed (.T.) or not (.F.). The .T. state remains permanent as long as the object has focus, as opposite to oPushButton:Modified where the value changes on any push by mouse click and keyboard.

oPushButton:Caption —> cText *ACCESS*
oPushButton:Caption := cText *ASSIGN*

A string representing the text that is displayed in the push button. If the text contains `"&"` in the string, gPushButton creates an automatic accelerator key for the character following the ampersand &. To display the ampersand itself, enter it twice. E.g. `"&You && Me"` will display button labeled "You & Me" and the button gets an automatic accelerator key, Alt-Y. The string returned by access is the plain text only. For the example above, oPushButton:Caption will return "You & Me".

oPushButton:Cargo –> value *ACCESS*
oPushButton:Cargo := value *ASSIGN*

Contains user data of any type, to store information retrieved later in the program. Not used by oPushButton itself.

oPushButton:ClassName( ) –> cName

Returns "PUSHBUTTON" string. You may also use IsObjClass() function which can determine the from PushButton inherited classes.

oPushButton:Col –> nCol *ACCESS*
oPushButton:Col := nCol *ASSIGN*

Contains a numeric value that indicates the screen column (upper left edge) where the push button is displayed. The <nCol> value is either in pixel or col/row coordinates depending on the current state of SET PIXEL on|OFF. On assignment, the push button widget is moved to the new position. It is similar to oPushButton:X() which allows you to specify or get the column value independent on the SET PIXEL state.

oPushButton:ColorSpec –> cColor *ACCESS*
oPushButton:ColorSpec := cColor *ASSIGN*

Contains a character string specifying the color attributes that are used by the oPushButon:Display() or oPushButton:Show() method. Ignored in GUI mode.

oPushButton:CurrentText –> cText *ACCESS*
oPushButton:CurrentText := cText *ASSIGN*

For compatibility to VO only, same as oPushButton:Caption.

oPushButton:Display( ) –> NIL

Display the PushButton, equivalent to oPushButton:Show(). See also oPushButton:Hide()

oPushButton:Enable([expL1]) –> lEnabled

Enable or disable the button availability. An enabled push button (the default) receives keyboard and mouse events; a disabled widget does not. Note that an enabled widget receives keyboard events only when it is in focus. A disabled button is shown grayed out. Argument (optional):

<**expL1**> enable (.T.) or disable (.F.) the button press. If not specified or is not logical, only the current status is returned.

<**lEnabled**> reports the enable status at the time of entering this method, i.e. before the possible status change.

oPushButton:Exec( ) —> lPushed

Display the PushButton widget and execute the default or used handler. Returns logical value signaling push.

oPushButton:ExitState —> iState *ACCESS*
oPushButton:ExitState := iState *ASSIGN*

Contains a numeric value indicating the desired action, or the state when the object was exited and is used in the user-modifiable READ (see <FlagShip_dir>/system/getsys.prg). Applicable only in @...GET PUSHBUT / READ and is same as Get:ExitState. If not in @..GET/READ, the <iState> value is -9.

Val	getexit.fh	Description
0	GE_NOEXIT	No exit attempted, prepare GET for editing
1	GE_UP	Go to previous GET
2	GE_DOWN	Go to next GET
3	GE_TOP	Go to first GET
4	GE_BOTTOM	Go to last GET
5	GE_ENTER	Normal end of GET editing
6	GE_WRITE	Terminate READ, save GET
7	GE_ESCAPE	Terminate READ, do not save GET
7	GE_EXIT	same as GE_ESCAPE
8	GE_WHEN	WHEN clause unsatisfied

oPushButton:FBlock —> cBlock *ACCESS*
oPushButton:FBlock := cBlock *ASSIGN*

Contains an optional code block ("focus block") that, when present, is evaluated each time the PushButton object receives or loses input focus. The code block receives two arguments: lFocusState and the object self. You also may use the PushButton:hasFocus instance to determine if the push button is receiving or losing input focus. A value of true (.T.) indicates that it is receiving input focus; otherwise, a value of false (.F.) indicates that it is losing input focus. See also oPushButton:Notify and oPushButton:SBlock for other call-back interfaces.

oPushButton:Font —> oFont *ACCESS*
oPushButton:Font := oFont *ASSIGN*
oPushButton:Font([oFont]) —> oFont

Set, get or redefine the used push buttons font.

<**oFont**> is the used Font object. If nor specified or is NIL, the default oApplic:Font is used. To set own font, assign new font object to PushButton, or to local var and than the font object to PushButton, e.g.

```
local oPushFont  := Font{"Arial", 12, "B")
oPushButton:Font := oPushFont
```

oPushButton:GuiColor —> cColor *ACCESS*
oPushButton:GuiColor := cColor *ASSIGN*

Set, get or redefine the push buttons color in GUI mode.

<**cColor**> is a string containing foreground and/or background color similar to SET COLOR.

oPushButton:HasFocus —> lFocus *ACCESS*

A logical value that is set to TRUE when the object receive input focus, and is reset to FALSE when the object loses the input focus. See also oPushButton:Buffer and oPushButton:FBlock

oPushButton:Height([expN1], [lPixel]) —> nHeight

Set and/or return the height (y size) of the push button widget. Arguments (optional):

<**expN1**> The height of the widget. If not given or is NIL, the value remain unchanged and only the current size is returned. On assignment, the push button widget is resized accordingly.

<**lPixel**> if true(.T.), the row and column data are in pixel; if false (.F.), data are in row/col coordinates, otherwise the current SET PIXEL is used.

<**nHeight**> The y (row) coordinate of the widget at the time of entering this method, either in pixel or col/row coordinates, depending on <expL2> argument.

oPushButton:Hide( ) —> NIL

Hide the PushButton. See also oPushButton:Show()

oPushButton:HitTest([expN1], [expN2], [lPixel]) —> nMouseStatus

Determining if the mouse cursor is within the region of the screen that the push button occupies. Arguments (optional):

<**expN1**> the current row position of the mouse cursor relative to the user window, passed e.g. as return value from the MROW() function. If not given, FlagShip determines the mouse position automatically.

<**expN2**> the current column position of the mouse cursor relative to the user window, passed e.g. as return value from the MCOL() function. If not given, FlagShip determines the mouse position automatically.

<**lPixel**> if true(.T.), the row and column data are in pixel; if false (.F.), data are in row/col coordinates, otherwise the current SET PIXEL is used.

<**returnN**> : a numeric value that indicates the relationship of the mouse cursor with the push button:

0 The mouse cursor is not within the region of the screen that the push button occupies.

< 0 The mouse cursor is within the region of the screen that the push button occupies. FlagShip generally returns HTCLIENT == -2049 when the mouse was clicked within the push button area. Note: the constants HTNOWHERE (== 0) and HTTOPLEFT ... HTCLIENT (all < 0) are supported as well and are specified in button.fh

oPushButton:KillFocus( ) —> self

Take input focus away from the PushButton. Upon calling this method and the object has a focus, the object redisplays itself and, if present, evaluates the code block assigned to oPushButton:FBlock, then the oPushButton:Buffer is set .F. See also oPushButton:SetFocus()

oPushButton:Message —> cMessage *ACCESS*
oPushButton:Message := cMessage *ASSIGN*

Contains a string that display short help in the window status line.

oPushButton:Modified —> lClick *ACCESS*

A logical value that is set to TRUE when the user clicks on a button, and reset to FALSE when the mouse button is released. See also oPushButton:Buffer

oPushButton:Notify —> cBlock *ACCESS*
oPushButton:Notify := cBlock *ASSIGN*

Contains an optional code block that, when present, is evaluated each time the PushButton is pressed to enable the application to react on the button press. The code block takes one argument, the object self. See also oPushButton:FBlock and oPushButton:SBlock for other call-back interfaces.

oPushButton:OnClickAction —> num *ACCESS*
oPushButton:OnClickAction := num *ASSIGN*

Contains either NIL or numeric value specifying next READ action (considered in getsys.prg handler). This request is usually set in get:Notify (or other) code block, and is same as oPushButton:ExitState property.

See also oPushButton:OnClickKeys

oPushButton:OnClickKeys –> cKeys *ACCESS*
oPushButton:OnClickKeys := cKeys *ASSIGN*

Contains either NIL or a string comparable to KEYBOARD, which keys are evaluated after exit from push:Notify (or other) code block. You may set in the code block e.g. obj:OnClickKeys := chr(K_UP, K_UP) to skip two fields up when this field is clicked. Considered in getsys.prg READ handler.

oPushButton:Origin –> oPoint *ACCESS*
oPushButton:Origin := oPoint *ASSIGN*

Contains the top left coordinate as Point object. Supported for VO compatibility only. Don't use for a new development, use oPushButton:X() and oPushButton:Y() instead.

oPushButton:Resize([expN1], [expN2], [lPixel]) –> self

Resize the push button widget to a new size. Arguments:

<**expN1**> the new size in rows (height, also decimal fraction) or a vertical size of the application window in pixel (depending on the current SET PIXEL setting and the <lPixel> value). If not given, the current height remains unchanged.

<**expN2**> the new size in columns (width, also decimal fraction) or a horizontal size of the application window in pixel (depending on the current SET PIXEL setting and the <lPixel> value). If not given, the current width remains unchanged.

<**lPixel**> if true(.T.), the row and column data are in pixel; if false (.F.), data are in row/col coordinates, otherwise the current SET PIXEL is used.

oPushButton:Row –> nRow *ACCESS*
oPushButton:Row := nRow *ASSIGN*

Contains a numeric value that indicates the screen row (upper left edge) where the push button is displayed. The <nRow> value is either in pixel or col/row coordinates depending on the current state of SET PIXEL on|OFF. On assignment, the push button widget is moved to the new position. Mainly for Clipper compatibility, for a new development preferably use oPushButton:Y() which allows you to specify or get the row value independent on the SET PIXEL state.

oPushButton:SBlock –> cBlock *ACCESS*
oPushButton:SBlock := cBlock *ASSIGN*

Contains an optional code block ("state block") that, when present, is evaluated each time the PushButton object state (i.e. press/release) changes. The code block receives two arguments: a logical argument which if .T. when the button is pressed, and .F. when released. The second argument is the object self. You also may check by oPushButton:Buffer whether the button was already pressed during this focus period. See also oPushButton:Notify (which is evaluated only on button push) and oPushButton:FBlock (which is evaluated on focus change) for other call-back interfaces.

oPushButton:Select([expN1]) —> self

Activates the Push button object, i.e. simulates the button press. Argument (optional):

<**expN1**> is a numeric value that indicates the key (as inkey() value) that triggered the push buttons activation. If passed, Select() waits for the key specified by <expN1> to be released before continuing.

When this method activated, it performs several operations: First, :Buffer is set to true (.T.). Then, it calls :Display() to show the button in its highlighted color or in the pushed GUI style. If <expN1> is passed, it waits for the key specified by <expN1> to be released. Then, if present, it evaluates its :SBlock code block.

A push buttons state is typically changed when the space bar or enter key is pressed or the mouse's left button is pressed when its cursor is within the push buttons region. Calling this method is meaningful only when the PushButton object has input focus, and is ignored otherwise.

oPushButton:SetFocus( ) —> self

Gives input focus to the PushButton object. Upon calling this method, and the object has not a focus yet, the object redisplays itself, sets the oPushButton:Buffer to .F. and, if present, evaluates the code block assigned to oPushButton:FBlock. See also oPushButton:KillFocus()

oPushButton:SetImage([cFile], [lFrame], [lCrop], [lFromVar]) —> self

Retrieve or assign an image to PushButton object.

<**cFile**> is the image file name. FS_SET() and SET PATH are considered. If not given, the method return the currently used image file name or NIL if none. All common image formats (.gif, .png, .bmp, .jpeg (.jpg), .xbm, .xpm and .pnm) are accepted. When <lFromVar> is set .T., the <cFile> is a character string containing the image self.

<**lFrame**> is optional logical value, specifying if the button frame should be drawn. The default is .T. which draw the frame.

<**lCrop**> is optional logical value, specifying whether the image should be scaled or cropped. The default is .F. which scales the image to given button height and/or width. For best scaling, to calculate the button size automatically from the image size, either set only button height or width, or reset one of them to 0 by pb:Hight(0) or pb:Width(0) - but before invoking the oPushButton:SetImage() method.

<**lFromVar**> is optional logical value, specifying whether image file should be read from disk file (.F.) or from character variable (.T.), default is .F.

Example:

```
oBut1 := PushButton(10,5, , , {|obj| alert("button 1 pressed")} )
oBut1:SetImage("myimage.jpeg") // height = 1 row, width = auto
oBut1:Display()
oBut2 := PushButton(12,5, , , {|obj| alert("button 2 pressed")} )
oBut2:Width(50,.T.)            // width = 50 pixel, height = auto
cImg := memoread("otherimage.png")
oBut2:SetImage(cImg, , ,.T.)
oBut2:Display()
```

oPushButton:SetStyle([expN1], [expL2]) —> nStyle

Set and/or return the style of the Push button widget. Arguments:

<**expN1**> A style constant applicable for the push button, see BS_* BUT_* and SETSTYLE_* constants in the button.fh file

<**expL2**> True (.T.) enables the specified style; False (.F.) disables it. If omitted, the default is True.

In GUI mode, only

```
oPushButton:SetStyle(SETSTYLE_DEFAULT)              = raised button
oPushButton:SetStyle(BUT_AUTOBORDER, .T.)           = raised button
oPushButton:SetStyle(SETSTYLE_FLAT) or ...(BS_FLAT) = no border
oPushButton:SetStyle(BUT_AUTOBORDER, .F.)           = no border
```

is considered.

oPushButton:Show( ) —> NIL

Display the PushButton, equivalent to oPushButton:Display(). See also oPushButton:Hide()

oPushButton:Size —> oSize *ACCESS*
oPushButton:Size := oSize *ASSIGN*

Contains the push button size as Dimension (or Size) object. Supported for VO compatibility only. Don't use for a new development, use oPushButton:Height() and oPushButton:Width() instead.

oPushButton:SizeX —> nCol *ACCESS*
oPushButton:SizeX := nCol *ASSIGN*

Contains a numeric value that indicates the horizontal size (width, columns) of the push button. The <nCol> value is either in pixel or col/row coordinates depending on the current state of SET PIXEL on|OFF. See also oPushButton:Width() which allows you to specify or receive the column value type independent on the SET PIXEL state.

oPushButton:SizeY –> nRow *ACCESS*
oPushButton:SizeY := nRow *ASSIGN*

Contains a numeric value that indicates the vertical size (height, rows) of the push button. The <nRow> value is either in pixel or col/row coordinates depending on the current state of SET PIXEL on|OFF. See also oPushButton:Height() which allows you to specify or receive the size value independent on the SET PIXEL state.

oPushButton:Style –> cStyle *ACCESS*
oPushButton:Style := cStyle *ASSIGN*

For terminal mode only: button is drawn as <text>, or by single or double line characters. Ignored in GUI mode, where oPushButton:SetStyle can be used.

oPushButton:ToolTip –> cTip *ACCESS*
oPushButton:ToolTip := cTip *ASSIGN*

Set or retrieve the ToolTip string. A Tool tip is a short, one-line text reminding the user of the push button widget. Apply for GUI mode only, ignored otherwise.

oPushButton:Visible –> lVisible *ACCESS*

Reports the push button visibility. It returns FALSE (.F.) only when the widget is hidden (invisible), see oPushButton:Hide(.T.). Otherwise, TRUE (.T.) is returned even if the button is not accessible, see oPushButton:Enable(.F.).

oPushButton:Width([expN1], [lPixel]) –> nWidth

Set and/or return the width (x size) of the push button widget.

<**expN1**> The width of the widget. If not given or is NIL, the value remain unchanged and only the current size is returned. On assignment, the push button widget is resized accordingly.

<**lPixel**> if true(.T.), the row and column data are in pixel; if false (.F.), data are in row/col coordinates, otherwise the current SET PIXEL is used.

<**nWidth**> The width (x size) coordinate of the widget at the time of entering this method, either in pixel or col/row coordinates, depending on <lPixel> argument.

oPushButton:X([expN1], [lPixel]) –> nColumn

Set and/or return the x (column) coordinate of the Push button widget.

<**expN1**> The x (column) coordinate of the widget. If not given or is NIL, the X value remain unchanged and only the current size is returned. On assignment, the push button widget is moved to the new position.

<**lPixel**> if true(.T.), the row and column data are in pixel; if false (.F.), data are in row/col coordinates, otherwise the current SET PIXEL is used.

<**nColumn**> The x (column) coordinate of the widget at the time of entering this method, either in pixel or col/row coordinates, depending on <lPixel> argument.

oPushButton:Y([expN1], [lPixel]) –> nRow

Set and/or return the y (row) coordinate of the Push button widget.

<**expN1**> The y (row) coordinate of the widget. If not given or is NIL, the Y value remain unchanged and only the current size is returned. On assignment, the push button widget is moved to the new position.

<**lPixel**> if true(.T.), the row and column data are in pixel; if false (.F.), data are in row/col coordinates, otherwise the current SET PIXEL is used.

<**nRow**> The y (row) coordinate of the widget at the time of entering this method, either in pixel or col/row coordinates, depending on <lPixel> argument.

RadioButton Class

Creates radio button, which is a widget (controls) that can be toggled ON or OFF by a user. A radio button is said to be "pressed" or "selected" when it is filled in, and the RadioButton: Pressed (as well as :Button and :Value) access is TRUE. The radio button object is usually not handled by its own, but in a group using the **RadioGroup** class.

Radio buttons are typically presented in related groups (see also the RadioGroup Class) and provide mutually exclusive responses to a condition where only one choice is appropriate. (For example, a group of radio buttons might allow you to choose Inches, Centimeters, Pixel or Picas for formatting, or Male/Female for gender, etc.)

Only one radio button can be on in each radio button group. When a different button is pressed, the previously selected button is turned off.

You can create a group of radio buttons by using RadioGroup object. FlagShip also support the use of RadioGroups via the common @..GET / READ interface.

Example 1: This example creates two radio buttons, one with a caption of "Male" and the other "Female" and groups them together using the RadioGroup class:

```
LOCAL oRadio1, oRadio2, oRgroup AS object
oRadio1 := RadioButton{10,5,"Male"}
oRadio1:CapCol := 9
oRadio1:CapRow := 10
oRadio2 := RadioButton(11,5)
oRadio2:Caption := "Female"
oRadio2:CapCol := 9
oRadio2:CapRow := 11

oRgroup := RadioGroup(9,3,12,16)
oRgroup:AddItem(oRadio1)
oRgroup:AddItem(oRadio2)
oRgroup:Show()
? "you are", if(oRadio1:Buffer, "male", ;
  if(oRadio2:Buffer, "female", "of unknown gender") )
```

Example 2: This example creates and integrates a radio button group within a GetList and activates it by performing a READ. The selected radio button is returned in the nGender variable.

```
   LOCAL cName := SPACE(20), cFirst := space(20)
   LOCAL nGender:= 1, aGender := array(2)
   SET COLOR TO "GR+/B,N/W,GR+/B,GR+/B,W+/G,R+/B"
   CLS
   aGender[1] := RadioButton{6,42,"&Male"}
// aGender[1]:ColorSpec := "GR+/B,W+/B,G+/B,R+/B,GR+/B,R+/B,R+/B"

   aGender[2] := RadioButton{7,42,"&Female"}

   @ if(AppIoMode() == "G", 4.5,4), 41 SAY "Gender"
   @ 5,10 SAY "Name " GET cName
   @ 5.5,40,8,53 GET nGender RADIOGROUP aGender ;
                 COLOR "W+/B,GR+/B,G+/B,R+/B,GR+/B,R+/B,R+/B"
   @ 7,10 SAY "First" GET cFirst
   READ
   @ 9,16 SAY if(nGender == 1, "Mr. ", "Mrs. ") + ;
              trim(cFirst) + " " + trim(cName)
   wait
```

As with other GUI classes in FlagShip, the general RadioButton class is internally inherited by three different sub-classes: _gRadioButton for GUI based application, _tRadioButton for terminal/text based mode, and _bRadioButton for basic i/o mode, all defined in the boxclass.fh header file. The proper class, corresponding to the used i/o mode, is set either at compile time with the compiler switch "-io=g|t|b", or latest at run-time depending on the currently used environment.

Note: in the basic i/o mode, only a rough radio button functionality is simulated by the sequential in/output.

RadioButton Class Index

Class RadioButton

Inherits from:	- (none)
Inherited by:	RadioGroup
Class prototype:	boxclass.fh
Defines:	button.fh, set.fh

Bitmaps	ACC/ASS	Available for compatibility to Clipper only
Buffer	ACC	Indicates whether the button is checked or not
CapCol	ACC/ASS	Screen column of the radio button caption
CapCol()	METHOD	Screen column of the radio button caption
CapRow	ACC/ASS	Screen row of the radio button caption
CapRow()	METHOD	Screen row of the radio button caption
Caption	ACC/ASS	String that describes the button caption
Cargo	ACC/ASS	A user value of any type
Col	ACC/ASS	Screen column where the radio button is displayed
Col()	METHOD	Screen column where the radio button is displayed
ColorSpec	ACC/ASS	Color attributes
Data	ACC/ASS	Returned value instead of relative position
Destroy()	METHOD	Destroys the RadioButton object
Display()	METHOD	Show the radio button and its caption on the screen
Enabled	ACC/ASS	Indicates whether radio button is selectable
Fblock	ACC/ASS	Code block evaluated at receiving/losing focus
HasFocus	ACC	Indicates whether the object has input focus
Height	ACC/ASS	The height of the radio button
Height()	METHOD	The height of the radio button
HitTest()	METHOD	Determines if the mouse cursor is within the button
IsAccel()	METHOD	Determines whether a key press is button's hot key
KillFocus()	METHOD	Take input focus away from a radio button object
Message	ACC/ASS	String displayed in the windows status bar
Pressed	ACC/ASS	Indicates whether the button is selected
Row	ACC/ASS	Screen row where the radio button is displayed
Row()	METHOD	Screen row where the radio button is displayed
Sblock	ACC/ASS	Code block evaluated at user selection
Select()	METHOD	Set/clear the buttons selected status
SetFocus()	METHOD	Set input focus to a radio button object
Show()	METHOD	Activates the default or user's input handler
Style	ACC/ASS	Delimiter and status display characters
ToolTip	ACC/ASS	Short pop-up info message
TypeOut	ACC	Always .F.
Value	ACC/ASS	Indicates whether the button is selected or not
Width	ACC/ASS	The width of the radio button
Width()	METHOD	The width of the radio button

RadioButton Class Instantiation

oRadBut := [_g|_t|_b]RadioButton { [nR],[nC],[cText],[uData],[lPix] } *[1]*
oRadBut := [_g|_t|_b]RadioButtonNew([nR],[nC],[cText],[uData],[lPix]) *[2]*

oRadBut := RadioButton ([nR], [nC], [cText], [uData], [lPixel]) *[3]*
oRadBut := RadioButton { [oOwn], [nId], [oPoint], [oDim], [<cText>] } *[4]*

Any of the above syntax instantiate new radio button object. Syntax [1] and [2] are standard FlagShip and should be preferred. Syntax [3] is supported for compatibility to Clipper 5.3, and [4] is supported for compatibility to VO.

The widget (control) remains invisible until you invoke oRadBut:Show() or oRadBut:Display(). This allows the program to set up the control correctly (with the correct size, position, and any other parameters), while avoiding the "visual noise" of changing controls.

<**nR**> row in coordinates or pixel, optional. If not specified, 0 is the default. See additional details in the oRadBut:Row description.

<**nC**> column in coordinates or pixel, optional. If not specified, 0 is the default. See additional details in the oRadBut:Col description.

<**cText**> caption text, optional. If not redefined by :CapCol and/or :CapRow, the text is displayed in the <nR> row and <nC> + 4 column.

<**uData**> optional, assigned data of any type, returned by RadioGroup:Value

<**lPixel**> if true(.T.), the row and column data are in pixel; if false (.F.), data are in row/col coordinates, otherwise the current SET PIXEL status is used.

<**oOwn**> owner object of the radio button, optional. Default is the oApplic object.

<**nId**> an unique ID between 1 and 8000 of the radio button, optional. If not specified, internal ID is used.

<**oPoint**> the origin of the radio button, in canvas coordinates

<**oDim**> the dimension of the radio button, in canvas coordinates

Compatibility: Available also in CL53 (syntax 3) and VO (syntax 4). See also: oRadBut:Destroy()

RadioButton Class Properties

oRadBut:Bitmaps —> aFile *ACCESS*
oRadBut:Bitmaps := aFile *ASSIGN*

This property is available for compatibility to Clipper (in semi- graphical mode) only and is not used by FlagShip object.

Compatibility: Available also in CL53.

oRadBut:Buffer —> lChecked *ACCESS*

<**lChecked**> is a logical value that indicates whether the radio button is selected or not. A value of true (.T.) indicates that it is selected and a value of false (.F.) indicates that it is not. Equivalent to oRadBut:Checked instance.

Compatibility: Available also in CL53.

See also: oRadBut:Checked, oRadBut:Select()

oRadBut:CapCol —> nCol *ACCESS*
oRadBut:CapCol := nCol *ASSIGN*
oRadBut:CapCol([nCol], [lPixel]) —> nCol

<**nCol**> is a numeric value that indicates the screen column where the radio button caption is displayed. The input and output value is either in coordinates or in pixels, depending on the current SET PIXEL setting. The default setting is oRadBut:Col + 4 columns at instantiation time.

<**lPixel**> is optional value indicating if the passed and returned value is in coordinates or pixels. If true(.T.), the column data are in pixel; if false (.F.), data are in coordinates, otherwise the current SET PIXEL status is used.

Compatibility: Access/assign is available in CL53.

See also: oRadBut:CapRow, oRadBut:Caption

oRadBut:CapRow —> nRow *ACCESS*
oRadBut:CapRow := nRow *ASSIGN*
oRadBut:CapRow([nRow], [lPixel]) —> nRow

<**nRow**> is a numeric value that indicates the screen row where the radio button caption is displayed. The input and output value is either in coordinates or in pixels, depending on the current SET PIXEL setting. The default setting is taken from oRadBut:Row at instantiation time.

<**lPixel**> is optional value indicating if the set/get value is in coordinates or pixels. If true(.T.), the row data are in pixel; if false (.F.), data are in coordinates, otherwise the current SET PIXEL status is used.

Compatibility: Available also in CL53.

See also: oRadBut:CapCol, oRadBut:Caption

oRadBut:Caption –> cText *ACCESS*
oRadBut:Caption := cText *ASSIGN*

<**cText**> is a string that describes the radio button caption. If not redefined by :CapCol and/or :CapRow, the text is displayed at the :Row and :Col + 4 position set at instantiation time. When present, the & character specifies that the character immediately following it in the caption is the radio button accelerator key. The accelerator key provides a quick and convenient mechanism for the user to move input focus from one data input control to a radio button. The user performs the selection by pressing the Alt key in combination with an accelerator key. The case of an accelerator key is ignored.

Compatibility: Available also in CL53 and VO.

See also: oRadBut:CapCol, oRadBut:Caption

oRadBut:Cargo –> exp *ACCESS*
oRadBut:Cargo := exp *ASSIGN*

<**exp**> is a value of any type. The RadioButton:Cargo slot holds any user-definable data which can be retrieved later. This property is not used by the RadioButton object itself.

Compatibility: Available also in CL53.

oRadBut:Col –> nCol *ACCESS*
oRadBut:Col := nCol *ASSIGN*
oRadBut:Col([nCol], [lPixel]) –> nCol

<**nCol**> is a numeric value that indicates the screen column where the radio button is displayed. With Access/assign, the value is either in coordinates or pixels according to the current SET PIXEL status.

<**lPixel**> is optional value indicating if the passed and returned value is in coordinates or pixels. If true(.T.), the column data are in pixel; if false (.F.), data are in coordinates, otherwise the current SET PIXEL status is used.

With terminal i/o, the <nCol> value specifies the column where the first character of oRadBut:Stype is displayed, i.e. where the left parenthesis (*) of the radio button representation display. The whole radio button occupy 3 columns.

With GUI i/o, the radio button is displayed as a widget (control) and <nCol> is the leftmost widget coordinate. To ensure the same look and feel to an application running in textual mode, and to display the widget at approx. the same screen position, the given <nCol> coordinate is automatically adapted by adding a pixel value taken from the global array elements `_aGlobSetting[GSET_G_N_RADBUT_COL]` and `_aGlobSetting [GSET_G_N_ RADBUT_WIDTH]` which may be positive or negative and are user modifiable.

Compatibility: Access/assign is available in CL53.

See also: oRadBut:Row, RadioButton{} instantiation

oRadBut:ColorSpec —> cAttrib *ACCESS*
oRadBut:ColorSpec := cAttrib *ASSIGN*

<**cAttrib**> is a character string specifying the color attributes that are used by the display() and show() method. May be also set or redefined by RadioGroup: SetColor(). The string must contain eight color specifiers, otherwise the rest remain unchanged.

Position in <cAttrib>	Applies To	Default value used from curr SET COLOR
1	Radio button without input focus, unselected	Unselected
2	Radio button without input focus, selected	Unselected
3	Radio button with input focus, unselected	Enhanced
4	Radio button with input focus, selected	Enhanced
5	Radio button's caption	Standard
6	Radio button caption's accel. key w/o focus	Standard
7	Radio button caption's accel. key with focus	Background
8	Radio button and caption, disabled	Border

Specifying "-" for foreground or background lets the original color unchanged, which enables you to change the required color attribute only.

Compatibility: Available also in CL53, which support seven attributes. This property is considered in terminal mode only, and ignored in GUI mode.

See also: oRadBut:HasFocus, :Enabled, SET COLOR, SET()

oRadBut:Data —> exp *ACCESS*
oRadBut:Data := exp *ASSIGN*

<**exp**> is a value of any type. If specified, the RadioGroup:Value returns this value instead of the relative position of selected radio button.

Compatibility: Available but undocumented in CL53.

See also: RadioGroup:Value

oRadBut:Destroy() —> NIL

Destroys the RadioButton object and restores the previous screen content. This method can be used when a RadioButton object is no longer needed. oRadBut:Destroy() de-instantiates the RadioButton object and allows you to close and free any resources that were opened or created by the object, without waiting for the garbage collector. This method calls internally oRadBut:Axit() which is the equivalence for :Destroy()

Compatibility: Available also in VO

See also: RadioButton{} instantiation

oRadBut:Display() —> self

Show the radio button, it frame and caption on the screen. The radio button widget (control) remains invisible until you invoke :Display() or oRadBut:Show(). This allows the program to set up the control correctly (with the correct size, position, and any other parameters), while avoiding the "visual noise" of changing controls.

oRadBut:Display() uses the values of the following instance variables to correctly show the list in its current context, in addition to providing maximum flexibility in the manner a radio button appears on the screen: Buffer, Caption, CapCol, CapRow, Col, ColorSpec, HasFocus, Row, and Style.

Compatibility: Available also in CL53

See also: oRadBut:Show()

oRadBut:Enabled —> lOk *ACCESS*
oRadBut:Enabled := lOk *ASSIGN*

<**lOk**> contains TRUE (.T.) if the radio button is selectable by user, and FALSE (.F) if it is not. The default is TRUE.

Compatibility: Available also in FS5 only

See also: oRadBut:ColorSpec

oRadBut:Fblock —> bBlock *ACCESS*
oRadBut:Fblock := bBlock *ASSIGN*

<**bBlock**> is a code block or NIL. The code block callback, when present, is evaluated each time the RadioButton object receives or loses input focus. The code block receives two arguments: the object self and the current :HasFocus status, which indicates whether the radio button is receiving (.T.) or losing (.F.) input focus. In GUI, the object receives focus every times the user clicks (or activates) the radio button widget and looses focus when other widget is selected.

Compatibility: Available also in CL53, but Clipper does not pass any arguments to the code block, and hence cannot use generalized but object specific code blocks which needs to check the current oRadBut:HasFocus status by itself.

See also: oRadBut:HasFocus, :SetFocus(), :KillFocus(), :Sblock

oRadBut:HasFocus —> lFocus *ACCESS*

<**lFocus**> is a logical value indicating whether the object has input focus (TRUE) or not. In GUI, the object receives focus every times the user clicks (or activates) the widget and looses the focus when other widget is selected.

Compatibility: Available also in CL53

See also: oRadBut:KillFocus, :SetFocus(), :Fblock

oRadBut:Height —> nRow *ACCESS*
oRadBut:Height := nRow *ASSIGN*
oRadBut:Height ([nRow], [lPixel]) —> nRow

<**nCol**> is a numeric value that indicates the height of the radio button. With Access and assign, the value is either in coordinates or pixels according to the current SET PIXEL status.

<**lPixel**> is optional value indicating if the passed and returned value is in coordinates or pixels. If true(.T.), the column data are in pixel; if false (.F.), data are in coordinates, otherwise the current SET PIXEL status is used.

Compatibility: Available also in FS5, apply for GUI mode only

oRadBut:HitTest(nMouseRow, nMouseCol, [lPixel]) —> nStatus

Determines if the mouse cursor is within the region of the screen that the radio button occupies.

<**nRow**> Numeric value representing the current or tested screen row position of the mouse cursor.

<**nCol**> Numeric value representing the current or tested screen row position of the mouse cursor.

<**lPixel**> If specified TRUE, the mouse coordinates are assumed in pixel. If FALSE, the mouse parameters are assumed in current row/col coordinates. If this parameter is not specified (i.e. NIL), the kind of passed mouse coordinates is determined from the current SET PIXEL setting.

<**nStatus**> Returned numeric value indicating the relationship of the mouse cursor with the radio button. The constants are specified in button.fh header file.

Value	Constant	Description
0	HTNOWHERE	The mouse is not located in the button region
-1025	HTCAPTION	The mouse cursor is on the button's caption
-2049	HTCLIENT	The mouse cursor is on the radio button

Compatibility: Available also in CL53

See also: Mrow(), Mcol()

oRadBut:Init([par1]...[par5]) —> self

This is an internal method invoked automatically at instantiation of the RadioButton object. It is not intended to be called by the application.

Compatibility: Available also in VO

See also: RadioButton{} instantiation

oRadBut:IsAccel(nKey) —> lOk

Determines whether a key press should be interpreted as a user request to select a radio button. Returns a logical value <lOk> that indicates whether the value specified by <nKey> should be treated as a hot key. A value of true (.T.) indicates that the key should be treated as a hot key; otherwise, a value of false (.F.) indicates that it should not. This is an internal method invoked automatically at instantiation of the RadioButton object. It is not intended to be called by the application.

Compatibility: Available also in CL53

See also: oRadBut:Caption

oRadBut:KillFocus() —> self

Take input focus away from a RadioButton object. Upon receiving this message, the RadioButton object redisplays itself and, if present, evaluates the code block specified by :Fblock. This message is meaningful only when the RadioButton object has input focus.

Compatibility: Available also in CL53. In Clipper

See also: oRadBut:HasFocus, :SetFocus(), :Fblock

oRadBut:Message —> cText *ACCESS*
oRadBut:Message := cText *ASSIGN*

<**cText**> is a character string displayed in the windows status bar (GUI), or in the screen line specified by SET MESSAGE (in terminal mode).

Compatibility: Available also in FS5 only

See also: oRadBut:ToolTip(), SET MESSAGE, oApplic:StatusMessage()

oRadBut:Pressed —> lOk *ACCESS*
oRadBut:Pressed := lOk *ASSIGN*

<**lOk**> contains TRUE (.T.) if the radio button is in the selected (ON) state, and FALSE (.F) if it is in the unselected state (OFF). Equivalent to oRadBut:Buffer and oRadBut:Pressed

Compatibility: Available also in VO

See also: oRadBut:Buffer, oRadBut:Value

oRadBut:Row —> nRow *ACCESS*
oRadBut:Row := nRow *ASSIGN*
oRadBut:Row([nRow], [lPixel]) —> nRow

<**nRow**> is a numeric value that indicates the screen row where the radio button is displayed. With Access/assign, the value is either in coordinates or pixels according to the current SET PIXEL status.

<**lPixel**> is optional value indicating if the set/get value is in coordinates or pixels. If true(.T.), the row data are in pixel; if false (.F.), data are in coordinates, otherwise the current SET PIXEL status is used.

With terminal i/o, the <nRow> value specifies the column where the three characters of radio button (*) display.

In GUI i/o mode, the radio button is displayed as a widget (control) and <nRow > is the topmost widget coordinate when the row is specified in pixel. If the <nRow > is given in coordinates, the widget position is automatically adapted, to ensure the same look and feel to an application running in textual mode, and to display the widget at approx. the same screen position. The topmost widget position is then calculated from the given <nRow> coordinate minus the current line height plus a value taken from the global array elements `_aGlobSetting[GSET_G_N_RADBUT_ROW]` and `_aGlobSetting [GSET_G_N_RADBUT_HEIGHT]` which is either positive or negative number of pixels and are user modifiable.

Compatibility: Access/assign is available in CL53.

See also: oRadBut:Col, RadioButton{} instantiation

oRadBut:Sblock —> bBlock *ACCESS*
oRadBut:Sblock := bBlock *ASSIGN*

<**bBlock**> is an optional code block or NIL. The code block callback, when present, is evaluated each time the RadioButton object's state changes. The name "Sblock" refers to state block. The code block receives two arguments: 1) the object self, and 2) the select status, i.e. the content of oRadBut:Buffer.

Compatibility: Available also in CL53, but Clipper does not pass any arguments to the code block; it hence cannot use generalized but object specific code blocks which must extract the required values from the known object by itself.

See also: oRadBut:Buffer, :Fblock

oRadBut:Select([lOnOff]) —> lOnOff

<**lOnOff**> is a logical value that indicates whether the radio button should be selected or not. Set to true (.T.) to select, or false (.F.) to deselect the button. If omitted, the radio button state will toggle to its opposing state. Considered only if the button has input focus, or when the radio button is a member of a RadioGroup object that has input focus.

The radio button state is typically changed when the space bar is pressed or the mouse's left button is pressed when its cursor is within the radio button's region of the screen. FlagShip's default handler used in oRadioGroup:Show() also accepts `+,T,t,Y,y` keys to set the status ON, and `-,F,f,N,n` keys to set the radio button status OFF, and space or "x" key to toggle the status.

Compatibility: Available also in CL53

See also: oRadBut:Buffer

oRadBut:SetFocus() —> self

Set input focus to a RadioButton object. Upon receiving this message, the RadioButton object redisplays itself and, if present, evaluates the code block specified by :Fblock. This message is meaningful only when the RadioButton object does not have input focus. In GUI, the object receives focus also every times the user clicks (or activates) the widget.

Compatibility: Available also in CL53.

See also: oRadBut:HasFocus, :KillFocus(), :Fblock, :HotBox

oRadBut:Show() —> self

Provided for compatibility to VO, performs the same action as :Display()

Compatibility: Available also in VO

See also: oRadBut:Display(), oRadBut:Handler

oRadBut:Style —> cStyle *ACCESS*
oRadBut:Style := cStyle *ASSIGN*

<**cStyle**> is a character string that indicates the delimiter characters that are used by the radio button's Display() and Show() method. The string must contain four

characters. The first is the left delimiter, the 2nd is the "selected" indicator, the 3rd is the "unselected" indicator, and the 4th character is the right delimiter. The default style is pre-defined in the global array element `_aGlobSetting [GSET_T_C_RADBUT_STYLE]` containing "(*)" at start-up; it may be re-defined by a simple assignment later. May be also set or redefined by RadioGroup:SetStyle().

Compatibility: Considered in terminal mode only, ignored in GUI. Available also in CL53.

