

#\$+K!

FL: a Frame Layout Framework

by Aleksandras Gluchovas and others

January 2nd 2002

Contents

[Copyright notice](#)

[Introduction](#)

[Alphabetical class reference](#)

[Classes by category](#)

[Topic overviews](#)

^Contents

^Contents

^browse00001

^K Contents

^DisableButton("Up")

Copyright notice

FL is copyright Aleksandras Gluchovas, 2001-2002.

The licence is the wxWindows Licence.

Copyright notice

topic0

rowse00002

Copyright notice

isableButton("Up")

\$#+K! Introduction

What is FL?

Compiling and using FL

FL concepts

Controlling dragging behaviour

^lntroduction

ⁱntroduction

^browse00003

^K Introduction

^DisableButton("Up")

\$#+K! **Alphabetical class reference**

[BagLayout](#)
[wxBarIterator](#)
[cbAntiflickerPlugin](#)
[cbBarDimHandlerBase](#)
[cbBarDragPlugin](#)
[cbBarHintsPlugin](#)
[cbBarInfo](#)
[cbBarShapeData](#)
[cbBarSpy](#)
[cbCloseBox](#)
[cbCollapseBox](#)
[cbCommonPaneProperties](#)
[cbCustomizeBarEvent](#)
[cbCustomizeLayoutEvent](#)
[cbDimInfo](#)
[cbDockBox](#)
[cbDockPane](#)
[cbDrawBarDecorEvent](#)
[cbDrawBarHandlesEvent](#)
[cbDrawHintRectEvent](#)
[cbDrawPaneBkGroundEvent](#)
[cbDrawPaneDecorEvent](#)
[cbDrawRowBkGroundEvent](#)
[cbDrawRowDecorEvent](#)
[cbDrawRowHandlesEvent](#)
[cbDynToolBarDimHandler](#)
[cbFinishDrawInAreaEvent](#)
[cbFloatedBarWindow](#)
[cbGCUpdatesMgr](#)
[cbHintAnimationPlugin](#)
[cbInsertBarEvent](#)
[cbLayoutRowEvent](#)
[cbLayoutRowsEvent](#)
[cbLeftDClickEvent](#)
[cbLeftDownEvent](#)
[cbLeftUpEvent](#)
[cbMiniButton](#)
[cbMotionEvent](#)
[cbPaneDrawPlugin](#)
[cbPluginBase](#)

^Alphabetical class reference

^Classref

^Browse00008

^K Alphabetical class reference

^DisableButton("Up")

cbPluginEvent
cbRemoveBarEvent
cbResizeBarEvent
cbResizeRowEvent
cbRightDownEvent
cbRightUpEvent
cbRowDragPlugin
cbRowInfo
cbRowLayoutPlugin
cbSimpleCustomizationPlugin
cbSimpleUpdatesMgr
cbSizeBarWndEvent
cbStartBarDraggingEvent
cbStartDrawInAreaEvent
cbUpdateMgrData
cbUpdatesManagerBase
wxDynamicToolBar
wxDynToolInfo
wxFrameLayout
wxFrameManager
GarbageCollector
LayoutManagerBase
wxNewBitmapButton
wxToolLayoutItem
wxToolWindow

^{\$#+K!}Classes by category

A classification of FL classes by category.

Plugin classes

Plugins can be added to frame layouts to extend behaviour.

<u>cbAntiflickerPlugin</u>	Double-buffering class
<u>cbBarDragPlugin</u>	Implements drag behaviour.
<u>cbBarHintsPlugin</u>	Implements bar decoration and sizing behaviour.
<u>cbHintAnimationPlugin</u>	Draws animated hints when the user drags a pane.
<u>cbPaneDrawPlugin</u>	Implements most of MFC-style control bar implementation.
<u>cbPluginBase</u>	Abstract base class for all control-bar related plugins.
<u>cbRowDragPlugin</u>	Implements row-dragging functionality.
<u>cbRowLayoutPlugin</u>	Implements row layout functionality.
<u>cbSimpleCustomizationPlugin</u>	Enables customization of a bar.
<u>cbBarSpy</u>	Helper class used for spying for unhandled mouse events on control bars and forwarding them to the frame layout.

Window classes

Windows classes (note that the mini-button implementations are not true windows in that they do not derive from wxWindow).

<u>wxToolWindow</u>	A small frame that paints its own titlebar.
<u>cbFloatedBarWindow</u>	A kind of wxToolWindow implementing floating windows.
<u>cbMiniButton</u>	Base class for wxToolWindow titlebar buttons.
<u>cbCloseBox</u>	Close button for wxToolWindow titlebar.
<u>cbCollapseBox</u>	Collapse button for wxToolWindow titlebar.

^Classes by category

^Classesbycat

^browse00609

^K Classes by category

^DisableButton("Up")

<u>cbDockBox</u>	Dock button for wxToolWindow titlebar.
<u>cbCloseBox</u>	Close button for wxToolWindow titlebar.
<u>wxNewBitmapButton</u>	Alternative bitmap button class.

Layout management classes

These classes relate to the layout management framework.

<u>cbDockPane</u>	Manages containment and control of bars in a parent frame.
<u>BagLayout</u>	BagLayout lays out items in left-to-right order from top to bottom.
<u>cbUpdatesManagerBase</u>	An abstract interface for display update optimization logic.
<u>cbSimpleUpdatesMgr</u>	Implements optimized logic for refreshing areas of frame layout that need to be updated.
<u>cbGCUpdatesMgr</u>	Implements optimized logic for refresh, based on a garbage collection algorithm.
<u>GarbageCollector</u>	A garbage collection algorithm for use in display refresh optimization.
<u>wxFrameLayout</u>	Manages containment and docking of control bars, which can be docked along the top, bottom, right, or left side of the parent frame.

Event classes

Events are used to decouple parts of the layout framework. For event macros and identifiers, please see the topic [Event macros and identifiers](#).

<u>cbCustomizeBarEvent</u>	Class for bar customization events.
<u>cbCustomizeLayoutEvent</u>	Class for layout customization events.
<u>cbDrawBarDecorEvent</u>	Class for bar decoration drawing events.
<u>cbDrawBarHandlesEvent</u>	Class for bar handles drawing events.
<u>cbDrawHintRectEvent</u>	Class for hint-rectangle drawing events.
<u>cbDrawPaneBkGroundEvent</u>	Class for pane background drawing events.
<u>cbDrawPaneDecorEvent</u>	Class for pane decoration drawing events.
<u>cbDrawRowBkGroundEvent</u>	Class for row background drawing events.
<u>cbDrawRowDecorEvent</u>	Class for row decoration drawing events.

<u>cbDrawRowHandlesEvent</u>	Class for row handles drawing events.
<u>cbFinishDrawInAreaEvent</u>	Class for finish drawing in area events.
<u>cbInsertBarEvent</u>	Class for bar insertion events.
<u>cbLayoutRowEvent</u>	Class for single row layout events.
<u>cbLayoutRowsEvent</u>	Class for multiple rows layout events.
<u>cbLeftDClickEvent</u>	Class for mouse left double click events.
<u>cbLeftDownEvent</u>	Class for mouse left down events.
<u>cbLeftUpEvent</u>	Class for mouse left up events.
<u>cbMotionEvent</u>	Class for mouse motion events.
<u>cbPluginEvent</u>	Base class for all control-bar plugin events.
<u>cbRemoveBarEvent</u>	Class for bar removal events.
<u>cbResizeBarEvent</u>	Class for bar resize events.
<u>cbResizeRowEvent</u>	Class for row resize events.
<u>cbRightDownEvent</u>	Class for mouse right down events.
<u>cbRightUpEvent</u>	Class for mouse right up events.
<u>cbSizeBarWndEvent</u>	Class for bar window resize events.
<u>cbStartBarDraggingEvent</u>	Class for start-bar-dragging events.
<u>cbStartDrawInAreaEvent</u>	Class for start drawing in area events.

Topic overviews

This chapter contains a selection of topic overviews, first things first:

[Notes on using the reference](#)

[Event macros and identifiers](#)

[FAQ](#)

^Topic overviews

^overviews

^browse00610

^K Topic overviews

^DisableButton("Up")

What is FL?

This manual describes FL (Frame Layout), a class library for managing sophisticated window layout, with panes that can be moved around the main window and customized. FL handles many decoration and dragging issues, giving applications the kind of docking facilities that Visual C++ and Netscape Navigator possess.

FL was written by Aleksandras Gluchovas, and is heavily used in wxWorkshop which he also wrote the bulk of.

Please note that this guide is in its infancy, and contributions from FL users are very welcome.

The following screenshot (from fl_demo1) shows a frame with a number of bars that can be dragged around. The vertical grippers with two lines allow a bar to be dragged in that row, changing the ordering of the bar if necessary. The dotted grippers (as in Netscape Navigator) allow a whole row to be moved, again changing the position of the row if required. While moving a bar or row, immediate feedback is given as the moving bar displaces other bars.

Other features: the splitter bar shows a dotted thick line as it's dragged. Single-clicking on a row handle minimizes it to a horizontal tab which is given its own narrow row. This allows the user to temporarily hide a row while allowing quick access to it when required.

A close button (x) hides a bar completely. You can get it back again by right-clicking and selecting the appropriate menu item.

A left, right, up or down arrow button expands the pane in that direction.

{bmc screen01.bmp}

What is FL?

What is fl

browser00004

What is FL?

EnableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp', 'introduction')")

^{\$\$\$K!}Compiling and using FL

FL can be found under the 'contrib' hierarchy, in the following directories:

```
contrib/src/fl
contrib/include/wx/fl
contrib/samples/fl
contrib/docs/latex/fl
docs/html/fl
docs/htmlhelp/fl.chm
docs/pdf/fl.pdf
docs/winhelp/fl.hlp
```

To compile FL:

{bmc bullet.bmp} Under Windows using VC++, open the flVC.dsw project and compile.

{bmc bullet.bmp} Under Unix, FL should be configured when you configured wxWindows. Make FL by changing directory to contrib/src/fl and type 'make'.

Note: there is currently a problem with the wxWindows build system that means that only the static version of library can be built at present.

To use FL:

{bmc bullet.bmp} Under Windows using VC++, link with fl[d].lib.

{bmc bullet.bmp} Under Unix, link with libfl[d].a.

^Compiling and using FL

^topic1

^browse00005

^K Compiling and using FL

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp', `introduction`)"")

FL concepts

These are typical steps when adding FL functionality to your application.

- {bmc bullet.bmp} include the appropriate header files;
- {bmc bullet.bmp} create a new wxFrameLayout passing the top-level frame and the window that is interpreted as the main 'client' window;
- {bmc bullet.bmp} set an updates manager for optimizing drag operations;
- {bmc bullet.bmp} add plugins for implementing various features;
- {bmc bullet.bmp} add bars;
- {bmc bullet.bmp} enable floating mode for the layout if required;
- {bmc bullet.bmp} delete the frame layout in the main frame's destructor.

The following is taken from fl_demo1 and shows the main code implementing the user interface as illustrated in What is FL?.

```
// fl headers
#include "wx/fl/controlbar.h"          // core API

// extra plugins
#include "wx/fl/barhintspl.h"         // bevel for bars with "X"s and
grooves
#include "wx/fl/rowdragpl.h"          // NC-look with draggable rows
#include "wx/fl/cbcustom.h"           // customization plugin
#include "wx/fl/hintanimpl.h"

// beauty-care
#include "wx/fl/gcupdatesmgr.h"       // smooth d&d
#include "wx/fl/antiflickpl.h"       // double-buffered repaint of
decorations
#include "wx/fl/dyntbar.h"            // auto-layout toolbar
#include "wx/fl/dyntbarhnd.h"         // control-bar dimension handler
for it

MyFrame::MyFrame(wxFrame *frame)
    : wxFrame( frame, wxID_ANY, "wxWindows 2.0 wxFrameLayout Test
Application", wxDefaultPosition,
              wxSize( 700, 500 ),
              wxCLIP_CHILDREN | wxMINIMIZE_BOX | wxMAXIMIZE_BOX |
              wxRESIZE_BORDER | wxSYSTEM_MENU | wxCAPTION,
              "freimas" )
```

^FL concepts

^topic2

^browse00006

^K FL concepts

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp', `introduction`)")

```

{
    mpClientWnd = CreateTextCtrl( "Client window" );

    mpLayout = new wxFrameLayout( this, mpClientWnd );

    mpLayout->SetUpdatesManager( new cbGCUpdatesMgr() );

    // setup plugins for testing
    mpLayout->PushDefaultPlugins();

    mpLayout->AddPlugin( CLASSINFO( cbBarHintsPlugin ) ); //
fancy "X"es and bevel for bars
    mpLayout->AddPlugin( CLASSINFO( cbHintAnimationPlugin ) );
    mpLayout->AddPlugin( CLASSINFO( cbRowDragPlugin ) );
    mpLayout->AddPlugin( CLASSINFO( cbAntiflickerPlugin ) );
    mpLayout->AddPlugin( CLASSINFO( cbSimpleCustomizationPlugin )
);

    // drop in some bars
    cbDimInfo sizes0( 200,45, // when docked horizontally
                     200,85, // when docked vertically
                     175,35, // when floated
                     FALSE,  // the bar is not fixed-size
                     4,      // vertical gap (bar border)
                     4       // horizontal gap (bar border)
                     );

    cbDimInfo sizes1( 150,35, // when docked horizontally
                     150,85, // when docked vertically
                     175,35, // when floated
                     TRUE,   // the bar is not fixed-size
                     4,      // vertical gap (bar border)
                     4       // horizontal gap (bar border)
                     );

    cbDimInfo sizes2( 175,45, // when docked horizontally
                     175,37, // when docked vertically
                     170,35, // when floated
                     TRUE,   // the bar is not fixed-size
                     4,      // vertical gap (bar border)
                     4,      // horizontal gap (bar border)
                     new cbDynToolBarDimHandler()
                     );

    mpLayout->AddBar( CreateTextCtrl("Hello"), // bar window
                     sizes0, FL_ALIGN_TOP,    // alignment ( 0-
top,1-bottom, etc)
                     0,                        // insert into
0th row (vert. position)
                     0,                        // offset from
the start of row (in pixels)
                     "InfoViewer1",          // name for
reference in customization pop-ups
                     TRUE
                     );

    mpLayout->AddBar( CreateTextCtrl("Bye"),    // bar window

```

```

        sizes0, FL_ALIGN_TOP,        // alignment ( 0-
top,1-bottom, etc)
        1,                            // insert into
0th row (vert. position)
        0,                            // offset from
the start of row (in pixels)
        "InfoViewer2",                // name for
reference in customization pop-ups
        TRUE
    );

    mpLayout->AddBar( CreateTextCtrl("Fixed0"), // bar window
        sizes1, FL_ALIGN_TOP,        // alignment ( 0-
top,1-bottom, etc)
        0,                            // insert into
0th row (vert. position)
        0,                            // offset from
the start of row (in pixels)
        "ToolBar1",                  // name for
reference in customization pop-ups
        TRUE
    );

    wxDynamicToolBar* pToolBar = new wxDynamicToolBar();

    pToolBar->Create( this, -1 );

    // 1001-1006 ids of command events fired by added tool-
buttons

    pToolBar->AddTool( 1001, BMP_DIR "new.bmp" );
    pToolBar->AddTool( 1002, BMP_DIR "open.bmp" );
    pToolBar->AddTool( 1003, BMP_DIR "save.bmp" );

    pToolBar->AddTool( 1004, BMP_DIR "cut.bmp" );
    pToolBar->AddTool( 1005, BMP_DIR "copy.bmp" );
    pToolBar->AddTool( 1006, BMP_DIR "paste.bmp" );

    mpLayout->AddBar( pToolBar,        // bar window (can be
NULL)
        sizes2, FL_ALIGN_TOP, // alignment ( 0-
top,1-bottom, etc)
        0,                            // insert into 0th
row (vert. position)
        0,                            // offset from the
start of row (in pixels)
        "ToolBar2",                  // name for reference
in customization pop-ups
        FALSE
    );

    mpLayout->EnableFloating( TRUE ); // off, thinking about
wxGtk...
}

MyFrame::~MyFrame()
{

```

```
    if ( mpLayout)
        delete mpLayout; // should be destroyed manually
}
```


^{\$#+K!}Controlling dragging behaviour

Various pane-dragging behaviours are supported. FL can show an outline of where the window would be docked if you stopped dragging at that point.

This is a list of properties of interest in the `cbCommonPaneProperties` structure:

```
bool mRealTimeUpdatesOn;      // default: ON
bool mOutOfPaneDragOn;       // default: ON
bool mExactDockPredictionOn; // default: OFF
bool mNonDestructFrictionOn; // default: OFF
```

To get behaviour similar to Microsoft's DevStudio drag-ghost behaviour, `mRealTimeUpdatesOn` have to be set to `FALSE`, for example:

```
cbCommonPaneProperties props;
....
....
props.mRealTimeUpdatesOn = FALSE;
fl->SetPaneProperties( props, wxALL_PANES );
```

mOutOfPaneDragOn specifies whether bars can be dragged away from this pane.
(Note: this may not currently be working.)

mExactDockPredictionOn is only relevant when *mRealTimeUpdatesOn* is `FALSE`, and then the hint rectangle behaves a little jumpily. It tries to show exactly how the bar would look and where it would be docked if the dragging finished right now, i.e. the final position, with all the 'friction-physics' calculated. Otherwise the hint flies smoothly above the surface only hinting whether the bar will be docked vertically or horizontally if dropped now. This is a feature you won't find anywhere else!

mNonDestructFrictionOn causes the bars not being dragged to stay where they are, while the currently dragged one is 'diving' through the underlying panes, docking itself in and out in real time. Otherwise the stationary bars would be pushed around messing up the composition permanently. This flag is irrelevant when *mRealTimeUpdatesOn* is `FALSE`, as the ghost-rect does not do any docking until the drag finishes.

^Controlling dragging behaviour

^controllingdragbehav

^browse00007

^K Controlling dragging behaviour

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp', `introduction`)")

BagLayout

BagLayout lays out items in left-to-right order from top to bottom.

Derived from

LayoutManagerBase

Include files

<wx/fl/dyntbar.h>

Data structures

Members

BagLayout::Layout

^BagLayout

^baglayout

^browse00009

^K BagLayout

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `classref)")

`$#+K!` **wxBarIterator**

Used for traversing through all bars of all rows in the pane.

Derived from

No base class

Include files

<wx/fl/controlbar.h>

Data structures

Members

[wxBarIterator::wxBarIterator](#)

[wxBarIterator::BarInfo](#)

[wxBarIterator::Next](#)

[wxBarIterator::Reset](#)

[wxBarIterator::RowInfo](#)

^wxBarIterator

^wxbariterator

^browse00011

^K wxBarIterator

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `classref')")

`cbAntiflickerPlugin`

Implements double-buffering to reduce flicker. Bitmap and memory DC buffers are shared 'resources' among all instances of antiflicker plugins within the application.

Locking for multithreaded applications is not yet implemented.

Derived from

[cbPluginBase](#)

Include files

<wx/fl/antiflickpl.h>

Data structures

Members

[cbAntiflickerPlugin::cbAntiflickerPlugin](#)
[cbAntiflickerPlugin::~~cbAntiflickerPlugin](#)
[cbAntiflickerPlugin::AllocNewBuffer](#)
[cbAntiflickerPlugin::FindSuitableBuffer](#)
[cbAntiflickerPlugin::GetClientDC](#)
[cbAntiflickerPlugin::GetWindowDC](#)
[cbAntiflickerPlugin::OnFinishDrawInArea](#)
[cbAntiflickerPlugin::OnStartDrawInArea](#)

`cbAntiflickerPlugin`

`cbantiflickerplugin`

`rowse00017`

`cbAntiflickerPlugin`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp', `classref`)"`

`cbBarDimHandlerBase`

Abstract interface for bar-size handler classes. These objects receive notifications whenever the docking state of the bar is changed, thus they provide the possibility to adjust the values in `cbDimInfo::mSizes` accordingly. Specific handlers can be hooked up to specific types of bar.

Derived from

[wxObject](#)

Include files

<wx/fl/controlbar.h>

Data structures

Members

[cbBarDimHandlerBase::cbBarDimHandlerBase](#)
[cbBarDimHandlerBase::AddRef](#)
[cbBarDimHandlerBase::OnChangeBarState](#)
[cbBarDimHandlerBase::OnResizeBar](#)
[cbBarDimHandlerBase::RemoveRef](#)

^cbBarDimHandlerBase

^cbbardimhandlerbase

^browse00026

^K cbBarDimHandlerBase

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `classref')")

\$#+K! **cbBarDragPlugin**

Plugin class implementing bar dragging.

Derived from

[cbPluginBase](#)

Include files

<wx/fl/bardragpl.h>

Data structures

Members

[cbBarDragPlugin::cbBarDragPlugin](#)
[cbBarDragPlugin::~~cbBarDragPlugin](#)
[cbBarDragPlugin::AdjustHintRect](#)
[cbBarDragPlugin::CalcOnScreenDims](#)
[cbBarDragPlugin::ClipPosInFrame](#)
[cbBarDragPlugin::ClipRectInFrame](#)
[cbBarDragPlugin::DoDrawHintRect](#)
[cbBarDragPlugin::DrawHintRect](#)
[cbBarDragPlugin::EraseHintRect](#)
[cbBarDragPlugin::FinishTracking](#)
[cbBarDragPlugin::GetBarHeightInPane](#)
[cbBarDragPlugin::GetBarWidthInPane](#)
[cbBarDragPlugin::GetDistanceToPane](#)
[cbBarDragPlugin::HitTestPanes](#)
[cbBarDragPlugin::HitsPane](#)
[cbBarDragPlugin::IsInClientArea](#)
[cbBarDragPlugin::IsInOtherPane](#)
[cbBarDragPlugin::OnDrawHintRect](#)
[cbBarDragPlugin::OnLButtonDown](#)
[cbBarDragPlugin::OnLButtonUp](#)
[cbBarDragPlugin::OnLDbClick](#)
[cbBarDragPlugin::OnMouseMove](#)
[cbBarDragPlugin::OnStartBarDragging](#)
[cbBarDragPlugin::RectToScr](#)
[cbBarDragPlugin::ShowHint](#)
[cbBarDragPlugin::StartTracking](#)
[cbBarDragPlugin::StickToPane](#)
[cbBarDragPlugin::UnstickFromPane](#)

^cbBarDragPlugin

^cbbardragplugin

^browse00032

^K cbBarDragPlugin

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp', `classref`)")

`$$$K!cbBarHintsPlugin`

This class intercepts bar-decoration and sizing events, and draws 3D hints around fixed and flexible bars, similar to those in Microsoft DevStudio 6.x

Derived from

[cbPluginBase](#)

Include files

<wx/fl/barhintspl.h>

Data structures

Members

[cbBarHintsPlugin::cbBarHintsPlugin](#)
[cbBarHintsPlugin::~~cbBarHintsPlugin](#)
[cbBarHintsPlugin::CreateBoxes](#)
[cbBarHintsPlugin::DoDrawHint](#)
[cbBarHintsPlugin::Draw3DBox](#)
[cbBarHintsPlugin::DrawCloseBox](#)
[cbBarHintsPlugin::DrawCollapseBox](#)
[cbBarHintsPlugin::DrawGrooves](#)
[cbBarHintsPlugin::ExcludeHints](#)
[cbBarHintsPlugin::GetHintsLayout](#)
[cbBarHintsPlugin::HitTestHints](#)
[cbBarHintsPlugin::OnDrawBarDecorations](#)
[cbBarHintsPlugin::OnInitPlugin](#)
[cbBarHintsPlugin::OnLeftDown](#)
[cbBarHintsPlugin::OnLeftUp](#)
[cbBarHintsPlugin::OnMotion](#)
[cbBarHintsPlugin::OnSizeBarWindow](#)
[cbBarHintsPlugin::SetGrooveCount](#)

`^bBarHintsPlugin`

`^bbarhintsplugin`

`^rowse00061`

`^K cbBarHintsPlugin`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^classref)")`

`$$$K!cbBarInfo`

Helper class used internally by the `wxFrameLayout` class. Holds and manages bar information.

Derived from

[`wxObject`](#)

Include files

`<wx/fl/controlbar.h>`

Data structures

Members

[`cbBarInfo::cbBarInfo`](#)
[`cbBarInfo::~~cbBarInfo`](#)
[`cbBarInfo::IsExpanded`](#)
[`cbBarInfo::IsFixed`](#)

`^bBarInfo`

`^bbarinfo`

`^rowse00080`

`^cbBarInfo`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^classref)")`

`cbBarShapeData`

Used for storing the original bar's positions in the row, when the 'non-destructive-friction' option is turned on.

Derived from

[wxObject](#)

Include files

<wx/fl/controlbar.h>

Data structures

`cbBarShapeData`

`bbarshapedata`

`rowse00085`

`cbBarShapeData`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `classref`)"`

`##+K!` **cbBarSpy**

Helper class, used for spying for unhandled mouse events on control bars and forwarding them to the frame layout.

Derived from

[wxEvtHandler](#)

Include files

<wx/fl/controlbar.h>

Data structures

```
typedef cbBarInfo* BarInfoPtrT
forward declarations
```

```
typedef cbRowInfo* RowInfoPtrT
enumeration of hittest results, see cbDockPane::HitTestPanelItems(..)enum
CB_HITTEST_RESULT
{
    CB_NO_ITEMS_HITTED,

    CB_UPPER_ROW_HANDLE_HITTED,
    CB_LOWER_ROW_HANDLE_HITTED,
    CB_LEFT_BAR_HANDLE_HITTED,
    CB_RIGHT_BAR_HANDLE_HITTED,
    CB_BAR_CONTENT_HITTED
}
```

Members

[cbBarSpy::cbBarSpy](#)
[cbBarSpy::ProcessEvent](#)
[cbBarSpy::SetBarWindow](#)

`^bBarSpy`

`^bbarspy`

`^rowse00086`

`^K cbBarSpy`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^classref)")`

cbCloseBox

cbCloseBox is a window close button, used in a wxToolWindow titlebar.

Derived from

[cbMiniButton](#)

Include files

<wx/fl/toolwnd.h>

Data structures

Members

[cbCloseBox::Draw](#)

^cbCloseBox

^cbclosebox

^browse00090

^K cbCloseBox

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ fl.hlp', `classref)")

`cbCollapseBox`

`cbCollapseBox` is a window collapse button, used in a `wxToolWindow` titlebar.

Derived from

[cbMiniButton](#)

Include files

`<wx/fl/toolwnd.h>`

Data structures

Members

[cbCollapseBox::Draw](#)

`cbCollapseBox`

`bcollapsebox`

`rowse00092`

`cbCollapseBox`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `classref`)"`

`cbCommonPaneProperties`

A structure holding configuration options, which are usually the same for all panes in a frame layout. For an explanation of the data members, please see [Controlling dragging behaviour](#).

Derived from

[wxObject](#)

Include files

<wx/fl/controlbar.h>

Data structures

```
class cbCommonPaneProperties : public wxObject
{
    DECLARE_DYNAMIC_CLASS( cbCommonPaneProperties )

    // Look-and-feel configuration

    bool mRealTimeUpdatesOn;      // default: ON
    bool mOutOfPaneDragOn;       // default: ON
    bool mExactDockPredictionOn; // default: OFF
    bool mNonDestructFrictionOn; // default: OFF

    bool mShow3DPaneBorderOn;     // default: ON

    // The following properties are reserved for the "future"

    bool mBarFloatingOn;          // default: OFF
    bool mRowProportionsOn;       // default: OFF
    bool mColProportionsOn;       // default: ON
    bool mBarCollapseIconsOn;     // default: OFF
    bool mBarDragHintsOn;         // default: OFF

    // Minimal dimensions for not-fixed bars in this pane (16x16
    default)

    wxSize mMinCBarDim;

    // Width/height of resizing sash

    int    mResizeHandleSize;

    // Default constructor.
```

`cbCommonPaneProperties`

`cbcommonpaneproperties`

`rowse00094`

`cbCommonPaneProperties`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp', `classref`)"`

```
        cbCommonPaneProperties(void);  
};
```

cbCustomizeBarEvent

Class for bar customization events.

Derived from

[cbPluginEvent](#)

Include files

<wx/fl/controlbar.h>

Data structures

Members

[cbCustomizeBarEvent::cbCustomizeBarEvent](#)

^cbCustomizeBarEvent

^cbcustomizebarevent

^browse00095

^K cbCustomizeBarEvent

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `classref`)")

`cbCustomizeLayoutEvent`

Class for layout customization events.

Derived from

[cbPluginEvent](#)

Include files

<wx/fl/controlbar.h>

Data structures

Members

[cbCustomizeLayoutEvent::cbCustomizeLayoutEvent](#)

`cbCustomizeLayoutEvent`

`bcustomizelayoutevent`

`rowse00097`

`cbCustomizeLayoutEvent`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `classref')")`

`cbDimInfo`

Helper class used internally by the `wxFrameLayout` class. Holds and manages information about bar dimensions.

Derived from

[wxObject](#)

Include files

`<wx/fl/controlbar.h>`

Data structures

Members

[cbDimInfo::cbDimInfo](#)
[cbDimInfo::~~cbDimInfo](#)
[cbDimInfo::GetDimHandler](#)
[cbDimInfo::operator=](#)

`cbDimInfo`

`cbdiminfo`

`rowse00099`

`cbDimInfo`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp', `classref`)"`

cbDockBox

cbDockBox is a window dock button, used in a wxToolWindow titlebar.

Derived from

[cbMiniButton](#)

Include files

<wx/fl/toolwnd.h>

Data structures

Members

[cbDockBox::Draw](#)

^cbDockBox

^cbdockbox

^browse00104

^K cbDockBox

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ fl.hlp', `classref)")

`$$$K!cbDockPane`

This class manages containment and control of control bars along one of the four edges of the parent frame.

Derived from

[wxObject](#)

Include files

<wx/fl/controlbar.h>

Data structures

Members

[cbDockPane::cbDockPane](#)
[cbDockPane::~~cbDockPane](#)
[cbDockPane::BarPresent](#)
[cbDockPane::CalcLengthRatios](#)
[cbDockPane::ContractBar](#)
[cbDockPane::DoInsertBar](#)
[cbDockPane::DrawHorizHandle](#)
[cbDockPane::DrawVertHandle](#)
[cbDockPane::ExpandBar](#)
[cbDockPane::FinishDrawInArea](#)
[cbDockPane::FrameToPane](#)
[cbDockPane::GetAlignment](#)
[cbDockPane::GetBarInfoByWindow](#)
[cbDockPane::GetBarResizeRange](#)
[cbDockPane::GetDockingState](#)
[cbDockPane::GetFirstRow](#)
[cbDockPane::GetMinimalRowHeight](#)
[cbDockPane::GetNotFixedBarsCount](#)
[cbDockPane::GetPaneHeight](#)
[cbDockPane::GetRealRect](#)
[cbDockPane::GetRow](#)
[cbDockPane::GetRowAt](#)
[cbDockPane::GetRowIndex](#)
[cbDockPane::GetRowList](#)
[cbDockPane::GetRowResizeRange](#)
[cbDockPane::GetRowShapeData](#)
[cbDockPane::GetRowY](#)

^cbDockPane

^cbdockpane

^browse00106

^K cbDockPane

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp', `classref`)")

cbDockPane::HasNotFixedBarsLeft
cbDockPane::HasNotFixedBarsRight
cbDockPane::HasNotFixedRowsAbove
cbDockPane::HasNotFixedRowsBelow
cbDockPane::HasPoint
cbDockPane::HitTestPanelItems
cbDockPane::InitLinksForRow
cbDockPane::InitLinksForRows
cbDockPane::InsertBar
cbDockPane::InsertRow
cbDockPane::IsFixedSize
cbDockPane::IsHorizontal
cbDockPane::MatchesMask
cbDockPane::PaintBar
cbDockPane::PaintBarDecorations
cbDockPane::PaintBarHandles
cbDockPane::PaintPane
cbDockPane::PaintPaneBackground
cbDockPane::PaintPaneDecorations
cbDockPane::PaintRow
cbDockPane::PaintRowBackground
cbDockPane::PaintRowDecorations
cbDockPane::PaintRowHandles
cbDockPane::PaneToFrame
cbDockPane::RecalcLayout
cbDockPane::RecalcRowLayout
cbDockPane::RemoveBar
cbDockPane::RemoveRow
cbDockPane::ResizeBar
cbDockPane::ResizeRow
cbDockPane::SetBoundsInParent
cbDockPane::SetMargins
cbDockPane::SetPaneWidth
cbDockPane::SetRowHeight
cbDockPane::SetRowShapeData
cbDockPane::SizeBar
cbDockPane::SizePaneObjects
cbDockPane::SizeRowObjects
cbDockPane::StartDrawInArea
cbDockPane::SyncRowFlags

`cbDrawBarDecorEvent`

Class for bar decoration drawing events.

Derived from

[cbPluginEvent](#)

Include files

<wx/fl/controlbar.h>

Data structures

Members

[cbDrawBarDecorEvent::cbDrawBarDecorEvent](#)

`cbDrawBarDecorEvent`

`cbdrawbardecocorevent`

`rowse00174`

`cbDrawBarDecorEvent`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^ fl.hlp', `classref`)"`

`cbDrawBarHandlesEvent`

Class for bar handles drawing events.

Derived from

[cbPluginEvent](#)

Include files

<wx/fl/controlbar.h>

Data structures

Members

[cbDrawBarHandlesEvent::cbDrawBarHandlesEvent](#)

`cbDrawBarHandlesEvent`

`cbdrawbarhandlesevent`

`rowse00176`

`cbDrawBarHandlesEvent`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `classref`)"`

`$$$K!cbDrawHintRectEvent`

Class for hint-rectangle drawing events.

Derived from

[cbPluginEvent](#)

Include files

`<wx/fl/controlbar.h>`

Data structures

Members

[cbDrawHintRectEvent::cbDrawHintRectEvent](#)

`^bDrawHintRectEvent`

`^bdrawhintrectevent`

`^rowse00178`

`^cbDrawHintRectEvent`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^classref)")`

`cbDrawPaneBkGroundEvent`

Class for pane background drawing events.

Derived from

[cbPluginEvent](#)

Include files

<wx/fl/controlbar.h>

Data structures

Members

[cbDrawPaneBkGroundEvent::cbDrawPaneBkGroundEvent](#)

`cbDrawPaneBkGroundEvent`

`cbdrawpanebkgroundevent`

`rowse00180`

`cbDrawPaneBkGroundEvent`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp', `classref`)"`

`cbDrawPaneDecorEvent`

Class for pane decoration drawing events.

Derived from

[cbPluginEvent](#)

Include files

<wx/fl/controlbar.h>

Data structures

Members

[cbDrawPaneDecorEvent::cbDrawPaneDecorEvent](#)

`cbDrawPaneDecorEvent`

`cbdrawpanedecorevent`

`rowse00182`

`cbDrawPaneDecorEvent`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp', `classref`)"`

`cbDrawRowBkGroundEvent`

Class for row background drawing events.

Derived from

[cbPluginEvent](#)

Include files

`<wx/fl/controlbar.h>`

Data structures

Members

[cbDrawRowBkGroundEvent::cbDrawRowBkGroundEvent](#)

`cbDrawRowBkGroundEvent`

`cbdrawrowbkgroundevent`

`rowse00184`

`cbDrawRowBkGroundEvent`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^ fl.hlp', `classref)")`

`cbDrawRowDecorEvent`

Class for row decoration drawing events.

Derived from

[cbPluginEvent](#)

Include files

<wx/fl/controlbar.h>

Data structures

Members

[cbDrawRowDecorEvent::cbDrawRowDecorEvent](#)

`cbDrawRowDecorEvent`

`cbdrawrowdecorevent`

`rowse00186`

`cbDrawRowDecorEvent`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^ fl.hlp', `classref`)"`

`cbDrawRowHandlesEvent`

Class for row handles drawing events.

Derived from

[cbPluginEvent](#)

Include files

<wx/fl/controlbar.h>

Data structures

Members

[cbDrawRowHandlesEvent::cbDrawRowHandlesEvent](#)

`cbDrawRowHandlesEvent`

`cbdrawrowhandlesevent`

`rowse00188`

`cbDrawRowHandlesEvent`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `classref')")`

`$$$K!cbDynToolBarDimHandler`

Dynamic toolbar dimension handler.

Derived from

[cbBarDimHandlerBase](#)

Include files

`<wx/fl/dyntbarhnd.h>`

Data structures

Members

[cbDynToolBarDimHandler::OnChangeBarState](#)

[cbDynToolBarDimHandler::OnResizeBar](#)

`cbDynToolBarDimHandler`

`cbdyntoolbardimhandler`

`rowse00190`

`cbDynToolBarDimHandler`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `classref`)"`

cbFinishDrawInAreaEvent

Class for finish drawing in area events.

