

#\$+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](#)

---

C  
ontents  
C  
ontents  
b  
rowse00001  
K  
Contents  
D  
isableButton("Up")

**Copyright notice**

FL is copyright Aleksandras Gluchovas, 2001-2002.

The licence is the wxWindows Licence.

---

Copyright notice  
topic0  
browse00002  
K Copyright notice  
DisableButton("Up")

## Introduction

What is FL?

Compiling and using FL

FL concepts

Controlling dragging behaviour

---

Introduction

Introduction

rowse0003

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](#)  
[cbPluginEvent](#)

---

<sup>A</sup> Alphabetical class reference  
<sup>C</sup> lassref  
<sup>b</sup> rowse00008  
<sup>K</sup> Alphabetical class reference  
<sup>D</sup> isableButton("Up")

[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.

<a href="#"><u>cbAntiflickerPlugin</u></a>	Double-buffering class
<a href="#"><u>cbBarDragPlugin</u></a>	Implements drag behaviour.
<a href="#"><u>cbBarHintsPlugin</u></a>	Implements bar decoration and sizing behaviour.
<a href="#"><u>cbHintAnimationPlugin</u></a>	Draws animated hints when the user drags a pane.
<a href="#"><u>cbPaneDrawPlugin</u></a>	Implements most of MFC-style control bar implementation.
<a href="#"><u>cbPluginBase</u></a>	Abstract base class for all control-bar related plugins.
<a href="#"><u>cbRowDragPlugin</u></a>	Implements row-dragging functionality.
<a href="#"><u>cbRowLayoutPlugin</u></a>	Implements row layout functionality.
<a href="#"><u>cbSimpleCustomizationPlugin</u></a>	Enables customization of a bar.
<a href="#"><u>cbBarSpy</u></a>	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`).

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

---

`C`lasses by category

`C`lassesbycat

`b`rowse00609

`K` Classes by category

`D`isableButton("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](#)

---

T  
o  
p  
i  
c  
o  
v  
e  
r  
v  
i  
e  
w  
s  
b  
r  
o  
w  
s  
e  
0  
0  
6  
1  
0  
K  
T  
o  
p  
i  
c  
o  
v  
e  
r  
v  
i  
e  
w  
s  
D  
i  
s  
a  
b  
l  
e  
B  
u  
t  
t  
o  
n  
(  
"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

browse00004

What is FL?

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

## 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

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, -1, "wxWindows 2.0 wxFrameLayout Test
Application", wxDefaultPosition,
wxSize( 700, 500 ),
wxCLIP_CHILDREN | wxMINIMIZE_BOX | wxMAXIMIZE_BOX |
wxTHICK_FRAME | wxSYSTEM_MENU | wxCAPTION,
"freimas" )
{
    mpClientWnd = CreateTextCtrl( "Client window" );
    mpLayout = new wxFrameLayout( this, mpClientWnd );
}
```

---

FL concepts

topic2

browse00006

FL concepts

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

```

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.

---

C ontrolling dragging behaviour

c ontrollingdragbehav

b rowse00007

K Controlling dragging behaviour

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

## `##+K!` **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](#)

---

`B` agLayout  
`b` aglayout  
`b` rowse00009  
`K` BagLayout  
`E` nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `classref)")

`##+K1` **wxBarLayout**

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**

[wxBarLayout::wxBarLayout](#)

[wxBarLayout::BarInfo](#)

[wxBarLayout::Next](#)

[wxBarLayout::Reset](#)

[wxBarLayout::RowInfo](#)

---

`w`xBarIterator

`w`xbariterator

`b`rowse00011

`K` wxBarIterator

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

## `$$$K!_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](#)

---

`C_bAntiflickerPlugin`

`C_bantiflickerplugin`

`b_rowse00017`

`K_cbAntiflickerPlugin`

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

## `$$$K!_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](#)

---

`C` `cbBarDimHandlerBase`

`C` `cbbardimhandlerbase`

`b` `rowse00026`

`K` `cbBarDimHandlerBase`

`E` `nableButton("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](#)

---

<sup>C</sup>bBarDragPlugin

<sup>C</sup>bbardragplugin

<sup>b</sup>rowse00032

<sup>K</sup> cbBarDragPlugin

<sup>E</sup>nableButton("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](#)

---

<sup>C</sup>bBarHintsPlugin

<sup>C</sup>bbarhintsplugin

<sup>b</sup>rowse00061

<sup>K</sup> cbBarHintsPlugin

<sup>E</sup>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](#)

---

`c_bBarInfo`

`c_bbarinfo`

`b_rowse00080`

`K_cbBarInfo`

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

## `$$$K!_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**

---

```
C_bBarShapeData  
C_bbarshapedata  
b_rowse00085  
K_cbBarShapeData  
E_nableButton("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](#)

---

<sup>c</sup>bBarSpy

<sup>c</sup>bbarspy

<sup>b</sup>rowse00086

<sup>K</sup> cbBarSpy

<sup>E</sup>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`

`bclosebox`

`rowse00090`

`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)")`

## `##+K!` **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(void);
};
```

---

`^`cbCommonPaneProperties

`^`bcommonpaneproperties

`^`rowse00094

`^`cbCommonPaneProperties

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

## `$$$K!` **cbCustomizeBarEvent**

Class for bar customization events.

### **Derived from**

[cbPluginEvent](#)

### **Include files**

<wx/fl/controlbar.h>

### **Data structures**

### **Members**

[cbCustomizeBarEvent::cbCustomizeBarEvent](#)

---

`C` `bCustomizeBarEvent`

`C` `bcustomizebarevent`

`b` `rowse00095`

`K` `cbCustomizeBarEvent`

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

`##+K!` **cbCustomizeLayoutEvent**

Class for layout customization events.

**Derived from**

[cbPluginEvent](#)

**Include files**

<wx/fl/controlbar.h>

**Data structures**

**Members**

[cbCustomizeLayoutEvent::cbCustomizeLayoutEvent](#)

---

`c`<sub>b</sub>CustomizeLayoutEvent  
`c`<sub>b</sub>customizelayoutevent  
`b`<sub>rowse00097</sub>  
`K` cbCustomizeLayoutEvent  
`E`<sub>n</sub>ableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `classref)")

## `$$$K!_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=](#)

---

`c_bDimInfo`

`c_bdiminfo`

`b_rowse00099`

`K_cbDimInfo`

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

## `##+K!` **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](#)

---

```
c_bDockBox  
c_bdockbox  
b_rowse00104  
K cbDockBox  
E_nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `classref)")
```

## `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::HasNotFixedBarsLeft](#)

---

`cbDockPane`

`cbdockpane`

`rowse00106`

`cbDockPane`

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

[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](#)

`##+K!` **cbDrawBarDecorEvent**

Class for bar decoration drawing events.

**Derived from**

[cbPluginEvent](#)

**Include files**

<wx/fl/controlbar.h>

**Data structures**

**Members**

[cbDrawBarDecorEvent::cbDrawBarDecorEvent](#)

---

`C` `cbDrawBarDecorEvent`

`C` `bdrawbardecocorevent`

`b` `rowse00174`

`K` `cbDrawBarDecorEvent`

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

`##+K! cbDrawBarHandlesEvent`

Class for bar handles drawing events.

**Derived from**

[cbPluginEvent](#)

**Include files**

<wx/fl/controlbar.h>

**Data structures**

**Members**

[cbDrawBarHandlesEvent::cbDrawBarHandlesEvent](#)

---

`C` bDrawBarHandlesEvent  
`C` bdrawbarhandlesevent  
`b` rowse00176  
`K` cbDrawBarHandlesEvent  
`E` nableButton("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](#)

---

`C` `cbDrawHintRectEvent`  
`C` `cbdrawhintrectevent`  
`b` `rowse00178`  
`K` `cbDrawHintRectEvent`  
`E` `nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `classref)")`

`##+K! cbDrawPaneBkGroundEvent`

Class for pane background drawing events.

**Derived from**

[cbPluginEvent](#)

**Include files**

<wx/fl/controlbar.h>

**Data structures**

**Members**

[cbDrawPaneBkGroundEvent::cbDrawPaneBkGroundEvent](#)

---

`C` bDrawPaneBkGroundEvent  
`C` bdrawpanebkgroundevent  
`b` rowse00180  
`K` cbDrawPaneBkGroundEvent  
`E` nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `classref)")

`##+K!` **cbDrawPaneDecorEvent**

Class for pane decoration drawing events.

**Derived from**

[cbPluginEvent](#)

**Include files**

<wx/fl/controlbar.h>

**Data structures**

**Members**

[cbDrawPaneDecorEvent::cbDrawPaneDecorEvent](#)

---

`C` bDrawPaneDecorEvent

`C` bdrawpanedecorevent

`b` rowse00182

`K` cbDrawPaneDecorEvent

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

## `$$$K!_cbDrawRowBkGroundEvent`

Class for row background drawing events.

### **Derived from**

[cbPluginEvent](#)

### **Include files**

<wx/fl/controlbar.h>

### **Data structures**

### **Members**

[cbDrawRowBkGroundEvent::cbDrawRowBkGroundEvent](#)

---

```
C_bDrawRowBkGroundEvent
C_bdrawrowbkgroundevent
b_rowse00184
K_cbDrawRowBkGroundEvent
E_nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `classref)")
```

`##+K! cbDrawRowDecorEvent`

Class for row decoration drawing events.

**Derived from**

[cbPluginEvent](#)

**Include files**

<wx/fl/controlbar.h>

**Data structures**

**Members**

[cbDrawRowDecorEvent::cbDrawRowDecorEvent](#)

---

`C` `cbDrawRowDecorEvent`

`C` `cbdrawrowdecorevent`

`b` `rowse00186`

`K` `cbDrawRowDecorEvent`

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

`##+K! cbDrawRowHandlesEvent`

Class for row handles drawing events.