See also: RadioGroup:SetStyle(), oRadBut:ColdBox, :HotBox, :Display()

oRadBut:ToolTip —> cText *ACCESS*
oRadBut:ToolTip := cText *ASSIGN*

<**cText**> is a string representing the displayed tool tip, i.e. a short info message which pop up's when the mouse is over the radio button.

Compatibility: Available in FS5 only, apply for GUI, ignored otherwise

See also: oRadBut:Message

oRadBut:TypeOut —> lVal *ACCESS*

<**lVal**> is a value always containing false (.F.). It is not used by the RadioButton object and is only provided for compatibility with the other GUI control classes.

Compatibility: Available also in FS5

oRadBut:Value —> exp *ACCESS*
oRadBut:Value := exp *ASSIGN*

<**exp**> contains TRUE (.T.) if the radio button is in the selected (ON) state, and FALSE (.F) if it is in the unselected state (OFF). Equivalent to oRadBut:Buffer

Compatibility: Available also in VO

See also: oRadBut:Buffer, oRadBut:Pressed

oRadBut:Width —> nCol *ACCESS*
oRadBut:Width := nCol *ASSIGN*
oRadBut:Width ([nCol], [lPixel]) ? nCol

<**nCol**> is a numeric value that indicates the width of the radio button. With Access and assign, the value is either in coordinates or pixels according to the current SET PIXEL status.

<**lPixel**> is optional value indicating if the passed and returned value is in coordinates or pixels. If true(.T.), the column data are in pixel; if false (.F.), data are in coordinates, otherwise the current SET PIXEL status is used.

Compatibility: Available also in FS5, apply for GUI mode only

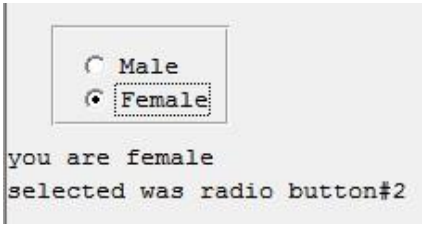
RadioGroup Class

Creates radio button group which provides a convenient mechanism for manipulating radio buttons, added in the radio group via AddItem() or InsItem() method.

Radio buttons are typically presented in related groups (i.e. using this RadioGroup Class) and provide mutually exclusive responses to a condition where only one choice is appropriate. Only one radio button can be ON in each radio button group. When a different button is pressed, the previously selected button is turned off.

Example 1: This example creates two radio buttons, one with a caption of "Male" and the other "Female" and groups them together using the RadioGroup class and uses the standard input handler via :Show()

```
oRadio1 := RadioButton{10,5,"Male"}
oRadio2 := RadioButton(11,5)
oRadio2:Caption := "Female"
oRgroup := RadioGroup(9,3,12,15)
oRgroup:AddItem(oRadio1)
oRgroup:AddItem(oRadio2)
oRgroup:ColorSpec := "W+/B,GR+/B,G+/B,R+/B,GR+/B,R+/B,R+/B"
oRgroup:Show()    // process automatically
setpos(12,0)
? "you are", if(oRadio1:Buffer, "male", ;
  if(oRadio2:Buffer, "female", "of unknown gender") )
? "selected was radio button#" + ltrim( oRgroup:Value )
```



Example 2: This is very similar to above example but handles the input by its own and uses Radio button :Data property:

```
#include "inkey.fh"
#include "box.fh"
oRadio1 := RadioButton{10,5,"Male","Mr."}
oRadio2 := RadioButton(11,5)
oRadio2:Caption := "Female"
oRadio2:Data := "Mrs."
oRgroup := RadioGroup(9,3,12,15)
oRgroup:AddItem(oRadio1)
oRgroup:AddItem(oRadio2)
oRgroup:ColdBox := B_PLAIN
oRgroup:HotBox  := B_PLAIN
```

```
// handle this radio group manually similarly to Clipper
SET WRAP ON // wrap from last to first selection
oRgroup:Display()
oRgroup:SetFocus()
oRgroup:FirstItem():Select(.T.)  // select first item
while .T.
   key := inkey()
   do case
   case key == K_DOWN
      oRgroup:NextItem()
   case key == K_UP
      oRgroup:PrevItem()
   case key == K_ESCAPE
      exit
   case key == K_CTRL_UP .or. key == K_HOME
      oRgroup:FirstItem()
   case key == K_CTRL_DOWN .or. key == K_END
      oRgroup:LastItem()
   case chr(key) $ " X"
      oRgroup:Select()     // toggle on/off
      exit
   case chr(key) $ "+yYtT"  .or. key == K_ENTER
      oRgroup:Select(NIL, .T.)
      exit
   case chr(key) $ "-nNfF"
            oRgroup:Select(.F.)
            exit
   endcase
enddo
oRgroup:KillFocus()
cSalutation := oRgroup:Value()   // Mr. or Mrs.
```

Example 3: See additional examples, e.g. the use via @..GET / READ in the RadioButton class description.

As with other GUI classes in FlagShip, the general RadioGroup class is internally inherited by three different sub-classes: _gRadioGroup for GUI based application, _tRadioGroup for terminal/text based mode, and _bRadioGroup for basic i/o mode, all defined in the boxclass.fh header file. The proper class, corresponding to the used i/o mode, is set either at compile time with the compiler switch "-io=g|t|b", or latest at run-time depending on the currently used environment.

Note: in the basic i/o mode, only a rough radio button functionality is simulated by the sequential in/output.

RadioGroup Class Index

Class RadioGroup
Inherits from: - (none)
Inherited by: - (none)
Class prototype: boxclass.fh
Defines: button.fh, set.fh

AddItem()	METHOD	Add new RadioButton item at the end of radio group list
Bottom	ACC/ASS	Screen bottom row of the radio group frame
Bottom()	METHOD	Screen bottom row of the radio group frame
Buffer	ACC	Position of the selected radio button in the group list
Button()	METHOD	Get the specified radio button object
CapCol	ACC/ASS	Screen column of the radio group caption
CapCol()	METHOD	Screen column of the radio group caption
CapRow	ACC/ASS	Screen row of the radio group caption
CapRow()	METHOD	Screen row of the radio group caption
Caption	ACC/ASS	String that describes the radio group caption
Cargo	ACC/ASS	A user value of any type
ClassName()	METHOD	For compatibility to Clipper's getsys.prg only
ColdBox	ACC/ASS	Frame of the radio group without focus
ColorSpec	ACC/ASS	Color attributes
CurrItemNo	ACC/ASS	Position of the selected radio button in the group list
Destroy()	METHOD	Destroys the RadioGroup object
DelItem()	METHOD	Remove specified item from the radio group list
Display()	METHOD	Show radio buttons, frame and caption on the screen
Exec()	METHOD	Process user input, same as :Show()
Fblock	ACC/ASS	Code block evaluated at receiving/losing focus
FirstItem()	METHOD	Selects the first selectable item in the group list
FrameStyle	ACC/ASS	Set kind of GUI frame around the radio group
GetAccel()	METHOD	Get the item position corresponding to the given key
GetItem()	METHOD	Get the specified radio button object
Handler	ACC/ASS	User defined keyboard handler
HasFocus	ACC	Indicates whether the object has input focus
Height	ACC/ASS	The height of the radio group widget
Height()	METHOD	The height of the radio group widget
HitTest()	METHOD	Determines if the mouse cursor is within the widget
HotBox	ACC/ASS	Frame of the radio group with focus
InsItem()	METHOD	Insert new RadioButton item at specified position
ItemCount	ACC	Total number of radio buttons in the RadioGroup list
KillFocus()	METHOD	Take input focus away from a CheckBox object
LastItem()	METHOD	Selects the last selectable item in the group list
Left	ACC/ASS	Leftmost screen column of the radio group frame
Left()	METHOD	Leftmost screen column of the radio group frame
Message	ACC/ASS	String displayed in the windows status bar
Modified	ACC/ASS	Indicates that the user clicks on a radio button
NextItem()	METHOD	Selects the next selectable item in the group list

PrevItem()	METHOD	Selects the previous selectable item in the group list
Right	ACC/ASS	Rightmost screen column of the radio group frame
Right()	METHOD	Rightmost screen column of the radio group frame
Sblock	ACC/ASS	Code block evaluated at user selection
Select()	METHOD	Select and set specific radio button on/off
SetColor()	METHOD	Set uniform color attributes for all radio buttons
SetFocus()	METHOD	Set input focus to a radio button object
SetStyle()	METHOD	Set uniform style attributes for all radio buttons
Show()	METHOD	Displays the widget and invoke the keyboard handler
ToolTip	ACC/ASS	Short pop-up info message
Top	ACC/ASS	Screen topmost row of the radio group frame
Top()	METHOD	Screen topmost row of the radio group frame
TypeOut	ACC/ASS	Indicates whether the group contains selectable buttons
Value	ACC/ASS	Relative position of the button toggled ON
Width	ACC/ASS	The width of the radio group widget
Width()	METHOD	The width of the radio group widget

RadioGroup Class Instantiation

oRadGrp := [_g|_t|_b]RadioGroup {[nTop],[nLeft],[nBott],[nRight],[lPix]} ***[1]***
oRadGrp := [_g|_t|_b]RadioGroupNew([nTop],[nLeft],[nBott],[nRight],[lPix]) ***[2]***

oRadGrp := RadioGroup ([nTop], [nLeft], [nBott], [nRight], [lPix]) ***[3]***
oRadGrp := RadioGroup { [oOwn], [nId], [oPoint], [oDim], [cCapt] } ***[4]***

Any of the above syntax instantiate new radio group object. Syntax [1] and [2] are standard FlagShip and should be preferred. Syntax [3] is supported for compatibility to Clipper 5.3, and [4] is supported for compatibility to VO.

The widget (control) remains invisible until you invoke oRadGrp:Show() or oRadGrp:Display(). This allows the program to set up the control correctly (with the correct size, position, and any other parameters), while avoiding the "visual noise" of changing controls.

<**nTop**> topmost row where the frame of radio group display in coordinates or pixel, optional. If not specified, the coordinates are calculated automatically from radio group items at the first oRadGrp:Display(). See additional details in the oRadGrp:Top description.

<**nLeft**> leftmost column where the frame of radio group display in coordinates or pixel, optional. If not specified, the coordinates are calculated automatically from radio group items at the first oRadGrp: Display(). See additional details in the oRadGrp:Left description.

<**nBott**> bottom row where the frame of radio group display in coordinates or pixel, optional. If not specified, the coordinates are calculated automatically from radio group items at the first oRadGrp:Display(). See additional details in the oRadGrp:Bottom description.

<**nRight**> rightmost column where the frame of radio group display in coordinates or pixel, optional. If not specified, the coordinates are calculated automatically from radio group items at the first oRadGrp: Display(). See additional details in the oRadGrp:Right description.

<**lPix**> if true(.T.), the row and column data are in pixel; if false (.F.), data are in row/col coordinates, otherwise the current SET PIXEL status is used.

<**oOwn**> owner object of the radio button, optional. Default is the oApplic object.

<**nId**> an unique ID between 1 and 8000 of the radio button, optional. If not specified, internal ID is used.

<**oPoint**> the origin of the radio button, in canvas coordinates

<**oDim**> the dimension of the radio button, in canvas coordinates

<**cCapt**> caption text, optional. The position of the text is specified by oRadGrp:CapRow and oRadGrp:CapCol

Compatibility: Available also in CL53 (syntax 3) and VO (syntax 4). Neither Clipper nor VO calculates the frame coordinates automatically but requires the input.

See also: oRadGrp:Destroy()

RadioGroup Class Properties

oRadGrp:AddItem(oRadButt) —> self

Add new RadioButton item to radio group list

<**oRadButt**> is the radio button object to be added at the and of the radio group list.

Compatibility: Available also in CL53 and VO.

See also: oRadGrp:InsItem(),oRadGrp:DelItem()

oRadGrp:Bottom —> nRow *ACCESS*
oRadGrp:Bottom := nRow *ASSIGN*
oRadGrp:Bottom ([nRow], [lPixel]) —> nRow

<**nRow**> is a numeric value that indicates the screen bottom row where the cold and hot box frame of the radio group is displayed. The input and output value is either in coordinates or in pixels, depending on the current SET PIXEL setting. The default coordinates are specified at radio group instantiation or are calculated automatically from radio group items at the first oRadGrp:Display() or :Show() invocation.

<**lPixel**> is optional value indicating if the set/get value is in coordinates or pixels. If true(.T.), the row data are in pixel; if false (.F.), data are in coordinates, otherwise the current SET PIXEL status is used.

With terminal i/o, the <nRow> value specifies the row where the frame of oRadGrp:ColdBox and :HotBox is displayed.

With GUI i/o, the radio group is displayed as a widget (control) and <nRow > is the bottom widget coordinate. To ensure the same look and feel to an application running in textual mode, and to display the widget at approx. the same screen position, the given <nRow> coordinate is automatically adapted by adding a pixel value taken from the global array element `_aGlobSetting[GSET_G_N_RADGRP_BOT]` which may be positive or negative. Additional adjustment is possible via oRadGrp:Right and :Height

Compatibility: Available also in CL53 which does not calculate the frame coordinates automatically but requires the input.

See also: oRadGrp:Top, :Left, :Right

oRadGrp:Buffer —> nPos *ACCESS*

<**nPos**> is a numeric value that indicates the position in the radio group of the selected radio button. Equivalent to oRadGrp:Value instance w/o data properties.

Compatibility: Available also in CL53.
See also: oRadGrp:Select()

oRadGrp:Button([nPos]) —> oRadioButton

Fully equivalent to oRadGrp:GetItem([nPos]), available for compatibility purpose.

Compatibility: Available also in VO.

See also: oRadGrp:GetItem (), :FirstItem(), :NextItem(), :LastItem(), :ItemCount

oRadGrp:CapCol —> nCol *ACCESS*
oRadGrp:CapCol := nCol *ASSIGN*
oRadGrp:CapCol([nCol], [lPixel]) —> nCol

<**nCol**> is a numeric value that indicates the screen column where the radio group caption is displayed, the default is 0. The input and output value is either in coordinates or in pixels, depending on the current SET PIXEL setting.

<**lPixel**> is optional value indicating if the passed and returned value is in coordinates or pixels. If true(.T.), the column data are in pixel; if false (.F.), data are in coordinates, otherwise the current SET PIXEL status is used.

Compatibility: Access/assign is available in CL53.

See also: oRadGrp:CapRow, oRadGrp:Caption

oRadGrp:CapRow —> nRow *ACCESS*
oRadGrp:CapRow := nRow *ASSIGN*
oRadGrp:CapRow([nRow], [lPixel]) := nRow

<**nRow**> is a numeric value that indicates the screen row where the radio group caption is displayed, the default is 0. The input and output value is either in coordinates or in pixels, depending on the current SET PIXEL setting.

<**lPixel**> is optional value indicating if the set/get value is in coordinates or pixels. If true(.T.), the row data are in pixel; if false (.F.), data are in coordinates, otherwise the current SET PIXEL status is used.

Compatibility: Available also in CL53.

See also: oRadGrp:CapCol, oRadGrp:Caption

oRadGrp:Caption —> cText *ACCESS*
oRadGrp:Caption := cText *ASSIGN*

<**cText**> is a string that describes the radio group caption. When present, the & character specifies that the character immediately following it in the caption is the radio group accelerator key. The accelerator key provides a quick and convenient mechanism for the user to move input focus from one data input control to a radio group. The user performs the selection by pressing the Alt key

in combination with an accelerator key. The case of an accelerator key is ignored.

Compatibility: Available also in CL53 and VO.

See also: oRadGrp:CapCol, oRadGrp:Caption

oRadGrp:Cargo —> exp *ACCESS*
oRadGrp:Cargo := exp *ASSIGN*

<**exp**> is a value of any type. The oRadGrp:Cargo slot holds any user- definable data which can be retrieved later. This property is not used by the RadioGroup object itself.

Compatibility: Available also in CL53.

oRadGrp:ClassName() —> cText

For compatibility to Clipper's getsys.prg only. Return fix "RADIOGROUP" regardless the subclass. In FlagShip, you may also use IsObjClass() which provides you with more detailed information.

Compatibility: Available but undocumented in CL53

See also: IsObjClass() and IsObjProperty() functions, getsys.prg source

oRadGrp:ColdBox —> cBox *ACCESS*
oRadGrp:ColdBox := cBox *ASSIGN*

<**cBox**> is an optional string that specifies the characters to use when drawing a box around the radio group when it does not have input focus. Its default value is pre-defined in the global array element _aGlobSetting[GSET_T_C_COLDBOX] and is usually B_SINGLE. The following <cBox> constants are defined in the box.fh file, the _aGlobSetting[] array constants in set.fh and initio.prg files.

Constant	Description
B_SINGLE	Single line box
B_DOUBLE	Double line box
B_SINGLE_DOUBLE	Single line top/bottom, double line sides
B_DOUBLE_SINGLE	Double line top/bottom, single line sides
B_PLAIN	Plain ASCII characters

Compatibility: Available also in CL53. This property is considered in terminal mode only and is ignored in GUI mode.

See also: oRadGrp:HotBox,:SetFocus(), :ColorSpec, @..BOX

oRadGrp:ColorSpec —> cAttrib *ACCESS*
oRadGrp:ColorSpec := cAttrib *ASSIGN*

<**cAttrib**> is a character string specifying the color attributes that are used by the Display() and Show() method. The string must contain three color specifiers, otherwise the rest is unchanged.

Position in <cAttrib>	Applies To	Default value used from curr SET COLOR
1	Radio group border	Border
2	Radio group caption	Standard
3	Radio group caption's key	Background

Specifying "-" for foreground or background lets the original color unchanged, which enables you to change the required color attribute only.

Compatibility: Available also in CL53. This property is considered in terminal mode only and is ignored in GUI mode.

See also: oRadGrp:HasFocus, :SetColor(), SET COLOR, SET()

oRadGrp:CurrItemNo —> nPos *ACCESS*
oRadGrp:CurrItemNo := nPos *ASSIGN*

<**nPos**> is a numeric value, between 1 and the :ItemCount, indicating which item is currently selected and is equivalent to :Buffer. If no item is selected, it is 0.The :CurrItemNo assign is equivalent to :GetItem(nPos).

Compatibility: Available in FS5 only

See also: oRadGrp:Buffer, :GetItem()

oRadGrp:Destroy() —> NIL

Destroys the RadioGroup object and restores the previous screen content. This method can be used when a RadioGroup object is no longer needed. oRadGrp:Destroy() de-instantiates the RadioGroup object and allows you to close and free any resources that were opened or created by the object, without waiting for the garbage collector. This method calls internally oRadGrp:Axit() which is the equivalence for :Destroy()

Compatibility: Available also in VO

See also: RadioGroup{} instantiation

oRadGrp:DelItem(nPos) —> self

<**nPos**> is a numeric value that indicates the position in the radio group list of the radio button to be deleted.

Compatibility: Available also in CL53 and VO

See also: oRadGrp:AddItem(), :InsItem(), :ItemCount

oRadGrp:Display() —> self

Show all the radio buttons available in the group list, the radio group frame and caption on the screen. If the radio group coordinates were not specified yet, they are calculated automatically from the radio button list.

Note: the radio group widget (control) remains invisible until you invoke oRadGrp:Display() or oRadGrp:Show(). This allows the program to set up the control correctly (with the correct size, position, and any other parameters), while avoiding the "visual noise" of changing controls.

Compatibility: Available also in CL53, which does not calculate the coordinates automatically bur requires the input.

See also: oRadGrp:Show(), :Top, :Bottom, :Left, Right

oRadGrp:Exec() —> self

This method is equivalent to oRadGrp:Show(). It activates either the default or user specific input handler (specified by :Handler) to process the user entry. See further details in :Show()

Compatibility: Available in FS5 only.

See also: oRadGrp:Show(), :GetItem(), :NextItem(), :LastItem()

oRadGrp:Fblock —> bBlock *ACCESS*
oRadGrp:Fblock := bBlock *ASSIGN*

<**bBlock**> is a code block or NIL. The code block callback, when present, is evaluated each time the RadioGroup object receives or loses input focus. The code block receives two arguments: the object self and the current :HasFocus status, which indicates whether the radio button is receiving (.T.) or losing (.F.) input focus. In GUI, the object receives focus every times the user clicks (or activates) the radio button widget and looses focus when other widget is selected.

Compatibility: Available also in CL53, but Clipper does not pass any arguments to the code block, and hence cannot use generalized but object specific code blocks which needs to check the current oRadGrp:HasFocus status by itself.

See also: oRadGrp:HasFocus, :SetFocus(), :KillFocus(), :Sblock

oRadGrp:FirstItem() —> oRadioButton

Selects the first available and selectable item in the group list, considering the oRadButton:Enabled status. If no selectable items are available, NIL is returned. Selecting the item does not change the radio button status.

Compatibility: Available in FS5 only.

See also: oRadGrp:GetItem(), :NextItem(), :LastItem(), :ItemCount

oRadGrp:FrameStyle —> nStyle *ACCESS*
oRadGrp:FrameStyle := nStyle *ASSIGN*

Set the frame style of RadioGroup. Assign is considered before first :Display(). The constants are defined in button.fh

<nStyle> constant	Action
BS_GROUPBOX_NONE	don't draw frame around the radio group
BS_GROUPBOX_SUNKEN	sunken box frame (default in GUI i/o)
BS_GROUPBOX_RAISED	raised box frame
BS_GROUPBOX_PLAIN	plain box frame (default in Terminal i/o)

oRadGrp:FrameWidth —> nWidth *ACCESS*
oRadGrp:FrameWidth := nWidth *ASSIGN*

Set the line width (in pixel) of RadioGroup frame. Apply for GUI only. The default width is 1. Assign is considered before first :Display()

oRadGrp:GetAccel (nKey) —> nPos

<**nKey** > is a numeric value that indicates the Inkey() value to check.

<**nPos**> is the returned numeric value in the range 1 to :ItemCount that indicates the first position in the list of items whose accelerator key matches the <nKey> value. If a corresponding accelerator is not found, 0 is returned.

Compatibility: Available also in CL53

See also: RadioButton:Caption, oRadGrp:ItemCount

oRadGrp:GetItem([nPos]) —> oRadioButton

<**nPos**> is a numeric value in the range 1 to :ItemCount that indicates the position in the list of the item that is being retrieved. If not specified or is 0 or NIL, the current radio button object is returned. Selecting the item does not change the radio button status.

<**oRadioButton**> is the RadioButton object specified by <nPos>, even if the button is disabled. If no item is available at the specified position, NIL is returned.

Compatibility: Available also inCL53

See also: oRadGrp:Button(), :FirstItem(), :NextItem(), :LastItem(), :ItemCount

oRadGrp:Handler –> bHandler *ACCESS*
oRadGrp:Handler := bHandler *ASSIGN*

<**bHandler**> is a code block or NIL. The code block, when present, is invoked from the oRadGrp:Show() method and replaces the default radio button handler available in the *<FlagShip_dir>/system/radiogrouphand.prg* source file. The code block receives one argument, the object self.

Compatibility: Available in FS5 only.

See also: oRadGrp:Show()

oRadGrp:HasFocus –> lFocus *ACCESS*

<**lFocus**> is a logical value indicating whether the radio group object has input focus (TRUE) or not. In GUI, the object receives focus every times the user clicks (or activates) the widget and looses the focus when other widget is selected.

Compatibility: Available also in CL53

See also: oRadGrp:KillFocus, :SetFocus(), :Fblock

oRadGrp:Height –> nRows *ACCESS*
oRadGrp:Height := nRows *ASSIGN*
oRadGrp:Height ([nRows], [lPixel]) –> nRows

<**nRows**> is a numeric value that indicates the height of the radio group widget (control) including the frame. With Access and assign, the value is either in coordinates or pixels according to the current SET PIXEL status. The default value is determined from oRadGrp:Top and oRadGrp:Bottom. Setting a new value overwrites oRadGrp:Bottom.

<**lPixel**> is optional value indicating if the passed and returned value is in coordinates or pixels. If true(.T.), the column data are in pixel; if false (.F.), data are in coordinates, otherwise the current SET PIXEL status is used.

Compatibility: Available in FS5 only. Apply for GUI mode and is ignored otherwise

See also: oRadGrp:Width, :Top, :Bottom

oRadGrp:HitTest(nMouseRow, nMouseCol, [lPixel]) —> nStatus

Determines if the mouse cursor is within the region of the screen that the radio button occupies.

<**nRow**> Numeric value representing the current or tested screen row position of the mouse cursor.

<**nCol**> Numeric value representing the current or tested screen column position of the mouse cursor.

<**lPixel**> If specified TRUE, the mouse coordinates are assumed in pixel. If FALSE, the mouse parameters are assumed in current row/col coordinates. If this parameter is not specified (i.e. NIL), the kind of passed mouse coordinates is determined from the current SET PIXEL setting.

<**nStatus**> Returned numeric value indicating the relationship of the mouse cursor with the radio button. The constants are specified in button.fh header file.

Value	Constant	Description: the mouse cursor is...
0	HTNOWHERE	not located in the box region
-1	HTTOPLEFT	on the top left corner of the object border
-2	HTTOP	on the object top border
-3	HTTOPRIGHT	on the top right corner of object border
-4	HTRIGHT	on the object right border
-5	HTBOTTOMRIGHT	on the bottom right corner of the obj border
-6	HTBOTTOM	on the object bottom border
-7	HTBOTTOMLEFT	on the bottom left corner of the obj border
-8	HTLEFT	on the object left border
-2049	HTCLIENT	within the radio group's screen region

Compatibility: Available also in CL53

See also: Mrow(), Mcol()

oRadGrp:HotBox —> cBox *ACCESS*
oRadGrp:HotBox := cBox *ASSIGN*

<**cBox**> is an optional string that specifies the characters to use when drawing a box around the radio button when it has input focus. Its default value is pre-defined in the global array element `_aGlobSetting [GSET_T_C_HOTBOX]` and is usually B_DOUBLE. The following <cBox> constants are defined in the box.fh file, the _aGlobSetting[] array constants in set.fh and initio.prg files.

Constant	Description
B_SINGLE	Single line box
B_DOUBLE	Double line box
B_SINGLE_DOUBLE	Single line top/bottom, double line sides
B_DOUBLE_SINGLE	Double line top/bottom, single line sides
B_PLAIN	Plain ASCII characters

Compatibility: Available also in CL53

See also: oRadGrp:ColdBox, :HasFocus, :SetFocus(), @..BOX

oRadGrp:Init([par1]...[par5]) —> self

This is an internal method invoked automatically at instantiation of the RadioButton object. It is not intended to be called by the application.

Compatibility: Available also in VO

See also: RadioGroup{} instantiation

oRadGrp:InsItem(nPos, oRadButt) —> self

<**nPos**> is a numeric value in the range of 1 to :ItemCount that indicates the position in the list at which the new item is inserted. Values less or equal to zero are treated as 1, values greater than :ItemCount performs the same action as :AddItem()

<**oRadButt**> is the radio button object to be inserted.

Compatibility: Available also in CL53 and VO.

See also: oRadGrp:AddItem(), :DelItem(), :ItemCount

oRadGrp:ItemCount —> nCount *ACCESS*

<**nCount**> is a numeric value that indicates the total number of radio buttons in the RadioGroup list.

Compatibility: Available also in CL53

See also: oRadGrp:AddItem(), :InsItem()

oRadGrp:KillFocus() —> self

Take input focus away from a RadioGroup object. Upon receiving this message, the RadioGroup object redisplays itself with the :ColdBox frame and, if present, evaluates the code block specified by :Fblock. This message is meaningful only when the RadioGroup object has input focus.

Compatibility: Available also in CL53

See also: oRadGrp:HasFocus, :SetFocus(), :Fblock

oRadGrp:LastItem() —> oRadioButton

Selects the last available and selectable item in the group list, considering the oRadButton:Enabled status. If no selectable items are available, NIL is returned. Selecting the item does not change the radio button status.

Compatibility: Available in FS5 only.

See also: oRadGrp:GetItem(), :FirstItem(), :NextItem(), :ItemCount

oRadGrp:Left —> nCol *ACCESS*
oRadGrp:Left := nCol *ASSIGN*
oRadGrp:Left([nCol], [lPixel]) —> nCol

<**nCol**> is a numeric value that indicates the leftmost screen column where the radio group frame is displayed. With Access/assign, the value is either in coordinates or pixels according to the current SET PIXEL status. The default is taken from object instantiation and, if not specified, the coordinates are calculated automatically from radio group items at the first oRadGrp:Display() or :Show() invocation.

<**lPixel**> is optional value indicating if the passed and returned value is in coordinates or pixels. If true (.T.), the column data are in pixel; if false (.F.), data are in coordinates, otherwise the current SET PIXEL status is used.

With terminal i/o, the <nCol> value specifies the column where the frame of oRadGrp:ColdBox and :HotBox is displayed.

With GUI i/o, the radio group is displayed as a widget (control) and <nCol > is the leftmost widget coordinate. To ensure the same look and feel to an application running in textual mode, and to display the widget at approx. the same screen position, the given <nCol> coordinate is automatically adapted by adding a pixel value taken from the global array element _aGlobSetting[GSET_G_N_RADGRP_LEFT] which may be positive or negative. Additional adjustment is possible via oRadGrp:Right and :Width

Compatibility: Access/assign is available in CL53.

See also: oRadGrp:Right, :Width, RadioGroup{} instantiation

oRadGrp:Message —> cText *ACCESS*
oRadGrp:Message := cText *ASSIGN*

<**cText**> is a character string displayed in the windows status bar (GUI), or in the screen line specified by SET MESSAGE (in terminal mode). Apply only if the current radio button has not own :Message which is preferred otherwise.

Compatibility: Available also in CL53.

See also: oRadGrp:Tooltip(), SET MESSAGE, RadioButton:Message

oRadGrp:Modified —> lOk *ACCESS*
oRadGrp:Modified := lOk *ASSIGN*

<**lOk**> is a logical value that is set to TRUE when the user clicks on a radio button, and reset to FALSE when the mouse button is released.

Compatibility: Available also in VO. Apply in GUI mode only.

oRadGrp:NextItem() —> oRadioButton

Selects the next available and selectable item in the group list, considering the oRadButton:Enabled status. If there are no further selectable items available, :LastItem() is executed with SET WRAP OFF and :FirstItem() with SET WRAP ON. If no selectable items are available, NIL is returned. Selecting the item does not change the radio button status.

Compatibility: Available also in CL53

See also:oRadGrp:GetItem(), :FirstItem(), :PrevItem(), :LastItem(), :ItemCount

oRadGrp:PrevItem() —> oRadioButton

Selects the previous available and selectable item in the group list, considering the oRadButton:Enabled status. If there are no previous selectable items available, :FirstItem() is executed with SET WRAP OFF and :LastItem() with SET WRAP ON. If no selectable items are available, NIL is returned. Selecting the item does not change the radio button status.

Compatibility: Available also in CL53

See also: oRadGrp:GetItem(), :FirstItem(), :Next Item(), :LastItem(), :ItemCount

oRadGrp:Right —> nCol *ACCESS*
oRadGrp:Right := nCol *ASSIGN*
oRadGrp:Right([nCol], [lPixel]) —> nCol

<**nCol**> is a numeric value that indicates the rightmost screen column where the radio group frame is displayed. With Access/assign, the value is either in coordinates or pixels according to the current SET PIXEL status. The default is taken from object instantiation and, if not specified, the coordinates are calculated automatically from radio group items at the first oRadGrp:Display() or :Show() invocation.

<**lPixel**> is optional value indicating if the passed and returned value is in coordinates or pixels. If true(.T.), the column data are in pixel; if false (.F.), data are in coordinates, otherwise the current SET PIXEL status is used.

With terminal i/o, the <nCol> value specifies the column where the frame of oRadGrp:ColdBox and :HotBox is displayed.

With GUI i/o, the radio group is displayed as a widget (control) and <nCol> is the rightmost widget coordinate. To ensure the same look and feel to an application running in textual mode, and to display the widget at approx. the same screen position, the given <nCol> coordinate is automatically adapted by adding a pixel value taken from the global array element _aGlobSetting[GSET_G_N_RADGRP_RIGH] which may be positive or negative. Additional adjustment is possible via oRadGrp:Left and :Width

Compatibility: Available also in CL53 which does not calculate the frame coordinates automatically but requires the input.

See also: oRadGrp:Left, :Width, RadioGroup{} instantiation

oRadGrp:Sblock —> bBlock *ACCESS*
oRadGrp:Sblock := bBlock *ASSIGN*

<**bBlock**> is an optional code block or NIL. The code block callback, when present, is evaluated each time the RadioGroup selection changes. The name "Sblock" refers to state block. The code block receives two arguments: 1) the object self, and 2) the current item number, i.e. the content of oRadGrp:Buffer.

Compatibility: Available also in CL53, but undocumented and w/o passing any arguments to the code block.

See also: oRadGrp:Buffer, :Fblock, :Select(), :GetItem(), :FirstItem(), :NextItem()

oRadGrp:Select(nPos, [lOnOff]) —> lOnOff

<**nPos**> is a numeric value in the range 1 to :ItemCount that indicates the position in the list of the item that is being retrieved. No action is taken if <nPos> is out of range. NIL specifies the current radio button without skipping forward or backward.

<**lOnOff**> Set to true (.T.) to check the radio button or false (.F.) to uncheck it. If omitted or NIL, the radio button state will toggle to its opposing state.

If none, or only one argument is given and it is logical, it is interpreted as (NIL, <lOnOff>) to ensure the logical compatibility to RadioButton:Select() method.

This method is provided for you convenience and is equivalent to oRadGrp:GetItem(nPos):Select(lOnOff). It selects the specified radio button item in the group list, and sets or toggles the radio button state. Hence, the RadioButton:Enabled status is not checked here as opposite to :FirstItem(), :NextItem() etc. which should be preferably used instead, to skip to other radio button, if some items are disabled.

Compatibility: Available also in CL53, which supports 1st argument only

See also:
oRadGrp:Buffer, :GetItem(), :FirstItem(), :NextItem(), :PrevItem(), :LastItem(), RadioButton:Select()

oRadGrp:SetColor([cAttrib]) —> cColor

This method is used for uniformly setting the color attributes of all the radio buttons in its group. It accomplishes this by assigning RadioButton:ColorSpec := <cAttrib> to each of the radio buttons in the group list.

<**cColor**> is a character string that indicates the color attributes that are used by the radio button's display() method. If the parameter is not specified or is NIL or empty, no action is taken and the current setting is returned. The <cAttrib> string must contain eight color specifiers, otherwise the rest remain unchanged.

Position in <cAttrib>	Applies To	Default value used from curr SET COLOR
1	Radio button without input focus, unselected	Unselected
2	Radio button without input focus, selected	Unselected
3	Radio button with input focus, unselected	Enhanced
4	Radio button with input focus, selected	Enhanced
5	Radio button's caption	Standard
6	Radio button caption's accel. key w/o focus	Standard
7	Radio button caption's accel. key with focus	Background
8	Radio button and caption, disabled	Border

Specifying "-" for foreground or background lets the original color unchanged, which enables you to change the required color attribute only.

Compatibility: Available also in CL53 with seven attributes and returns self. This property is considered in terminal mode only and is ignored in GUI mode.

See also: oRadGrp:ColorSpec, :SetStyle(), RadioButton:ColorSpec

oRadGrp:SetFocus() —> self

Set input focus to a RadioGroup object. Upon receiving this message, the RadioGroup object redisplays itself with all assigned RadioBox'es, with the :HotBox frame and, if present, evaluates the code block specified by :Fblock. This message is meaningful only when the RadioGroup object does not have input focus. In GUI, the object receives focus also every times the user clicks (or activates) the widget.

Compatibility: Available also in CL53

See also: oRadGrp:HasFocus, :KillFocus(), :Fblock, :HotBox

oRadGrp:SetStyle ([cStyle]) —> cStyle

This method is used for uniformly setting the style attributes of all the radio buttons in its group. It accomplishes this by assigning RadioButton:Style := <cStyle> to each of the radio buttons in the group list.

<**cStyle**> is a character string that indicates the delimiter characters that are used by the radio group Display() and Show() method. If the parameter is not specified or is NIL or empty, no action is taken and the current setting is returned. The string must contain four characters. The first is the left delimiter, the 2nd is the "selected" indicator, the 3rd is the "unselected" indicator, and the 4th character is the right delimiter. The default style is pre-defined in the global array element `_aGlobSetting[GSET_T_C_RADGRP_STYLE]` containing "(*)" at start-up; it may be re-defined by a simple assignment later.

Compatibility: Available also in CL53. Considered in terminal mode only, ignored in GUI.

See also: oRadGrp:SetColor(), RadioButton:Style

oRadGrp:Show([lMust]) —> self

This method activates either the default or user specific input handler. Is displays all the radio buttons in the list, activates the group focus, waits for user input and sets the selected radio button status to on/off status accordingly, and on exit, kills the group focus. The default handler is available in the *<FlagShip_dir>/system /radiogrphand.prg* source file and is roughly equivalent to the manual code sequence given in the second example in front of this class description. If all radio items are disabled, :Show() exits immediately. Changes of the RadioButton or RadioGroup properties (possible e.g. via the :Sblock or SET KEY, ON KEY callbacks) are not considered anymore during the user input in the standard handler.

<**lMust**> is an optional logical value. True (.T.) indicates that one radio item must be ON, otherwise an user exit is disabled. If <lMust> is false, NIL or not given, :Show() accepts also unselected radio group items at the exit via Return or Escape key.

You may assign your own handler by the oRadGrp:Handler property.

Compatibility: Same named method is available also in VO which returns NIL

See also: oRadGrp:Display(), oRadGrp:Handler

oRadGrp:ToolTip —> cText *ACCESS*
oRadGrp:ToolTip := cText *ASSIGN*

<**cText**> is a string representing the displayed tool tip, i.e. a short info message which pop up's when the mouse is over the radio group widget.

Compatibility: Available in FS5 only, apply for GUI, ignored otherwise
See also: oRadGrp:Message

oRadGrp:Top –> nRow *ACCESS*
oRadGrp:Top := nRow *ASSIGN*
oRadGrp:Top ([nRow], [lPixel]) –> nRow

<**nRow**> is a numeric value that indicates the screen topmost row where the cold and hot box frame of the radio group is displayed. The input and output value is either in coordinates or in pixels, depending on the current SET PIXEL setting. The default coordinates are specified at radio group instantiation or are calculated automatically from radio group items at the first oRadGrp:Display() or oRadGrp:Show() invocation.

<**lPixel**> is optional value indicating if the set/get value is in coordinates or pixels. If true(.T.), the row data are in pixel; if false (.F.), data are in coordinates, otherwise the current SET PIXEL status is used.

With terminal i/o, the <nRow> value specifies the row where the frame of oRadGrp:ColdBox and :HotBox is displayed.

With GUI i/o, the radio group is displayed as a widget (control) and <nRow> is the top widget coordinate. To ensure the same look and feel to an application running in textual mode, and to display the widget at approx. the same screen position, the given <nRow> coordinate is automatically adapted by adding a pixel value taken from the global array element `_aGlobSetting[GSET_G_N_RADGRP_TOP]` which may be positive or negative. Additional adjustment is possible via oRadGrp:Bottom and oRadGrp:Height

Compatibility: Available also in CL53 which does not calculate the frame coordinates automatically but requires the input.

See also: oRadGrp:Bottom, :Left, :Right

oRadGrp:TypeOut –> lVal *ACCESS*

<**lVal**> is a logical value that indicates whether the group contains any selectable buttons. A value of true (.T.) indicates the group contains selectable buttons; a false (.F.) value indicates that the group is empty or that all items are disabled.