Derived from

[cbPluginEvent](#)

Include files

<wx/fl/controlbar.h>

Data structures

Members

[cbFinishDrawInAreaEvent::cbFinishDrawInAreaEvent](#)

^cbFinishDrawInAreaEvent

^cbfinishdrawinareaevent

^browse00193

^K cbFinishDrawInAreaEvent

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ fl.hlp', `classref)")

`$$$K!cbFloatedBarWindow`

`cbFloatedBarWindow` is a kind of `wxToolWindow`, implementing floating toolbars.

Derived from

[wxToolWindow](#)

Include files

`<wx/fl/toolwnd.h>`

Data structures

Members

[cbFloatedBarWindow::cbFloatedBarWindow](#)
[cbFloatedBarWindow::GetBar](#)
[cbFloatedBarWindow::GetPreferredSize](#)
[cbFloatedBarWindow::HandleTitleClick](#)
[cbFloatedBarWindow::OnDbClick](#)
[cbFloatedBarWindow::OnMiniButtonClicked](#)
[cbFloatedBarWindow::PositionFloatedWnd](#)
[cbFloatedBarWindow::SetBar](#)
[cbFloatedBarWindow::SetLayout](#)

`^bFloatedBarWindow`

`^bfloatedbarwindow`

`^rowse00195`

`^K cbFloatedBarWindow`

`^enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^classref)")`

cbGCUpdatesMgr

This class implements optimized logic for refreshing the areas of frame layout that actually need to be updated. It is used as the default updates manager by wxFrameLayout.

It is called 'Garbage Collecting' updates manager because its implementation tries to find out dependencies between bars, and to order them into a 'hierarchy'. This hierarchical sorting resembles the implementation of heap-garbage collectors, which resolve dependencies between references.

Example: there are situations where the order in which the user moves windows does matter.



Past/future positions of A and B windows completely overlap, i.e. depend on each other, and there is no solution for moving the windows without refreshing both of them -- we have a cyclic dependency here. The garbage collection algorithm will find this cyclic dependency and will force refresh after movement.



cbGCUpdatesMgr

cbgcupdatesmgr

rowse00205

cbGCUpdatesMgr

enableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp', `classref`)"

In this case past/future positions do not overlap, so it is enough only to move windows without refreshing them. Garbage collection will 'notice' this.

There is also a third case, when overlapping is partial. In this case the refreshing can also be avoided by moving windows in the order of 'most-dependant' towards the 'least-dependent'. GC handles this automatically, by sorting windows by their dependency-level (or 'hierarchy').

See `garbagec.h` for more details of this method; `garbagec.h/cpp` implement sorting of generic dependencies and does not deal with graphical objects directly.

Summary: garbage collection improves performance when complex or large windows are moved around, by reducing the number of repaints. It also helps to avoid dirty non-client areas of moved windows in some special cases of 'overlapping anomalies'.

Derived from

[cbSimpleUpdatesMgr](#)

Include files

`<wx/fl/gcupdatesmgr.h>`

Data structures

Members

[cbGCUpdatesMgr::cbGCUpdatesMgr](#)
[cbGCUpdatesMgr::AddItem](#)
[cbGCUpdatesMgr::DoRepositionItems](#)
[cbGCUpdatesMgr::OnStartChanges](#)
[cbGCUpdatesMgr::UpdateNow](#)

`cbHintAnimationPlugin`

A plugin to draw animated hints when the user drags a pane.

Derived from

[cbPluginBase](#)

Include files

<wx/fl/hintanimpl.h>

Data structures

Members

[cbHintAnimationPlugin::cbHintAnimationPlugin](#)
[cbHintAnimationPlugin::~~cbHintAnimationPlugin](#)
[cbHintAnimationPlugin::DoDrawHintRect](#)
[cbHintAnimationPlugin::DrawHintRect](#)
[cbHintAnimationPlugin::EraseHintRect](#)
[cbHintAnimationPlugin::FinishTracking](#)
[cbHintAnimationPlugin::OnDrawHintRect](#)
[cbHintAnimationPlugin::RectToScr](#)
[cbHintAnimationPlugin::StartTracking](#)

^cbHintAnimationPlugin

^cbhintanimationplugin

^browse00211

^K cbHintAnimationPlugin

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `classref')")

`$$$K!cbInsertBarEvent`

Class for bar insertion events.

Derived from

[cbPluginEvent](#)

Include files

`<wx/fl/controlbar.h>`

Data structures

Members

[cbInsertBarEvent::cbInsertBarEvent](#)

`^bInsertBarEvent`

`^binsertbarevent`

`^rowse00221`

`^cbInsertBarEvent`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^ fl.hlp', ^classref)")`

`$$$K!cbLayoutRowEvent`

Class for single row layout events.

Derived from

[cbPluginEvent](#)

Include files

`<wx/fl/controlbar.h>`

Data structures

Members

[cbLayoutRowEvent::cbLayoutRowEvent](#)

`^bLayoutRowEvent`

`^blayoutrowevent`

`^rowse00223`

`^cbLayoutRowEvent`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `classref`)"`

`cbLayoutRowsEvent`

Class for multiple rows layout events.

Derived from

[cbPluginEvent](#)

Include files

<wx/fl/controlbar.h>

Data structures

Members

[cbLayoutRowsEvent::cbLayoutRowsEvent](#)

`cbLayoutRowsEvent`

`cbayoutrowsevent`

`rowse00225`

`cbLayoutRowsEvent`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `classref`)"`

`$$$K!cbLeftDClickEvent`

Class for mouse left double click events.

Derived from

[cbPluginEvent](#)

Include files

<wx/fl/controlbar.h>

Data structures

Members

[cbLeftDClickEvent::cbLeftDClickEvent](#)

`^bLeftDClickEvent`

`^bleftdclিকেvent`

`^rowse00227`

`^cbLeftDClickEvent`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `classref`)"`

`$$$K!cbLeftDownEvent`

Class for mouse left down events.

Derived from

[cbPluginEvent](#)

Include files

<wx/fl/controlbar.h>

Data structures

Members

[cbLeftDownEvent::cbLeftDownEvent](#)

`^bLeftDownEvent`

`^bleftdownevent`

`^rowse00229`

`^cbLeftDownEvent`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `classref`)"`

`$$$K!cbLeftUpEvent`

Class for mouse left up events.

Derived from

[cbPluginEvent](#)

Include files

<wx/fl/controlbar.h>

Data structures

Members

[cbLeftUpEvent::cbLeftUpEvent](#)

`^bLeftUpEvent`

`^bleftupevent`

`^rowse00231`

`^cbLeftUpEvent`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^classref)")`

cbMiniButton

cbMiniButton is the base class for a small button that can be placed in a wxToolWindow titlebar.

Derived from

[wxObject](#)

Include files

<wx/fl/toolwnd.h>

Data structures

Members

[cbMiniButton::cbMiniButton](#)
[cbMiniButton::Draw](#)
[cbMiniButton::Enable](#)
[cbMiniButton::HitTest](#)
[cbMiniButton::IsPressed](#)
[cbMiniButton::OnLeftDown](#)
[cbMiniButton::OnLeftUp](#)
[cbMiniButton::OnMotion](#)
[cbMiniButton::Refresh](#)
[cbMiniButton::Reset](#)
[cbMiniButton::SetPos](#)
[cbMiniButton::WasClicked](#)

^cbMiniButton

^cbminibutton

^browse00233

^K cbMiniButton

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `classref)")

`$$$K!cbMotionEvent`

Class for mouse motion events.

Derived from

[cbPluginEvent](#)

Include files

<wx/fl/controlbar.h>

Data structures

Members

[cbMotionEvent::cbMotionEvent](#)

`cbMotionEvent`

`cbmotionevent`

`rowse00246`

`cbMotionEvent`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^ fl.hlp', `classref`)"`

`$$$K!cbPaneDrawPlugin`

Simple but all-in-one plugin implementation. Resembles the look and feel of to MFC control-bars. The class handles painting of the pane and the items in it; it generates bar/layout customization events, when the user right-clicks the bar/pane. Hooking an instance of this and row-layout plugins for each pane would be enough for the frame layout to function properly (they are plugged in automatically by the `wxFrameLayout` class).

Derived from

[cbPluginBase](#)

Include files

`<wx/fl/panedrawpl.h>`

Data structures

Members

[cbPaneDrawPlugin::cbPaneDrawPlugin](#)
[cbPaneDrawPlugin::~~cbPaneDrawPlugin](#)
[cbPaneDrawPlugin::Clone](#)
[cbPaneDrawPlugin::DrawBarInnerShadeRect](#)
[cbPaneDrawPlugin::DrawDraggedHandle](#)
[cbPaneDrawPlugin::DrawLowerRowHandle](#)
[cbPaneDrawPlugin::DrawLowerRowShades](#)
[cbPaneDrawPlugin::DrawPaneShade](#)
[cbPaneDrawPlugin::DrawPaneShadeForRow](#)
[cbPaneDrawPlugin::DrawShade](#)
[cbPaneDrawPlugin::DrawShade1](#)
[cbPaneDrawPlugin::DrawUpperRowHandle](#)
[cbPaneDrawPlugin::DrawUpperRowShades](#)
[cbPaneDrawPlugin::OnDrawBarDecorations](#)
[cbPaneDrawPlugin::OnDrawBarHandles](#)
[cbPaneDrawPlugin::OnDrawPaneBackground](#)
[cbPaneDrawPlugin::OnDrawPaneDecorations](#)
[cbPaneDrawPlugin::OnDrawRowBackground](#)
[cbPaneDrawPlugin::OnDrawRowDecorations](#)
[cbPaneDrawPlugin::OnDrawRowHandles](#)
[cbPaneDrawPlugin::OnFinishDrawInArea](#)
[cbPaneDrawPlugin::OnLButtonDown](#)
[cbPaneDrawPlugin::OnLButtonUp](#)

`^bPaneDrawPlugin`

`^bpanedrawplugin`

`^rowse00248`

`^K cbPaneDrawPlugin`

`^enableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp', `classref')")`

cbPaneDrawPlugin::OnLDbClick
cbPaneDrawPlugin::OnMouseMove
cbPaneDrawPlugin::OnRButtonUp
cbPaneDrawPlugin::OnSizeBarWindow
cbPaneDrawPlugin::OnStartDrawInArea
cbPaneDrawPlugin::SetDarkPixel
cbPaneDrawPlugin::SetLightPixel

`$$$K!cbPluginBase`

Abstract base class for all control-bar related plugins. Note: pointer positions of mouse events sent to plugins are always in the pane's coordinates (the pane to which this plugin is hooked).

Derived from

[wxEvtHandler](#)

Include files

<wx/fl/controlbar.h>

Data structures

Members

[cbPluginBase::cbPluginBase](#)
[cbPluginBase::~~cbPluginBase](#)
[cbPluginBase::GetPaneMask](#)
[cbPluginBase::IsReady](#)
[cbPluginBase::OnInitPlugin](#)
[cbPluginBase::ProcessEvent](#)

^cbPluginBase

^cbpluginbase

^browse00279

^K cbPluginBase

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `classref')")

`$$$K!cbPluginEvent`

Base class for all control-bar plugin events. This is not a dynamically-creatable class.

Derived from

[wxEvent](#)

Include files

<wx/fl/controlbar.h>

Data structures

Members

[cbPluginEvent::cbPluginEvent](#)

[cbPluginEvent::Clone](#)

`^bPluginEvent`

`^bpluginevent`

`^rowse00286`

`^K cbPluginEvent`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^ fl.hlp', ^classref)")`

`$$$K!cbRemoveBarEvent`

Class for bar removal events.

Derived from

[cbPluginEvent](#)

Include files

`<wx/fl/controlbar.h>`

Data structures

Members

[cbRemoveBarEvent::cbRemoveBarEvent](#)

`^bRemoveBarEvent`

`^bremovebarevent`

`^rowse00289`

`^cbRemoveBarEvent`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^classref)")`

`$$$K!cbResizeBarEvent`

Class for bar resize events.

Derived from

[cbPluginEvent](#)

Include files

`<wx/fl/controlbar.h>`

Data structures

Members

[cbResizeBarEvent::cbResizeBarEvent](#)

`^bResizeBarEvent`

`^bresizebarevent`

`^rowse00291`

`^cbResizeBarEvent`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^classref)")`

`$$$K!cbResizeRowEvent`

Class for row resize events.

Derived from

[cbPluginEvent](#)

Include files

`<wx/fl/controlbar.h>`

Data structures

Members

[cbResizeRowEvent::cbResizeRowEvent](#)

`^bResizeRowEvent`

`^bresizerowevent`

`^rowse00293`

`^cbResizeRowEvent`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^classref)")`

`cbRightDownEvent`

Class for mouse right down events.

Derived from

[cbPluginEvent](#)

Include files

<wx/fl/controlbar.h>

Data structures

Members

[cbRightDownEvent::cbRightDownEvent](#)

`cbRightDownEvent`

`brightdownevent`

`rowse00295`

`cbRightDownEvent`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `classref`)"`

`$$$K!cbRightUpEvent`

Class for mouse right up events.

Derived from

[cbPluginEvent](#)

Include files

<wx/fl/controlbar.h>

Data structures

Members

[cbRightUpEvent::cbRightUpEvent](#)

`^bRightUpEvent`

`^brightupevent`

`^rowse00297`

`^cbRightUpEvent`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^ fl.hlp', ^classref)")`

`cbRowDragPlugin`

This plugin adds row-dragging functionality to the pane. It handles mouse movement and pane background-erasing plugin events. The behaviour and appearance resembles drag and drop positioning of the toolbar rows in Netscape Communicator 4.xx.

Derived from

[cbPluginBase](#)

Include files

`<wx/fl/rowdragpl.h>`

Data structures

Members

[cbRowDragPlugin::cbRowDragPlugin](#)
[cbRowDragPlugin::~~cbRowDragPlugin](#)
[cbRowDragPlugin::CaptureDCArea](#)
[cbRowDragPlugin::CheckPrevItemInFocus](#)
[cbRowDragPlugin::Clone](#)
[cbRowDragPlugin::CollapseRow](#)
[cbRowDragPlugin::Draw3DPattern](#)
[cbRowDragPlugin::Draw3DRect](#)
[cbRowDragPlugin::DrawCollapsedRowIcon](#)
[cbRowDragPlugin::DrawCollapsedRowsBorder](#)
[cbRowDragPlugin::DrawEmptyRow](#)
[cbRowDragPlugin::DrawOtoRomb](#)
[cbRowDragPlugin::DrawRectShade](#)
[cbRowDragPlugin::DrawRomb](#)
[cbRowDragPlugin::DrawRombShades](#)
[cbRowDragPlugin::DrawRowDragHint](#)
[cbRowDragPlugin::DrawRowsDragHintsBorder](#)
[cbRowDragPlugin::DrawTrianDown](#)
[cbRowDragPlugin::DrawTrianRight](#)
[cbRowDragPlugin::DrawTrianUp](#)
[cbRowDragPlugin::ExpandRow](#)
[cbRowDragPlugin::FinishOnScreenDraw](#)
[cbRowDragPlugin::GetCollapsedIconsPos](#)
[cbRowDragPlugin::GetCollapsedInconRect](#)
[cbRowDragPlugin::GetCollapsedRowIconHeight](#)
[cbRowDragPlugin::GetFirstRow](#)

`cbRowDragPlugin`

`cbrowdragplugin`

`rowse00299`

`cbRowDragPlugin`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp', `classref`)"`

cbRowDragPlugin::GetHRowsCountForPane
cbRowDragPlugin::GetRowDragHintWidth
cbRowDragPlugin::GetRowHintRect
cbRowDragPlugin::HitTestCollapsedRowIcon
cbRowDragPlugin::HitTestRowDragHint
cbRowDragPlugin::InsertDraggedRowBefore
cbRowDragPlugin::ItemsInFocus
cbRowDragPlugin::OnDrawPaneBackground
cbRowDragPlugin::OnInitPlugin
cbRowDragPlugin::OnLButtonDown
cbRowDragPlugin::OnLButtonUp
cbRowDragPlugin::OnMouseMove
cbRowDragPlugin::PrepareForRowDrag
cbRowDragPlugin::SetMouseCapture
cbRowDragPlugin::SetPaneMargins
cbRowDragPlugin::ShowDraggedRow
cbRowDragPlugin::ShowPanelImage
cbRowDragPlugin::UnhighlightItemInFocus

`$$$K!cbRowInfo`

Helper class used internally by the `wxFrameLayout` class. Holds and manages information about bar rows.

Derived from

[wxObject](#)

Include files

`<wx/fl/controlbar.h>`

Data structures

Members

[cbRowInfo::cbRowInfo](#)
[cbRowInfo::~cbRowInfo](#)
[cbRowInfo::GetFirstBar](#)

`^bRowInfo`

`^browinfo`

`^rowse00344`

`^cbRowInfo`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^classref)")`

cbRowLayoutPlugin

Simple implementation of a plugin which handles row layout requests sent from a frame layout.

Derived from

[cbPluginBase](#)

Include files

<wx/fl/rowlayoutpl.h>

Data structures

Members

[cbRowLayoutPlugin::cbRowLayoutPlugin](#)
[cbRowLayoutPlugin::AdjustLengthOfInserted](#)
[cbRowLayoutPlugin::ApplyLengthRatios](#)
[cbRowLayoutPlugin::CalcRowHeight](#)
[cbRowLayoutPlugin::CheckIfAtTheBoundary](#)
[cbRowLayoutPlugin::DetectBarHandles](#)
[cbRowLayoutPlugin::DoInsertBar](#)
[cbRowLayoutPlugin::ExpandNotFixedBars](#)
[cbRowLayoutPlugin::FitBarsToRange](#)
[cbRowLayoutPlugin::GetRowFreeSpace](#)
[cbRowLayoutPlugin::InsertBefore](#)
[cbRowLayoutPlugin::LayoutItemsVertically](#)
[cbRowLayoutPlugin::MinimizeNotFixedBars](#)
[cbRowLayoutPlugin::OnInsertBar](#)
[cbRowLayoutPlugin::OnLayoutRow](#)
[cbRowLayoutPlugin::OnLayoutRows](#)
[cbRowLayoutPlugin::OnRemoveBar](#)
[cbRowLayoutPlugin::OnResizeRow](#)
[cbRowLayoutPlugin::RecalcLengthRatios](#)
[cbRowLayoutPlugin::RelayoutNotFixedBarsAround](#)
[cbRowLayoutPlugin::ShiftLeftThreshold](#)
[cbRowLayoutPlugin::ShiftRightThreshold](#)
[cbRowLayoutPlugin::SlideLeftSideBars](#)
[cbRowLayoutPlugin::SlideRightSideBars](#)
[cbRowLayoutPlugin::StickRightSideBars](#)

^cbRowLayoutPlugin

^cbrowlayoutplugin

^browse00348

^K cbRowLayoutPlugin

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp', `classref`)")

`##+K!` **cbSimpleCustomizationPlugin**

This class enables customization of a bar, popping up a menu and handling basic customization such as floating and horizontal/vertical alignment of the bar.

Derived from

[cbPluginBase](#)

Include files

<wx/fl/cbcustom.h>

Data structures

Members

[cbSimpleCustomizationPlugin::cbSimpleCustomizationPlugin](#)

[cbSimpleCustomizationPlugin::OnCustomizeBar](#)

[cbSimpleCustomizationPlugin::OnCustomizeLayout](#)

[cbSimpleCustomizationPlugin::OnMenuItemSelected](#)

`^bSimpleCustomizationPlugin`

`^bsimplecustomizationplugin`

`^rowse00374`

`^K cbSimpleCustomizationPlugin`

`^enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^classref)")`

`$$$K!cbSimpleUpdatesMgr`

This class implements slightly optimized logic for refreshing the areas of frame layout that actually need to be updated.

Derived from

[cbUpdatesManagerBase](#)

Include files

<wx/fl/updatesmgr.h>

Data structures

Members

[cbSimpleUpdatesMgr::cbSimpleUpdatesMgr](#)
[cbSimpleUpdatesMgr::OnBarWillChange](#)
[cbSimpleUpdatesMgr::OnFinishChanges](#)
[cbSimpleUpdatesMgr::OnPaneMarginsWillChange](#)
[cbSimpleUpdatesMgr::OnPaneWillChange](#)
[cbSimpleUpdatesMgr::OnRowWillChange](#)
[cbSimpleUpdatesMgr::OnStartChanges](#)
[cbSimpleUpdatesMgr::UpdateNow](#)
[cbSimpleUpdatesMgr::WasChanged](#)

`^bSimpleUpdatesMgr`

`^bsimpleupdatesmgr`

`^rowse00379`

`^K cbSimpleUpdatesMgr`

`^enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^classref)")`

`$$$K!cbSizeBarWndEvent`

Class for bar window resize events.

Derived from

[cbPluginEvent](#)

Include files

`<wx/fl/controlbar.h>`

Data structures

Members

[cbSizeBarWndEvent::cbSizeBarWndEvent](#)

`^bSizeBarWndEvent`

`^bsizebarwndevent`

`^rowse00389`

`^cbSizeBarWndEvent`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^classref)")`

`cbStartBarDraggingEvent`

Class for start-bar-dragging events.

Derived from

[cbPluginEvent](#)

Include files

<wx/fl/controlbar.h>

Data structures

Members

[cbStartBarDraggingEvent::cbStartBarDraggingEvent](#)

`cbStartBarDraggingEvent`

`cbstartbardraggingevent`

`rowse00391`

`cbStartBarDraggingEvent`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `classref`)"`

cbStartDrawInAreaEvent

Class for start drawing in area events.

Derived from

[cbPluginEvent](#)

Include files

<wx/fl/controlbar.h>

Data structures

Members

[cbStartDrawInAreaEvent::cbStartDrawInAreaEvent](#)

^cbStartDrawInAreaEvent

^cbstartdrawinareaevent

^browse00393

^K cbStartDrawInAreaEvent

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ fl.hlp', `classref)")

`$$$K!cbUpdateMgrData`

A structure that is present in each item of layout, used by any particular updates-manager to store auxiliary information to be used by its updating algorithm.

Derived from

[wxObject](#)

Include files

<wx/fl/controlbar.h>

Data structures

Members

[cbUpdateMgrData::cbUpdateMgrData](#)

[cbUpdateMgrData::IsDirty](#)

[cbUpdateMgrData::SetCustomData](#)

[cbUpdateMgrData::SetDirty](#)

[cbUpdateMgrData::StoreItemState](#)

`^bUpdateMgrData`

`^bupdatemgrdata`

`^rowse00395`

`^K cbUpdateMgrData`

`^enableButton("Up");ChangeButtonBinding("Up", "JumpId(^ fl.hlp', ^classref)")`

cbUpdatesManagerBase

This class declares an abstract interface for optimized logic that should refresh areas of frame layout that actually need to be updated. This should be extended in future to implement a custom updating strategy.

Derived from

wxObject

Include files

<wx/fl/controlbar.h>

Data structures

Members

cbUpdatesManagerBase::cbUpdatesManagerBase
cbUpdatesManagerBase::~~cbUpdatesManagerBase
cbUpdatesManagerBase::OnBarWillChange
cbUpdatesManagerBase::OnFinishChanges
cbUpdatesManagerBase::OnPaneMarginsWillChange
cbUpdatesManagerBase::OnPaneWillChange
cbUpdatesManagerBase::OnRowWillChange
cbUpdatesManagerBase::OnStartChanges
cbUpdatesManagerBase::SetLayout
cbUpdatesManagerBase::UpdateNow

^cbUpdatesManagerBase

^cbupdatesmanagerbase

^browse00401

^K cbUpdatesManagerBase

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp', `classref`)")

`wxDynamicToolBar`

`wxDynamicToolBar` manages containment and layout of tool windows.

Derived from

[wxToolBarBase](#)

Include files

`<wx/fl/dyntbar.h>`

Data structures

Members

[wxDynamicToolBar::wxDynamicToolBar](#)
[wxDynamicToolBar::~~wxDynamicToolBar](#)
[wxDynamicToolBar::AddSeparator](#)
[wxDynamicToolBar::AddTool](#)
[wxDynamicToolBar::Create](#)
[wxDynamicToolBar::CreateDefaultLayout](#)
[wxDynamicToolBar::CreateTool](#)
[wxDynamicToolBar::DoDeleteTool](#)
[wxDynamicToolBar::DoEnableTool](#)
[wxDynamicToolBar::DoInsertTool](#)
[wxDynamicToolBar::DoSetToggle](#)
[wxDynamicToolBar::DoToggleTool](#)
[wxDynamicToolBar::DrawSeparator](#)
[wxDynamicToolBar::EnableTool](#)
[wxDynamicToolBar::FindToolForPosition](#)
[wxDynamicToolBar::GetPreferredDim](#)
[wxDynamicToolBar::GetToolInfo](#)
[wxDynamicToolBar::Layout](#)
[wxDynamicToolBar::OnEraseBackground](#)
[wxDynamicToolBar::OnPaint](#)
[wxDynamicToolBar::OnSize](#)
[wxDynamicToolBar::Realize](#)
[wxDynamicToolBar::RemoveTool](#)
[wxDynamicToolBar::SetLayout](#)
[wxDynamicToolBar::SizeToolWindows](#)

`wxDynamicToolBar`

`wxdynamictoolbar`

`rowse00412`

`wxDynamicToolBar`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp', `classref`)"`

`wxDynToolInfo`

This class holds dynamic toolbar item information.

Derived from

[wxToolLayoutItem](#)

Include files

<wx/fl/dyntbar.h>

Data structures

`wxDynToolInfo`

`wxdyntoolinfo`

`rowse00438`

`wxDynToolInfo`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `classref')")`

\$#+K! **wxFrameworkLayout**

wxFrameworkLayout manages containment and docking of control bars, which can be docked along the top, bottom, right, or left side of the parent frame.

Derived from

[wxEvtHandler](#)

Include files

<wx/fl/controlbar.h>

Data structures

Members

[wxFrameworkLayout::wxFrameworkLayout](#)
[wxFrameworkLayout::~~wxFrameworkLayout](#)
[wxFrameworkLayout::Activate](#)
[wxFrameworkLayout::AddBar](#)
[wxFrameworkLayout::AddPlugin](#)
[wxFrameworkLayout::AddPluginBefore](#)
[wxFrameworkLayout::ApplyBarProperties](#)
[wxFrameworkLayout::CanReparent](#)
[wxFrameworkLayout::CaptureEventsForPane](#)
[wxFrameworkLayout::CaptureEventsForPlugin](#)
[wxFrameworkLayout::CreateCursors](#)
[wxFrameworkLayout::CreateUpdatesManager](#)
[wxFrameworkLayout::Deactivate](#)
[wxFrameworkLayout::DestroyBarWindows](#)
[wxFrameworkLayout::DoSetBarState](#)
[wxFrameworkLayout::EnableFloating](#)
[wxFrameworkLayout::FindBarByName](#)
[wxFrameworkLayout::FindBarByWindow](#)
[wxFrameworkLayout::FindPlugin](#)
[wxFrameworkLayout::FirePluginEvent](#)
[wxFrameworkLayout::ForwardMouseEvent](#)
[wxFrameworkLayout::GetBarPane](#)
[wxFrameworkLayout::GetBars](#)
[wxFrameworkLayout::GetClientHeight](#)
[wxFrameworkLayout::GetClientRect](#)
[wxFrameworkLayout::GetClientWidth](#)
[wxFrameworkLayout::GetFrameClient](#)

wxFrameworkLayout

wxframelayout

browse00439

K wxFrameworkLayout

EnableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp', `classref`)")

wxFrameLayout::GetPane
wxFrameLayout::GetPaneProperties
wxFrameLayout::GetPanelsArray
wxFrameLayout::GetParentFrame
wxFrameLayout::GetPrevClientRect
wxFrameLayout::GetTopPlugin
wxFrameLayout::GetUpdatesManager
wxFrameLayout::HasTopPlugin
wxFrameLayout::HideBarWindows
wxFrameLayout::HitTestPane
wxFrameLayout::HitTestPanels
wxFrameLayout::HookUpToFrame
wxFrameLayout::InverseVisibility
wxFrameLayout::LocateBar
wxFrameLayout::OnActivate
wxFrameLayout::OnEraseBackground
wxFrameLayout::OnIdle
wxFrameLayout::OnKillFocus
wxFrameLayout::OnLButtonDown
wxFrameLayout::OnLButtonUp
wxFrameLayout::OnLDbClick
wxFrameLayout::OnMouseMove
wxFrameLayout::OnPaint
wxFrameLayout::OnRButtonDown
wxFrameLayout::OnRButtonUp
wxFrameLayout::OnSetFocus
wxFrameLayout::OnSize
wxFrameLayout::PopAllPlugins
wxFrameLayout::PopPlugin
wxFrameLayout::PositionClientWindow
wxFrameLayout::PositionPanels
wxFrameLayout::PushDefaultPlugins
wxFrameLayout::PushPlugin
wxFrameLayout::RecalcLayout
wxFrameLayout::RedockBar
wxFrameLayout::RefreshNow
wxFrameLayout::ReleaseEventsFromPane
wxFrameLayout::ReleaseEventsFromPlugin
wxFrameLayout::RemoveBar
wxFrameLayout::RemovePlugin
wxFrameLayout::ReparentWindow
wxFrameLayout::RepositionFloatedBar
wxFrameLayout::RouteMouseEvent
wxFrameLayout::SetBarState
wxFrameLayout::SetFrameClient
wxFrameLayout::SetMargins
wxFrameLayout::SetPaneBackground
wxFrameLayout::SetPaneProperties
wxFrameLayout::SetTopPlugin
wxFrameLayout::SetUpdatesManager
wxFrameLayout::ShowFloatedWindows

wxFrameLayout::UnhookFromFrame

\$#+K! **wxFrameManager**

Derived from

wxObject

Data structures

Members

wxFrameManager::wxFrameManager
wxFrameManager::~wxFrameManager
wxFrameManager::ActivateView
wxFrameManager::AddView
wxFrameManager::DeactivateCurrentView
wxFrameManager::DestroyViews
wxFrameManager::DoSerialize
wxFrameManager::EnableMenusForView
wxFrameManager::GetActiveView
wxFrameManager::GetActiveViewNo
wxFrameManager::GetActiveViewNode
wxFrameManager::GetClientWindow
wxFrameManager::GetObjectStore
wxFrameManager::GetParentFrame
wxFrameManager::GetParentWindow
wxFrameManager::GetView
wxFrameManager::GetViewNo
wxFrameManager::Init
wxFrameManager::ReloadViews
wxFrameManager::RemoveView
wxFrameManager::SaveViewsNow
wxFrameManager::SetClientWindow
wxFrameManager::SyncAllMenus
wxFrameManager::ViewsAreLoaded

^wxFrameManager

^wxframemanager

^browse00519

^K wxFrameManager

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp', `classref`)")

`##+K!` **GarbageCollector**

This class implements an extremely slow but simple garbage collection algorithm.

Derived from

No base class

Include files

<wx/fl/garbagec.h>

Data structures

Members

[GarbageCollector::GarbageCollector](#)
[GarbageCollector::~~GarbageCollector](#)
[GarbageCollector::AddDependency](#)
[GarbageCollector::AddObject](#)
[GarbageCollector::ArrangeCollection](#)
[GarbageCollector::DestroyItemList](#)
[GarbageCollector::FindItemNode](#)
[GarbageCollector::FindReferenceFreeItemNode](#)
[GarbageCollector::GetCycledObjects](#)
[GarbageCollector::GetRegularObjects](#)
[GarbageCollector::RemoveReferencesToNode](#)
[GarbageCollector::Reset](#)
[GarbageCollector::ResolveReferences](#)

^GarbageCollector

^garbagecollector

^browse00544

^K GarbageCollector

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp', `classref`)")

`$$$K!` **LayoutManagerBase**

This is a base class for layout algorithm implementations.

Derived from

No base class

Include files

<wx/fl/dyntbar.h>

Data structures

Members

[LayoutManagerBase::~~LayoutManagerBase](#)
[LayoutManagerBase::Layout](#)

`L`ayoutManagerBase

`l`ayoutmanagerbase

`b`rowse00558

`K` LayoutManagerBase

`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `classref')")

`wxNewBitmapButton`

This is an alternative class to `wxBitmapButton`. It is used in the implementation of dynamic toolbars.

Derived from

[`wxPanel`](#)

Include files

<wx/fl/newbmpbtn.h>

Data structures

Members

[`wxNewBitmapButton::wxNewBitmapButton`](#)
[`wxNewBitmapButton::~~wxNewBitmapButton`](#)
[`wxNewBitmapButton::DestroyLabels`](#)
[`wxNewBitmapButton::DrawDecorations`](#)
[`wxNewBitmapButton::DrawLabel`](#)
[`wxNewBitmapButton::DrawShade`](#)
[`wxNewBitmapButton::GetStateLabel`](#)
[`wxNewBitmapButton::IsInWindow`](#)
[`wxNewBitmapButton::OnEraseBackground`](#)
[`wxNewBitmapButton::OnKillFocus`](#)
[`wxNewBitmapButton::OnLButtonDown`](#)
[`wxNewBitmapButton::OnLButtonUp`](#)
[`wxNewBitmapButton::OnMouseMove`](#)
[`wxNewBitmapButton::OnPaint`](#)
[`wxNewBitmapButton::OnSize`](#)
[`wxNewBitmapButton::RenderAllLabelImages`](#)
[`wxNewBitmapButton::RenderLabelImage`](#)
[`wxNewBitmapButton::RenderLabelImages`](#)
[`wxNewBitmapButton::Reshape`](#)
[`wxNewBitmapButton::SetAlignments`](#)
[`wxNewBitmapButton::SetLabel`](#)

`wxNewBitmapButton`

`wxnewbitmapbutton`

`rowse00561`

`wxNewBitmapButton`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp', `classref`)"`

`$#+K!` **wxToolLayoutItem**

Tool layout item.

Derived from

[wxObject](#)

Include files

<wx/fl/dyntbar.h>

Data structures

```
typedef wxToolLayoutItem* wxToolLayoutItemPtrT
```

```
typedef wxDynToolInfo* wxDynToolInfoPtrT
```

^wxToolLayoutItem

^wxtoollayoutitem

^browse00583

^K wxToolLayoutItem

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `classref')")

`wxToolWindow`

A tool window is a special kind of frame that paints its own title, and can be used to implement small floating windows.

Derived from

[wxFrame](#)

Include files

`<wx/fl/toolwnd.h>`

Data structures

Members

[wxToolWindow::wxToolWindow](#)
[wxToolWindow::~~wxToolWindow](#)
[wxToolWindow::AddMiniButton](#)
[wxToolWindow::AdjustRectPos](#)
[wxToolWindow::CalcResizedRect](#)
[wxToolWindow::DrawHintRect](#)
[wxToolWindow::GetClient](#)
[wxToolWindow::GetMinimalWndDim](#)
[wxToolWindow::GetPreferredSize](#)
[wxToolWindow::GetScrMousePos](#)
[wxToolWindow::GetScrWindowRect](#)
[wxToolWindow::HandleTitleClick](#)
[wxToolWindow::HitTestWindow](#)
[wxToolWindow::LayoutMiniButtons](#)
[wxToolWindow::OnEraseBackground](#)
[wxToolWindow::OnLeftDown](#)
[wxToolWindow::OnLeftUp](#)
[wxToolWindow::OnMiniButtonClicked](#)
[wxToolWindow::OnMotion](#)
[wxToolWindow::OnPaint](#)
[wxToolWindow::OnSize](#)
[wxToolWindow::SetClient](#)
[wxToolWindow::SetHintCursor](#)
[wxToolWindow::SetTitleFont](#)

`wxToolWindow`

`wxtoolwindow`

`rowse00584`

`wxToolWindow`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp', `classref`)"`

Notes on using the reference

In the descriptions of the wxWindows classes and their member functions, note that descriptions of inherited member functions are not duplicated in derived classes unless their behaviour is different. So in using a class such as wxScrolledWindow, be aware that wxWindow functions may be relevant.

Note also that arguments with default values may be omitted from a function call, for brevity. Size and position arguments may usually be given a value of -1 (the default), in which case wxWindows will choose a suitable value.

Most strings are returned as wxString objects. However, for remaining char * return values, the strings are allocated and deallocated by wxWindows. Therefore, return values should always be copied for long-term use, especially since the same buffer is often used by wxWindows.

The member functions are given in alphabetical order except for constructors and destructors which appear first.

^Notes on using the reference

^referencenotes

^browse00611

^K Notes on using the reference

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `overviews)")

Event macros and identifiers

These are the event macros and event identifiers defined by FL.