**Derived from**

[cbPluginEvent](#)

**Include files**

<wx/fl/controlbar.h>

**Data structures**

**Members**

[cbDrawRowHandlesEvent::cbDrawRowHandlesEvent](#)

---

`C` `cbDrawRowHandlesEvent`

`C` `cbdrawrowhandlesevent`

`b` `rowse00188`

`K` `cbDrawRowHandlesEvent`

`E` `nableButton("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](#)

---

`c_bDynToolBarDimHandler`

`c_bdyntoolbardimhandler`

`b_rowse00190`

`K cbDynToolBarDimHandler`

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

`##+K! cbFinishDrawInAreaEvent`

Class for finish drawing in area events.

**Derived from**

[cbPluginEvent](#)

**Include files**

<wx/fl/controlbar.h>

**Data structures**

**Members**

[cbFinishDrawInAreaEvent::cbFinishDrawInAreaEvent](#)

---

`C` `cbFinishDrawInAreaEvent`

`C` `bfinishdrawinareaevent`

`b` `rowse00193`

`K` `cbFinishDrawInAreaEvent`

`E` `nableButton("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](#)

---

`C` bFloatedBarWindow

`C` bfloatedbarwindow

`b` rowse00195

`K` cbFloatedBarWindow

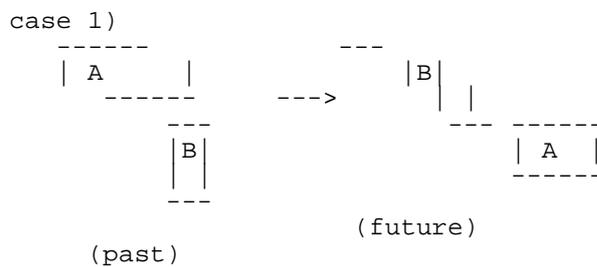
`E` nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `classref)")

## cbGCUpdatesMgr

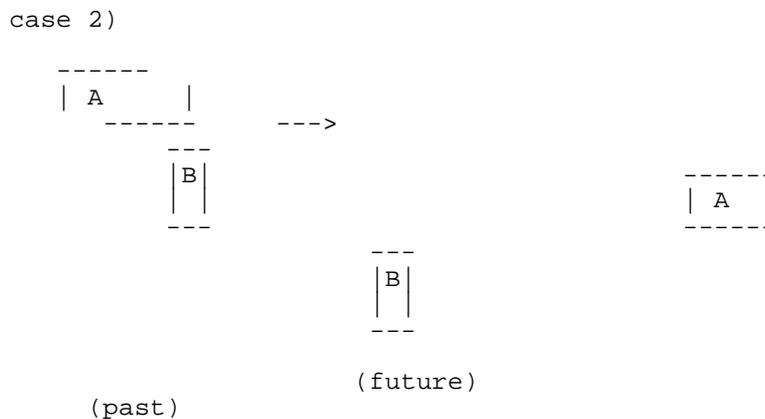
This class implements optimized logic for refreshing the areas of frame layout that actually need to be updated. It is used as the default updates manager by wxFrameLayout.

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.



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.

---

`c` bGCUpdatesMgr  
`c` bgcupdatesmgr  
`b` rowse00205  
`K` cbGCUpdatesMgr  
`E` nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `classref)")

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`

`rowse00211`

`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](#)

---

`C` `cbInsertBarEvent`

`C` `binsertbarevent`

`b` `rowse00221`

`K` `cbInsertBarEvent`

`E` `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](#)

---

`cbLayoutRowEvent`

`cblayoutrowevent`

`browse00223`

`K cbLayoutRowEvent`

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

`##+K! cbLayoutRowsEvent`

Class for multiple rows layout events.

**Derived from**

[cbPluginEvent](#)

**Include files**

<wx/fl/controlbar.h>

**Data structures**

**Members**

[cbLayoutRowsEvent::cbLayoutRowsEvent](#)

---

`cbLayoutRowsEvent`

`cblayoutrowsevent`

`browse00225`

`KcbLayoutRowsEvent`

`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](#)

---

`c_bLeftDClickEvent`

`c_bleftdclিকেvent`

`b_rowse00227`

`K_cbLeftDClickEvent`

`E_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](#)

---

`c_bLeftDownEvent`

`c_bleftdownevent`

`b_rowse00229`

`K_cbLeftDownEvent`

`E_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](#)

---

`c_bLeftUpEvent`

`c_bleftupevent`

`b_rowse00231`

`K_cbLeftUpEvent`

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

## `$$$K!_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`](#)

---

`C`<sub>bMiniButton</sub>

`C`<sub>bminibutton</sub>

`b`<sub>rowse00233</sub>

`K`<sub>cbMiniButton</sub>

`E`<sub>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `classref)")</sub>

`##+K! cbMotionEvent`

Class for mouse motion events.

**Derived from**

[cbPluginEvent](#)

**Include files**

<wx/fl/controlbar.h>

**Data structures**

**Members**

[cbMotionEvent::cbMotionEvent](#)

---

`cbMotionEvent`

`cbmotionevent`

`browse00246`

`KcbMotionEvent`

`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](#)  
[cbPaneDrawPlugin::OnLDbClick](#)

---

`C_bPaneDrawPlugin`

`C_bpanedrawplugin`

`b_rowse00248`

`K_cbPaneDrawPlugin`

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

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](#)

---

`C` bPluginBase  
`C` bpluginbase  
`b` rowse00279  
`K` cbPluginBase  
`E` nableButton("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](#)

---

`C_bPluginEvent`

`C_bpluginevent`

`b_rowse00286`

`K_cbPluginEvent`

`E_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](#)

---

`C` `cbRemoveBarEvent`

`C` `bremovebarevent`

`b` `rowse00289`

`K` `cbRemoveBarEvent`

`E` `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](#)

---

`C` `cbResizeBarEvent`

`C` `bresizebarevent`

`b` `rowse00291`

`K` `cbResizeBarEvent`

`E` `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](#)

---

`C_bResizeRowEvent`

`C_bresizerowevent`

`b_rowse00293`

`K_cbResizeRowEvent`

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

`##+K! cbRightDownEvent`

Class for mouse right down events.

**Derived from**

[cbPluginEvent](#)

**Include files**

<wx/fl/controlbar.h>

**Data structures**

**Members**

[cbRightDownEvent::cbRightDownEvent](#)

---

`C` `bRightDownEvent`

`C` `brightdownevent`

`b` `rowse00295`

`K` `cbRightDownEvent`

`E` `nableButton("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](#)

---

`CbRightUpEvent`

`Cbrightupevent`

`brorowse00297`

`KcbRightUpEvent`

`EnableButton("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::DrawOrtoRomb](#)  
[cbRowDragPlugin::DrawRectShade](#)  
[cbRowDragPlugin::DrawRomb](#)  
[cbRowDragPlugin::DrawRombShades](#)  
[cbRowDragPlugin::DrawRowDragHint](#)  
[cbRowDragPlugin::DrawRowsDragHintsBorder](#)  
[cbRowDragPlugin::DrawTrianDown](#)  
[cbRowDragPlugin::DrawTrianRight](#)  
[cbRowDragPlugin::DrawTrianUp](#)  
[cbRowDragPlugin::ExpandRow](#)  
[cbRowDragPlugin::FinishOnScreenDraw](#)  
[cbRowDragPlugin::GetCollapsedIconsPos](#)  
[cbRowDragPlugin::GetCollapsedIconRect](#)  
[cbRowDragPlugin::GetCollapsedRowIconHeight](#)  
[cbRowDragPlugin::GetFirstRow](#)  
[cbRowDragPlugin::GetHRowsCountForPane](#)

---

`cbRowDragPlugin`

`cbrowdragplugin`

`rowse00299`

`cbRowDragPlugin`

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

cbRowDragPlugin::GetRowDragHintWidth  
cbRowDragPlugin::GetRowHintRect  
cbRowDragPlugin::HitTestCollapsedRowIcon  
cbRowDragPlugin::HitTestRowDragHint  
cbRowDragPlugin::InsertDraggedRowBefore  
cbRowDragPlugin::ItemIsInFocus  
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](#)

---

`c_bRowInfo`  
`c_browinfo`  
`b_rowse00344`  
`K_cbRowInfo`  
`E_nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `classref)")`

## `$$$K!_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::ShiftLeftTrashold](#)  
[cbRowLayoutPlugin::ShiftRightTrashold](#)  
[cbRowLayoutPlugin::SlideLeftSideBars](#)  
[cbRowLayoutPlugin::SlideRightSideBars](#)  
[cbRowLayoutPlugin::StickRightSideBars](#)

---

`C_bRowLayoutPlugin`

`C_browlayoutplugin`

`b_rowse00348`

`K_cbRowLayoutPlugin`

`E_nableButton("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](#)

---

`C` `cbSimpleCustomizationPlugin`

`C` `cbsimplecustomizationplugin`

`b` `rowse00374`

`K` `cbSimpleCustomizationPlugin`

`E` `nableButton("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](#)

---

`C` `cbSimpleUpdatesMgr`

`C` `bsimpleupdatesmgr`

`b` `rowse00379`

`K` `cbSimpleUpdatesMgr`

`E` `nableButton("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](#)

---

`C` `cbSizeBarWndEvent`

`C` `bsizebarwndevent`

`b` `rowse00389`

`K` `cbSizeBarWndEvent`

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

`##+K! cbStartBarDraggingEvent`

Class for start-bar-dragging events.

**Derived from**

[cbPluginEvent](#)

**Include files**

<wx/fl/controlbar.h>

**Data structures**

**Members**

[cbStartBarDraggingEvent::cbStartBarDraggingEvent](#)

---

`C` `cbStartBarDraggingEvent`

`C` `bstartbardraggingevent`

`b` `rowse00391`

`K` `cbStartBarDraggingEvent`

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

`##+K! cbStartDrawInAreaEvent`

Class for start drawing in area events.

