wxHarbour doc

by Teo Fonrouge (teo/at/windtelsoft/com)

Version 0.5.0

June 8, 2010

Contents

Part I.

GUI Reference

1. Commands

Please keep in mind that wxHarbour does not try to make a 'compatible' (at any level) set of Clipper commands

1.1. @ BROWSE ...

Description.

Creates a new wxhBrowse object

Syntax

@ BROWSE [VAR <*oVar*>]
[DATASOURCE <*ds*>]
[CLASS <*cClassName*>]
[LABEL <*cLabel*>]
[PARENT <*oParentWindow*>]
[ID <*nId*>]
[WIDTH <*nWidth*>] [HEIGHT <*nHeight*>]
[MINSIZE <*nMinWidth*>, <*nMinHeight*>]
[STYLE <*nStyle*>]
[NAME <*cName*>]
[ONKEY <*bOnKey*>]
[ONSELECTCELL <*bOnSelectCell*>]
[READONLY]

Arguments

Description

Examples

1.2. @ PUSHVALIDATOR ...

Description.

Creates a new wxhValidator object and assign it to the last control added. Normally the user doesn't need to call this command directly since it is called automatically on almost all control creation commands.

Syntax

@ PUSHVALIDATOR [<*dataVar*>]

[PICTURE <*cPicture*>]

[WARNING <*bWarning*>]

[ACTION <*bAction*>]

Arguments

<*dataVar*> specifies the variable to assign to the control

<*cPicture*> contains the picture value to be used when displaying the control value

<*bWarning*> contains an expression to be evaluated in order to validate the control value

<*bAction*> contains an expression to be executed when the control value is changed

Description

Examples

1.3. BEGIN BOXSIZER

Description.

Begins a box sizer section, it can be either of vertical or horizontal
the END SIZER command indicates the end of the sizer section.

Syntax

BEGIN BOXSIZER *VERTICAL | HORIZONTAL*
[VAR <*var*>]
[[LABEL] <*cLabel*>]
[SIZERINFO clauses, ...]

Arguments

Description

The basic idea behind a wxBoxSizer is that windows will most often be laid out in rather simple basic geometry, typically in a row or a column or several hierarchies of either.

As an example, we will construct a dialog that will contain a text field at the top and two buttons at the bottom. This can be seen as a top-hierarchy column with the text at the top and buttons at the bottom and a low-hierarchy row with an OK button to the left and a Cancel button to the right. In many cases (particularly dialogs under Unix and normal frames) the main window will be resizable by the user and this change of size will have to get propagated to its children. In our case, we want the text area to grow with the dialog, whereas the button shall have a fixed size. In addition, there will be a thin border around all controls to make the dialog look nice and - to make matter worse - the buttons shall be centred as the width of the dialog changes.

It is the unique feature of a box sizer, that it can grow in both directions (height and width) but can distribute its growth in the main direction (horizontal for a row) unevenly among its children. In our example case, the vertical sizer is supposed to propagate all its height changes to only the text area, not to the button area. This is determined by the proportion parameter when adding a window (or another sizer) to a sizer. It is interpreted as a weight factor, i.e. it can be zero, indicating that the window may not be resized at all, or above zero. If several windows have a value above zero, the value is interpreted relative to the sum of all weight factors of the sizer, so when adding two windows with a value of 1, they will both get resized equally much and each half as much as the sizer owning them. Then what do we do when a column sizer changes its width? This behaviour is controlled by flags (the second parameter of the Add() function): Zero or no flag indicates that the window will preserve it is original size, wxGROW flag (same as wxEXPAND) forces the window to grow with the sizer, and wxSHAPED flag tells the window to change it is size

proportionally, preserving original aspect ratio. When wxGROW flag is not used, the item can be aligned within available space. wxALIGN_LEFT, wxALIGN_TOP, wxALIGN_RIGHT, wxALIGN_BOTTOM, wxALIGN_CENTER_HORIZONTAL and wxALIGN_CENTER_VERTICAL do what they say. wxALIGN_CENTRE (same as wxALIGN_CENTER) is defined as (wxALIGN_CENTER_HORIZONTAL | wxALIGN_CENTER_VERTICAL). Default alignment is wxALIGN_LEFT | wxALIGN_TOP.

As mentioned above, any window belonging to a sizer may have border, and it can be specified which of the four sides may have this border, using the wxTOP, wxLEFT, wxRIGHT and wxBOTTOM constants or wxALL for all directions (and you may also use wxNORTH, wxWEST etc instead). These flags can be used in combination with the alignment flags above as the second parameter of the Add() method using the binary or operator |. The sizer of the border also must be made known, and it is the third parameter in the Add() method. This means, that the entire behaviour of a sizer and its children can be controlled by the three parameters of the Add() method.

Examples

The following section of sample code creates a vertical sizer with two buttons vertically aligned:

```
BEGIN BOXSIZER VERTICAL
        @ BUTTON
        @ BUTTON
END SIZER
```

The following section of sample code creates a horizontal sizer with two buttons horizontally aligned:

```
BEGIN BOXSIZER HORIZONTAL
        @ BUTTON
        @ BUTTON
END SIZER
```

Part II.

RADOX Reference

2. Commands

3. Classes

3.1. TField

3.2. TIndex

3.3. TTable

This class that give functionality to hold a table info; fields, indexes

3.3.1. Properties

Active (*ReadOnly*). Returns .T. if the database has been opened.

autoCreate If .T. the table is created if it not exist already, otherwise an error is raised when trying to open an not existent table

autoEdit When .T. it allows to automatically call ::Edit() when any TField is required to write data to the database

autoOpen When .F. it allows to open the database in two steps by calling explicitly ::Open() after instantiating the class.

TableFileName Contains the table file name.

3.3.2. Events

OnCreate() Procedure executed on the object creation.

OnAfterCancel() Procedure executed after the Cancel() operation.

OnAfterChange() Procedure executed after the Post() operation and only if any field was changed (TField:Changed)

OnAfterDelete() Procedure executed after a successful Delete() operation.

OnAfterInsert() Procedure executed after a successful Insert() operation.

OnAfterOpen() Procedure executed after the Open() operation.

OnAfterPost() Procedure executed after a successful Post() operation. Is executed before OnAfterChange().

OnBeforeInsert Function executed just before the adding record operation, it must return a logical value.

OnBeforePost() Function executed just before the Post() operation, it must return a logical value.

OnDataChange() Procedure executed after change is detected on record position.

OnPickList(param) Function executed when a TField requires a call to the table selection mode.

OnStateChange(oldState) Procedure executed right after a change in the State() property.

OnSyncFromMasterSource() Procedure executed when the Table is synched with his MasterSource (if any).

3.3.3. Methods

BuildTable