Compatibility: Available also in CL53

oRadGrp:Value –> exp *ACCESS*
oRadGrp:Value := exp *ASSIGN*

<**exp**> contains the relative position (1 to :ItemCount) of the radio button toggled to ON. If this radio button contains :Data (i.e. the RadioButton:Data is not NIL), it value is returned instead.

Compatibility: Available also in CL53 (undocumented) and in VO

See also: oRadGrp:Show(), RadioButton:Data

oRadGrp:Width –> nCol ***ACCESS***
oRadGrp:Width := nCol ***ASSIGN***
oRadGrp:Width ([nCol], [lPixel]) –> nCol

<**nCol**> is a numeric value that indicates the width of the radio group. With Access and assign, the value is either in coordinates or pixels according to the current SET PIXEL status. The default value is determined from oRadGrp:Left and oRadGrp:Right. Setting a new value overwrites oRadGrp:Right instance.

<**lPixel**> is optional value indicating if the passed and returned value is in coordinates or pixels. If true(.T.), the column data are in pixel; if false (.F.), data are in coordinates, otherwise the current SET PIXEL status is used.

Compatibility: Available in FS5, apply for GUI mode only

See also: oRadGrp:Height, :Left, :Right

TBROWSE Class

A TBROWSE object is a general purpose browsing mechanism for table- oriented data, i.e. arrays or databases. It provides mechanisms for acquiring, formatting and displaying data.

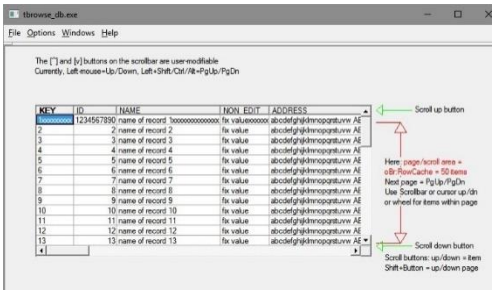
The output of TBROWSE can be created as with BROWSE() or DBEDIT(), but TBROWSE is the most powerful. It differs from the standard functions in its degree of exclusive control. On the other hand, it does not replace DBEDIT() or BROWSE(), since for a simple structure and display they are significantly easier to handle by the programmer. In fact, DBEDIT() uses TBROWSE, see its source code in *<FlagShip_dir>/system/ dbedit.prg*

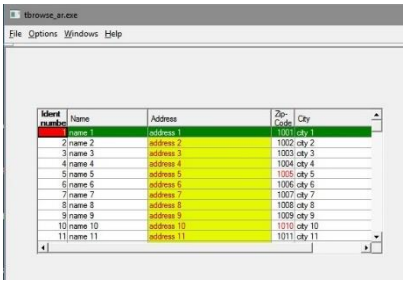
See also section LNG.7 for general Tbrowse description and it handling.

1. Creating an Object

Any TBROWSE object contains one or more TBCOLUMN objects (see TbColumn class). In general, the TBROWSE instances and methods control browsing and positioning in the data table, while TBCOLUMN formatting and data output.

The TBROWSE object is created using the standard function TBROWSENEW() or the specialized TBROWSEDB() or TBROWSEARR(), both available in source code in *<FlagShip_dir>/system/tbr*.prg*. TbrowseNew() creates generic TBROWSE object, while the **TbrowseDb()** an object customized and partially initialized for browsing databases, and the **TbrowseArr()** for browsing arrays. See *<FlagShip_dir>/examples/tbrowse_db.prg* and *tbrowse_ar.prg* as well as example in Chapter 4 below.





2. Specifying the Columns

Data columns to be displayed from the table are initialized by invoking TBCOLUMNEW() and assigning the resulting column object to TBROWSE using the tb:ADDCOLUMN() method. See examples in chapter 4 and TbColumn class. At least one column needs to be assigned.

Data display and browsing, according to the user request, is controlled by the source code using a loop. In the loop body, the key-press is checked and the corresponding action is performed calling a TBROWSE method. Alternatively, you may use FlagShip's pre-defined

handler (available in the library but also in source code for an easy customizing) assigned by the :Handler property and used by the :EXEC () method.

3. Stabilizing the Display

To permit greater control over the browsing system, TBROWSE allows to move the data pointer asynchronously (in the background), with respect to the currently visible screen. This may visually speed up the preparation of displayed data. This asynchronous process is called stabilization.

Every time a movement in the data table is requested, the system becomes "unstable". TBROWSE does not display the data immediately but waits until the object is stabilized with the tb:STABILIZE() method. When this message is received, the browse displays one data record. When data movement and data display is finished, the TBROWSE system becomes "stable". If any user key is pressed in the meantime, the stabilize and/or output process may be interrupted for the next action required. Otherwise, the stabilizing method is repeated until the whole screen (region) is displayed and the tb:STABILIZE() method or the tb:STABLE instance returns TRUE. The usual method is

```
WHILE .T.
   WHILE !mybrow:STABILIZE()              // wait for data display,
      IF NEXTKEY() != 0                   // optional:
   EXIT                                   // manage async.input
      ENDIF
   ENDDO

   key := INKEY(0)                        // process key pressed
   IF key == K_ESC
EXIT
   ELSEIF key ==  ....                    // process movement
   ENDIF                                  // see chapter 5
ENDDO
```

Using the tb:FORCESTABLE() method instead of tb:STABILIZE() will display all data belonging on the screen, which avoids invoking the stabilization several times.

```
WHILE .T.
   mybrow:FORCESTABLE()                   // wait for data display,
   key := INKEY(0)                        // process key pressed
   IF key == K_ESC
EXIT
   ELSEIF key ==  ....                    // process movement
   ENDIF
ENDDO
```

4. Data Movement

TBROWSE manages any data in table form, such as arrays and databases. Data retrieval and file or array positioning are performed via user-supplied code blocks, allowing a high degree of flexibility and interaction between the browsing mechanism and programming.

There are three instance variables containing a code block, which handle data repositioning: tb:SKIPBLOCK to move one record (or row) forward or backward, tb:GOTOPBLOCK and tb:GOBOTTOMBLOCK to reach the first or last data record. The :SKIPBLOCK is mandatory, :GOBOTTOM and :GOTOP optional but highly recommended. An example to move array data (see also example in TBROWSENEW() and in <FlagShip_dir>/examples/ tbrowse_db.prg or tbrowse_ar.prg):

```
*** test.prg
myname := {"one","two","three"}
mydata := {{1, 2, 3}, {"aa","bb","cc"}, {11,12,13}}
browsearr (mydata, myname)

#define KNOWN_ARR_SIZE

FUNCTION browsearr (arr, names)               // multi-dim. data
LOCAL   mybrow  := TBROWSENEW (10,10, 17,40)  // create object
PRIVATE element := 1                          // current element
PRIVATE arrdata := arr

mybrow:GOTOPBLOCK  := {|| element := 1 }
mybrow:GOBOTTOMBLOCK := {|| element:= LEN(arrdata) }
mybrow:SKIPBLOCK   := {|how,obj| skipper (arrdata, @element, how) }

mybrow:COLSEP  := " " + CHR(179) + " "      // or " : "
mybrow:HEADSEP := "Ä┼Ä"                     // or "-+-"
#ifdef KNOWN_ARR_SIZE
   mybrow:ADDCOLUMN (TBCOLUMNNEW (names[1], {|| arrdata[1,element] }))
   mybrow:ADDCOLUMN (TBCOLUMNNEW (names[2], {|| arrdata[2,element] }))
   mybrow:ADDCOLUMN (TBCOLUMNNEW (names[3], {|write| IF (write==NIL, ;
           arrdata[3,element], arrdata[3,element] := write) }))
#else
   FOR ii = 1 TO LEN(names)                 // variable array size,
      idx = LTRIM(STR(ii))                  // 3rd and following
      IF ii < 3                             // columns are editable
         mybrow:ADDCOLUMN (TBCOLUMNNEW (names[ii], ;
                           {|| arrdata[&idx., element] } ))
      ELSE
         mybrow:ADDCOLUMN (TBCOLUMNNEW (names[ii], {|write| ;
            IF(write==NIL, arrdata[&idx., element], ;
                           arrdata[&idx., element] := write) } ))
      ENDIF
   NEXT
#endif
mybrowhandle (mybrow)                       // see chapter 5
RETURN
```

```
FUNCTION skipper (arr, elem, how)
LOCAL old := elem
elem += how
DO CASE
CASE how > 0                                  // skip forward
   IF elem > LEN(arr)
      elem = LEN(arr)
   ENDIF
CASE how < 0                                  // skip backward
   IF elem < 1
      elem = 1
   ENDIF
ENDCASE
RETURN elem - old                             // elements skipped
```

To browse a database, either the TBROWSEDB() function which creates all the three skip blocks automatically, or the similar program sequence may be used:

```
FUNCTION browsedbf ()
LOCAL mybrow
#ifdef USING_TBROWSEDB
   mybrow := TBROWSEDB (0,0, 7,20)            // creates dbf object
#else
   mybrow := TBROWSENEW (0,0, 7,20)           // creates generic object
   mybrow:GOTOPBLOCK  := {|| DBGOTOP() }
   mybrow:GOBOTTBLOCK := {|| DBGOBOTTOM() }
   mybrow:SKIPBLOCK   := {|how,obj| skipdb (how) }
#endif

FOR ii = 1 TO FCOUNT()                        // see chapter 4
   mybrow:ADDCOLUMN (TBCOLUMNNEW (FIELDNAME(ii), ;
                     FIELDBLOCK (FIELDNAME(ii)) )
NEXT

mybrowhandle (mybrow)                         // see chapter 5
RETURN
#ifndef USING_TBROWSEDB
   FUNCTION skipdb (how)                      // user supplied fn
   LOCAL countmoved := 0
   DO CASE
   CASE how == 0                              // flush only
      DBCOMMIT ()
   CASE how > 0                               // skip forward
      WHILE countmoved < how .and. ! EOF()
         SKIP
         IF EOF()
            ?? CHR(7)
            SKIP -1
         ELSE
            countmoved++
         ENDIF
      ENDDO
   CASE how < 0                               // skip backward
      WHILE countmoved > how .and. ! BOF()
         SKIP -1
         IF BOF()
```

```
            ?? CHR(7)
         ELSE
            countmoved--
         ENDIF
      ENDDO
   ENDCASE
   RETURN countmoved                        // records skipped
#endif
```

When searching for the next data (record), by executing the movement method, TBROWSE will invisibly post it at the next screen row, if available. The data found is displayed during the stabilization process.

You may invoke several requests for data movement between two user key-presses and then display the data using tb:FORCESTABLE() or repeating tb:STABILIZE().

When tb:STABILIZE() is invoked, it tries to leave the same row (data item) highlighted as before, unless the visual movement of the highlight bar was requested.

During operation, a TBROWSE object retrieves data by evaluating the code blocks supplied. The data is organized into rows and columns and displayed in a specified region of the screen. The TBROWSE object maintains an internal browse cursor. Its vertical or horizontal movement on the screen and within the processed data is controlled by the TBROWSE methods, which usually performs the user keystrokes. A highlight bar marks the current position within the processed data.

Initially, the browse cursor is placed on the first data item associated with the first column.

5. Handling a User Request

To comply with the user request to display a special portion of browsed data, the keystroke is read by e.g. INKEY() function, and the TBROWSE movement for vertical or horizontal display or the data movement is performed invoking the corresponding TBROWSE method.

For standard use, there are ready-to-use keyboard handlers available, see

tbrowsehand.prg = for TbrowseArr() and array-based Tbrowse, and
tbrowsedbhand.prg = for TbrowseDb() and database-based Tbrowse

in the *<FlagShip_dir>/system* directory. These handlers are used per default by oTbrowse:Exec() method. You may define your own handler as well, and either assign it via oTbrowse:Handler when using oTbrowse:Exec(), or simply invoke your handler directly.

Example for executing the user request (continuing from chapter 4):

```
FUNCTION mybrowhandle (mybrow)
 LOCAL key
 WHILE .T.
    WHILE (!mybrow:STABLE)                    // (re)build screen,
       mybrow:STABILIZE()                     // wait for stabilizing
       IF NEXTKEY() != 0                      // optional:
    EXIT                                      // manage async.input
       ENDIF
    ENDDO

    key := INKEY(0)                           // get key pressed
    DO CASE
    CASE key = K_LEFT .or. key = K_CTRL_S     // cursor left
       mybrow:LEFT()                          // = column left
    CASE key = K_RIGHT .or. key= K_CTRL_D     // cursor right
       mybrow:RIGHT()                         // = column right
    CASE key = K_UP   .or. key = K_CTRL_E     // cursor up
       mybrow:UP()                            // = previous record
    CASE key = K_DOWN .or. key = K_CTRL_X     // cursor down
       mybrow:DOWN()                          // = next record
    CASE key = K_PGUP .or. key = K_CTRL_R     // page-up
       mybrow:PAGEUP()                        // = previous window
    CASE key = K_PGDN .or. key = K_CTRL_C     // page-down
       mybrow:PAGEDOWN()                      // = next window
    CASE key = K_CTRL_PGUP                    // Ctrl page-up
       mybrow:GOTOP()                         // = first record
    CASE key = K_CTRL_PGDN                    // Ctrl page-down
       mybrow:GOBOTTOM()                      // = last record
    CASE key = K_HOME .or. key = K_CTRL_A     // home
       mybrow:HOME()                          // = first screen column
    CASE key = K_END  .or. key = K_CTRL_F     // end
       mybrow:HOME()                          // = last screen column

    CASE key = K_ESC                          // escape
```

```
      EXIT                                  // terminates browsing
   CASE key = K_RETURN                      // return, enter
      myedit (mybrow)                       // edit cell, see chapt 6
   OTHERWISE
      ?? CHR(7)                             // invalid key
   ENDCASE
ENDDO                                       // system is unstable now
RETURN NIL
```

Note that TBROWSE actions are handled in the background, e.g. when Pg-Dn and cursor right keys are pressed twice rapidly in sequence, the TBROWSE system first skips the data two pages forward, re-arranges the columns and then displays the data reached, if no other key is pending.

See also the source of *<FlagShip_dir>/system/dbedit.prg* for an example of the implementation.

Generally, if using the :Exec() or :Handler property of Tbrowse, you don't need to create own handler with TbrowseDb() or TbrowseArr(), since a default handler is already available and assigned. You may re-assign it to your own handler using the :Handler Tbrowse property. The source code for database and array handler is available in *<FlagShip_dir>/system* directory.

6. Editing Data

You may edit the current data cell at user request, using e.g. the standard GET/READ system. Specify a TBCOLUMN read/write code block to allow the replacement of the data. See the third block in Chapter 4. Example (continued from Chapter 4 and 5):

```
FUNCTION myedit (brow)
LOCAL data, row := ROW(), col := COL()
LOCAL sSave, colobj, block, lMouse, getlist := {}
IF brow:COLPOS < 3
   ?? CHR(7)
   @ 0,0 SAY "The first two columns are not editable"
   SETPOS (row, col)
   RETURN NIL
ENDIF
IF .not. brow:STABLE
   @ 0,0 SAY "Let's stabilize first"
   SETPOS (row, col)
   RETURN NIL
ENDIF
@ 0,0

lMouse := brow:EnableMouseClick
brow:EnableMouseClick := .F.  // disable mouse click during edit
colobj := brow:GETCOLUMN(brow:COLPOS)
block  := colobj:BLOCK
data   := EVAL(block)       // Retrieve data using the column block
sSave  := SaveScreen(row,col,row,brow:nRight)
@ row, col GET data COLOR "W+/BG,W+/BG"
READ
RestScreen(row,col,row,brow:nRight,sSave)

IF LASTKEY() != 27
   EVAL (block, data)       // REPLACE if the block is read/write
   brow:RefreshAll()        // and ensure the Tbrowse display
   brow:ForceStable()       // the changes
ENDIF
brow:EnableMouseClick := lMouse  // enable mouse click
SETPOS (row, col)
RETURN NIL
```

To disable unintended mouse click on another field during editing, use the oTbr:EnableMouseClick property, as demonstrated above. When you are using the default keyboard handler, the editing is already implemented there. You may disallow editing by oTbrowse:ReadOnly := .T. or by oTbColumn:ReadOnly := .T. for specific column.

Unicode: In GUI mode, FlagShip supports also Unicode (UTF-8 and UTF-16). Since each glyph is stored in UTF-8 encoding which results in one to four bytes each - usually as chr(128..255), the database or array field containing glyphs needs to be set correspondingly (e.g. to 30 or more characters to accept 10 Japanese or Chinese glyphs). To display the Unicode field, the current or default font must be set to Unicode by SET GUICHARSET FONT_UNICODE or oApplic:Font:CharSet(FONT_UNICODE). For editing fields with glyphs, SET MULTIBYTE ON is required.

Tbrowse Class Instantiation

TbrowseNew ()

Syntax 1:

```
obj = TbrowseNew ([expN1], [expN2], [expN3],
                  [expN4], [expL5], [expL6], [expC7],
                  [expO8], [expN9])
```

Syntax 2:

```
obj = Tbrowse { [expN1], [expN2], [expN3], [expN4],
                [expL5], [expL6], [expC7], [expO8],
                [expN9] }
```

Purpose:

Creates a new, generic TBROWSE object, optionally initialized by the arguments supplied.

Options:

<**expN1**> is the top screen row where the TBROWSE is displayed. This argument is equivalent to assigning the obj:NTOP with the same value. The valid range is 0...MAXROW(). The default value is zero.

<**expN2**> is the leftmost screen column where the TBROWSE is displayed. This argument is equivalent to assigning the obj:NLEFT with the same value. The valid range is 0...MAXCOL(). The default value is zero.

<**expN3**> is the bottom screen row where the TBROWSE is displayed. This argument is equivalent to assigning the obj:NBOTTOM with the same value. The valid range is <expN1>...MAXROW(). The default value is MAXROW().

<**expN4**> is the rightmost screen column where TBROWSE is displayed. This argument is equivalent to assigning the obj:NRIGHT with the same value. The valid range is <expN2>...MAXCOL(). The default value is MAXCOL().

<**expL5**> is the pixel specification. If .T., the coordinates given are assumed in pixel. If .F., the coordinates are in row/column. If not given or NIL, the current SET PIXEL is considered.

<**expL6**> specifies whether the Tbrowse widget is re-sizeable by user or not. Default is .F. which means the Tbrowse widget is fix. Applies for GUI mode only, ignored otherwise.

<**expC7**> is a ToolTip string. Applies for GUI mode only, ignored otherwise.

<**expO8**> is a Font object. If not specified, the oApplic:Font is used. Applies for GUI mode, ignored otherwise.

<**expN9**> specify the row height in pixel, see also tb:RowHeight. If not specified, the size of one row is used. Applies for GUI mode, ignored otherwise.

Returns:

<**obj**> is the newly allocated TBROWSE object, usually assigned to a regular FlagShip variable or to an array element.

Description:

TBROWSENEW() creates a new, empty TBROWSE object for generic use. To create a partially predefined TBROWSE object for browsing databases, TBROWSEDB() may be used instead.

If the optional arguments are supplied, the corresponding instance variables are filled with these values.

Prior to using the TBROWSE object, at least a skip block (see tb:SKIPBLOCK) and one or more TBCOLUMNs (see tb:ADDCOLUMN()) must be specified.

Tuning:

In GUI mode, you may adjust the Tbrowse display by assigning: To allow adjustment of rows/columns, set

```
  _aGlobSetting[GSET_G_L_TBROW_ADJ    ] := .T. // default
```

If above is set .T., you may set values (in pixels) for

```
  _aGlobSetting[GSET_G_N_TBROW_TOP    ] := -3  // Tbrowse top row
  _aGlobSetting[GSET_G_N_TBROW_BOT    ] := 0   // Tbrowse bottom row
  _aGlobSetting[GSET_G_N_TBROW_LEFT   ] := 0   // Tbrowse left col
  _aGlobSetting[GSET_G_N_TBROW_RIGH   ] := 0   // Tbrowse right col
  _aGlobSetting[GSET_G_N_TBROW_RHEIGHT] := 0   // Tbrowse row height
  _aGlobSetting[GSET_G_N_TBROW_CWIDTH ] := 4   // add TbColumn width
```

You also may adjust columns by assigning 0 to center, or greater zero for number of pixels left

```
 _aGlobSetting[GSET_G_N_TBROW_COLADJ] := 0    // default
```

The cursor during oTbr:stabilize() is "wait" = -9, but can be changed to 0 if normal mouse cursor is desirable, or to other values according to SetGuiCursor() values

```
 _aGlobSetting[GSET_G_N_TBROW_WAIT  ] := -9  // hour glas
```

Multi-line headings (splitted by chr(10) or ";") in GUI are enabled by default by

```
 _aGlobSetting[GSET_G_L_TBROW_MULTIHEAD] := .T. // .F. = single line
```

which is equivalent to oTbr:HeadStyle(.T.)

To emulate row/page movement by mouse click on the [^] and [v] vertical scrollbar button in GUI mode, use

```
  _aGlobSetting[GSET_G_A_TBROW_VSBUTT] := {{1,2,2,2}, ;  // [1]
                                          {2,2,2,2}, ;  // [2]
                                          {2,2,2,2}, ;  // [3]
                                          .T.}          // [4]
```

where the sub-array [1] assigns action for left mouse click, [2] for mid mouse click and [3] for right mouse click. The four numeric elements are mouse click modifiers: {plain click, +Shift, +Ctrl, +Alt}. Value of 0 specify no action, 1 = line up/down, 2 = page up/down. The .T. in element [4] says to emulate Up/Down/PgUp/PgDn key, while .F. process the movement directly.

Example 1:

This example demonstrates many of the TBROWSE facilities. It will browse and sort a given directory, using different color settings for each column.

```
*** test.prg, compile: FlagShip test.prg -na -Mmain
STATIC elem := 1

function main()                    // entry point
browsedir ("*")                    // display all

#include "box.fh"
#include "inkey.fh"

FUNCTION browsedir (skeleton)
*****************************
LOCAL brow  := TBROWSENEW ()       // create empty TBCOLUMN
LOCAL ii, sort, column, dir

dir := DIRECTORY(skeleton)
IF LEN(dir) = 0
   ? CHR(7) + "No directory entries for '" + skeleton + "'"
   RETURN 0
ENDIF

SET COLOR TO "W+/B,N/W"

brow:NTOP    := 5
brow:NLEFT   := 10
brow:NBOTTOM := MAXROW() -5
brow:NRIGHT  := MAXCOL() -10
brow:SKIPBLOCK := {|input, obj, temp| temp := elem,   ;
                    elem := MAX(1, MIN(LEN(dir), ;
                    elem + input)), elem - temp }
* brow:GOTOPBLOCK:= {||elem := 1 }        // not used here
* brow:GOBOTTOMBL:= {||elem := LEN(dir)}  // not used here
brow:COLSEP := " | "
brow:HEADSEP:= "Ä┼Ä"
brow:COLORSPEC := "W/B, W+/B, BG+/W, GR+/B, R+/B, N/W"
brow:FOOTSEP:= "Ä┴Ä"
*** Create columns
browdircolumn (brow, dir)             // see chapter 4

*** draw box around TBROWSE region
@ brow:NTOP -1, brow:NLEFT -1, brow:NBOTTOM +3, ;
   brow:NRIGHT +1  BOX B_DOUBLE + " "         // draw box
@ brow:NBOTTOM +1, brow:NLEFT  TO ;
  brow:NBOTTOM +1, brow:NRIGHT                // draw line
@ brow:NBOTTOM +2, brow:NLEFT SAY " Move: " + ;
  "Cursor or PgUp,PgDn.   Sort: select column, press S"

*** main loop to browse and perform an user action
WHILE (.T.)
   IF LEN(dir) > brow:ROWCOUNT
      WHILE ! brow:STABILIZE() .and. NEXTKEY() == 0
      ENDDO
   ELSE
```

```
      brow:FORCESTABLE()
   ENDIF
   key := INKEY(0)                       // get key pressed

   DO CASE
   CASE key = K_LEFT                     // left
      brow:LEFT()
   CASE key = K_RIGHT                    // right
      brow:RIGHT()
   CASE key = K_UP                       // up
      brow:UP()
   CASE key = K_DOWN                     // down
      brow:DOWN()
   CASE key = K_PGUP                     // PgUp
      brow:PAGEUP()
   CASE key = K_PGDN                     // PgDown
      brow:PAGEDOWN()
   CASE key = K_ESC .or. key = K_ENTER  // ESC or ENTER
      EXIT
   CASE UPPER(CHR(key)) == "S"    // sort on current column
      sort = brow:COLPOS
      ASORT (dir, , , {|x,y| x[sort] <= y[sort] })
      FOR ii = 1 TO 5
         column = brow:GETCOLUMN (ii)
         column:FOOTING := IF (ii==sort, "sorted", "")
         brow:SETCOLUMN (ii, column)
      NEXT
      brow:REFRESHALL()
   OTHERWISE
      ?? CHR(7)
   ENDCASE
ENDDO
RETURN elem

* function browdircolumn (brow, dir)  // see TBCOLUMNNEW()
*  :
* RETURN
```

Example 2:

See also the *<FlagShip_dir>/system/dbedit.prg* file for a complete example of the TBROWSE usage.

Classification:

programming

Class:

TBROWSE class, prototyped in <FlagShip_dir>/include/tbrclass.fh

Compatibility:

Available in FS4, C5 and VO. The alternative syntax 2 and the possibility of inheriting it into an own subclass is available in FlagShip only. Arguments <expL5> to <expN9> are new in FS5.

Related:

TBCOLUMN, ACHOICE(), DBEDIT(), MEMOEDIT()

TbrowseArr ()

Syntax:

```
obj = TbrowseArr ([expN1], [expN2], [expN3],
                  [expN4], [expL5], [expL6], [expC7],
                  [expO8], [expN9], [expA10])
```

Purpose:

Creates a new TBROWSE object with predefined array movement blocks, optionally initialized by the arguments supplied.

Options:

<**expN1**> is the top screen row where TBROWSE is displayed. This argument is equivalent to assigning the obj:NTOP with the same value. The valid range is 0...MAXROW(). The default value is zero.

<**expN2**> is the leftmost screen column where TBROWSE is displayed. This argument is equivalent to assigning the obj:NLEFT with the same value, the valid range is 0...MAXCOL(). The default value is zero.

<**expN3**> is the bottom screen row where TBROWSE is displayed. This argument is equivalent to assigning the obj:NBOTTOM with the same value. The valid range is <expN1>...MAXROW(). The default value is MAXROW().

<**expN4**> is the rightmost screen column TBROWSE is displayed. This argument is equivalent to assigning the obj:NRIGHT with the same value. The valid range is <expN2>...MAXCOL(). The default value is MAXCOL().

<**expL5**> is the pixel specification. If .T., the coordinates given are assumed in pixel. If .F., the coordinates are in row/column. If not given or NIL, the current SET PIXEL is considered.

<**expL6**> specifies whether the Tbrowse widget is resizeable by user or not. Default is .F. which means the Tbrowse widget is fix. Applies for GUI mode only, ignored otherwise.

<**expC7**> is a ToolTip string. Applies for GUI mode only, ignored otherwise.

<**expO8**> is a Font object. If not specified, the oApplic:Font is used. Applies for GUI mode, ignored otherwise.

<**expN9**> specify the row height in pixel, see also tb:RowHeight. If not specified, the size of one row is used. Applies for GUI mode, ignored otherwise.

<**expA10**> is the array to browse. If nor specified, the array need to be assigned by tb:UserArray

Returns:

<**obj**> is the newly allocated TBROWSE object, usually assigned to a regular FlagShip variable.

Description:

TbrowseArr() is similar to the generic TBROWSENEW(), but will already predefine tb:SKIPBLOCK, tb:GOTOPBLOCK and tb:GOBOTTOMBLOCK and is working on the specified array.

If the optional arguments are supplied, the corresponding instance variables are filled with these values.

Prior to using the TBROWSE object, at least one or more TBCOLUMNs (see tb:ADDCOLUMN()) must be specified.

Tuning:

See TbrowseNew()

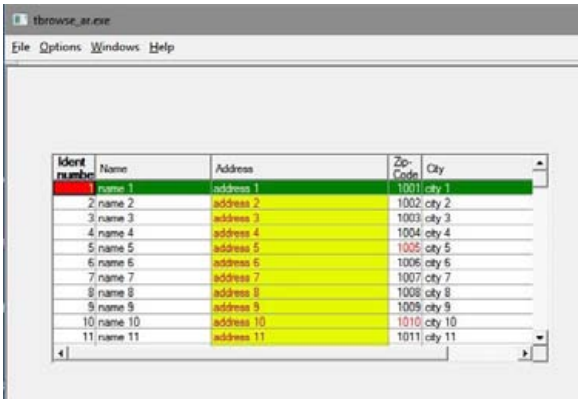
Example 1:

Browse thru multi-dimensional array 'myArray', uses the default keyboard handler defined in .../system/tbrowsearrhand.prg. This is an extract from the example in *<FlagShip_dir>/examples/tbrowse_ar.prg*

```
oBr := TbrowseArr(6,5, 20,60, NIL, NIL, "My Browse")
oBr:UserArray := myArray  // assign array
* oBr:ReadOnly  := .T.    // enable if editing is not desired
* oBr:CanAppend := .F.    // if append is not desired, def. is .T.
for ii := 1 to len(myArray[1])
   oTbcol := TbColumnNew(aHeader[ii], .T.) // use def. array block
   if ii == 1
      oTbcol:Picture := "9999"
      oTbcol:ReadOnly := .T.            // 1st column not editable
   elseif ii == 4
      oTbcol:Picture := "999999"
   endif
   oBr:AddColumn(oTbcol)
next
* oBr:Trim := .T.           // optional, trim displayed char data
oBr:Exec()                  // calls UDF assigned by :Handler
```

Example 2:

complete example is in *<FlagShip_dir>/examples/tbrowse_ar.prg*



Classification:

programming

Class:

uses TBROWSE class, prototyped in <FlagShip_dir>/include/ tbrclass.fh

Compatibility:

Available in FS5 only.

Source:

Source is available in *<FlagShip_dir>/system/tbrowsearr.prg* and in *<FlagShip_dir>/system/tbrowsehand.prg*

Related:

TbrowseNew(), TbrowseDb(), TbColumn

TbrowseDB ()

Syntax:

```
obj = TbrowseDb ([expN1], [expN2], [expN3],
                 [expN4], [expL5], [expL6], [expC7],
                 [expO8], [expN9])
```

Purpose:

Creates a new TBROWSE object with predefined database movement blocks, optionally initialized by the arguments supplied.

Options:

<**expN1**> is the top screen row where TBROWSE is displayed. This argument is equivalent to assigning the obj:NTOP with the same value. The valid range is 0...MAXROW(). The default value is zero.

<**expN2**> is the leftmost screen column where TBROWSE is displayed. This argument is equivalent to assigning the obj:NLEFT with the same value, the valid range is 0...MAXCOL(). The default value is zero.

<**expN3**> is the bottom screen row where TBROWSE is displayed. This argument is equivalent to assigning the obj:NBOTTOM with the same value. The valid range is <expN1>...MAXROW(). The default value is MAXROW().

<**expN4**> is the rightmost screen column TBROWSE is displayed. This argument is equivalent to assigning the obj:NRIGHT with the same value. The valid range is <expN2>...MAXCOL(). The default value is MAXCOL().

<**expL5**> is the pixel specification. If .T., the coordinates given are assumed in pixel. If .F., the coordinates are in row/column. If not given or NIL, the current SET PIXEL is considered.

<**expL6**> specifies whether the Tbrowse widget is resizeable by user or not. Default is .F. which means the Tbrowse widget is fix. Applies for GUI mode only, ignored otherwise.

<**expC7**> is a ToolTip string. Applies for GUI mode only, ignored otherwise.

<**expO8**> is a Font object. If not specified, the oApplic:Font is used. Applies for GUI mode, ignored otherwise.

<**expN9**> specify the row height in pixel, see also tb:RowHeight. If not specified, the size of one row is used. Applies for GUI mode, ignored otherwise.

Returns:

<**obj**> is the newly allocated TBROWSE object, usually assigned to a regular FlagShip variable or to an array element.

Description:

TBROWSEDB() is similar to the generic TBROWSENEW(), but will already predefine tb:SKIPBLOCK, tb:GOTOPBLOCK and tb:GOBOTTOMBLOCK.

If the optional arguments are supplied, the corresponding instance variables are filled with these values.

Prior to using the TBROWSE object, at least one or more TBCOLUMNs (see tb:ADDCOLUMN()) must be specified.

Tuning:

See TbrowseNew()

Example 1:

Browse through the database 'mydata.dbf', uses the default keyboard handles defined in *<FlagShip_dir>/system/tbrowsedbhand.prg*. This is an extract from the example in *<FlagShip_dir>/examples/tbrowse_db.prg*

```
use mydada SHARED NEW
@ 5,4,21,61 box B_PLAIN color ("gr+/b")   // for Terminal i/o mode
oBr := TbrowseDb(6,5, 20,60, NIL, NIL, "My Browse")

for ii := 1 to Fcount()
   oBr:AddColumn( TbColumnNew(FieldName(ii), ;
                  FieldBlock(FieldName(ii)) ))
next
* oBr:Handler := {|obj| TbrDbHandler(obj) }  // default setting
oBr:Exec()                                   // process browsing
```

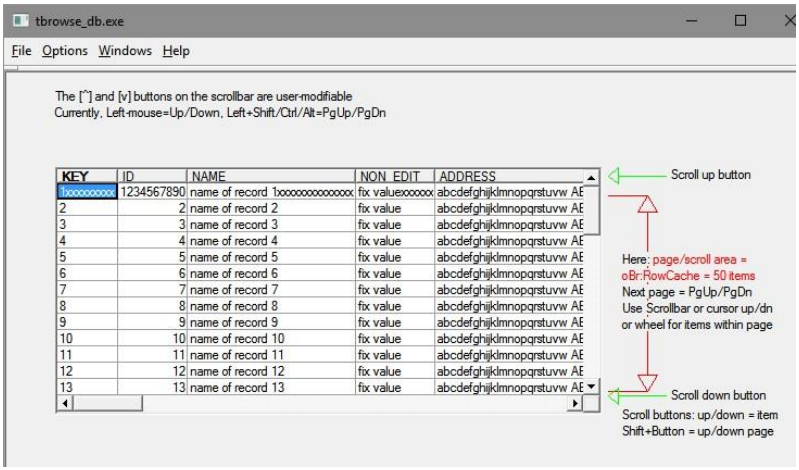
Example 2:

This example demonstrates many of the TBROWSE facilities. It will browse and sort a given directory, using different color settings for each column.

```
USE address INDEX adr1
browsedb()
FUNCTION browsedbf ()
LOCAL mybrow := TBROWSEDB (1,0, MAXROW()-1, MAXCOL())
* mybrow:GOTOPBLOCK := {|| DBGOTOP() }       // predefined
* mybrow:GOBOTTOMBL := {|| DBGOBOTTOM() }    // predefined
* mybrow:SKIPBLOCK  := {|how,obj| skipdb (how) } // see chapt 4

FOR ii = 1 TO FCOUNT()                 // see TbColumn class
   mybrow:ADDCOLUMN (TBCOLUMNNEW (FIELDNAME(ii), ;
                     FIELDBLOCK (FIELDNAME(ii)) )
NEXT

mybrowhandle (mybrow)                  // see chapter 5
RETURN NIL
```

Example 3:

See also the <*FlagShip_dir>/system/dbedit.prg* file for a complete example of the TBROWSE usage. A complete example is available also in <*FlagShip_dir>/ examples/tbrowse_db.prg*



Classification:

programming

Class:

uses TBROWSE class, prototyped in <FlagShip_dir>/include/ tbrclass.fh

Compatibility:

Available in FS4, C5 and VO. Arguments <expL5> to <expN9> are new in FS5.