**Derived from**

[cbPluginEvent](#)

**Include files**

<wx/fl/controlbar.h>

**Data structures**

**Members**

[cbStartDrawInAreaEvent::cbStartDrawInAreaEvent](#)

---

`C` `cbStartDrawInAreaEvent`

`C` `bstartdrawinareaevent`

`b` `rowse00393`

`K` `cbStartDrawInAreaEvent`

`E` `nableButton("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](#)

---

`C_bUpdateMgrData`

`C_bupdatemgrdata`

`b_rowse00395`

`K_cbUpdateMgrData`

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

## `$$$K!_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](#)

---

`C` bUpdatesManagerBase

`C` bupdatesmanagerbase

`b` rowse00401

`K` cbUpdatesManagerBase

`E` nableButton("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`

`K wxDynamicToolBar`

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

`$$$K1` **wxDynToolInfo**

This class holds dynamic toolbar item information.

**Derived from**

[wxToolLayoutItem](#)

**Include files**

<wx/fl/dyntbar.h>

**Data structures**

---

`wxDynToolInfo`  
`wxdyntoolinfo`  
`rowse00438`  
`K wxDynToolInfo`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `classref)")`

## \$#+K1 wxFrameLayout

wxFrameLayout 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

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

---

<sup>w</sup>xFrameLayout

<sup>w</sup>xframelayou

<sup>b</sup>rowse00439

<sup>K</sup> wxFrameLayout

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `classref)")

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

\$#+K1 **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](#)

---

<sup>w</sup>xFrameManager

<sup>w</sup>xframemanager

<sup>b</sup>rowse00519

<sup>K</sup> wxFrameManager

<sup>E</sup>nableButton("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](#)

---

<sup>G</sup>arbageCollector

<sup>g</sup>arbagecollector

<sup>b</sup>rowse00544

<sup>K</sup> GarbageCollector

<sup>E</sup>nableButton("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  
l  
b  
K  
E  
LayoutManagerBase  
layoutmanagerbase  
rowse00558  
LayoutManagerBase  
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')")`

`##+K1` **wxToolLayoutItem**

Tool layout item.

**Derived from**

wxObject

**Include files**

<wx/fl/dyntbar.h>

**Data structures**

```
typedef wxToolLayoutItem* wxToolLayoutItemPtrT
```

```
typedef wxDynToolInfo* wxDynToolInfoPtrT
```

---

`w`xToolLayoutItem  
`w`xtoollayoutitem  
`b`rowse00583  
`K` wxToolLayoutItem  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `classref)")

## \$#+K1. **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](#)

---

<sup>w</sup>xToolWindow

<sup>w</sup>xtoolwindow

<sup>b</sup>rowse00584

<sup>K</sup> wxToolWindow

<sup>E</sup>nableButton("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

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

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

\$#+K! **FAQ**

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

---

F AQ  
f aq  
b rowse00613  
K FAQ  
E 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.

---

`B agLayout::Layout  
b aglayoutlayout  
b rowse00010  
K BagLayout Layout  
E nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `baglayout')")  
K Layout`

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

**wxBarIterator(RowArrayT& rows)<sup>K</sup>**

Constructor, taking row array.

---

`w` wxBarIterator::wxBarIterator

`w` wxbariteratorwxbariterator

`b` rowse00012

`K` wxBarIterator wxBarIterator

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

`K` wxBarIterator

`$#+K! wxBarIterator::BarInfo`

`cbBarInfo& BarInfo()K`

Gets the current bar information.

---

`w wxBarIterator::BarInfo  
w wxbariteratorbarinfo  
b rowse00013  
K wxBarIterator BarInfo  
E nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxbariterator)")  
K BarInfo`