Event macro	Event identifier
EVT_PL_LEFT_DOWN(func)	cbEVT_PL_LEFT_DOWN
EVT_PL_LEFT_UP(func)	cbEVT_PL_LEFT_UP
EVT_PL_RIGHT_DOWN(func)	cbEVT_PL_RIGHT_DOWN
EVT_PL_RIGHT_UP(func)	cbEVT_PL_RIGHT_UP
EVT_PL_MOTION(func)	cbEVT_PL_MOTION
EVT_PL_LEFT_DCLICK(func)	cbEVT_PL_LEFT_DCLICK
EVT_PL_LAYOUT_ROW(func)	cbEVT_PL_LAYOUT_ROW
EVT_PL_RESIZE_ROW(func)	cbEVT_PL_RESIZE_ROW
EVT_PL_LAYOUT_ROWS(func)	cbEVT_PL_LAYOUT_ROWS
EVT_PL_INSERT_BAR(func)	cbEVT_PL_INSERT_BAR
EVT_PL_RESIZE_BAR(func)	cbEVT_PL_RESIZE_BAR
EVT_PL_REMOVE_BAR(func)	cbEVT_PL_REMOVE_BAR
EVT_PL_SIZE_BAR_WND(func)	cbEVT_PL_SIZE_BAR_WND
EVT_PL_DRAW_BAR_DECOR(func)	cbEVT_PL_DRAW_BAR_DECOR
EVT_PL_DRAW_ROW_DECOR(func)	cbEVT_PL_DRAW_ROW_DECOR
EVT_PL_DRAW_PANE_DECOR(func)	cbEVT_PL_DRAW_PANE_DECOR
EVT_PL_DRAW_BAR_HANDLES(func)	cbEVT_PL_DRAW_BAR_HANDLES
EVT_PL_DRAW_ROW_HANDLES(func)	cbEVT_PL_DRAW_ROW_HANDLES
EVT_PL_DRAW_ROW_BKGROUND(func)	cbEVT_PL_DRAW_ROW_BKGROUND
EVT_PL_DRAW_PANE_BKGROUND(func)	cbEVT_PL_DRAW_PANE_BKGROUND
EVT_PL_START_BAR_DRAGGING(func)	cbEVT_PL_START_BAR_DRAGGING

Event macros and identifiers

events

rowse00612

^K Event macros and identifiers

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp', `overviews')")

EVT_PL_DRAW_HINT_RECT(func) cbEVT_PL_DRAW_HINT_RECT
EVT_PL_START_DRAW_IN_AREA(func) cbEVT_PL_START_DRAW_IN_AREA
EVT_PL_FINISH_DRAW_IN_AREA(func) cbEVT_PL_FINISH_DRAW_IN_AREA
EVT_PL_CUSTOMIZE_BAR(func) cbEVT_PL_CUSTOMIZE_BAR
EVT_PL_CUSTOMIZE_LAYOUT(func) cbEVT_PL_CUSTOMIZE_LAYOUT

See also the classes:

<u>cbCustomizeBarEvent</u>	Class for bar customization events.
<u>cbCustomizeLayoutEvent</u>	Class for layout customization events.
<u>cbDrawBarDecorEvent</u>	Class for bar decoration drawing events.
<u>cbDrawBarHandlesEvent</u>	Class for bar handles drawing events.
<u>cbDrawHintRectEvent</u>	Class for hint-rectangle drawing events.
<u>cbDrawPaneBkGroundEvent</u>	Class for pane background drawing events.
<u>cbDrawPaneDecorEvent</u>	Class for pane decoration drawing events.
<u>cbDrawRowBkGroundEvent</u>	Class for row background drawing events.
<u>cbDrawRowDecorEvent</u>	Class for row decoration drawing events.
<u>cbDrawRowHandlesEvent</u>	Class for row handles drawing events.
<u>cbFinishDrawInAreaEvent</u>	Class for finish drawing in area events.
<u>cbInsertBarEvent</u>	Class for bar insertion events.
<u>cbLayoutRowEvent</u>	Class for single row layout events.
<u>cbLayoutRowsEvent</u>	Class for multiple rows layout events.
<u>cbLeftDClickEvent</u>	Class for mouse left double click events.
<u>cbLeftDownEvent</u>	Class for mouse left down events.
<u>cbLeftUpEvent</u>	Class for mouse left up events.
<u>cbMotionEvent</u>	Class for mouse motion events.
<u>cbPluginEvent</u>	Base class for all control-bar plugin events.
<u>cbRemoveBarEvent</u>	Class for bar removal events.
<u>cbResizeBarEvent</u>	Class for bar resize events.
<u>cbResizeRowEvent</u>	Class for row resize events.
<u>cbRightDownEvent</u>	Class for mouse right down events.

cbRightUpEvent

Class for mouse right up events.

cbSizeBarWndEvent

Class for bar window resize events.

cbStartBarDraggingEvent

Class for start-bar-dragging events.

cbStartDrawInAreaEvent

Class for start drawing in area events.

FAQ

A row of all non-fixed bars don't position properly

FAQ
faq
rowse00613
FAQ
nableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp', `overviews`)")

`$#+K!` **BagLayout::Layout**

**void Layout(const wxSize& *parentDim*, wxSize& *resultingDim*,
wxLayoutItemArrayT& *items*, int *horizGap*, int *vertGap*)**^K

Constructor.

^BagLayout::Layout
^baglayoutlayout
^browse00010
^K BagLayout Layout
^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `baglayout')")
^K Layout

`$#+K! wxBarIterator::wxBarIterator`

`wxBarIterator(RowArrayT& rows)K`

Constructor, taking row array.

`wxBarIterator::wxBarIterator`

`wxbariteratorwxbariterator`

`rowse00012`

`K wxBarIterator wxBarIterator`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxbariterator')")`

`K wxBarIterator`

wxBarIterator::BarInfo

cbBarInfo& BarInfo()

Gets the current bar information.

wxBarIterator::BarInfo

wxbariteratorbarinfo

rowse00013

wxBarIterator BarInfo

enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp',`wxbariterator')")

BarInfo

\$#+K! **wxBarLayout::Next**

bool Next()^K

Advances the iterator and returns TRUE if a bar is available.

wxBarLayout::Next

wxbariteratornext

browse00014

K wxBarLayout Next

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`wxbariterator')")

K Next

wxBarIterator::Reset

void Reset()

Resets the iterator to the start of the first row.

wxBarIterator::Reset

wxbariteratorreset

rowse00015

wxBarIterator Reset

enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `wxbariterator')")

Reset

`wxBarIterator::RowInfo`

`cbRowInfo& RowInfo()`

Returns a reference to the currently traversed row.

`wxBarIterator::RowInfo`

`wxbariteratorrowinfo`

`rowse00016`

`wxBarIterator RowInfo`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `wxbariterator')")`

`RowInfo`

cbAntiflickerPlugin::cbAntiflickerPlugin

cbAntiflickerPlugin()^K

Default constructor.

cbAntiflickerPlugin(wxFrameLayout* pPanel, int paneMask = wxALL_PANES)^K

Constructor taking frame layout panel, and pane mask.

```
cbAntiflickerPlugin::cbAntiflickerPlugin
cbantiflickerplugincbantiflickerplugin
browse00018
K cbAntiflickerPlugin cbAntiflickerPlugin
EnableButton("Up");ChangeButtonBinding("Up", "JumpId( fl.hlp',
`cbantiflickerplugin')")
K cbAntiflickerPlugin
K cbAntiflickerPlugin
```

cbAntiflickerPlugin::~cbAntiflickerPlugin

~cbAntiflickerPlugin()^K

Destructor.

cbAntiflickerPlugin::~cbAntiflickerPlugin

cbantiflickerplugindtor

rowse00019

cbAntiflickerPlugin ~cbAntiflickerPlugin

enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',

`cbantiflickerplugin')

~cbAntiflickerPlugin

`$#+K!cbAntiflickerPlugin::AllocNewBuffer`

`wxDC* AllocNewBuffer(const wxRect& forArea)K`

Allocates a suitable buffer.

`^bAntiflickerPlugin::AllocNewBuffer`

`^bantiflickerpluginallocnewbuffer`

`^rowse00020`

`K cbAntiflickerPlugin AllocNewBuffer`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`

``cbantiflickerplugin')")`

`K AllocNewBuffer`

`cbAntiflickerPlugin::FindSuitableBuffer`

`wxDC* FindSuitableBuffer(const wxRect& forArea)`^K

Finds a suitable buffer. Returns NULL if a suitable buffer is not present.

^c`bAntiflickerPlugin::FindSuitableBuffer`

^c`bantiflickerpluginfindsuitablebuffer`

^b`rowse00021`

^K `cbAntiflickerPlugin FindSuitableBuffer`

^E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`

[`]`cbantiflickerplugin')`

^K `FindSuitableBuffer`

`$#+K!cbAntiflickerPlugin::GetClientDC`

`wxDC& GetClientDC()`^K

Gets the client device context.

`^bAntiflickerPlugin::GetClientDC`

`^bantiflickerpluggingetclientdc`

`^rowse00022`

`^K cbAntiflickerPlugin GetClientDC`

`^enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
^cbantiflickerplugin')")`

`^K GetClientDC`

`$#+K!cbAntiflickerPlugin::GetWindowDC`

`wxDC& GetWindowDC()`^K

Gets the window device context.

`^bAntiflickerPlugin::GetWindowDC`

`^bantiflickerpluggingetwindowdc`

`^rowse00023`

`^K cbAntiflickerPlugin GetWindowDC`

`^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
^bantiflickerplugin')")`

`^K GetWindowDC`

`cbAntiflickerPlugin::OnFinishDrawInArea`

`void OnFinishDrawInArea(cbFinishDrawInAreaEvent& event)`^K

Handler for plugin event.

`cbAntiflickerPlugin::OnFinishDrawInArea`

`cbantiflickerpluginonfinishdrawinarea`

`rowse00024`

`cbAntiflickerPlugin OnFinishDrawInArea`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`

``cbantiflickerplugin')")`

`OnFinishDrawInArea`

`cbAntiflickerPlugin::OnStartDrawInArea`

`void OnStartDrawInArea(cbStartDrawInAreaEvent& event)`^K

Handler for plugin event.

`cbAntiflickerPlugin::OnStartDrawInArea`

`cbantiflickerpluginonstartdrawinarea`

`rowse00025`

`cbAntiflickerPlugin OnStartDrawInArea`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp',`

``cbantiflickerplugin')")`

`OnStartDrawInArea`

cbBarDimHandlerBase::cbBarDimHandlerBase

cbBarDimHandlerBase()^K

to multiple bars, it's instance is reference-counted Default constructor. The initial reference count is 0, since the handler is not used until the first invocation of AddRef().

^cbBarDimHandlerBase::cbBarDimHandlerBase
^cbbardimhandlerbasecbbardimhandlerbase
^browse00027
^K cbBarDimHandlerBase cbBarDimHandlerBase
^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbbardimhandlerbase')")
^K cbBarDimHandlerBase

cbBarDimHandlerBase::AddRef

void AddRef()

Increments the reference count.

cbBarDimHandlerBase::AddRef

cbbardimhandlerbaseaddref

rowse00028

cbBarDimHandlerBase AddRef

**enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbbardimhandlerbase')")**

AddRef

^{\$#+K!}**cbBarDimHandlerBase::OnChangeBarState**

void OnChangeBarState(cbBarInfo* *pBar*, int *newState*)^K

Responds to "bar-state-changes" notifications.

[^]bBarDimHandlerBase::OnChangeBarState

[^]bbardimhandlerbaseonchangebarstate

^browse00029

^K cbBarDimHandlerBase OnChangeBarState

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbbardimhandlerbase')")

^K OnChangeBarState

cbBarDimHandlerBase::OnResizeBar

void OnResizeBar(cbBarInfo* *pBar*, const wxSize& *given*, wxSize& *preferred*)^K

Responds to bar resize notifications.

^cbBarDimHandlerBase::OnResizeBar

^cbbardimhandlerbaseonresizebar

^browse00030

^K cbBarDimHandlerBase OnResizeBar

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbbardimhandlerbase')")

^K OnResizeBar

cbBarDimHandlerBase::RemoveRef

void RemoveRef()

Decrements the reference count, and if the count is at zero, delete 'this'.

```
cbBarDimHandlerBase::RemoveRef
cbbardimhandlerbaseremoveref
rowse00031
cbBarDimHandlerBase RemoveRef
enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbbardimhandlerbase')")
RemoveRef
```

`$#+K!cbBarDragPlugin::cbBarDragPlugin`

`cbBarDragPlugin()`^K

Default constructor.

`cbBarDragPlugin(wxFrameLayout* pPanel, int paneMask = wxALL_PANES)`^K

Constructor taking a parent frame, and flag. See `cbPluginBase`.

`cbBarDragPlugin::cbBarDragPlugin`

`cbbardragplugincbbardragplugin`

`rowse00033`

`cbBarDragPlugin cbBarDragPlugin`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp',`cbbardragplugin')")`

`cbBarDragPlugin`

`cbBarDragPlugin`

`$#+K!cbBarDragPlugin::~~cbBarDragPlugin`

`~cbBarDragPlugin()`^K

Destructor.

`^bBarDragPlugin::~~cbBarDragPlugin`

`^bbardragplugindtor`

`^rowse00034`

`KcbBarDragPlugin ~cbBarDragPlugin`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardragplugin')")`

`K ~cbBarDragPlugin`

`cbBarDragPlugin::AdjustHintRect`

`void AdjustHintRect(wxPoint& mousePos)`^K

the thicker rectangle is drawn using hatched brush, the default border width for this rectangle is 8 pix. Internal implementation function.

^cbBarDragPlugin::AdjustHintRect

^cbbardragpluginadjusthintrect

^browse00035

^K cbBarDragPlugin AdjustHintRect

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbbardragplugin')")

^K AdjustHintRect

`$#+K!cbBarDragPlugin::CalcOnScreenDims`

`void CalcOnScreenDims(wxRect& rect)K`

Internal implementation function.

`cbBarDragPlugin::CalcOnScreenDims`

`cbbardragplugincalconscreendims`

`browse00036`

`KcbBarDragPlugin CalcOnScreenDims`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp',`cbbardragplugin')")`

`KCalcOnScreenDims`

`$#+K!cbBarDragPlugin::ClipPosInFrame`

`void ClipPosInFrame(wxPoint& pos)K`

Internal implementation function.

`cbBarDragPlugin::ClipPosInFrame`

`cbbardragpluginclipposinframe`

`browse00037`

`KcbBarDragPlugin ClipPosInFrame`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp',`cbbardragplugin')")`

`KClipPosInFrame`

`cbBarDragPlugin::ClipRectInFrame`

`void ClipRectInFrame(wxRect& rect)`^K

Internal implementation function.

`cbBarDragPlugin::ClipRectInFrame`

`cbbardragplugincliprectinframe`

`rowse00038`

`cbBarDragPlugin ClipRectInFrame`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp',`cbbardragplugin')")`

`ClipRectInFrame`

`cbBarDragPlugin::DoDrawHintRect`

`void DoDrawHintRect(wxRect& rect, bool isInClientRect)`^K

Internal implementation function. Draw the hint rectangle.

`cbBarDragPlugin::DoDrawHintRect`

`cbbardragplugindodrawhintrect`

`rowse00039`

`cbBarDragPlugin DoDrawHintRect`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp',`cbbardragplugin')")`

`DoDrawHintRect`

`cbBarDragPlugin::DrawHintRect`

`void DrawHintRect(wxRect& rect, bool isInClientRect)`^K

Internal implementation function. Draw the visual hint while dragging.

^cbBarDragPlugin::DrawHintRect

^cbbardragplugindrawhintrect

^browse00040

^K cbBarDragPlugin DrawHintRect

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbbardragplugin')")

^K DrawHintRect

`cbBarDragPlugin::EraseHintRect`

`void EraseHintRect(wxRect& rect, bool isInClientRect)`^K

Internal implementation function. Erase the visual hint while dragging.

`cbBarDragPlugin::EraseHintRect`

`cbbardragpluginerasehintrect`

`rowse00041`

`cbBarDragPlugin EraseHintRect`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `cbbardragplugin')")`

`EraseHintRect`

cbBarDragPlugin::FinishTracking

void FinishTracking()

Internal implementation function. Stop showing the visual hint while dragging.

cbBarDragPlugin::FinishTracking

cbbardragpluginfinishtracking

rowse00042

cbBarDragPlugin FinishTracking

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp',`cbbardragplugin')")

FinishTracking

`cbBarDragPlugin::GetBarHeightInPane`

`int GetBarHeightInPane(cbDockPane* pPane)`^K

Internal implementation function.

`cbBarDragPlugin::GetBarHeightInPane`

`cbbardragplugingetbarheightinpane`

`rowse00043`

^K `cbBarDragPlugin GetBarHeightInPane`

^E `enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`cbbardragplugin')")`

^K `GetBarHeightInPane`

`$#+K!cbBarDragPlugin::GetBarWidthInPane`

`int GetBarWidthInPane(cbDockPane* pPane)K`

Internal implementation function.

`^bBarDragPlugin::GetBarWidthInPane`

`^bbardragplugingetbarwidthinpane`

`^rowse00044`

`KcbBarDragPlugin GetBarWidthInPane`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`cbbardragplugin')")`

`KGetBarWidthInPane`

`$#+K!cbBarDragPlugin::GetDistanceToPane`

`int GetDistanceToPane(cbDockPane* pPane, wxPoint& mousePos)`^K

Internal implementation function.

`^bBarDragPlugin::GetDistanceToPane`

`^bbardragplugingetdistancetopane`

`^rowse00045`

`^K cbBarDragPlugin GetDistanceToPane`

`^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardragplugin')")`

`^K GetDistanceToPane`

`$#+K!cbBarDragPlugin::HitTestPanes`

`cbDockPane* HitTestPanes(wxRect& rect)K`

Internal implementation function. Finds the pane under the specified rectangle.

`cbDockPane* HitTestPanes(wxPoint& pos)K`

Internal implementation function. Finds the pane under the specified point.

`^bBarDragPlugin::HitTestPanes`

`^bbardragpluginhittestpanes`

`^rowse00046`

`^K cbBarDragPlugin HitTestPanes`

`^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardragplugin')")`

`^K HitTestPanes`

`^K HitTestPanes`

`$$$K!cbBarDragPlugin::HitsPane`

`bool HitsPane(cbDockPane* pPane, wxRect& rect)K`

Internal implementation function.

`^bBarDragPlugin::HitsPane`

`^bbardragpluginhitspane`

`^rowse00047`

`KcbBarDragPlugin HitsPane`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^cbbardragplugin')")`

`KHitsPane`

`$#+K!cbBarDragPlugin::IsInClientArea`

`bool IsInClientArea(wxPoint& mousePos)K`

Internal implementation function.

`bool IsInClientArea(wxRect& rect)K`

Internal implementation function.

`^bBarDragPlugin::IsInClientArea`

`^bbardragpluginisinclientarea`

`^rowse00048`

`^K cbBarDragPlugin IsInClientArea`

`^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardragplugin')")`

`^K IsInClientArea`

`^K IsInClientArea`

`$#+K!` **cbBarDragPlugin::IsInOtherPane**

bool IsInOtherPane(wxPoint& *mousePos*)^K

Internal implementation function.

`^bBarDragPlugin::IsInOtherPane`

`^bbardragpluginisinotherpane`

`^rowse00049`

`^K cbBarDragPlugin IsInOtherPane`

`^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardragplugin')")`

`^K IsInOtherPane`

`cbBarDragPlugin::OnDrawHintRect`

`void OnDrawHintRect(cbDrawHintRectEvent& event)`^K

Handles event, which originates from itself.

`cbBarDragPlugin::OnDrawHintRect`

`cbbardragpluginondrawhintrect`

`rowse00050`

`cbBarDragPlugin OnDrawHintRect`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`cbbardragplugin')")`

`OnDrawHintRect`

cbBarDragPlugin::OnLButtonDown

void OnLButtonDown(cbLeftDownEvent& event)^K

Handler for plugin event.

^cbBarDragPlugin::OnLButtonDown

^cbbardragpluginonlbuttondown

^browse00051

^K cbBarDragPlugin OnLButtonDown

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbbardragplugin')")

^K OnLButtonDown

cbBarDragPlugin::OnLButtonUp

void OnLButtonUp(cbLeftUpEvent& event)

Handler for plugin event.

cbBarDragPlugin::OnLButtonUp

cbbardragpluginonlbuttonup

rowse00052

cbBarDragPlugin OnLButtonUp

enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp',`cbbardragplugin')")

OnLButtonUp

`cbBarDragPlugin::OnLDbClick`

`void OnLDbClick(cbLeftDClickEvent& event)`^K

Handler for plugin event.

`cbBarDragPlugin::OnLDbClick`

`cbbardragpluginonldbclick`

`rowse00053`

`cbBarDragPlugin OnLDbClick`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `cbbardragplugin')")`

`OnLDbClick`

`cbBarDragPlugin::OnMouseMove`

`void OnMouseMove(cbMotionEvent& event)`^K

Handler for plugin event.

`cbBarDragPlugin::OnMouseMove`

`cbbardragpluginonmousemove`

`rowse00054`

`cbBarDragPlugin OnMouseMove`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `cbbardragplugin')")`

`OnMouseMove`

`$#+K!cbBarDragPlugin::OnStartBarDragging`

`void OnStartBarDragging(cbStartBarDraggingEvent& event)K`

Handler for plugin event.

`^bBarDragPlugin::OnStartBarDragging`

`^bbardragpluginonstartbardragging`

`^rowse00055`

`KcbBarDragPlugin OnStartBarDragging`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardragplugin')")`

`K OnStartBarDragging`

`cbBarDragPlugin::RectToScr`

`void RectToScr(wxRect& frameRect, wxRect& scrRect)`^K

Internal implementation function. Converts the given rectangle from window to screen coordinates.

`cbBarDragPlugin::RectToScr`

`cbbardragpluginrecttoscr`

`rowse00056`

`cbBarDragPlugin RectToScr`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `cbbardragplugin')")`

`RectToScr`

`cbBarDragPlugin::ShowHint`

`void ShowHint(bool prevWasInClient)`^K

Internal implementation function. Show the hint; called within OnMouseMove.

`cbBarDragPlugin::ShowHint`

`cbbardragpluginshowhint`

`rowse00057`

`cbBarDragPlugin ShowHint`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp',`cbbardragplugin')")`

`ShowHint`

`$#+K!cbBarDragPlugin::StartTracking`

`void StartTracking()`^K

on-screen hint-tracking related methods Internal implementation function. Start showing a visual hint while dragging.

`^bBarDragPlugin::StartTracking`

`^bbardragpluginstarttracking`

`^rowse00058`

`^cbBarDragPlugin StartTracking`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardragplugin')")`

`^ StartTracking`

`$#+K!cbBarDragPlugin::StickToPane`

`void StickToPane(cbDockPane* pPane, wxPoint& mousePos)K`

Internal implementation function.

`cbBarDragPlugin::StickToPane`

`cbbardragpluginsticktopane`

`browse00059`

`KcbBarDragPlugin StickToPane`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp',`cbbardragplugin')")`

`KStickToPane`

`$#+K!cbBarDragPlugin::UnstickFromPane`

`void UnstickFromPane(cbDockPane* pPane, wxPoint& mousePos)K`

Internal implementation function.

`cbBarDragPlugin::UnstickFromPane`

`cbbardragpluginunstickfrompane`

`browse00060`

`KcbBarDragPlugin UnstickFromPane`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp',`cbbardragplugin')")`

`KUnstickFromPane`

cbBarHintsPlugin::cbBarHintsPlugin

cbBarHintsPlugin()^K

Default constructor.

cbBarHintsPlugin(wxFrameLayout* pLayout, int paneMask = wxALL_PANES)^K

Constructor, taking parent frame and pane mask flag.

^cbBarHintsPlugin::cbBarHintsPlugin

^cbbarhintsplugin**cbbarhintsplugin**

^browse00062

^K cbBarHintsPlugin cbBarHintsPlugin

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`cbbarhintsplugin')")

^K cbBarHintsPlugin

^K cbBarHintsPlugin

`$#+K!cbBarHintsPlugin::~~cbBarHintsPlugin`

`~cbBarHintsPlugin()`^K

Destructor.

`^bBarHintsPlugin::~~cbBarHintsPlugin`

`^bbarhintsplugindtor`

`^rowse00063`

^K `cbBarHintsPlugin ~cbBarHintsPlugin`

^E `nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbbarhintsplugin')")`

^K `~cbBarHintsPlugin`

`$#+K!cbBarHintsPlugin::CreateBoxes`

`void CreateBoxes()`^K

Helper function: creates close and collapse boxes.

^cbBarHintsPlugin::CreateBoxes

^cbbarhintsplugincreateboxes

^browse00064

^K cbBarHintsPlugin CreateBoxes

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbbbarhintsplugin')")

^K CreateBoxes

`$#+K!cbBarHintsPlugin::DoDrawHint`

`void DoDrawHint(wxDC& dc, wxRect& rect, int pos, int boxOfs, int grooveOfs, bool isFixed)K`

Helper function: draws a hint.

`cbBarHintsPlugin::DoDrawHint`

`cbbarhintsplugindodrawhint`

`browse00065`

`KcbBarHintsPlugin DoDrawHint`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp',`cbbbarhintsplugin')")`

`KDoDrawHint`

`$#+K!cbBarHintsPlugin::Draw3DBox`

`void Draw3DBox(wxDC& dc, const wxPoint& pos, bool pressed)K`

Helper function: draws a 3D box.

`cbBarHintsPlugin::Draw3DBox`

`cbbarhintsplugindraw3dbox`

`browse00066`

`KcbBarHintsPlugin Draw3DBox`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbbbarhintsplugin')")`

`KDraw3DBox`

^{\$#+K!}**cbBarHintsPlugin::DrawCloseBox**

void DrawCloseBox(wxDC& *dc*, **const wxPoint&** *pos*, **bool** *pressed*)^K

Helper function: draws a close box.

^cbBarHintsPlugin::DrawCloseBox

^cbbarhintsplugindrawclosebox

^browse00067

^K cbBarHintsPlugin DrawCloseBox

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbbbarhintsplugin')")

^K DrawCloseBox

`$#+K!cbBarHintsPlugin::DrawCollapseBox`

`void DrawCollapseBox(wxDC& dc, const wxPoint& pos, bool atLeft, bool disabled, bool pressed)K`

Helper function: draws a collapse box.

`cbBarHintsPlugin::DrawCollapseBox`

`cbbarhintsplugindrawcollapsebox`

`browse00068`

`KcbBarHintsPlugin DrawCollapseBox`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbbbarhintsplugin')")`

`K DrawCollapseBox`

`cbBarHintsPlugin::DrawGrooves`

`void DrawGrooves(wxDC& dc, const wxPoint& pos, int length)`^K

Helper function: draws grooves.

`cbBarHintsPlugin::DrawGrooves`

`cbbarhintsplugindrawgrooves`

`rowse00069`

`cbBarHintsPlugin DrawGrooves`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbbarhintsplugin')")`

`DrawGrooves`

^{\$#+K!}**cbBarHintsPlugin::ExcludeHints**

void ExcludeHints(**wxRect&** *rect*, **cbBarInfo&** *info*)^K

Helper function.

^cbBarHintsPlugin::ExcludeHints

^cbbarhintspluginexcludehints

^browse00070

^K cbBarHintsPlugin ExcludeHints

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbbbarhintsplugin')")

^K ExcludeHints

`$#+K!cbBarHintsPlugin::GetHintsLayout`

`void GetHintsLayout(wxRect& rect, cbBarInfo& info, int& boxOfs, int& grooveOfs, int& pos)K`

Helper function: gets the layout of a hint.

`cbBarHintsPlugin::GetHintsLayout`

`cbbarhintsplugingethintslayout`

`browse00071`

`KcbBarHintsPlugin GetHintsLayout`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbbbarhintsplugin')")`

`KGetHintsLayout`

`$#+K!` **cbBarHintsPlugin::HitTestHints**

int HitTestHints(cbBarInfo& *info*, const wxPoint& *pos*)^K

Helper function: returns information about the hint under the given position.

`^bBarHintsPlugin::HitTestHints`

`^bbarhintspluginhittesthints`

`^rowse00072`

`^K cbBarHintsPlugin HitTestHints`

`^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^cbarhintsplugin')")`

`^K HitTestHints`

^{\$#+K!}**cbBarHintsPlugin::OnDrawBarDecorations**

void OnDrawBarDecorations(cbDrawBarDecorEvent& *event*)^K

Handles a plugin event.

^cbBarHintsPlugin::OnDrawBarDecorations

^cbbarhintspluginondrawbardecorations

^browse00073

^K cbBarHintsPlugin OnDrawBarDecorations

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbbbarhintsplugin')")

^K OnDrawBarDecorations

cbBarHintsPlugin::OnInitPlugin

void OnInitPlugin()

Called to initialize this plugin.

cbBarHintsPlugin::OnInitPlugin

cbbarhintspluginoninitplugin

rowse00074

cbBarHintsPlugin OnInitPlugin

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp',`cbbarhintsplugin')")

OnInitPlugin

`cbBarHintsPlugin::OnLeftDown`

`void OnLeftDown(cbLeftDownEvent& event)`^K

Handles a plugin event.

`cbBarHintsPlugin::OnLeftDown`

`cbbarhintspluginonleftdown`

`rowse00075`

`cbBarHintsPlugin OnLeftDown`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbbarhintsplugin')")`

`OnLeftDown`

`cbBarHintsPlugin::OnLeftUp`

`void OnLeftUp(cbLeftUpEvent& event)`^K

Handles a plugin event.

`cbBarHintsPlugin::OnLeftUp`

`cbbarhintspluginonleftup`

`rowse00076`

`cbBarHintsPlugin OnLeftUp`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp',`cbbarhintsplugin')")`

`OnLeftUp`

`cbBarHintsPlugin::OnMotion`

`void OnMotion(cbMotionEvent& event)`^K

Handles a plugin event.

`cbBarHintsPlugin::OnMotion`

`cbbarhintspluginonmotion`

`rowse00077`

`cbBarHintsPlugin OnMotion`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp',`cbbarhintsplugin')")`

`OnMotion`

`$#+K!cbBarHintsPlugin::OnSizeBarWindow`

`void OnSizeBarWindow(cbSizeBarWndEvent& event)K`

Handles a plugin event.

`cbBarHintsPlugin::OnSizeBarWindow`

`cbbarhintspluginonsizebarwindow`

`browse00078`

`KcbBarHintsPlugin OnSizeBarWindow`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`cbbbarhintsplugin')")`

`KOnSizeBarWindow`

`$#+K!cbBarHintsPlugin::SetGrooveCount`

`void SetGrooveCount(int nGrooves)K`

Set the number of grooves to be shown in the pane.

`cbBarHintsPlugin::SetGrooveCount`

`cbbarhintspluginsetgroovecount`

`browse00079`

`KcbBarHintsPlugin SetGrooveCount`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbbbarhintsplugin')")`

`KSetGrooveCount`

cbBarInfo::cbBarInfo

cbBarInfo()

Constructor.

cbBarInfo::cbBarInfo

cbBarInfo::cbBarInfo

cbBarInfo::cbBarInfo

cbBarInfo::cbBarInfo

cbBarInfo::cbBarInfo

cbBarInfo

cbBarInfo::~~cbBarInfo

~cbBarInfo()^K

Destructor.

cbBarInfo::~~cbBarInfo

cbbarinfodtor

rowse00082

cbBarInfo ~cbBarInfo

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbbarinfo')")

cbBarInfo

`$#+KK!cbBarInfo::IsExpanded`

`bool IsExpanded() const`

Returns TRUE if this bar is expanded.

`cbBarInfo::IsExpanded`

`cbbarinfoisexpanded`

`browse00083`

`K cbBarInfo IsExpanded`

`K IsExpanded`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbbarinfo')")`

`$#+KK!cbBarInfo::IsFixed`

`bool IsFixed() const`

Returns TRUE if this bar is fixed.

`cbBarInfo::IsFixed`

`cbbarinfoisfixed`

`browse00084`

`K cbBarInfo IsFixed`

`K IsFixed`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbbarinfo')")`

cbBarSpy::cbBarSpy

cbBarSpy(wxFrameLayout* *pPanel*)^K

Constructor, taking a parent pane.

cbBarSpy()^K

Default constructor.

cbBarSpy::cbBarSpy

cbbarspycbbarspy

rowse00087

cbBarSpy cbBarSpy

enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbarspy')")

cbBarSpy

cbBarSpy

`cbBarSpy::ProcessEvent`

`bool ProcessEvent(wxEvent& event)`

Performs special event processing.

`cbBarSpy::ProcessEvent`

`cbbarspyprocessevent`

`rowse00088`

`cbBarSpy ProcessEvent`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `cbbarspy')")`

`ProcessEvent`

cbBarSpy::SetBarWindow

void SetBarWindow(wxWindow* pWnd)

Sets the bar window.

cbBarSpy::SetBarWindow

cbbarspysetbarwindow

rowse00089

cbBarSpy SetBarWindow

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `cbbarspy')")

SetBarWindow

`$#+K!cbCloseBox::Draw`

`void Draw(wxDC& dc)K`

Draws the close button appearance.

`cbCloseBox::Draw`

`cbcloseboxdraw`

`browse00091`

`KcbCloseBox Draw`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`cbclosebox')")`

`KDraw`

cbCollapseBox::Draw

void Draw(wxDC& dc)

Draws the collapse button appearance.

cbCollapseBox::Draw

cbcollapseboxdraw

rowse00093

cbCollapseBox Draw

enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `cbcollapsebox')")

Draw

cbCustomizeBarEvent::cbCustomizeBarEvent

cbCustomizeBarEvent(**cbBarInfo*** *pBar*, **const wxPoint&** *clickPos*, **cbDockPane***
pPane)^K

Constructor, taking bar information, mouse position, and pane.

^QbCustomizeBarEvent::cbCustomizeBarEvent
^Qbcustomizebareventcbcustomizebarevent
^browse00096
^K cbCustomizeBarEvent cbCustomizeBarEvent
^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`bcustomizebarevent')")
^K cbCustomizeBarEvent

`cbCustomizeLayoutEvent::cbCustomizeLayoutEvent`

`cbCustomizeLayoutEvent(const wxPoint& clickPos)`^K

Constructor, taking mouse position.

`cbCustomizeLayoutEvent::cbCustomizeLayoutEvent`

`cbcustomizelayouteventcbcustomizelayoutevent`

`rowse00098`

`cbCustomizeLayoutEvent cbCustomizeLayoutEvent`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbcustomizelayoutevent')")`

`cbCustomizeLayoutEvent`

`$#+K!cbDimInfo::cbDimInfo`

`cbDimInfo(cbBarDimHandlerBase* pDimHandler, bool isFixed)`^K

Constructor. `isFixed` is `TRUE` if vertical/horizontal dimensions cannot be manually adjusted by the user using resizing handles. If `FALSE`, the frame-layout automatically places resizing handles among bars that do are not fixed.

`cbDimInfo(int dh_x, int dh_y, int dv_x, int dv_y, int f_x, int f_y, bool isFixed = TRUE, int horizGap = 6, int vertGap = 6, cbBarDimHandlerBase* pDimHandler = NULL)`^K

Constructor taking dimension information. `dh_x`, `dh_y` are the dimensions when docked horizontally. `dv_x`, `dv_y` are the dimensions when docked vertically. `f_x`, `f_y` are the dimensions when floating. For information on `isFixed`, see comments above. `horizGap` is the left/right gap, separating decorations from the bar's actual window, filled with the frame's background colour. The dimension is given in the frame's coordinates. `vertGap` is the top/bottom gap, separating decorations from the bar's actual window, filled with the frame's background colour. The dimension is given in the frame's coordinates.

`cbDimInfo(int x, int y, bool isFixed = TRUE, int gap = 6, cbBarDimHandlerBase* pDimHandler = NULL)`^K

Constructor.

`cbDimInfo()`^K

Default constructor.

`^bDimInfo::cbDimInfo`

`^bdiminforcbdiminfo`

`^rowse00100`

^K `cbDimInfo cbDimInfo`

^E `enableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp', `cbdinfo)")`

^K `cbDimInfo`

^K `cbDimInfo`

^K `cbDimInfo`

^K `cbDimInfo`

`$#+K!cbDimInfo::~~cbDimInfo`

`~cbDimInfo()`^K

Destructor. Destroys handler automatically, if present.

`cbDimInfo::~~cbDimInfo`

`cbdiminfodtor`

`rowse00101`

^K `cbDimInfo ~cbDimInfo`

^E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdiminfo'))`

^K `~cbDimInfo`

\$#+K! **cbDimInfo::GetDimHandler**

cbBarDimHandlerBase* GetDimHandler()^K

Returns the handler, if any.

^bDimInfo::GetDimHandler

^bdiminfogetdimhandler

^rowse00102

K cbDimInfo GetDimHandler

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdiminfo')")

K GetDimHandler

cbDimInfo::operator=

const cbDimInfo& operator operator=(const cbDimInfo& *other*)^K

Assignment operator.

cbDimInfo::operator=

cbdiminfooperatorassign

rowse00103

cbDimInfo operator=

enableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdiminfo'))

operator=

cbDockBox::Draw

void Draw(wxDC& dc)

Draws the dock button appearance.

cbDockBox::Draw

cbdockboxdraw

rowse00105

cbDockBox Draw

enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `cbdockbox')")

Draw

cbDockPane::cbDockPane

cbDockPane(int *alignment*, wxFrameLayout* *pPanel*)^K

Constructor, taking alignment and layout panel.

cbDockPane()^K

public members Default constructor.