Source:

Source is available in <FlagShip_dir>/system/tbrowsedb.prg and in <FlagShip_dir>/ system/tbrowsedbhand.prg

Related:

TBROWSENEW(), TBCOLUMN, DBEDIT()

Tbrowse Class Index

Class Tbrowse
Inherits from: -
Inherited by: -
Class prototype: tbrclass.fh
Defines: tbrowse.fh

AddColumn()	METHOD	Add new TbColumn object
ApplyKey()	METHOD	Evaluates the Tb:SetKey() code block
AutoLite	Export	Highlights current cell automatically
AutoRefresh	ACC/ASS	Set/get auto refresh seconds or 0 if none
AutoRefresh()	Method	Process auto refresh each specified seconds
Border	ACC/ASS	Character value drawn around the TBrowse
CanAppend	ACC/ASS	Appending of new records allowed ?
Cargo	Export	Any user data
Col()	METHOD	Column coordinate of currently selected cell
ColAdjust	Export	Adjust large columns (left/centered)
ColCount	ACCESS	Total number of data columns
ColorRect()	METHOD	Alters the color of a rectangular group of cells
ColorSpec	ACC/ASS	Color attribute for the Tbrowse display
ColPos	ACC/ASS	Column number of current selection
ColSep	ACC/ASS	Character value of column separator
ColSepEof	ACC/ASS	Display column separator in empty rows?
ColVisibleCoord()	METHOD	Coordinate of specified visible column
ColVisibleWidth()	METHOD	Returns the really visible column width
ColWidth()	METHOD	Width of specified column
Configure()	METHOD	Reexamine all instances
Data()	METHOD	Get/set cell data
DataChangedBlock	ACC/ASS	Code block returning .T. when database changed
DbAlias	ACCESS	Alias name of the main database
DeHilite()	METHOD	De-highlight current cell
DelColumn()	METHOD	Deletes specified column
Destroy()	METHOD	Destroy Tbrowse object
Down()	METHOD	Moves the Tbrowse cursor down one row
EnableMouseClick	ACC/ASS	Enable/disable mouse click
End()	METHOD	Moves the Tbrowse cursor to the rightmost column
Exec()	METHOD	Process browsing
FootSep	ACC/ASS	Character of column footing separator
ForceStable()	METHOD	Performs a full stabilization
ForceStabl()	METHOD	same as ForceStable()
Freeze	ACC/ASS	Data columns frozen to the left
Font	ACC/ASS	Used font object for Tbrowse
GetColumn()	METHOD	Returns the specified TBCOLUMN object
GoBottomBlock	ACC/ASS	Code block for the tb:GOBOTTOM() method

GoBottom()	METHOD	Moves the data to the last logical record
GoMousePos()	METHOD	Perform mouse related activities on TBrowse
GoTopBlock	ACC/ASS	Code block for the tb:GOTOP() method
GoTop()	METHOD	Moves the data to the first logical record
GuiColorSpec	ACC/ASS	Array of ColorPair objects for color attribute
GuiFontSpec	ACC/ASS	Array of oFont objects
GuiGrid	ACC/ASS	Enables/disables drawing the grid
Handler	ACC/ASS	Codeblock invoking the keyboard/mouse handler
HeadSep	ACC/ASS	Character of column heading separator
HeadStyle()	METHOD	Set/get header style in GUI mode
Hide()	METHOD	Hides Tbrowse until tb:Show()
Hilite()	METHOD	Highlights current cell
HitBottom	ACC/ASS	Attempt to navigate beyond the end-of-data ?
HitTest()	METHOD	Checks if the given coordinates are in Tbrowse
HitTop	ACC/ASS	Attempt to navigate beyond the beg-of-data ?
Home()	METHOD	Moves the Tbrowse cursor to leftmost column
HScrollBar()	ASSIGN	Sets the horizontal scrollbar visibility
IncrSearch	ACC/ASS	Incremental search requested?
InsColumn()	METHOD	Inserts a TBCOLUMN object
Invalidate()	METHOD	Re-draw the entire TBROWSE display at stabil
KillFocus()	METHOD	For @..Get/Read only
Left()	METHOD	Moves the Tbrowse cursor left one data column
LeftVisible	ACCESS	Position of the leftmost unfrozen column
LineCursor	ACC/ASS	internal
McolPos	ACC/ASS	Sets/gets column where mouse cursor is located
Message	ACC/ASS	Message displayed in @..Get/Read
MouseOn()	METHOD	Enable/disable the mouse in GUI Tbrowse
MrowPos	ACC/ASS	Sets/gets row where mouse cursor is located
NBottom	ACC/ASS	Bottom screen row
NBottom()	METHOD	same as NBottom ACC/ASS
NLeft	ACC/ASS	Leftmost screen column
NLeft()	METHOD	same as NLeft ACC/ASS
NRight	ACC/ASS	Rightmost screen column
NRight()	METHOD	same as NRight ACC/ASS
NTop	ACC/ASS	First screen row
NTop()	METHOD	same as NTop ACC/ASS
OnUpdate()	METHOD	Triggers UDF if edited or added data
PageDown()	METHOD	Moves the data one window page downwards
PageUp()	METHOD	Moves the data one window page upwards
PageSkip	ACC/ASS	Redefines behavior of PageDown() and PageUp()
PanEnd()	METHOD	Moves the browse cursor to rightmost column
PanHome()	METHOD	Moves the browse cursor to leftmost column
PanLeft()	METHOD	Moves the browse cursor to left column
PanRight()	METHOD	Moves the browse cursor to right column
ReadOnly	ACC/ASS	Are Tbrowse fields editable?
RefreshAll()	METHOD	Marks all data rows as invalid
RefreshCur()	METHOD	Marks current data row as invalid

RefreshCurrent()	METHOD	same as RefreshCur()
Right()	METHOD	Moves the browse cursor right one data column
RightVisible	ACCESS	Position of the rightmost unfrozen column
Row()	METHOD	Row coordinate of currently selected cell
RowCache	ACC/ASS	Size of the browse cache for page skip
RowCount	ACCESS	Number of visible data rows
RowHeight	ACC/ASS	Sets/gets the height of each row in pixel
RowPos	ACC/ASS	Current row number
ScrollLeft()	METHOD	Scroll view to left
ScrollRight()	METHOD	Scroll view to right
SelectedCol	ACC/ASS	currently selected column
SelectedRow	ACC/ASS	currently selected row
SelectedRecno	ACC/ASS	currently selected record
SelectedValue	ACC/ASS	currently selected value
SetColumn()	METHOD	Replaces the specified Tbcolumn
SetFocus()	METHOD	For @..Get/Read only
SetKey()	METHOD	Set/get code block associated to Inkey value
SetKeyDef()	METHOD	Set default tb:SetKey() redirections
SetStyle()	METHOD	Manage a 1-dimensional array with log flags
Show()	METHOD	Re-display hidden Tbrowse
SkipBlock	ACC/ASS	Code block for tb:DOWN/UP(), tb:PAGEDOWN/UP()
Stabilize()	METHOD	Performs incremental stabilization
Stable	ACC/ASS	Is the TBROWSE object stable?
TimeOut	ACC/ASS	Set/get TimeOut seconds
ToolTip	ACC/ASS	Set/get the tooltip string
Trim	ACC/ASS	Trim character fields
Up()	METHOD	Moves the TBROWSE cursor up one row
UserArray	ACC/ASS	Array with Tbrowse data for TbrowseArr()
UserArrayPos	ACC/ASS	Current row in data-array, used by skipper
Visible	ACCESS	Is Tbrowse visible or hidden?
VScrollBar	ASSIGN	Sets the vertical scrollbar visibility

Tbrowse Class Properties

[tc =] tb:ADDCOLUMN (<expO1>)

Adds a new TBCOLUMN object <**expO1**> to the TBROWSE object and increases the tb:COLCOUNT instance by one. See examples in Chapter 4, TBROWSENEW(), TBROWSEDB() and TbColumn class.

tb:AUTOLITE ***Access/Assign***

Contains a logical value. When set to TRUE (the default), the stabilize method automatically highlights the current cell as part of stabilization.

tb:AUTOREFRESH ***Access/Assign***

Set/get auto-refresh timeout in seconds or 0 if none set. Used in the default DbEdit() handler (see <FlagShip_dir>/system/dbedit.prg) and considered in default TbrowseDb() handler (see <FlagShip_dir>/system/tbrowsedbhand.prg). It can be freely set and used in your own Tbrowse handler too. If you assign numeric value between 1 and 86399 (= 1 second to 24 hours), the oTbr:AutoRefresh() method will check for database changes by other users and refresh Tbrowse display if required. To disable AutoRefresh, assign 0.

<retN> := tb:AUTOREFRESH([<expN|expL>])

Process auto refresh each specified seconds or use previously set value, if any. The tb:Autorefresh() considers changes by other users on the same database and will refresh the display if any changes occured. If <**expN**> is given and numeric, it behaves like tb:AutoRefresh assignment. If <**expL**> is .T. the check is performed regardless tb:AutoRefresh value.

This method is used in DbEdit() (see <FlagShip_dir>/system/dbedit.prg) and default TbrowseDb() handler (see <FlagShip_dir>/system/tbrowsedbhand.prg). It can be freely used in your own Tbrowse handler, as demonstrated in <FlagShip_dir>/ examples/tdbedit.prg

The method may be invoked in the Tbrowse handler on idle status. If the last check + tb:AutoRefresh is greater than current seconds(), it checks the database (if any) for use also by others and if so, checks the .dbf update counter (see DbObject():Info(DBI_INTGRCOUNT)) for changes. If so, it then performs refresh by tb:RefreshAll().

<retN> := tb:APPLYKEY([<expN>])

Evaluates the code block associated with the key in tb:SetKey(<expN>, <codeBlock>) setting. The return value <retN> is passed from the code block's return value, and specifies the manner in which the key should be processed by the handler:

Constant	Value	Meaning
TBR_EXIT	-1	User request for the browse to lose input focus and to exit Tbrowse, ignores corresp. SET KEY
TBR_CONTINUE	0	Key <expN> is set by tb:SetKey() or SetKeyDef(), code block associated with <nExp> was evaluated, do not process default handler action, nor previously set SET KEY
TBR_EXCEPTION	1	Key <expN> not set by tb:SetKey() or SetKeyDef() Evaluate corresponding SET KEY if set, the handler should then process default key action
TBR_DEFACTION	2	Key <expN> is set by tb:SetKey() or SetKeyDef(), the handler should process default key action, but ignores corresponding SET KEY if such set.

The TBR_* constants are available in tbrowse.fh include file. If the code block for <expN> is not set, tb:ApplyKey() returns TBR_EXCEPTION. If the code block returns invalid value, TBR_CONTINUE is returned from tb:ApplyKey().

tb:CANAPPEND ***Access/Assign***

Logical value specifying whether appending of new records is allowed. The default value is TRUE, new records can be appended.

tb:CARGO ***Access/Assign***

Contains any user data of any type to store information retrieved later in the program. Not used by the TBROWSE system itself. The default is NIL.

retN = tb:COL ([expL1])

Returns the coordinate of currently selected column, comparable to COL() function. <expL1> is the pixel specification for GUI. If .T., the return value is in pixel. If .F., <retN> is in row/column. If <expL1> is not given or is NIL, current SET PIXEL is considered.

tb:COLADJUST ***Access/Assign***

Controls adjustment of large columns. If 0 (the default), a large column which does not fit in visible window area, will be centered. Specifying value > 0, the column will be moved so, that at least <value> pixels are visible at the left site, in front of the column. You may achieve the same behavior by setting the global variable

`_aGlobSetting[GSET_G_N_TBROW_COLADJ]:= value` which may preferably be used for Tbrowse wrappers like DbEdit(). Considered in GUI mode, ignored otherwise.

tb:COLCOUNT *Access*

Contains a numeric value indicating the total number of data columns specified in the TBROWSE object using the tb:ADDCOLUMN() method.

[tb =] tb:COLORRECT (<expA1>, <expA2>)

Alters the color of a rectangular group of cells. Applies for Terminal i/o mode only, ignored in GUI mode.

The <**expA1**> is an array of four numeric coordinates (top row, left column, bottom row, and right column) referring to cells within the current TBROWSE data display, not to the physical screen coordinates. The valid range is 1,1...tb:ROWCOUNT, tb:COLCOUNT. The tb:COLORRECT() is stronger than tc:COLORBLOCK on the same coordinates.

The <**expA2**> argument is an array of two numbers, specifying the color index in tb:COLORSPEC for normal and highlighted color.

Such re-colored cells retain the new color until the cells are scrolled down or up out of the screen, or tb:REFRESH*() is executed. Horizontal panning does not change the new coloring. In fact, the currently invisible cells to the left and right can be colored using tb:COLORRECT().

Example for re-coloring all cells to yellow on blue of the entire window, with the exception of the first and last column:

```
tbr:COLORSPEC := "W/B, W+/B, BG+/W, GR+/B, R+/B, N/W"
tbr:COLORRECT ( {1,2, brow:ROWCOUNT, brow:COLCOUNT -1}, {4,3} )
```

An alternative to tb:ColorRect() is 3rd array element of tc:ColorBlock() and tc:GuiColorBlock() which highlights current Tbrowse row.

tb:COLORSPEC *Access/Assign*

Contains a character string defining a color attribute for the TBROWSE display. Unlike SET COLOR TO, you may specify as many attributes as you require. Each color attribute is internally stored in an array element, whose index is used in tc:COLORBLOCK. When TBROWSE is being created, the current SETCOLOR() value is copied into tb:COLORSPEC. Applicable for Terminal i/o mode only, ignored in GUI where the tb:GUICOLORSPEC may be used instead.

Example to set standard display yellow on blue and the highlight bar red on cyan:

```
tb = TBROWSEDB()
tc = TBCOLUMNNEW("Name", {|| FIELD->name})
tb:COLORSPEC := SETCOLOR() + "W+/B, R+/BG"  // total 5+2=7 elements
tc:COLORBLOCK := {|x| {6,7}}                // use element 6 and 7
tbr:ADDCOLUMN(tc)
```

See <FlagShip_dir>/examples/tbrowse_ar.prg for complete example

tb:COLPOS *Access/Assign*

Contains a numeric value (starting at one) indicating the data column where the TBROWSE cursor is currently located. On assignment, only columns of the currently visible area are accepted. When you need to move to invisible column, use tb:right() or tb:left(), for example to display the 12th column automatically in visible area:

```
if tb:rightvisible >= 12
   while tb:leftvisible > 12 .and. tb:rightvisible > 12
      tb:left()
   enddo
elseif tb:colcount >= 12
   while tb:leftvisible < 12 .and. tb:rightvisible < 12
      tb:right()
   enddo
endif
tb:colpos := 12
tb:refreshall() ; tb:forcestable()
```

tb:COLSEP *Access/Assign*

Contains a character value that defines a column separator for TBCOLUMN which does not containing a column separator of its own. The default is one space. Applicable for Terminal mode only, ignored in GUI mode where tb:GUIGRID may be used instead.

tb:COLSEPEOF *Access/Assign*

Contains logical value that specifies whether column separators should be displayed even if the row is empty, i.e. for not available data. The default is .T.

retN = tb:COLVISIBLECOORD ([expN1], [expL2])

Returns the coordinate of currently visible column <**expN1**>. If <expN1> is not given or is NIL, currently selected column is used.

<**expL2**> is the pixel specification for GUI. If .T., the returned value is in pixel. If .F., <retN> is in row/column. If <expL2> is not given or is NIL, current SET PIXEL is considered.

If the returned value is negative (usually -999), the given column is currently invisible, i.e. <expN1> is not in range tb:LEFTVISIBLE to tb:RIGHTVISIBLE. If the leftmost column is only partially visible, the returned <retN> value is lower than tb:NLEFT.

retN = tb:COLVISIBLEWIDTH ([expN1], [expL2])
retN = tb:COLVISIBLEWIDTH ([expL2])

Returns the visible width of column number **<expN1>** or of current column if <expN1> is not given or is NIL. If <retN> is negative, the column is currently invisible. If <retN> is less than tb:COLWIDTH(<expN1>), the column is only partially visible.

<**expL2**> is the pixel specification for GUI. If .T., the returned value is in pixel. If .F., <retN> is in row/column. If <expL2> is not given or is NIL, current SET PIXEL is considered.

retN = tb:COLWIDTH ([expN1], [expL2])

Returns the display width of column number <**expN1**> or of current column if <expN1> is not given or is NIL. If <expN1> is out of the valid range 1...tb:COLCOUNT, <retN> is <= 0

<**expL2**> is the pixel specification for GUI. If .T., the returned value is in pixel. If .F., <retN> is in row/column. If <expL2> is not given or is NIL, current SET PIXEL is considered.

[tb =] tb:CONFIGURE ()

Causes the TBROWSE object to reexamine all instance variables and TBCOLUMN objects, reconfiguring its internal settings as required. This method forces reconfiguration when a TBCOLUMN object is modified directly.

ret = tb:DATA ([expN1], [exp2])

Get or set data of current or specified cell within current row. This method is a shorthand for EVAL((tb:GetColumn(expN1)):Block, exp2)

<**expN1**> is the column number (in range 1 to tb:COLCOUNT). If <expN1> is not given or is NIL, current column is used.

<**exp2**> is optional value to be set. If <exp2> is not given or is NIL, only the current cell value is returned. Otherwise, the valtype() must be equivalent to valtype() of the cell.

<**ret**> is the current (or modified) cell value, NIL signals an error.

tb:DBALIAS ***Access***

Contains the ALIAS name of the selected database at the time of invoking TBrowse or latest at invoking this property, or is "" otherwise. Used in default skippers to support SET RELATION.

[tb =] tb:DEHILITE ()

Causes the current cell (the cell to which the browse cursor is positioned) to be de-highlighted. This method is designed for use when tb:AUTOLITE is set to TRUE (the default).

tc = tb:DELCOLUMN (<expN1>)

Deletes the specified column <**expN1**> from TBROWSE. The returning value is a TBCOLUMN object which can be preserved by assigning the method to a FlagShip variable.

[tb =] tb:DOWN ()

Moves the TBROWSE cursor down one row. If the cursor is already on the bottom row, the display is scrolled up and a new row is brought into view. If the data pointer is already at the logical end-of-file and the browse cursor is already on the bottom row, tb:HITBOTTOM instance is set TRUE.

tb:ENABLEMOUSECLICK ***Access/Assign***

Contins logical value. When set to TRUE (the default), a mouse click selects the corresponding item. You may assign FALSE (.F.) before edit the cell to avoid unintended re-positioning by mouse click, and set .T. when the new value is replaced. Applicable in GUI only, ignored otherwise. See also default handlers <FlagShip_dir>/system/tbrowsehand.prg and tbrowsedbhand.prg

[tb =] tb:END ()

Moves the browse cursor to the rightmost data column currently visible. The highlight bar remains at the same row.

[tb =] tb:EXEC ()

Process browsing using the default or by tb:Handler assigned keyboard handler. Standard handlers (tbrowsehand.prg and tbrowsedbhand.prg) supports following keys and actions:

Key		Action	in READ	
Cursor up	ctrl-E	Up one row	yes	
Cursor down ctrl-X		Down one row or append record	*	yes
Cursor <- ctrl-S		Column left		yes
Cursor -> ctrl-D		Column right		yes
TAB ctrl-H		Scroll right	(next GET)	no
shift-TAB shift-ctrl-H		Scroll left	(prev GET)	no
PgUp ctrl-R		Previous window		yes

PgDn ctrl-C	Next window		yes
Home ctrl-A	Leftmost curr. column		yes
End ctrl-F	Rightmost curr.column		yes
ctrl-Home ctrl-]	First item in window		yes
ctrl-End ctrl-F	Last item in window		yes
ctrl-PgUp ctrl--	First screen row		yes
ctrl-PgDn ctrl-^	Last screen row		yes
Esc	Terminate Tbrowse()	(next GET)	yes
Enter ctrl-M	Edit current cell	**	yes
Mouse-Left-Doubleclick	Edit current cell	** ***	yes
Mouse-Wheel	previous/next row	***	yes
Mouse-Wheel+Shift/Alt/Ctrl	previous/next window	***	yes

* Append only when oTb:CanAppend is .T. and current row = last row
** Available only when oTb:ReadOnly is .F.
*** in GUI mode only

tb:FOOTSEP *Access/Assign*

Contains a character or string which specifies the column footing separator. The string is displayed to the left of the current column, if it is not the first one. The last character of the string is used repetitively for the footing line underlining the column. This tb:FOOTSEP separator is used as default, when a column separator tc:FOOTSEP is not specified. Null-string "" is preset. This completely omits displaying the footing separator. See example in TBCOLUMN class.

[tb =] tb:FORCESTABLE ()

Performs a full stabilization of the TBROWSE, displaying all visible TBROWSE data. It is similar to performing

```
DO WHILE ! tb:STABILIZE()
ENDDO
```

tb:FREEZE *Access/Assign*

Contains a numeric value that defines the number of data columns frozen to the left of the display. Frozen columns are always visible, even when other columns are panned off the display. The default is zero (no frozen columns). Available in Terminal i/o mode only.

tb:FONT *Access/Assign*

The used font object for Tbrowse. If not specified, oApplic:Font is taken. Apply for GUI mode only, ignored otherwise.

tc = tb:GETCOLUMN (<expN1>)

Returns the TBCOLUMN object specified by <**expN1**>.

[tb =] tb:GOBOTTOM ()

Moves the data to the last logical record by evaluating the tb:GOBOTTOMBLOCK code block. The TBROWSE display is refilled with the bottommost available data, the cursor moved to the row containing the last record. The pan position of the window remains unchanged.

tb:GOBOTTOMBLOCK ***Access/Assign***

Contains a code block executed in response to repositioning at the last data element when using the tb:GOBOTTOM() method. One argument <oSelf> is passed to the block. It is the Tbrowse object self which can be e.g. passed to an UDF function, instead of using public variables. The code block body typically contains an index of the last array element, or the result of the database movement with the DBGOBOTTOM() function (predefined when using TBROWSEDB()).

If there is no tb:GOBOTTOMBLOCK assigned, Tbrowse moves forwards using the tb:SKIPBLOCK, which is in the most cases less effective. It is roughly comparable to `GO BOTTOM` vs. `WHILE !eof() ; SKIP ; ENDDO` on databases. If the tb:GOBOTTOM-BLOCK is available, it may be used internally also by other movements.

Compatibility note: Clipper do not pass any argument to the code block. But when you specify (and not use) the tbrowse object as 1st parameter, your source remain backward compatible to Clipper.

[tb =] tb:GOTOP ()

Moves the data to the first logical record by evaluating the tb:GOTOPBLOCK code block. The TBROWSE display is refilled with the topmost available data; the cursor moved to the first row. The pan position of the window remains unchanged.

tb:GOTOPBLOCK ***Access/Assign***

Contains a code block executed in response to repositioning at the first data element when using the tb:GOTOP() method. One argument is passed to the block. It is the Tbrowse object self which can be e.g. passed to an UDF function, instead of using public variables. The code block body typically contains an index of the first array element, or the result of the database movement with the DBGOTOP() function (predefined when using TBROWSEDB()).

If there is no tb:GOTOPBLOCK assigned, Tbrowse moves backwards using the tb:SKIPBLOCK, which is in the most cases less effective. It is roughly comparable to `GO TOP` vs. `WHILE !bof() ; SKIP -1 ; ENDDO` on databases. If the tb:GOTOPBLOCK is available, it may be used internally also by other movements.

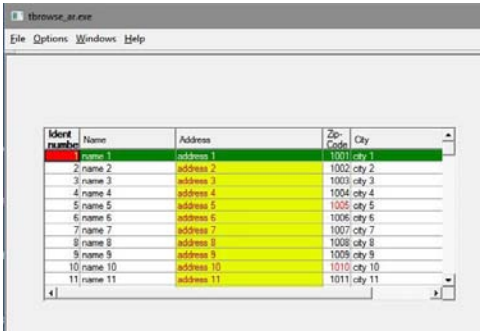
Compatibility note: Clipper do not pass any argument to the code block. But when you specify (and not use) the tbrowse object as 1st parameter, your source remain backward compatible to Clipper.

tb:GUICOLORSPEC *Access/Assign*

GuiColorSpec is similar to ColorSpec property for Terminal i/o mode. It contains an array of ColorPair objects or color strings (see SET COLOR) defining the color attribute for the TBROWSE display in GUI mode. You may specify as many attributes as you require. The corresponding color element is used in tc:GUICOLORBLOCK. Applicable for GUI i/o mode only, ignored in Terminal i/o, where the tb:COLORSPEC may be used instead. Example:

```
tb := TBROWSEDB()
tb:GUICOLORSPEC := {"N/W+", ;
                    ColorPair{Color{255,0,0},Color{0,127,127}}, ;
                   "#FFFFFF/#E5F902" }
tc := TBCOLUMNNEW("Name", {|| FIELD->name})
tc:GUICOLORBLOCK := {|x| {1,2}}             // use element 1 and 2
tb:ADDCOLUMN(tc)
```

See <*FlagShip_dir*>*/examples/tbrowse_ar.prg* for complete example



tb:GUIGRID *Access/Assign*

Enables/disables drawing the grid = separator between columns and rows. Applicable for GUI mode only, ignored otherwise, where the tb:COLSEP apply.

tb:HANDLER *Access/Assign*

Retrieve or assign a codeblock invoking the keyboard handler when the tb:Exec() method is called. The codeblock receives one argument, the Tbrowse object self. The default codeblock depends on the Tbrowse instantiation and is either TbrHandler() available in source in <FlagShip_dir>/system/tbrowsehand.prg, or the TbrDbHanler() for database access, available in .../system/tbrowsedbhand.prg

tb:HEADSEP **Access/Assign**

Contains a character or string which specifies the column heading separator. The string is displayed to the left of the current column, if it is not the first one. The last character of the string is used repetitively for the heading line displayed over the column. The tb:HEADSEP is used as default, when a column separator tc:HEADSEP is not specified. Null-string "" is preset, which completely omits the display of the heading separator. See example in TBCOLUMN class and TBROWSENEW().

[retA =] tb:HEADSTYLE ([<expL1>], [<expN2>], [<expN3>])

Sets and/or gets header style in GUI mode, ignored otherwise. All Tbrowse columns are dispayed in the same style.

<**expL1**> is optional logical value, where .T. sets the header to multi- line mode (separated by ";" or chr(10) in columns header string), .F. displays single-line Tbrowse header. See "Tuning" for defaults and global settings.

<**expN2**> is optional numeric value specifying horizontal adjustment in multi-line mode. The constant is defined in tbrowse.fh

Constant Va	lue	Meaning
TBR_HEAD_LEFT	0	Align header left (default)
TBR_HEAD_RIGHT	1	Align header right
TBR_HEAD_HCENTER	2	Center horizontal

<**expN3**> is optional numeric value specifying vertical adjustment in multi-line mode. The constant is defined in tbrowse.fh

Constant Va	lue	Meaning
TBR_HEAD_VCENTER	0	Center verical (default)
TBR_HEAD_TOP	1	Align at top
TBR_HEAD_BOTTOM	2	Align at bottom

<**retA**> returns current setting as an array with three elements, corresponding to <expL1>, <expN2> and <expN3> of above description.

[tb =] tb:HILITE ()

Causes the current cell (the cell to which the browse cursor is positioned) to be highlighted. This method is designed for use when tb:AUTOLITE is set to FALSE.

tb:HITBOTTOM **Access/Assign**

Contains a logical value indicating whether an attempt was made to navigate beyond the end of the available data. Normally, the value contains FALSE. During stabilization, the value is set TRUE if it was unable to skip forward as many records as requested.

tb:HitTest(nMouseRow, nMouseCol, [lPixel]) —> nStatus

Determines if the mouse cursor is within the region of the screen that the Tbrowse occupies. Applicable in GUI mode only.

<**nRow**> Numeric value representing the current or tested screen row position of the mouse cursor.

<**nCol**> Numeric value representing the current or tested screen row position of the mouse cursor.

<**lPixel**> If specified TRUE, the mouse coordinates are assumed in pixel. If FALSE, the mouse parameters are assumed in current row/col coordinates. If this parameter is not specified (i.e. NIL), the kind of passed mouse coordinates is determined from the current SET PIXEL setting.

<**nStatus**> Returned numeric value indicating the relationship of the mouse cursor with the Tbrowse. The constants are specified in button.fh header file.

Value	Constant	The mouse cursor is ...
0	HTNOWHERE	not within the region of the screen that the Tbrowse occupies
-1	HTTOPLEFT	on the top left corner of the Tbrowse border
-2	HTTOP	on Tbrowse top border
-3	HTTOPRIGHT	on the top right corner of the Tbrowse border
-4	HTRIGHT	on Tbrowse right border
-5	HTBOTTOMRIGHT	on the bottom right corner of Tbrowse border
-6	HTBOTTOM	on Tbrowse bottom border
-7	HTBOTTOMLEFT	on the bottom left corner of Tbrowse border
-8	HTLEFT	on Tbrowse left border
-5121	HTCELL	on any of Tbrowse data cell
-5122	HTHEADING	on Tbrowse heading
-5124	HTHEADSEP	on Tbrowse heading separator
-5131	HTVSCROLLBAR	on Tbrowse vertical scrollbar
-5132	HTHSCROLLBAR	on Tbrowse horizontal scrollbar
-5133	HTVSCROLLBARUP	on Tbrowse vertical scrollbar, button up
-5134	HTVSCROLLBARDOWN	on Tbrowse vertical scrollbar, button down
-5135	HTHSCROLLBARLEFT	on Tbrowse horizontal scrollbar, button left
-5136	HTHSCROLLBARRIGHT	on Tbrowse horizontal scrollbar, button right
-5137	HTSCROLLBAREDGE	on Tbrowse scrollbar, button at bottom right

tb:HITTOP ***Access/Assign***

Contains a logical value indicating whether an attempt was made to navigate past the beginning of the available data. Normally, the value contains FALSE. During stabilization, the value is set TRUE if it was unable to skip backward as many records as requested.

[tb =] tb:HOME ()

Moves the browse cursor to the leftmost unfrozen data column. The high-light bar remains at the same row.

tb:HSCROLLBAR *Assign*

Change visibility of horizontal scrollbar in GUI mode, default s true.

tb:INCRSEARCH *Access/Assign*

Contains logical value indicating whether an incremental search should apply at character input or not. The default is .F. = disabled search. The incremental search allows you to search for any character data in the current index by simply typing the requested data. Implemented for the default handler assigned with TbrowseDb(). You may re-implement the <FlagShip_dir>/system/tbrowsedbhand.prg in your source when non-standard search is required.

[tc =] tb:INSCOLUMN (<expN1>, <expO2>)

Inserts a TBCOLUMN object <**expO2**> at the specified position <**expN1**>. Unlike tb:ADDCOLUMN(), which adds columns at the end, tb:INSERTCOLUMN() inserts new columns anywhere in the TBROWSE.

[tb =] tb:INVALIDATE ()

Invoking this method causes the next stabilization to re-draw the entire TBROWSE display, including headings, footings and all data rows. This method does not refresh the visible data. This can be performed using the tb:REFRESHALL() method.

[tb =] tb:LEFT ()

Moves the browse cursor left one data column. If the cursor is on the leftmost displayed column, the display is horizontally scrolled to bring the previous data column (if there is one) into view, similar to tb:PANLEFT().

tb:LEFTVISIBLE *Access*

Contains a numeric value indicating the position of the leftmost unfrozen column visible in the browse display. If all columns are frozen, the value contains zero, one otherwise.

tb:NBOTTOM *Access/Assign*
retN = tb:NBOTTOM ([expN1], [expL2])

Contains a numeric value specifying the bottom screen row where the TBROWSE is displayed. The value is preset by arguments of TBROWSENEW() or TBROWSEDB(). If not specified, the default is MAXROW().

The tb:NBOTTOM() method is generalized tb:NBOTTOM acc/assign property. It returns the tb:NBOTTOM value, where <expN1> is optional new bottom screen row and <expL2> is pixel specification for GUI. If <expL2> is .T., the <expN1> and the return value are in pixel. If <expL2> is .F., <expN1> and <retN> are in row/column. If <expL2> is not given, current SET PIXEL is considered, same as in tb:NBOTTOM access.

tb:NLEFT *Access/Assign*
retN = tb:NLEFT ([expN1], [expL2])

Contains a numeric value specifying the leftmost screen column where the TBROWSE is displayed. The value is preset by arguments of TBROWSENEW() or TBROWSEDB(). If not specified, the default is zero.

The tb:NLEFT() method is generalized tb:NLEFT access/assign property. It returns the tb:NLEFT value, where <expN1> is optional new leftmost screen column and <expL2> is pixel specification for GUI. If <expL2> is .T., the <expN1> and the return value are in pixel. If <expL2> is .F., <expN1> and <retN> are in row/column. If <expL2> is not given, current SET PIXEL is considered, same as in tb:NLEFT access.

tb:NRIGHT *Access/Assign*
retN = tb:NRIGHT ([expN1], [expL2])

Contains a numeric value specifying the rightmost screen column where the TBROWSE is displayed. The value is preset by arguments of TBROWSENEW() or TBROWSEDB(). If not specified, the default is MAXCOL().

The tb:NRIGHT() method is generalized tb:NRIGHT access/assign property. It returns the tb:NRIGHT value, where <expN1> is optional new rightmost screen column and <expL2> is pixel specification for GUI. If <expL2> is .T., the <expN1> and the return value are in pixel. If <expL2> is .F., <expN1> and <retN> are in row/column. If <expL2> is not given, current SET PIXEL is considered, same as in tb:NRIGHT access.

tb:NTOP *Access/Assign*
retN = tb:NTOP ([expN1], [expL2])

Contains a numeric value specifying the first screen row where the TBROWSE is displayed. The value is preset by arguments of TBROWSENEW() or TBROWSEDB(). If not specified, the default is zero.

The tb:NTOP() method is generalized tb:NTOP access and assign property. It returns the tb:NTOP value, where <expN1> is optional new topmost screen row and <expL2> is pixel specification for GUI. If <expL2> is .T., the <expN1> and the return value are in pixel. If <expL2> is .F., <expN1> and <retN> are in row/column. If <expL2> is not given, current SET PIXEL is considered, same as in tb:NTOP access.

[expB =] tb:ONUPDATE [:= expB] ***Access/Assign***

Triggers an UDF from the default handler for edited or added data. <expB> is a code block receiving two arguments, the Tbrowse object and the curent changed mode: 1 = new record appended, 2 = value chanded, 3 = data changed after append. Example:

```
tb := TbrowseDb(...)  // or TbrowseArr(...) or TbrowseNew(...)
cbID   := {|| ORDERS->ID }
cbName := {|val| if(val == NIL, ORDERS->Name, ORDERS->Name := val}
tb:AddColumn := TbColumnNew("ID-Num", cbID)  // uses r/o codeblock
tb:AddColumn := TbColumnNew("Name", cbName)  // uses r/w codeblock
tb:OnUpdate := {|obj,mode| MySaveRec(obj,mode)}
tb:Exec()
...
/* ----------------------------------------------------------
* MySaveRec() triggered by tb:OnUpdate{...}
* Note: the REPLACE is done automatically by default handler
*/
Function MySaveRec(oTbr,mode)
local iColNum := oTbr:ColPos         // Current column number
local oTbCol  := oTbr:GetColumn(iColNum)
local data    := Eval(oTbCol:Block)  // Retrieve current data

alert("Saving changed data in column " + trim(ltrim(iColNum)) + ;
      ";new value = " + trim(ltrim(data)) + ;
      if(oTbr:SelectedRecno > 0, ;
         ";for record# " + ltrim(oTbr:SelectedRecno), ;
         ";for record with ID = " + ;
         trim(ltrim(Eval(oTbr:GetColumn(1):Block))) )  )
return
```

[tb =] tb:PAGEDOWN ()

Moves the data one window page downwards skipping tb:ROWCOUNT records (from the first visible row) by evaluating the tb:SKIPBLOCK code block. The cursor remains on the same row if possible or is moved to the last row containing data. If the end-of-data is reached. When issuing the stabilization method, the TBROWSE display is refilled with the bottommost available data and tb:HITBOTTOM is set TRUE.

[tb =] tb:PAGEUP ()

Moves the data one window page upwards skipping tb:ROWCOUNT records (from the first visible row) by evaluating the tb:SKIPBLOCK code block. The cursor remains on

the same row. If the logical first data record is already shown, the cursor is set to the first row and tb:HITTOP is set TRUE during the invocation of a stabilizing method.

[expN =] tb:PAGESKIP [:= expN] **Access/Assign**

Redefines the bahavior of PageDown() and PageUp(). Considered in GUI mode only.

<expN> =	0:	default skip = tb:ROWCACHE if set, otherwise tb:ROWCOUNT
	-1:	skip by visible rows/page = tb:ROWCOUNT
	-2:	skip by half visible rows/page = tb:ROWCOUNT/2
	nn:	skip by specified rows, tb:ROWCOUNT/2 <= nn <= tb:ROWCACHE

[tb =] tb:PANEND ()

Moves the browse cursor to the rightmost data column, causing the display to be panned completely to the right.

[tb =] tb:PANHOME ()

Moves the browse cursor to the leftmost data column, causing the display to be panned completely to the left.

[tb =] tb:PANLEFT ()

Pans the display without changing the browse cursor. If a left column is available, the screen is scrolled horizontally right to display a new left column. As opposed to tb:LEFT(), tb:PANLEFT() will always scroll the columns (if possible) and does not move the cursor to the left column.

[tb =] tb:PANRIGHT ()

Pans the display without changing the browse cursor. If a right column is available, the screen is scrolled horizontally left to display a new right column. As opposed to tb:RIGHT(), tb:PANRIGHT() will always scroll the columns (if possible and the available columns are not frozen) and does not move the cursor to the right column.

tb:READONLY **Access/Assign**

Logical value specifying that all fields of Tbrowse can or cannot be edited. The default value is FALSE, the fields are editable. See also oTbColumn:READONLY for column setting.

[tb =] tb:REFRESHALL ()

Internally marks all data rows as invalid, causing them to be refilled and redisplayed at the next stabilization.

[tb =] tb:REFRESHCURRENT ()

Internally marks the current data row as invalid, causing it to be refilled and redisplayed at the next stabilization.

[tb =] tb:RIGHT ()

Moves the browse cursor right one data column. If the cursor is on the rightmost displayed column, the display is horizontally scrolled to bring the next data column (if there is one) into view, similar to tb:PANRIGHT().

tb:RIGHTVISIBLE ***Access***

Contains a numeric value indicating the position of the rightmost unfrozen column visible in the browse display. If all columns are frozen, the value contains zero, tb:COLCOUNT otherwise.

retN = tb:ROW ([expL1])

Returns row coordinate of currently selected cell, comparable to ROW() function. <expL1> is the pixel specification for GUI. If .T., the return value is in pixel. If .F., <retN> is in row/column. If <expL2> is not given or is NIL, current SET PIXEL is considered.

tb:ROWCACHE ***Access/Assign***

Numeric value specifying the size of the browse cache. If this value is greater than the number of visible rows (tb:RowCount), the cache is filled at once and the cached area can be scrolled by mouse using the vertical scrollbar or the Cursor Up/Down key. The default cache size is the number of visible rows. You may set the tb:RowCache higher (e.g. to 10 * tb:RowCount) to be able to skip faster thru the database via PgDn or PdUp key. Applies in GUI mode only, ignored otherwise.

Note: for a small database (or an array), you may specify

```
tb:RowCache := reccount()   -or-   tb:RowCache := LEN(myArray)
```

to be able to scroll thru the whole table (database or array) via the vertical scrollbar. Keep in mind, all the data from the RowCache size must be hold in memory, so consider the memory use and the Tbrowse speed for refreshing of large tables (with thousands or millions of records); usually only the current <RowCache> slice of the table needs to be refreshed until the next table slice is read on user request (e.g. by the PgUp/PgDn key press).

tb:ROWCOUNT ***Access***

Contains a numeric value indicating the number of data rows visible in the TBROWSE display. Heading and footing lines are not included in that value.

tb:ROWPOS *Access/Assign*

Contains a numeric value indicating the data row where the TBROWSE cursor is currently located. The valid range is 1.. ..tb:ROWCOUNT.

[tc =] tb:SCROLLLEFT ([expL1])

Scroll the view to left, if possible. If <expL1> is .T. or NIL or not given, the leftmost column is "protected", i.e. displayed at rightmost position after scroll, if possible. Otherwise the new rightmost position is the old leftmost visible column +1.

[tc =] tb:SCROLLRIGHT ([expL1])

Scroll the view to right, if possible. If <expL1> is .T. or NIL or not given, the rightmost column is "protected", i.e. displayed at leftmost position after scroll, if possible. Otherwise the new leftmost position is the old rightmost visible column +1.

tb:SELECTEDCOL *Access/Assign*

Contains currently selected column (1..n). Equivalent to tb:COLPOS but is available also after exit from tb:EXEC(). This value is not set by the class self, but by the handler (per default tbrowsedbhand.prg or tbrowsehand.prg).

tb:SELECTEDROW *Access/Assign*

Contains currently selected row (1..nBuff). Equivalent to tb:ROWPOS but is available also after exit from tb: EXEC(). This value is not set by the class self, but by the handler (per default tbrowsedbhand.prg or tbrowsehand.prg).

tb:SELECTEDRECNO *Access/Assign*

Contains currently selected record number or the array index (1..n). Equivalent to Recno() or tb:UserArrayPos but is available also after exit from tb: EXEC(). This value is not set by the class self, but by the handler (per default tbrowsehand.prg or tbrowsedbhand.prg).

tb:SELECTEDVALUE *Access/Assign*

Contains the value of currently selected item. Equivalent to tb:Data() but is available also after exit from tb:Exec(). This value is not set by the class self, but by the handler (per default tbrowsehand.prg or tbrowsedbhand.prg).

[tc =] tb:SETCOLUMN (<expN1>, <expO2>)

Replaces the column <**expN1**> with the TBCOLUMN object <**expO**2>. The returned value of <**tc**> is the old TBCOLUMN object.

[tc =] tb:SETFOCUS (<expL>)

Considered and used in @..GET...Tbrowse only in getsys.prg

[<expB>] := tb:SETKEY (<expN>, [<expB>])

Set/get a code block <expB> associated to Inkey value <expN> for this Tbrowse object. It is similar to standard SET KEY command or SetKey() function, but tb:SetKey() re-direction do not interferes previously set SET KEY. It is used in default Tbrowse handlers (tbrowsehand.prg and tbrowsedbhand.prg) or handled by tb:ApplyKey() method. This allows to keep and handle SET KEY values in your program independent of Tbrowse handling. The codeblock receives 2 parameters, current Tbrowse object and key value. The code block is evaluated by tb:ApplyKey(). The code block should return:

Constant	Value	Meaning
TBR_EXIT	-1	User request for the browse to lose input focus and to exit Tbrowse, ignores corresp. SET KEY
TBR_CONTINUE	0	Code block associated with <nExp> was evaluated, do not process default handler action, nor previously set SET KEY
TBR_EXCEPTION	1	Evaluate corresponding SET KEY if set, the handler should then process default key action
TBR_DEFACTION	2	The handler should process default key action, but ignores corresponding SET KEY if such set.