`$#+K1 wxBarIterator::Next`

`bool Next()`<sup>K</sup>

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

---

`w_xBarIterator::Next`

`w_xbariteratornext`

`b_rowse00014`

`K wxBarIterator Next`

`E_nableButton("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`

`K wxBarIterator RowInfo`

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

`K RowInfo`

`##+K! 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
```

**`$#+K! cbAntiflickerPlugin::~cbAntiflickerPlugin`**

**`~cbAntiflickerPlugin()`<sup>K</sup>**

Destructor.

---

<sup>C</sup>`bAntiflickerPlugin::~cbAntiflickerPlugin`

<sup>C</sup>`bantiflickerplugindtor`

<sup>b</sup>`rowse00019`

<sup>K</sup>`cbAntiflickerPlugin ~cbAntiflickerPlugin`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbantiflickerplugin)")`

<sup>K</sup>`~cbAntiflickerPlugin`

**`$#+K! cbAntiflickerPlugin::AllocNewBuffer`**

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

Allocates a suitable buffer.

---

`CbAntiflickerPlugin::AllocNewBuffer`  
`Cbantiflickerpluginallocnewbuffer`  
`browse00020`  
`K cbAntiflickerPlugin AllocNewBuffer`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbantiflickerplugin')")`  
`K AllocNewBuffer`

**`$#+K!` `cbAntiflickerPlugin::FindSuitableBuffer`**

**`wxDC*` `FindSuitableBuffer(const wxRect& forArea)`<sup>K</sup>**

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

---

<sup>C</sup>`bAntiflickerPlugin::FindSuitableBuffer`

<sup>C</sup>`bantiflickerpluginfindsuitablebuffer`

<sup>b</sup>`rowse00021`

<sup>K</sup>`cbAntiflickerPlugin FindSuitableBuffer`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbantiflickerplugin)")`

<sup>K</sup>`FindSuitableBuffer`

**`$#+K! cbAntiflickerPlugin::GetClientDC`**

**`wxDC& GetClientDC()`**<sup>K</sup>

Gets the client device context.

---

<sup>C</sup>`bAntiflickerPlugin::GetClientDC`  
<sup>C</sup>`bantiflickerpluggingetclientdc`  
<sup>b</sup>`rowse00022`  
<sup>K</sup>`cbAntiflickerPlugin GetClientDC`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbantiflickerplugin')")`  
<sup>K</sup>`GetClientDC`

**\$#+K! cbAntiflickerPlugin::GetWindowDC**

**wxDC& GetWindowDC()**<sup>K</sup>

Gets the window device context.

---

<sup>C</sup>bAntiflickerPlugin::GetWindowDC  
<sup>C</sup>bantiflickerpluggingetwindowdc  
<sup>b</sup>rowse00023  
<sup>K</sup> cbAntiflickerPlugin GetWindowDC  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbantiflickerplugin)")  
<sup>K</sup> GetWindowDC

**`$#+K! cbAntiflickerPlugin::OnFinishDrawInArea`**

**`void OnFinishDrawInArea(cbFinishDrawInAreaEvent& event)K`**

Handler for plugin event.

---

<sup>C</sup>`bAntiflickerPlugin::OnFinishDrawInArea`

<sup>C</sup>`bantiflickerpluginonfinishdrawinarea`

<sup>b</sup>`rowse00024`

<sup>K</sup>`cbAntiflickerPlugin OnFinishDrawInArea`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbantiflickerplugin)")`

<sup>K</sup>`OnFinishDrawInArea`

**`cbAntiflickerPlugin::OnStartDrawInArea`**

**`void OnStartDrawInArea(cbStartDrawInAreaEvent& event)`**<sup>K</sup>

Handler for plugin event.

---

<sup>C</sup>`bAntiflickerPlugin::OnStartDrawInArea`

<sup>C</sup>`bantiflickerpluginonstartdrawinarea`

<sup>b</sup>`rowse00025`

<sup>K</sup>`cbAntiflickerPlugin OnStartDrawInArea`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbantiflickerplugin)")`

<sup>K</sup>`OnStartDrawInArea`

**`$#+K! cbBarDimHandlerBase::cbBarDimHandlerBase`**

**`cbBarDimHandlerBase()`<sup>K</sup>**

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()`.

---

<sup>C</sup>`cbBarDimHandlerBase::cbBarDimHandlerBase`

<sup>C</sup>`cbbardimhandlerbasecbbardimhandlerbase`

<sup>b</sup>`rowse00027`

<sup>K</sup>`cbBarDimHandlerBase cbBarDimHandlerBase`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardimhandlerbase')")`

<sup>K</sup>`cbBarDimHandlerBase`

**`$#+K! cbBarDimHandlerBase::AddRef`**

**`void AddRef()`**<sup>K</sup>

Increments the reference count.

---

<sup>C</sup>`bBarDimHandlerBase::AddRef`

<sup>C</sup>`bbardimhandlerbaseaddref`

<sup>b</sup>`rowse00028`

<sup>K</sup>`cbBarDimHandlerBase AddRef`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardimhandlerbase')")`

<sup>K</sup>`AddRef`

**`cbBarDimHandlerBase::OnChangeBarState`**

**`void OnChangeBarState(cbBarInfo* pBar, int newState)`**<sup>K</sup>

Responds to "bar-state-changes" notifications.

---

<sup>C</sup>`bBarDimHandlerBase::OnChangeBarState`  
<sup>C</sup>`bbardimhandlerbaseonchangebarstate`  
<sup>b</sup>`rowse00029`  
<sup>K</sup>`cbBarDimHandlerBase OnChangeBarState`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardimhandlerbase')")`  
<sup>K</sup>`OnChangeBarState`

<sup>S#+K!</sup> **cbBarDimHandlerBase::OnResizeBar**

**void OnResizeBar(cbBarInfo\* pBar, const wxSize& given, wxSize& preferred)<sup>K</sup>**

Responds to bar resize notifications.

---

<sup>C</sup>bBarDimHandlerBase::OnResizeBar  
<sup>C</sup>bbardimhandlerbaseonresizebar  
<sup>b</sup>rowse00030  
<sup>K</sup> cbBarDimHandlerBase OnResizeBar  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardimhandlerbase')")  
<sup>K</sup> OnResizeBar

**`$#+K! cbBarDimHandlerBase::RemoveRef`**

**`void RemoveRef()`**<sup>K</sup>

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

---

<sup>C</sup>`bBarDimHandlerBase::RemoveRef`

<sup>C</sup>`bbardimhandlerbaseremoveref`

<sup>b</sup>`rowse00031`

<sup>K</sup>`cbBarDimHandlerBase RemoveRef`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardimhandlerbase')")`

<sup>K</sup>`RemoveRef`

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

**cbBarDragPlugin()**<sup>K</sup>

Default constructor.

**cbBarDragPlugin(wxFrameLayout\* pPanel, int paneMask = wxALL\_PANES)**<sup>K</sup>

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

---

<sup>C</sup>bBarDragPlugin::cbBarDragPlugin

<sup>C</sup>bbardragplugin::cbBarDragPlugin

<sup>b</sup>rowse00033

<sup>K</sup> cbBarDragPlugin cbBarDragPlugin

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbbardragplugin')")

<sup>K</sup> cbBarDragPlugin

<sup>K</sup> cbBarDragPlugin

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

**`~cbBarDragPlugin()`**<sup>K</sup>

Destructor.

---

<sup>C</sup>`bBarDragPlugin::~cbBarDragPlugin`

<sup>C</sup>`bbardragplugindtor`

<sup>b</sup>`rowse00034`

<sup>K</sup>`cbBarDragPlugin ~cbBarDragPlugin`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardragplugin')")`

<sup>K</sup>`~cbBarDragPlugin`

**`cbBarDragPlugin::AdjustHintRect`**

**`void AdjustHintRect(wxPoint& mousePos)`**<sup>K</sup>

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

---

`cbBarDragPlugin::AdjustHintRect`  
`cbbardragpluginadjusthintrect`  
`rowse00035`  
`cbBarDragPlugin AdjustHintRect`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardragplugin')")`  
`AdjustHintRect`

**`cbBarDragPlugin::CalcOnScreenDims`**

**`void CalcOnScreenDims(wxRect& rect)`**<sup>K</sup>

Internal implementation function.

---

<sup>C</sup>`cbBarDragPlugin::CalcOnScreenDims`

<sup>C</sup>`cbbardragplugincalconscreendims`

<sup>b</sup>`rowse00036`

<sup>K</sup>`cbBarDragPlugin CalcOnScreenDims`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbbardragplugin')")`

<sup>K</sup>`CalcOnScreenDims`

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

**void ClipPosInFrame(wxPoint& pos)<sup>K</sup>**

Internal implementation function.

---

`CbBarDragPlugin::ClipPosInFrame`  
`Cbbardragpluginclipposinframe`  
`browse00037`  
`K cbBarDragPlugin ClipPosInFrame`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardragplugin')")`  
`K ClipPosInFrame`

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

**`void ClipRectInFrame(wxRect& rect)K`**

Internal implementation function.

---

`CbBarDragPlugin::ClipRectInFrame`  
`Ccbbardragplugincliprectinframe`  
`browse00038`  
`K cbBarDragPlugin ClipRectInFrame`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardragplugin')")`  
`K ClipRectInFrame`

**`cbBarDragPlugin::DoDrawHintRect`**

**`void DoDrawHintRect(wxRect& rect, bool isInClientRect)`**<sup>K</sup>

Internal implementation function. Draw the hint rectangle.

---

<sup>C</sup>`cbBarDragPlugin::DoDrawHintRect`  
<sup>C</sup>`cbbardragplugindodrawhintrect`  
<sup>b</sup>`rowse00039`  
<sup>K</sup>`cbBarDragPlugin DoDrawHintRect`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardragplugin')")`  
<sup>K</sup>`DoDrawHintRect`

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

**void DrawHintRect(wxRect& *rect*, bool *isInClientRect*)<sup>K</sup>**

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)`**<sup>K</sup>

Internal implementation function. Erase the visual hint while dragging.

---

<sup>C</sup>`cbBarDragPlugin::EraseHintRect`

<sup>C</sup>`cbbardragpluginerasehintrect`

<sup>b</sup>`rowse00041`