^cbDockPane::cbDockPane

^cbdockpanecbdockpane

^browse00107

^K cbDockPane cbDockPane

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdockpane')")

^K cbDockPane

^K cbDockPane

`$#+K!cbDockPane::~cbDockPane`

`~cbDockPane()`^K

Destructor.

`cbDockPane::~cbDockPane`

`cbdockpaneditor`

`rowse00108`

`K cbDockPane ~cbDockPane`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdockpane')")`

`K ~cbDockPane`

`$#+K!cbDockPane::BarPresent`

`bool BarPresent(cbBarInfo* pBar)K`

Returns TRUE if the given bar is present in this pane.

`^bDockPane::BarPresent`

`^bdockpanebarpresent`

`^rowse00109`

`KcbDockPane BarPresent`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^cbdockpane')")`

`KBarPresent`

`$#+K!cbDockPane::CalcLengthRatios`

`void CalcLengthRatios(cbRowInfo* pInRow)K`

Calculate lengths. Internal function called by plugins.

`cbDockPane::CalcLengthRatios`

`cbdockpanecalclengthratios`

`browse00110`

`KcbDockPane CalcLengthRatios`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`,`cbdockpane`))`

`KCalcLengthRatios`

`cbDockPane::ContractBar`

`void ContractBar(cbBarInfo* pBar)`^K

Contracts the bar. Internal function called by plugins.

`cbDockPane::ContractBar`

`cbdockpanecontractbar`

`rowse00111`

`cbDockPane ContractBar`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `cbdockpane`))`

`ContractBar`

cbDockPane::DoInsertBar

void DoInsertBar(cbBarInfo* *pBar*, int *rowNo*)

Inserts the bar at the given row number. Internal function called by plugins.

cbDockPane::DoInsertBar

cbdockpanedoininsertbar

rowse00112

cbDockPane DoInsertBar

enableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`,`cbdockpane`))

DoInsertBar

\$#+K! **cbDockPane::DrawHorizHandle**

void DrawHorizHandle(wxDC& *dc*, int *x*, int *y*, int *width*)^K

Row/bar resizing related helper-method.

^bDockPane::DrawHorizHandle

^bdockpanedrawhorizhandle

browse00113

K cbDockPane DrawHorizHandle

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdockpane')")

K DrawHorizHandle

`$#+K!cbDockPane::DrawVertHandle`

`void DrawVertHandle(wxDC& dc, int x, int y, int height)K`

protected really (accessed only by plugins) Row/bar resizing related helper-method.

`cbDockPane::DrawVertHandle`

`cbdockpanedrawverthandle`

`rowse00114`

`KcbDockPane DrawVertHandle`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")`

`K DrawVertHandle`

`$#+K!cbDockPane::ExpandBar`

`void ExpandBar(cbBarInfo* pBar)K`

Expands the bar. Internal function called by plugins.

`^bDockPane::ExpandBar`

`^bdockpaneexpandbar`

`^rowse00115`

`^cbDockPane ExpandBar`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdockpane')")`

`^ ExpandBar`

`cbDockPane::FinishDrawInArea`

`void FinishDrawInArea(const wxRect& area)`^K

Generates `cbFinishDrawInAreaEvent` and sends it to the layout. Internal function called by plugins.

`cbDockPane::FinishDrawInArea`

`cbdockpanefinishdrawinarea`

`rowse00116`

`cbDockPane FinishDrawInArea`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `cbdockpane')")`

`FinishDrawInArea`

`cbDockPane::FrameToPane`

`void FrameToPane(int* x, int* y)`^K

Coordinate translation between parent's frame and this pane. Internal function called by plugins.

`void FrameToPane(wxRect* pRect)`^K

Coordinate translation between parent's frame and this pane. Internal function called by plugins.

`cbDockPane::FrameToPane`

`cbdockpaneframetopane`

`rowse00117`

`cbDockPane FrameToPane`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `cbdockpane')")`

`FrameToPane`

`FrameToPane`

cbDockPane::GetAlignment

int GetAlignment()^K

Returns the alignment for this pane. The value is one of FL_ALIGN_TOP, FL_ALIGN_BOTTOM, FL_ALIGN_LEFT, FL_ALIGN_RIGHT.

^cbDockPane::GetAlignment

^cbdockpanegetalignment

^browse00118

^K cbDockPane GetAlignment

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")

^K GetAlignment

^{\$#+K!}**cbDockPane::GetBarInfoByWindow**

cbBarInfo* **GetBarInfoByWindow**(**wxWindow*** *pBarWnd*)^K

Finds the bar information by corresponding window.

^cbDockPane::GetBarInfoByWindow

^cbdockpanegetbarinfobywindow

^browse00119

^K cbDockPane GetBarInfoByWindow

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")

^K GetBarInfoByWindow

`cbDockPane::GetBarResizeRange`

`void GetBarResizeRange(cbBarInfo* pBar, int* from, int* till, bool forLeftHandle)`^K

Returns the bar's resize range.

`cbDockPane::GetBarResizeRange`

`cbdockpanegetbarresizerange`

`rowse00120`

^K `cbDockPane GetBarResizeRange`

^E `nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdockpane')")`

^K `GetBarResizeRange`

`$#+K!cbDockPane::GetDockingState`

`int GetDockingState()`^K

Returns wxCBAR_DOCKED_HORIZONTALLY if the alignment is top or bottom, or wxCBAR_DOCKED_VERTICALLY otherwise.

`^bDockPane::GetDockingState`

`^bdockpanegetdockingstate`

`^rowse00121`

`^K cbDockPane GetDockingState`

`^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^cbdockpane')")`

`^K GetDockingState`

`$#+K!cbDockPane::GetFirstRow`

`cbRowInfo* GetFirstRow()`^K

Returns the first row.

^cbDockPane::GetFirstRow

^cbdockpanegetfirstrow

^browse00122

^K cbDockPane GetFirstRow

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `cbdockpane`)"

^K GetFirstRow

`$#+K!cbDockPane::GetMinimalRowHeight`

`int GetMinimalRowHeight(cbRowInfo* pRow)`^K

Returns the minimal row height for the given row. Internal function called by plugins.

^cbDockPane::GetMinimalRowHeight

^cbdockpanegetminimalrowheight

^browse00123

^K cbDockPane GetMinimalRowHeight

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `cbdockpane`)"

^K GetMinimalRowHeight

`$#+K!cbDockPane::GetNotFixedBarsCount`

`int GetNotFixedBarsCount(cbRowInfo* pRow)K`

Returns the number of bars whose size is not fixed. Internal function called by plugins.

`^bDockPane::GetNotFixedBarsCount`

`^bdockpanegetnotfixedbarscount`

`^rowse00124`

`^cbDockPane GetNotFixedBarsCount`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^cbdockpane')")`

`^GetNotFixedBarsCount`

`$#+K!cbDockPane::GetPaneHeight`

`int GetPaneHeight()`^K

Returns the height in the pane's coordinates.

`^bDockPane::GetPaneHeight`

`^bdockpanegetpaneheight`

`^rowse00125`

`^cbDockPane GetPaneHeight`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^cbdockpane')")`

`^GetPaneHeight`

`cbDockPane::GetRealRect`

`wxRect& GetRealRect()`

Returns the bounds of the pane, in parent coordinates.

`cbDockPane::GetRealRect`

`cbdockpanegetrealrect`

`rowse00126`

`cbDockPane GetRealRect`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `cbdockpane')")`

`GetRealRect`

`$#+K!cbDockPane::GetRow`

`cbRowInfo* GetRow(int row)`^K

protected really (accessed only by plugins) Returns the row info for a row index. Internal function called by plugins.

`^bDockPane::GetRow`

`^bdockpanegetrow`

`^rowse00127`

`^cbDockPane GetRow`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^cbdockpane')")`

`^GetRow`

\$#+K! **cbDockPane::GetRowAt**

int GetRowAt(int *paneY*)^K

Returns the row at the given vertical position. Returns -1 if the row is not present at given vertical position. Internal function called by plugins.

int GetRowAt(int *upperY*, int *lowerY*)^K

Returns the row between the given vertical positions. Returns -1 if the row is not present. Internal function called by plugins.

^cbDockPane::GetRowAt

^cbdockpanegetrowat

^browse00128

^K cbDockPane GetRowAt

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `cbdockpane`)")

^K GetRowAt

^K GetRowAt

`$#+K!cbDockPane::GetRowIndex`

`int GetRowIndex(cbRowInfo* pRow)K`

Returns the row index for the given row info. Internal function called by plugins.

`^bDockPane::GetRowIndex`

`^bdockpanegetrowindex`

`^rowse00129`

`^cbDockPane GetRowIndex`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^cbdockpane')")`

`^GetRowIndex`

`$#+K!cbDockPane::GetRowList`

`RowArrayT& GetRowList()`^K

Returns an array of rows. Used by updates-managers.

`cbDockPane::GetRowList`

`cbdockpanegetrowlist`

`rowse00130`

`cbDockPane GetRowList`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `cbdockpane`))`

`GetRowList`

`$#+K!cbDockPane::GetRowResizeRange`

**`void GetRowResizeRange(cbRowInfo* pRow, int* from, int* till, bool
forUpperHandle)K`**

Returns the row's resize range.

`cbDockPane::GetRowResizeRange`

`cbdockpanegetrowresizerange`

`browse00131`

`KcbDockPane GetRowResizeRange`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `cbdockpane`))`

`KGetRowResizeRange`

`cbDockPane::GetRowShapeData`

`void GetRowShapeData(cbRowInfo* pRow, wxList* pLst)`^K

Returns row shape data. `cbBarShapeData` objects will be added to the given `pLst`. `cbBarShapeData` is used for storing the original bar's positions in the row, when the 'non-destructive-friction' option is turned on.

`cbDockPane::GetRowShapeData`

`cbdockpanegetrowshapedata`

`rowse00132`

`cbDockPane GetRowShapeData`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `cbdockpane`))`

`GetRowShapeData`

`$#+K!cbDockPane::GetRowY`

`int GetRowY(cbRowInfo* pRow)K`

Gets the vertical position at the given row. Internal function called by plugins.

`cbDockPane::GetRowY`

`cbdockpanegetrowy`

`browse00133`

`KcbDockPane GetRowY`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `cbdockpane`))`

`KGetRowY`

`$#+K!cbDockPane::HasNotFixedBarsLeft`

`bool HasNotFixedBarsLeft(cbBarInfo* pBar)K`

Returns TRUE if there are any variable-sized rows to the left of this one. Internal function called by plugins.

`^bDockPane::HasNotFixedBarsLeft`

`^bdockpanehasnotfixedbarsleft`

`^rowse00134`

`^K cbDockPane HasNotFixedBarsLeft`

`^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdockpane')")`

`^K HasNotFixedBarsLeft`

`$#+K!cbDockPane::HasNotFixedBarsRight`

`bool HasNotFixedBarsRight(cbBarInfo* pBar)K`

Returns TRUE if there are any variable-sized rows to the right of this one. Internal function called by plugins.

`^bDockPane::HasNotFixedBarsRight`

`^bdockpanehasnotfixedbarsright`

`^rowse00135`

`KcbDockPane HasNotFixedBarsRight`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^cbdockpane')")`

`KHasNotFixedBarsRight`

`$#+K!cbDockPane::HasNotFixedRowsAbove`

`bool HasNotFixedRowsAbove(cbRowInfo* pRow)K`

Returns TRUE if there are any variable-sized rows above this one. Internal function called by plugins.

`^bDockPane::HasNotFixedRowsAbove`

`^bdockpanehasnotfixedrowsabove`

`^rowse00136`

`^cbDockPane HasNotFixedRowsAbove`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdockpane')")`

`^ HasNotFixedRowsAbove`

`$#+K!cbDockPane::HasNotFixedRowsBelow`

`bool HasNotFixedRowsBelow(cbRowInfo* pRow)`^K

Returns TRUE if there are any variable-sized rows below this one. Internal function called by plugins.

^cbDockPane::HasNotFixedRowsBelow

^cbdockpanehasnotfixedrowsbelow

^browse00137

^K cbDockPane HasNotFixedRowsBelow

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`cbdockpane')")

^K HasNotFixedRowsBelow

\$#+K! **cbDockPane::HasPoint**

bool HasPoint(const wxPoint& *pos*, int *x*, int *y*, int *width*, int *height*)^K

Returns TRUE if *pos* is within the given rectangle. Internal function called by plugins.

^bDockPane::HasPoint

^bdockpanehaspoint

^rowse00138

^cbDockPane HasPoint

^nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")

^HasPoint

^{\$#+K!}**cbDockPane::HitTestPanelItems**

int HitTestPanelItems(const wxPoint& *pos*, cbRowInfo *ppRow*, cbBarInfo** *ppBar*)**^K

Returns the result of hit-testing items in the pane. See CB_HITTEST_RESULT enumerated type. *pos* is the position in this pane's coordinates.

^cbDockPane::HitTestPanelItems

^cbdockpaneittestpaneitems

^browse00139

^K cbDockPane HitTestPanelItems

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")

^K HitTestPanelItems

`cbDockPane::InitLinksForRow`

`void InitLinksForRow(cbRowInfo* pRow)`^K

Sets up links between bars. Internal function called by plugins.

`cbDockPane::InitLinksForRow`

`cbdockpaneinitlinksforrow`

`rowse00140`

`cbDockPane InitLinksForRow`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")`

`InitLinksForRow`

`cbDockPane::InitLinksForRows`

`void InitLinksForRows()`^K

Sets up links between bars. Internal function called by plugins.

`cbDockPane::InitLinksForRows`

`cbdockpaneinitlinksforrows`

`rowse00141`

`cbDockPane InitLinksForRows`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `cbdockpane`))`

`InitLinksForRows`

cbDockPane::InsertBar

void InsertBar(cbBarInfo* pBarInfo)^K

Inserts bar and sets its position according to the preferred settings given in pBarInfo.

void InsertBar(cbBarInfo* pBar, const wxRect& rect)^K

Inserts the bar into this pane. rect is given in the parent frame's coordinates.

void InsertBar(cbBarInfo* pBar, cbRowInfo* pIntoRow)^K

Inserts the bar into the given row, with dimensions and position stored in pBarInfo->mBounds. Returns the node of inserted bar.

^cbDockPane::InsertBar

^cbdockpaneinsertbar

^browse00142

^K cbDockPane InsertBar

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `cbdockpane')")

^K InsertBar

^K InsertBar

^K InsertBar

cbDockPane::InsertRow

void InsertRow(cbRowInfo* pRow, cbRowInfo* pBeforeRow)^K

Inserts a row. Does not refresh the inserted row immediately. If pBeforeRowNode is NULL, the row is appended to the end of pane's row list.

^cbDockPane::InsertRow

^cbdockpaneinsertrow

^browse00143

^K cbDockPane InsertRow

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")

^K InsertRow

`$#+K!cbDockPane::IsFixedSize`

`bool IsFixedSize(cbBarInfo* pInfo)K`

Returns TRUE if the bar's dimension information indicates a fixed size. Internal function called by plugins.

`^bDockPane::IsFixedSize`

`^bdockpaneisfixedsize`

`^rowse00144`

`^cbDockPane IsFixedSize`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^cbdockpane')")`

`^ IsFixedSize`

`$#+K!cbDockPane::IsHorizontal`

`bool IsHorizontal()`^K

Returns TRUE if the pane is aligned to the top or bottom.

`cbDockPane::IsHorizontal`

`cbdockpaneishorizontal`

`rowse00145`

`cbDockPane IsHorizontal`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `cbdockpane`))`

`IsHorizontal`

`$$$+K!cbDockPane::MatchesMask`

`bool MatchesMask(int paneMask)K`

Returns TRUE if the given mask matches the pane's mask.

`^bDockPane::MatchesMask`

`^bdockpanematchesmask`

`^rowse00146`

`KcbDockPane MatchesMask`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdockpane')")`

`KMatchesMask`

`$#+K!` **cbDockPane::PaintBar**

void PaintBar(cbBarInfo* *pBar*, wxDC& *dc*)^K

Calls PaintBarDecorations and PaintBarHandles. Internal function called by plugins.

`^bDockPane::PaintBar`

`^bdockpanepaintbar`

`^rowse00147`

`^cbDockPane PaintBar`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^cbdockpane')")`

`^ PaintBar`

`cbDockPane::PaintBarDecorations`

`void PaintBarDecorations(cbBarInfo* pBar, wxDC& dc)`^K

protected really (accessed only by plugins) Generates a `cbDrawBarDecorEvent` and sends it to the layout to paint the bar decorations. Internal function called by plugins.

^c`bDockPane::PaintBarDecorations`

^c`bdockpanepaintbardecorations`

^b`rowse00148`

^K `cbDockPane PaintBarDecorations`

^E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")`

^K `PaintBarDecorations`

`cbDockPane::PaintBarHandles`

`void PaintBarHandles(cbBarInfo* pBar, wxDC& dc)`^K

Generates a `cbDrawBarHandlesEvent` and sends it to the layout to paint the bar handles. Internal function called by plugins.

`cbDockPane::PaintBarHandles`

`cbdockpanepaintbarhandles`

`rowse00149`

^K `cbDockPane PaintBarHandles`

^E `enableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")`

^K `PaintBarHandles`

`$#+K!` **cbDockPane::PaintPane**

void PaintPane(wxDC& dc)^K

Paints the pane background, the row background and decorations, and finally the pane decorations. Internal function called by plugins.

`^bDockPane::PaintPane`

`^bdockpanepaintpane`

`^rowse00150`

`^cbDockPane PaintPane`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^cbdockpane')")`

`^ PaintPane`

`$#+K!cbDockPane::PaintPaneBackground`

`void PaintPaneBackground(wxDC& dc)`^K

Generates `cbDrawPaneBkGroundEvent` and sends it to the layout. Internal function called by plugins.

`cbDockPane::PaintPaneBackground`

`cbdockpanepaintpanebackground`

`rowse00151`

^K `cbDockPane PaintPaneBackground`

^E `nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")`

^K `PaintPaneBackground`

`cbDockPane::PaintPaneDecorations`

`void PaintPaneDecorations(wxDC& dc)`^K

Generates `cbDrawPaneDecorEvent` and sends it to the layout. Internal function called by plugins.

^c`bDockPane::PaintPaneDecorations`

^c`bdockpanepaintpanedecorations`

^b`rowse00152`

^K `cbDockPane PaintPaneDecorations`

^E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")`

^K `PaintPaneDecorations`

`cbDockPane::PaintRow`

`void PaintRow(cbRowInfo* pRow, wxDC& dc)`^K

Calls PaintRowBackground, PaintRowDecorations, PaintRowHandles. Internal function called by plugins.

^cbDockPane::PaintRow

^cbdockpanepaintrow

^browse00153

^K cbDockPane PaintRow

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")

^K PaintRow

`cbDockPane::PaintRowBackground`

`void PaintRowBackground(cbRowInfo* pRow, wxDC& dc)`^K

Generates `cbDrawRowBkGroundEvent` and sends it to the layout. Internal function called by plugins.

`cbDockPane::PaintRowBackground`

`cbdockpanepaintrowbackground`

`rowse00154`

^K `cbDockPane PaintRowBackground`

^E `enableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")`

^K `PaintRowBackground`

^{\$#+K!}**cbDockPane::PaintRowDecorations**

void PaintRowDecorations(cbRowInfo* *pRow*, wxDC& *dc*)^K

Calls PaintBarDecorations for each row. Internal function called by plugins.

^cbDockPane::PaintRowDecorations

^cbdockpanepaintrowdecorations

^browse00155

^K cbDockPane PaintRowDecorations

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")

^K PaintRowDecorations

`$#+K!cbDockPane::PaintRowHandles`

`void PaintRowHandles(cbRowInfo* pRow, wxDC& dc)`^K

Generates `cbDrawRowHandlesEvent` and `cbDrawRowDecorEvent` and sends them to the layout. Internal function called by plugins.

^c`bDockPane::PaintRowHandles`

^c`bdockpanepaintrowhandles`

^b`rowse00156`

^K`cbDockPane PaintRowHandles`

^E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")`

^K`PaintRowHandles`

`$#+K!cbDockPane::PaneToFrame`

`void PaneToFrame(wxRect* pRect)`^K

Coordinate translation between parent's frame and this pane. Internal function called by plugins.

`void PaneToFrame(int* x, int* y)`^K

Coordinate translation between parent's frame and this pane. Internal function called by plugins.

`^bDockPane::PaneToFrame`

`^bdockpanepanetoframe`

`^rowse00157`

`^cbDockPane PaneToFrame`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdockpane')")`

`^ PaneToFrame`

`^ PaneToFrame`

cbDockPane::RecalcLayout

void RecalcLayout()^K

Generates events to perform layout calculations.

^cbDockPane::RecalcLayout

^cbdockpanerecalclayout

^browse00158

^K cbDockPane RecalcLayout

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `cbdockpane`)"

^K RecalcLayout

cbDockPane::RecalcRowLayout

void RecalcRowLayout(cbRowInfo* pRow)

Generates a cbLayoutRowEvent event to recalculate row layouts. Internal function called by plugins.

cbDockPane::RecalcRowLayout

cbdockpanerecalcrowlayout

rowse00159

cbDockPane RecalcRowLayout

enableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`,`cbdockpane`))

RecalcRowLayout

cbDockPane::RemoveBar

void RemoveBar(cbBarInfo* *pBar*)

Removes the bar from this pane. Does not destroy the bar.

cbDockPane::RemoveBar

cbdockpaneremovebar

rowse00160

cbDockPane RemoveBar

enableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `cbdockpane`))

RemoveBar

`cbDockPane::RemoveRow`

`void RemoveRow(cbRowInfo* pRow)`^K

Removes the row from this pane. Does not destroy the row object.

`cbDockPane::RemoveRow`

`cbdockpaneremoverow`

`rowse00161`

`cbDockPane RemoveRow`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdockpane')")`

`RemoveRow`

##+K! **cbDockPane::ResizeBar**

void ResizeBar(**cbBarInfo*** *pBar*, **int** *ofs*, **bool** *forLeftHandle*)^K

Row/bar resizing related helper-method.

^bDockPane::ResizeBar

^bdockpaneresizebar

^rowse00162

^cbDockPane ResizeBar

^nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")

^ ResizeBar

cbDockPane::ResizeRow

void ResizeRow(**cbRowInfo*** *pRow*, **int** *ofs*, **bool** *forUpperHandle*)^K

Row/bar resizing related helper-method.

^cbDockPane::ResizeRow

^cbdockpaneresizerow

^browse00163

^K cbDockPane ResizeRow

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")

^K ResizeRow

`$#+K!cbDockPane::SetBoundsInParent`

`void SetBoundsInParent(const wxRect& rect)K`

Set the position and dimensions of the pane in the parent frame's coordinates.

`^bDockPane::SetBoundsInParent`

`^bdockpanesetboundsinparent`

`^rowse00164`

`KcbDockPane SetBoundsInParent`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^cbdockpane')")`

`KSetBoundsInParent`

`$#+K!` **cbDockPane::SetMargins**

void SetMargins(**int** *top*, **int** *bottom*, **int** *left*, **int** *right*)^K

Sets pane's margins in frame's coordinate orientations.

`^bDockPane::SetMargins`

`^bdockpanesetmargins`

`^rowse00165`

`^cbDockPane SetMargins`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^cbdockpane')")`

`^SetMargins`

`$#+K!cbDockPane::SetPaneWidth`

`void SetPaneWidth(int width)K`

Sets pane's width in the pane's coordinates (including margins).

`^bDockPane::SetPaneWidth`

`^bdockpanesetpanewidth`

`^rowse00166`

`KcbDockPane SetPaneWidth`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^cbdockpane')")`

`KSetPaneWidth`

`$#+K!cbDockPane::SetRowHeight`

`void SetRowHeight(cbRowInfo* pRow, int newHeight)K`

Sets the row height for the given height. newHeight includes the height of row handles, if present. Internal function called by plugins.

`cbDockPane::SetRowHeight`

`cbdockpanesetrowheight`

`browse00167`

`KcbDockPane SetRowHeight`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")`

`KSetRowHeight`

`cbDockPane::SetRowShapeData`

`void SetRowShapeData(cbRowInfo* pRowNode, wxList* pLst)`^K

Sets the shape data for the given row, using the data provided in pLst. cbBarShapeData is used for storing the original bar's positions in the row, when the 'non-destructive-friction' option is turned on.

^cbDockPane::SetRowShapeData

^cbdockpanesetrowshapedata

^browse00168

^K cbDockPane SetRowShapeData

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")

^K SetRowShapeData

\$#+K!cbDockPane::SizeBar

void SizeBar(cbBarInfo* pBar)^K

Generates a cbSizeBarWndEvent and sends it to the layout. Internal function called by plugins.

^cbDockPane::SizeBar

^cbdockpanesizebar

^browse00169

^K cbDockPane SizeBar

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")

^K SizeBar

`$#+K!cbDockPane::SizePaneObjects`

`void SizePaneObjects()`^K

Calls `SizeRowObjects` for each row. Internal function called by plugins.

^c`bDockPane::SizePaneObjects`

^c`bdockpanesizepaneobjects`

^b`rowse00170`

^K`cbDockPane SizePaneObjects`

^E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")`

^K`SizePaneObjects`

`$#+K!cbDockPane::SizeRowObjects`

`void SizeRowObjects(cbRowInfo* pRow)K`

Calls SizeBar for each bar in the row. Internal function called by plugins.

`cbDockPane::SizeRowObjects`

`cbdockpanesizerowobjects`

`browse00171`

`KcbDockPane SizeRowObjects`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")`

`KSizeRowObjects`

`cbDockPane::StartDrawInArea`

`wxDC* StartDrawInArea(const wxRect& area)`^K

Generates `cbStartDrawInAreaEvent` and sends it to the layout. Internal function called by plugins.

^c`bDockPane::StartDrawInArea`

^c`bdockpanestartdrawinarea`

^b`rowse00172`

^K`cbDockPane StartDrawInArea`

^E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")`

^K`StartDrawInArea`

`$#+K!cbDockPane::SyncRowFlags`

`void SyncRowFlags(cbRowInfo* pRow)K`

Sets up flags in the row information structure, so that they match the changed state of row items correctly. Internal function called by plugins.

`cbDockPane::SyncRowFlags`

`cbdockpanesyncrowflags`

`browse00173`

`KcbDockPane SyncRowFlags`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")`

`K SyncRowFlags`

^{\$#+K!}**cbDrawBarDecorEvent::cbDrawBarDecorEvent**

cbDrawBarDecorEvent(**cbBarInfo*** *pBar*, **wxDC&** *dc*, **cbDockPane*** *pPane*)^K

Constructor, taking bar information, device context, and pane.

^cbDrawBarDecorEvent::cbDrawBarDecorEvent

^cbdrawbardecocoreventcbdrawbardecocorevent

^browse00175

^K cbDrawBarDecorEvent cbDrawBarDecorEvent

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbdrawbardecocorevent')")

^K cbDrawBarDecorEvent

^{\$#+K!}**cbDrawBarHandlesEvent::cbDrawBarHandlesEvent**

cbDrawBarHandlesEvent(**cbBarInfo*** *pBar*, **wxDC&** *dc*, **cbDockPane*** *pPane*)^K

Constructor, taking bar information, device context, and pane.

[°]**cbDrawBarHandlesEvent::cbDrawBarHandlesEvent**

[°]**cbdrawbarhandleseventcbdrawbarhandlesevent**

^browse00177

^K **cbDrawBarHandlesEvent** **cbDrawBarHandlesEvent**

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbdrawbarhandlesevent')")

^K **cbDrawBarHandlesEvent**

`cbDrawHintRectEvent::cbDrawHintRectEvent`

`cbDrawHintRectEvent(const wxRect& rect, bool isInClient, bool eraseRect, bool lastTime)`^K

e.g. with fat/hatched border Constructor, taking hint rectangle and three flags.

`cbDrawHintRectEvent::cbDrawHintRectEvent`
`cbdrawhintrecteventcbdrawhintrectevent`
`rowse00179`
`cbDrawHintRectEvent cbDrawHintRectEvent`
`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`
``cbdrawhintrectevent')`
`cbDrawHintRectEvent`

`cbDrawPaneBkGroundEvent::cbDrawPaneBkGroundEvent`

`cbDrawPaneBkGroundEvent(wxDC& dc, cbDockPane* pPane)`^K

Constructor, taking device context and pane.

`cbDrawPaneBkGroundEvent::cbDrawPaneBkGroundEvent`
`cbdrawpanebkgroundeventcbdrawpanebkgroundevent`
`rowse00181`
`cbDrawPaneBkGroundEvent cbDrawPaneBkGroundEvent`
`enableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp',`
``cbdrawpanebkgroundevent'))`
`cbDrawPaneBkGroundEvent`

cbDrawPaneDecorEvent::cbDrawPaneDecorEvent

cbDrawPaneDecorEvent(wxDC& *dc*, cbDockPane* *pPane*)^K

Constructor, taking device context and pane.

^cbDrawPaneDecorEvent::cbDrawPaneDecorEvent

^cbdrawpanedecoreventcbdrawpanedecorevent

^browse00183

^K cbDrawPaneDecorEvent cbDrawPaneDecorEvent

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',

`cbdrawpanedecorevent')

^K cbDrawPaneDecorEvent

`cbDrawRowBkGroundEvent::cbDrawRowBkGroundEvent`

`cbDrawRowBkGroundEvent(cbRowInfo* pRow, wxDC& dc, cbDockPane* pPane)`^K

Constructor, taking row information, device context, and pane.

`cbDrawRowBkGroundEvent::cbDrawRowBkGroundEvent`
`cbdrawrowbkgroundeventcbdrawrowbkgroundevent`
`rowse00185`
`cbDrawRowBkGroundEvent cbDrawRowBkGroundEvent`
`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`
``cbdrawrowbkgroundevent')`)
`cbDrawRowBkGroundEvent`

`$#+K!cbDrawRowDecorEvent::cbDrawRowDecorEvent`

`cbDrawRowDecorEvent(cbRowInfo* pRow, wxDC& dc, cbDockPane* pPane)K`

Constructor, taking row information, device context, and pane.

`cbDrawRowDecorEvent::cbDrawRowDecorEvent`

`cbdrawrowdecoreventcbdrawrowdecorevent`

`rowse00187`

`K cbDrawRowDecorEvent cbDrawRowDecorEvent`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbdrawrowdecorevent')")`

`K cbDrawRowDecorEvent`

`cbDrawRowHandlesEvent::cbDrawRowHandlesEvent`

`cbDrawRowHandlesEvent(cbRowInfo* pRow, wxDC& dc, cbDockPane* pPane)`^K

Constructor, taking row information, device context, and pane.

`cbDrawRowHandlesEvent::cbDrawRowHandlesEvent`

`cbdrawrowhandleseventcbdrawrowhandlesevent`

`rowse00189`

`cbDrawRowHandlesEvent cbDrawRowHandlesEvent`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbdrawrowhandlesevent")")`

`cbDrawRowHandlesEvent`

^{\$#+K!}**cbDynToolBarDimHandler::OnChangeBarState**

void OnChangeBarState(cbBarInfo* *pBar*, int *newState*)^K

Called when the bar changes state.

^cbDynToolBarDimHandler::OnChangeBarState

^cbdyntoolbardimhandleronchangebarstate

^browse00191

^K cbDynToolBarDimHandler OnChangeBarState

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbdyntoolbardimhandler')")

^K OnChangeBarState

^{##+K!}**cbDynToolBarDimHandler::OnResizeBar**

void OnResizeBar(**cbBarInfo*** *pBar*, **const wxSize&** *given*, **wxSize&** *preferred*)^K

Called when a bar is resized.

^cbDynToolBarDimHandler::OnResizeBar

^cbdyntoolbardimhandleronresizebar

^browse00192

^K cbDynToolBarDimHandler OnResizeBar

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',

[`]cbdyntoolbardimhandler')

^K OnResizeBar

cbFinishDrawInAreaEvent::cbFinishDrawInAreaEvent

cbFinishDrawInAreaEvent(const wxRect& *area*, cbDockPane* *pPane*)^K

Constructor, taking rectangular area and pane.

cbFinishDrawInAreaEvent::cbFinishDrawInAreaEvent

cbfinishdrawinareaeventcbfinishdrawinareaevent

rowse00194

cbFinishDrawInAreaEvent cbFinishDrawInAreaEvent

**enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbfinishdrawinareaevent'))**

cbFinishDrawInAreaEvent

cbFloatedBarWindow::cbFloatedBarWindow

cbFloatedBarWindow()^K

Default constructor.

^cbFloatedBarWindow::cbFloatedBarWindow
^cbfloatedbarwindowcbfloatedbarwindow
^browse00196
^K cbFloatedBarWindow cbFloatedBarWindow
^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbfloatedbarwindow')")
^K cbFloatedBarWindow

`$#+K!cbFloatedBarWindow::GetBar`

`cbBarInfo* GetBar()`^K

Returns the bar information for this window.

`°bFloatedBarWindow::GetBar`

`°bfloatedbarwindowgetbar`

`°rowse00197`

`°K cbFloatedBarWindow GetBar`

`°enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`

``cbfloatedbarwindow')")`

`°K GetBar`

`$#+K!` **cbFloatedBarWindow::GetPreferredSize**

wxSize **GetPreferredSize**(const **wxSize**& *given*)^K

Overridden function returning the preferred size.

`^bFloatedBarWindow::GetPreferredSize`

`^bfloatedbarwindowgetpreferredsize`

`^rowse00198`

`^K cbFloatedBarWindow GetPreferredSize`

`^enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
^cbfloatedbarwindow')")`

`^K GetPreferredSize`

`cbFloatedBarWindow::HandleTitleClick`

`bool HandleTitleClick(wxMouseEvent& event)`^K

Overridden function responding to mouse button clicks on the titlebar.

`cbFloatedBarWindow::HandleTitleClick`

`cbfloatedbarwindowhandletitleclick`

`rowse00199`

`cbFloatedBarWindow HandleTitleClick`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbfloatedbarwindow')")`

`HandleTitleClick`

`cbFloatedBarWindow::OnDbClick`

`void OnDbClick(wxMouseEvent& event)`^K

Responds to double-click mouse events.

`cbFloatedBarWindow::OnDbClick`

`cbfloatedbarwindowondbclick`

`rowse00200`

`cbFloatedBarWindow OnDbClick`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbfloatedbarwindow')")`

`OnDbClick`

`cbFloatedBarWindow::OnMiniButtonClicked`

`void OnMiniButtonClicked(int btnIdx)`^K

Overridden function responding to mouse clicks on mini-buttons.

`cbFloatedBarWindow::OnMiniButtonClicked`
`cbfloatedbarwindowonminibuttonclicked`
`rowse00201`
`cbFloatedBarWindow OnMiniButtonClicked`
`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`
``cbfloatedbarwindow')")`
`OnMiniButtonClicked`^K

`$#+K!cbFloatedBarWindow::PositionFloatedWnd`

`void PositionFloatedWnd(int scrX, int scrY, int width, int height)K`

Position the floating window. The given coordinates are those of the bar itself; the floated container window's position and size are adjusted accordingly.

`^bFloatedBarWindow::PositionFloatedWnd
^bfloatedbarwindowpositionfloatedwnd
^rowse00202
^cbFloatedBarWindow PositionFloatedWnd
^enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
^cbfloatedbarwindow')")
^PositionFloatedWnd`

cbFloatedBarWindow::SetBar

void SetBar(cbBarInfo* *pBar*)^K

Sets the bar information for this window.

^cbFloatedBarWindow::SetBar

^cbfloatedbarwindowsetbar

^browse00203

^K cbFloatedBarWindow SetBar

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',

`cbfloatedbarwindow')")

^K SetBar

`cbFloatedBarWindow::SetLayout`

`void SetLayout(wxFrameLayout* pLayout)`^K

Sets the layout for this window.

`cbFloatedBarWindow::SetLayout`

`cbfloatedbarwindowsetlayout`

`rowse00204`

`cbFloatedBarWindow SetLayout`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbfloatedbarwindow')")`

`SetLayout`

`cbGCUpdatesMgr::cbGCUpdatesMgr`

`cbGCUpdatesMgr()`^K

Default constructor.

`cbGCUpdatesMgr(wxFrameLayout* pPanel)`^K

Constructor, taking a frame layout.

^cbGCUpdatesMgr::cbGCUpdatesMgr

^cbgcupdatesmgrcbgcupdatesmgr

^browse00206

^K cbGCUpdatesMgr cbGCUpdatesMgr

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`cbgcupdatesmgr')")

^K cbGCUpdatesMgr

^K cbGCUpdatesMgr

`cbGCUpdatesMgr::AddItem`

`void AddItem(wxList& itemList, cbBarInfo* pBar, cbDockPane* pPane, wxRect& curBounds, wxRect& prevBounds)`^K

Internal function for repositioning items.

^cbGCUpdatesMgr::AddItem

^cbgcupdatesmgradditem

^browse00207

^K cbGCUpdatesMgr AddItem

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbgcupdatesmgr')")

^K AddItem

`$#+K!cbGCUpdatesMgr::DoRepositionItems`

`void DoRepositionItems(wxList& items)K`

Internal function for repositioning items.