The TBR_* constants are available in tbrowse.fh include file. If the code block returns invalid value, tb:ApplyKey() returns TBR_CONTINUE. You may retrieve the associated codeblock by myblock := tb:SETKEY(key) or delete previous setting by tb:SETKEY(key,NIL). The standard Tbrowse SetKey() actions can be set by tb:SetKeyDef() method, and are set in TbrowseArr() and TbrowseDb() functions by default.

[<expB>] := tb:SETKEYDEF([<expL>])

Sets default tb:SetKey() redirections. If <expL> is not given, following SET KEY redirection is set for Tbrowse actions (same as Clipper 5.3):

```
oTb:SetKey(K_DOWN,       {|oTb,key| oTb:Down(),     TBR_CONTINUE} )
oTb:SetKey(K_END,        {|oTb,key| oTb:End(),      TBR_CONTINUE} )
oTb:SetKey(K_CTRL_PGDN,  {|oTb,key| oTb:GoBottom(), TBR_CONTINUE} )
oTb:SetKey(K_CTRL_PGUP,  {|oTb,key| oTb:GoTop(),    TBR_CONTINUE} )
oTb:SetKey(K_HOME,       {|oTb,key| oTb:Home(),     TBR_CONTINUE} )
oTb:SetKey(K_LEFT,       {|oTb,key| oTb:Left(),     TBR_CONTINUE} )
```

```
oTb:SetKey(K_PGDN,       {|oTb,key| oTb:PageDown(),TBR_CONTINUE} )
oTb:SetKey(K_PGUP,       {|oTb,key| oTb:PageUp(),  TBR_CONTINUE} )
oTb:SetKey(K_CTRL_END,   {|oTb,key| oTb:PanEnd(),  TBR_CONTINUE} )
oTb:SetKey(K_CTRL_HOME,  {|oTb,key| oTb:PanHome(), TBR_CONTINUE} )
oTb:SetKey(K_CTRL_LEFT,  {|oTb,key| oTb:PanLeft(), TBR_CONTINUE} )
oTb:SetKey(K_CTRL_RIGHT, {|oTb,key| oTb:PanRight(),TBR_CONTINUE} )
oTb:SetKey(K_RIGHT,      {|oTb,key| oTb:Right(),   TBR_CONTINUE} )
oTb:SetKey(K_UP,         {|oTb,key| oTb:Up(),      TBR_CONTINUE} )
oTb:SetKey(K_ESC,        {|oTb,key|                TBR_EXIT    } )
```

If <expL> is .T., all above key redirections are set to

```
oTb:SetKey(K_..., {|oTb,key| TBR_DEFACTION} )
```

which triggers default handler action but ignores previous SET KEY, ON KEY and SET FUNCTION redirections. Set by default in TbrowseArr() and TbrowseDb() functions. If <expL> is .F., all tb:SetKey() redirections are removed. See tb:ApplyKey() for constants and code block evaluation.

tb:SKIPBLOCK *Access/Assign*

Contains a code block executed in response to repositioning the data using the tb:DOWN(), tb:UP(), tb:PAGEDOWN(), tb:PAGEUP() methods. Two arguments are passed to the block: <nSkip> and <oSelf>. The <nSkip> is numeric argument representing the number of records to be skipped. A positive value means skip forward, and a negative value means skip backward. A zero argument does not indicate a repositioning request, but rather that a data refresh of the current record is required. The <oSelf> is the Tbrowse object self which can be e.g. passed to the UDF function, instead of declaring the object public.

Assigning the tb:SKIPBLOCK is mandatory and must be done latest before any Tbrowse movement and/or before using stabilizing via tb:STABILIZE() or tb:FORCESTABLE().

The code block body typically calculates a new array index or executes a user defined function performing SKIP <arg> for a database movement (predefined when using TBROWSEDB()). The block must return the number of rows (positive, negative, or zero) actually skipped. If the value returned is not the same as the code block argument <arg>, the TBROWSE object assumes that the skip operation encountered the beginning or end of file or of the array boundary. See examples in Chapter 4 and in functions TBROWSENEW(), TBROWSEDB() and TBROWSEARR().

Compatibility note: Clipper passes only one argument to the code block. But when you specify (and not use) the tbrowse object as 2nd parameter, your source remain backward compatible to Clipper.

[retL =] tb:STABILIZE ()

Performs incremental stabilization. Each time this message is sent, some part of the stabilization process is performed. Stabilization is performed in increments so that it can be interrupted by a keystroke or another asynchronous event. If the TBROWSE object is already stable, the method returns TRUE and the tb:STABLE instance is also

set to TRUE. Otherwise, a FALSE value indicates that further stabilize messages should be sent. The TBROWSE is stable when all data has been retrieved and displayed, the data pointer has been repositioned to the record corresponding to the browse cursor, and the current cell has been highlighted. For more details see Chapter 3.

tb:STABLE *Access/Assign*

Contains a logical value indicating whether the TBROWSE object is stable, when TRUE. The browse is considered stable when all data has been retrieved and displayed, the data source has been repositioned to the record corresponding to the browse cursor, and the current cell has been highlighted. When a data movement method is requested, the value is set to FALSE. The invocation of tb:FORCESTABLE() or multiple invocation of tb:STABILIZE() will set the value to TRUE.

tb:TIMEOUT *Access/Assign*

Set or get time-out value in seconds. If you assign numeric value between 1 and 86399 (= 1 sec to 24 hours), the tbrowse handler will exit browsing (similar to ESC key) when a key press (or mouse press) did not occurred within this period since tbrowse start, or last key/mouse press. Default value is 0 which disables time-out. It is used in the standard Tbrowse handler <FlagShip_dir>/system/tbrowsehand.prg and tbrowsedbhand.prg.

tb:TRIM *Access/Assign*

Logical value. If TRUE, sizes columns of character fields to trimmed length of it largest value. This will usually display more columns on the screen at a time, especially with long, only partially filled fields. Applies in GUI mode only, ignored otherwise. The default is .F. See also Tc:Width for additional tuning.

[tb =] tb:UP ()

Moves the TBROWSE cursor up one row. If the cursor is already on the top row, the display is scrolled down and a new row is brought into view. If the data pointer is already at the logical top-of-data and the browse cursor is in the first row, tb:HITTOP instance is set to TRUE.

tb:USERARRAY *Access/Assign*

Assign (or get) an two-dimensional data-array for TbrowseArr(). It is equivalent to <expA10> parameter of TbrowseArr(). The number of elements in each row (i.e. the size of sub-arrays) must be equivalent and the element type (C/N/L/D) in each column must not change. At least one row and column {{"single"}} is required. You may format the column data by TbColumn properties, e.g. tc:Picture, tc:ColorBlock etc.

Example:

```
oTbr:UserArray := {{"row1col1", 1, "row1col3", .T.}, ;
                   {"row2col1", 2, "row2col3", .F.}, ;
                   {"row3col1", 3, "row3col3", .T.} }
```

tb:USERARRAYPOS *Access/Assign*

Assign (or get) current row of array, used in tbrowsehand.prg handler.

tb:VISIBLE *Access*

Returns true when Tbrowse is visible, false if hidden.

tb:VSCROLLBAR *Assign*

Change visibility of vertical scrollbar in GUI mode, default s true.

TbColumn Class

A TBCOLUMN objects contains all the information required to specify a TBROWSE column. Since TBCOLUMN is used only at the conclusion of TBROWSE, this class has no methods, but only instance variables.

Usually one or more newly created TBCOLUMN objects are assigned to a FlagShip variable or directly to the TBROWSE using the tb:ADDCOLUMN() or tb:INSERTCOLUMN() method.

A new TBCOLUMN object is created by TBCOLUMNNEW() and then contains the minimal column information. Additional settings can be specified using the TBCOLUMN instances.

Note that assigning the TBCOLUMN object to TBROWSE will assign the address of the object only, similar to assigning arrays using the = operator. Therefore, additional changes on the TBCOLUMN variable will also automatically apply to TBROWSE until a new object is assigned to that column variable.

After the TBCOLUMN object holding variable (the 'tc' below) is assigned to TBROWSE, you may re-use the same named variable to create another column with TBCOLUMNNEW(), e.g.

```
LOCAL tb, tc, cii
USE mydbf
tb := TBROWSEDB(1,1, maxrow()-2, maxcol()-2)
FOR ii = 1 TO FCOUNT()
   if ii == 1                       // first column is read-only
      cii := ltrim(ii)              // required below
      tc := TBCOLUMNNEW (FieldName(ii), {|| FieldGet(&cii) } )
   else                             // other columns are editable
      tc := TBCOLUMNNEW (FieldName(ii), FieldBlock(FieldName(ii)) )
   endif
   tb:AddColumn(tc)
   // ? EVAL (tc:BLOCK)             // optinal: display data
NEXT
tb:ReadOnly := .F.                  // allow editing
tb:Exec()
```

See also <FlagShip_dir>/system/dbedit.prg and ../examples/tbrowse_*.prg for examples of the implementation.

TbColumnNew ()

Syntax 1:

obj = TBCOLUMNNEW (expC1, expBL2)

Syntax 2:

obj = TBCOLUMN { expC1, expBL2 }

Purpose:

Creates a new TBCOLUMN object initialized by the arguments supplied .

Arguments:

<**expC1**> is a string containing the header text displayed by TBROWSE at the top of this column. <expC1> is stored into the tc:HEADING instance. Multi-line headers (separated by CHR(10) or ";" within the text) are supported by default in GUI same as in Terminal i/o, you however may change it by oTbrowse:HeadStyle() or globally - see Tuning section in the Tbrowse class description.

<**expB2**> is a code block returning the current value of the column data. TBROWSE does not pass any argument to the code block. <expB2> is stored into the tc:BLOCK instance.

<**expL2**> can also be logical TRUE which advises TbColumn to generate a skip block for an array access.

Returns:

<**obj**> is the newly allocated TBCOLUMN object, usually assigned to a regular FlagShip variable or directly to TBROWSE using e.g. tb:ADDCOLUMN().

Description:

TBCOLUMNEW() creates a new object, used for specifying the displayed TBROWSE data. An additional setting of the column can be assigned using the instance variables.

Prior to using the TBROWSE object, one or more TBCOLUMNs must be specified and assigned to TBROWSE.

Example 1:

Used in the example for TBROWSENEW() function, Tbrowse class

```
FUNCTION browdircolumn (brow, dir)
LOCAL col[5], ii

col[1] := TBCOLUMNNEW ("File name", {|| dir[elem,1] })
col[2] := TBCOLUMNNEW ("Size",      {|| dir[elem,2] })
col[3] := TBCOLUMNNEW ("Date",      {|| dir[elem,3] })
col[4] := TBCOLUMNNEW ("Time",      {|| dir[elem,4] })
col[5] := TBCOLUMNNEW ("Attrib",    {|| dir[elem,5] })

* Specify different color attributes for column cell, see
* color attributes in brow:COLORSPEC in TBCOLUMNNEW():
* brow:COLORSPEC := "W/B, W+/B, BG+/W, GR+/B, R+/B, N/W"
```

```
* - Databases are displayed yellow, .prg sources bright
* - File size > 50 KB bright, > 1 MB yellow
* - Date older than 2 months bright white
* - Executables (x attrib) are yellow, r/o white, dirs red

col[1]:COLORBLOCK := {|x| IF(".DB" $ UPPER(x), {4,6}, ;
                         IF(".prg" $ x, {2,6}, {1,6} ))}
col[2]:COLORBLOCK := {|x| IF(x > 1000000, {4,3}, ;
                         IF(x > 50000, {2,3}, {1,3} ))}
col[3]:COLORBLOCK := {|x| IF(DATE()-x > 60, {2,3}, {1,3} )}
col[5]:COLORBLOCK := {|x| IF(LEFT(x,1) == "d", {5,3}, ;
                      IF("x" $ x, {4,3}, ;
                      IF(SUBSTR(x,2,1)!="r", {1,3},{2,3} )))}

col[1]:WIDTH := 10             // adjust column width
col[2]:WIDTH := 4
FOR ii := 1 TO LEN(dir)
    col[1]:WIDTH := MAX(col[1]:WIDTH, StrLen2col(dir[ii,1]))
    col[2]:WIDTH := MAX(col[2]:WIDTH, StrLen2col(Ltrim(dir[ii,2])))
NEXT
col[1]:FOOTING := "unsorted"   // Preset footing msg

FOR ii = 1 to 5                // Assign columns to TBROWSE
   col[ii]:DEFCOLOR := {1, 2 }
   brow:ADDCOLUMN (col[ii])
NEXT
RETURN
```

Example 2:

See also the <FlagShip_dir>/system/dbedit.prg file for a complete example of the TBROWSE and TBCOLUMN usage.

Classification:

programming

Class:

TBCOLUMN class, prototyped in <FlagShip_dir>/include/tbrclass.fh

Compatibility:

Available in FS4, C5 and VO. The alternative syntax 2 and the possibility of inheriting it into an own subclass is available in FlagShip only.

Related:

TBROWSENEW(), TBROWSEDB()

TbColumn Class Index

Class TbColumn

Inherits from:	-
Inherited by:	-
Class prototype:	tbrclass.fh
Defines:	tbrowse.fh

Alignment	ACC/ASS	Set/get the column alignment
Block	ACC/ASS	Code block that retrieves the column data
ColorBlock	ACC/ASS	Code block managing the displayed cell color
ColPos	ACCESS	Get current column number
_ColPos()	METHOD	Set column position (internal)
ColSep	ACC/ASS	Column separator character
Data	ACC/ASS	Get/set current cell data
DefColor	ACC/ASS	Array managing required color attribute
FootColor	ACC/ASS	Set/get the footing color
Footing	ACC/ASS	String displayed at the column footing
FootSep	ACC/ASS	Column footer separator character
GuiColorBlock	ACC/ASS	Code block managing the GUI cell color
GuiDefColor	ACC/ASS	Array managing required GUI color attribute
GuiFontBlock	ACC/ASS	Set/get the GUI Font code block
HeadColor	ACC/ASS	Set/get the header color
Heading	ACC/ASS	String displayed in the column header
HeadSep	ACC/ASS	Column heading separator character
MemoPos	ACC/ASS	Object specifying the position of MemoEdit()
Parent	ACCESS	Get parent (Tbrowse) object
Parent()	METHOD	Set parent object (internal)
Picture	ACC/ASS	Set/get the picture string for column formatting
PostBlock	ACC/ASS	For @..Get/Read
PreBlock	ACC/ASS	For @..Get/Read
ReadOnly	ACC/ASS	Are fields of this column editable?
SetPtrEx()	METHOD	internal
SetStyle()	METHOD	internal
Width	ACC/ASS	Set/get the column width in chars
WidthPixel	ACC/ASS	Set/get the column width in pixel
WidthVisible	ACC/ASS	Set/get the visible column width in pixel

TbColumn Class Properties

tc:BLOCK *Access/Assign*

Contains a code block that retrieves data for the column, equivalent to the <expB2> argument of TBCOLUMNNEW(). Any code block is valid. No block arguments are supplied when the block is evaluated. The code block must return the appropriate data value for the column. Example:

```
USE mydbf
tb := TBROWSEDB()
FOR ii = 1 TO FCOUNT()
   tc := TBCOLUMNNEW (FIELDNAME(ii), FIELDBLOCK (FIELDNAME(ii)) )
   tb:ADDCOLUMN (tc)
   ? EVAL (tc:BLOCK)
NEXT
```

tc:CARGO *Access/Assign*

Contains any user data of any type, to store column information retrieved later in the program using the TBROWSE, for example:

```
USE mydbf
tb := TBROWSEDB()
FOR ii = 1 TO FCOUNT()
   tc := TBCOLUMNNEW (FIELDNAME(ii), FIELDBLOCK (FIELDNAME(ii)) )
   tc:CARGO := "Text for the column " + LTRIM(STR(ii))
   tb:ADDCOLUMN (tc)
NEXT
// later, executing TBROWSE
col := tb:GETCOLUMN (tb:COLPOS)
@ MAXROW(), 0 CLEAR
IF col:CARGO != NIL
   @ MAXROW(), 0 SAY col:CARGO
ENDIF
```

tc:COLORBLOCK *Access/Assign*

Contains an optional code block that determines the color of the displayed data cell. If present, the block is evaluated every time a new value is retrieved in TBROWSE via the tc:BLOCK. The TBROWSE passes the new data element as an argument to the tc:COLORBLOCK. The body of the code block must return an array with two or three numeric elements, specifying the index position of the required color attribute according to the tb:COLORSPEC setting. The first element (color pair index) is used to display unselected cells, the second element specifies color pair for selected cell. The 3rd element, if present and if > 0, is used to paint all unselected cells in current row by this color. For example, to display all negative data red, positive data white and values greater than 1000 yellow/blue, use:

```
brow:COLORSPEC   := "W/B, N/W, W+/B, R+/B, GR+/B"
tc := TBCOLUMNNEW ("Price", FIELDBLOCK ("PRICE") )
#ifdef INLINE_CODED
  tc:COLORBLOCK := {|data| IF (data < 0, {4,2}, ;
                              IF (data > 1000, {5,2}, {3,2} )) }
#else
  tc:COLORBLOCK := {|data| mydisplay(data) }
#endif
brow:ADDCOLUMN (tc)

#ifndef INLINE_CODED
  FUNCTION mydisplay(data)
  LOCAL out[2]
  IF data < 0
     out[1] = 4        // 4th element in DEFCOLOR = "R+/B"  unsel
  ELSEIF data > 1000
     out[1] = 5        // 5th element in DEFCOLOR = "GR+/B" unsel
  ELSE
     out[1] = 3        // 3th element in DEFCOLOR = "W+/B"  unsel
  ENDIF
  out[2] = 2           // 2nd element in DEFCOLOR = "N/W" selected
  RETURN out
#endif
```

To specify cell colors for GUI mode, use tc:GUICOLORBLOCK instead. See <FlagShip_dir>/examples/tbrowse_ar.prg for complete example, including use of 3rd array element for highlighting of the whole line.

tc:COLSEP *Access/Assign*

Contains an optional string that defines the character(s) drawn to the left of this column, if a left TBROWSE column exists. If tc:COLSEP is not specified, the default tb:COLSEP is used by TBROWSE. Applies for terminal i/o mode only, ignored otherwise. Example:

```
element := 1
column := TBCOLUMNNEW ("Second", {|| myarray[element, 2] )
column:COLSEP := " : "
brow:INSCOLUMN (2, column)
```

tc:DATA *Access/Assign*

retVal := tc:DATA (access) returns current cell data and is equivalent to executing retVal := EVAL(tc:BLOCK). tc:DATA := value (assign) sets current cell data and is equivalent to executing EVAL(tc:BLOCK, value).

tc:DEFCOLOR *Access/Assign*

Contains a numeric array with two elements, specifying the index position of the required color attribute according to tb:COLORSPEC to display this column. The first element specifies the attribute index of the normal output (including headings, footings and the column data), while the second element the color index highlighted

output of the TBROWSE cursor which are in these columns. The default setting is {1, 2}, which selects the "normal" and "selected" color pair of the SETCOLOR() attributes, the default for tb:COLORSPEC. Example:

```
brow:COLORSPEC   := SETCOLOR() + "W/B, N/W, W+/B, R+/B, GR+/B"
col := TBCOLUMNNEW ("Name", FIELDBLOCK ("NAME") )
col:DEFCOLOR := { 10, 7 }                    // "GR+/B" and "N/W"
brow:ADDCOLUMN (col)
```

tc:FOOTING *Access/Assign*

Contains a string displayed at the footing of this column. The use of tc:FOOTSEP to separate the footing text from the column data is also recommended. Applies for terminal i/o mode only, ignored otherwise. See next example.

tc:FOOTSEP *Access/Assign*

Contains a character or string which specifies the column footing separator. The string is displayed to the left of the current column, if it is not the first one. The last character of the string is used repetitively for the footing line underlining the column. If tc:FOOTSEP is not specified or contains a null-string "", the default tb:FOOTSEP is used by TBROWSE. Applies for terminal i/o mode only, ignored otherwise. For example:

```
tbr:FOOTSEP := "-+-"
tbr:COLSEP  := " | "            //  xxxxxxxxxx?:!xxxxxxxxx | xxxxx
col:FOOTING := "Column 2"       //  xxxxxxxxxx?:!xxxxxxxxx | xxxxx
col:COLSEP  := "?:!"            //  ----------.:!=========-+------
col:FOOTSEP := ".:!="           //               Column 2
```

tc:GUICOLORBLOCK *Access/Assign*

Same as tc:COLORBLOCK but is used in GUI i/o mode to select colors from the oTbrowse:GUICOLORSPEC array of color pairs. It contains an optional code block that determines the color of the displayed data cell. If present, the block is evaluated every time a new value is retrieved in TBROWSE via the tc:BLOCK. The TBROWSE passes the call value as argument to the tc:GUICOLORBLOCK. The code block body must return array with two or three numeric elements, specifying an index position of the required color attribute according to the tb:GUICOLORSPEC setting, zero signals to use default color. The first element (i.e. color pair index) is used to display unselected cells, the 2nd element specifies color pair for selected cell. The 3rd element, if present, is used to paint unselected cells in current row. For example, to display all unselected data black on while (except in column 2 which is red on yellow), highlight the current row by white on green and the selected cell by yellow/red, use (see full source in *<FlagShip_dir>/examples/tbrowse_ar.prg*):

```
oBr:GuiColorSpec := {"N/W+", ;               // 1: black on white
                     "GR+/R+", ;             // 2: yellow on red
                     "#CC0000/#E5F902", ;    // 3: red on yellow
                     "W+/G" }                // 4: white on green
```

```
for ii := 1 to len(myArray[1])
   oTbcol := TbColumnNew(aHeader[ii], .T.)     // create TbColumn
   if ii == 2                                  // for 2nd column:
      oTbcol:GuiColorBlock := {|val| {3,2,4}}  // = R/Y, Y/R, W/G
   else                                        // other columns:
      oTbcol:GuiColorBlock := {|val| {1,2,4}}  // = B/W, Y/R, W/G
   endif
   oBr:AddColumn(oTbcol)            // assign TbColumn to Tbrowse
next
```

To specify cell colors for Terminal i/o mode, use tc:COLORBLOCK instead.

tc:HEADING *Access/Assign*

Contains a string displayed at the top of this column over the heading separator (line), if tc:HEADSEP is given. Equivalent to the argument <expC1> of TBCOLUMNNEW(). See also example there.

tc:HEADSEP *Access/Assign*

Contains a character or string which specifies the column heading separator. The string is displayed left of the current column, if it is not the first one. The last character of the string is used repetitively for the heading line displayed over the column. If tc:HEADSEP is not specified or contains a null-string "", the default tb:HEADSEP is used instead. See example in tc:FOOTSEP and in TBROWSENEW(). Applies for terminal i/o mode only, ignored otherwise.

tc:MEMOPOS *Access/Assign*

Contains an object of Rectangle class (top,left,bottom,right) specifying the position of MemoEdit() for editing of MEMO fields. When NIL, the position is calculated automatically.

tc:PICTURE *Access/Assign*

Optional string containing the "picture" for formatting the column data. Same as Picture template of @..SAY command or Transform() function. If not available, the default formatting in dependence on the data type and the column width is used.

tc:READONLY *Access/Assign*

Logical value specifying that the fields of this column can or cannot be edited. The default value is FALSE, the fields are editable. See also oTbrowse:READONLY for global setting.

tc:WIDTH *Access/Assign*

Contains a numeric value specifying the display width for the column. If tc:WIDTH is not specified, the column width is calculated as MAX (LEN(tc:HEADING), LEN (tc:FOOTING), LEN(first column data)). If tc:WIDTH is set, all headings, footings and data will be truncated to the specified length. Only character data may be truncated, all other data types expand the column width. In GUI, the WIDTH specifies the minimal column width to be displayed. When the real column size exceeds it setting, the WIDTH is automatically increased. When tc:WIDTH is not set, Tbrowse tries to display as many data as possible in the available space. It calculates the column width for every displayed page and if this increases, it automatically update the visible column size. If the row size is larger than the available Tbrowse width, a horizontal scroll bar is displayed. See also tb:Trim for additional width tuning.

tc:WIDTHPIXEL *Access/Assign*

Same as tc:WIDTH but returns or assigns values in pixel instead of the cols width.

DataServer and DBserver Class

In FlagShip, the database and index access is performed using a replaceable database driver (see section RDD). The high-level database and index functions, described in sections CMD and FUN, invoke methods from the DBSERVER class.

DATASERVER is a "pseudo-class" with predefined method names only, to ensure a proper hybrid use of the procedural vs. RDD object access (see more below). This class should be inherited from other RDDs, which then define their own instances and the required, supported methods. The DataServer class prototype is specified in the stdclass.fh file.

The DBSERVER and DBFIDX classes also inherit the general DATASERVER class. DBSERVER is compatible to CA-VisualObjects, but not available in Clipper. Since the use of the DBSERVER, DBFIDX or any other RDD inheriting the general DATASERVER class is the same, the DBSERVER stands in the following description also for all other similar RDDs.

FlagShip fully **supports** hybrid database operation for all RDDs created (inheriting) from the DATASERVER, DBSERVER or DBFIDX class, as opposed to VO. Hybrid operation means that command and function calls are fully interchangeable with invoking object methods for the same database access. Invoking the database command or function is usually the more comfortable programming way, but you may use the object oriented programming style directly as well.

Same as the high-level database commands and functions operate on the currently selected working area (see LNG.4.3 and CMD.SELECT), the objects of a DataServer or DBserver class perform operations on an automatically opened working area. Therefore, for any open database (and its associated memo fields and indexes), a separate DBserver object exists, created automatically with the USE command, DBUSEAREA() function, or by instantiating the DBserver object.

In FlagShip, as opposed to VO, you may open a database in the current (or a new) working area by:

- the USE command or the DBUSEAREA() function, along with the optional RDD driver name,
- creating an object variable with the DBSERVERNEW() creator function or the DBSERVER {..}

To select the required working area, you may alternatively use

- the SELECT command or the DBSELECTAREA() function,
- the object variable itself, created by the DBSERVERNEW() function or the DBSERVER{}
- the object variable of a specified working area, retrieved by the DBOBJECT() function.

You may interchangeably access the database fields by:

- specifying the field name itself (see LNG.4.2),
- specifying the field name prefixed with an alias (see LNG.4.4),
- the ordinal field number using FIELDNAME() and FIELDPOS() functions,

- invoking the FIELDGET() and FIELDPUT() functions,
- using the object variable, send operator and the field name,
- using the object variable, send operator and one of the methods described below.

Performance hint: the fastest access to a database field is performed by using the field name in the current WA directly (since the addressing is already resolved at compile-time), followed by alias-><field>, the FIELDGET() function or method, then alias->FIELDGET(), the use of object:<field>, and a <field> of a related database.

As with all objects, using TYPED variables (of the known RDD or DATASERVER type) will speed up the application significantly, since already the FlagShip compiler will resolve the object addresses. Otherwise, the run- time system has to search for the class property name for any access to it.

1. Scope and Filters

When using procedural programming, several database commands have clauses to define the scope of records of the database on which to execute. These clauses are FOR, WHILE, ALL, REST, NEXT <nRecords>, RECORD <nRecord> and are described in section CMD.Notation.

In the DBserver class, three instances (oRdd:FORBLOCK, oRdd:WHILEBLOCK and oRdd:SCOPE) are available for defining a global scope. When none of the scoping arguments of a particular method (e.g. oRdd:APPENDDB() etc.) are specified, the global DBserver instances are used by default. This means, the general scope applies whenever one of the bulk processing methods is invoked without an explicit scope.

The <for> argument of some DBserver methods, and the oRdd:FORBLOCK instance specify, that the method will be repeatedly executed for all records according to the <scope>. The condition is stored as a code block, or converted to a code block if given as a string. If the global condition is not required, set it to NIL, the default value.

The <while> argument of some DBserver methods, and the oRdd:WHILEBLOCK instance specify that the repetitive execution of the method stops when a record does not meet the condition. The condition is stored as a code block, or converted to a code block if given as a string. If the global condition is not required, set it to NIL, the default value.

The <scope> argument of the DBserver methods, and the oRdd:SCOPE instance specify partial execution of the method or a range for the for/while condition. The syntax differs slightly from the command notation:

Scope content	Value	Description
DBSCOPEALL	.F.	The scope is ALL records, or REST with WHILE.
DBSCOPEREST	.T.	The scope is the remaining records starting from the current position.
any number	> 0	The scope is NEXT nRecords
set to	NIL	The scope is ALL records, or REST with WHILE.

The above constants are specified in the #include "rddsys.fh" file. Note, that there is no counterpart to the RECORD <nRecord> command scope, since it is very seldom used. If required, you may use the equivalent FlagShip command or function, or issue oRdd:GOTO(nRecord) and set the <scope> to 1.

Remember to restore/reset the general scope and conditions when they are not needed any more. The scope is persistent and applies to all scope- based methods until reset to NIL or via the oRdd: CLEARSCOPE() method.

Filters: in addition to the global scope and conditions, two global filters are available, SET FILTER or oRdd:FILTER and the SET DELETED flag. These filters are considered on any data movement (except GOTO), even in the repetitive execution of methods according to the given or general scope.

2. Summary of Properties

The following table summarize properties of the DATASERVER and DBSERVER class. See their availability in the different RDDs in the section RDD. You may also check the selected RDD driver via the ISOBJPROPERTY (oRdd, <name>, <type>, 1) function.

DATASERVER Name	Type	Descript, CMD/FUN equival.
Alias	Access,Assign	= ALIAS()
AliasSym	Access	symbol of the alias
Append()	Method	= APPEND BLANK
AppendDB()	Method	= APPEND FROM
AppendDelimited()	Method	= APPEND FROM ... DELIM
AppendSDF()	Method	= APPEND FROM ... SDF
AsString()	Method	name of the data server
Average()	Method	= AVERAGE
Axit()	Method	internal, clean up
BlobDirectExport()	Method	export bin. large object to file
BlobDirectGet()	Method	retrieve data from blob file
BlobDirectImport()	Method	import bin. large obj. from file
BlobDirectPut()	Method	write data to blob file
BlobExport()	Method	write data to blob file
BlobGet()	Method	read the blob data
BlobImport()	Method	copy a blob file
BlobRootGet()	Method	read the blob root area
BlobRootLock()	Method	lock root area of the blob file
BlobRootPut()	Method	write the blob root area
BlobRootUnlock()	Method	unlock root area of blob file
BOF	Access	= BOF()
ClearFilter()	Method	clears RDD global filter
ClearIndex()	Method	= CLOSE INDEX
ClearLocate()	Method	clears LOCAL condition
ClearRelation()	Method	= SET RELATION TO
ClearScope()	Method	clears RDD global scope
Close()	Method	= CLOSE
Commit()	Method	= DBCOMMIT()
ConcurrencyControl	Access,Assign	similar to SET AUTOLOCK
Continue()	Method	= CONTINUE
CopyDB()	Method	= COPY TO
CopyDelimited()	Method	= COPY TO ... DELIM
CopySDF()	Method	= COPY TO ... SDF
CopyStructure()	Method	= COPY STRUCT TO
Count()	Method	= COUNT
CreateDB()	Method	= DBCREATE()
CreateIndex()	Method	= INDEX ON ... TO
CreateOrder()	Method	= ORDCREATE()

DataField()	Method	= FIELDGET()
DBStruct()	Method	= DBSTRUCT()
Delete()	Method	= DELETE
DeleteAll()	Method	= DELETE ALL
Deleted	Access	= DELETED()
DeleteOrder()	Method	= ORDDESTROY()
Driver	Access	name of the RDD driver
EOF	Access	= EOF()
ErrInfo	Access	error obj of previous error
Error()	Method	error object / handler
Eval()	Method	= DBEVAL()
FCount	Access	= FCOUNT()
FieldGet()	Method	= FIELDGET()
FieldGetFormatted()	Method	formatted FIELDGET()
FieldHyperLabel()	Method	hyperlabel of the field
FieldInfo()	Method	= FIELDxxx()
FieldName()	Method	= FIELDNAME()
FieldPos()	Method	= FIELDPOS()
FieldPut()	Method	= FIELDPUT()
FieldSpec()	Method	object of the field
FieldStatus()	Method	status of the field operation
FieldSym()	Method	name of a field from symbol
FieldValidate()	Method	validate accord. to field obj
FileSpec	Access	
Filter	Access,Assign	= DBSETFILTER()
FLock()	Method	= FLOCK()
ForBlock	Access,Assign	global RDD 'for' block
Found	Access	= FOUND()
GetArray()	Method	multiple FIELDGET()s
GetArrFields()	Method	multiple FIELDGET()s
GetLocate()	Method	get the LOCATE code block
GetLookupTable()	Method	FIELDGET()s of several rec
GoBottom()	Method	= GO BOTTOM
GoTo()	Method	= GOTO
GoTop()	Method	= GO TOP
Header	Access	= HEADER()
IndexCheck	Access	= INDEXCHECK()
IndexCount	Access	= INDEXCOUNT()
IndexExt	Access	= INDEXEXT()
IndexKey	Access	= INDEXKEY()
IndexKey()	Method	= INDEXKEY()
IndexLock	Access	locks the index
IndexOrd()	Method	= INDEXORD()
Info()	Method	various infos about the RDD
Init()	Method	= USE ... or DBUSEAREA()
IsRelation	Access,Assign	activate/deactivate relations
Join()	Method	= JOIN

LastRec	Access	= LASTREC(), RECCOUNT()
Locate()	Method	= LOCATE
LockCurrentRecord()	Method	= RLOCK()
LockSelection()	Method	multiple RLOCK()s
LUpdate	Access	= LUPDATE()
Name	Access	= RDDSETDEFAULT()
NoiVarGet()	Method	exception Access handler
NoiVarPut()	Method	exception Assign handler
NoMethod()	Method	exception Method handler
Notify()	Method	event handler
OrderBottomScope	Access,Assign	control value of bottom
OrderDescend()	Method	similar to DESCEND clause
OrderInfo()	Method	various infos about the order
OrderIsUnique()	Method	similar to UNIQUE clause
OrderKeyAdd()	Method	add a key into order
OrderKeyCount()	Method	no of keys in order
OrderKeyDel()	Method	delete key in order
OrderKeyGoTo()	Method	move to record no
OrderKeyNo	Access,Assign	logical record number
OrderKeyNo()	Method	logical record number
OrderKeyVal	Access	= &(INDEXKEY())
OrderScope()	Method	boundary scope on order
OrderSkipUnique()	Method	skip unique in order
OrderTopScope	Access,Assign	control value of top
Pack()	Method	= PACK
QuickFieldGet()	Method	= FIELDGET()
QuickFieldPut()	Method	= FIELDPUT()
RDDInfo()	Method	various infos about RDD
RDDName	Access	= RDDSETDRIVER()
ReadOnly	Access	status of USE open
Recall()	Method	= RECALL
RecallAll()	Method	= RECALL ALL
RecCount	Access	= LASTREC(), RECCOUNT()
RecNo	Access,Assign	= RECNO()
RecordInfo()	Method	various infos about record
RecSize	Access	= RECSIZE()
Refresh()	Method	undo record changes
RegisterClient()	Method	register a window
Reindex()	Method	= REINDEX
Relation()	Method	= DBRELATION()
RelationObject()	Method	object of the relation
Replace()	Method	= REPLACE
ResetNotification()	Method	suppress notifying
RLock()	Method	= RLOCK()
RLockList	Access	= RLOCKLIST()
RLockVerify()	Method	similar to RLOCK()
RollBack()	Method	roll back

Scope	Access,Assign	general RDD scope
Seek()	Method	= SEEK
SeekEval()	Method	= SEEK EVAL
SetDataField()	Method	assign object to field
SetFilter()	Method	= DBSETFILTER()
SetIndex()	Method	= SET INDEX TO...
SetOrder()	Method	= DBSETORDER()
SetOrderCondition()	Method	condit. of INDEX..FOR
SetRelation()	Method	= DBSETRELATION()
SetSelectiveRelation()	Method	set selective relation
Shared	Access	= ! ISDBEXCL()
Skip()	Method	= SKIP
Sort()	Method	= SORT
Status	Access	
Sum()	Method	= SUM
SuspendNotification()	Method	suspend notification
Total()	Method	= TOTAL
Unlock()	Method	= UNLOCK
Update()	Method	= UPDATE
Used	Access	= USED()
UsersDbf()	Method	= USERSDBF()
WhileBlock	Access,Assign	global RDD 'while' block
Zap()	Method	= ZAP

By default, all methods of the DATASERVER class are empty and call the predefined DataServer:NoMethod(), all Access methods call DataServer:NoiVarGet() and all Assign methods call the DataServer:NoiVarPut() method. A very minimal (hybrid) RDD driver should therefore at least specify it's own INIT(), CLOSE() and FIELDGET() methods and the USED Access method. See also section RDD and an example in the <FlagShip_dir>/system/ smallrdd.prg file.

Compatibility: the DATASERVER is a superset of the CA/VO class of the same name. The DBSERVER class is generally compatible to CA/VO and to other FlagShip RDDs. If slight differences exist, they are given in the description of the particular method below. Neither the DataServer, nor the DBserver class are available in CA/Clipper.

DBSERVERNEW() and DBFIDXNEW()

Syntax 1:

```
obj = DBSERVER {expC1, [expL2], [expL3], [expC4],
            [expA5], [expL6]}
```

or:

```
obj = DBSERVERNEW (expC1, [expL2],
            [expL3],[expC4],[expA5],[expL6])
```

Syntax 2:

```
obj = DBFIDX {expC1, [expL2], [expL3], [NIL],
            [NIL], [expL6] }
```

or:

```
obj = DBFIDXNEW (expC1, [expL2], [expL3], [NIL],
            [NIL], [expL6] )
```

Purpose:

Creates a new DBSERVER object for the DBFIDX driver, optionally initialized by the supplied arguments. Opens the specified database (and its associated memo file when memo fields exist) in the current or the first free working area, equivalent to the USE command or DBUSEAREA() function.

DBSERVER{} or DBSERVERNEW() according to syntax 1 is designed for generic RDD purposes and may be slightly slower than alternatively using the RDD driver name itself according to syntax 2, see text below.

Arguments:

<**expC1**> specifies the name of the database file to open in the current or the first free working area. If no extension is specified, the default .dbf extension is assumed. Upper/lower case translation is performed according to FS_SET(), the search path may be specified with SET PATH or SET DEFAULT.

Options:

<**expL2**> is a synonym for the SHARED clause of the USE command. If specified TRUE (or if the DB_SHARED constant is used), the database is open for shared use in multiuser, multitasking network or concurrent mode. If the argument is FALSE, the database is opened in EXCLUSIVE mode. If not specified, the current SET EXCLUSIVE status is used.

<**expL3**> is a synonym for the READONLY clause of the USE command. If the argument is TRUE (or if the DB_READONLY constant is used), the database is opened for read-only purposes. The Unix access rights -r-- are sufficient for the database and memo <file> (but not for index files (.idx of the DBFIDX driver), which must always be -rw-). In an attempt to REPLACE or APPEND a record, a run-time error is brought up. If the argument is FALSE or not specified, the database is open in read-write mode.

<**expC4**> is the driver name of the DBSERVER class. If not specified, it defaults to the driver specified by RDDSETDEFAULT() which in turn defaults to DBFIDX. If

specified, and the name differs from the default driver, you also have to include the EXTERN <expC4>NEW statement somewhere in the application, or explicitly link in the RDD driver.