<sup>K</sup>`cbBarDragPlugin EraseHintRect`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardragplugin')")`

<sup>K</sup>`EraseHintRect`

`##+K! cbBarDragPlugin::FinishTracking`

`void FinishTracking()K`

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

---

`CbBarDragPlugin::FinishTracking  
Ccbbardragpluginfinishtracking  
browse00042  
K cbBarDragPlugin FinishTracking  
EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardragplugin')")  
K FinishTracking`

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

**`int GetBarHeightInPane(cbDockPane* pPane)`<sup>K</sup>**

Internal implementation function.

---

<sup>C</sup>`cbBarDragPlugin::GetBarHeightInPane`

<sup>C</sup>`cbbardragpluggingetbarheightinpane`

<sup>b</sup>`rowse00043`

<sup>K</sup>`cbBarDragPlugin GetBarHeightInPane`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardragplugin')")`

<sup>K</sup>`GetBarHeightInPane`

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

**`int GetBarWidthInPane(cbDockPane* pPane)K`**

Internal implementation function.

---

`CbBarDragPlugin::GetBarWidthInPane`  
`Ccbbardragplugingetbarwidthinpane`  
`browse00044`  
`K cbBarDragPlugin GetBarWidthInPane`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardragplugin')")`  
`K GetBarWidthInPane`

**`$$$K!cbBarDragPlugin::GetDistanceToPane`**

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

Internal implementation function.

---

`^cbBarDragPlugin::GetDistanceToPane`

`^bbardragpluggingetdistancetopane`

`^rowse00045`

`^K cbBarDragPlugin GetDistanceToPane`

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

`^K GetDistanceToPane`

**`cbBarDragPlugin::HitTestPanes`**

**`cbDockPane* HitTestPanes(wxRect& rect)`**<sup>K</sup>

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

**`cbDockPane* HitTestPanes(wxPoint& pos)`**<sup>K</sup>

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

---

<sup>C</sup>`cbBarDragPlugin::HitTestPanes`

<sup>C</sup>`cbbardragpluginhittestpanes`

<sup>b</sup>`rowse00046`

<sup>K</sup>`cbBarDragPlugin HitTestPanes`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardragplugin')")`

<sup>K</sup>`HitTestPanes`

<sup>K</sup>`HitTestPanes`

`##+K! cbBarDragPlugin::HitsPane`

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

Internal implementation function.

---

`cbBarDragPlugin::HitsPane`

`cbbardragpluginhitspane`

`rowse00047`

`K cbBarDragPlugin HitsPane`

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

`K HitsPane`

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

**`bool IsInClientArea(wxPoint& mousePos)`<sup>K</sup>**

Internal implementation function.

**`bool IsInClientArea(wxRect& rect)`<sup>K</sup>**

Internal implementation function.

---

`^`cbBarDragPlugin::IsInClientArea  
`^`cbbardragpluginisinclientarea  
`^`rowse00048  
`^`cbBarDragPlugin IsInClientArea  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbbardragplugin')")  
`^`IsInClientArea  
`^`IsInClientArea

**`cbBarDragPlugin::IsInOtherPane`**

**`bool IsInOtherPane(wxPoint& mousePos)`**<sup>K</sup>

Internal implementation function.

---

<sup>C</sup>`cbBarDragPlugin::IsInOtherPane`  
<sup>C</sup>`cbbardragpluginisinootherpane`  
<sup>b</sup>`rowse00049`  
<sup>K</sup>`cbBarDragPlugin IsInOtherPane`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbbardragplugin')")`  
<sup>K</sup>`IsInOtherPane`

**`cbBarDragPlugin::OnDrawHintRect`**

**`void OnDrawHintRect(cbDrawHintRectEvent& event)`**<sup>K</sup>

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)`**<sup>K</sup>

Handler for plugin event.

---

`cbBarDragPlugin::OnLButtonDown`  
`cbbardragpluginonlbuttondown`  
`rowse00051`  
`cbBarDragPlugin OnLButtonDown`  
`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardragplugin')")`  
`OnLButtonDown`

`$#+K! cbBarDragPlugin::OnLButtonUp`

`void OnLButtonUp(cbLeftUpEvent& event)K`

Handler for plugin event.

---

`CbBarDragPlugin::OnLButtonUp  
Cbbardragpluginonlbuttonup  
browse00052  
K cbBarDragPlugin OnLButtonUp  
EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardragplugin')")  
K OnLButtonUp`

`##+K! cbBarDragPlugin::OnLDbIcIck`

`void OnLDbIcIck(cbLeftDcIckEvent& event)K`

Handler for plugin event.

---

`^cbBarDragPlugin::OnLDbIcIck  
^cbbardragpluginonldbIck  
^rowse00053  
K cbBarDragPlugin OnLDbIcIck  
E nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardragplugin')")  
K OnLDbIcIck`

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

**`void OnMouseMove(cbMotionEvent& event)K`**

Handler for plugin event.

---

`CbBarDragPlugin::OnMouseMove`  
`Cbbardragpluginonmousemove`  
`browse00054`  
`K cbBarDragPlugin OnMouseMove`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardragplugin')")`  
`K OnMouseMove`

**`cbBarDragPlugin::OnStartBarDragging`**

**`void OnStartBarDragging(cbStartBarDraggingEvent& event)`**<sup>K</sup>

Handler for plugin event.

---

<sup>C</sup>`bBarDragPlugin::OnStartBarDragging`

<sup>C</sup>`bbardragpluginonstartbardragging`

<sup>b</sup>`rowse00055`

<sup>K</sup>`cbBarDragPlugin OnStartBarDragging`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardragplugin')")`

<sup>K</sup>`OnStartBarDragging`

**`cbBarDragPlugin::RectToScr`**

**`void RectToScr(wxRect& frameRect, wxRect& scrRect)`**<sup>K</sup>

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

---

<sup>C</sup>`cbBarDragPlugin::RectToScr`  
<sup>C</sup>`cbbardragpluginrecttoscr`  
<sup>b</sup>`rowse00056`  
<sup>K</sup>`cbBarDragPlugin RectToScr`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardragplugin')")`  
<sup>K</sup>`RectToScr`

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

**`void ShowHint(bool prevWasInClient)`**<sup>K</sup>

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

---

`^`cbBarDragPlugin::ShowHint  
`^`cbbardragpluginshowhint  
`b`rowse00057  
`K` cbBarDragPlugin ShowHint  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardragplugin')")  
`K` ShowHint

**`cbBarDragPlugin::StartTracking`**

**`void StartTracking()`**

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

---

`cbBarDragPlugin::StartTracking`  
`cbbardragpluginstarttracking`  
`rowse00058`  
`cbBarDragPlugin StartTracking`  
`enableButton("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  
K cbBarDragPlugin StickToPane  
EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardragplugin')")  
K StickToPane`

**`cbBarDragPlugin::UnstickFromPane`**

**`void UnstickFromPane(cbDockPane* pPane, wxPoint& mousePos)`**<sup>K</sup>

Internal implementation function.

---

<sup>C</sup>`cbBarDragPlugin::UnstickFromPane`  
<sup>C</sup>`cbbardragpluginunstickfrompane`  
<sup>b</sup>`rowse00060`  
<sup>K</sup>`cbBarDragPlugin UnstickFromPane`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbardragplugin')")`  
<sup>K</sup>`UnstickFromPane`

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

**cbBarHintsPlugin()**<sup>K</sup>

Default constructor.

**cbBarHintsPlugin(wxFrameLayout\* pLayout, int paneMask = wxALL\_PANES)**<sup>K</sup>

Constructor, taking parent frame and pane mask flag.

---

`^`cbBarHintsPlugin::cbBarHintsPlugin  
`^`bbarhintspluginbbarhintsplugin  
`b`rowse00062  
`K` cbBarHintsPlugin cbBarHintsPlugin  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbbarhintsplugin')")  
`K` cbBarHintsPlugin  
`K` cbBarHintsPlugin

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

`~cbBarHintsPlugin()K`

Destructor.

---

`CbBarHintsPlugin::~~cbBarHintsPlugin`

`Cbbarhintsplugindtor`

`browse00063`

`K cbBarHintsPlugin ~cbBarHintsPlugin`

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

`K ~cbBarHintsPlugin`

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

**`void CreateBoxes()`<sup>K</sup>**

Helper function: creates close and collapse boxes.

---

`c`<sub>1</sub> `cbBarHintsPlugin::CreateBoxes`  
`c`<sub>1</sub> `cbbarhintsplugincreateboxes`  
`b`<sub>1</sub> `rowse00064`  
`K` `cbBarHintsPlugin CreateBoxes`  
`E`<sub>1</sub> `nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbarhintsplugin')")`  
`K` `CreateBoxes`

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

**void DoDrawHint(wxDC& dc, wxRect& rect, int pos, int boxOfs, int grooveOfs, bool isFixed)<sup>K</sup>**

Helper function: draws a hint.

---

`CbBarHintsPlugin::DoDrawHint`  
`Cbbarhintsplugindodrawhint`  
`browse00065`  
`K cbBarHintsPlugin DoDrawHint`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbarhintsplugin')")`  
`K DoDrawHint`

`$#+K!` **cbBarHintsPlugin::Draw3DBox**

**void Draw3DBox(wxDC& *dc*, const wxPoint& *pos*, bool *pressed*)**<sup>K</sup>

Helper function: draws a 3D box.

---

<sup>C</sup>bBarHintsPlugin::Draw3DBox

<sup>C</sup>bbarhintsplugindraw3dbox

<sup>b</sup>rowse00066

<sup>K</sup> cbBarHintsPlugin Draw3DBox

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbarhintsplugin')")