^cbGCUpdatesMgr::DoRepositionItems

^cbgcupdatesmgrdorepositionitems

^browse00208

^K cbGCUpdatesMgr DoRepositionItems

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbgcupdatesmgr')")

^K DoRepositionItems

`cbGCUpdatesMgr::OnStartChanges`

`void OnStartChanges()`^K

Receives notifications from the frame layout.

`cbGCUpdatesMgr::OnStartChanges`

`cbgcupdatesmgronstartchanges`

`rowse00209`

^K `cbGCUpdatesMgr OnStartChanges`

^E `nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`cbgcupdatesmgr')")`

^K `OnStartChanges`

`$#+K!cbGCUpdatesMgr::UpdateNow`

`void UpdateNow()`^K

Refreshes the parts of the frame layout which need an update.

`^bGCUpdatesMgr::UpdateNow`
`^bgcupdatesmgrupdatenow`
`^rowse00210`
`^cbGCUpdatesMgr UpdateNow`
`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbgcupdatesmgr')")`
`^ UpdateNow`

`cbHintAnimationPlugin::cbHintAnimationPlugin`

`cbHintAnimationPlugin()`^K

Default constructor.

`cbHintAnimationPlugin(wxFrameLayout* pPanel, int paneMask = wxALL_PANES)`^K

Constructor, taking a layout panel and pane mask.

`cbHintAnimationPlugin::cbHintAnimationPlugin`
`cbhintanimationplugin::cbhintanimationplugin`
`rowse00212`
`cbHintAnimationPlugin cbHintAnimationPlugin`
`enableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp',`
``cbhintanimationplugin')`)
`cbHintAnimationPlugin`
`cbHintAnimationPlugin`

cbHintAnimationPlugin::~cbHintAnimationPlugin

~cbHintAnimationPlugin()^K

Destructor.

cbHintAnimationPlugin::~cbHintAnimationPlugin

cbhintanimationpluginindtor

rowse00213

cbHintAnimationPlugin ~cbHintAnimationPlugin

enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',

`cbhintanimationplugin'))

~cbHintAnimationPlugin

`cbHintAnimationPlugin::DoDrawHintRect`

`void DoDrawHintRect(wxRect& rect, bool isInClientRect)`^K

Internal function for drawing a hint rectangle.

`cbHintAnimationPlugin::DoDrawHintRect`

`cbhintanimationplugindodrawhintrect`

`rowse00214`

`cbHintAnimationPlugin DoDrawHintRect`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp',
`cbhintanimationplugin')")`

`DoDrawHintRect`

^{\$#+K!}**cbHintAnimationPlugin::DrawHintRect**

void DrawHintRect(wxRect& *rect*, **bool** *isInClientRect*)^K

Internal function for drawing a hint rectangle.

^cbHintAnimationPlugin::DrawHintRect

^cbhintanimationplugindrawhintrect

^browse00215

^K cbHintAnimationPlugin DrawHintRect

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',

`cbhintanimationplugin')")

^K DrawHintRect

`$#+K!cbHintAnimationPlugin::EraseHintRect`

`void EraseHintRect(wxRect& rect, bool isInClientRect)`^K

Internal function for erasing a hint rectangle.

`cbHintAnimationPlugin::EraseHintRect`

`cbhintanimationpluginerasehintrect`

`rowse00216`

`cbHintAnimationPlugin EraseHintRect`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbhintanimationplugin')")`

`EraseHintRect`

`cbHintAnimationPlugin::FinishTracking`

`void FinishTracking()`^K

Internal function for finishing tracking.

`cbHintAnimationPlugin::FinishTracking`
`cbhintanimationpluginfinishtracking`
`rowse00217`
`cbHintAnimationPlugin FinishTracking`
`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`
``cbhintanimationplugin')`)
`FinishTracking`

`cbHintAnimationPlugin::OnDrawHintRect`

`void OnDrawHintRect(cbDrawHintRectEvent& event)`^K

Event handler respoding to hint draw events.

`cbHintAnimationPlugin::OnDrawHintRect`

`cbhintanimationpluginondrawhintrect`

`rowse00218`

`cbHintAnimationPlugin OnDrawHintRect`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbhintanimationplugin')`

`OnDrawHintRect`

`cbHintAnimationPlugin::RectToScr`

`void RectToScr(wxRect& frameRect, wxRect& scrRect)`^K

Internal function for translating coordinates.

`cbHintAnimationPlugin::RectToScr`

`cbhintanimationpluginrecttoscr`

`rowse00219`

`cbHintAnimationPlugin RectToScr`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbhintanimationplugin')`

`RectToScr`

`$#+K!cbHintAnimationPlugin::StartTracking`

`void StartTracking()`^K

speed is constant. Default: TRUE TBD:: get/set methods for above members Internal function for starting tracking.

`^bHintAnimationPlugin::StartTracking`
`^bhintanimationpluginstarttracking`
`^rowse00220`
`^K cbHintAnimationPlugin StartTracking`
`^enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`
`^cbhintanimationplugin')`
`^K StartTracking`

cbInsertBarEvent::cbInsertBarEvent

cbInsertBarEvent(cbBarInfo* *pBar*, cbRowInfo* *pIntoRow*, cbDockPane* *pPane*)^K

Constructor, taking bar information, row information, and pane.

^cbInsertBarEvent::cbInsertBarEvent

^cbinsertbareventcbinsertbarevent

^browse00222

^K cbInsertBarEvent cbInsertBarEvent

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbinsertbarevent')")

^K cbInsertBarEvent

`cbLayoutRowEvent::cbLayoutRowEvent`

`cbLayoutRowEvent(cbRowInfo* pRow, cbDockPane* pPane)`^K

Constructor, taking row information and pane.

`cbLayoutRowEvent::cbLayoutRowEvent`

`cbayoutroweventcbayoutrowevent`

`rowse00224`

`cbLayoutRowEvent cbLayoutRowEvent`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`cbayoutrowevent')")`

`cbLayoutRowEvent`

\$#+K! cbLayoutRowsEvent::cbLayoutRowsEvent

cbLayoutRowsEvent(cbDockPane* *pPane*)^k

Constructor, taking pane.

```

^bLayoutRowsEvent::cbLayoutRowsEvent
^blayoutrowseventcblayoutrowsevent
^rowse00226
^K cbLayoutRowsEvent cbLayoutRowsEvent
^enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
^cblayoutrowsevent')")
^K cbLayoutRowsEvent

```


`cbLeftDClickEvent::cbLeftDClickEvent`

`cbLeftDClickEvent(const wxPoint& pos, cbDockPane* pPane)`^K

Constructor, taking mouse position and pane.

`cbLeftDClickEvent::cbLeftDClickEvent`

`cbLeftDClickEvent::cbLeftDClickEvent`

`cbLeftDClickEvent`

`cbLeftDClickEvent cbLeftDClickEvent`

`cbLeftDClickEvent cbLeftDClickEvent`

`cbLeftDClickEvent`

cbLeftDownEvent::cbLeftDownEvent

cbLeftDownEvent(const wxPoint& pos, cbDockPane* pPane)^K

Constructor, taking mouse position and pane.

^cbLeftDownEvent::cbLeftDownEvent

^cbleftdowneventcbleftdownevent

^browse00230

^K cbLeftDownEvent cbLeftDownEvent

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbleftdownevent')")

^K cbLeftDownEvent

`cbLeftUpEvent::cbLeftUpEvent`

`cbLeftUpEvent(const wxPoint& pos, cbDockPane* pPane)`^K

Constructor, taking mouse position and pane.

`cbLeftUpEvent::cbLeftUpEvent`

`cbLeftUpEvent::cbLeftUpEvent`

`cbLeftUpEvent`

`cbLeftUpEvent`

`cbLeftUpEvent`

`cbLeftUpEvent`

cbMiniButton::cbMiniButton

cbMiniButton()^K

Default constructor.

cbMiniButton::cbMiniButton

cbminibuttoncbminibutton

rowse00234

cbMiniButton cbMiniButton

enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`cbminibutton')")

cbMiniButton

`$#+K!cbMiniButton::Draw`

`void Draw(wxDC& dc)K`

Draws the button. Override this to implement the desired appearance.

`cbMiniButton::Draw`

`cbminibuttondraw`

`browse00235`

`KcbMiniButton Draw`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp',`cbminibutton')")`

`KDraw`

`$#+K!cbMiniButton::Enable`

`void Enable(bool enable)K`

Enable or disable the button.

`^bMiniButton::Enable`

`^bminibuttonenable`

`^rowse00236`

`KcbMiniButton Enable`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^cbminibutton')")`

`K Enable`

`cbMiniButton::HitTest`

`bool HitTest(const wxPoint& pos)`^K

Returns TRUE if the given position was over the button.

`cbMiniButton::HitTest`

`cbminibuttonhittest`

`rowse00237`

`cbMiniButton HitTest`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp',`cbminibutton')")`

`HitTest`

`$#+K!cbMiniButton::IsPressed`

`bool IsPressed()`^K

Returns TRUE if this button is pressed.

^cbMiniButton::IsPressed

^cbminibuttonispressed

^browse00238

^K cbMiniButton IsPressed

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbminibutton')")

^K IsPressed

`$#+K!cbMiniButton::OnLeftDown`

`void OnLeftDown(const wxPoint& pos)K`

Responds to a left down event.

`cbMiniButton::OnLeftDown`

`cbminibuttononleftdown`

`browse00239`

`KcbMiniButton OnLeftDown`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbminibutton')")`

`KOnLeftDown`

cbMiniButton::OnLeftUp

void OnLeftUp(const wxPoint& pos)^K

Responds to a left up event.

^cbMiniButton::OnLeftUp

^cbminibuttononleftup

^browse00240

^K cbMiniButton OnLeftUp

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbminibutton')")

^K OnLeftUp

cbMiniButton::OnMotion

void OnMotion(const wxPoint& pos)^K

Responds to a mouse move event.

^cbMiniButton::OnMotion

^cbminibuttononmotion

^browse00241

^K cbMiniButton OnMotion

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbminibutton')")

^K OnMotion

cbMiniButton::Refresh

void Refresh()

Refreshes the button.

cbMiniButton::Refresh

cbminibuttonrefresh

rowse00242

cbMiniButton Refresh

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp',`cbminibutton')")

Refresh

cbMiniButton::Reset

void Reset()

Reset the button.

cbMiniButton::Reset

cbminibuttonreset

rowse00243

cbMiniButton Reset

enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp',`cbminibutton')")

Reset

cbMiniButton::SetPos

void SetPos(const wxPoint& pos)

Set the position of the button.

cbMiniButton::SetPos

cbminibuttonsetpos

rowse00244

cbMiniButton SetPos

enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp',`cbminibutton')")

SetPos

`cbMiniButton::WasClicked`

`bool WasClicked()`

Returns TRUE if the button was clicked.

`cbMiniButton::WasClicked`

`cbminibuttonwasclicked`

`rowse00245`

`cbMiniButton WasClicked`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp',`cbminibutton')")`

`WasClicked`

`cbMotionEvent::cbMotionEvent`

`cbMotionEvent(const wxPoint& pos, cbDockPane* pPane)`^K

Constructor, taking mouse position and pane.

`cbMotionEvent::cbMotionEvent`

`cbmotioneventcbmotionevent`

`rowse00247`

`cbMotionEvent cbMotionEvent`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbmotionevent')")`

`cbMotionEvent`

cbPaneDrawPlugin::cbPaneDrawPlugin

cbPaneDrawPlugin(wxFrameLayout* *pPanel*, int *paneMask* = wxALL_PANES)^K

Constructor taking frame layout pane and a pane mask.

cbPaneDrawPlugin()^K

Default constructor.

^cbPaneDrawPlugin::cbPaneDrawPlugin

^cbpanedrawplugincbpanedrawplugin

^browse00249

^K cbPaneDrawPlugin cbPaneDrawPlugin

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`cbpanedrawplugin')")

^K cbPaneDrawPlugin

^K cbPaneDrawPlugin

`$#+K!cbPaneDrawPlugin::~~cbPaneDrawPlugin`

`~cbPaneDrawPlugin()`^K

Destructor.

`cbPaneDrawPlugin::~~cbPaneDrawPlugin`

`cbpanedrawplugindtor`

`rowse00250`

`KcbPaneDrawPlugin ~cbPaneDrawPlugin`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbpanedrawplugin')")`

`K ~cbPaneDrawPlugin`

cbPaneDrawPlugin::Clone

cbPluginBase* Clone()^K

Clone function, returning a new instance of this class.

^cbPaneDrawPlugin::Clone

^cbpanedrawpluginclone

^browse00251

^K cbPaneDrawPlugin Clone

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbpanedrawplugin')")

^K Clone

cbPaneDrawPlugin::DrawBarInnerShadeRect

void DrawBarInnerShadeRect(cbBarInfo* pBar, wxDC& dc)^K

Internal helper: draws the inner bar shading.

^cbPaneDrawPlugin::DrawBarInnerShadeRect

^cbpanedrawplugindrawbarinnershaderect

^browse00252

^K cbPaneDrawPlugin DrawBarInnerShadeRect

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbpanedrawplugin')")

^K DrawBarInnerShadeRect

`$#+K!cbPaneDrawPlugin::DrawDraggedHandle`

`void DrawDraggedHandle(const wxPoint& pos, cbDockPane& pane)K`

Internal helper: draws the dragged handle.

`cbPaneDrawPlugin::DrawDraggedHandle`

`cbpanedrawplugindrawdraggedhandle`

`browse00253`

`KcbPaneDrawPlugin DrawDraggedHandle`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbpanedrawplugin')")`

`KDrawDraggedHandle`

`cbPaneDrawPlugin::DrawLowerRowHandle`

`void DrawLowerRowHandle(cbRowInfo* pRow, wxDC& dc)`^K

Internal helper: draws the lower row handle.

^cbPaneDrawPlugin::DrawLowerRowHandle

^cbpanedrawplugindrawlowerrowhandle

^browse00254

^K cbPaneDrawPlugin DrawLowerRowHandle

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbpanedrawplugin')")

^K DrawLowerRowHandle

`cbPaneDrawPlugin::DrawLowerRowShades`

`void DrawLowerRowShades(cbRowInfo* pRow, wxDC& dc, int level)`^K

Internal helper: draws the lower row shading.

`cbPaneDrawPlugin::DrawLowerRowShades`
`cbpanedrawplugindrawlowerrowshades`
`rowse00255`
`cbPaneDrawPlugin DrawLowerRowShades`
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `cbpanedrawplugin')")`
`DrawLowerRowShades`

`$#+K!cbPaneDrawPlugin::DrawPaneShade`

`void DrawPaneShade(wxDC& dc, int alignment)K`

Internal helper: draws the pane shading.

`cbPaneDrawPlugin::DrawPaneShade`

`cbpanedrawplugindrawpaneshade`

`browse00256`

`KcbPaneDrawPlugin DrawPaneShade`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbpanedrawplugin')")`

`KDrawPaneShade`

cbPaneDrawPlugin::DrawPaneShadeForRow

void DrawPaneShadeForRow(cbRowInfo* pRow, wxDC& dc)^K

Internal helper: draws the pane shading for a row.

^cbPaneDrawPlugin::DrawPaneShadeForRow

^cbpanedrawplugindrawpaneshadeforrow

^browse00257

^K cbPaneDrawPlugin DrawPaneShadeForRow

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbpanedrawplugin')")

^K DrawPaneShadeForRow

cbPaneDrawPlugin::DrawShade

void DrawShade(int *level*, **wxRect&** *rect*, int *alignment*, **wxDC&** *dc*)^K

Internal helper: draws shading.

^cbPaneDrawPlugin::DrawShade

^cbpanedrawplugindrawshade

^browse00258

^K cbPaneDrawPlugin DrawShade

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbpanedrawplugin')")

^K DrawShade

`$#+K!cbPaneDrawPlugin::DrawShade1`

`void DrawShade1(int level, wxRect& rect, int alignment, wxDC& dc)K`

Internal helper: draws shading.

`cbPaneDrawPlugin::DrawShade1`

`cbpanedrawplugindrawshade1`

`browse00259`

`KcbPaneDrawPlugin DrawShade1`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbpanedrawplugin')")`

`KDrawShade1`

`$#+K!cbPaneDrawPlugin::DrawUpperRowHandle`

`void DrawUpperRowHandle(cbRowInfo* pRow, wxDC& dc)K`

Internal helper: draws the upper row handle.

`cbPaneDrawPlugin::DrawUpperRowHandle
cbpanedrawplugindrawupperrowhandle
browse00260
KcbPaneDrawPlugin DrawUpperRowHandle
EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbpanedrawplugin')")
KDrawUpperRowHandle`

`$#+K!cbPaneDrawPlugin::DrawUpperRowShades`

`void DrawUpperRowShades(cbRowInfo* pRow, wxDC& dc, int level)K`

Internal helper: draws the upper row shading.

`cbPaneDrawPlugin::DrawUpperRowShades`

`cbpanedrawplugindrawupperrowshades`

`browse00261`

`KcbPaneDrawPlugin DrawUpperRowShades`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbpanedrawplugin')")`

`KDrawUpperRowShades`

^{\$#+K!}**cbPaneDrawPlugin::OnDrawBarDecorations**

void OnDrawBarDecorations(cbDrawBarDecorEvent& *event*)^K

Handler for draw bar decorations events.

^cbPaneDrawPlugin::OnDrawBarDecorations

^cbpanedrawpluginondrawbardecorations

^browse00262

^K cbPaneDrawPlugin OnDrawBarDecorations

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbpanedrawplugin')")

^K OnDrawBarDecorations

^{\$#+K!}**cbPaneDrawPlugin::OnDrawBarHandles**

void OnDrawBarHandles(cbDrawBarHandlesEvent& *event*)^K

Handler for draw bar handles events.

^cbPaneDrawPlugin::OnDrawBarHandles

^cbpanedrawpluginondrawbarhandles

^browse00263

^K cbPaneDrawPlugin OnDrawBarHandles

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbpanedrawplugin')")

^K OnDrawBarHandles

`$#+K!cbPaneDrawPlugin::OnDrawPaneBackground`

`void OnDrawPaneBackground(cbDrawPaneBkGroundEvent& event)K`

Handler for draw pane background events.

`cbPaneDrawPlugin::OnDrawPaneBackground
cbpanedrawpluginondrawpanebackground
browse00264
K cbPaneDrawPlugin OnDrawPaneBackground
EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbpanedrawplugin')")
K OnDrawPaneBackground`

^{\$#+K!}**cbPaneDrawPlugin::OnDrawPaneDecorations**

void OnDrawPaneDecorations(cbDrawPaneDecorEvent& *event*)^K

Handler for draw pane decoration events.

^cbPaneDrawPlugin::OnDrawPaneDecorations

^cbpanedrawpluginondrawpanedecorations

^browse00265

^K cbPaneDrawPlugin OnDrawPaneDecorations

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbpanedrawplugin')")

^K OnDrawPaneDecorations

`$#+K!cbPaneDrawPlugin::OnDrawRowBackground`

`void OnDrawRowBackground(cbDrawRowBkGroundEvent& event)K`

Handler for draw row background events.

`cbPaneDrawPlugin::OnDrawRowBackground`
`cbpanedrawpluginondrawrowbackground`
`rowse00266`
`KcbPaneDrawPlugin OnDrawRowBackground`
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbpanedrawplugin')")`
`K OnDrawRowBackground`

cbPaneDrawPlugin::OnDrawRowDecorations

void OnDrawRowDecorations(cbDrawRowDecorEvent& *event*)^K

Handler for draw row decoration events.

^cbPaneDrawPlugin::OnDrawRowDecorations

^cbpanedrawpluginondrawrowdecorations

^browse00267

^K cbPaneDrawPlugin OnDrawRowDecorations

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbpanedrawplugin')")

^K OnDrawRowDecorations

`cbPaneDrawPlugin::OnDrawRowHandles`

`void OnDrawRowHandles(cbDrawRowHandlesEvent& event)`^K

Handler for draw row handles events.

`cbPaneDrawPlugin::OnDrawRowHandles`

`cbpanedrawpluginondrawrowhandles`

`rowse00268`

`cbPaneDrawPlugin OnDrawRowHandles`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `cbpanedrawplugin')")`

`OnDrawRowHandles`

^{\$#+K!}**cbPaneDrawPlugin::OnFinishDrawInArea**

void OnFinishDrawInArea(cbFinishDrawInAreaEvent& *event*)^K

Handler for finish draw in area events.

^cbPaneDrawPlugin::OnFinishDrawInArea

^cbpanedrawpluginonfinishdrawinarea

^browse00269

^K cbPaneDrawPlugin OnFinishDrawInArea

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`cbpanedrawplugin')")

^K OnFinishDrawInArea

`$#+K!cbPaneDrawPlugin::OnLButtonDown`

`void OnLButtonDown(cbLeftDownEvent& event)K`

Handler for left mouse button down events.

`cbPaneDrawPlugin::OnLButtonDown`

`cbpanedrawpluginonlbuttondown`

`rowse00270`

`KcbPaneDrawPlugin OnLButtonDown`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbpanedrawplugin')")`

`K OnLButtonDown`

cbPaneDrawPlugin::OnLButtonUp

void OnLButtonUp(cbLeftUpEvent& event)^K

Handler for left mouse button up events.

^cbPaneDrawPlugin::OnLButtonUp

^cbpanedrawpluginonlbuttonup

^browse00271

^K cbPaneDrawPlugin OnLButtonUp

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbpanedrawplugin')")

^K OnLButtonUp

`$#+K!cbPaneDrawPlugin::OnLDbIClick`

`void OnLDbIClick(cbLeftDClickEvent& event)K`

Handler for left double-click mouse button down events.

^cbPaneDrawPlugin::OnLDbIClick

^cbpanedrawpluginonldbclick

^browse00272

^K cbPaneDrawPlugin OnLDbIClick

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbpanedrawplugin')")

^K OnLDbIClick

cbPaneDrawPlugin::OnMouseMove

void OnMouseMove(cbMotionEvent& *event*)^K

Handler for mouse move events.

^cbPaneDrawPlugin::OnMouseMove

^cbpanedrawpluginonmousemove

^browse00273

^K cbPaneDrawPlugin OnMouseMove

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbpanedrawplugin')")

^K OnMouseMove

`$#+K!cbPaneDrawPlugin::OnRButtonUp`

`void OnRButtonUp(cbRightUpEvent& event)K`

Handler for right mouse button up events.

`cbPaneDrawPlugin::OnRButtonUp`

`cbpanedrawpluginonrbuttonup`

`rowse00274`

`KcbPaneDrawPlugin OnRButtonUp`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbpanedrawplugin')")`

`K OnRButtonUp`

`$#+K!cbPaneDrawPlugin::OnSizeBarWindow`

`void OnSizeBarWindow(cbSizeBarWndEvent& event)K`

Handler for bar size events.

`cbPaneDrawPlugin::OnSizeBarWindow`

`cbpanedrawpluginonsizebarwindow`

`rowse00275`

`KcbPaneDrawPlugin OnSizeBarWindow`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`cbpanedrawplugin')")`

`K OnSizeBarWindow`

^{\$#+K!}**cbPaneDrawPlugin::OnStartDrawInArea**

void OnStartDrawInArea(cbStartDrawInAreaEvent& *event*)^K

Handler for start draw in area events.

^cbPaneDrawPlugin::OnStartDrawInArea

^cbpanedrawpluginonstartdrawinarea

^browse00276

^K cbPaneDrawPlugin OnStartDrawInArea

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbpanedrawplugin')")

^K OnStartDrawInArea

`$#+K!cbPaneDrawPlugin::SetDarkPixel`

`void SetDarkPixel(int x, int y, wxDC& dc)K`

Internal helper: sets a dark pixel at the given location.

`^bPaneDrawPlugin::SetDarkPixel`

`^bpanedrawpluginsetdarkpixel`

`^rowse00277`

`KcbPaneDrawPlugin SetDarkPixel`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^cbpanedrawplugin')")`

`KSetDarkPixel`

`$#+K!cbPaneDrawPlugin::SetLightPixel`

`void SetLightPixel(int x, int y, wxDC& dc)K`

Internal helper: sets a light pixel at the given location.

`cbPaneDrawPlugin::SetLightPixel`

`cbpanedrawpluginsetlightpixel`

`browse00278`

`KcbPaneDrawPlugin SetLightPixel`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbpanedrawplugin')")`

`KSetLightPixel`

cbPluginBase::cbPluginBase

cbPluginBase(wxFrameLayout* pPanel, int paneMask = wxALL_PANES)^K

Constructor taking layout panel and a mask.

cbPluginBase()^K

Default constructor.

^cbPluginBase::cbPluginBase

^cbpluginbasecbpluginbase

^browse00280

^K cbPluginBase cbPluginBase

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`cbpluginbase')")

^K cbPluginBase

^K cbPluginBase

cbPluginBase::~~cbPluginBase

~cbPluginBase()^K

Destructor. Destroys the whole plugin chain of connected plugins.

cbPluginBase::~~cbPluginBase

cbpluginbasedtor

rowse00281

cbPluginBase ~cbPluginBase

enableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbpluginbase')")

~cbPluginBase

cbPluginBase::GetPaneMask

int GetPaneMask()

Returns the pane mask.

cbPluginBase::GetPaneMask
cbpluginbasegetpanemask
rowse00282
cbPluginBase GetPaneMask
EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp',`cbpluginbase')")
GetPaneMask

cbPluginBase::IsReady

bool IsReady()

Returns TRUE if the plugin is ready to receive events.

cbPluginBase::IsReady

cbpluginbaseisready

rowse00283

cbPluginBase IsReady

enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `cbpluginbase')")

IsReady

cbPluginBase::OnInitPlugin

void OnInitPlugin()

Override this method to do plugin-specific initialization. At this point plugin is already attached to the frame layout, and pane masks are set.

cbPluginBase::OnInitPlugin
cbpluginbaseoninitplugin
rowse00284
cbPluginBase OnInitPlugin
EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `cbpluginbase')")
OnInitPlugin

`cbPluginBase::ProcessEvent`

`bool ProcessEvent(wxEvent& event)`^K

Overridden to determine whether the target pane specified in the event matches the pane mask of this plugin (specific plugins do not override this method).

`cbPluginBase::ProcessEvent`

`cbpluginbaseprocessevent`

`rowse00285`

`cbPluginBase ProcessEvent`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbpluginbase')")`

`ProcessEvent`

`cbPluginEvent::cbPluginEvent`

`cbPluginEvent(wxEventType eventType, cbDockPane* pPane)`^K

Constructor, taking event type and pane.

`cbPluginEvent::cbPluginEvent`

`cbplugineventcbpluginevent`

`rowse00287`

`cbPluginEvent cbPluginEvent`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `cbpluginevent')")`

`cbPluginEvent`

`$#+KK!cbPluginEvent::Clone`

`wxEvent* Clone() const`

Not used, but required.

`cbPluginEvent::Clone`

`cbplugineventclone`

`browse00288`

`KcbPluginEvent Clone`

`KClone`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`cbpluginevent')")`

`$#+K!cbRemoveBarEvent::cbRemoveBarEvent`

`cbRemoveBarEvent(cbBarInfo* pBar, cbDockPane* pPane)K`

Constructor, taking bar information and pane.

`cbRemoveBarEvent::cbRemoveBarEvent`

`cbremovebareventcbremovebarevent`

`browse00290`

`KcbRemoveBarEvent cbRemoveBarEvent`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbremovebarevent')")`

`KcbRemoveBarEvent`

cbResizeBarEvent::cbResizeBarEvent

cbResizeBarEvent(cbBarInfo* pBar, cbRowInfo* pRow, cbDockPane* pPane)^K

Constructor, taking bar information, row information, and pane.

^cbResizeBarEvent::cbResizeBarEvent

^cbresizebareventcbresizebarevent

^browse00292

^K cbResizeBarEvent cbResizeBarEvent

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbresizebarevent')")

^K cbResizeBarEvent

`cbResizeRowEvent::cbResizeRowEvent`

`cbResizeRowEvent(cbRowInfo* pRow, int handleOfs, bool forUpperHandle, cbDockPane* pPane)`^K

Constructor, taking row information, two parameters of currently unknown use, and pane.

^c**`bResizeRowEvent::cbResizeRowEvent`**

^c**`bresizeroweventcbresizerowevent`**

^b**`rowse00294`**

^K**`cbResizeRowEvent cbResizeRowEvent`**

^E**`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbresizerowevent')")`**

^K**`cbResizeRowEvent`**

`cbRightDownEvent::cbRightDownEvent`

`cbRightDownEvent(const wxPoint& pos, cbDockPane* pPane)`^K

Constructor, taking mouse position and pane.

`cbRightDownEvent::cbRightDownEvent`

`cbrightdowneventcbrightdownevent`

`rowse00296`

`cbRightDownEvent cbRightDownEvent`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `cbrightdownevent')")`

`cbRightDownEvent`

`cbRightUpEvent::cbRightUpEvent`

`cbRightUpEvent(const wxPoint& pos, cbDockPane* pPane)`^K

Constructor, taking mouse position and pane.

`cbRightUpEvent::cbRightUpEvent`

`cbrightupeventcbrightupevent`

`rowse00298`

`cbRightUpEvent cbRightUpEvent`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `cbrightupevent')")`

`cbRightUpEvent`

cbRowDragPlugin::cbRowDragPlugin

cbRowDragPlugin(wxFrameLayout* *pLayout*, int *paneMask* = wxALL_PANES)^K

Constructor, taking paren layout frame and pane mask.

cbRowDragPlugin()^K

Default constructor.

^cbRowDragPlugin::cbRowDragPlugin

^cbrowdragplugin**cbrowdragplugin**

^browse00300

^K cbRowDragPlugin cbRowDragPlugin

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`cbrowdragplugin')")

^K cbRowDragPlugin

^K cbRowDragPlugin

`$#+K!cbRowDragPlugin::~~cbRowDragPlugin`

`~cbRowDragPlugin()`^K

Destructor.

`^bRowDragPlugin::~~cbRowDragPlugin`

`^browdragplugindtor`

`^rowse00301`

`KcbRowDragPlugin ~cbRowDragPlugin`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^cbrowdragplugin')")`

`K ~cbRowDragPlugin`

^{\$#+K!}**cbRowDragPlugin::CaptureDCArea**

wxBitmap* CaptureDCArea(**wxDC&** *dc*, **wxRect&** *area*)^K

Helper for drag and drop.

^cbRowDragPlugin::CaptureDCArea

^cbrowdragplugincapturedcare

^browse00302

^K cbRowDragPlugin CaptureDCArea

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `cbrowdragplugin`)"

^K CaptureDCArea

^{\$#+K!}**cbRowDragPlugin::CheckPrevItemInFocus**

void CheckPrevItemInFocus(cbRowInfo* *pRow*, int *iconIdx*)^K

Helper for drag and drop.

^cbRowDragPlugin::CheckPrevItemInFocus

^cbrowdragplugincheckpreviteminfocus

^browse00303

^K cbRowDragPlugin CheckPrevItemInFocus

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbrowdragplugin')")

^K CheckPrevItemInFocus

cbRowDragPlugin::Clone

cbPluginBase* Clone()

Clone function, returning a new instance of this class.

cbRowDragPlugin::Clone

cbrowdragpluginclone

rowse00304

cbRowDragPlugin Clone

enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp',`cbrowdragplugin')")

Clone

`$#+K!cbRowDragPlugin::CollapseRow`

`void CollapseRow(cbRowInfo* pRow)K`

Helper for drag and drop.

`cbRowDragPlugin::CollapseRow`

`cbrowdragplugincollapserow`

`browse00305`

`KcbRowDragPlugin CollapseRow`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `cbrowdragplugin`))`

`K CollapseRow`

cbRowDragPlugin::Draw3DPattern

void Draw3DPattern(wxRect& *inRect*, wxDC& *dc*)^K

Implements 'hard-coded metafile' for Netscape Navigator look.

^cbRowDragPlugin::Draw3DPattern

^cbrowdragplugindraw3dpattern

^browse00306

^K cbRowDragPlugin Draw3DPattern

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")

^K Draw3DPattern

`cbRowDragPlugin::Draw3DRect`

`void Draw3DRect(wxRect& inRect, wxDC& dc, wxBrush& bkBrush)`^K

Implements 'hard-coded metafile' for Netscape Navigator look.

^c`bRowDragPlugin::Draw3DRect`

^c`browdragplugindraw3direct`

^b`rowse00307`

^K`cbRowDragPlugin Draw3DRect`

^E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")`

^K`Draw3DRect`

`cbRowDragPlugin::DrawCollapsedRowIcon`

`void DrawCollapsedRowIcon(int index, wxDC& dc, bool isHighlighted)`^K

Draws collapsed row icon (appearance-dependent).

^cbRowDragPlugin::DrawCollapsedRowIcon

^cbrowdragplugindrawcollapsedrowicon

^browse00308

^K cbRowDragPlugin DrawCollapsedRowIcon

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")

^K DrawCollapsedRowIcon

`$#+K!cbRowDragPlugin::DrawCollapsedRowsBorder`

`void DrawCollapsedRowsBorder(wxDC& dc)K`

Draws collapsed rows border (appearance-dependent).

`cbRowDragPlugin::DrawCollapsedRowsBorder`

`cbrowdragplugindrawcollapsedrowsborder`

`browse00309`

`KcbRowDragPlugin DrawCollapsedRowsBorder`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")`

`KDrawCollapsedRowsBorder`

`cbRowDragPlugin::DrawEmptyRow`

`void DrawEmptyRow(wxDC& dc, wxRect& rowBounds)`^K

Draws empty row (appearance-dependent).

`cbRowDragPlugin::DrawEmptyRow`

`cbrowdragplugindrawemptyrow`

`rowse00310`

`cbRowDragPlugin DrawEmptyRow`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp, `cbrowdragplugin')")`

`DrawEmptyRow`

`cbRowDragPlugin::DrawOrtoRomb`

`void DrawOrtoRomb(wxRect& inRect, wxDC& dc, wxBrush& bkBrush)`^K

Implements 'hard-coded metafile' for Netscape Navigator look.

`cbRowDragPlugin::DrawOrtoRomb`

`cbrowdragplugindrawortoromb`

`rowse00311`

`cbRowDragPlugin DrawOrtoRomb`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `cbrowdragplugin')")`

`DrawOrtoRomb`

`cbRowDragPlugin::DrawRectShade`

`void DrawRectShade(wxRect& inRect, wxDC& dc, int level, wxPen& upperPen, wxPen& lowerPen)`^K

Implements 'hard-coded metafile' for Netscape Navigator look.

^cbRowDragPlugin::DrawRectShade

^cbrowdragplugindrawrectshade

^browse00312

^K cbRowDragPlugin DrawRectShade

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")

^K DrawRectShade

`cbRowDragPlugin::DrawRomb`

`void DrawRomb(wxRect& inRect, wxDC& dc, wxBrush& bkBrush)`^K

Implements 'hard-coded metafile' for Netscape Navigator look.

^cbRowDragPlugin::DrawRomb

^cbrowdragplugindrawromb

^browse00313

^K cbRowDragPlugin DrawRomb

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")

^K DrawRomb

`cbRowDragPlugin::DrawRombShades`

`void DrawRombShades(wxPoint& p1, wxPoint& p2, wxPoint& p3, wxPoint& p4, wxDC& dc)`^K

Implements 'hard-coded metafile' for Netscape Navigator look.

^cbRowDragPlugin::DrawRombShades

^cbrowdragplugindrawrombshades

^browse00314

^K cbRowDragPlugin DrawRombShades

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")

^K DrawRombShades

`cbRowDragPlugin::DrawRowDragHint`

`void DrawRowDragHint(cbRowInfo* pRow, wxDC& dc, bool isHighlighted)`^K

Draws row drag hint (appearance-dependent).

^cbRowDragPlugin::DrawRowDragHint

^cbrowdragplugindrawrowdraghint

^browse00315

^K cbRowDragPlugin DrawRowDragHint

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")

^K DrawRowDragHint

`$#+K!cbRowDragPlugin::DrawRowsDragHintsBorder`

`void DrawRowsDragHintsBorder(wxDC& dc)K`

Draws rows drag hints border (appearance-dependent).

`cbRowDragPlugin::DrawRowsDragHintsBorder`

`cbrowdragplugindrawrowsdraghintsborder`

`browse00316`

`KcbRowDragPlugin DrawRowsDragHintsBorder`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")`

`KDrawRowsDragHintsBorder`

`cbRowDragPlugin::DrawTrianDown`

`void DrawTrianDown(wxRect& inRect, wxDC& dc)`^K

Implements 'hard-coded metafile' for Netscape Navigator look.

`cbRowDragPlugin::DrawTrianDown`

`cbrowdragplugindrawtriandown`

`rowse00317`

`cbRowDragPlugin DrawTrianDown`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp, `cbrowdragplugin')")`

`DrawTrianDown`

`cbRowDragPlugin::DrawTrianRight`

`void DrawTrianRight(wxRect& inRect, wxDC& dc)`^K

Implements 'hard-coded metafile' for Netscape Navigator look.