Alternatively, you may explicitly invoke the RDD driver itself according to syntax 2, e.g. DBFIDXNEW(...), CB4CDX{..} etc, to avoid this parameter (or specify it NIL). The FlagShip high-level USE command and DBUSEAREA() function call the default driver (usually DBFIDXNEW()), if the VIA clause is not given.

<**expA5**> is not used and placed here for compatibility to VO only. You may specify any value, the default is NIL. In FlagShip, you may create an inherited object also from the DBSERVER class.

<**expL6**> is a synonym for the NEW clause of the USE command. If <expL6> is specified TRUE (the DB_NEW constant), or not given, an unused working area is selected first, making it the current one, and the database <expC1> is opened there. If the argument is FALSE or the DB_SELECTED constant, the database is opened in the currently SELECTed working area, closing any active database occupying that working area.

Returns:

<**obj**> is the newly allocated DBSERVER or RDD object, usually as- signed to a regular FlagShip variable or to an array element. Before using the object, verify that the database was successfully opened by using the oRdd:USED instance, or the USED() function.

Description:

DBSERVERNEW() creates a new DBserver object. The functionality is equivalent to the standard USE command, the DBUSEAREA() function or the instantiating of the <defaultRDD> object. The automatically called INIT() method opens an existing database .dbf file, and its associated memo .dbt file in the current (or the first available) working area.

After successfully opening the database (the oRdd:USED instance or the USED() function returns TRUE), the record pointer points to the first record. If the database is empty, both BOF() and EOF() are set to TRUE. You may then assign another alias, indices etc. to the object.

For more information, refer to the USE command.

As with all objects, using TYPED variables (of the known RDD or DBSERVER type) and prototypes (by default included in the stdclass.fh file) will speed up the application significantly (e.g. specifying LOCAL oDbf AS DBSERVER).

Performance:

The direct usage of DBFIDX{} or DBFIDXNEW(), instead of the general purpose DBSERVER{} or DBSETVERNEW() will result in faster applications.

Multiuser:

If a multiuser, multitasking and/or network access is required, database files can be opened EXCLUSIVEly or SHARED, using the <expL2> argument, alternatively by using the SET EXCLUSIVE command.

Opening a database EXCLUSIVEly will succeed only if it is not already in use by the same or another user. Attempting to open a database SHARED will succeed only if the database is not opened exclusively by another user (or concurrently in another working area). Always check the oRdd:USED instance, USED() or NETERR() functions or the return value <retL> to see whether the database has been successfully opened.

For special purposes, FlagShip allows the same database to be used simultaneously in different working areas, when the given ALIAS names (given in the oRdd:ALIAS instance, or specified by the 4th argument in DBUSEAREA() function) differ. On the object instantiation, FlagShip automatically creates a new ALIAS name, if such already exist. The handling of concurrent databases is the same, as the use of shared databases in multiuser mode.

In SHARED mode, any **write** attempt to the database or memo file (like REPLACE, DELETE, RECALL, oRdd:FieldName := ... or alias->FieldName := ...) requires that the current record or the whole file is locked beforehand using RLOCK() or FLOCK(). This will ensure data integrity denying other users a write access to the same record or database. When the write access is finished, use UNLOCK or UNLOCK ALL to release the previously set record and file locks, so that another user may lock the file or record.

FlagShip's RDD allows **automatic** record and file locking/unlocking, when a RLOCK() or FLOCK() is not already specified by the programmer. The auto-locking capability is specified in the oRdd:ConcurrencyControl instance. During object creation, this instance will be set according to the current SET AUTOLOCK state. You may redefine the instance at any later time.

Global changes to the physical record storage order (PACK and ZAP) or rebuilding the index files (INDEX, REINDEX) require an EXCLUSIVE open mode (which cannot be handled by the automatic concurrence control).

Refer to the USE command and LNG.4.8 for more information about multiuser programming.

Example 1:

```
LOCAL dbf5 AS DBSERVER                          // optional
SELECT 5
dbf5 := DBSERVERNEW ("mydbf",,,,, .F.)          // minimal usage
**   := DBSERVERNEW ("mydbf",,,"dbfidx",,.F.) // equivalent
**   := DBFIDXNEW   ("mydbf",,,,, .F.)          // equivalent
**   := DBSERVER    {"mydbf"}                   // ditto, NEW
**   := DBSERVERNEW ("mydbf")                   // ditto, NEW

if !dbf5:USED                                   // check success
   ? "Couldn't open mydbf.dbf file in WA5"
   QUIT
endif

? "The " + dbf5:NAME + " database was open in " + ;
  if(dbf5:SHARED, "shared", "exclusive") + ;
  if(dbf5:READONLY, ", read-only", "") + ;
  " mode. The automatic locking is " + ;
```

```
  if(!dbf5:SHARED, "not required.", ;
     if(db5:CONCURRENCY, "enabled.", "disabled."))
```

Example 2:

Hybrid use of objects and commands is possible in FS4

```
LOCAL oAdr AS DBSERVER          // or ...AS DBFIDX
SET AUTOLOCK TO 0
oAdr := DBFIDXNEW ("address", .T., .F.)
if !USED()                      // == if NETERR()
   QUIT
endif
APPEND BLANK                    // == oAdr:APPEND()
REPLACE field->Name WITH "Smith"
oAdr:First := "John"            // == address->First := ...

oAdr:APPEND()                   // == DBAPPEND()
Name := "Miller"                // == address->Name  := ...
oAdr:First := "Peter"           // == address->First := ...
```

Example 3:

Other hybrid use of objects, functions and commands

```
LOCAL oAdr AS DBSERVER
USE address INDEX adr1,adr2 NEW SHARED
if NETERR()
   quit
endif
? DBF() + " is open in working area " + ;
  ltrim(str(SELECT())) + ", alias = " + ALIAS()
DBAPPEND()
FIELDPUT (1, "Smith")
SEEK "Miller"
? "Miller was " + if(FOUND(), "found at record " + ;
  ltrim(str(RECNO())), "not found")

oAdr := DBOBJECT()              // retrieve object
? oAdr:NAME + " is open in working area " + ;
  ltrim(str(SELECT())) + ", alias = " + oAdr:ALIAS
oAdr:SEEK("Smith")
? "Smith was " + if(oAdr:FOUND, "found at record " + ;
  ltrim(str(oAdr:RECNO)), "not found")
```

Classification:

programming

Class:

<FlagShip_dir>/include/

datserver.fh	= prototype of DATASERVER class
dbserver.fh	= prototype of DBSERVER class
dbfidx.fh	= prototype of DBFIDX class

Include:

The constants are available in "rddsys.fh"

Compatibility:

Available in FS4 and VO only. FS4 does not support the VO's optional use of <expO1>, nor an array of RDDs in <expA5>. VO does not support the 6th parameter <expL6>, nor the hybrid use of database commands and functions with objects.

Related:

USE, DBUSEAREA(), USED(), RDDSETDEFAULT(), CLASS, PROTOTYPE, oRdd:INFO(), LNG.2.11, other drivers in sect. RDD #

DataServer and DBserver Properties

Note: you may determine current database object <oRdd> by DbObject(), so e.g. the return of Alias() function is equivalent to DbObject():Alias
The used DBI_* and DBOI_* constants/manifests are defined in rddsys.fh

oRdd:ALIAS <—> expC *Access/Assign*

Contains a string representing the alias of the work area. It is set to the database name on instantiation. Equivalent to and invoked from the ALIAS() function and oRdd:INFO(DBI_ALIAS) method. Compatibility: VO supports access only.

Related: ALIAS(), oRdd:INFO(DBI_ALIAS)

oRdd:APPEND ([expL1]) —> retL *Method*

Adds a new empty record to the end of the currently selected database. The availability is signaled by oRdd:INFO(DBI_CANPUTREC). Equivalent to and invoked from the APPEND BLANK command or the DBAPPEND() function.

Optional arguments: <**expL1**> indicates if the existing record locks should be released. If not specified or TRUE, all record locks are cleared, then the new record appended and locked. This is equivalent to the behavior of the APPEND BLANK command. If specified FALSE, the previous records remain locked (until oRdd:UNLOCK() or oRdd:UNLOCK(recNo) methods, or the UNLOCK command is executed), and the new record is added to the lock list.

Returns: <**retL**> is a logical value, TRUE signals the success, an error otherwise.

Example:

```
LOCAL ii, oAdr
oAdr := DBSERVER {"address", .T.} // open shared
FOR ii := 1 TO 10
   oAdr:APPEND(.F.)               // hold locks
NEXT
aeval(oAdr:RlockList(), {|x| QOUT("Lock:",x)})
if oAdr:APPEND()                  // release RLOCKs first
   oAdr:Name := "Smith"           // replace
   oAdr:UNLOCK()                  // unlock all
endif
```

Related: APPEND BLANK, DBAPPEND(), UNLOCK, DBRUNLOCK(), oRdd:UNLOCK(), oRdd:FIELDINFO(), oRdd:INFO(DBI_CANPUTREC)

oRdd:APPENDDB (expC1...) —> retL *Method*

Adds records to the current (target) database file from another (source) database file. Equivalent to the APPEND FROM command.

```
retL = oRdd:APPENDDB (expC1|expO1, [expA2],
          [expC3|expB3], [expC4|expB4],
          [expN5|expL5], [expC6])
```

Arguments: <**expC1**>|<**expO1**> is the name or object of the source database. If no extension is specified, it is assumed to be .dbf, or the standard extension according to the RDD driver of <expC6>. If <expC1> is specified, the source database is used in shared, read- only mode. If <expO1> is given, the RDD server object is used.

Options: <**expA2**> is an array of character values, specifying the field names of the source and target database to be included. If not specified, all fields of the source database are transferred. Equivalent to the FIELDS clause of APPEND FROM. FlagShip performs an automatic type translation, if necessary.

<**expC3**>|<**expB3**> is equivalent to the FOR scope. The condition, given as a string or code block, is evaluated for each record of the source scope.

<**expC4**>|<**expB4**> is equivalent to the WHILE scope. The condition, given as a string or code block, is evaluated for each record of the source from the current record until it returns FALSE.

<**expN5**>|<**expL5**> is the range of records in the source, providing the same functionality as the ALL, REST and NEXT clause of commands. See chapter 6.2 for the scope values.

<**expC6**> is equivalent to the VIA clause of the APPEND FROM command. It specifies the RDD of the source database, if <expC1> is used.

Scope: If none of the arguments 3 to 5 are specified, the global source server scope is used, see chapter 6.1.

Returns: <**retL**> signals the success if TRUE or failure otherwise.

Compatibility: in VO, the first 2 arguments are mandatory and the <expC6> argument is not available.

Related: APPEND FROM, oRdd:APPENDSDF(), oRdd:APPENDDELIMITED(), oRdd:INFO()

oRdd:APPENDDELIMITED (expC1...) —> retL *Method*

Adds records to the current (target) database file from an ASCII text file in a "comma-separated-value" CSV file format (source). Equivalent to APPEND FROM ... DELIMITED command.

```
retL = oRdd:APPENDDELIMITED (expC1, [expC2],
          [expA3], [expC4|expB4], [expC5|expB5],
          [expN6|expL6])
```

Arguments: <**expC1**> is the name of the ASCII source file. If no extension is specified, it is assumed to be .txt.

Options: <**expC2**> is a single character specifying the delimiter of character fields, equivalent to the DELIMITED WITH clause of APPEND FROM ... command. If not specified, the oRdd:INFO(SETDELIMITER) character, or double quotation mark (") is assumed.

<**expA3**> is an array of character values, specifying the field names of the target database and the order of the fields in the text file. If not specified, the order of fields corresponds to the field order of the target database. Equivalent to the FIELDS clause of APPEND FROM.

<**expC4**>|<**expB4**> is equivalent to the FOR scope. The condition, given as a string or code block, is evaluated for each record of the scope.

<**expC5**>|<**expB5**> is equivalent to the WHILE scope. The condition, given as a string or code block, is evaluated for each record from the current record until it returns FALSE.

<**expN6**>|<**expL6**> is the range of records, providing the same functionality as the ALL, REST and NEXT clause of commands. See chapter 6.2 for the scope values.

Scope: The global server scope according to chapter 6.1 is not applicable for the text (source) file. If the parameters 4 to 6 are not specified, all records of the source are transferred.

Returns: <**retL**> signals the success if TRUE or failure otherwise.

Compatibility: the first 3 arguments are mandatory in VO.

Related: APPEND FROM ... DELIMITED, oRdd:APPENDSDF(), oRdd:APPENDDBF(), oRdd:INFO()

oRdd:APPENDSDF (expC1...) —> retL *Method*

Adds records to the current database file (target) from an ASCII text file in SDF format (source). Equivalent to the APPEND FROM ... SDF command.

```
retL = oRdd:APPENDSDF (expC1, [expA2],
          [expC3|expB3], [expC4|expB4],
          [expN5|expL5])
```

Arguments: <**expC1**> is the name of the ASCII source file in SDF format. If no extension is specified, .txt is assumed.

Options: <**expA2**> is an array of character values, specifying the field names of the target database and the order of the fields in the text file. If not specified, the order of fields corresponds to the field order of the target database. Equivalent to the FIELDS clause of APPEND FROM.

<**expC3**>|<**expB3**> is equivalent to the FOR scope. The condition, given as a string or code block, is evaluated for each record of the source scope.

<**expC4**>|<**expB4**> is equivalent to the WHILE scope. The condition, given as a string or code block, is evaluated for each record from the current record until it returns FALSE.

<**expN5**>|<**expL5**> is the range of records, providing the same functionality as the ALL, REST and NEXT clause of commands. See chapter 6.2 for the scope values.

Scope: The global server scope according to chapter 6.1 is not applicable for the text (source) file. If the parameters 3 to 5 are not specified, all records of the source are transferred.

Returns: <**retL**> signals the success if TRUE or failure otherwise.

Compatibility: the first 2 arguments are mandatory in VO.

Related: APPEND FROM ... SDF, oRdd:APPENDDBF(), oRdd:APPEND-DELIMITED(), example in sect RDD.2

oRdd:ASSTRING ( ) —> retC *Method*

This method is equivalent to the oRdd:NAME access. It returns the main part of the used database file name, e.g. "MyFile". Available for compatibility purposes to CA/VO.

oRdd:AVERAGE (expC1...) —> retA *Method*

Calculates the average of a series of numeric expressions for a range of records in the current database file and puts the results in the returned array. Similar to the AVERAGE command.

```
retA = oRdd:AVERAGE (expC1|expB1|expA1,
          [expC2|expB2], [expC3|expB3],
          [expN4|expL4])
```

Arguments: <**expC1**>|<**expB1**>|<**expA1**> is a string which specifies the numeric expression (e.g. field names) to be averaged, a code block to be executed, or an array of expressions or code blocks.

Options: <**expC2**>|<**expB2**> is equivalent to the FOR scope. The condition, given as a string or code block, is evaluated for each record of the scope.

<**expC3**>|<**expB3**> is equivalent to the WHILE scope. The condition, given as a string or code block, is evaluated for each record from the current record until it returns FALSE.

<**expN4**>|<**expL4**> is the scope, a range of records, providing the same functionality as the ALL, REST and NEXT clause of commands. See chapter 6.2 for the scope values.

Scope: If none of the arguments 2 to 4 are specified, the global server scope is used, see chapter 6.1.

Returns: <**retA**> is an array of results. If a single expression or code block was specified, an array length of 1 is returned.

Related: AVERAGE command

oRdd:AXIT ( ) —> retL|NIL *Method*

This method performs an internal garbage collection of the object, just before the object is destroyed. It is invoked automatically, you should **not** invoke it manually. See additional description in section LNG.11.3 and RDD.2.3.3. To ensure the correct functionality, an inheriting class should invoke SUPER:AXIT() method, if a separate oRdd:AXIT() method is required and specified (check it by ISOBJPROPERTY() function).

oRdd:BOF —> expL *Access*

Contains a logical value indicating whether there was an attempt to move past the beginning of the current database file. It also returns TRUE if the database contains no records. Equivalent to and invoked from the BOF() function.

oRdd:CARGO <—> exp Export (access/assign)

Contains user data of any type, to store information retrieved later in the program. Not used by the RDDs itself. Compatibility: not available in VO.

oRdd:CLEARFILTER ( ) —> retL *Method*

Clears the global filter condition (see also chapter 6.1) specified with the oRdd:SETFILTER() method or the SET FILTER command. Equivalent to and invoked from the SET FILTER TO command w/o parameters or the DBCLEARFILTER() function.

Returns: <**retL**> signals success, if TRUE.

Related: oRdd:SETFILTER(), oRdd:FILTER, SET(), SET FILTER, DBCLEARFILTER()

oRdd:CLEARINDEX ( ) —> retL *Method*

Clears all indexes currently associated with the server. Equivalent to and invoked from the CLOSE INDEX or SET INDEX TO commands w/o parameters.

Returns: <**retL**> signals success, if TRUE.

Related: oRdd:SETINDEX(), SET INDEX, CLOSE INDEX

oRdd:CLEARLOCATE ( ) —> retL *Method*

Clears the LOCATE condition set by the <for> argument of oRdd:LOCATE() method or the FOR clause of the LOCATE command. Note, that this condition is different from the global oRdd:FORBLOCK instance, described also in chapter 6.1.

Returns: <**retL**> signals success, if TRUE.

Related: oRdd:LOCATE(), oRdd:CONTINUE(), LOCATE, CONTINUE

oRdd:CLEARRELATION ( ) —> retL *Method*

Clears all relations to other database servers. Equivalent to and invoked from the command SET RELATION TO w/o parameters.

Returns: <**retL**> signals success, if TRUE.

Related: oRdd:SETRELATION(), oRdd:RELATION(), SET RELATION

oRdd:CLEARSCOPE ( ) —> retL *Method*

Sets the global scope instances oRdd:SCOPE, oRdd:FORBLOCK and oRdd:WHILEBLOCK to NIL. See also chapter 6.1 for an additional discussion of the global scope.

Returns: <**retL**> signals success, if TRUE.

Related: oRdd:SCOPE, oRdd:FORBLOCK, oRdd:WHILEBLOCK

oRdd:CLOSE ( ) —> retL *Method*

Closes the database file and its associated index and memo files, if any. Clears the relations set to other databases. Clears all global scopes and filters for the database server. Equivalent to and invoked from the commands CLOSE DATABASE or USE w/o parameters. Invoked automatically, when the application terminates.

Returns: <**retL**> signals success, if TRUE.

Related: DBSERVERNEW(), oRdd:INIT(), oRdd:SETORDER(), oRdd:SETINDEX(), oRdd:CLEARSCOPE(), oRdd:CLEARRELATION(), CLOSE DATABASES, USE, QUIT

oRdd:COMMIT ( ) —> retL *Method*

Commits all changes of the server fields to disk, ensuring that all buffers are flushed. Equivalent to and invoked from the DBCOMMIT() function, oRdd:SKIP(0) method or SKIP 0 command. Note, that this flushing is performed asynchronously in background,

as opposed to the immediate, synchronous flushing by the COMMIT command or DBCOMMITALL() function.

Returns: <**retL**> signals success, if TRUE.

Related: oRdd:SKIP(), COMMIT, DBCOMMIT(), DBCOMMITALL()

oRdd:CONCURRENCYCONTROL <–> expN ***Access/Assign***

Contains a numeric value indicating the mode of automatic concurrence control for this data server, determining when and how records are locked and released. Preset during instantiation according to the state of the SET AUTOLOCK switch. The following constants are available in the #include "rddsys.fh" file.

Constant	Value	Description
CCNONE	0	The data server provides no automatic record locking; the application is required to do all the locking explicitly.
CCOPTIMISTIC	1	Execute the AUTOxLOCK() function or method, if lock is required. The data server locks and unlocks the record (or the file on multiple record replacement) automatically, but only if no programmer's RLOCK() or FLOCK() was detected. This option follows the SET AUTOLOCK setting and is therefore performed only, if SET AUTOLOCK was not set < 2.
CCSTABLE	2	currently equivalent to CCOPTIMISTIC.
CCREPEATABLE	3	currently equivalent to CCOPTIMISTIC.
CCFILE	4	currently equivalent to CCOPTIMISTIC.
negat number	< 0	Similar to CCOPTIMISTIC, but the trial period is specified here in negative seconds. To try the lock for 3 seconds, specify -3, to try it forever, specify e.g. -9999999.

Compatibility: the AUTOxLOCK() functionality is not available in VO.

Related: SET AUTOLOCK, AUTOxLOCK(), RLOCK(), FLOCK(), SET MULTILOCKS, oRdd:FIELDPUT(), oRdd:INFO()

oRdd:CONTINUE ( ) –> retL ***Method***

Continues the pending LOCATE or oRdd:LOCATE() search from the current record, using the <for> condition of LOCATE, but ignoring its <while> condition and <scope>. Equivalent to and invoked from the CONTINUE command. Hint: If you want to continue searching with the <while> condition, set the <scope> to REST and perform another oRdd:LOCATE().

Returns: <**retL**> signals success, equivalent to the oRdd:FOUND instance. If the search was successful, the matching record becomes the current record, and this method, the FOUND() function or oRdd:FOUND instance returns TRUE. If not

found, the record pointer is positioned on EOF or the first record outside the FOR scope, and FALSE is returned.

Related: oRdd:LOCATE(), oRdd:CLEARLOCATE(), oRdd:INFO(), LOCATE, CONTINUE

oRdd:COPYDB (expC1...) —> retL *Method*

Copies records from the current database file (source) to another database file (target). Equivalent to the COPY TO command.

```
retL = oRdd:COPYDB (expC1|expO1, [expA2],
          [expC3|expB3], [expC4|expB4],
          [expN5|expL5], [expC6])
```

Arguments: <**expC1**>|<**expO1**> is the name or object of the target database. If no extension is specified, it is assumed to be .dbf, or the standard extension according to the RDD driver of <expC6>. If <expC1> is specified, the target database is opened exclusively. If <expO1> is given, the RDD server object is used and locked automatically.

Options: <**expA2**> is an array of character values, specifying the field names of the source and target database to be copied. If not specified, all fields of the source database are transferred. Equivalent to the FIELDS clause of COPY TO. FlagShip performs an automatic type translation, if necessary.

<**expC3**>|<**expB3**> is equivalent to the FOR scope. The condition, given as a string or code block, is evaluated for each record of the scope.

<**expC4**>|<**expB4**> is equivalent to the WHILE scope. The condition, given as a string or code block, is evaluated for each record from the current position in the source database until <expB4> returns FALSE.

<**expN5**>|<**expL5**> is the range of records, providing the same functionality as the ALL, REST and NEXT clause of commands. See chapter 6.2 for the scope values.

<**expC6**> is equivalent to the VIA clause of the COPY TO command. It specifies the RDD of the target database, if <expC1> is used.

Scope: If none of the arguments 3 to 5 are specified, the global scope of the current server is used, see chapter 6.1.

Returns: <**retL**> signals success, if TRUE, or failure (e.g. the open mode) otherwise.

Compatibility: in VO, the first 2 arguments are mandatory and <expC6> is not available.

Related: COPY TO, oRdd:COPYSDF(), oRdd:COPYDELIMITED(), APPEND FROM, COPY FILE, RENAME

oRdd:COPYDELIMITED (expC1...) —> retL *Method*

Copies records from the current database file (source) to a "comma- separated-value" CSV file format (target). Equivalent to the COPY TO ... DELIMITED command.

```
retL = oRdd:COPYDELIMITED (expC1, [expC2], [expA3],
          [expC4|expB4], [expC5|expB5],
          [expN6|expL6])
```

Arguments: <**expC1**> is the name of the target ASCII file. If no extension is specified, it is assumed to be .txt.

Options: <**expC2**> is a single character specifying the delimiter of character fields, equivalent to the DELIMITED WITH clause of the COPY TO ... command. If not specified, the oRdd:INFO(SETDELIMITER) character, or double quotation mark (") is assumed.

<**expA3**> is an array of character values, specifying the field names of the source database and the order of the fields in the target text file. If not specified, all fields of the source are transferred, the order of fields corresponds to the field order of the source database. Equivalent to the FIELDS clause of COPY TO.

<**expC4**>|<**expB4**> is equivalent to the FOR scope. The condition, given as a string or code block, is evaluated for each record of the scope.

<**expC5**>|<**expB5**> is equivalent to the WHILE scope. The condition, given as a string or code block, is evaluated for each record from the current position until <expB5> returns FALSE.

<**expN6**>|<**expL6**> is the range of records, providing the same functionality as the ALL, REST and NEXT clause of commands. See chapter 6.2 for the scope values.

Scope: If none of the arguments 4 to 6 are specified, the global scope of the current server is used, see chapter 6.1. If not available, all records are transferred.

Returns: <**retL**> signals success, if TRUE, or failure otherwise.

Compatibility: the first 3 arguments are mandatory in VO.

Related: COPY TO ... DELIMITED, oRdd:COPYSDF(), oRdd:COPYDBF(), oRdd:INFO()

oRdd:COPYSDF (expC1...) —> retL *Method*

Copies records from the current database file (source) to an ASCII text file in SDF format (target). See the additional description of the format in the COPY TO ... SDF command.

```
retL = oRdd:COPYSDF (expC1, [expA2], [expC3|expB3],
          [expC4|expB4], [expN5|expL5])
```

Arguments: <**expC1**> is the name of the ASCII target file in SDF format. If no extension is specified, .txt is assumed.

Options: <**expA2**> is an array of character values, specifying the field names of the source database and the order of the fields in the target text file. If not specified, all fields of the source are transferred, the order of fields corresponds to the field order of the source database. Equivalent to the FIELDS clause of COPY TO.

<**expC3**>|<**expB3**> is equivalent to the FOR scope. The condition, given as a string or code block, is evaluated for each record of the source scope.

<**expC4**>|<**expB4**> is equivalent to the WHILE scope. The condition, given as a string or code block, is evaluated for each record from the current position until <expB4> returns FALSE.

<**expN5**>|<**expL5**> is the range of records, providing the same functionality as the ALL, REST and NEXT clause of commands. See chapter 6.2 for the scope values.

Scope: If none of the arguments 3 to 5 are specified, the global scope of the current server is used, see chapter 6.1. If not available, all records are transferred.

Returns: <**retL**> signals success, if TRUE, or failure otherwise.

Compatibility: the first 2 arguments are mandatory in VO.

Related: COPY TO ... SDF, oRdd:COPYDBF(), oRdd:COPYDELIMITED()

oRdd:COPYSTRUCTURE (expC1...) —> retL *Method*

Creates an empty database (target) with field definitions from the current (source) database. Equivalent to the COPY STRUCTURE TO command.

```
retL = oRdd:COPYSTRUCTURE (expC1, [expA2], [expC3])
```

Arguments: <**expC1**> is the name of the target database. If no extension is specified, it is assumed to be .dbf, or the standard extension according to the RDD driver of <expC3>.

Options: <**expA2**> is an array of character values, specifying the field names of the source database to be included in the given order in the target database. If not specified, the target database overtakes the structure of the source. Equivalent to the FIELDS clause of COPY STRUCTURE.

<**expC3**> is equivalent to the VIA clause of the COPY STRUCTURE command. It specifies the RDD of the target database, if not the default one.

Returns: <**retL**> signals success, if TRUE, or failure (e.g. the create mode) otherwise.

Compatibility: in VO, the first 2 arguments are mandatory and the <expC3> argument is not available.

Related: COPY STRUCTURE TO, oRdd:CREATEDB()

oRdd:COUNT ([expC1...]) —> retN *Method*

Counts records in the current working area, which fall into the given scope and fulfill the specified conditions. The result is returned. Equivalent to and invoked from the COUNT command.

```
retN = oRdd:COUNT ([expC1|expB1], [expC2|expB2],
          [expN3|expL3])
```

Options: <**expC1**>|<**expB1**> is equivalent to the FOR scope. The condition, given as a string or code block, is evaluated for each record of the source scope.

<**expC2>**|<**expB2**> is equivalent to the WHILE scope. The condition, given as a string or code block, is evaluated for each record from the current position until <expB2> returns FALSE.

<**expN3**>|<**expL3**> is the range of records, providing the same functionality as the ALL, REST and NEXT clause of commands. See chapter 6.1 for the scope values.

Scope: If none of the arguments 1 to 3 are specified, the global scope of the current server is used, see chapter 6.1. If not available and none of the global filters (SET DELETE or SET FILTER) were specified, oRdd:RECCOUNT is returned.

Returns: <**retN**> is the number of records which fall into the given scope.

Related: COUNT, oRdd:RECCOUNT, oRdd:AVERAGE(), oRdd:TOTAL()

oRdd:CREATEDB (expC1...) —> retL *Method*

Creates a new, empty database (and the associated memo file, if memo fields exist) according to the structure in the given array. Equivalent to and invoked from the DBCREATE() function, which should preferably be used. The main reason for this method is to exist as an entity of the RDD, which allows any RDD to create its own, required structure.

```
retL = oRdd:CREATEDB (expC1, expA2, [expN3|expC3])
```

Arguments: <**expC1**> is the name of the new created database. If no extension is specified, it is assumed to be of the standard RDD extension(s) (e.g. .dbf and .dbt). If a path is not specified, the file is placed into the current or the SET DEFAULT directory. The automatic case conversion according to FS_SET() is considered. If a file of the same name exists, it will be overwritten.

<**expA2**> is a two-dimensional array that contains the structure of the database to be created. See detailed description in the DBCREATE() function. The supported field type and field length (DBS_TYPE and DBS_LEN, the 2nd and 3rd element of the subarray) may depend on the RDD used.

Options: <**expN3**>|<**expC3**> are the associated access rights to the file given in an semi-octal notation or as a string (e.g. 664 = "rw-rw-r- -"), or passed from the 4th parameter of DBCREATE(). The semi-octal notation includes three digits (for the

owner, group, world), each in the range 0..7 are required. 0 specifies no permission, 4 = read only, 2 = write only, 6 = read/ write. If not specified, the default "umask" is used.

Returns: <**retL**> signals success, if TRUE, or failure (e.g. in- sufficient directory access rights, wrong field definition etc.) otherwise. The return value is passed to DBCREATE().

Example:

```
LOCAL oNew
if DBCREATE ("newdb", {{"name","C",20,0}, {"date","D",8,0}},
             "CB4CDX", 664)
   oNew := DBSERVER {"newdb",,, "CB4CDX"}
   ? "Records in " + oNew:NAME, oNew:RECCOUNT
   ? "The file is placed in " + oNew:INFO(DBI_FULLPATH)
endif
```

Compatibility: this method in not available in VO. For other than the default "DBFIDX" RDD, differences may apply.

Related: DBCREATE(), oRdd:DBSTRUCT, FILE(), DBSTRUCT(), CREATE FROM

oRdd:CREATEINDEX (expC1...) –> retL *Method*

Creates an index file or, if the RDD supports multiple orders, the first order within an index file. If the index file exists, it is overwritten. If another order name than <expC1> is required for multiple order indexes, use the oRdd:CREATEORDER() method instead. Equivalent to and invoked from the INDEX ON command, DBCREATEINDEX() or ORDCREATE() functions.

`retL = oRdd:CREATEINDEX (expC1, expC2, [expB3], [expL4])`

Arguments: <**expC1**> is the name of the newly created index file. If no extension is specified, it is assumed to be of the standard RDD index extension, i.e. .idx for the DBFIDX driver. If a path is not specified, the file is placed into the current or the SET DEFAULT directory. The automatic case conversion according to FS_SET() is considered. If a file of the same name exists, it will be overwritten. The access right of the database applies for the index file.

<**expC2**> is a string specifying the index expression, e.g. the field name. This expression is stored in the index (order) header and evaluated later on any index (order) access. See additional info in the DBCREATEINDEX() description. To create a descending order, you may use the DESCEND() function, or specify it with the oRdd:SETORDERCONDITION(...) method (alternatively with oRdd: ORDERDESCEND(...) or oRdd:ORDERINFO(DBOI_ISDESC, ...) methods).

Options: <**expB3**> is a code block used for the index (order) creation. Its result should match the <expC2> result, but the code block body can contain additional code

used at creation time only, e.g. to display the creation status. If <expB3> is not specified, the <expC2> argument is used.

<**expL4**> is a logical value equivalent to the UNIQUE clause. If not specified, the current state of SET UNIQUE is used.

Scope: the condition set with oRdd::SETORDERCONDITION() is considered.

Returns: <**retL**> signals success, if TRUE, or failure (e.g. in- sufficient directory access rights, wrong field definition etc.) otherwise. The return value is passed to the DBCREATEINDEX() or ORDCREATE() function, if this was used.

Multiuser: if the database is opened (or the class instantiated) in SHAREd mode, at least FLOCK() is required (or AUTOFLOCK() used, if possible according to oRdd:ConcurrencyControl) to ensure the index integrity. You may disable this lock requirement by assigning a FALSE value to the oRdd:INDEXLOCK instance.

Example:

```
USE address NEW
INDEX ON name+first TO addr1 FOR name="Smith" DESCEND

-is equivalent to:

oMyDbf := DBSERVERNEW ("address")
oMyDbf:SETORDERCONDITION ('name="Smith"',,,,,,,,,.T.)
oMyDbf:CREATEINDEX ("addr1","name+first",{|| name+first},.F.)
oMyDbf:SETORDERCONDITION ()
```

Compatibility: Compatible to VO. For other than the default "DBFIDX" RDD, differences may apply.

Related: INDEX ON, DBCREATEINDEX(), ORDCREATE(), ORDCONDSET(), oRdd:CREATEORDER(), oRdd:INFO(), oRdd:ORDERINFO(), oRdd:SETORDER- CONDITION()

oRdd:CREATEORDER (expC1...) –> retL *Method*

When the RDD supports multiple orders, it creates an additional or replaces an existing order within an existing index, or creates the first order within a new index file. This is similar to and invoked from the ORDCREATE() function or INDEX command. If the RDD supports a single order only, like the default DBFIDX, the functionality is equivalent to the oRdd:CREATEINDEX() method.

```
retL = oRdd:CREATEORDER (expC1, expC2, expC3,
          [expB4], [expL5])
```

Arguments: <**expC1**> is the name of the existing, or a newly created index file. If no extension is specified, it is assumed to be of the standard RDD index extension, i.e. .idx for the DBFIDX driver. If a path is not specified, the file is searched/placed in the current or the SET PATH, SET DEFAULT directory. The automatic case conversion according to FS_SET() is considered.

<**expC2**> is a string specifying the order name within the index file. If single-order is supported only, the order name is equivalent to <expC1>.

<**expC3**> is a string specifying the index expression, e.g. the field name. This expression is stored in the index (order) header and evaluated later on any index (order) access. See additional info in the DBCREATEINDEX() description. To create a descending order, you may use the DESCEND() function, or specify it with the oRdd:SETORDERCONDITION(...) method (alternatively with oRdd: ORDERDESCEND(...) or oRdd:ORDERINFO (DBOI_ISDESC, ...) methods).

Options: <**expB4**> is a code block used for the index (order) creation. Its result should match the <expC3> result, but the code block body can contain additional code used at creation time only, e.g. to display the creation status. If <expB4> is not specified, the <expC3> argument is used.

<**expL5**> is a logical value equivalent to the UNIQUE clause. If not specified, the current state of SET UNIQUE is used.

Scope: the condition set with oRdd::SETORDERCONDITION() is considered.

Returns: <**retL**> signals success, if TRUE, or failure (e.g. in- sufficient directory access rights, wrong field definition etc.) otherwise. The return value is passed to the DBCREATEINDEX() or ORDCREATE() function, if this was used.

Multiuser: if the database is opened (or the class instantiated) in SHAREd mode, at least FLOCK() is required to ensure the index integrity. You may disable the locking check by assigning a FALSE value to the oRdd:INDEXLOCK instance.

Example:

```
oMyDbf := DBSERVERNEW ("address")
oMyDbf:CREATEORDER ("addr1","order2","city",{|| city},.F.)
oMyDbf:SETINDEX ("addr1")
oMyDbf:SETORDER ("city")
? oMyDbf:SEEK("Munich")
```

Compatibility: Compatible to VO. For other than the default "DBFIDX" RDD, differences may apply.

Related: INDEX ON, DBCREATEINDEX(), ORDCREATE(), ORDCONDSET(), oRdd:ORDERINFO(), oRdd:SETORDERCONDITION(), oRdd:DELETEORDER()

oRdd:DBSTRUCT ( ) —> retA ***Method***

Returns a two-dimensional array, compatible to that of the DBSTRUCT() function, containing the structure of this data server. The array length is equal to the number of fields in the server. The subelements contains the field name, type, length and number of decimal places. It is equivalent to and invoked from the DBSTRUCT() function. Please see the DBSTRUCT() function for additional information.

oRdd:DELETE ([expC1...]) —> retL *Method*

Deletes the current record or a range of records according to the given scope. Equivalent to and invoked from the DELETE command or the DBDELETE() function.

```
retN = oRdd:DELETE ([expC1|expB1], [expC2|expB2],
                    [expN3|expL3])
```

Options: <**expC1**>|<**expB1**> is equivalent to the FOR clause of DELETE. The condition, given as a string or code block, is evaluated for each record of the source scope.

<**expC2>**|<**expB2**> is equivalent to the WHILE scope. The condition, given as a string or code block, is evaluated for each record from the current position, until <expB2> returns FALSE.

<**expN3**>|<**expL3**> is the range of records, providing the same functionality as the ALL, REST and NEXT clause of commands. See chapter 6.2 for the scope values.

Scope: If none of the arguments 1 to 3 are specified, the global scope of the current server is used, see chapter 6.1. If not set, the current record is DELETEd.

Returns: <**retL**> signals success, if TRUE, or failure (e.g. failed lock) otherwise.

Multiuser: if the database is opened (or the class instantiated) in SHAREd mode, at least RLOCK() for deleting a single record, or FLOCK() for multiple record processing is required, if oRdd:CONCURRENCY is set to 0.

Related: DELETE, DBDELETE(), DELETED(), oRdd:DELETEALL(), oRdd:DELETED, oRdd:CONCURRENCYCONTROL

oRdd:DELETEALL ( ) —> retL *Method*

Deletes all records of the database table. Equivalent to the DELETE ALL command or the `oRdd:GOTOP() ; oRdd:DELETE( {|| .T.} )` sequence.

Scope: The global scope according to chapter 6.1 does not apply.

Returns: <**retL**> signals success, if TRUE, or failure (e.g. failed lock) otherwise.

Multiuser: if the database is opened (or the class instantiated) in SHAREd mode, FLOCK() is required, if oRdd:CONCURRENCY is set to 0.

Related: DELETE, DBDELETE(), DELETED(), oRdd:DELETEALL(), oRdd:DELETED, oRdd:CONCURRENCYCONTROL

oRdd:DELETED —> retL *Access*

A logical value indicating whether the current record is marked as deleted. Equivalent to and invoked from the DELETED() function.

oRdd:DELETEORDER (expC1...) —> retL *Method*

When the RDD supports multiple orders, it deletes (destroys) the specified order within an existing index file. Similar to and invoked from the ORDDESTROY() function. Not supported by RDDs with single orders, such as the default DBFIDX.

`retL = oRdd:DELETEORDER (expC1|expN1, [expC2])`

Arguments: <**expC1**>|<**expN1**> is the order name within the index file, or the ordinal order number within the active order list.

Options: <**expC2**> is a string specifying the index file name. Use this argument only if the index is not assigned to the server, and the <expN1> argument is not used.

Returns: <**retL**> signals success, if TRUE, or failure (e.g. the index is open by others etc.) otherwise.

Multiuser: an active order cannot be destroyed if the database is opened (or the class instantiated) in SHAREd mode, since it may also be used by others.