<sup>K</sup> Draw3DBox

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

**void DrawCloseBox(wxDC& *dc*, const wxPoint& *pos*, bool *pressed*)**<sup>K</sup>

Helper function: draws a close box.

---

<sup>C</sup>bBarHintsPlugin::DrawCloseBox  
<sup>C</sup>bbarhintsplugindrawclosebox  
<sup>b</sup>rowse00067  
<sup>K</sup> cbBarHintsPlugin DrawCloseBox  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbarhintsplugin')")  
<sup>K</sup> DrawCloseBox

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

**`void DrawCollapseBox(wxDC& dc, const wxPoint& pos, bool atLeft, bool disabled, bool pressed)`**<sup>K</sup>

Helper function: draws a collapse box.

---

`^``bBarHintsPlugin::DrawCollapseBox`  
`^``bbarhintsplugindrawcollapsebox`  
`b``rowse00068`  
`K``cbBarHintsPlugin DrawCollapseBox`  
`E``nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbarhintsplugin')")`  
`K``DrawCollapseBox`

**`cbBarHintsPlugin::DrawGrooves`**

**`void DrawGrooves(wxDC& dc, const wxPoint& pos, int length)`**<sup>K</sup>

Helper function: draws grooves.

---

<sup>C</sup>`cbBarHintsPlugin::DrawGrooves`

<sup>C</sup>`bbarhintsplugindrawgrooves`

<sup>b</sup>`rowse00069`

<sup>K</sup>`cbBarHintsPlugin DrawGrooves`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbarhintsplugin')")`

<sup>K</sup>`DrawGrooves`

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

**void ExcludeHints(wxRect& *rect*, cbBarInfo& *info*)<sup>K</sup>**

Helper function.

---

<sup>C</sup>bBarHintsPlugin::ExcludeHints

<sup>C</sup>bbarhintspluginexcludehints

<sup>b</sup>rowse00070

<sup>K</sup> cbBarHintsPlugin ExcludeHints

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbarhintsplugin')")

<sup>K</sup> ExcludeHints

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

**void GetHintsLayout(wxRect& *rect*, cbBarInfo& *info*, int& *boxOfs*, int& *grooveOfs*, int& *pos*)<sup>K</sup>**

Helper function: gets the layout of a hint.

---

`CbBarHintsPlugin::GetHintsLayout`  
`Cbbarhintspluggingethintslayout`  
`browse00071`  
`K cbBarHintsPlugin GetHintsLayout`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbarhintsplugin')")`  
`K GetHintsLayout`

**`cbBarHintsPlugin::HitTestHints`**

**`int HitTestHints(cbBarInfo& info, const wxPoint& pos)`**<sup>K</sup>

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

---

<sup>C</sup>`cbBarHintsPlugin::HitTestHints`  
<sup>C</sup>`cbbarhintspluginhittesthints`  
<sup>b</sup>`rowse00072`  
<sup>K</sup>`cbBarHintsPlugin HitTestHints`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbbarhintsplugin')")`  
<sup>K</sup>`HitTestHints`

<sup>S#+K!</sup>**cbBarHintsPlugin::OnDrawBarDecorations**

**void OnDrawBarDecorations(cbDrawBarDecorEvent& event)<sup>K</sup>**

Handles a plugin event.

---

<sup>C</sup>bBarHintsPlugin::OnDrawBarDecorations

<sup>C</sup>bbarhintspluginondrawbardecorations

<sup>b</sup>rowse00073

<sup>K</sup> cbBarHintsPlugin OnDrawBarDecorations

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbarhintsplugin')")

<sup>K</sup> OnDrawBarDecorations

**`cbBarHintsPlugin::OnInitPlugin`**

**`void OnInitPlugin()`**

Called to initialize this plugin.

---

`cbBarHintsPlugin::OnInitPlugin`  
`cbbarhintspluginoninitplugin`  
`rowse00074`  
`cbBarHintsPlugin OnInitPlugin`  
`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbarhintsplugin')")`  
`OnInitPlugin`

**`cbBarHintsPlugin::OnLeftDown`**

**`void OnLeftDown(cbLeftDownEvent& event)`**<sup>K</sup>

Handles a plugin event.

---

<sup>C</sup>`cbBarHintsPlugin::OnLeftDown`

<sup>C</sup>`bbarhintspluginonleftdown`

<sup>b</sup>`rowse00075`

<sup>K</sup>`cbBarHintsPlugin OnLeftDown`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbarhintsplugin')")`

<sup>K</sup>`OnLeftDown`

**`cbBarHintsPlugin::OnLeftUp`**

**`void OnLeftUp(cbLeftUpEvent& event)`**<sup>K</sup>

Handles a plugin event.

---

<sup>C</sup>`cbBarHintsPlugin::OnLeftUp`

<sup>C</sup>`bbarhintspluginonleftup`

<sup>b</sup>`rowse00076`

<sup>K</sup>`cbBarHintsPlugin OnLeftUp`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbarhintsplugin')")`

<sup>K</sup>`OnLeftUp`

**`cbBarHintsPlugin::OnMotion`**

**`void OnMotion(cbMotionEvent& event)`**<sup>K</sup>

Handles a plugin event.

---

`cbBarHintsPlugin::OnMotion`

`cbbarhintspluginonmotion`

`rowse00077`

`cbBarHintsPlugin OnMotion`

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

`OnMotion`

**`cbBarHintsPlugin::OnSizeBarWindow`**

**`void OnSizeBarWindow(cbSizeBarWndEvent& event)`**<sup>K</sup>

Handles a plugin event.

---

<sup>C</sup>`cbBarHintsPlugin::OnSizeBarWindow`

<sup>C</sup>`bbarhintspluginonsizebarwindow`

<sup>b</sup>`rowse00078`

<sup>K</sup>`cbBarHintsPlugin OnSizeBarWindow`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbarhintsplugin')")`

<sup>K</sup>`OnSizeBarWindow`

**`cbBarHintsPlugin::SetGrooveCount`**

**`void SetGrooveCount(int nGrooves)`**<sup>K</sup>

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

---

<sup>C</sup>`cbBarHintsPlugin::SetGrooveCount`  
<sup>C</sup>`bbarhintspluginsetgroovecount`  
<sup>b</sup>`rowse00079`  
<sup>K</sup>`cbBarHintsPlugin SetGrooveCount`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbarhintsplugin')")`  
<sup>K</sup>`SetGrooveCount`

`$#+K!` **cbBarInfo::cbBarInfo**

**cbBarInfo()**<sup>K</sup>

Constructor.

---

`C` bBarInfo::cbBarInfo

`C` bbarinfo**cbbarinfo**

`b` rowse00081

`K` cbBarInfo cbBarInfo

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

`K` cbBarInfo

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

`~cbBarInfo()`<sup>K</sup>

Destructor.

---

`c_bBarInfo::~~cbBarInfo`

`c_bbarinfodtor`

`b_rowse00082`

`K cbBarInfo ~cbBarInfo`

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

`K ~cbBarInfo`

`$#+KK!` **cbBarInfo::IsExpanded**

**bool IsExpanded() const**

Returns TRUE if this bar is expanded.

---

`C`bBarInfo::IsExpanded  
`C`bbarinfoisexpanded  
`b`rowse00083  
`K` cbBarInfo IsExpanded  
`K` IsExpanded  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbarinfo')")

`$#+KK!` **cbBarInfo::IsFixed**

**bool IsFixed() const**

Returns TRUE if this bar is fixed.

---

`C` bBarInfo::IsFixed

`C` bbarinfoisfixed

`b` rowse00084

`K` cbBarInfo IsFixed

`K` IsFixed

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

`$#+K!` **cbBarSpy::cbBarSpy**

**cbBarSpy(wxFrameLayout\* *pPanel*)**<sup>K</sup>

Constructor, taking a parent pane.

**cbBarSpy()**<sup>K</sup>

Default constructor.

---

`^`cbBarSpy::cbBarSpy

`^`cbbarspycbbarspy

`^`rowse00087

`^`cbBarSpy cbBarSpy

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

`^`cbBarSpy

`^`cbBarSpy

**`cbBarSpy::ProcessEvent`**

**`bool ProcessEvent(wxEvent& event)`**<sup>K</sup>

Performs special event processing.

---

<sup>C</sup>`cbBarSpy::ProcessEvent`

<sup>C</sup>`cbbarspyprocessevent`

<sup>b</sup>`rowse00088`

<sup>K</sup>`cbBarSpy ProcessEvent`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbbarspy)")`

<sup>K</sup>`ProcessEvent`

**`$#+K! cbBarSpy::SetBarWindow`**

**`void SetBarWindow(wxWindow* pWnd)K`**

Sets the bar window.

---

`CbBarSpy::SetBarWindow`

`Cbbarspysetbarwindow`

`browse00089`

`K cbBarSpy SetBarWindow`

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

`K SetBarWindow`

**`cbCloseBox::Draw`**

**`void Draw(wxDC& dc)`**

Draws the close button appearance.

---

`cbCloseBox::Draw`

`bcloseboxdraw`

`rowse00091`

`cbCloseBox Draw`

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

`Draw`

**`$#+K! cbCollapseBox::Draw`**

**`void Draw(wxDC& dc)K`**

Draws the collapse button appearance.

---

`CbCollapseBox::Draw  
Cbcollapseboxdraw  
browse00093  
K cbCollapseBox Draw  
EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `bcollapsebox')")  
K Draw`

**`cbCustomizeBarEvent::cbCustomizeBarEvent`**

**`cbCustomizeBarEvent(cbBarInfo* pBar, const wxPoint& clickPos, cbDockPane* pPane)`**<sup>K</sup>

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

---

<sup>C</sup>`bCustomizeBarEvent::cbCustomizeBarEvent`  
<sup>C</sup>`bcustomizebareventcbcustomizebarevent`  
<sup>b</sup>`rowse00096`  
<sup>K</sup>`cbCustomizeBarEvent cbCustomizeBarEvent`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `bcustomizebarevent')`)  
<sup>K</sup>`cbCustomizeBarEvent`

**`cbCustomizeLayoutEvent::cbCustomizeLayoutEvent`**

**`cbCustomizeLayoutEvent(const wxPoint& clickPos)`**<sup>K</sup>

Constructor, taking mouse position.

---

<sup>C</sup>`cbCustomizeLayoutEvent::cbCustomizeLayoutEvent`  
<sup>C</sup>`cbcustomizelayouteventcbcustomizelayoutevent`  
<sup>b</sup>`rowse00098`  
<sup>K</sup>`cbCustomizeLayoutEvent cbCustomizeLayoutEvent`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbcustomizelayoutevent')")`  
<sup>K</sup>`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.

---