`cbRowDragPlugin::DrawTrianRight`

`cbrowdragplugindrawtrianright`

`rowse00318`

`cbRowDragPlugin DrawTrianRight`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `cbrowdragplugin')")`

`DrawTrianRight`

`cbRowDragPlugin::DrawTrianUp`

`void DrawTrianUp(wxRect& inRect, wxDC& dc)`^K

Implements 'hard-coded metafile' for Netscape Navigator look.

^cbRowDragPlugin::DrawTrianUp

^cbrowdragplugindrawtrianup

^browse00319

^K cbRowDragPlugin DrawTrianUp

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")

^K DrawTrianUp

`$#+K!cbRowDragPlugin::ExpandRow`

`void ExpandRow(int collapsedIconIdx)K`

Helper for drag and drop.

`cbRowDragPlugin::ExpandRow`

`cbrowdragpluginexpandrow`

`browse00320`

`KcbRowDragPlugin ExpandRow`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `cbrowdragplugin`))`

`KExpandRow`

`$#+K!cbRowDragPlugin::FinishOnScreenDraw`

`void FinishOnScreenDraw()`^K

Helper for drag and drop.

`cbRowDragPlugin::FinishOnScreenDraw`

`cbrowdragpluginfinishonscreendraw`

`browse00321`

^K `cbRowDragPlugin FinishOnScreenDraw`

^E `nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbrowdragplugin')")`

^K `FinishOnScreenDraw`

`$#+K!cbRowDragPlugin::GetCollapsedIconsPos`

`int GetCollapsedIconsPos()`^K

Helper for drag and drop.

^cbRowDragPlugin::GetCollapsedIconsPos

^cbrowdragplugingetcollapsediconspos

^browse00322

^KcbRowDragPlugin GetCollapsedIconsPos

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `cbrowdragplugin`)"

^KGetCollapsedIconsPos

`$#+K!cbRowDragPlugin::GetCollapsedInconRect`

`void GetCollapsedInconRect(int iconIdx, wxRect& rect)K`

Helper for drag and drop.

^cbRowDragPlugin::GetCollapsedInconRect

^cbrowdragplugingetcollapsedinconrect

^browse00323

^KcbRowDragPlugin GetCollapsedInconRect

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `cbrowdragplugin`)"

^KGetCollapsedInconRect

`$#+K!cbRowDragPlugin::GetCollapsedRowIconHeight`

`int GetCollapsedRowIconHeight()`^K

Gets the collapsed row icon height.

`^bRowDragPlugin::GetCollapsedRowIconHeight`

`^browdragplugingetcollapsedrowiconheight`

`^rowse00324`

`^K cbRowDragPlugin GetCollapsedRowIconHeight`

`^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ fl.hlp', ^cbrowdragplugin')")`

`^K GetCollapsedRowIconHeight`

`$#+K!cbRowDragPlugin::GetFirstRow`

`cbRowInfo* GetFirstRow()`^K

Helper for drag and drop.

^cbRowDragPlugin::GetFirstRow

^cbrowdragplugingetfirstrow

^browse00325

^KcbRowDragPlugin GetFirstRow

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `cbrowdragplugin`)"

^KGetFirstRow

`$#+K!cbRowDragPlugin::GetHRowCountForPane`

`int GetHRowCountForPane(cbDockPane* pPane)K`

Helper for drag and drop.

`cbRowDragPlugin::GetHRowCountForPane`

`cbrowdragplugingethrowscountforpane`

`browse00326`

`KcbRowDragPlugin GetHRowCountForPane`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `cbrowdragplugin`)"`

`KGetHRowCountForPane`

`$#+K!cbRowDragPlugin::GetRowDragHintWidth`

`int GetRowDragHintWidth()`^K

Gets the row drag hint width.

^cbRowDragPlugin::GetRowDragHintWidth

^cbrowdragplugingetrowdraghintwidth

^browse00327

^K cbRowDragPlugin GetRowDragHintWidth

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`cbrowdragplugin')")

^K GetRowDragHintWidth

`$#+K!cbRowDragPlugin::GetRowHintRect`

`void GetRowHintRect(cbRowInfo* pRow, wxRect& rect)K`

Helper for drag and drop.

`cbRowDragPlugin::GetRowHintRect`

`cbrowdragplugingetrowhintrect`

`browse00328`

`KcbRowDragPlugin GetRowHintRect`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")`

`KGetRowHintRect`

`cbRowDragPlugin::HitTestCollapsedRowIcon`

`bool HitTestCollapsedRowIcon(int iconIdx, const wxPoint& pos)`^K

Test for the collapsed row icon position.

^c`bRowDragPlugin::HitTestCollapsedRowIcon`

^c`browdragpluginhittestcollapsedrowicon`

^b`rowse00329`

^K`cbRowDragPlugin HitTestCollapsedRowIcon`

^E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")`

^K`HitTestCollapsedRowIcon`

`cbRowDragPlugin::HitTestRowDragHint`

`bool HitTestRowDragHint(cbRowInfo* pRow, const wxPoint& pos)`^K

Test for the row drag hint position.

`cbRowDragPlugin::HitTestRowDragHint`

`cbrowdragpluginhittestrowdraghint`

`rowse00330`

`cbRowDragPlugin HitTestRowDragHint`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `cbrowdragplugin')")`

`HitTestRowDragHint`

`cbRowDragPlugin::InsertDraggedRowBefore`

`void InsertDraggedRowBefore(cbRowInfo* pBeforeRow)`^K

Helper for drag and drop.

`cbRowDragPlugin::InsertDraggedRowBefore`

`cbrowdragplugininsertdraggedrowbefore`

`rowse00331`

^K `cbRowDragPlugin InsertDraggedRowBefore`

^E `nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbrowdragplugin')")`

^K `InsertDraggedRowBefore`

`cbRowDragPlugin::ItemIsInFocus`

`bool ItemIsInFocus()`^K

Helper for drag and drop.

^cbRowDragPlugin::ItemIsInFocus

^cbrowdragpluginitemisinfocus

^browse00332

^K cbRowDragPlugin ItemIsInFocus

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `cbrowdragplugin`)"

^K ItemIsInFocus

cbRowDragPlugin::OnDrawPaneBackground

void OnDrawPaneBackground(cbDrawPaneDecorEvent& event)^K

Handles pane drawing plugin events (appearance-independent logic).

^cbRowDragPlugin::OnDrawPaneBackground

^cbrowdragpluginondrawpanebackground

^browse00333

^K cbRowDragPlugin OnDrawPaneBackground

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `cbrowdragplugin`)"

^K OnDrawPaneBackground

cbRowDragPlugin::OnInitPlugin

void OnInitPlugin()

Called to initialize this plugin.

cbRowDragPlugin::OnInitPlugin

cbrowdragpluginoninitplugin

rowse00334

cbRowDragPlugin OnInitPlugin

enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp',`cbrowdragplugin')")

OnInitPlugin

`cbRowDragPlugin::OnLButtonDown`

`void OnLButtonDown(cbLeftDownEvent& event)`^K

Handles left button down plugin events (appearance-independent logic).

`cbRowDragPlugin::OnLButtonDown`

`cbrowdragpluginonlbuttondown`

`rowse00335`

`cbRowDragPlugin OnLButtonDown`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp',`cbrowdragplugin')")`

`OnLButtonDown`

`$#+K!cbRowDragPlugin::OnLButtonUp`

`void OnLButtonUp(cbLeftUpEvent& event)K`

Handles left button up plugin events (appearance-independent logic).

`cbRowDragPlugin::OnLButtonUp`

`cbrowdragpluginonlbuttonup`

`browse00336`

`KcbRowDragPlugin OnLButtonUp`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `cbrowdragplugin`)"`

`K OnLButtonUp`

`cbRowDragPlugin::OnMouseMove`

`void OnMouseMove(cbMotionEvent& event)`^K

Handles mouse move plugin events (appearance-independent logic).

`cbRowDragPlugin::OnMouseMove`

`cbrowdragpluginonmousemove`

`rowse00337`

`cbRowDragPlugin OnMouseMove`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp',`cbrowdragplugin')")`

`OnMouseMove`

`$#+K!cbRowDragPlugin::PrepareForRowDrag`

`void PrepareForRowDrag()`^K

Helper for drag and drop.

`^bRowDragPlugin::PrepareForRowDrag`

`^browdragpluginprepareforrowdrag`

`^rowse00338`

`^cbRowDragPlugin PrepareForRowDrag`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^cbrowdragplugin')")`

`^ PrepareForRowDrag`

`$#+K!` **cbRowDragPlugin::SetMouseCapture**

void SetMouseCapture(**bool** *captureOn*)^K

Helper for drag and drop.

`^bRowDragPlugin::SetMouseCapture`

`^browdragpluginsetmousecapture`

`^rowse00339`

`^K cbRowDragPlugin SetMouseCapture`

`^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^cbrowdragplugin')")`

`^K SetMouseCapture`

`$#+K!cbRowDragPlugin::SetPaneMargins`

`void SetPaneMargins()`^K

Sets the pane margins.

^cbRowDragPlugin::SetPaneMargins

^cbrowdragpluginsetpanemargins

^browse00340

^KcbRowDragPlugin SetPaneMargins

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")

^KSetPaneMargins

`$#+K!cbRowDragPlugin::ShowDraggedRow`

`void ShowDraggedRow(int offset)K`

Helper for drag and drop.

`cbRowDragPlugin::ShowDraggedRow`

`cbrowdragpluginshowdraggedrow`

`browse00341`

`KcbRowDragPlugin ShowDraggedRow`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `cbrowdragplugin`))`

`KShowDraggedRow`

`$#+K!cbRowDragPlugin::ShowPanelImage`

`void ShowPanelImage()`^K

Helper for drag and drop.

`^bRowDragPlugin::ShowPaneImage`

`^browdragpluginshowpaneimage`

`^rowse00342`

`^cbRowDragPlugin ShowPaneImage`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^cbrowdragplugin')")`

`^ ShowPaneImage`

`$#+K!cbRowDragPlugin::UnhighlightItemInFocus`

`void UnhighlightItemInFocus()`^K

Helper for drag and drop.

^cbRowDragPlugin::UnhighlightItemInFocus

^cbrowdragpluginunhighlightiteminfocus

^browse00343

^K cbRowDragPlugin UnhighlightItemInFocus

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `cbrowdragplugin`)"

^K UnhighlightItemInFocus

`$#+K!cbRowInfo::cbRowInfo`

`cbRowInfo()`^K

Constructor.

`cbRowInfo::cbRowInfo`
`cbrowinfocbrowinfo`
`browse00345`
`K cbRowInfo cbRowInfo`
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowinfo')")`
`K cbRowInfo`

\$#+K!cbRowInfo::~~cbRowInfo

~cbRowInfo()^K

Destructor.

^bRowInfo::~~cbRowInfo

^browinfodtor

^rowse00346

K cbRowInfo ~cbRowInfo

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ fl.hlp', ^cbrowinfo')")

K ~cbRowInfo

`$#+K!cbRowInfo::GetFirstBar`

`cbBarInfo* GetFirstBar()`^K

Returns the first bar.

`^bRowInfo::GetFirstBar`

`^browinfogetfirstbar`

`^rowse00347`

`^cbRowInfo GetFirstBar`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', ^cbrowinfo')")`

`^GetFirstBar`

cbRowLayoutPlugin::cbRowLayoutPlugin

cbRowLayoutPlugin(wxFrameLayout* *pPanel*, int *paneMask* = wxALL_PANES)^K

Constructor taking frame layout pane and pane mask.

cbRowLayoutPlugin()^K

Default constructor.

^cbRowLayoutPlugin::cbRowLayoutPlugin

^cbrowlayoutplugin**cbrowlayoutplugin**

^browse00349

^K cbRowLayoutPlugin cbRowLayoutPlugin

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp',

`cbrowlayoutplugin')")

^K cbRowLayoutPlugin

^K cbRowLayoutPlugin

`cbRowLayoutPlugin::AdjustLengthOfInserted`

`void AdjustLengthOfInserted(cbRowInfo* pRow, cbBarInfo* pTheBar)`^K

Internal helper relating to not-fixed-bars layout.

`cbRowLayoutPlugin::AdjustLengthOfInserted`
`cbrowlayoutpluginadjustlengthofinserted`
`browse00350`
`cbRowLayoutPlugin AdjustLengthOfInserted`
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`
``cbrowlayoutplugin')`
`AdjustLengthOfInserted`

`cbRowLayoutPlugin::ApplyLengthRatios`

`void ApplyLengthRatios(cbRowInfo* pRow)`^K

Internal helper relating to not-fixed-bars layout.

`cbRowLayoutPlugin::ApplyLengthRatios`

`cbrowlayoutpluginapplylengthratios`

`browse00351`

`cbRowLayoutPlugin ApplyLengthRatios`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbrowlayoutplugin')")`

`ApplyLengthRatios`

`$#+K!cbRowLayoutPlugin::CalcRowHeight`

`int CalcRowHeight(cbRowInfo& row)K`

Row layout helper simulating bar 'friction'.

`cbRowLayoutPlugin::CalcRowHeight`

`cbrowlayoutplugincalcrowheight`

`browse00352`

`K cbRowLayoutPlugin CalcRowHeight`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbrowlayoutplugin')")`

`K CalcRowHeight`

^{\$#+K!}**cbRowLayoutPlugin::CheckIfAtTheBoundary**

void CheckIfAtTheBoundary(cbBarInfo* *pTheBar*, cbRowInfo& *rowInfo*)^K

Internal helper relating to not-fixed-bars layout.

[°]bRowLayoutPlugin::CheckIfAtTheBoundary

[°]browlayoutplugincheckifattheboundary

^browse00353

^K cbRowLayoutPlugin CheckIfAtTheBoundary

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',

[`]cbrowlayoutplugin')

^K CheckIfAtTheBoundary

`cbRowLayoutPlugin::DetectBarHandles`

`void DetectBarHandles(cbRowInfo* pRow)`^K

Internal helper relating to not-fixed-bars layout.

`cbRowLayoutPlugin::DetectBarHandles`

`cbrowlayoutplugindetectbarhandles`

`browse00354`

`cbRowLayoutPlugin DetectBarHandles`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`

``cbrowlayoutplugin')`

`DetectBarHandles`

cbRowLayoutPlugin::DoInsertBar

void DoInsertBar(cbBarInfo* *pTheBar*, cbRowInfo& *row*)^K

Insert the bar before the given row.

^cbRowLayoutPlugin::DoInsertBar

^cbrowlayoutplugindoinsertbar

^browse00355

^K cbRowLayoutPlugin DoInsertBar

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',

`cbrowlayoutplugin')")

^K DoInsertBar

`cbRowLayoutPlugin::ExpandNotFixedBars`

`void ExpandNotFixedBars(cbRowInfo* pRow)`^K

Internal helper relating to not-fixed-bars layout.

`cbRowLayoutPlugin::ExpandNotFixedBars`

`cbrowlayoutpluginexpandnotfixedbars`

`rowse00356`

`cbRowLayoutPlugin ExpandNotFixedBars`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbrowlayoutplugin')")`

`ExpandNotFixedBars`

`cbRowLayoutPlugin::FitBarsToRange`

`void FitBarsToRange(int from, int till, cbBarInfo* pTheBar, cbRowInfo* pRow)`^K

Internal helper relating to not-fixed-bars layout.

`cbRowLayoutPlugin::FitBarsToRange`

`cbrowlayoutpluginfitbarstorage`

`browse00357`

^K `cbRowLayoutPlugin FitBarsToRange`

^E `nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbrowlayoutplugin')`

^K `FitBarsToRange`

`$#+K!cbRowLayoutPlugin::GetRowFreeSpace`

`int GetRowFreeSpace(cbRowInfo* pRow)K`

Internal helper relating to not-fixed-bars layout.

`^bRowLayoutPlugin::GetRowFreeSpace`

`^browlayoutpluggingetrowfreespace`

`^rowse00358`

`K cbRowLayoutPlugin GetRowFreeSpace`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbrowlayoutplugin')")`

`K GetRowFreeSpace`

`cbRowLayoutPlugin::InsertBefore`

`void InsertBefore(cbBarInfo* pBeforeBar, cbBarInfo* pTheBar, cbRowInfo& row)`^K

Insert the bar before the given row.

`cbRowLayoutPlugin::InsertBefore`

`cbrowlayoutplugininsertbefore`

`rowse00359`

`cbRowLayoutPlugin InsertBefore`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbrowlayoutplugin')`

`InsertBefore`

`cbRowLayoutPlugin::LayoutItemsVertically`

`void LayoutItemsVertically(cbRowInfo& row)`^K

Row layout helper simulating bar 'friction'.

`cbRowLayoutPlugin::LayoutItemsVertically`
`cbrowlayoutpluginlayoutitemsvertically`
`rowse00360`
`cbRowLayoutPlugin LayoutItemsVertically`
`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`
``cbrowlayoutplugin')`
`LayoutItemsVertically`

`cbRowLayoutPlugin::MinimizeNotFixedBars`

`void MinimizeNotFixedBars(cbRowInfo* pRow, cbBarInfo* pBarToPreserve)`^K

Internal helper relating to not-fixed-bars layout.

`cbRowLayoutPlugin::MinimizeNotFixedBars`

`cbrowlayoutpluginminimizenotfixedbars`

`browse00361`

^K `cbRowLayoutPlugin MinimizeNotFixedBars`

^E `nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbrowlayoutplugin')")`

^K `MinimizeNotFixedBars`

`cbRowLayoutPlugin::OnInsertBar`

`void OnInsertBar(cbInsertBarEvent& event)`^K

Responds to bar insertion event.

`cbRowLayoutPlugin::OnInsertBar`

`cbrowlayoutpluginoninsertbar`

`rowse00362`

`cbRowLayoutPlugin OnInsertBar`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbrowlayoutplugin')")`

`OnInsertBar`

`cbRowLayoutPlugin::OnLayoutRow`

`void OnLayoutRow(cbLayoutRowEvent& event)`^K

Responds to row layout event.

`cbRowLayoutPlugin::OnLayoutRow`

`cbrowlayoutpluginonlayoutrow`

`rowse00363`

`cbRowLayoutPlugin OnLayoutRow`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbrowlayoutplugin')")`

`OnLayoutRow`

`cbRowLayoutPlugin::OnLayoutRows`

`void OnLayoutRows(cbLayoutRowsEvent& event)`^K

Responds to rows layout event.

`cbRowLayoutPlugin::OnLayoutRows`

`cbrowlayoutpluginonlayoutrows`

`rowse00364`

`cbRowLayoutPlugin OnLayoutRows`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbrowlayoutplugin')")`

`OnLayoutRows`

`cbRowLayoutPlugin::OnRemoveBar`

`void OnRemoveBar(cbRemoveBarEvent& event)`^K

Responds to bar removal event.

`cbRowLayoutPlugin::OnRemoveBar`

`cbrowlayoutpluginonremovebar`

`browse00365`

`cbRowLayoutPlugin OnRemoveBar`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbrowlayoutplugin')`

`OnRemoveBar`

cbRowLayoutPlugin::OnResizeRow

void OnResizeRow(cbResizeRowEvent& event)^K

Responds to row resize event.

^cbRowLayoutPlugin::OnResizeRow

^cbrowlayoutpluginonresizerow

^browse00366

^K cbRowLayoutPlugin OnResizeRow

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',

`cbrowlayoutplugin')")

^K OnResizeRow

`cbRowLayoutPlugin::RecalcLengthRatios`

`void RecalcLengthRatios(cbRowInfo* pRow)`^K

Internal helper relating to not-fixed-bars layout.

`cbRowLayoutPlugin::RecalcLengthRatios`

`cbrowlayoutpluginrecalcLengthRatios`

`cbrowse00367`

`cbRowLayoutPlugin RecalcLengthRatios`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbrowlayoutplugin')")`

`RecalcLengthRatios`

`cbRowLayoutPlugin::RelayoutNotFixedBarsAround`

`void RelayoutNotFixedBarsAround(cbBarInfo* pTheBar, cbRowInfo* pRow)`^K

Internal helper relating to not-fixed-bars layout.

`cbRowLayoutPlugin::RelayoutNotFixedBarsAround`

`cbrowlayoutpluginrelayoutnotfixedbarsaround`

`browse00368`

`cbRowLayoutPlugin RelayoutNotFixedBarsAround`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbrowlayoutplugin')")`

`RelayoutNotFixedBarsAround`

`cbRowLayoutPlugin::ShiftLeftTrashold`

`void ShiftLeftTrashold(cbBarInfo* pTheBar, cbRowInfo& row)`^K

Row layout helper simulating bar 'friction'.

`cbRowLayoutPlugin::ShiftLeftTrashold`

`cbrowlayoutpluginshiftlefttrashold`

`browse00369`

`cbRowLayoutPlugin ShiftLeftTrashold`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbrowlayoutplugin')")`

`ShiftLeftTrashold`

`cbRowLayoutPlugin::ShiftRightTrashold`

`void ShiftRightTrashold(cbBarInfo* pTheBar, cbRowInfo& row)`^K

Row layout helper simulating bar 'friction'.

`cbRowLayoutPlugin::ShiftRightTrashold`

`cbrowlayoutpluginshifttrighttrashold`

`rowse00370`

`cbRowLayoutPlugin ShiftRightTrashold`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`

``cbrowlayoutplugin')")`

`ShiftRightTrashold`

`$#+K!cbRowLayoutPlugin::SlideLeftSideBars`

`void SlideLeftSideBars(cbBarInfo* pTheBar)K`

Row layout helper simulating bar 'friction'.

`^bRowLayoutPlugin::SlideLeftSideBars`

`^browlayoutpluginslideleftsidebars`

`^rowse00371`

`^K cbRowLayoutPlugin SlideLeftSideBars`

`^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
^cbrowlayoutplugin')")`

`^K SlideLeftSideBars`

`cbRowLayoutPlugin::SlideRightSideBars`

`void SlideRightSideBars(cbBarInfo* pTheBar)`^K

Row layout helper simulating bar 'friction'.

`cbRowLayoutPlugin::SlideRightSideBars`

`cbrowlayoutpluginsliderightsidebars`

`browse00372`

`cbRowLayoutPlugin SlideRightSideBars`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`

``cbrowlayoutplugin')")`

`SlideRightSideBars`

`$#+K!cbRowLayoutPlugin::StickRightSideBars`

`void StickRightSideBars(cbBarInfo* pToBar)K`

Row layout helper simulating bar 'friction'.

`cbRowLayoutPlugin::StickRightSideBars`

`cbrowlayoutpluginstickrightsidebars`

`browse00373`

`K cbRowLayoutPlugin StickRightSideBars`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`

``cbrowlayoutplugin')")`

`K StickRightSideBars`

cbSimpleCustomizationPlugin::cbSimpleCustomizationPlugin

cbSimpleCustomizationPlugin(wxFrameLayout* *pPanel*, int *paneMask* = *wxALL_PANES*)^K

Constructor, taking parent pane and a pane mask flag.

cbSimpleCustomizationPlugin()^K

Default constructor.

^cbSimpleCustomizationPlugin::cbSimpleCustomizationPlugin
^cbsimplecustomizationplugin**cbSimpleCustomizationPlugin**
^browse00375
^K cbSimpleCustomizationPlugin cbSimpleCustomizationPlugin
^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp',
`cbSimpleCustomizationplugin`")
^K cbSimpleCustomizationPlugin
^K cbSimpleCustomizationPlugin

`cbSimpleCustomizationPlugin::OnCustomizeBar`

`void OnCustomizeBar(cbCustomizeBarEvent& event)`^K

Plugin event handler for cbCustomizeBarEvent.

`cbSimpleCustomizationPlugin::OnCustomizeBar`

`cbSimpleCustomizationPlugin::OnCustomizeBar`

`rowse00376`

`cbSimpleCustomizationPlugin OnCustomizeBar`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbSimpleCustomizationPlugin")")`

`OnCustomizeBar`

`cbSimpleCustomizationPlugin::OnCustomizeLayout`

`void OnCustomizeLayout(cbCustomizeLayoutEvent& event)`^K

Plugin event handler for cbCustomizeLayoutEvent.

`cbSimpleCustomizationPlugin::OnCustomizeLayout`

`cbSimpleCustomizationPluginOnCustomizeLayout`

`rowse00377`

`cbSimpleCustomizationPlugin OnCustomizeLayout`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbSimpleCustomizationPlugin")")`

`OnCustomizeLayout`

`cbSimpleCustomizationPlugin::OnMenuItemSelected`

`void OnMenuItemSelected(wxCommandEvent& event)`^K

Menu event handler.

`cbSimpleCustomizationPlugin::OnMenuItemSelected`

`cbSimpleCustomizationPluginOnMenuItemSelected`

`rowse00378`

`cbSimpleCustomizationPlugin OnMenuItemSelected`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbSimpleCustomizationPlugin")")`

`OnMenuItemSelected`

`cbSimpleUpdatesMgr::cbSimpleUpdatesMgr`

`cbSimpleUpdatesMgr()`^K

Default constructor.

`cbSimpleUpdatesMgr(wxFrameLayout* pPanel)`^K

Constructor taking frame layout panel.

```
cbSimpleUpdatesMgr::cbSimpleUpdatesMgr
cbSimpleUpdatesMgrcbSimpleUpdatesMgr
rowse00380
K cbSimpleUpdatesMgr cbSimpleUpdatesMgr
EnableButton("Up");ChangeButtonBinding("Up", "JumpId( fl.hlp',
`cbSimpleUpdatesMgr')")
K cbSimpleUpdatesMgr
K cbSimpleUpdatesMgr
```


`$#+K!cbSimpleUpdatesMgr::OnBarWillChange`

`void OnBarWillChange(cbBarInfo* pBar, cbRowInfo* pInRow, cbDockPane* pInPane)K`

Notification received from Frame Layout in the order in which they would usually be invoked.

`^bSimpleUpdatesMgr::OnBarWillChange
^bsimpleupdatesmgronbarwillchange
^rowse00381
^K cbSimpleUpdatesMgr OnBarWillChange
^enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
^`cbsimpleupdatesmgr')")
^K OnBarWillChange`

`cbSimpleUpdatesMgr::OnFinishChanges`

`void OnFinishChanges()`^K

Notification received from Frame Layout in the order in which they would usually be invoked.

`cbSimpleUpdatesMgr::OnFinishChanges`

`bsimpleupdatesmgronfinishchanges`

`rowse00382`

`cbSimpleUpdatesMgr OnFinishChanges`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp',`

``bsimpleupdatesmgr')")`

`OnFinishChanges`

cbSimpleUpdatesMgr::OnPaneMarginsWillChange

void OnPaneMarginsWillChange(cbDockPane* *pPane*)

Notification received from Frame Layout in the order in which they would usually be invoked.

```
cbSimpleUpdatesMgr::OnPaneMarginsWillChange
cbSimpleUpdatesMgr::OnPaneMarginsWillChange
rowse00383
cbSimpleUpdatesMgr OnPaneMarginsWillChange
enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbSimpleUpdatesMgr")
OnPaneMarginsWillChange
```

`$#+K!cbSimpleUpdatesMgr::OnPaneWillChange`

`void OnPaneWillChange(cbDockPane* pPane)K`

Notification received from Frame Layout in the order in which they would usually be invoked.

`^bSimpleUpdatesMgr::OnPaneWillChange
^bsimpleupdatesmgronpanewillchange
^rowse00384
^K cbSimpleUpdatesMgr OnPaneWillChange
^enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
^`cbsimpleupdatesmgr')")
^K OnPaneWillChange`

`cbSimpleUpdatesMgr::OnRowWillChange`

`void OnRowWillChange(cbRowInfo* pRow, cbDockPane* pInPane)`^K

Notification received from Frame Layout in the order in which they would usually be invoked.

`cbSimpleUpdatesMgr::OnRowWillChange`
`cbsimpleupdatesmgronrowwillchange`
`rowse00385`
`cbSimpleUpdatesMgr OnRowWillChange`
`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`
``cbsimpleupdatesmgr')`)
`OnRowWillChange`

`$#+K!cbSimpleUpdatesMgr::OnStartChanges`

`void OnStartChanges()`^K

Notification received from Frame Layout in the order in which they would usually be invoked.

`^bSimpleUpdatesMgr::OnStartChanges`

`^bsimpleupdatesmgronstartchanges`

`^rowse00386`

`^K cbSimpleUpdatesMgr OnStartChanges`

`^enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
^cbsimpleupdatesmgr)")`

`^K OnStartChanges`

`$#+K!cbSimpleUpdatesMgr::UpdateNow`

`void UpdateNow()`^K

Refreshes the parts of the frame layoutthat need an update.

`^bSimpleUpdatesMgr::UpdateNow`

`^bsimpleupdatesmgrupdatenow`

`^rowse00387`

`^cbSimpleUpdatesMgr UpdateNow`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
^bsimpleupdatesmgr)")`

`^ UpdateNow`

`$#+K!` **cbSimpleUpdatesMgr::WasChanged**

bool WasChanged(cbUpdateMgrData& *data*, wxRect& *currentBounds*)^K

Helper function.

`^bSimpleUpdatesMgr::WasChanged`

`^bsimpleupdatesmgrwaschanged`

`^rowse00388`

`^ cbSimpleUpdatesMgr WasChanged`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbsimpleupdatesmgr')")`

`^ WasChanged`

cbSizeBarWndEvent::cbSizeBarWndEvent

cbSizeBarWndEvent(cbBarInfo* pBar, cbDockPane* pPane)^K

Constructor, taking bar information and pane.

^cbSizeBarWndEvent::cbSizeBarWndEvent

^cbsizebarwndeventcbsizebarwndevent

^browse00390

^K cbSizeBarWndEvent cbSizeBarWndEvent

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbsizebarwndevent')")

^K cbSizeBarWndEvent

`$#+K!cbStartBarDraggingEvent::cbStartBarDraggingEvent`

`cbStartBarDraggingEvent(cbBarInfo* pBar, const wxPoint& pos, cbDockPane* pPane)K`

Constructor, taking bar information, mouse position, and pane.

`^bStartBarDraggingEvent::cbStartBarDraggingEvent
^bstartbardraggingeventcbstartbardraggingevent
^rowse00392
^K cbStartBarDraggingEvent cbStartBarDraggingEvent
^enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbstartbardraggingevent')")
^K cbStartBarDraggingEvent`

cbStartDrawInAreaEvent::cbStartDrawInAreaEvent

cbStartDrawInAreaEvent(const wxRect& *area*, wxDC** *ppDCForArea*,
cbDockPane* *pPane*)^K

to the obtained buffer-context should be placed Constructor, taking rectangular area,
device context pointer to a pointer, and pane.

^cbStartDrawInAreaEvent::cbStartDrawInAreaEvent

^cbstartdrawinareaeventcbstartdrawinareaevent

^browse00394

^K cbStartDrawInAreaEvent cbStartDrawInAreaEvent

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',

[`]cbstartdrawinareaevent')

^K cbStartDrawInAreaEvent

`cbUpdateMgrData::cbUpdateMgrData`

`cbUpdateMgrData()`^K

Default constructor. Is-dirty flag is set TRUE initially.

^cbUpdateMgrData::cbUpdateMgrData

^cbupdatemgrdatacbupdatemgrdata

^browse00396

^K cbUpdateMgrData cbUpdateMgrData

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbupdatemgrdata')")

^K cbUpdateMgrData

`cbUpdateMgrData::IsDirty`

`bool IsDirty()`^K

Returns the is-dirty flag.

`cbUpdateMgrData::IsDirty`

`cbupdatemgrdataisdirty`

`rowse00397`

`cbUpdateMgrData IsDirty`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbupdatemgrdata')")`

`IsDirty`

`cbUpdateMgrData::SetCustomData`

`void SetCustomData(wxObject* pCustomData)`^K

Set custom data.

^cbUpdateMgrData::SetCustomData

^cbupdatemgrdatasetcustomdata

^browse00398

^K cbUpdateMgrData SetCustomData

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbupdatemgrdata')")

^K SetCustomData

`$#+K!cbUpdateMgrData::SetDirty`

`void SetDirty(bool isDirty = TRUE)K`

Set the dirty flag.

`^bUpdateMgrData::SetDirty`

`^bupdatemgrdatasetdirty`

`^rowse00399`

`KcbUpdateMgrData SetDirty`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbupdatemgrdata')")`

`KSetDirty`

`cbUpdateMgrData::StoreItemState`

`void StoreItemState(const wxRect& boundsInParent)`^K

Store the item state.

^cbUpdateMgrData::StoreItemState

^cbupdatemgrdatastoreitemstate

^browse00400

^K cbUpdateMgrData StoreItemState

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbupdatemgrdata')")

^K StoreItemState

cbUpdatesManagerBase::cbUpdatesManagerBase

cbUpdatesManagerBase(wxFrameLayout* *pPanel*)^K

Constructor taking layout panel.

cbUpdatesManagerBase()^K

Default constructor

cbUpdatesManagerBase::cbUpdatesManagerBase

cbupdatesmanagerbasecbupdatesmanagerbase

rowse00402

cbUpdatesManagerBase cbUpdatesManagerBase

enableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp',

`cbupdatesmanagerbase'))

cbUpdatesManagerBase

cbUpdatesManagerBase

$\$#+K!$ cbUpdatesManagerBase::~~cbUpdatesManagerBase

~cbUpdatesManagerBase()^K

Destructor.

$^{\text{c}}$ bUpdatesManagerBase::~~cbUpdatesManagerBase

$^{\text{c}}$ bupdatesmanagerbasedtor

$^{\text{b}}$ rowse00403

$^{\text{K}}$ cbUpdatesManagerBase ~cbUpdatesManagerBase

**$^{\text{E}}$ nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`cbupdatesmanagerbase')")**

$^{\text{K}}$ ~cbUpdatesManagerBase

`cbUpdatesManagerBase::OnBarWillChange`

`void OnBarWillChange(cbBarInfo* pBar, cbRowInfo* pInRow, cbDockPane* pInPane)`^K

This function receives a notification from the frame layout (in the order in which they would usually be invoked). Custom updates-managers may utilize these notifications to implement a more fine-grained updating strategy.

`cbUpdatesManagerBase::OnBarWillChange`
`cbupdatesmanagerbaseonbarwillchange`
`rowse00404`
`cbUpdatesManagerBase OnBarWillChange`
`enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp',`
``cbupdatesmanagerbase')")`
`OnBarWillChange`

`cbUpdatesManagerBase::OnFinishChanges`

`void OnFinishChanges()`^K

This function receives a notification from the frame layout (in the order in which they would usually be invoked). Custom updates-managers may utilize these notifications to implement a more fine-grained updating strategy.

`cbUpdatesManagerBase::OnFinishChanges`
`cbupdatesmanagerbaseonfinishchanges`
`rowse00405`
`cbUpdatesManagerBase OnFinishChanges`
`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`
``cbupdatesmanagerbase')`
`OnFinishChanges`

`cbUpdatesManagerBase::OnPaneMarginsWillChange`

`void OnPaneMarginsWillChange(cbDockPane* pPane)`^K

This function receives a notification from the frame layout (in the order in which they would usually be invoked). Custom updates-managers may utilize these notifications to implement a more fine-grained updating strategy.

`cbUpdatesManagerBase::OnPaneMarginsWillChange`
`cbupdatesmanagerbaseonpanemarginswillchange`
`rowse00406`
`cbUpdatesManagerBase OnPaneMarginsWillChange`
`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`
``cbupdatesmanagerbase')`
`OnPaneMarginsWillChange`^K

`$#+K!cbUpdatesManagerBase::OnPaneWillChange`

`void OnPaneWillChange(cbDockPane* pPane)K`

This function receives a notification from the frame layout (in the order in which they would usually be invoked). Custom updates-managers may utilize these notifications to implement a more fine-grained updating strategy.

`^bUpdatesManagerBase::OnPaneWillChange`
`^bupdatesmanagerbaseonpanewillchange`
`^rowse00407`
`^K cbUpdatesManagerBase OnPaneWillChange`
`^enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`
`^cbupdatesmanagerbase)")`
`^K OnPaneWillChange`

`cbUpdatesManagerBase::OnRowWillChange`

`void OnRowWillChange(cbRowInfo* pRow, cbDockPane* pInPane)`^K

This function receives a notification from the frame layout (in the order in which they would usually be invoked). Custom updates-managers may utilize these notifications to implement a more fine-grained updating strategy.

`cbUpdatesManagerBase::OnRowWillChange`
`cbupdatesmanagerbaseonrowwillchange`
`rowse00408`
`cbUpdatesManagerBase OnRowWillChange`
`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`
``cbupdatesmanagerbase')")`
`OnRowWillChange`

`$#+K!cbUpdatesManagerBase::OnStartChanges`

`void OnStartChanges()`^K

This function receives a notification from the frame layout (in the order in which they would usually be invoked). Custom updates-managers may utilize these notifications to implement a more fine-grained updating strategy.

`^bUpdatesManagerBase::OnStartChanges`
`^bupdatesmanagerbaseonstartchanges`
`^rowse00409`
`^K cbUpdatesManagerBase OnStartChanges`
`^enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`
`^cbupdatesmanagerbase')")`
`^K OnStartChanges`

`$#+K!cbUpdatesManagerBase::SetLayout`

`void SetLayout(wxFrameLayout* pLayout)K`

Sets the associated layout.