Related: ORDDESTROY(), oRdd:ORDERINFO()

oRdd:DRIVER —> expC *Access*

Retrieves the name of the currently used RDD driver/server, e.g. "DBFIDX" for the default driver, equivalent to oRdd:RDDNAME. The server can be specified as an instantiation parameter to the DBserver. If no driver is set, the default driver is used; it can be set by RDDSETDEFAULT() or DBSETDRIVER().

oRdd:EOF —> expL *Access*

A logical value indicating whether there was an attempt to move past the end of the current database file. It also returns TRUE if the database contains no records. Equivalent to and invoked from the EOF() function.

oRdd:ERRINFO —> expO *Access*

Returns the error object (see Error class) of the previous DBserver operation, or NIL if neither RTE, I/O or Developer's error occurred there. May be used e.g. in the RECOVER clause of the BEGIN...END sequence, when the user defined oRdd:ERROR() method executes the BREAK statement.

oRdd:ERROR (expO1...) —> NIL *Method*

Provides a method for handling error conditions raised during database processing. This error handler is automatically called by other methods of this DBserver when an RTE, i/o, Fatal or Developer's error occurs. You may redefine the default error handler by executing the ERRORBLOCK() function.

[NIL =] oRdd:ERROR (expO1, expC2)

Arguments: <**expO1**> is the error object (see Error) passed to the default, or the user-specified error handler.

<**expC2**> is a string specifying the name of the method, stored in the oErr:OPERATION instance.

Returns: always NIL.

Related: ERRORBLOCK(), oRdd:ERRINFO, FS_SET("develop"), FSC.4, file <FlagShip_dir>/system/FSerror.prg

oRdd:EVAL (expB1...) —> retL *Method*

Evaluates a code block for each record matching a scope and condition. Equivalent to and invoked from the DBEVAL() function.

retL = oRdd:EVAL (expB1, [expC2|expB2], [expC3|expB3], [expL4|expN4])

Arguments: <**expB1**> is a code block to execute for each record processed.

Options: <**expC2**>|<**expB2**> is equivalent to the FOR clause. The condition, given as a string or code block, is evaluated for each record of the scope, for which <expB2> returns TRUE.

<**expC3>**|<**expB3**> is equivalent to the WHILE scope. The condition, given as a string or code block, evaluated for each record from the current position until <expB3> returns FALSE.

<**expN4**>|<**expL4**> is the range of records, providing the same functionality as the ALL, REST and NEXT clause of commands. See chapter 6.2 for the scope values.

Scope: If none of the arguments 2 to 4 are specified, the global scope of the current server is used, see chapter 6.1.

Returns: <**retL**> signals success, if TRUE, or failure otherwise.

Related: DBEVAL()

oRdd:FCOUNT —> expN *Access*

A numeric value representing the number of fields in the current server. Equivalent to and invoked from the FCOUNT() function.

oRdd:FIELDGET (expN1...) —> ret *Method*

Retrieves the value of a field using the ordinal position of the field in the database structure or the field name. Equivalent to and invoked from the FIELDGET() function.

ret = oRdd:FIELDGET (expN1|expC1)

Arguments: <**expN1**>|<**expC1**> is the ordinal position of the field in the record structure or the field name. Specifying numeric argument may perform slightly faster on databases with a large number of fields.

Returns: <**ret**> is the value of the specified field. If <expN> is out of range of FCOUNT(), the return value is NIL.

Related: FIELDGET(), <exp> := oRdd:<field>, oRdd:NOIVARGET()

oRdd:FIELDINFO (expN1...) –> ret *Method*

Retrieves information about a field, similar to oRdd:DBSTRUCT() method.

```
ret = oRdd:FIELDINFO (expN1, expN2|expC2)
```

Arguments: <**expN1**> specifies the required type of the field information, given as a constant (see "rddsys.fh") or a numeric value:

Constant	Value	Ret	Returns
DBS_NAME	1	retC	name of the field
DBS_TYPE	2	retC	type of the field
DBS_LEN	3	retN	length of the field
DBS_DEC	4	retN	number of decimal places for the field

<**expN2**>|<**expC2**> is the ordinal position of the field in the record structure or the field name. Specifying a numeric argument may perform slightly faster on databases with a large number of fields.

Returns: <**retN**>|<**retC**> is the required field information, or NIL on error.

Compatibility: RDDs other than the default DataServer may contain additional field features.

Related: DBSTRUCT(), oRdd:DBSTRUCT(), oRdd:INFO()

oRdd:FIELDNAME (expN1) –> retC *Method*

Retrieves the name of the field at the given ordinal position. Equivalent to the FIELDNAME() function.

```
retC = oRdd:FIELDPOS (expN1)
```

Arguments: <**expN1**> is the ordinal position of the requested field.

Returns: <**retC**> is the field name in upper case, or "" on error.

Related: FFIELDNAME(), IELDPOS(), oRdd:DBSTRUCT(), oRdd:INFO(), oRdd:FIELDINFO()

oRdd:FIELDPOS (expC1) –> retN *Method*

Retrieves the ordinal position of the specified field in the database structure. Equivalent to and invoked from the FIELDPOS() function.

`retN = oRdd:FIELDPOS (expC1)`

Arguments: <**expC1**> is the requested field name. The input is automatically converted to upper case and abbreviated, if necessary.

Returns: <**retN**> is the ordinal position in the structure starting with 1. It corresponds to the element number of the array returned by oRdd:DBSTRUCT() method. If <expC> is not a valid field of the DBserver, the return value is 0.

Related: FIELDPOS(), oRdd:DBSTRUCT(), oRdd:INFO(), oRdd:FIELDINFO()

oRdd:FIELDPUT (expN1...) –> ret *Method*

Assigns the given value to a field specified by the ordinal position in the database structure or by the field name. Equivalent to and invoked from the FIELDPUT() function.

`ret = oRdd:FIELDPUT (expN1|expC1, exp2)`

Arguments: <**expN1**>|<**expC1**> is the ordinal position of the field in the record structure, or the field name. Specifying a numeric argument may perform slightly faster on databases with a large number of fields.

<**exp2**> is the value to be assigned to the field. The data type must match the data type of the field.

Returns: <**ret**> is the newly assigned value, if the operation was successful, or NIL otherwise.

Multiuser: if the database is opened in SHAREd mode, at least RLOCK() is required, if oRdd:CONCURRENCY is set to 0.

Related: FIELDPUT(), oRdd:RLOCK(), oRdd:CONCURRENCY, oRdd:<field> := <exp>, oRdd:NOIVARPUT()

oRdd:FILTER <–> expC *Access/Assign*

Determines or sets the global filter condition string. Equivalent to and invoked from the SET FILTER command, DBSETFILTER() function, or oRdd:INFO(DBI_DBFILTER) method. Note, that this instance may return an empty string "" even with an active filter, when only the filter code block was assigned. Usually, only the filter code block is evaluated for scoping (see oRdd:INFO (DBI_FILTERBLOCK)), this instance has an informative character only. To reset the filter condition, use the SET FILTER TO command, DBCLEARFILTER() function or the oRdd:CLEARFILTER() method.

Related: SET FILTER, DBSETFILTER(), DBCLEARFILTER(), oRdd:CLEARFILTER(), oRdd:INFO()

oRdd:FLOCK ( ) —> retL *Method*

Locks all records of the table (database) to perform a global write access or to protect it against write access from another user or process. Meaningful in SHAREd mode only. Equivalent to and invoked from the FLOCK() function.

Returns: <**retL**> signals success. On error, FALSE is returned, which reports that the database is FLOCKed or RLOCKed by another user, application or DBserver.

Multiuser: if the database is opened in SHAREd mode, and the automatic locking is disabled (oRdd:CONCURRENCY is set to 0), FLOCK() is required prior to write accessing a range of records or prior to issuing the INDEX ON command or the oRdd:CREATEORDER(), oRdd:CREATEINDEX() methods. If oRdd:CONCURRENCY is active, FLOCK() overrides the automatic locking and the lock remains active until oRdd:UNLOCK() is executed. If the database was open in EXCLUSIVE mode (the default), no locking is required and FLOCK() is ignored.

Related: FLOCK(), oRdd:RLOCK(), oRdd:UNLOCK(), oRdd:CONCURRENCY

oRdd:FORBLOCK <—> expB|NIL *Access/Assign*

The FOR block is a component of the general server scope, described in chapter 1. It affects several bulk processing methods if they are called with no explicit scope. The FOR block can be specified as a code block or a string. Accessing this instance always returns a code block or NIL if not set. To reset the global FOR block, assign NIL to it or execute oRdd:CLEARSCOPE() for a global reset.

Related: oRdd:WHILEBLOCK, oRdd:SCOPE, oRdd:CLEARSCOPE()

oRdd:FOUND —> expL *Access*

Determines the success of a previously executed SEEK, LOCATE or CONTINUE command, function or method. Equivalent to and invoked from the FOUND() function.

Related: FOUND(), oRdd:SEEK(), oRdd:LOCATE(), oRdd:CONTINUE(), oRdd:EOF()

oRdd:GETARRAY ([expN1...]) —> retA *Method*

Assigns the values of several records of the specified field into a one-dimensional array. This method is a subset of the oRdd:GETARRAYFIELDS() method, which retrieves several fields per record at once.

```
retA = oRdd:GETARRAY ([expN1], [expC2|expN2],
          [exp3])
```

Options: <**expN1**> is the maximum number of records that should be retrieved and the size of the returned array. If omitted, 100 records is the default.

<**expN2**>|<**expC2**> is the ordinal position of the retrieved field in the record structure, or the field name. If omitted, values of the first field are stored into the array.

<**exp3**> is a "seek" expression for searching the first retrieved value, equivalent to executing the oRdd:SEEK() method with the same parameter. If not specified, the storage of field values starts at the current record, considering the global scope, according to chapter 6.1.

Returns: <**retA**> is a one-dimensional array with the retrieved field values. If the end-of-file is reached or the scope is out of range before the given record count is proceeded, the array size is adapted accordingly. If no record is found, LEN(retA) is 0.

Example:

```
oMyDbf := DBSERVER {"article", DB_SHARED, DB_READONLY}
if oMyDbf:USED .and. oMyDbf:SETINDEX("articleno")
   aArtNo := GETARRAY (500, "ArtNo", 2001)
   oMyDbf:Close()
endif
```

Related: FIELDGET(), oRdd:FIELDGET(), oRdd:GETARRFIELDS(), oRdd:GET-LOOKUPTABLE()

oRdd:GETARRFIELDS ([expN1...]) —> retA ***Method***

Assigns the values of several records of the specified fields into a multi-dimensional array. This method is a superset of the oRdd:GETARRAY() method.

```
retA = oRdd:GETARRFIELDS ([expN1],
          [expA2|expC2|expN2], [exp3])
```

Options: <**expN1**> is the maximum number of records that should be retrieved and the size of the returned array. If omitted, 100 records is the default.

<**expA2**>|<**expN2**>|<**expC2**> is the ordinal position of the retrieved field in the record structure, or the field name, or an array containing field names or ordinal positions. If omitted, or the array element is NIL, values of the first field are stored into the array.

<**exp3**> is a "seek" expression for searching the first retrieved value, equivalent to executing the oRdd:SEEK() method with the same parameter. If not specified, the storage of field values starts at the current record, considering the global scope, according to chapter 6.1.

Returns: <**retA**> is a multi-dimensional array with the retrieved field values. If the end-of-file is reached or the scope is out of range before the given record count is processed, the array size is adapted accordingly. If no record is found, LEN(retA) is 0.

Example:

```
oMyDbf := DBSERVERNEW ("address")
if oMyDbf:USED .and. oMyDbf:SETINDEX("addrname")
   aField := GETARRFIELDS (50, {"First","Name",4}, "SMITH")
   AEVAL (aField, {|elem| QOUT(elem[1], elem[2], elem[3]) })
   wait
   myTbrowse (aField)
endif
```

Compatibility: This method is not available in VO.

Related: FIELDGET(), oRdd:FIELDGET(), oRdd:GETARRAAY(), oRdd:GET-LOOKUPTABLE()

oRdd:GETLOCATE ( ) —> retB *Method*

Retrieves the code block of the current LOCATE condition, or returns NIL if no previous LOCATE command or oRdd:LOCATE() method was executed.

Returns: <**retB**> is the code block of the FOR clause of LOCATE command, or the code block used in the oRdd:LOCATE() method. The condition may also be retrieved or set by the oRdd:INFO(DBI_GETSCOPE) method.

Related: LOCATE, CONTINUE, oRdd:LOCATE(), oRdd:CONTINUE(), oRdd:INFO()

oRdd:GETLOOKUPTABLE ([expN1...]) —> retA *Method*

Assigns the values of several records of the specified fields into a two-dimensional array. This method is a subset of the oRdd:GETARRAYFIELDS() method with a slightly different syntax.

`retA = oRdd:GETLOOKUPTABLE ([expN1], [expC2|expN2], [expC3|expN3], [exp4])`

Options: <**expN1**> is the maximum number of records that should be retrieved and the size of the returned array. If omitted, 100 records is the default.

<**expN2**>|<**expC2**> is the ordinal position of the first retrieved field in the record structure, or the field name. If omitted, values of the first field are stored into the array.

<**expN3**>|<**expC3**> is the ordinal position of the second retrieved field in the record structure, or the field name. If omitted, values of the second field are stored into the array.

<**exp4**> is a "seek" expression for searching for the first value to retrieve, equivalent to executing the oRdd:SEEK() method with the same parameter. If not specified, the storage of field values starts at the current record, considering the global scope, according to chapter 6.1.

Returns: <**retA**> is a two-dimensional array with the retrieved field values. If the end-of-file is reached or the scope is out of range before the given record count is proceeded, the array size is adapted accordingly. If no record is found, the LEN(retA) is 0.

Example:

```
oMyDbf := DBSERVERNEW ("address")
if oMyDbf:USED .and. oMyDbf:SETINDEX("addrname")
   aField := GETLOOKUPTABLE (500, "First", "Name", "SMITH")
   ? "All Smith's:"
   AEVAL (aField, {|elem| QOUT(elem[1], elem[2]) })
endif
```

Compatibility: The VO symbols for the 2nd and 3rd argument are not supported by FS.

Related: FIELDGET(), oRdd:FIELDGET(), oRdd:GETARRAAY(), oRdd:GET-ARRFIELDS()

oRdd:GOBOTTOM ( ) —> retL *Method*

Moves the database pointer to the last logical record. Equivalent to and invoked from the GO BOTTOM command and the DBGOBOTTOM() function.

`retL = oRdd:GOBOTTOM ()`

Returns: <**retL**> signals success. On error, FALSE is returned, which reports that the database is empty, or the conditional index, general filters or scope do not match any record of the database.

Related: GOBOTTOM, oRdd:GOTO(), oRdd:GOTOP(), oRdd:SKIP()

oRdd:GOTO (expN1) —> retL *Method*

Moves the database pointer to the specified record. Equivalent to and invoked from the GO TO command and the DBGOTO() function.

`retL = oRdd:GOTO (expN1)`

Arguments: <**expN1**> is the record number at which the server should be positioned. Neither the filters, nor global scope according to chapter 6.1 apply for this fix-record movement.

Returns: <**retL**> signals success. On error, FALSE is returned, which reports that the specified record is out of the current database size.

Related: GOTO, oRdd:GOTOP(), oRdd:GOBOTTOM(), oRdd:SKIP()

oRdd:GOTOP ( ) —> retL *Method*

Moves the database pointer to the first logical record. Equivalent to and invoked from the GO TOP command and DBGOTOP() function.

```
retL = oRdd:GOTOP ()
```

Returns: <**retL**> signals the success. On error, FALSE is returned, which reports that the database is empty, or the conditional index, general filters or scope do not match any record of the database.

Related: GOTOP, oRdd:GOTO(), oRdd:GOBOTTOM(), oRdd:SKIP()

oRdd:HEADER —> expN *Access*

Determines the size of the database file header in bytes. Equivalent to and invoked from the HEADER() function or the oRdd:INFO(DBI_GETHEADERSIZE) method.

Related: HEADER(), oRdd:INFO()

oRdd:INDEXCHECK ([expN1]) —> retN *Method*

Checks the index integrity, whether the database is consistent with its associated indices. Equivalent to and invoked from the INDEXCHECK() function.

```
retN = oRdd:INDEXCHECK ([expN1])
```

Options: <**expN1**> is the ordinal position of the index in the list of currently open indices, starting at one. Zero specifies the current controlling index, which is the default value, if the argument is not given.

Returns: <**retN**> is a numeric value indicating the integrity status of the index file: -1 signals error, 0 a correct integrity, 1 a possible corruption, 2 a found integrity corruption. See the additional description in the INDEXCHECK() function and in section LNG.4.5.

Compatibility: Not available in VO.

Related: INDEXCHECK()

oRdd:INDEXCOUNT —> expN *Access*

Determines the number of open indices (0...15) used in this RDD. Equivalent to and invoked from the INDEXCOUNT() function.

Compatibility: Not available in VO.

Related: INDEXEXT(), oRdd:INFO()

oRdd:INDEXEXT —> expC *Access*

Determines the default extension of index files of this RDD driver, e.g. ".idx" for the DBFIDX server. Equivalent to and invoked from the INDEXEXT() function or the oRdd:INFO(DBI_INDEXEXT) method.

Related: INDEXEXT(), oRdd:INFO()

oRdd:INDEXKEY —> expC *Access*

Determines the key expression of the current controlling index. Equivalent to the INDEXKEY() function or the oRdd:INDEXKEY(0) method.

Compatibility: Not available in VO.

Related: INDEXKEY(), oRdd:INDEXKEY()

oRdd:INDEXKEY (expN1) —> retC *Method*

Determines the key expression of a specified index. Equivalent to and invoked from the INDEXKEY() function.

```
retC = oRdd:INDEXKEY (expN1)
```

Arguments: <**expN1**> is the ordinal position of the index in the list of currently open indexes, starting at one. Zero specifies the current controlling index, equivalent to the oRdd:INDEXKEY instance.

Returns: <**retC**> is the required key expression, stored in the index file header by oRdd:CREATEINDEX() or oRdd:CREATEORDER(). On error, i.e. if no index file is open, null string "" is returned.

Related: INDEXKEY(), oRdd:INDEXKEY

oRdd:INDEXORD ( ) —> retN *Method*

Returns the ordinal position of the controlling order in the order list. Equivalent to and invoked from the INDEXORD() function.

Returns: <**retN**> is equal to the position of the controlling index in the list of open indexes for the current work area. A zero value indicates that the current database is treated in the natural order.

Related: INDEXORD(), SET ORDER, oRdd:SETORDER(), oRdd:SETINDEX()

oRdd:INFO (expN1...) —> ret **Method**

Returns and optionally changes information about a data server. Additional information is also available by invoking the oRdd:ORDERINFO() method.

`ret = oRdd:INFO (expN1, [exp2])`

Arguments: <**expN1**> is a numeric value or a constant (preferred, see "rddsys.fh") specifying the type of the information.

Constant	Value	Ret	Returns	Change
DBI_ALIAS	33*	retC	alias name = oRdd:ALIAS	yes
DBI_ACCESSRIGHTS	5000*	retN	dbf access rights from DBSERVERNEW() in octal representation,e.g. 644 for rw-r--r--	no
DBI_BOF	26	retL	BOF status = oRdd:BOF	yes
DBI_CANPUTREC	2	retL	Replace supported by RDD ?	no
DBI_CHILDCOUNT	22	retN	number of open relations	no
DBI_DBFILTER	28	retC	global filter = oRdd:FILTER	yes
DBI_DB_VERSION	101	retN	release of the host RDD driver	no
DBI_EOF	27	retL	EOF status = oRdd:EOF	yes
DBI_FCOUNT	30	retN	no. of fields = oRdd:FCOUNT	no
DBI_FILEHANDLE	23	retN	the used file handle	no
DBI_FILTERBLOCK	5001	retB	global filter, code block, set by oRdd:SETFILTER()	yes
DBI_FOUND	29	retL	FOUND status = oRdd:FOUND	yes
DBI_FULLPATH	24	retC	dbf name incl.path	no
DBI_GETDELIMITER	5	retC	delimiter for oRdd:COPYDELI../ oRdd:APPENDDELIMITED()	yes
DBI_GETHEADERSIZE	3	retN	header size = oRdd:HEADER	no
DBI_GETLOCKARRAY	8	retA	array of RLOCKs() = oRdd:RLOCKLIST	no
DBI_GETRECSIZE	7	retN	record size = oRdd:RECSIZE	no
DBI_GETSCOPE	34	retB	locate condition = oRdd:GETLOCATE()	yes
DBI_HAS_DBT	5020*	retL	.DBT memo file structure ?	no
DBI_HAS_DBV	5021*	retL	.DBV memo file structure ?	no
DBI_HAS_FPT	5022*	retL	.FPT memo file structure ?	no
DBI_INDEXEXT	5002*	retC	default extension of the index file, e.g. ".idx". See oRdd:ORDERINFO()	yes
DBI_ISANSI	25	retL	the database supports ANSI PC-8 char set, ISO otherwise	no
DBI_ISDBF	1	retL	is the .dbf format supported?	no
DBI_ISFLOCK	20	retL	is FLOCK() active?	no
DBI_LASTUPDATE	4	retD	date of last .dbf modif.	no
DBI_LOCKCOUNT	31	retN	number of RLOCKed() records	no
DBI_LOCK_MODE	5016*	retN	locking scheme currently used	no
DBI_MEMOBLOCKSIZE	39	retN	block size of .dbt file	no

DBI_MEMOEXT	37	retC	default extension of the memo file, e.g. ".dbt"	yes
DBI_MEMOHANDLE	38	retN	handle no. of memo file	no
DBI_RDD_VERSION	102	retN	release no. of this RDD	no
DBI_READONLY	203*	retL	is database open read-only?	no
DBI_RELAT_COUNT	5017*	retN	number of relations in current WA	no
DBI_SETDELIMITER	6	retC	equiv.to DBI_GETDELIMITER	yes
DBI_SHARED	36	retL	shared usage = oRdd:SHARED	no
DBI_TABLEEXT	9	retC	default extension of the database file, e.g. ".dbf"	yes
DBI_VALIDBUFFER	32	retL	is the access buffer valid?	no
DBI_USER	1000*	retN	users active = USERSDBF()	no

Options: **<exp2**> is the new value to be set. Considered only when the value is changeable (see table above).

Compatibility: Items marked with "*" perform extended functionality or are not supported by VO.

Returns: <**ret**> is the required information or the current setting before resetting. If <expN1> is invalid, NIL is returned.

Example:

```
oMyDbf := DBSERVERNEW ("address")
// or: USE address ; oMyDbf := DbObject()
if oMyDbf:USED
   ? "The " + oMyDbf:INFO(DBI_FULLPATH) + ;
     " database is open in " + ;
     if (oMyDbf:SHARED, "", "non-") + "shared, read-" + ;
     if (oMyDbf:READONLY, "only", "write") + " mode and " + ;
     str(oMyDbf:INFO(DBI_ACCESSRIGHTS),3) + " permission."
   ? "The RDD " + oMyDbf:RDDNAME + ", rel." + ;
     ltrim(str(oMyDbf:INFO(DBI_RDD_VERSION),6,2) + " does " + ;
     if (oMyDbf:INFO(DBI_CANPUTREC), "", "not") + ;
     " support APPENDing and REPLACEing."
endif
```

Related: most of these information is also handled by other instances and methods or functions.

oRdd:INIT (expC1...) —> SELF ***Method***

Initializes the object and its default values, opens the database, passes data to the FlagShip run-time system. This method is invoked automatically from DBSERVERNEW(), you should **not** invoke it manually. See the additional description in section LNG.11.3 and RDD.2.3.1. To ensure its functionality, a class inheriting this one should invoke the SUPER:INIT(...) method, if a separate oRdd:INIT() method is specified.

```
retO = oRdd:INIT (expC1, [expL2], [expL3], [expC4],
          [expA5], [expL6])
```

Arguments: <**expC1**> is equivalent to the first DBSERVERNEW() argument.

Options: <**expL2**>...<**expL6**> are equivalent to the 2nd to 6th DBSERVERNEW() optional arguments.

Returns: <**retO**> is the server object SELF.

Related: DBSERVERNEW(), oRdd:AXIT(), <FlagShip_dir>/system/rddcb4a.c available in the .../RDDcb4.tar.Z file.

oRdd:ISRELATION <—> expL *Access/Assign*

Determines if a relation is active (TRUE), or temporarily activates/deactivates all relations set with the oRdd:SETRELATION() method or SET RELATION command. Assigning FALSE temporarily disables the relation movement and evaluation, whilst TRUE activates it again.

Compatibility: not available in CA/VO.

Related: oRdd:RELATION(), oRdd:SETRELATION(), oRdd:RELATIONOBJECT(), SET RELATION

oRdd:JOIN (expC1...) —> retL *Method*

Creates a new database by merging certain specified records with another database (or DBserver). Equivalent to the JOIN command.

```
retL = oRdd:JOIN (expC1|expO1, expC2|expO2,
          [expA3], [expB4])
```

Arguments: <**expC1**>|<**expO1**> specifies the second source, an already opened database, which should be merged with the current DBserver. The argument is either the alias name <expC1>, or the RDD object <expO1>.

<**expC2**>|<**expO2**> is the database file name or RDD object of the target database, holding the merging results.

Options: <**expA3**> is an array of fields to be included in the join operation, similar to the FIELDS clause of JOIN. If omitted, all records are included.

<**expB4**> is the condition evaluated for each record in the scope; if TRUE, the record is included in the processing. It provides the same functionality as the FOR clause of record processing commands. If not specified, the global scope according to chapter 6.1 is considered. If not available, the target database contains the product of records of both source files.

Returns: <**retL**> signals success, otherwise FALSE is returned.

Multiuser: if the <expO2> database is opened (or the class instantiated) in SHAREd mode, at least FLOCK() is required, if oRdd:CONCURRENCY is set to 0. The newly created database <expC2> is opened in exclusive mode.

Related: JOIN

oRdd:LASTREC —> expN *Access*

Specifies the number of records in the current database. Filtering commands such as SET FILTER or SET DELETED have no effect on the return value. Equivalent to and invoked from the LASTREC() or RECCOUNT() function.

Related: LASTREC(), RECCOUNT()

oRdd:LOCATE ([expC1...]) —> retL *Method*

Searches for the first record meeting the specified condition. Equivalent to and invoked from the LOCATE command.

```
retL = oRdd:LOCATE ([expC1|expB1], [expC2|expB2],
                    [expN3|expL3])
```

Options: <**expC1**>|<**expB1**> is equivalent to the FOR scope of LOCATE. The condition, given as a string or code block, is evaluated for each record of the scope. This condition is used for subsequent CONTINUE operations and may be retrieved or modified by the oRdd:GETLOCATE(), oRdd:INFO (DBI_GETSCOPE) or cleared by the oRdd:CLEARLOCATE() method.

<**expC2**>|<**expB2**> is equivalent to the WHILE scope. The condition, given as a string or code block, is evaluated for each record from the current position until <expB2> returns FALSE. This condition is not used for subsequent CONTINUE operations.

<**expN3**>|<**expL3**> is the range of records, providing the same functionality as the ALL, REST and NEXT clause of commands. See chapter 6.2 for the scope values. This condition is not used for subsequent CONTINUE operations.

Scope: If none of the arguments 1 to 3 are specified, the global source server scope is used, see chapter 6.1.

Returns: <**retL**> signals success, if TRUE, or failure otherwise. If the search was successful, the matching record becomes the current record, and this method, the FOUND() function or oRdd:FOUND instance returns TRUE. If not found, the record pointer is positioned on EOF or the next record outside the FOR scope, and FALSE is returned.

Related: LOCATE, CONTINUE, oRdd:CONTINUE(), oRdd:CLEARLOCATE(), oRdd:FOUND, oRdd:INFO()

oRdd:LOCKCURRENTRECORD ( ) —> retL *Method*

Locks the current record to perform a write access or to protect this record against write access from another user or process. Meaningful in SHAREd mode only. Equivalent to and invoked from the RLOCK() function, or identical to invoking the oRdd:RLOCK(oRdd:RECNO) method. The superset of this method is oRdd:RLOCKVERIFY().

Returns: <**retL**> signals success. On error, FALSE is returned, which reports that the database is FLOCKed or the record RLOCKed by another user, application or DBserver, or that the record pointer is on EOF.

Multiuser: if the database is opened in SHAREd mode, and the automatic locking is disabled (oRdd:CONCURRENCY is set to 0), RLOCK() is required prior to write accessing a single record. If oRdd:CONCURRENCY is active, RLOCK() overrides the automatic locking and the lock remains active until oRdd:UNLOCK() is executed. If the database is opened in EXCLUSIVE mode (the default), no locking is required and RLOCK() is ignored.

Related: RLOCK(), oRdd:RLOCK(), Rdd:RLOCKLIST, oRdd:RLOCKVERIFY(), oRdd:UNLOCK(), oRdd:CONCURRENCY

oRdd:LUPDATE —> expD *Access*

Retrieves the last modification date of the database file. Equivalent to and invoked from the LUPDATE() function.

Related: LUPDATE(), oRdd:INFO(DBI_LASTUPDATE)

oRdd:NAME —> expC *Access*

Returns the main part of the used database file name. It returns e.g. "MyFile" for the obj := DBSERVER {"MyFile"} usage, also if a path or extension was specified; or "myfile" when the file is named "myfile.dbf" and fs_set("lower",.T.) is active. Refer to oRdd:INFO(DBI_FULLPATH) for the full naming information.

oRdd:NOIVARGET (expC1) —> ret *Method*

Provides general error interception that is automatically called (in any class) whenever an access reference is made to a non-existent exported instance variable or Access method, see also LNG.2.11.3. In the DBServer class, it is used to implement the access to a virtual field variable (since the name is known first at run-time). This method is called by the FIELDGET() method, it should **not** be called directly from the application.

```
ret = oRdd:NOIVARGET (expC1)
```

Arguments: <**expC1**> is the name of the referenced instance, e.g. the field name.

Returns: <**ret**> is the contents of the virtual instance (i.e. the field contents). If the instance (the field) or Access method does not exist, a run-time error occurs and is reported via the oRdd:ERROR() method if available, or via the standard error handler otherwise.

Compatibility: symbolic names of VO are not supported.

Related: oRdd:NOIVARPUT(), oRdd:FIELDGET(), oRdd:ERROR(), <ret> := <expC1>

oRdd:NOIVARPUT (expC1...) –> retL *Method*

Provides general error interception that is automatically called (in any class) whenever an assign reference is made to a non-existent exported instance variable or Assign method, see also LNG.2.11.3. In the DBServer class, it is used to implement the assign to a virtual field variable (since the name is known first at run-time). This method is called by the FIELDPUT() method, it should **not** be called directly from the application.

```
retL = oRdd:NOIVARPUT (expC1, exp2)
```

Arguments: <**expC1**> is the name of the referenced instance, e.g. the field name.

<**exp2**> is the value to assign to the field. The data type must be equivalent or compatible to the datatype of the field.

Returns: <**retL**> signals success. On error, FALSE is returned, which reports e.g. a lock failure according to oRdd: FIELDPUT().

Compatibility: symbolic names of VO are not supported.

Related: oRdd:NOIVARGET(), oRdd:FIELDPUT(), <expC1> := <exp2>, REPLACE

oRdd:NOMETHOD (expC1...) –> ret *Method*

Provides general error interception that is automatically called (in any class) whenever a method reference is made to a non-declared method, see also LNG.2.11.3. In the DBServer class, it is used to display an RTE error, if the method does not exist. This method is called by the FlagShip run-time system, it should **not** be called directly from the application.

```
ret = oRdd:NOMETHOD (expC1, exp2...)
```

Arguments: <**expC1**> is the name of the referenced method.

<**exp2**>...<**expN**> are the parameters passed to the virtual method.

Returns: <**ret**> is the value returned from the variable method. If no such method exist, a run-time error occurs and is reported via the oRdd:ERROR() method if available, or via the standard error handler otherwise.

Compatibility: not supported by VO.

oRdd:ORDERBOTTOMSCOPE <—> exp *Access/Assign*

A value, controlling the last visible index key of the current selected order and is considered for all database movement operations. Together with oRdd:ORDERTOPSCOPE, it allows to "filter" the index for a specified range of index keys. Assigning NIL (the default value) to the instance will reset the bottom boundary to the last available key. See example in oRdd:ORDERSCOPE().

Related: SKIP, GOBOTTOM, oRdd:ORDERTOPSCOPE, oRdd:ORDERCOPE()

oRdd:ORDERDESCEND ([expN1...]) —> retL *Method*

Returns or dynamically changes the descending flag of an order, regardless of the original indexing strategy.

```
retL = oRdd:ORDERDESCEND ([expN1|expC1], [expC2],
          [expL3])
```

Options: <**expN1**>|<**expC1**> is the ordinal position of the order in the list of open indices (similar to SET ORDER), or the order name. If omitted, the controlling order is assumed.

<**expC2**> is the name of the index file, if <expC1> was specified.

<**expL3**> is the new, temporary descending flag. Specifying TRUE dynamically turns on the descending sequence, regardless of the indexing method. It is considered for all movement and search operations. Similarly, FALSE dynamically activates an ascending order sequence. This flag does not affect the stored/replaced sequence order, but the current database movement only.

Returns: <**retL**> is the current descending flag, or the value of the previous setting.

Example:

```
oAdr := DBSERVER {"address"}
if !oAdr:USED ; return ; endif
oAdr:SETINDEX("adrname")

flag := oAdr:ORDERDESCEND()
? "Default" ; oAdr:GOTOP() ; mydisplay (oAdr, 10, flag)
flag := oAdr:ORDERDESCEND(,,!flag)
? "New" ; oAdr:GOTOP() ; mydisplay (oAdr, 10, flag)

function mydisplay (oRdd AS DBSERVER, max, flag)
LOCAL oii := 1
?? " = " + if(flag, "des","as") + "cending order"
oRdd:EVAL ({|| QOUT(oRdd:name)},,{|| ii++, if(ii <= max,.T.,F.)})
return NIL
```

Related: SKIP, GOTOP, GOBOTTOM, SEEK, LOCATE

oRdd:ORDERINFO (expN1...) —> ret *Method*

Returns and optionally changes information about orders and index files.

```
ret = oRdd:ORDERINFO (expN1, [expC2],
          [expN3|expC3], [exp4])
```

Arguments: <**expN1**> is a numeric value or constant (preferred, see "rddsys.fh") specifying the type of information.

rddsys.fh	Value	Ret	Returns	Change
DBOI_CONDITION	1	retC	order condition string	no
DBOI_CUSTOM	45	retL	custom RDD order built ?	*yes
DBOI_DBFNAME	5001	retC	FS only, dbf name of INDEX	*no
DBOI_EXPRESSION	2	retC	order expression string	no
DBOI_FILEHANDLE	21	retN	file handle no	*no
DBOI_FULLPATH	20	retC	full path & name	no
DBOI_HPLOCKING	29	retL	high perform CA/VO lock	*no
DBOI_INDEXCHECK	5002	retN	FS only, index ok?	*no
DBOI_INDEXEXT	8	retC	index extension (".cdx")	yes
DBOI_INDEXNAME	7	retC	index file name main part	no
DBOI_ISCOND	23	retL	condition flag set ?	no
DBOI_ISDESC	22	retL	descending flag set?	no
DBOI_KEYCOUNT	26	retN	no.of records in the order	*no
DBOI_KEYDEC	28	retN	no.of decimals in the key	no
DBOI_KEYSINCLUDED	48	retN	no.of keys of cond.index	no
DBOI_KEYSIZE	25	retN	size of the key	no
DBOI_KEYTYPE	24	retC	type of the key	no
DBOI_KEYVAL	38	ret	value of the current key	*no
DBOI_LOCKOFFSET	35	retL	lock offset NewIndexLock()	*no
DBOI_NAME	5	retC	order name	no
DBOI_NUMBER	6	retN	posit.of the order in list	no
DBOI_ORDERCOUNT	44	retN	no.of orders in the file	no
DBOI_POSITION	3	retN	logical record no.in order	*no
DBOI_RECNO	4	retN	physic. record no.in order	no
DBOI_SCOPEBOTTOM	40	ret	bottom boundary value or NIL	*no
DBOI_SCOPETOP	39	ret	top boundary value or NIL	*no
DBOI_SETCODEBLOCK	27	retB	key converted to code block	no
DBOI_UNIQUE	43	retL	unique flag set?	no

Options: <**expC2**> is the name of the index file, if <expN3> is specified.

<**expN3**>|<**expC3**> is the ordinal number specifying the order position in the open order list, or the order name. If neither argument 2 nor argument 3 is specified or is NIL, the controlling order is assumed.

<**exp4**> is the new value to be set.

Returns: <**ret**> is the required information or the current value setting before resetting. If <expN1> is invalid, NIL is returned.

Example:

```
SET PATH TO ./;/usr/data;/tmp
oAdr := DBSERVER {"address"}
if !oAdr:USED ; return ; endif
oAdr:SETINDEX("adrname")
oAdr:SETINDEX("adrcity")

? oAdr:INFO(DBI_NAME)                   // "address"
? oAdr:ORDERINFO(DBOI_FULLPATH)         // "/usr/data/adrname.idx"
oAdr:SETORDER(2)
? oAdr:ORDERINFO(DBOI_FULLPATH)         // "/tmp/adrcity.idx"
? oAdr:ORDERINFO(DBOI_EXPRESSION)       // "str(zip,5)+name"
? INDEXKEY()                            // "str(zip,5)+name"
```

Compatibility: The items marked with "*" cannot be set by the default RDD driver.

Related: the results are equivalent to several Access/Assign instances or DBserver Methods, especially oRdd:ORDER*(). For additional settings, see also oRdd:INFO(), oRdd:RECINFO() and oRdd:RDDINFO() methods as well as the SET() and INDEX*() functions .

oRdd:ORDERISUNIQUE ([expN1...]) —> retL *Method*

Returns the status of the unique flag for a given order. Equivalent to invoking the oRdd:ORDERINFO(DBOI_UNIQUE) method.

`retL = oRdd:ORDERISUNIQUE ([expN1|expC1], [expC2])`

Options: <**expN1**>|<**expC1**> is the ordinal position of the order in the list of open indices (similar to SET ORDER), or the order name. If omitted, the controlling order is assumed.

<**expC2**> is the name of the index file, if <expC1> was specified.

Returns: <**retL**> is the unique flag of the order, set on indexing.

Related: oRdd:ORDERINFO(), oRdd:SETORDER(), oRdd:CREATEORDER(), oRdd:CREATEINDEX(), INDEX ON, SET UNIQUE

oRdd:ORDERKEYCOUNT ([expN1...]) —> retN *Method*

Returns the number of keys (records) in an order. Equivalent to invoking the oRdd:ORDERINFO(DBOI_KEYCOUNT) method.

`retN = oRdd:ORDERKEYCOUNT ([expN1|expC1], [expC2])`

Options: <**expN1**>|<**expC1**> is the ordinal position of the order in the list of open indices (similar to SET ORDER), or the order name. If omitted, the controlling order is assumed.

<**expC2**> is the name of the index file, if <expC1> was specified.

Returns: <**retN**> is the number of keys (database records) included in the order. If the order is not conditional or unique, and no scope has been set for it, the return value is equal to oRdd:RECCOUNT. Note, that depending on the RDD, the key counting may be a time consuming task for large indices.