```
cbDimInfo::cbDimInfo
cbdiminfocbdiminfo
browse00100
K cbDimInfo cbDimInfo
E nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdiminfo)")
K cbDimInfo
K cbDimInfo
K cbDimInfo
K cbDimInfo
```

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

`~cbDimInfo()`<sup>K</sup>

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()`<sup>K</sup>**

Returns the handler, if any.

---

<sup>C</sup>`bDimInfo::GetDimHandler`

<sup>C</sup>`bdiminfogetdimhandler`

<sup>b</sup>`rowse00102`

<sup>K</sup>`cbDimInfo GetDimHandler`

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

<sup>K</sup>`GetDimHandler`

`$#+K! cbDimInfo::operator=`

`const cbDimInfo& operator operator=(const cbDimInfo& other)K`

Assignment operator.

---

`CbDimInfo::operator=  
Cbdiminfooperatorassign  
browse00103  
K cbDimInfo operator=  
EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdiminfo')")  
K operator=`

`$#+K! cbDockBox::Draw`

`void Draw(wxDC& dc)K`

Draws the dock button appearance.

---

`CbDockBox::Draw`

`Cbdockboxdraw`

`browse00105`

`K cbDockBox Draw`

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

`K Draw`

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

**cbDockPane(int *alignment*, wxFrameLayout\* *pPanel*)**<sup>K</sup>

Constructor, taking alignment and layout panel.

**cbDockPane()**<sup>K</sup>

public members Default constructor.

---

`^`cbDockPane::cbDockPane

`^`cbdockpanecbdockpane

`b`rowse00107

`K` cbDockPane cbDockPane

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

`K` cbDockPane

`K` cbDockPane

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

`~cbDockPane()`<sup>K</sup>

Destructor.

---

`c_bDockPane::~cbDockPane`

`c_bdockpanedtor`

`b_rowse00108`

`K cbDockPane ~cbDockPane`

`E_nableButton("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.

---

`CbDockPane::BarPresent`

`Cbdockpanebarpresent`

`browse00109`

`K cbDockPane BarPresent`

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

`K BarPresent`

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

**`void CalcLengthRatios(cbRowInfo* pInRow)K`**

Calculate lengths. Internal function called by plugins.

---

`CbDockPane::CalcLengthRatios`

`Cbdockpanecalclengthratios`

`browse00110`

`K cbDockPane CalcLengthRatios`

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

`K CalcLengthRatios`

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

**void ContractBar**(`cbBarInfo*` *pBar*)<sup>K</sup>

Contracts the bar. Internal function called by plugins.

---

`^`cbDockPane::ContractBar

`^`bdockpanecontractbar

`^`rowse00111

`^`cbDockPane ContractBar

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

`^` ContractBar

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

**`void DoInsertBar(cbBarInfo* pBar, int rowNo)`<sup>K</sup>**

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

---

`CbDockPane::DoInsertBar`  
`Cbdockpanedoinsertbar`  
`browse00112`  
`K cbDockPane DoInsertBar`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdockpane')")`  
`K DoInsertBar`

<sup>S#+K!</sup>**cbDockPane::DrawHorizHandle**

**void DrawHorizHandle(wxDC& *dc*, int *x*, int *y*, int *width*)<sup>K</sup>**

Row/bar resizing related helper-method.

---

<sup>C</sup>bDockPane::DrawHorizHandle

<sup>C</sup>bdockpanedrawhorizhandle

<sup>b</sup>rowse00113

<sup>K</sup> cbDockPane DrawHorizHandle

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")

<sup>K</sup> 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`

`browse00114`

`KcbDockPane DrawVertHandle`

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

`KDrawVertHandle`

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

**`void ExpandBar(cbBarInfo* pBar)K`**

Expands the bar. Internal function called by plugins.

---

`CbDockPane::ExpandBar`

`Cbdockpaneexpandbar`

`browse00115`

`K cbDockPane ExpandBar`

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

`K ExpandBar`

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

**void FinishDrawInArea(const wxRect& area)<sup>K</sup>**

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

---

`CbDockPane::FinishDrawInArea`

`Cbdockpanefinishdrawinarea`

`browse00116`

`K cbDockPane FinishDrawInArea`

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

`K FinishDrawInArea`

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

**void FrameToPane(int\* x, int\* y)<sup>K</sup>**

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

**void FrameToPane(wxRect\* pRect)<sup>K</sup>**

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

---

`CbDockPane::FrameToPane`

`Cbdockpaneframetopane`

`browse00117`

`KcbDockPane FrameToPane`

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

`KFrameToPane`

`KFrameToPane`

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

**`int GetAlignment()`<sup>K</sup>**

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`

`rowse00118`

`cbDockPane GetAlignment`

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

`GetAlignment`

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

**`cbBarInfo*` `GetBarInfoByWindow(wxWindow* pBarWnd)`<sup>K</sup>**

Finds the bar information by corresponding window.

---

<sup>C</sup>`cbDockPane::GetBarInfoByWindow`

<sup>C</sup>`cbdockpanegetbarinfobywindow`

<sup>b</sup>`rowse00119`

<sup>K</sup>`cbDockPane GetBarInfoByWindow`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")`

<sup>K</sup>`GetBarInfoByWindow`

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

**void GetBarResizeRange**(**cbBarInfo\*** *pBar*, **int\*** *from*, **int\*** *till*, **bool** *forLeftHandle*)<sup>K</sup>

Returns the bar's resize range.

---

`^`cbDockPane::GetBarResizeRange

`^`bdockpanegetbarresizerange

`^`rowse00120

`^`cbDockPane GetBarResizeRange

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

`^`GetBarResizeRange

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

**int GetDockingState()**<sup>K</sup>

Returns wxCBAR\_DOCKED\_HORIZONTALLY if the alignment is top or bottom, or wxCBAR\_DOCKED\_VERTICALLY otherwise.

---

`^`cbDockPane::GetDockingState  
`^`bdockpanegetdockingstate  
`^`rowse00121  
`^`cbDockPane GetDockingState  
`^`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `bdockpane')")  
`^`GetDockingState

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

**`cbRowInfo*` `GetFirstRow()`<sup>K</sup>**

Returns the first row.

---

`^``cbDockPane::GetFirstRow`

`^``bdockpanegetfirstrow`

`^``rowse00122`

`^``cbDockPane GetFirstRow`

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

`^``GetFirstRow`

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

**`int GetMinimalRowHeight(cbRowInfo* pRow)`<sup>K</sup>**

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

---

`^`cbDockPane::GetMinimalRowHeight  
`^`bdockpanegetminimalrowheight  
`^`rowse00123  
`^`cbDockPane GetMinimalRowHeight  
`^`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdockpane')")  
`^`GetMinimalRowHeight

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

**`int GetNotFixedBarsCount(cbRowInfo* pRow)`<sup>K</sup>**

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

---

`^``cbDockPane::GetNotFixedBarsCount`  
`^``bdockpanegetnotfixedbarscount`  
`^``rowse00124`  
`^``cbDockPane GetNotFixedBarsCount`  
`^``nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdockpane')")`  
`^``GetNotFixedBarsCount`

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

**int GetPaneHeight()**<sup>K</sup>

Returns the height in the pane's coordinates.

---

<sup>C</sup>bDockPane::GetPaneHeight  
<sup>C</sup>bdockpanegetpaneheight  
<sup>b</sup>rowse00125  
<sup>K</sup> cbDockPane GetPaneHeight  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdockpane')")  
<sup>K</sup> GetPaneHeight

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

**`wxRect& GetRealRect()`**<sup>K</sup>

Returns the bounds of the pane, in parent coordinates.

---

<sup>C</sup>`bDockPane::GetRealRect`

<sup>C</sup>`bdockpanegetrealrect`

<sup>b</sup>`rowse00126`

<sup>K</sup>`cbDockPane GetRealRect`

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

<sup>K</sup>`GetRealRect`

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

**`cbRowInfo*` `GetRow(int row)`<sup>K</sup>**

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

---

`^` `cbDockPane::GetRow`  
`^` `cbdockpanegetrow`  
`^` `rowse00127`  
`^` `cbDockPane GetRow`  
`E` `nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdockpane')")`  
`^` `GetRow`

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

**`int GetRowAt(int paneY)`<sup>K</sup>**

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)`<sup>K</sup>**

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

---

`^cbDockPane::GetRowAt`

`^bdockpanegetrowat`

`^rowse00128`

`^cbDockPane GetRowAt`

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

`^GetRowAt`

`^GetRowAt`

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

**`int GetRowIndex(cbRowInfo* pRow)`<sup>K</sup>**

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

---

`^``cbDockPane::GetRowIndex`  
`^``cbdockpanegetrowindex`  
`^``rowse00129`  
`^``cbDockPane GetRowIndex`  
`^``enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdockpane')")`  
`^``GetRowIndex`

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

**RowArrayT& GetRowList()**<sup>K</sup>

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

---

`^`cbDockPane::GetRowList

`^`bdockpanegetrowlist

`^`rowse00130

`^`cbDockPane GetRowList

`^`nableButton("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.

---

`CcbDockPane::GetRowResizeRange`  
`Cbdockpanegetrowresizerange`  
`browse00131`  
`K cbDockPane GetRowResizeRange`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdockpane')")`  
`K GetRowResizeRange`

**`$#+K! 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`  
`browse00132`  
`K cbDockPane GetRowShapeData`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdockpane')")`  
`K GetRowShapeData`

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

**int GetRowY(cbRowInfo\* pRow)**<sup>K</sup>

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

---

`^`cbDockPane::GetRowY

`^`bdockpanegetrowy

`^`rowse00133

`^`cbDockPane GetRowY

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

`^`GetRowY

**`$#+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.

---

`CbDockPane::HasNotFixedBarsLeft`

`Cbdockpanehasnotfixedbarsleft`

`browse00134`

`K cbDockPane HasNotFixedBarsLeft`

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

`K HasNotFixedBarsLeft`

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

**bool HasNotFixedBarsRight(cbBarInfo\* pBar)<sup>K</sup>**

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

---

`CbDockPane::HasNotFixedBarsRight  
Cbdockpanehasnotfixedbarsright  
browse00135  
K cbDockPane HasNotFixedBarsRight  
EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdockpane')")  
K HasNotFixedBarsRight`

**`$#+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.

---

`CbDockPane::HasNotFixedRowsAbove`  
`Cbdockpanehasnotfixedrowsabove`  
`browse00136`  
`K cbDockPane HasNotFixedRowsAbove`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdockpane')")`  
`K HasNotFixedRowsAbove`

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

**bool HasNotFixedRowsBelow(cbRowInfo\* pRow)<sup>K</sup>**

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

---

<sup>C</sup>bDockPane::HasNotFixedRowsBelow

<sup>C</sup>bdockpanehasnotfixedrowsbelow

<sup>b</sup>rowse00137

<sup>K</sup> cbDockPane HasNotFixedRowsBelow

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

<sup>K</sup> HasNotFixedRowsBelow

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

**bool HasPoint(const wxPoint& pos, int x, int y, int width, int height)<sup>K</sup>**

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

---

`CbDockPane::HasPoint`

`Cbdockpanehaspoint`

`browse00138`

`K cbDockPane HasPoint`

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

`K HasPoint`

<sup>S#+K!</sup>**cbDockPane::HitTestPanelItems**

**int HitTestPanelItems(const wxPoint& pos, cbRowInfo\*\* ppRow, cbBarInfo\*\* ppBar)<sup>K</sup>**

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.

---

<sup>C</sup>bDockPane::HitTestPanelItems

<sup>C</sup>bdockpanehitestpaneitems

<sup>b</sup>rowse00139

<sup>K</sup> cbDockPane HitTestPanelItems

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")

<sup>K</sup> HitTestPanelItems

**`cbDockPane::InitLinksForRow`**

**`void InitLinksForRow(cbRowInfo* pRow)`**<sup>K</sup>

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

---

<sup>C</sup>`cbDockPane::InitLinksForRow`

<sup>C</sup>`cbdockpaneinitlinksforrow`

<sup>b</sup>`rowse00140`

<sup>K</sup>`cbDockPane InitLinksForRow`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")`