`^bUpdatesManagerBase::SetLayout`

`^bupdatesmanagerbasesetlayout`

`^rowse00410`

`K cbUpdatesManagerBase SetLayout`

`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`

``cbupdatesmanagerbase')")`

`K SetLayout`

`$#+K!cbUpdatesManagerBase::UpdateNow`

`void UpdateNow()`^K

Refreshes parts of the frame layout that need an update.

`^bUpdatesManagerBase::UpdateNow`

`^bupdatesmanagerbaseupdatenow`

`^rowse00411`

`^K cbUpdatesManagerBase UpdateNow`

`^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
^cbupdatesmanagerbase)")`

`^K UpdateNow`

wxDynamicToolBar::wxDynamicToolBar

wxDynamicToolBar()^K

Default constructor.

wxDynamicToolBar(wxWindow* parent, const wxWindowID id, const wxPoint& pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, const long style = wxNO_BORDER, const int orientation = wxVERTICAL, const int RowsOrColumns = 1, const wxString& name = wxToolBarNameStr)^K

Constructor: see the documentation for wxToolBar for details.

^wxDynamicToolBar::wxDynamicToolBar

^wxdynamictoolbarwxdynamictoolbar

^browse00413

^K wxDynamicToolBar wxDynamicToolBar

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp',
`wxdynamictoolbar')")

^K wxDynamicToolBar

^K wxDynamicToolBar

^{\$#+K!}**wxDynamicToolBar::~~wxDynamicToolBar**

~wxDynamicToolBar()^K

Destructor.

^wxDynamicToolBar::~~wxDynamicToolBar

^wxdynamictoolbarctor

^browse00414

^K wxDynamicToolBar ~wxDynamicToolBar

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`xdynamictoolbar')")

^K ~wxDynamicToolBar

^{\$#+K!}**wxDynamicToolBar::AddSeparator**

void AddSeparator(wxWindow* pSeparatorWnd = NULL)^K

Adds a separator. See the documentation for wxToolBar for details.

^wxDynamicToolBar::AddSeparator

^wxdynamictoolbaraddseparator

^browse00415

^K wxDynamicToolBar AddSeparator

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`wxdynamictoolbar')")

^K AddSeparator

wxDynamicToolBar::AddTool

void AddTool(int *toolIndex*, **wxWindow*** *pToolWindow*, **const wxSize&** *size* = *wxDefaultSize*)^K

Adds a tool. See the documentation for wxToolBar for details.

void AddTool(int *toolIndex*, **const wxString&** *imageFileName*, **wxBitmapType** *imageFileType* = *wxBITMAP_TYPE_BMP*, **const wxString&** *labelText* = "", **bool** *alignTextRight* = *FALSE*, **bool** *isFlat* = *TRUE*)^K

Adds a tool. See the documentation for wxToolBar for details.

void AddTool(int *toolIndex*, **wxBitmap** *labelBmp*, **const wxString&** *labelText* = "", **bool** *alignTextRight* = *FALSE*, **bool** *isFlat* = *TRUE*)^K

Adds a tool. See the documentation for wxToolBar for details.

wxToolBarToolBase* **AddTool**(**const int** *toolIndex*, **const wxBitmap&** *bitmap*, **const wxBitmap&** *pushedBitmap* = *wxNullBitmap*, **const bool** *toggle* = *FALSE*, **const long** *xPos* = -1, **const long** *yPos* = -1, **wxObject*** *clientData* = *NULL*, **const wxString&** *helpString1* = "", **const wxString&** *helpString2* = "")^K

Method from wxToolBarBase (for compatibility), only the first two arguments are valid. See the documentation for wxToolBar for details.

^wxDynamicToolBar::AddTool

^wxdynamictoolbaraddtool

^browse00416

^K wxDynamicToolBar AddTool

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp',
`wxdynamictoolbar')")

^K AddTool

^K AddTool

^K AddTool

^K AddTool

\$#+K! **wxDynamicToolBar::Create**

bool Create(wxWindow* *parent*, **const wxWindowID** *id*, **const wxPoint&** *pos* = *wxDefaultPosition*, **const wxSize&** *size* = *wxDefaultSize*, **const long** *style* = *wxNO_BORDER*, **const int** *orientation* = *wxVERTICAL*, **const int** *RowsOrColumns* = 1, **const wxString&** *name* = *wxToolBarNameStr*)^K

Creation function: see the documentation for wxToolBar for details.

WxDynamicToolBar::Create

Wxdynamictoolbarcreate

Browse00417

K wxDynamicToolBar Create

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`wxdynamictoolbar')")

K Create

`wxDynamicToolBar::CreateDefaultLayout`

`LayoutManagerBase* CreateDefaultLayout()`^K

Creates the default layout (BagLayout).

`wxDynamicToolBar::CreateDefaultLayout`

`wxdynamictoolbarcreatedefaultlayout`

`rowse00418`

`wxDynamicToolBar CreateDefaultLayout`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`wxdynamictoolbar')")`

`CreateDefaultLayout`

^{\$#+K!}**wxDynamicToolBar::CreateTool**

wxToolBarToolBase* CreateTool(wxControl* control)^K

Creates a toolbar tool.

wxToolBarToolBase* CreateTool(int id, const wxBitmap& bitmap1, const wxBitmap& bitmap2, bool toggle, wxObject* clientData, const wxString& shortHelpString, const wxString& longHelpString)^K

Creates a toolbar tool.

^wxDynamicToolBar::CreateTool

^wxdynamictoolbarcreatetool

^browse00419

^K wxDynamicToolBar CreateTool

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp',`wxdynamictoolbar')")

^K CreateTool

^K CreateTool

wxDynamicToolBar::DoDeleteTool

bool DoDeleteTool(**size_t** *pos*, **wxToolBarToolBase*** *tool*)^K

Deletes a tool. The tool is still in `m_tools` list when this function is called, and it will only be deleted from it if it succeeds.

^wxDynamicToolBar::DoDeleteTool
^wxdynamictoolbardeletetool
^browse00420
^K wxDynamicToolBar DoDeleteTool
^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`wxdynamictoolbar')")
^K DoDeleteTool

wxDynamicToolBar::DoEnableTool

void DoEnableTool(wxToolBarToolBase* *tool*, bool *enable*)^K

Called when the tools enabled flag changes.

^wxDynamicToolBar::DoEnableTool

^wxdynamictoolbar::doenabletool

^browse00421

^K wxDynamicToolBar DoEnableTool

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`wxdynamictoolbar')")

^K DoEnableTool

^{\$#+K!}**wxDynamicToolBar::DoInsertTool**

bool DoInsertTool(**size_t** *pos*, **wxToolBarToolBase*** *tool*)^K

Inserts a tool at the given position.

^wxDynamicToolBar::DoInsertTool

^wxdynamictoolbar^doinser^ttool

^browse00422

^K wxDynamicToolBar DoInsertTool

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',

`wxdynamictoolbar')")

^K DoInsertTool

^{\$#+K!}**wxDynamicToolBar::DoSetToggle**

void DoSetToggle(wxToolBarToolBase* *tool*, bool *toggle*)^K

Called when the tools 'can be toggled' flag changes.

^wxDynamicToolBar::DoSetToggle

^wxdynamictoolbar::dosettoggle

^browse00423

^K wxDynamicToolBar DoSetToggle

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`wxdynamictoolbar')")

^K DoSetToggle

wxDynamicToolBar::DoToggleTool

void DoToggleTool(wxToolBarToolBase* *tool*, bool *toggle*)^K

Called when the tool is toggled.

^wxDynamicToolBar::DoToggleTool

^wxdynamictoolbar_dot_toggle_tool

^browse00424

^K wxDynamicToolBar DoToggleTool

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`wxdynamictoolbar')")

^K DoToggleTool

\$#+K! **wxDynamicToolBar::DrawSeparator**

void DrawSeparator(wxDynToolInfo& *info*, wxDC& *dc*)^K

Draws a separator. The default implementation draws a shaded line.

wxDynamicToolBar::DrawSeparator

wxdynamictoolbardrawseparator

browse00425

K wxDynamicToolBar DrawSeparator

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`wxdynamictoolbar')")

K DrawSeparator

\$#+K! **wxDynamicToolBar::EnableTool**

void EnableTool(**const int** *toolIndex*, **const bool** *enable = TRUE*)^K

Enables or disables the given tool.

wxDynamicToolBar::EnableTool

wxdynamictoolbarenabletool

browse00426

K wxDynamicToolBar EnableTool

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`wxdynamictoolbar')")

K EnableTool

`$#+KK!` **wxDynamicToolBar::FindToolForPosition**

wxToolBarToolBase* FindToolForPosition(wxCoord x, wxCoord y) const

Finds a tool for the given position.

`wxDynamicToolBar::FindToolForPosition`

`wxdynamictoolbarfindtoolforposition`

`browse00427`

`K wxDynamicToolBar FindToolForPosition`

`K FindToolForPosition`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`

``wxdynamictoolbar')")`

wxDynamicToolBar::GetPreferredDim

void GetPreferredDim(const wxSize& *givenDim*, wxSize& *prefDim*)^K

Returns the preferred dimension, taking the given dimension and a reference to the result.

^wxDynamicToolBar::GetPreferredDim

^wxdynamictoolbargetpreferredDim

^browse00428

^K wxDynamicToolBar GetPreferredDim

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`wxdynamictoolbar')")

^K GetPreferredDim

^{\$#+K!}**wxDynamicToolBar::GetToolInfo**

wxDynToolInfo* GetToolInfo(int *toolIndex*)^K

Returns tool information for the given tool index.

^wxDynamicToolBar::GetToolInfo

^wxdynamictoolbar::gettoolinfo

^browse00429

^K wxDynamicToolBar GetToolInfo

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`wxdynamictoolbar')")

^K GetToolInfo

\$#+K! **wxDynamicToolBar::Layout**

bool Layout()^K

Performs layout. See definitions of orientation types.

wxDynamicToolBar::Layout

wxdynamictoolbarlayout

browse00430

K wxDynamicToolBar Layout

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`wxdynamictoolbar')")

K Layout

`wxDynamicToolBar::OnEraseBackground`

`void OnEraseBackground(wxEraseEvent& event)`^K

Responds to background erase events. Currently does nothing.

`wxDynamicToolBar::OnEraseBackground`

`wxdynamictoolbaronerasebackground`

`rowse00431`

`wxDynamicToolBar OnEraseBackground`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`wxdynamictoolbar')")`

`OnEraseBackground`

\$#+K! **wxDynamicToolBar::OnPaint**

void OnPaint(wxPaintEvent& *event*)^K

Responds to paint events, drawing separators.

wxDynamicToolBar::OnPaint

wxdynamictoolbaronpaint

browse00432

K wxDynamicToolBar OnPaint

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`wxdynamictoolbar')")

K OnPaint

\$#+K! **wxDynamicToolBar::OnSize**

void OnSize(wxSizeEvent& *event*)^K

Responds to size events, calling Layout.

wxDynamicToolBar::OnSize

wxdynamictoolbaronsize

browse00433

K wxDynamicToolBar OnSize

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`wxdynamictoolbar')")

K OnSize

wxDynamicToolBar::Realize

bool Realize()

Overriden from wxToolBarBase; does nothing.

wxDynamicToolBar::Realize

wxdynamictoolbarrealize

rowse00434

wxDynamicToolBar Realize

enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`wxdynamictoolbar')")

Realize

`$#+K!` **wxDynamicToolBar::RemveTool**

void RemveTool(int *toolIndex*)^K

Removes the given tool. Misspelt in order not to clash with a similar function in the base class.

`wxDynamicToolBar::RemveTool`
`wxdynamictoolbarremvetool`
`rowse00435`
`K wxDynamicToolBar RemveTool`
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`
``wxdynamictoolbar')`
`K RemveTool`

\$#+K! **wxDynamicToolBar::SetLayout**

void SetLayout(LayoutManagerBase* *pLayout*)^K

Sets the layout for this toolbar.

^wxDynamicToolBar::SetLayout
^wxdynamictoolbarsetLayout
^browse00436
^K wxDynamicToolBar SetLayout
^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`wxdynamictoolbar')")
^K SetLayout

`$#+K! wxDynamicToolBar::SizeToolWindows`

`void SizeToolWindows()`^K

Internal function for sizing tool windows.

^w`xDynamicToolBar::SizeToolWindows`

^w`xdynamictoolbarstoolwindows`

^b`rowse00437`

^K `wxDynamicToolBar SizeToolWindows`

^E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`wxdynamictoolbar')")`

^K `SizeToolWindows`

`wxFrameLayout::wxFrameLayout`

`wxFrameLayout(wxWindow* pParentFrame, wxWindow* pFrameClient = NULL, bool activateNow = TRUE)`^K

Constructor, taking parent window, the (MDI) client of the parent if there is one, and flag specifying whether to activate the layout.

`wxFrameLayout()`^K

Default constructor, used only for serialization.

^w`wxFrameLayout::wxFrameLayout`

^w`wxframelayouthwxframelayout`

^b`rowse00440`

^K `wxFrameLayout wxFrameLayout`

^E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`wxframelayout')")`

^K `wxFrameLayout`

^K `wxFrameLayout`

`$#+K!wxFrameLayout::~~wxFrameLayout`

`~wxFrameLayout()`^K

Destructor. It does not destroy the bar windows.

`wxFrameLayout::~~wxFrameLayout`

`wxframelayouthtor`

`browse00441`

`K wxFrameLayout ~wxFrameLayout`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ fl.hlp', `wxframelayout')")`

`K ~wxFrameLayout`

`$#+K! wxFrameLayout::Activate`

`void Activate()`^K

Activate can be called after some other layout has been deactivated, and this one must take over the current contents of the frame window. Effectively hooks itself to the frame window, re-displays all non-hidden bar windows and repaints the decorations.

^wxFrameLayout::Activate

^wxframelayoutactivate

^browse00442

^K wxFrameLayout Activate

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")

^K Activate

`$#+K!` **wxFrameLayout::AddBar**

```
void AddBar(wxWindow* pBarWnd, const cbDimInfo& dimInfo, int alignment =  
FL_ALIGN_TOP, int rowNo = 0, int columnPos = 0, const wxString& name = "bar",  
bool spyEvents = FALSE, int state = wxCBAR_DOCKED_HORIZONTALLY)K
```

Adds bar information to the frame layout. The appearance of the layout is not refreshed immediately; RefreshNow() can be called if necessary. Notes: the argument pBarWnd can be NULL, resulting in bar decorations to be drawn around the empty rectangle (filled with default background colour). Argument dimInfo can be reused for adding any number of bars, since it is not used directly - instead its members are copied. If the dimensions handler is present, its instance is shared (reference counted). The dimension handler should always be allocated on the heap. pBarWnd is the window to be managed. dimInfo contains dimension information. alignment is a value such as FL_ALIGN_TOP. rowNo is the vertical position or row in the pane (if in docked state). columnPos is the horizontal position within the row in pixels (if in docked state). name is a name by which the bar can be referred in layout customization dialogs. If spyEvents is TRUE, input events for the bar should be "spied" in order to forward unhandled mouse clicks to the frame layout, for example to enable easy druggability of toolbars just by clicking on their interior regions. For widgets like text/tree control this value should be FALSE, since there's no certain way to detect whether the event was actually handled. state is the initial state, such as wxCBAR_DOCKED_HORIZONTALLY, wxCBAR_FLOATING, wxCBAR_HIDDEN.

`wxFrameLayout::AddBar`

`wxframelayouaddbar`

`browse00443`

`K wxFrameLayout AddBar`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `wxframelayout')")`

`K AddBar`

\$#+K! **wxFrameLayout::AddPlugin**

void AddPlugin(wxClassInfo* pPInfo, int paneMask = wxALL_PANES)^K

An advanced methods for plugin configuration using their dynamic class information, for example CLASSINFO(pluginClass). First checks if the plugin of the given class is already "hooked up". If not, adds it to the top of the plugins chain.

wxFrameLayout::AddPlugin

wxframelayoutaddplugin

browse00444

K wxFrameLayout AddPlugin

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")

K AddPlugin

^{\$#+K!}**wxFrameLayout::AddPluginBefore**

**void AddPluginBefore(wxClassInfo* pNextPInfo, wxClassInfo* pPInfo, int
paneMask = wxALL_PANES)^K**

First checks if the plugin of the given class is already hooked. If so, removes it, and then inserts it into the chain before the plugin of the class given by pNextPInfo. Note: this method is handy in some cases where the order of the plugin-chain could be important, for example when one plugin overrides some functionality of another already-hooked plugin, so that the former plugin should be hooked before the one whose functionality is being overridden.

^wxFrameLayout::AddPluginBefore

^wxframelayoutaddpluginbefore

^browse00445

^K wxFrameLayout AddPluginBefore

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframelayout`)"

^K AddPluginBefore

\$#+K! **wxFrameLayout::ApplyBarProperties**

void ApplyBarProperties(cbBarInfo* pBar)^K

Reflects changes in bar information structure visually. For example, moves the bar, changes its dimension information, or changes the pane to which it is docked.

wxFrameLayout::ApplyBarProperties

wxframelayoutapplybarproperties

browse00446

K wxFrameLayout ApplyBarProperties

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframelayout`)"

K ApplyBarProperties

\$#+K! **wxFrameLayout::CanReparent**

bool CanReparent()^K

Returns TRUE if the platform allows reparenting. This may not return TRUE for all platforms. Reparenting allows control bars to be floated.

wxFrameLayout::CanReparent

wxframelayoucanreparent

browse00447

K wxFrameLayout CanReparent

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframelayout`)"

K CanReparent

`wxFrameLayout::CaptureEventsForPane`

`void CaptureEventsForPane(cbDockPane* toPane)`^K

Called by plugins; also captures the mouse in the parent frame.

^wxFrameLayout::CaptureEventsForPane

^wxframelayoutcaptureeventsforpane

^browse00448

^K wxFrameLayout CaptureEventsForPane

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxframelayout')")

^K CaptureEventsForPane

\$#+K! **wxFrameLayout::CaptureEventsForPlugin**

void CaptureEventsForPlugin(cbPluginBase* *pPlugin*)^K

Captures user input events for the given plugin. Input events are: mouse movement, mouse clicks, keyboard input.

wxFrameLayout::CaptureEventsForPlugin

wxframelayoutcaptureeventsforplugin

browse00449

K wxFrameLayout CaptureEventsForPlugin

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxframelayout')")

K CaptureEventsForPlugin

`$#+K! wxFrameLayout::CreateCursors`

`void CreateCursors()`^K

Creates the cursors.

`wxFrameLayout::CreateCursors`

`wxframelayoucreatecursors`

`browse00450`

`K wxFrameLayout CreateCursors`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ fl.hlp', `wxframelayout')")`

`K CreateCursors`

^{\$#+K!}**wxFrameLayout::CreateUpdatesManager**

cbUpdatesManagerBase* CreateUpdatesManager()^K

Returns a new cbGCUpdatesMgr object.

^wxFrameLayout::CreateUpdatesManager

^wxframelayoucreateupdatesmanager

^browse00451

^K wxFrameLayout CreateUpdatesManager

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ fl.hlp', `wxframelayout')")

^K CreateUpdatesManager

wxFrameLayout::Deactivate

void Deactivate()

Deactivate unhooks itself from frame window, and hides all non-hidden windows. Note: two frame layouts should not be active at the same time in the same frame window, since it would cause messy overlapping of bar windows from both layouts.

wxFrameLayout::Deactivate

wxframelayoutdeactivate

rowse00452

wxFrameLayout Deactivate

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `wxframelayout')")

Deactivate

`wxFrameLayout::DestroyBarWindows`

`void DestroyBarWindows()`^K

Destroys the bar windows.

^wxFrameLayout::DestroyBarWindows

^wxframelayoutdestroybarwindows

^browse00453

^K wxFrameLayout DestroyBarWindows

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`wxframelayout')")

^K DestroyBarWindows

^{\$#+K!}**wxFrameLayout::DoSetBarState**

void DoSetBarState(**cbBarInfo*** *pBar*)^K

Applies the state to the window objects.

^wxFrameLayout::DoSetBarState

^wxframelayouthdosetbarstate

^browse00454

^K wxFrameLayout DoSetBarState

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")

^K DoSetBarState

`$#+K!wxFrameLayout::EnableFloating`

`void EnableFloating(bool enable = TRUE)K`

Enables floating behaviour. By default floating of control bars is on.

`wxFrameLayout::EnableFloating`

`wxframelayouenablefloating`

`rowse00455`

`K wxFrameLayout EnableFloating`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`wxframelayout')")`

`K EnableFloating`

`wxFrameLayout::FindBarByName`

`cbBarInfo* FindBarByName(const wxString& name)`^K

Finds the bar in the framelayout, by name.

^wxFrameLayout::FindBarByName

^wxframelayoutfindbarbyname

^browse00456

^K wxFrameLayout FindBarByName

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframelayout`)")

^K FindBarByName

`wxFrameLayout::FindBarByWindow`

`cbBarInfo* FindBarByWindow(const wxWindow* pWnd)`^K

Finds the bar in the framelayout, by window.

^wxFrameLayout::FindBarByWindow

^wxframelayoutfindbarbywindow

^browse00457

^K wxFrameLayout FindBarByWindow

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframelayout`)"

^K FindBarByWindow

`wxFrameLayout::FindPlugin`

`cbPluginBase* FindPlugin(wxClassInfo* pPInfo)`^K

Finds a plugin with the given class, or returns NULL if a plugin of the given class is not hooked.

`wxFrameLayout::FindPlugin`

`wxframelayoutfindplugin`

`rowse00458`

`wxFrameLayout FindPlugin`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `wxframelayout')")`

`FindPlugin`

`$#+K!wxFrameLayout::FirePluginEvent`

`void FirePluginEvent(cbPluginEvent& event)K`

This function should be used instead of passing the event to the ProcessEvent method of the top-level plugin directly. This method checks if events are currently captured and ensures that plugin-event is routed correctly.

`wxFrameLayout::FirePluginEvent`

`wxframelayoufirepluginevent`

`browse00459`

`K wxFrameLayout FirePluginEvent`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")`

`K FirePluginEvent`

`wxFrameLayout::ForwardMouseEvent`

`void ForwardMouseEvent(wxMouseEvent& event, cbDockPane* pToPane, int eventType)`^K

Delegated from "bar-spy".

^wxFrameLayout::ForwardMouseEvent

^wxframelayoutforwardmouseevent

^browse00460

^K wxFrameLayout ForwardMouseEvent

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxframelayout')")

^K ForwardMouseEvent

wxFrameLayout::GetBarPane

cbDockPane* GetBarPane(cbBarInfo* pBar)^K

Returns the pane to which the given bar belongs.

^wxFrameLayout::GetBarPane

^wxframelayougetbarpane

^browse00461

^K wxFrameLayout GetBarPane

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ fl.hlp', `wxframelayout')")

^K GetBarPane

\$#+K! **wxFrameLayout::GetBars**

BarArrayT& GetBars()^K

Gets an array of bars.

wxFrameLayout::GetBars

wxframelayougetbars

browse00462

K wxFrameLayout GetBars

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframelayout`)"

K GetBars

\$#+K! **wxFrameLayout::GetClientHeight**

int GetClientHeight()^K

Returns the client height.

^wxFrameLayout::GetClientHeight

^wxframelayougetclientheight

^browse00463

^K wxFrameLayout GetClientHeight

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ fl.hlp', `wxframelayout')")

^K GetClientHeight

`wxFrameLayout::GetClientRect`

`wxRect& GetClientRect()`^K

Returns the client's rectangle.

^w`wxFrameLayout::GetClientRect`

^w`wxframelayougetclientrect`

^b`rowse00464`

^K`wxFrameLayout GetClientRect`

^E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`wxframelayout')")`

^K`GetClientRect`

\$#+K! **wxFrameLayout::GetClientWidth**

int GetClientWidth()^K

Returns the client width.

^wxFrameLayout::GetClientWidth

^wxframelayougetclientwidth

^browse00465

^K wxFrameLayout GetClientWidth

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframelayout`)"

^K GetClientWidth

`wxFrameLayout::GetFrameClient`

`wxWindow* GetFrameClient()`^K

Returns the frame client, or NULL if not present.

^wxFrameLayout::GetFrameClient

^wxframelayougetframeclient

^browse00466

^K wxFrameLayout GetFrameClient

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframelayout`)"

^K GetFrameClient

\$#+K! **wxFrameLayout::GetPane**

cbDockPane* **GetPane**(int *alignment*)^K

Returns a pane for the given alignment. See pane alignment types.

wxFrameLayout::GetPane

wxframelayougetpane

browse00467

K wxFrameLayout GetPane

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")

K GetPane

\$#+K! **wxFrameLayout::GetPaneProperties**

void GetPaneProperties(**cbCommonPaneProperties&** *props*, **int** *alignment* = *FL_ALIGN_TOP*)^K

Gets the pane properties for the given alignment.

wxFrameLayout::GetPaneProperties

wxframelayougetpaneproperties

browse00468

K wxFrameLayout GetPaneProperties

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframelayout`)"

K GetPaneProperties

`$#+K! wxFrameLayout::GetPanesArray`

`cbDockPane** GetPanesArray()`^K

Returns an array of panes. Used by update managers.

`wxFrameLayout::GetPanesArray`

`wxframelayougetpanesarray`

`browse00469`

`K wxFrameLayout GetPanesArray`

`E nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")`

`K GetPanesArray`

`wxFrameLayout::GetParentFrame`

`wxWindow& GetParentFrame()`^K

Returns the parent frame.

^wxFrameLayout::GetParentFrame

^wxframelayougetparentframe

^browse00470

^K wxFrameLayout GetParentFrame

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframelayout`)"

^K GetParentFrame

`$#+K! wxFrameLayout::GetPrevClientRect`

`wxRect& GetPrevClientRect()`^K

Returns the previous client window rectangle.

^wxFrameLayout::GetPrevClientRect

^wxframelayougetprevclientrect

^browse00471

^K wxFrameLayout GetPrevClientRect

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")

^K GetPrevClientRect

`$#+K! wxFrameLayout::GetTopPlugin`

`cbPluginBase& GetTopPlugin()`^K

Returns the current top-level plugin (the one that receives events first, except if input events are currently captured by some other plugin).

`wxFrameLayout::GetTopPlugin`

`wxframelayougettopplugin`

`browse00472`

`K wxFrameLayout GetTopPlugin`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")`

`K GetTopPlugin`

`wxFrameLayout::GetUpdatesManager`

`cbUpdatesManagerBase& GetUpdatesManager()`^K

Returns a reference to the updates manager. Note: in future, the updates manager will become a normal plugin.

^wxFrameLayout::GetUpdatesManager

^wxframelayoutgetupdatesmanager

^browse00473

^K wxFrameLayout GetUpdatesManager

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframelayout`)"

^K GetUpdatesManager

`$#+K!wxFrameLayout::HasTopPlugin`

`bool HasTopPlugin()`^K

Returns true if there is a top plugin.

`wxFrameLayout::HasTopPlugin`

`wxframelayouthastopplugin`

`rowse00474`

`wxFrameLayout HasTopPlugin`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `wxframelayout')")`

`HasTopPlugin`

wxFrameLayout::HideBarWindows

void HideBarWindows()^K

Hides the bar windows, and also the client window if present.

^wxFrameLayout::HideBarWindows

^wxframelayouthidebarwindows

^browse00475

^K wxFrameLayout HideBarWindows

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`wxframelayout')")

^K HideBarWindows

\$#+K! **wxFrameLayout::HitTestPane**

bool HitTestPane(cbDockPane* *pPane*, int *x*, int *y*)^K

Returns TRUE if the position is within the given pane.

wxFrameLayout::HitTestPane

wxframelayouthittestpane

browse00476

K wxFrameLayout HitTestPane

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframelayout`)"

K HitTestPane

^{\$#+K!}**wxFrameLayout::HitTestPanels**

cbDockPane* HitTestPanels(**const wxRect&** *rect*, **cbDockPane*** *pCurPane*)^K

Returns the pane for which the rectangle hit test succeeds, giving preference to the given pane if supplied.

^wxFrameLayout::HitTestPanels

^wxframelayouthittestpanes

^browse00477

^K wxFrameLayout HitTestPanels

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframelayout`)"

^K HitTestPanels

`$#+K!wxFrameLayout::HookUpToFrame`

`void HookUpToFrame()`^K

Hooks the layout up to the frame (pushes the layout onto the frame's event handler stack).

`wxFrameLayout::HookUpToFrame`

`wxframelayouthookuptoframe`

`rowse00478`

`wxFrameLayout HookUpToFrame`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `wxframelayout')")`

`HookUpToFrame`

\$#+K! **wxFrameLayout::InverseVisibility**

void InverseVisibility(cbBarInfo* *pBar*)^K

Toggles the bar between visible and hidden.

wxFrameLayout::InverseVisibility

wxframelayouinversevisibility

browse00479

K wxFrameLayout InverseVisibility

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`wxframelayout')")

K InverseVisibility

\$#+K! **wxFrameLayout::LocateBar**

bool LocateBar(**cbBarInfo*** *pBarInfo*, **cbRowInfo**** *ppRow*, **cbDockPane**** *ppPane*)^K

The purpose of this function is unknown.

wxFrameLayout::LocateBar

wxframelayoutlocatebar

browse00480

K wxFrameLayout LocateBar

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframelayout`)"

K LocateBar

\$#+K! **wxFrameLayout::OnActivate**

void OnActivate(wxActivateEvent& *event*)^K

Handles activation events. Currently does nothing.

wxFrameLayout::OnActivate

wxframelayoutonactivate

browse00481

K wxFrameLayout OnActivate

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`wxframelayout')")

K OnActivate

`$#+K! wxFrameLayout::OnEraseBackground`

`void OnEraseBackground(wxEraseEvent& event)K`

Handles background erase events. Currently does nothing.

`wxFrameLayout::OnEraseBackground`

`wxframelayoutonerasebackground`

`browse00482`

`K wxFrameLayout OnEraseBackground`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")`

`K OnEraseBackground`

`wxFrameLayout::OnIdle`

`void OnIdle(wxIdleEvent& event)`

Handles idle events.

`wxFrameLayout::OnIdle`

`wxframelayoutonidle`

`rowse00483`

`wxFrameLayout OnIdle`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `wxframelayout')")`

`OnIdle`

wxFrameLayout::OnKillFocus

void OnKillFocus(wxFocusEvent& *event*)^K

Handles focus kill events. Currently does nothing.

^wxFrameLayout::OnKillFocus

^wxframelayoutonkillfocus

^browse00484

^K wxFrameLayout OnKillFocus

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframelayout`)"

^K OnKillFocus

`wxFrameLayout::OnLButtonDown`

`void OnLButtonDown(wxMouseEvent& event)`^K

Event handler for a left down button event.

^wxFrameLayout::OnLButtonDown

^wxframelayoutonlbuttondown

^browse00485

^K wxFrameLayout OnLButtonDown

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`wxframelayout')")

^K OnLButtonDown

`wxFrameLayout::OnLButtonUp`

`void OnLButtonUp(wxMouseEvent& event)`^K

Event handler for a left button up event.

`wxFrameLayout::OnLButtonUp`

`wxframelayoutonlbuttonup`

`rowse00486`

`wxFrameLayout OnLButtonUp`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `wxframelayout')")`

`OnLButtonUp`

wxFrameLayout::OnLDbIClick

void OnLDbIClick(wxMouseEvent& *event*)^K

Event handler for a left doubleclick button event.

^wxFrameLayout::OnLDbIClick

^wxframelayoutonldblick

^browse00487

^K wxFrameLayout OnLDbIClick

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")

^K OnLDbIClick

^{\$#+K!}**wxFrameLayout::OnMouseMove**

void OnMouseMove(wxMouseEvent& *event*)^K

Event handler for a mouse move event.

^wxFrameLayout::OnMouseMove

^wxframelayoutonmousemove

^browse00488

^K wxFrameLayout OnMouseMove

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframelayout`)"

^K OnMouseMove

wxFrameLayout::OnPaint

void OnPaint(wxPaintEvent& event)^K

Handles paint events, calling PaintPane for each pane.

^wxFrameLayout::OnPaint

^wxframelayoutonpaint

^browse00489

^K wxFrameLayout OnPaint

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ fl.hlp', `wxframelayout')")

^K OnPaint

wxFrameLayout::OnRButtonDown

void OnRButtonDown(wxMouseEvent& event)^K

Event handler for a right button down event.

^wxFrameLayout::OnRButtonDown

^wxframelayoutonrbuttondown

^browse00490

^K wxFrameLayout OnRButtonDown

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`wxframelayout')")

^K OnRButtonDown

wxFrameLayout::OnRButtonUp

void OnRButtonUp(wxMouseEvent& *event*)^K

Event handler for a right button up event.

^wxFrameLayout::OnRButtonUp

^wxframelayoutonrbuttonup

^browse00491

^K wxFrameLayout OnRButtonUp

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframelayout`)"

^K OnRButtonUp

`$#+K! wxFrameLayout::OnSetFocus`

`void OnSetFocus(wxFocusEvent& event)K`

Handles focus set events. Currently does nothing.

`wxFrameLayout::OnSetFocus`

`wxframelayoutonsetfocus`

`browse00492`

`K wxFrameLayout OnSetFocus`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframelayout`))`

`K OnSetFocus`

`wxFrameLayout::OnSize`

`void OnSize(wxSizeEvent& event)`^K

Event handler for a size event.

`wxFrameLayout::OnSize`

`wxframelayoutonsize`

`rowse00493`

`wxFrameLayout OnSize`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `wxframelayout')")`

`OnSize`

wxFrameLayout::PopAllPlugins

void PopAllPlugins()

Pop all plugins.

wxFrameLayout::PopAllPlugins

wxframelayoutpopallplugins

rowse00494

wxFrameLayout PopAllPlugins

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `wxframelayout')")

PopAllPlugins

\$#+K! **wxFrameLayout::PopPlugin**

void PopPlugin()^K

Similar to wxWindow's "push/pop-event-handler" methods, execept that the plugin is deleted upon "popping".

wxFrameLayout::PopPlugin

wxframelayoutpopplugin

browse00495

K wxFrameLayout PopPlugin

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframelayout`)"

K PopPlugin

`wxFrameLayout::PositionClientWindow`

`void PositionClientWindow()`^K

Called to apply the calculated layout to window objects.

^wxFrameLayout::PositionClientWindow

^wxframelayouthpositionclientwindow

^browse00496

^K wxFrameLayout PositionClientWindow

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ fl.hlp', `wxframelayout')")

^K PositionClientWindow

\$#+K! **wxFrameLayout::PositionPanes**

void PositionPanes()^K

Called to apply the calculated layout to window objects.

wxFrameLayout::PositionPanes

wxframelayouthpositionpanes

browse00497

K wxFrameLayout PositionPanes

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ fl.hlp', `wxframelayout')")

K PositionPanes

^{\$#+K!}**wxFrameLayout::PushDefaultPlugins**

void PushDefaultPlugins()^K

Adds the default plugins. These are cbPaneDrawPlugin, cbRowLayoutPlugin, cbBarDragPlugin, cbAntiflickerPlugin, cbSimpleCustomizePlugin. This method is automatically invoked if no plugins were found upon firing of the first plugin-event, i.e. when wxFrameLayout configures itself.

^wxFrameLayout::PushDefaultPlugins

^wxframelayouthpushdefaultplugins

^browse00498

^K wxFrameLayout PushDefaultPlugins

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframelayout`)"

^K PushDefaultPlugins

wxFrameLayout::PushPlugin

void PushPlugin(cbPluginBase* pPugin)^K

Similar to wxWindow's "push/pop-event-handler" methods, execept that the plugin is deleted upon "popping".

^wxFrameLayout::PushPlugin

^wxframelayouthpushplugin

^browse00499

^K wxFrameLayout PushPlugin

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframelayout`)"

^K PushPlugin

`$#+K!wxFrameLayout::RecalcLayout`

`void RecalcLayout(bool repositionBarsNow = FALSE)K`

Recalculates the layout of panes, and all bars/rows in each pane.

`wxFrameLayout::RecalcLayout`

`wxframelayoutrcalclayout`

`browse00500`

`KwxFrameLayout RecalcLayout`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframelayout`)"`

`K RecalcLayout`

\$#+K! **wxFrameworkLayout::RedockBar**

bool RedockBar(**cbBarInfo*** *pBar*, **const wxRect&** *shapeInParent*, **cbDockPane*** *pToPane* = *NULL*, **bool** *updateNow* = *TRUE*)^K

RedockBar can be used for repositioning existing bars. The given bar is first removed from the pane it currently belongs to, and inserted into the pane, which "matches" the given rectangular area. If *pToPane* is not *NULL*, the bar is docked to this given pane. To dock a bar which is floating, use the `wxFrameworkLayout::DockBar` method.