Related: oRdd:ORDERINFO(), oRdd:SETORDER(), oRdd:CREATEORDER(), oRdd:CREATEINDEX(), oRdd:RECCOUNT, INDEX ON, SET UNIQUE

oRdd:ORDERKEYGOTO (expN1) —> retL *Method*

Moves to a record specified by its logical record number in the controlling order.

`retL = oRdd:ORDERKEYGOTO (expN1)`

Arguments: <**expN1**> is the logical record number. If the value specified does not satisfy the scope or FOR condition for the order, the record pointer is positioned at the end-of-file.

Returns: <**retL**> reports success. It returns FALSE if the <expN1> is out of range.

Related: oRdd:SKIP(), oRdd:GOTOP(), oRdd:GOBOTTOM(), oRdd:ORDERKEYCOUNT(), oRdd:ORDERKEYNO()

oRdd:ORDERKEYNO ([expN1...]) —> retN *Method*

Returns the logical record number of the current record. Equivalent to invoking the oRdd:ORDERINFO(DBOI_POSITION) method.

`retN = oRdd:ORDERKEYNO ([expN1|expC1], [expC2])`

Options: <**expN1**>|<**expC1**> is the ordinal position of the order in the list of open indices (similar to SET ORDER), or the order name. If omitted or NIL, the controlling order is assumed.

<**expC2**> is the name of the index file, if <expC1> was specified.

Returns: <**retN**> is the relative position of the current record in the specified order. Zero is returned if the current record is out of scope or positioned on EOF().

Related: oRdd:ORDERINFO(), oRdd:SETORDER(), oRdd:CREATEORDER(), oRdd:CREATEINDEX(), oRdd:ORDERKEYGOTO()

oRdd:ORDERKEYVAL —> exp *Access*

Returns the value of the current index (order) key. This value and its type is equivalent to macro-evaluating the &(INDEXKEY()) function or to invoking the oRdd:ORDERINFO(DBOI_KEYVAL) and oRdd:ORDERINFO(DBOI_KEYTYPE) methods.

Related: oRdd:ORDERINFO(), INDEXKEY()

oRdd:ORDERSCOPE (expN1...) —> ret *Method*

Sets the boundaries for scoping key values in the controlling order.

`ret = oRdd:ORDERSCOPE (expN1, [exp2])`

Arguments: <**expN1**> is a numeric value or constant (preferred, see "rddsys.fh") specifying the type of information.

Constant	Value	Specifies
TOPSCOPE	0	top of the index boundary
BOTTOMSCOPE	1	bottom of the index boundary

Options: <**exp2**> is the new top or bottom value to be set. The comparison of the current key against the top/bottom boundary is performed by the usual <= and >= relational operators, similar to ok := IF (oRdd:ORDERKAYVAL >= <top-boundary> .and. oRdd:ORDERKAYVAL <= <bottomboundary>), therefore the comparison rules according to section LNG.2.9 apply. Omitting this argument or specifying it NIL resets the boundary to its original default (the first or last logical record) position.

Returns: <**ret**> is the current set boundary value (before resetting), equivalent to oRdd:ORDERINFO(DBOI_SCOPE*). If <expN1> is invalid or not set, NIL is returned.

Example:

```
oAdr := DBSERVER {"address"}
oAdr:SETINDEX("adrname")
oAdr:ORDERINFO(DBOI_EXPRESSION)         // "upper(NAME)"
oAdr:GOTOP()    ; ? oAdr:Name           // "Anders"
oAdr:GOBOTTOM() ; ? oAdr:Name, Name     // "address", "Zulu"
oAdr:ORDERSCOPE(TOPSCOPE,    padr("MILLER",20))
oAdr:ORDERSCOPE(BOTTOMSCOPE, "S")
SET EXACT OFF                           // soft comparison
oAdr:GOTOP()    ; ? oAdr:Name           // "Miller"
oAdr:GOBOTTOM() ; ? oAdr:Name           // "Smith"
```

Related: oRdd:GOTOP(), oRdd:GOBOTTOM(), oRdd:SKIP(), oRdd:ORDERKEYGOTO(), oRdd:ORDERINFO(), oRdd:ORDERKEYCOUNT(), oRdd:BOTTOMSCOPE, oRdd:TOPSCOPE, oRdd:SEEK(), oRdd:LOCATE(), SKIP, GO TOP, GO BOTTOM, FIND, SEEK, LOCATE

oRdd:ORDERSKIPUNIQUE ([expN1]) –> retL *Method*

Moves the record pointer to the next or previous keys which differs from the current one, regardless of the UNIQUE index flag. Note that for many equivalent keys, a UNIQUE indexed order may result in faster movement than this filtering.

`retL = oRdd:ORDERSKIPUNIQUE ([expN1])`

Options: <**expN1**> is the skip direction, similar to the oRdd:SKIP(). Positive values skip forward in the end-of-file direction, negative backward. If not specified, 1 is assumed.

Returns: <**retL**> reports the success. It returns FALSE if the EOF() or

BOF() is reached. Example:

```
oAdr := SEEK ("MILLER  ")
while !eof() .and. "MILLER" == upper(trim(oAdr:Name))
   oAdr:SKIP()
enddo
? oAdr:Name                          // Millerman
  * is equivalent to:
oAdr := SEEK ("MILLER  ")
oAdr := ORDERSKIPUNIQUE()
? oAdr:Name                          // Millerman
```

Related: oRdd:SKIP(), oRdd:EOF, oRdd:BOF, oRdd:ORDESCOPE(), oRdd:ORDERINFO(), oRdd:ORDERISUNIQUE()

oRdd:ORDERTOPSCOPE <–> exp *Access/Assign*

A value, controlling the first visible index key of the currently selected order and considered for all database movement operations. Together with oRdd:ORDER-BOTTOMSCOPE, it allows to "filter" the index for a specified range of index keys. Assigning NIL (the default value) to the instance will reset the top boundary to the first available key. See example in oRdd:ORDERSCOPE().

Related: oRdd:ORDERSCOPE(), oRdd:ORDERBOTTOMSCOPE, SKIP, GO TOP

oRdd:PACK ( ) –> retL *Method*

Removes all records marked for deletion from the DBserver database. Equivalent to the PACK command.

`retL = oRdd:PACK ()`

Returns: <**retL**> signals the success.

Multiuser: the database must be opened (or the class instantiated) in EXCLUSIVE mode.

Related: PACK, DELETE, oRdd:DELETE()

oRdd:QUICKFIELDGET (...) –> ret *Method*

Retrieves a field value, by knowing the internal field address. This method is used internally in the RDD and compiler (instead of FieldGet for accessing a field by name), if specified. It is NOT designed for use by the user's .prg program, use oRdd:FIELDGET() instead. See the rddcb4a.c and rddcb4b.c files for additional descriptions, if required. This method is NOT a part of the default DataServer class.

oRdd:QUICKFIELDPUT (...) –> ret *Method*

Stores a value into a field, by knowing the internal field address. This method is used internally in the RDD and compiler (instead of FieldPut for assigning a field by name), if specified. It is NOT designed for use by the user's .prg program, use oRdd:FIELDPUT() instead. See the rddcb4a.c and rddcb4b.c files for additional descriptions, if required. This method is NOT a part of the default DataServer class.

oRdd:RDDINFO (expN1...) –> ret *Method*

Returns and optionally changes information about the currently used RDD.

`ret = oRdd:RDDINFO (expN1, [exp2])`

Arguments: <**expN1**> is a numeric value or constant (preferred, see "set.fh") specifying the type of information.

Constant	Value	Ret		Description	Change
_SET_AUTOOPEN	104	retL		automatically opens the production indices together with the .dbf	*yes
_SET_AUTOORDER	105	retN	1 0	specifies, that the production index automatically sets the controlling order. indicates the requirement of SETORDER() invocation	*yes
_SET_AUTOSHARE	108	retN	0 1 2	disables the automatic sharing control, it overrides the SET AUTOLOCK command. will use the files in the specified mode. will open all files in exclusive mode, overriding the SET EXCLUSIV command and the SHARE clause.	E *yes
_SET_DEFAULTRDD	102	retC		returns the name of the default RDD driver, equivalent to RDDSETDEFA() function	no

_SET_HPLOCKING	106	retL	not used in FlagShip	*yes
_SET_MEMOBLCKSIZE	101	retN	block size (in bytes) of the memo file (.dbt)	*yes
_SET_MEMOEXT	103	retC	extension of the memo file, equiv. to oRdd:ORDERINFO (DBOI_INDEXEXT)	yes
_SET_NEWINDEXLOCK	107	retL	not used in FlagShip	*yes
_SET_OPTIMIZE	111	retL	additional index optimization available?	*yes
_SET_STRICTREAD	109	retL	not used in FlagShip	*yes

Options: <**exp2**> is the new value to be set.

Returns: <**ret**> is the required information or the current value before resetting. If <expN1> is invalid, NIL is returned.

Compatibility: Setting the items marked with "*" is ignored by the default RDD driver DBFIDX.

Related:oRdd:INFO(), oRdd:RECINFO(), SET() and INDEX*() functions

oRdd:RDDNAME —> expC *Access*

Returns a string representing the name of the RDD. The server can be specified as a parameter during instantiation, or by invoking the RDDSETDEFAULT() or DBSETDRIVER() function. If not set, "DBFIDX" is the default driver.

Related: RDDSETDEFAULT(), DBSETDRIVER()

oRdd:READONLY —> expL *Access*

Returns a logical value indicating whether the file was opened as a read-only file during the instantiation.

Related: DBDERVERNEW(), USE, DBUSEAREA()

oRdd:RECALL ([expC1...]) —> retL *Method*

Reinstates the current, DELETEd record or a range of records according to the given scope. Equivalent to and invoked from the RECALL command or the DBRECALL() function.

```
retN = oRdd:RECALL ([expC1|expB1], [expC2|expB2],
                    [expN3|expL3])
```

Options: <**expC1**>|<**expB1**> is equivalent to the FOR clause of RECALL. The condition, given as a string or code block, is evaluated for each record of the source scope.

<**expC2>**|<**expB2**> is equivalent to the WHILE scope. The condition, given as a string or code block, is evaluated for each record from the current position until <expB2> returns FALSE.

<**expN3**>|<**expL3**> is the range of records, providing the same functionality as the ALL, REST and NEXT clause of commands. See chapter 6.2 for the scope values.

Scope: If none of the arguments 1 to 3 are specified, the global scope of the current server is used, see chapter 6.1. If not set, the current record is RECALLed.

Returns: <**retL**> signals success, if TRUE, or failure (e.g. failed lock) otherwise. Non-deleted records are ignored.

Multiuser: if the database is opened (or the class instantiated) in SHAREd mode, at least RLOCK() for deleting a single record, or FLOCK() for multiple record processing is required, if oRdd:CONCURRENCY is set to 0.

Related: DELETE, DBDELETE(), DELETED(), oRdd:DELETEALL(), oRdd:DELETED, oRdd:CONCURRENCYCONTROL

oRdd:RECALLALL ( ) —> retL *Method*

Reinstates all DELETEd records of the database table. Equivalent to the DELETE ALL command or the `oRdd:GOTOP() ; oRdd:DELETE( {|| .T.} )` sequence.

Scope: The global scope according to chapter 6.1 does not apply. Returns: <**retL**> signals success, if TRUE, or failure (e.g. failed lock) otherwise.

Multiuser: if the database is opened (or the class instantiated) in SHAREd mode, FLOCK() is required, if oRdd:CONCURRENCY is set to 0.

Related: RECALL, DBRECALL(), DELETED(), oRdd:RECALL(), oRdd:DELETE(), oRdd:DELETED, oRdd:CONCURRENCYCONTROL

oRdd:RECCOUNT —> expN *Access*

Returns the number of physical records in the database. Equivalent to and invoked from the RECCOUNT() or LASTREC() function.

Related: RECCOUNT(), LASTREC(), oRdd:RECORDINFO()

oRdd:RECNO <—> expN *Access/Assign*

A numeric value, representing the current record number. Equivalent to and invoked from the RECNO() function. Assigning a value to oRdd:RECNO is equivalent to executing the GOTO() method or DBGOTO() function. See additional details in the RECNO() function.

Related: oRdd:RECCOUNT, oRdd:GOTO(), oRdd:RECORDINFO(), RECNO(), GO TO, DBGOTO()

oRdd:RECORDINFO (expN1...) —> ret *Method*

Returns information about the current or specified record.

`ret = oRdd:RECORDINFO (expN1, [expN2])`

Arguments: <**expN1**> is a numeric value or constant (preferred, see "rddsys.fh") specifying the type of information.

Constant	Value	Ret	Description
DBRI_DELETED	1	retL	is record deleted?
DBRI_LOCKED	2	retL	is record locked?
DBRI_RECSIZE	3	retN	record size in bytes
DBRI_RECNO	4	retN	current record number

Options: <**expN2**> is the record number for which information is to be retrieved. If omitted or set to zero (0) or NIL, the current record is assumed.

Returns: <**ret**> is the required information or the current or specified record. If <expN2> is out of range, NIL is returned.

Related:oRdd:DELETED, oRdd:RECSIZE, oRdd:INFO(), SET() and INDEX*() functions

oRdd:RECSIZE —> expN *Access*

Returns the record size in bytes. Equivalent to and invoked from the RECSIZE() function.

Related: RECSIZE()

oRdd:REFRESH ( ) —> retL *Method*

Rereads the current record from the database, discarding (undo) any changes that have been made. However, it cannot roll back changes that have been committed with the oRdd:COMMIT(), oRdd:SKIP() or with the associated commands and functions.

`retL = oRdd:REFRESH ()`

Returns: <**retL**> signals success.

Related: oRdd:COMMIT(), oRdd:SKIP(), COMMIT, SKIP, DBCOMMIT()

oRdd:REINDEX ( ) —> retL *Method*

Rebuilds all open indices for this DBserver. Equivalent to and invoked from the REINDEX command or DBREINDEX() function.

```
retL = oRdd:REINDEX ()
```

Returns: <**retL**> signals success.

Multiuser: the database must be opened (or the class instantiated) in the EXCLUSIVE mode.

Related: oRdd:CREATEINDEX(), REINDEX, DBREINDEX()

oRdd:RELATION ([expN1]) —> retC *Method*

Retrieves the relation string, if any, set with SET RELATION command or the oRdd:SETRELATION() method. Equivalent to and invoked from the DBRELATION() function.

```
retC = oRdd:RELATION ([expN1])
```

Options: <**expN1**> is the ordinal number of the relation in the list of current relations starting at one. Zero and NIL is equivalent to one, the first relation in the list.

Returns: <**retC**> is the string of the relation expression, if specified, or null string "" otherwise.

Related: oRdd:SETRELATION(), oRdd:ISRELATION, DBRELATION(), SET RELATION

oRdd:RELATIONOBJECT ([expN1]) —> retO *Method*

Retrieves the DBserver object of the specified relation.

```
retC = oRdd:RELATIONOBJECT ([expN1])
```

Options: <**expN1**> is the ordinal number of the relation in the list of current relations starting at one. Zero and NIL is equivalent to one, the first relation in the list.

Returns: <**retO**> is the object of the related DBserver, or NIL if no relation is set. The object is returned also for temporarily disabled relations with the oRdd:RELATION assign method.

Compatibility: not available in CA/VO.

Related: oRdd:SETRELATION(), SET RELATION, oRdd:RELATION()

oRdd:REPLACE (exp1...) —> retL *Method*

Replaces one or several fields with a new value. Equivalent to and invoked from the REPLACE command.

```
retL = oRdd:REPLACE (exp1|expB1|expA1,
          expC2|expN2|expA2, [expC3|expB3],
          [expC4|expB4], [expN5|expL5])
```

Arguments: <**exp1**>|<**expB1**>|<**expA1**> is the expression or value, equivalent to the WITH clause of the REPLACE command. When a code block is specified, the result of evaluating it replaces the field. If the argument is an array, its elements specify the expressions or code blocks to yield the replacement value. The type of the <exp1> expression has to correspond to the field type <exp2>, otherwise RTE occurs.

<**expC2>**|<**expN2**>|<**expA2**> is the name or ordinal position of the field to be replaced with the <exp1> value. If the argument is an array, its elements specify the field names or their ordinal positions. When both of the first two arguments are arrays, the shorter array dimension is evaluated for any record of the given scope. If the first argument is a single expression or code block and the second argument an array, the <exp1> value replaces all fields of <expA2>.

Options: <**expC3**>|<**expB3**> is equivalent to the FOR clause of REPLACE. The condition, given as a string or code block, is evaluated for each record of the source scope.

<**expC4>**|<**expB4**> is equivalent to the WHILE scope. The condition, given as a string or code block, is evaluated for each record from the current position until <expB4> returns FALSE.

<**expN5**>|<**expL5**> is the range of records, providing the same functionality as the ALL, REST and NEXT clause of commands. See chapter 6.2 for the scope values.

Scope: If none of the arguments 3 to 5 are specified, the global scope of the current server is used, see chapter 6.1. If not set, the current record is REPLACEd.

Returns: <**retL**> signals the success, if TRUE, or failure (e.g. failed lock) otherwise.

Multiuser: if the database is opened (or the class instantiated) in SHAREd mode, at least RLOCK() for deleting a single record, or FLOCK() for multiple record processing is required, if oRdd:CONCURRENCY is set to 0.

Related: REPLACE, FIELDPUT(), oRdd:FIELDPUT(), <FieldName> := <exp>

oRdd:RLOCK ([expN1]) —> retL *Method*

Locks the specified record to perform a write access or to protect this record against write access from another user or process. Meaningful in SHAREd mode only. Equivalent to and invoked from the RLOCK() function, or identical to invoking the

oRdd:RLOCK(oRdd:RECNO) method. The superset of this method is oRdd:RLOCKVERIFY().

Options: <**expN1**> specifies the record number to be locked. The current record pointer RECNO remains unchanged. When omitted, NIL or zero, the current record is locked, and all previous locks are released.

Returns: <**retL**> signals success. On error, FALSE is returned, which reports that the database is FLOCKed or the record RLOCKed by another user, application or DBserver, or that <expN1> is out of range.

Multiuser: if the database is opened in SHAREd mode, and the automatic locking is disabled (oRdd:CONCURRENCY is set to 0), RLOCK() is required prior to write accessing a single record. If oRdd:CONCURRENCY is active, RLOCK() overrides the automatic locking and the lock remains active until oRdd:UNLOCK() is executed. If the database is opened in EXCLUSIVE mode (the default), no locking is required and RLOCK() is ignored.

Related: RLOCK(), oRdd:LOCKCURRENTRECORD(), oRdd:RLOCKLIST, oRdd:RLOCKVERIFY(), oRdd:UNLOCK(), oRdd:CONCURRENCY

oRdd:RLOCKLIST —> expA *Access*

Returns an array of record numbers that are currently RLOCKed. If no records are RLOCKed, an empty array is returned.

Related: RLOCK(), oRdd:RLOCK()

oRdd:RLOCKVERIFY ( ) —> retL *Method*

Determines if the current record is still unmodified since last accessed by the current application and whether a record update is safe. If so, the record is RLOCKed. This is a superset of the RLOCK() method.

```
retL = oRdd:RLOCKVERIFY ()
```

Returns: <**retL**> TRUE signals that the record contents is equal to that of the last field access/assign/positioning of the current application, and the record is successfully RLOCKed for subsequent field REPLACEment. FALSE signals, that the record was changed by another user or process in the meantime, the subsequent RLOCK and replacement may not be safe and therefore no RLOCK is issued. The application should then determine the changed field to ensure a correct transaction, see example in the RLOCKVERIFY() function. Since the fields are not automatically updated after oRdd:RLOCKVERIFY(), you may store the "old" data, issue oRdd:SKIP(0) and retrieve the new values. A FALSE value may also signal, that oRdd:RLOCK() failed.

Multiuser: when the database is opened (or the class instantiated) in EXCLUSIVE mode, the method always returns TRUE.

Related: oRdd:RLOCK(), oRdd:LOCKCURRENTRECORD(), RLOCKVERIFY()

oRdd:SCOPE <—> expL|expN *Access/Assign*

Determines or sets the general server scope according to chapter 6.2. The scope provides the same functionality as the ALL, REST and NEXT clause of commands and affects several processing methods if these are called with no explicit scope. The constants are available in "rddsys. fh" file.

Scope content	Value	Description
DBSCOPEALL	.F.	The scope is ALL records, or REST with WHILE.
DBSCOPEREST	.T.	The scope is the remaining records starting from the current position.
any number	> 0	The scope is NEXT nRecords.
specifying	NIL	The scope is ALL records, or REST with WHILE.

oRdd:SEEK (exp1...) —> retL *Method*

Seeks through the current index/order for the first or last key matching the giving expression, starting at the first logical record. Equivalent to and invoked from the SEEK command or the DBSEEK() function.

`retL = oRdd:SEEK (exp1, [expL2], [expL3])`

Arguments: <**exp1**> is the expression to be matched with the index key, equivalent to the <exp> of the SEEK command.

Options: <**expL2**> is equivalent to the SOFTSEEK clause of the SEEK command. If set to TRUE (or FALSE), a soft seek is (not) performed, regardless to the SET SOFTSEEK state. If NIL or not specified, the current state of SET SOFTSEEK is considered.

<**expL3>** specifies, if the first or last occurrence of the key value is seek'ed for. If TRUE, the database is positioned to the last matching key; or the first otherwise. The TRUE option is supported by some RDDs only.

Returns: <**retL**> signals if the record was found, and is equivalent to the oRdd:FOUND instance.

Example:

```
SET EXACT OFF
? oAdr:SEEK ("MILLER")            // find first Miller...
? oAdr:SEEK ("MILLER", , .T.)     // find last  Miller...
```

Related: SEEK, DBSEEK(), oRdd:SEEKEVAL(), oRdd:FOUND, oRdd:EOF

oRdd:SEEKEVAL (expB1...) —> retL *Method*

Seeks through the current index/order for the first or next key matching the evaluated code block expression, starting with the current record. Equivalent to and invoked from the SEEK EVAL command.

`retL = oRdd:SEEKEVAL (expB1, [expL2])`

Arguments: <**expB1**> is the code block, which performs the comparison of the index. The current index/order key value and the corresponding record number are passed to the code block as parameters, the code block should return TRUE if the match succeeds, or FALSE to continue the index search. The code block body may not move the database pointer itself by means of GOTO or SKIP.

Options: <**expL2**> specifies for some RDD drivers, that the corresponding record number should be read during the index search. If not given, specified TRUE or NIL (the default), the DBserver will move the database pointer according to the index key for any index skip. If specified FALSE in some RDD drivers, the database pointer remains unchanged until the code block returns TRUE or the end of the database was reached. This results in significant speed-up; but the record status and its content are not available for the code block, also SET DELETED and SET FILTER are ignored in this case. If <expL2> is not specified (or TRUE) in the default DBSERVER and DBFIDX driver, the database record is repositioned during the seek process only on request, i.e. if a field access (or field status access) is specified within the code block body.

Returns: <**retL**> signals if the record was found, and is equivalent to the oRdd:FOUND instance.

Example:

```
? oAdr:INDEXKEY()                  // "upper(NAME + FIRST)"
bSeek := {|key, rec| ;
          "MILLER" $ key .and. "PETER" $ key}
oAdr:GOTOP()
while oAdr:SEEKEVAL (bSeek, .F.)   // process index key only
  ? oAdr:NAME, oAdr:FIRST, CITY    // for all ..Peter..Miller..
  oAdr:SKIP()
enddo
bSeek := {|key, rec| "MILLER" $ upper(key) .and. ;
          !deleted() .and. "MUNICH" $ upper(oAdr:CITY)}
oAdr:GOTOP()
? oAdr:SEEKEVAL (bSeek)            // .dbf access enabled
```

Compatibility: not available in CA/VO.

Related: SEEK EVAL, DBSEEK(), oRdd:SEEK(), oRdd:FOUND, oRdd:EOF

oRdd:SETFILTER (expB1...) —> retL *Method*

Sets a global DBserver filter condition (see also chapter 6.1), whereby the specified or self-created code block is significant. Equivalent to and invoked from the SET FILTER command or the DBSETFILTER() function.

```
retL = oRdd:SETFILTER ([expB1], [expC2])
retL = oRdd:SETFILTER (expC2)
```

Arguments: <**expB1**> is the code block that is evaluated for any database movement. If <expC2> is omitted, <expB1> must be specified, but oRdd:FILTER and DBFILTER() returns a null string.

<**expC2>** is a string, which is internally macro-compiled into the <expB1> code block, if <expB1> is not specified. Otherwise, it contains only the information for oRdd:FILTER and DBFILTER() reports.

Returns: <**retL**> signals success.

Related: SET FILTER, DBSETFILTER(), oRdd:CLEARFILTER()

oRdd:SETINDEX (expC1...) —> retL *Method*

Opens an index file and select its order as the controlling order, if the order/ index list is still empty. Equivalent to and invoked from the SET INDEX command or the DBSETINDEX() function.

```
retL = oRdd:SETINDEX ([expC1], [expC2], [expL3])
```

Arguments: <**expC1**> is the name of the index file, optionally prefaced with the directory. If no extension is specified, the default oRdd:INDEXEXT is used. If the path is not specified, the current, SET PATH and SET DEFAULT directories are searched. If <expC1> is NIL, null string, or not specified, the oRdd:CLEARINDEX() method is invoked.

Options:<**expC2>** specifies the order tag name within the index file in multiple tag indices, ignored otherwise.

<**expL3>** specifies whether the index should be opened EXCLUSIVE to that application. If not given, the default is FALSE, which opens the index in shared mode.

Returns: <**retL**> signals success.

Compatibility: only one argument is available in CA/VO.

Related: SET INDEX, DBSETINDEX(), oRdd:CLEARINDEX(), oRdd:SETORDER()

oRdd:SETORDER (expN1...) —> retC *Method*

Selects an index/order from the list of open indices and makes it to the controlling order. Moves the record pointer to the first logical record. Equivalent to and invoked from the SET ORDER command or the DBSETORDER() function.

```
retC = oRdd:SETORDER ([expN1|expC1], [expC2])
```

Arguments: <**expN1**>|<**expC1**> is the ordinal number (1..15) specifying the position in the list of open indices/ orders. Zero (0), NIL or null string "" disables the controlling index order, but all indices remain open. <expC1> will search for the order name in the list of open orders.

Options: <**expC2>** specifies the index file name, when an order name <expC1> is given, which is not unique within the order list. Returns: <**retC**> is the last order name or "".

Related: SET ORDER, DBSETORDER(), oRdd:SETINDEX()

oRdd:SETORDERCONDITION (expN1...) —> retL *Method*

Sets conditions that are applied during the index and the order creation. If oRdd:SETORDERCONDITION() has not been called, orders are not conditional. This method is called from the INDEX ON FOR.. command and the ORDCONDSET() function .

```
retL = oRdd:SETORDERCONDITION ([expC1], [expB2],
          [expL3], [expB4], [expB5], [expN6],
          [expN7], [expN8], [expN9], [expL10],
          [expL11])
```

Options: <**expC1**> is a string equivalent to the FOR clause and stored in the index header. Only this argument affects the later index update and reindex. If conditional access is not required later, null string "" is equivalent to a NIL value. If no arguments are specified at all, the condition is reset.

<**expB2>** is a code block that defines a FOR condition that each record within the scope must meet in order to include this key into the index/ order file during the order creation.

<**expL3>** specifies if all orders in the current or specified working area (if TRUE or NIL) are affected, which is the default on INDEX ON.

<**expB4>** is a code block that defines a WHILE condition. The indexing is performed as long as the code block returns TRUE and is aborted when the condition return FALSE.

<**expB5>** is a code block that defines an EVAL clause. This code block is evaluated for every record that is processed and often used for displaying of the indexing progress.

<**expN6>** is a numeric expression, which modifies the number of times <expB**5>** is evaluated. It offers a performance enhancement by evaluating the condition for every nth record instead of for every record ordered. Zero (0) is equivalent to NIL, the default.

<**expN7>** is a numeric expression specifying the starting record number, Zero (0) is equivalent to NIL, which specifies to start from the first record, if such is available.

<**expN8>** is a numeric expression, equivalent to the NEXT<n> clause, which specifies the number of records to be processed.

<**expN9>** is a numeric expression, equivalent to the RECORD <n> clause, which specifies the record to be processed.

<**expL10>** is a logical expression, FALSE is equivalent to the ALL clause (default), whilst TRUE to the REST clause.

<**expL11>** is a logical expression, FALSE is equivalent to the ASCENDING sort order (default), whilst TRUE to the DESCENDING clause.

Returns: <**retL**> signals success.

Related: ORDCONDSET()

oRdd:SETRELATION (expO1...) —> retL ***Method***

Sets a relation from this DBserver to a child server. Equivalent to and invoked from the SET RELATION command or the DBSETRELATION() function.

```
retL = oRdd:SETRELATION (expO1, expB2|expC2|expA2,
       [expC3])
retL = oRdd:SETRELATION ( )
```

Arguments: <**expO1**> is the object specifying the child database. If omitted, the oRdd:CLEARRELATION() method is invoked in order to clear all relations for the current server.

<**expB2**>|<**expC2**>|<**expA2**> is a code block or string specifying the relation, equivalent to the TO clause of SET RELATION. The <expC2> string (e.g. the field name, corresponding to the child's index) is automatically macro compiled into a code block. The elements of an array of strings <expA2> are concatenated with a plus sign and the result is macro-compiled into a code block. As always, the expression should match or be a partial index of the controlling index of the child's work area.

Options: <**expC3>** specifies the relation string to be reported by ordd:RELATION() or DBRELATION(), when <expB2> is used.

Returns: <**retL**> signals success.

Example:

```
? oAdr:INDEXKEY()                    // CustID
? oOrd:INDEXKEY()                    // AdrNum
oAdr:SETRELATION(oOrd, {|| oAdr:CustID}, "CustID")
oAdr:SEEK(12345)
if oAdr:FOUND .and. oOrd:FOUND
   ? oAdr:CustID, oOrd:Name
endif
oOrd:SETORDER(2)
oAdr:SETRELATION(oOrd, {"Name", "First", "City"})
```

Related: oRdd:ISRELATION, oRdd:CLERRELATION(), SET RELATION, DBSETRELATION()

oRdd:SHARED –> expL *Access*

Returns a logical value indicating the open mode. If the database is opened or the object instantiated in SHARED mode, TRUE is returned. FALSE indicates an exclusive open.

Related: DBSERVERNEW(), USE, DBUSEAREA(), oRdd:INFO(), ISDBEXCL()

oRdd:SKIP ([expN1]) –> retN *Method*

Moves the record pointer forward or backward a specified number of records. Equivalent to and invoked from the SKIP command or DBSKIP() function.

`retN = oRdd:SKIP ([expN1])`

Options: <**expN1**> is the number of records to move, relative to the current record. A positive value means to move forward, and a negative value means to move backward. If omitted, 1 is assumed.

Returns: <**retN**> is the number of records actually skipped.

Related: SKIP, DBSKIP(), oRdd:GOTO()

oRdd:SORT (expC1...) –> retL *Method*

Copies records from the current database file (source) in sorted order to another database file (target). Equivalent to the SORT command.

`retL = oRdd:SORT (expC1|expO1, [expA2],`
`[expC3|expB3], [expC4|expB4],`
`[expN5|expL5])`

Arguments: <**expC1**>|<**expO1**> is the name or DBserver object of the target database. If no extension is specified with <expC1>, it is assumed to be .dbf, or the standard extension according to the RDD driver. If <expC1> is specified, the

target database is opened exclusively. If <expO1> is given, the RDD server object is used and the records are appended.

Options: <**expA2**> is an array of character values, specifying the field names used as a sorting order. Any field may include the sorting order (/A, /D, /C, see SORT command). If <expA2> is not specified, the records are transferred in the physical or logical order of the source server.

<**expC3**>|<**expB3**> is equivalent to the FOR scope. The condition, given as a string or code block, is evaluated for each record of the scope.

<**expC4**>|<**expB4**> is equivalent to the WHILE scope. The condition, given as a string or code block, is evaluated for each record from the current position in the source database until <expB4> returns FALSE.

<**expN5**>|<**expL5**> is the range of records, providing the same functionality as the ALL, REST and NEXT clause of commands. See chapter 6.2 for the scope values.

Scope: If none of the arguments 3 to 5 are specified, the global scope of the current server is used, see chapter 6.1. If not set, the default is ALL records.

Returns: <**retL**> signals success, if TRUE, or failure (e.g. the open mode) otherwise.

Example:

```
ok := oAdr:SORT ("newadr", {"Name /C","First","left(ZIP,3) /D"},;
                 {|| "MILLER" $ upper(Name)})
// which is equivalent to

oAdr:SETORDERCONDITION ('"MILLER" $ upper(Name)')
oAdr:CREATEINDEX ("tmp", "upper(Name)+First+descend(left(zip,3)))
oAdr:COPYDB ("newadr")
```

Related: SORT, INDEX, COPY TO

oRdd:SUM (expC1...) —> retA *Method*

Calculates the sum of a series of numeric expressions. Similar to the SUM command.

```
retA = oRdd:SUM (expC1|expL1|expB1|expA1,
          [expC2|expB2], [expC3|expB3],
          [expN4|expL4])
```

Arguments: <**expC1**>|<**expL1**>|<**expB1**>|<**expA1**> is a single expression (e.g. field name), a code block or an array of expressions or code blocks to be summarized. The expression must evaluate to numeric or logical to be summarized, whereby TRUE adds one to the result.

Options: <**expC2**>|<**expB2**> is equivalent to the FOR scope. The condition, given as a string or code block, is evaluated for each record of the scope.

<**expC3**>|<**expB3**> is equivalent to the WHILE scope. The condition, given as a string or code block, is evaluated for each record from the current position in the source database until <expB3> returns FALSE.

<**expN4**>|<**expL4**> is the range of records, providing the same functionality as the ALL, REST and NEXT clause of commands. See chapter 6.2 for the scope values.

Scope: If none of the arguments 2 to 4 are specified, the global scope of the current server is used, see chapter 6.1. If not set, the default is ALL records.

Returns: <**retA**> is an array that contains the sums for each expression or field specified. If a single expression is specified, the array dimension is one, otherwise the dimension of <expA1> is returned.

Example:

```
aResult := oAdr:SUM ({"Turnover", {||dept >= 4711}, "Commis"} ,;
                     {|| "MILLER" $ upper(Name)} ,, DBSCOPEREST)
```

Related: SUM, AVERAGE, TOTAL

oRdd:TOTAL (expC1...) —> retL *Method*

Summarizes records by key value, producing grouped summarizations, and writes the aggregate values to another (target) database. Similar to the TOTAL command.

```
retL = oRdd:TOTAL (expC1|expO1, expC2|expB2,
          [expA3], [expC4| expB4], [expC5|expB5],
          [expN6|expL6])
```

Arguments: <**expC1**> is the name of the target database. If no extension is specified, it is assumed to be .dbf, or the standard extension according to the RDD driver. The target database is opened exclusively.

<**expC2**>|<**expB2**> is the key field that is the basis for the summarization groups, that produce a new record in the target database. Equivalent to the ON clause of the TOTAL command. The database should be indexed or sorted on that key.

Options: <**expA3**> is an array of names of numeric fields to total, equivalent to the FIELDS clause of TOTAL. If the argument is not specified, the target record contains the value of the first record matching the second argument.

<**expC4**>|<**expB4**> is equivalent to the FOR scope. The condition, given as a string or code block, is evaluated for each record of the scope.

<**expC5**>|<**expB5**> is equivalent to the WHILE scope. The condition, given as a string or code block, is evaluated for each record from the current position in the source database until <expB5> returns FALSE.

<**expN6**>|<**expL6**> is the range of records, providing the same functionality as the ALL, REST and NEXT clause of commands. See chapter 6.2 for the scope values.

Scope: If none of the arguments 4 to 6 are specified, the global scope of the current server is used, see chapter 6.1. If not set, the default is ALL records.

Returns: <**retL**> signals success.

Example:

```
? oAdr:INDEXKEY()                    // str(Departm,5) + city
ok := oAdr:TOTAL ("depsum", "Departm", {"Turnover", "Commis"})
oDep := DBSERVER {"depsum"}
oDep:EVAL ({|| qout( recno(), Turnover, Commis)} )
```

Related: TOTAL, SUM, AVERAGE

oRdd:UNLOCK ([expN1]) —> retL *Method*

Releases a specified lock or all locks. Equivalent to and invoked from the UNLOCK command.

`retL = oRdd:UNLOCK ([expN1])`

Options: <**expN1**> is the number of the desired record. If 0 or omitted, all locks for this server are released, record locks as well as file locks.

Returns: <**retL**> signals success.

Multiuser: if the database is opened exclusive, all locks and un-locks are ignored.

Related: UNLOCK, oRdd:RLOCK(), oRdd:FLOCK()

oRdd:UPDATE (expC1...) —> retL *Method*

Updates this server (target) with data from another database server (source). Equivalent to the UPDATE command.

`retL = oRdd:UPDATE (expC1|expO1, expC2|expB2,`
`[expL3], expB4)`

Arguments: <**expC1**>|<**expO1**> is the name or DBserver object of the source database, equivalent to the FROM clause of UPDATE. If no extension is specified, it is assumed to be .dbf, or the standard extension according to the RDD driver. If <expC1> is specified, the source database is used shared, in read-only mode. If <expO1> is given, the RDD server object is used.

<**expC2**>|<**expB2**> is the key field that defines how records are matched between the servers. Equivalent to the ON clause of UPDATE.

<**expB4**> is a code block that manages the update and replace operations.

Options: <**expL3**> is equivalent to the RANDOM clause. If specified TRUE, it indicates that the records in the other database are allowed to be unsorted. Otherwise, the source database must be sorted or indexed on the <exp2> key.

Returns: <**retL**> signals success, if TRUE, or failure otherwise.

Multiuser: if the (target) database is opened (or the class instantiated) in SHAREd mode, at least FLOCK() is required, if oRdd:CONCURRENCY is set to 0.

Example:

```
oFrm:INDEXKEY()                 // Departm
oAdr:FLOCK()
oAdr:UPDATE (oFrm, {|| Departm}, , ;
    {|| oAdr:TurnOver += oFrm:cost, oAdr:Count++ })
```

Related: UPDATE

oRdd:USED —> expL *Access*

Returns a logical value indicating whether the server is open. Even if the USE command or instantiating the object fail to open the database, a DBserver object is created. Also, if the database is closed in the meantime, the object remains visible during the variable visibility (life-time) scope. Therefore, if the usability status is unknown, check the oRdd:USED status for TRUE before any DBserver manipulation. This instance is equivalent to and invoked from the USED() function.

Related: DBSERVERNEW(), USE, DBUSEAREA(), USED(), DBOBJECT()

oRdd:WHILEBLOCK <—> expB|NIL *Access/Assign*

The WHILE block is a component of the general server scope, described in chapter 1. It affects several bulk processing methods if they are called with no explicit scope. The WHILE block can be specified as a code block or a string. An access to this instance always returns a code block or NIL if not set. To reset the global WHILE block, assign NIL to it or execute oRdd:CLEARSCOPE() for a global reset.

Related: oRdd:FORBLOCK, oRdd:SCOPE, oRdd:CLEARSCOPE()

oRdd:ZAP ( ) —> retL *Method*

Permanently removes all records from the DBserver database (and memo file), leaving the database empty. Equivalent to the ZAP command.

`retL = oRdd:ZAP ()`

Returns: <**retL**> signals success.

Multiuser: the database must be opened (or the class instantiated) in EXCLUSIVE mode.

Index

Notes

multisoft Datentechnik
Schönaustr. 7
D-84036 Landshut

http://www.fship.com
sales@multisoft.de
support@flagship.de