WxFrameworkLayout::RedockBar

Wxframeworklayoutredockbar

Browse00501

K wxFrameworkLayout RedockBar

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframeworklayout')")

K RedockBar

\$#+K! **wxFrameLayout::RefreshNow**

void RefreshNow(**bool** *recalcLayout* = *TRUE*)^K

Recalculates layout and performs on-screen update of all panes.

wxFrameLayout::RefreshNow

wxframelayourefreshnow

browse00502

K wxFrameLayout RefreshNow

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")

K RefreshNow

`wxFrameLayout::ReleaseEventsFromPane`

`void ReleaseEventsFromPane(cbDockPane* fromPane)`^K

Called by plugins; also releases mouse in the parent frame.

^wxFrameLayout::ReleaseEventsFromPane

^wxframelayoutheaseeventsfrompane

^browse00503

^K wxFrameLayout ReleaseEventsFromPane

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")

^K ReleaseEventsFromPane

wxFrameLayout::ReleaseEventsFromPlugin

void ReleaseEventsFromPlugin(cbPluginBase* *pPlugin*)^K

Releases user input events for the given plugin. Input events are: mouse movement, mouse clicks, keyboard input

^wxFrameLayout::ReleaseEventsFromPlugin

^wxframelayoutheaseeventsfromplugin

^browse00504

^K wxFrameLayout ReleaseEventsFromPlugin

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^ fl.hlp', `wxframelayout')")

^K ReleaseEventsFromPlugin

wxFrameLayout::RemoveBar

void RemoveBar(cbBarInfo* pBar)

Removes the bar from the layout permanently, and hides its corresponding window if present.

wxFrameLayout::RemoveBar

wxframelayoutremovebar

rowse00505

wxFrameLayout RemoveBar

enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `wxframelayout')")

RemoveBar

\$#+K! **wxFrameLayout::RemovePlugin**

void RemovePlugin(wxCClassInfo* pPInfo)^K

Checks if the plugin of the given class is hooked, and removes it if found.

wxFrameLayout::RemovePlugin

wxframelayouremoveplugin

browse00506

K wxFrameLayout RemovePlugin

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframelayout`)"

K RemovePlugin

`wxFrameLayout::ReparentWindow`

`void ReparentWindow(wxWindow* pChild, wxWindow* pNewParent)`^K

Reparents pChild to have parent pNewParent.

^wxFrameLayout::ReparentWindow

^wxframelayoutrparentwindow

^browse00507

^K wxFrameLayout ReparentWindow

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframelayout`)"

^K ReparentWindow

`wxFrameLayout::RepositionFloatedBar`

`void RepositionFloatedBar(cbBarInfo* pBar)`^K

Applies the calculated layout to a floating bar.

`wxFrameLayout::RepositionFloatedBar`

`wxframelayoutrpositionfloatedbar`

`rowse00508`

`wxFrameLayout RepositionFloatedBar`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`wxframelayout')")`

`RepositionFloatedBar`

\$#+K! **wxFrameLayout::RouteMouseEvent**

void RouteMouseEvent(**wxMouseEvent&** *event*, **int** *pluginEvtType*)^K

Routes the mouse event to the appropriate pane.

wxFrameLayout::RouteMouseEvent

wxframelayoutroutemouseevent

browse00509

K wxFrameLayout RouteMouseEvent

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframelayout`)"

K RouteMouseEvent

^{\$#+K!}**wxFrameLayout::SetBarState**

void SetBarState(**cbBarInfo*** *pBar*, **int** *newState*, **bool** *updateNow*)^K

Changes the bar's docking state (see possible control bar states).

^wxFrameLayout::SetBarState

^wxframelayoutsetbarstate

^browse00510

^K wxFrameLayout SetBarState

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")

^K SetBarState

\$#+K! **wxFrameLayout::SetFrameClient**

void SetFrameClient(**wxWindow*** *pFrameClient*)^K

Passes the client window (e.g. MDI client window) to be controlled by frame layout, the size and position of which should be adjusted to be surrounded by controlbar panes, whenever the frame is resized or the dimensions of control panes change.

wxFrameLayout::SetFrameClient

wxframelayoutsetframeclient

browse00511

K wxFrameLayout SetFrameClient

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")

K SetFrameClient

\$#+K! **wxFrameLayout::SetMargins**

void SetMargins(*int top*, *int bottom*, *int left*, *int right*, *int paneMask* = *wxALL_PANES*)^K

Sets the margins for the given panes. The margins should go into `cbCommonPaneProperties` in the future. Note: this method should be called before any custom plugins are attached.

wxFrameLayout::SetMargins

wxframelayoutsetmargins

browse00512

K wxFrameLayout SetMargins

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframelayout`)"

K SetMargins

\$#+K! **wxFrameLayout::SetPaneBackground**

void SetPaneBackground(const wxColour& *colour*)^K

Sets the pane background colour.

^wxFrameLayout::SetPaneBackground

^wxframelayousetpanebackground

^browse00513

^K wxFrameLayout SetPaneBackground

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframelayout`)"

^K SetPaneBackground

\$#+K! **wxFrameLayout::SetPaneProperties**

void SetPaneProperties(const cbCommonPaneProperties& *props*, int *paneMask* = *wxALL_PANES*)^K

Sets the pane properties for the given alignment. Note: changing properties of panes does not result immediate on-screen update.

wxFrameLayout::SetPaneProperties

wxframelayoutsetpaneproperties

browse00514

K wxFrameLayout SetPaneProperties

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")

K SetPaneProperties

`wxFrameLayout::SetTopPlugin`

`void SetTopPlugin(cbPluginBase* pPlugin)`^K

Hooking custom plugins to frame layout. Note: when hooking one plugin on top of the other, use `SetNextHandler` or similar methods of `wxEvtHandler` class to compose the chain of plugins, than pass the left-most handler in this chain to the above methods (assuming that events are delegated from left-most towards right-most handler). This sceneario is very inconvenient and "low-level", so use the `Add/Push/PopPlugin` methods instead.

^w`wxFrameLayout::SetTopPlugin`

^w`wxframelayousettopplugin`

^b`rowse00515`

^K `wxFrameLayout SetTopPlugin`

^E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")`

^K `SetTopPlugin`

^{\$#+K!}**wxFrameLayout::SetUpdatesManager**

void SetUpdatesManager(cbUpdatesManagerBase* *pUMgr*)^K

Destroys the previous manager if any, and sets the new one.

^wxFrameLayout::SetUpdatesManager

^wxframelayoutsetupdatesmanager

^browse00516

^K wxFrameLayout SetUpdatesManager

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")

^K SetUpdatesManager

\$#+K! **wxFrameLayout::ShowFloatedWindows**

void ShowFloatedWindows(bool *show*)^K

Shows all floated windows.

wxFrameLayout::ShowFloatedWindows

wxframelayouthowfloatedwindows

browse00517

K wxFrameLayout ShowFloatedWindows

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframelayout`)"

K ShowFloatedWindows

`wxFrameLayout::UnhookFromFrame`

`void UnhookFromFrame()`^K

Unhooks the layout from the frame.

^w`wxFrameLayout::UnhookFromFrame`

^w`wxframelayoutunhookfromframe`

^b`rowse00518`

^K`wxFrameLayout UnhookFromFrame`

^E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframelayout`))`

^K`UnhookFromFrame`

^{\$#+K!}**wxFrameManager::wxFrameManager**

wxFrameManager()^K

^wxFrameManager::wxFrameManager

^wxframemanagerwxframemanager

^browse00520

^K wxFrameManager wxFrameManager

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframemanager')")

^K wxFrameManager

$\$ \# + K!$ **wxFrameManager::~wxFrameManager**

~wxFrameManager()^K

^wxFrameManager::~wxFrameManager

^wxframemanagerdctor

^browse00521

^K wxFrameManager ~wxFrameManager

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframemanager')")

^K ~wxFrameManager

`$#+K! wxFrameManager::ActivateView`

`void ActivateView(wxFrameView* pFrmView)K`

`void ActivateView(int viewNo)K`

^wxFrameManager::ActivateView

^wxframemanageractivateview

^browse00522

^K wxFrameManager ActivateView

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`wxframemanager')")

^K ActivateView

^K ActivateView

`wxFrameManager::AddView`

`void AddView(wxFrameView* pFrmView)`^K

^w`wxFrameManager::AddView`

^w`wxframemanageraddview`

^b`rowse00523`

^K`wxFrameManager AddView`

^E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframemanager')")`

^K`AddView`

`$#+K!wxFrameManager::DeactivateCurrentView`

`void DeactivateCurrentView()`^K

^wxFrameManager::DeactivateCurrentView

^wxframemanagerdeactivatecurrentview

^browse00524

^K wxFrameManager DeactivateCurrentView

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`wxframemanager')")

^K DeactivateCurrentView

`wxFrameManager::DestroyViews`

`void DestroyViews()`

`wxFrameManager::DestroyViews`

`wxframemanagerdestroyviews`

`rowse00525`

`wxFrameManager DestroyViews`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `wxframemanager')")`

`DestroyViews`

`wxFrameManager::DoSerialize`

`void DoSerialize(wxObjectStorage& store)`^K

`wxFrameManager::DoSerialize`

`wxframemanagerdoserialize`

`rowse00526`

`wxFrameManager DoSerialize`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp, `wxframemanager')")`

`DoSerialize`

^{\$#+K!}**wxFrameManager::EnableMenusForView**

void EnableMenusForView(wxFrameView* *pView*, **bool** *enable*)^K

^wxFrameManager::EnableMenusForView

^wxframemanagerenablemenusforview

^browse00527

^K wxFrameManager EnableMenusForView

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframemanager')")

^K EnableMenusForView

$\$#+K!$ wxFrameManager::GetActiveView

wxFrameView* GetActiveView()^K

^wxFrameManager::GetActiveView

^wxframemanagergetactiveview

^browse00528

^K wxFrameManager GetActiveView

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframemanager`)"

^K GetActiveView

`$#+K!wxFrameManager::GetActiveViewNo`

`int GetActiveViewNo()`^K

^wxFrameManager::GetActiveViewNo

^wxframemanagergetactiveviewno

^browse00529

^K wxFrameManager GetActiveViewNo

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`wxframemanager')")

^K GetActiveViewNo

$\$#+K!$ wxFrameManager::GetActiveViewNode

wxNode* GetActiveViewNode()^K

^wxFrameManager::GetActiveViewNode

^wxframemanagergetactiveviewnode

^browse00530

^K wxFrameManager GetActiveViewNode

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframemanager')")

^K GetActiveViewNode

$\$#+K!$ wxFrameManager::GetClientWindow

wxWindow* GetClientWindow()^K

^wxFrameManager::GetClientWindow

^wxframemanagergetclientwindow

^browse00531

^K wxFrameManager GetClientWindow

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframemanager')")

^K GetClientWindow

$\$#+K!$ wxFrameManager::GetObjectStore

wxObjectStorage& GetObjectStore()^K

^wxFrameManager::GetObjectStore

^wxframemanagergetobjectstore

^browse00532

^K wxFrameManager GetObjectStore

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframemanager`)"

^K GetObjectStore

\$#+K! **wxFrameManager::GetParentFrame**

wxFrame* GetParentFrame()^K

synonyms

^wxFrameManager::GetParentFrame

^wxframemanagergetparentframe

^browse00533

^K wxFrameManager GetParentFrame

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframemanager')")

^K GetParentFrame

\$#+K! wxFrameManager::GetParentWindow

wxWindow* GetParentWindow()^K

^wxFrameManager::GetParentWindow

^wxframemanagergetparentwindow

^browse00534

^K wxFrameManager GetParentWindow

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframemanager')")

^K GetParentWindow

$\$#+K!$ wxFrameManager::GetView

wxFrameView* GetView(int *viewNo*)^K

^wxFrameManager::GetView

^wxframemanagergetview

^browse00535

^K wxFrameManager GetView

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframemanager')")

^K GetView

^{\$#+K!}**wxFrameManager::GetViewNo**

int GetViewNo(wxFrameView* *pView*)^K

^wxFrameManager::GetViewNo

^wxframemanagergetviewno

^browse00536

^K wxFrameManager GetViewNo

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframemanager')")

^K GetViewNo

`$#+K!wxFrameManager::Init`

`void Init(wxWindow* pMainFrame, const wxString& settingsFile = "")K`

if file name is empty, views are are not saved/loaded

`wxFrameManager::Init`

`wxframemanagerinit`

`browse00537`

`KwxFrameManager Init`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframemanager')")`

`K Init`

`$#+K!wxFrameManager::ReloadViews`

`bool ReloadViews()`^K

^wxFrameManager::ReloadViews

^wxframemanagerreloadviews

^browse00538

^K wxFrameManager ReloadViews

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxframemanager`)"

^K ReloadViews

\$#+K! **wxFrameManager::RemoveView**

void RemoveView(wxFrameView* *pFrmView*)^K

^wxFrameManager::RemoveView

^wxframemanagerremoveview

^browse00539

^K wxFrameManager RemoveView

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframemanager')")

^K RemoveView

\$#+K!wxFrameManager::SaveViewsNow

void SaveViewsNow()^K

^wxFrameManager::SaveViewsNow

^wxframemanagersaveviewsnow

^browse00540

^K wxFrameManager SaveViewsNow

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxframemanager')")

^K SaveViewsNow

^{\$#+K!}**wxFrameManager::SetClnetWindow**

void SetClnetWindow(wxWindow* *pFrameClient*)^K

^wxFrameManager::SetClnetWindow

^wxframemanagersetclinetwindow

^browse00541

^K wxFrameManager SetClnetWindow

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframemanager')")

^K SetClnetWindow

`wxFrameManager::SyncAllMenus`

`void SyncAllMenus()`

`wxFrameManager::SyncAllMenus`

`wxframemanagersyncallmenus`

`rowse00542`

`wxFrameManager SyncAllMenus`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp, `wxframemanager')")`

`SyncAllMenus`

\$#+K!wxFrameManager::ViewsAreLoaded

bool ViewsAreLoaded()^K

^wxFrameManager::ViewsAreLoaded

^wxframemanagerviewsareloaded

^browse00543

^K wxFrameManager ViewsAreLoaded

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`wxframemanager')")

^K ViewsAreLoaded

^{\$#+K!}**GarbageCollector::GarbageCollector**

GarbageCollector()^K

Default constructor.

^GarbageCollector::GarbageCollector

^garbagecollectorgarbagecollector

^browse00545

^K GarbageCollector GarbageCollector

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `garbagecollector')")

^K GarbageCollector

^{\$#+K!}**GarbageCollector::~~GarbageCollector**

~GarbageCollector()^K

Destructor.

^GarbageCollector::~~GarbageCollector

^garbagecollectordtor

^browse00546

^K GarbageCollector ~GarbageCollector

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `garbagecollector')")

^K ~GarbageCollector

^{\$#+K!}**GarbageCollector::AddDependency**

void AddDependency(void* *pObj*, void* *pDependsOnObj*)^K

Prepare data for garbage collection.

^GarbageCollector::AddDependency

^garbagecollectoradddependency

^browse00547

^K GarbageCollector AddDependency

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `garbagecollector')")

^K AddDependency

^{\$#+K!}**GarbageCollector::AddObject**

void AddObject(void* *pObj*, int *refCnt* = 1)^K

Prepare data for garbage collection.

^GarbageCollector::AddObject

^garbagecollectoraddobject

^browse00548

^K GarbageCollector AddObject

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `garbagecollector')")

^K AddObject

^{\$#+K!}**GarbageCollector::ArrangeCollection**

void ArrangeCollection()^K

Executes garbage collection algorithm.

^GarbageCollector::ArrangeCollection

^garbagecollectorarrangecollection

^browse00549

^K GarbageCollector ArrangeCollection

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `garbagecollector')")

^K ArrangeCollection

`$#+K!` GarbageCollector::DestroyItemList

`void DestroyItemList(wxList& /st)`^K

Destroys a list of items.

^GarbageCollector::DestroyItemList

^garbagecollectordestroyitemlist

^browse00550

^K GarbageCollector DestroyItemList

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `garbagecollector')")

^K DestroyItemList

^{\$#+K!}**GarbageCollector::FindItemNode**

wxNode* FindItemNode(void* *pForObj*)^K

Internal method for finding a node.

^GarbageCollector::FindItemNode

^garbagecollectorfinditemnode

^browse00551

^K GarbageCollector FindItemNode

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `garbagecollector')")

^K FindItemNode

^{\$#+K!}**GarbageCollector::FindReferenceFreeItemNode**

wxNode* FindReferenceFreeItemNode()^K

Internal method for findind and freeing a node.

^GarbageCollector::FindReferenceFreeItemNode

^garbagecollectorfindreferencefreeitemnode

^browse00552

^K GarbageCollector FindReferenceFreeItemNode

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `garbagecollector')")

^K FindReferenceFreeItemNode

\$#+K! GarbageCollector::GetCycledObjects

wxList& GetCycledObjects()^K

Get cycled objects.

^GarbageCollector::GetCycledObjects

^garbagecollectorgetcycledobjects

^browse00553

^K GarbageCollector GetCycledObjects

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `garbagecollector`)"

^K GetCycledObjects

^{\$#+K!}**GarbageCollector::GetRegularObjects**

wxList& GetRegularObjects()^K

Accesses the results of the algorithm.

^GarbageCollector::GetRegularObjects

^garbagecollectorgetregularobjects

^browse00554

^K GarbageCollector GetRegularObjects

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `garbagecollector')")

^K GetRegularObjects

^{\$#+K!}**GarbageCollector::RemoveReferencesToNode**

void RemoveReferencesToNode(wxNode* *pItemNode*)^K

Remove references to this node.

^GarbageCollector::RemoveReferencesToNode

^garbagecollectorremovereferencestonode

^browse00555

^K GarbageCollector RemoveReferencesToNode

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `garbagecollector)")

^K RemoveReferencesToNode

GarbageCollector::Reset

void Reset()

Removes all data from the garbage collector.

GarbageCollector::Reset

garbagecollectorreset

rowse00556

GarbageCollector Reset

enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `garbagecollector')")

Reset

`GarbageCollector::ResolveReferences`

`void ResolveReferences()`^K

Internal method for resolving references.

^G`GarbageCollector::ResolveReferences`

^g`garbagecollectorresolveReferences`

^b`rowse00557`

^K`GarbageCollector ResolveReferences`

^E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^ fl.hlp', `garbagecollector")`

^K`ResolveReferences`

\$#+K!LayoutManagerBase::~~LayoutManagerBase

~LayoutManagerBase()^K

Destructor.

^LLayoutManagerBase::~~LayoutManagerBase

^Ilayoutmanagerbasedtor

^browse00559

^K LayoutManagerBase ~LayoutManagerBase

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',

[`]layoutmanagerbase')

^K ~LayoutManagerBase

LayoutManagerBase::Layout

**void Layout(const wxSize& *parentDim*, wxSize& *resultingDim*,
wxLayoutItemArrayT& *items*, int *horizGap*, int *vertGap*)**^K

Constructor.

^LLayoutManagerBase::Layout

^Ilayoutmanagerbaselayout

^browse00560

^K LayoutManagerBase Layout

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',

`layoutmanagerbase')")

^K Layout

`wxNewBitmapButton::wxNewBitmapButton`

`wxNewBitmapButton(const wxBitmap& labelBitmap = wxNullBitmap, const wxString& labelText = "", int alignText = NB_ALIGN_TEXT_BOTTOM, bool isFlat = TRUE, int firedEventType = wxEVT_COMMAND_MENU_SELECTED, int marginX = 2, int marginY = 2, int textToLabelGap = 2, bool isSticky = FALSE)`^K

Constructor.

`wxNewBitmapButton(const wxString& bitmapFileName, const wxBitmapType bitmapFileType = wxBITMAP_TYPE_BMP, const wxString& labelText = "", int alignText = NB_ALIGN_TEXT_BOTTOM, bool isFlat = TRUE, int firedEventType = wxEVT_COMMAND_MENU_SELECTED, int marginX = 2, int marginY = 2, int textToLabelGap = 2, bool isSticky = FALSE)`^K

Use this constructor if buttons have to be persistent

`wxNewBitmapButton::wxNewBitmapButton`
`wxnewbitmapbuttonwxnewbitmapbutton`
`rowse00562`
`wxNewBitmapButton wxNewBitmapButton`
`enableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp',`
``wxnewbitmapbutton')`
`wxNewBitmapButton`
`wxNewBitmapButton`

^{\$#+K!}**wxNewBitmapButton::~~wxNewBitmapButton**

~wxNewBitmapButton()^K

Destructor.

^wxNewBitmapButton::~~wxNewBitmapButton

^wxnewbitmapbuttonondtor

^browse00563

^K wxNewBitmapButton ~wxNewBitmapButton

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`wxnewbitmapbutton')")

^K ~wxNewBitmapButton

`wxNewBitmapButton::DestroyLabels`

`void DestroyLabels()`^K

Internal function for destroying labels.

`wxNewBitmapButton::DestroyLabels`

`wxnewbitmapbuttondestroylabels`

`rowse00564`

`wxNewBitmapButton DestroyLabels`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`wxnewbitmapbutton')")`

`DestroyLabels`

`wxNewBitmapButton::DrawDecorations`

`void DrawDecorations(wxDC& dc)`^K

Draws the decorations.

`wxNewBitmapButton::DrawDecorations`

`wxnewbitmapbuttondrawdecorations`

`rowse00565`

`wxNewBitmapButton DrawDecorations`

`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`wxnewbitmapbutton')")`

`DrawDecorations`

\$#+K! **wxNewBitmapButton::DrawLabel**

void DrawLabel(wxDC& *dc*)^K

Draws the label.

^wxNewBitmapButton::DrawLabel

^wxnewbitmapbuttondrawlabel

^browse00566

^K wxNewBitmapButton DrawLabel

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp",

`wxnewbitmapbutton')")

^K DrawLabel

\$#+K! **wxNewBitmapButton::DrawShade**

void DrawShade(int *outerLevel*, **wxDC&** *dc*, **wxPen&** *upperLeftSidePen*, **wxPen&** *lowerRightSidePen*)^K

Draws shading on the button.

wxNewBitmapButton::DrawShade
wxnewbitmapbuttondrawshade
browse00567
K wxNewBitmapButton DrawShade
EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`wxnewbitmapbutton')")
K DrawShade

\$#+K! **wxNewBitmapButton::GetStateLabel**

wxBitmap* GetStateLabel()^K

Returns the label that matches the current button state.

wxNewBitmapButton::GetStateLabel

wxnewbitmapbuttongetstatelabel

browse00568

K wxNewBitmapButton GetStateLabel

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`wxnewbitmapbutton')")

K GetStateLabel

`$#+K!` **wxNewBitmapButton::IsInWindow**

bool IsInWindow(int x, int y)^K

Returns TRUE if the given point is in the window.

`w`xNewBitmapButton::IsInWindow

`w`xnewbitmapbuttonisinwindow

`b`rowse00569

`K` wxNewBitmapButton IsInWindow

`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',

```wxnewbitmapbutton')")

`K` IsInWindow

\$#+K! **wxNewBitmapButton::OnEraseBackground**

**void OnEraseBackground(wxEraseEvent& *event*)**<sup>K</sup>

Responds to an erase background event.

---

wxNewBitmapButton::OnEraseBackground

wxnewbitmapbuttononerasebackground

browse00570

K wxNewBitmapButton OnEraseBackground

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',

`wxnewbitmapbutton')

K OnEraseBackground

`$#+K!` **wxNewBitmapButton::OnKillFocus**

**void OnKillFocus(wxFocusEvent& *event*)**<sup>K</sup>

Responds to a kill focus event.

---

`w`xNewBitmapButton::OnKillFocus

`w`xnewbitmapbuttononkillfocus

`b`rowse00571

`K` wxNewBitmapButton OnKillFocus

`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',  
`wxnewbitmapbutton')")

`K` OnKillFocus

**wxNewBitmapButton::OnLButtonDown**

**void OnLButtonDown(wxMouseEvent& *event*)**<sup>K</sup>

Responds to a left mouse button down event.

---

<sup>w</sup>xNewBitmapButton::OnLButtonDown

<sup>w</sup>xnewbitmapbuttononlbuttondown

<sup>b</sup>rowse00572

<sup>K</sup> wxNewBitmapButton OnLButtonDown

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',  
`wxnewbitmapbutton')")

<sup>K</sup> OnLButtonDown

\$#+K! **wxNewBitmapButton::OnLButtonUp**

**void OnLButtonUp(wxMouseEvent& *event*)**<sup>K</sup>

Responds to a left mouse button up event.

---

wxNewBitmapButton::OnLButtonUp

wxnewbitmapbuttononlbuttonup

browse00573

K wxNewBitmapButton OnLButtonUp

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp",

`wxnewbitmapbutton')

K OnLButtonUp

<sup>\$#+K!</sup>**wxNewBitmapButton::OnMouseMove**

**void OnMouseMove(wxMouseEvent& *event*)**<sup>K</sup>

Responds to a mouse move event.

---

<sup>w</sup>xNewBitmapButton::OnMouseMove

<sup>w</sup>xnewbitmapbuttononmousemove

<sup>b</sup>rowse00574

<sup>K</sup> wxNewBitmapButton OnMouseMove

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',  
`wxnewbitmapbutton')")

<sup>K</sup> OnMouseMove

\$#+K! **wxNewBitmapButton::OnPaint**

**void OnPaint(wxPaintEvent& *event*)**<sup>K</sup>

Responds to a paint event.

---

wxNewBitmapButton::OnPaint

wxnewbitmapbuttononpaint

browse00575

K wxNewBitmapButton OnPaint

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',

`wxnewbitmapbutton')")

K OnPaint

\$#+K! **wxNewBitmapButton::OnSize**

**void OnSize(wxSizeEvent& *event*)**<sup>K</sup>

Responds to a size event.

---

wxNewBitmapButton::OnSize

wxnewbitmapbuttononsize

browse00576

K wxNewBitmapButton OnSize

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',

`wxnewbitmapbutton')

K OnSize



`$#+K!` **wxNewBitmapButton::RenderAllLabelImages**

**void RenderAllLabelImages()**<sup>K</sup>

Renders label images.

---

`w`xNewBitmapButton::RenderAllLabelImages

`w`xnewbitmapbuttonrenderalllabelimages

`b`rowse00577

`K` wxNewBitmapButton RenderAllLabelImages

`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',

```wxnewbitmapbutton')

`K` RenderAllLabelImages

`$#+K!wxNewBitmapButton::RenderLabelImage`

`void RenderLabelImage(wxBitmap* & destBmp, wxBitmap* srcBmp, bool isEnabled = TRUE, bool isPressed = FALSE)K`

Renders the label image.

`wxNewBitmapButton::RenderLabelImage`

`wxnewbitmapbuttonrenderlabelimage`

`browse00578`

`K wxNewBitmapButton RenderLabelImage`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`

``wxnewbitmapbutton')")`

`K RenderLabelImage`

wxNewBitmapButton::RenderLabelImages

void RenderLabelImages()^K

Renders label images.

^wxNewBitmapButton::RenderLabelImages

^wxnewbitmapbuttonrenderlabelimages

^browse00579

^K wxNewBitmapButton RenderLabelImages

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',

`wxnewbitmapbutton')")

^K RenderLabelImages

`$#+K!wxNewBitmapButton::Reshape`

`void Reshape()`^K

This function should be called after Create. It renders the labels, having reloaded the button image if necessary.

`wxNewBitmapButton::Reshape`

`wxnewbitmapbuttonreshape`

`browse00580`

`K wxNewBitmapButton Reshape`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`

``wxnewbitmapbutton')")`

`K Reshape`

\$#+K! **wxNewBitmapButton::SetAlignments**

void SetAlignments(*int alignText = NB_ALIGN_TEXT_BOTTOM*, **int** *marginX = 2*, **int** *marginY = 2*, **int** *textToLabelGap = 2*)^K

Sets the text alignment and margins.

WxNewBitmapButton::SetAlignments

Wxnewbitmapbuttonsetalignments

browse00581

K wxNewBitmapButton SetAlignments

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp",

`wxnewbitmapbutton')

K SetAlignments

\$#+K! **wxNewBitmapButton::SetLabel**

void SetLabel(const wxBitmap& *labelBitmap*, const wxString& *labelText* = "")K

Sets the label and optionally label text.

wxNewBitmapButton::SetLabel
wxnewbitmapbuttonsetlabel
browse00582
K wxNewBitmapButton SetLabel
EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',
`wxnewbitmapbutton')")
K SetLabel

`$#+K! wxToolWindow::wxToolWindow`

`wxToolWindow()`^K

Default constructor.

`wxToolWindow::wxToolWindow`

`wxtoolwindowwxtoolwindow`

`browse00585`

`K wxToolWindow wxToolWindow`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxtoolwindow')")`

`K wxToolWindow`

`$#+K! wxToolWindow::~~wxToolWindow`

`~wxToolWindow()`^K

Destructor.

`wxToolWindow::~~wxToolWindow`

`wxtoolwindowdtor`

`browse00586`

`K wxToolWindow ~wxToolWindow`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxtoolwindow')")`

`K ~wxToolWindow`

wxToolWindow::AddMiniButton

void AddMiniButton(cbMiniButton* pBtn)

Adds a button. Buttons are added in right-to-left order.

wxToolWindow::AddMiniButton

wxtoolwindowaddminibutton

rowse00587

wxToolWindow AddMiniButton

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `wxtoolwindow')")

AddMiniButton

\$#+K! **wxToolWindow::AdjustRectPos**

void AdjustRectPos(const wxRect& *original*, const wxSize& *newDim*, wxRect& *newRect*)^K

Helper function.

^wxToolWindow::AdjustRectPos

^wxtoolwindowadjustrectpos

^browse00588

^K wxToolWindow AdjustRectPos

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxtoolwindow')")

^K AdjustRectPos

`wxToolWindow::CalcResizedRect`

`void CalcResizedRect(wxRect& rect, wxPoint& delta, const wxSize& minDim)`^K

Calculate resized rectangle.

^wxToolWindow::CalcResizedRect

^wxtoolwindowcalcre-sizedrect

^browse00589

^K wxToolWindow CalcResizedRect

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`wxtoolwindow')")

^K CalcResizedRect

\$#+K! **wxToolWindow::DrawHintRect**

void DrawHintRect(const wxRect& r)^K

Draws the hint rectangle.

wxToolWindow::DrawHintRect

wxtoolwindowdrawhintrect

browse00590

K wxToolWindow DrawHintRect

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`wxtoolwindow')")

K DrawHintRect

\$#+K! **wxToolWindow::GetClient**

wxWindow* **GetClient()**^K

Returns the client window.

wxToolWindow::GetClient

wxtoolwindowgetclient

browse00591

K wxToolWindow GetClient

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxtoolwindow`)"

K GetClient

`$#+K! wxToolWindow::GetMinimalWndDim`

`wxSize GetMinimalWndDim()`^K

Helper function.

`wxToolWindow::GetMinimalWndDim`

`wxtoolwindowgetminimalwnddim`

`rowse00592`

^K `wxToolWindow GetMinimalWndDim`

^E `nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxtoolwindow')")`

^K `GetMinimalWndDim`

\$#+K! **wxToolWindow::GetPreferredSize**

wxSize **GetPreferredSize**(const **wxSize**& *given*)^K

Returns the preferred size for the window.

^wxToolWindow::GetPreferredSize

^wxtoolwindowgetpreferredsize

^browse00593

^K wxToolWindow GetPreferredSize

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxtoolwindow')")

^K GetPreferredSize

^{\$#+K!}**wxToolWindow::GetScrMousePos**

void GetScrMousePos(**wxMouseEvent&** *event*, **wxPoint&** *pos*)^K

Gets the mouse position in screen coordinates.

^wxToolWindow::GetScrMousePos

^wxtoolwindowgetscrmousepos

^browse00594

^K wxToolWindow GetScrMousePos

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp`, `wxtoolwindow`)"

^K GetScrMousePos

`$#+K! wxToolWindow::GetScrWindowRect`

`void GetScrWindowRect(wxRect& r)K`

Maps client coordinates to screen coordinates.

`wxToolWindow::GetScrWindowRect`

`wxtoolwindowgetscrwindowrect`

`browse00595`

`K wxToolWindow GetScrWindowRect`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`wxtoolwindow')")`

`K GetScrWindowRect`

\$#+K! **wxToolWindow::HandleTitleClick**

bool HandleTitleClick(wxMouseEvent& *event*)^K

Handles clicking on the title. By default, does nothing.

wxToolWindow::HandleTitleClick

wxtoolwindowhandletitleclick

browse00596

K wxToolWindow HandleTitleClick

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxtoolwindow')")

K HandleTitleClick

`$#+K! wxToolWindow::HitTestWindow`

`int HitTestWindow(wxMouseEvent& event)K`

Tests if the mouse position is in this window.

`wxToolWindow::HitTestWindow`

`wxtoolwindowhittestwindow`

`browse00597`

`K wxToolWindow HitTestWindow`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxtoolwindow')")`

`K HitTestWindow`

`wxToolWindow::LayoutMiniButtons`

`void LayoutMiniButtons()`^K

Lays out the buttons.

`wxToolWindow::LayoutMiniButtons`

`wxtoolwindowlayoutminibuttons`

`rowse00598`

`wxToolWindow LayoutMiniButtons`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp, `wxtoolwindow')")`

`LayoutMiniButtons`

`wxToolWindow::OnEraseBackground`

`void OnEraseBackground(wxEraseEvent& event)`^K

Responds to an erase background event.

`wxToolWindow::OnEraseBackground`

`wxtoolwindowonerasebackground`

`rowse00599`

`wxToolWindow OnEraseBackground`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `wxtoolwindow')")`

`OnEraseBackground`

wxToolWindow::OnLeftDown

void OnLeftDown(wxMouseEvent& *event*)^K

Responds to a mouse left down event.

^wxToolWindow::OnLeftDown

^wxtoolwindowonleftdown

^browse00600

^K wxToolWindow OnLeftDown

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxtoolwindow')")

^K OnLeftDown

`wxToolWindow::OnLeftUp`

`void OnLeftUp(wxMouseEvent& event)`^K

Responds to a mouse left up event.

`wxToolWindow::OnLeftUp`

`wxtoolwindowonleftup`

`rowse00601`

`wxToolWindow OnLeftUp`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `wxtoolwindow')")`

`OnLeftUp`

wxToolWindow::OnMiniButtonClicked

void OnMiniButtonClicked(int *btnIdx*)^K

Called when a mini button is clicked. By default, does nothing.

^wxToolWindow::OnMiniButtonClicked

^wxtoolwindowonminibuttonclicked

^browse00602

^K wxToolWindow OnMiniButtonClicked

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`wxtoolwindow')")

^K OnMiniButtonClicked

`wxToolWindow::OnMotion`

`void OnMotion(wxMouseEvent& event)`^K

Responds to a mouse move event.

`wxToolWindow::OnMotion`

`wxtoolwindowonmotion`

`rowse00603`

`wxToolWindow OnMotion`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `wxtoolwindow')")`

`OnMotion`

\$#+K! **wxToolWindow::OnPaint**

void OnPaint(wxPaintEvent& *event*)^K

Responds to a paint event.

wxToolWindow::OnPaint

wxtoolwindowonpaint

browse00604

K wxToolWindow OnPaint

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxtoolwindow')")

K OnPaint

`wxToolWindow::OnSize`

`void OnSize(wxSizeEvent& event)`^K

Responds to a size event.

`wxToolWindow::OnSize`

`wxtoolwindow::size`

`rowse00605`

`wxToolWindow OnSize`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(fl.hlp', `wxtoolwindow')")`

`OnSize`

\$#+K! **wxToolWindow::SetClient**

void SetClient(wxWindow* *pWnd*)^K

Sets the client for this tool window.

^wxToolWindow::SetClient

^wxtoolwindowsetclient

^browse00606

^K wxToolWindow SetClient

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp',`wxtoolwindow')")

^K SetClient

\$#+K! **wxToolWindow::SetHintCursor**

void SetHintCursor(int *type*)^K

Sets the hint cursor.

wxToolWindow::SetHintCursor

wxtoolwindowsethintcursor

browse00607

K wxToolWindow SetHintCursor

EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxtoolwindow')")

K SetHintCursor

`$#+K! wxToolWindow::SetTitleFont`

`void SetTitleFont(wxFont& font)K`

Sets the title font.

`wxToolWindow::SetTitleFont`

`wxtoolwindowsettitlefont`

`browse00608`

`K wxToolWindow SetTitleFont`

`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxtoolwindow')")`

`K SetTitleFont`

^{\$\$\$+K!} A row of all non-fixed bars don't position properly

By Julian Smart.

I found that if I added all non-fixed bars, bars would overlap. This seems to be because the proportional resizing doesn't work before the window is laid out. I worked around this by setting pane sizes *before* the bars are added:

```
wxSize sz = GetClientSize();

// Set width for panes to help it do the calculations
int i;
for (i = 0; i < 2; i++)
{
    cbDockPane& pane = * (m_frameLayout->GetPane(i));
    pane.SetPaneWidth(sz.x);
}
```

^A row of all non-fixed bars don't position properly

^topic3

^browse00614

^K A row of all non-fixed bars don't position properly

^EnableButton("Up");ChangeButtonBinding("Up", "JumpId('fl.hlp', `faq')")