<sup>K</sup>`InitLinksForRow`

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

**void InitLinksForRows()**<sup>K</sup>

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

---

`^`cbDockPane::InitLinksForRows  
`^`bdockpaneinitlinksforrows  
`b`rowse00141  
`K` cbDockPane InitLinksForRows  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdockpane')")  
`K` InitLinksForRows

**`cbDockPane::InsertBar`**

**`void InsertBar(cbBarInfo* pBarInfo)`**<sup>K</sup>

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

**`void InsertBar(cbBarInfo* pBar, const wxRect& rect)`**<sup>K</sup>

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

**`void InsertBar(cbBarInfo* pBar, cbRowInfo* pIntoRow)`**<sup>K</sup>

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

---

`cbDockPane::InsertBar`

`cbdockpaneinsertbar`

`rowse00142`

`cbDockPane InsertBar`

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

`InsertBar`

`InsertBar`

`InsertBar`

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

**`void InsertRow(cbRowInfo* pRow, cbRowInfo* pBeforeRow)`<sup>K</sup>**

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`

`^``rowse00143`

`^``cbDockPane InsertRow`

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

`^``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.

---

`CbDockPane::IsFixedSize`  
`Cbdockpaneisfixedsize`  
`browse00144`  
`K cbDockPane IsFixedSize`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdockpane')")`  
`K IsFixedSize`

`$#+K!cbDockPane::IsHorizontal`

`bool IsHorizontal()`<sup>K</sup>

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

---

`^cbDockPane::IsHorizontal`

`^bdockpaneishorizontal`

`^rowse00145`

`^cbDockPane IsHorizontal`

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

`^IsHorizontal`

**`cbDockPane::MatchesMask`**

**`bool MatchesMask(int paneMask)`**<sup>K</sup>

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

---

<sup>C</sup>`cbDockPane::MatchesMask`

<sup>C</sup>`bdockpanematchesmask`

<sup>b</sup>`rowse00146`

<sup>K</sup>`cbDockPane MatchesMask`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")`

<sup>K</sup>`MatchesMask`

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

**void PaintBar**(**cbBarInfo\*** *pBar*, **wxDC&** *dc*)<sup>K</sup>

Calls PaintBarDecorations and PaintBarHandles. Internal function called by plugins.

---

`^`cbDockPane::PaintBar

`^`bdockpanepaintbar

`^`rowse00147

`^`cbDockPane PaintBar

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

`^`PaintBar

<sup>S#+K!</sup>**cbDockPane::PaintBarDecorations**

**void PaintBarDecorations(cbBarInfo\* pBar, wxDC& dc)<sup>K</sup>**

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.

---

<sup>C</sup>bDockPane::PaintBarDecorations

<sup>C</sup>bdockpanepaintbardecorations

<sup>b</sup>rowse00148

<sup>K</sup> cbDockPane PaintBarDecorations

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

<sup>K</sup> PaintBarDecorations

**`cbDockPane::PaintBarHandles`**

**`void PaintBarHandles(cbBarInfo* pBar, wxDC& dc)`**<sup>K</sup>

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

---

<sup>C</sup>`cbDockPane::PaintBarHandles`

<sup>C</sup>`cbdockpanepaintbarhandles`

<sup>b</sup>`rowse00149`

<sup>K</sup>`cbDockPane PaintBarHandles`

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

<sup>K</sup>`PaintBarHandles`

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

**void PaintPane(wxDC& dc)**<sup>K</sup>

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

---

`^`cbDockPane::PaintPane  
`^`bdockpanepaintpane  
`^`rowse00150  
`^`cbDockPane PaintPane  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdockpane')")  
`^`PaintPane

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

**`void PaintPaneBackground(wxDC& dc)`<sup>K</sup>**

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

---

`^``cbDockPane::PaintPaneBackground`

`^``bdockpanepaintpanebackground`

`^``rowse00151`

`^``cbDockPane PaintPaneBackground`

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

`^``PaintPaneBackground`

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

**`void PaintPaneDecorations(wxDC& dc)`<sup>K</sup>**

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

---

`CcbDockPane::PaintPaneDecorations`

`Cbdockpanepaintpanedecorations`

`browse00152`

`KcbDockPane PaintPaneDecorations`

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

`KPaintPaneDecorations`

`$$$K!cbDockPane::PaintRow`

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

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

---

`CbDockPane::PaintRow`

`Cbdockpanepaintrow`

`browse00153`

`KcbDockPane PaintRow`

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

`KPaintRow`

**`cbDockPane::PaintRowBackground`**

**`void PaintRowBackground(cbRowInfo* pRow, wxDC& dc)`**<sup>K</sup>

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

---

<sup>C</sup>`cbDockPane::PaintRowBackground`  
<sup>C</sup>`cbdockpanepaintrowbackground`  
<sup>b</sup>`rowse00154`  
<sup>K</sup>`cbDockPane PaintRowBackground`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")`  
<sup>K</sup>`PaintRowBackground`

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

**void PaintRowDecorations(cbRowInfo\* pRow, wxDC& dc)<sup>K</sup>**

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`

**`cbDockPane::PaintRowHandles`**

**`void PaintRowHandles(cbRowInfo* pRow, wxDC& dc)`**<sup>K</sup>

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

---

<sup>C</sup>`cbDockPane::PaintRowHandles`  
<sup>C</sup>`cbdockpanepaintrowhandles`  
<sup>b</sup>`rowse00156`  
<sup>K</sup>`cbDockPane PaintRowHandles`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdockpane')")`  
<sup>K</sup>`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.

---

`^cbDockPane::PaneToFrame`

`^bdockpanepanetoframe`

`^rowse00157`

`^cbDockPane PaneToFrame`

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

`^PaneToFrame`

`^PaneToFrame`

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

**`void RecalcLayout()`**<sup>K</sup>

Generates events to perform layout calculations.

---

<sup>C</sup>`bDockPane::RecalcLayout`

<sup>C</sup>`bdockpanerecalclayout`

<sup>b</sup>`rowse00158`

<sup>K</sup>`cbDockPane RecalcLayout`

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

<sup>K</sup>`RecalcLayout`

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

**`void RecalcRowLayout(cbRowInfo* pRow)`<sup>K</sup>**

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

---

<sup>C</sup>`cbDockPane::RecalcRowLayout`

<sup>C</sup>`bdockpanerecalcrowlayout`

<sup>b</sup>`rowse00159`

<sup>K</sup>`cbDockPane RecalcRowLayout`

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

<sup>K</sup>`RecalcRowLayout`

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

**`void RemoveBar(cbBarInfo* pBar)K`**

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

---

`CbDockPane::RemoveBar`

`Cbdockpaneremovebar`

`browse00160`

`K cbDockPane RemoveBar`

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

`K RemoveBar`

**`cbDockPane::RemoveRow`**

**`void RemoveRow(cbRowInfo* pRow)`**<sup>K</sup>

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

---

<sup>C</sup>`cbDockPane::RemoveRow`

<sup>C</sup>`cbdockpaneremoverow`

<sup>b</sup>`rowse00161`

<sup>K</sup>`cbDockPane RemoveRow`

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

<sup>K</sup>`RemoveRow`

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

**void ResizeBar**(**cbBarInfo\*** *pBar*, **int** *ofs*, **bool** *forLeftHandle*)<sup>K</sup>

Row/bar resizing related helper-method.

---

<sup>C</sup>cbDockPane::ResizeBar

<sup>C</sup>bdockpaneresizebar

<sup>b</sup>rowse00162

<sup>K</sup> cbDockPane ResizeBar

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")

<sup>K</sup> ResizeBar

<sup>S#+K!</sup>**cbDockPane::ResizeRow**

**void ResizeRow**(**cbRowInfo\*** *pRow*, **int** *ofs*, **bool** *forUpperHandle*)<sup>K</sup>

Row/bar resizing related helper-method.

---

<sup>C</sup>bDockPane::ResizeRow

<sup>C</sup>bdockpaneresizerow

<sup>b</sup>rowse00163

<sup>K</sup> cbDockPane ResizeRow

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")

<sup>K</sup> ResizeRow

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

**`void SetBoundsInParent(const wxRect& rect)`<sup>K</sup>**

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

---

<sup>C</sup>`cbDockPane::SetBoundsInParent`

<sup>C</sup>`bdockpanesetboundsinparent`

<sup>b</sup>`rowse00164`

<sup>K</sup>`cbDockPane SetBoundsInParent`

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

<sup>K</sup>`SetBoundsInParent`

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

**void SetMargins(int *top*, int *bottom*, int *left*, int *right*)<sup>K</sup>**

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

---

`CbDockPane::SetMargins`

`Cbdockpanesetmargins`

`browse00165`

`K cbDockPane SetMargins`

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

`K SetMargins`

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

**`void SetPaneWidth(int width)`<sup>K</sup>**

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

---

`^``cbDockPane::SetPaneWidth`

`^``bdockpanesetpanewidth`

`b``rowse00166`

`K``cbDockPane SetPaneWidth`

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

`K``SetPaneWidth`

<sup>S#+K!</sup>**cbDockPane::SetRowHeight**

**void SetRowHeight(cbRowInfo\* pRow, int newHeight)<sup>K</sup>**

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

---

<sup>C</sup>bDockPane::SetRowHeight  
<sup>C</sup>bdockpanesetrowheight  
<sup>b</sup>rowse00167  
<sup>K</sup> cbDockPane SetRowHeight  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdockpane')")  
<sup>K</sup> SetRowHeight

<sup>S#+K!</sup> **cbDockPane::SetRowShapeData**

**void SetRowShapeData**(**cbRowInfo\*** *pRowNode*, **wxList\*** *pLst*)<sup>K</sup>

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.

---

<sup>C</sup> **cbDockPane::SetRowShapeData**

<sup>C</sup> **cbdockpanesetrowshapedata**

<sup>b</sup> rowse00168

<sup>K</sup> **cbDockPane** **SetRowShapeData**

<sup>E</sup> **enableButton**("Up");**ChangeButtonBinding**("Up", "JumpId(`fl.hlp', `cbdockpane')")

<sup>K</sup> **SetRowShapeData**

**`cbDockPane::SizeBar`**

**`void SizeBar(cbBarInfo* pBar)`**<sup>K</sup>

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

---

`cbDockPane::SizeBar`

`cbdockpanesizebar`

`rowse00169`

`cbDockPane SizeBar`

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

`SizeBar`

**`cbDockPane::SizePaneObjects`**

**`void SizePaneObjects()`**<sup>K</sup>

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

---

<sup>C</sup>`cbDockPane::SizePaneObjects`

<sup>C</sup>`bdockpanesizepaneobjects`

<sup>b</sup>`rowse00170`

<sup>K</sup>`cbDockPane SizePaneObjects`

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

<sup>K</sup>`SizePaneObjects`

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

**`void SizeRowObjects(cbRowInfo* pRow)`<sup>K</sup>**

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

---

`^``cbDockPane::SizeRowObjects`

`^``bdockpanesizerowobjects`

`b``rowse00171`

`K``cbDockPane SizeRowObjects`

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

`K``SizeRowObjects`

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

**`wxDC*` `StartDrawInArea(const wxRect& area)`<sup>K</sup>**

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

---

<sup>C</sup>`cbDockPane::StartDrawInArea`

<sup>C</sup>`cbdockpanestartdrawinarea`

<sup>b</sup>`rowse00172`

<sup>K</sup>`cbDockPane StartDrawInArea`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdockpane')")`

<sup>K</sup>`StartDrawInArea`

**`cbDockPane::SyncRowFlags`**

**`void SyncRowFlags(cbRowInfo* pRow)`**

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`  
`rowse00173`  
`cbDockPane SyncRowFlags`  
`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdockpane')")`  
`SyncRowFlags`

**`cbDrawBarDecorEvent::cbDrawBarDecorEvent`**

**`cbDrawBarDecorEvent(cbBarInfo* pBar, wxDC& dc, cbDockPane* pPane)`**<sup>K</sup>

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

---

<sup>C</sup>`cbDrawBarDecorEvent::cbDrawBarDecorEvent`

<sup>C</sup>`cbdrawbardecoreventcbdrawbardecorevent`

<sup>b</sup>`rowse00175`

<sup>K</sup>`cbDrawBarDecorEvent cbDrawBarDecorEvent`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdrawbardecorevent')`

<sup>K</sup>`cbDrawBarDecorEvent`

**`cbDrawBarHandlesEvent::cbDrawBarHandlesEvent`**

**`cbDrawBarHandlesEvent(cbBarInfo* pBar, wxDC& dc, cbDockPane* pPane)`**<sup>K</sup>

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

---

<sup>C</sup>`cbDrawBarHandlesEvent::cbDrawBarHandlesEvent`

<sup>C</sup>`cbdrawbarhandleseventcbdrawbarhandlesevent`

<sup>b</sup>`rowse00177`

<sup>K</sup>`cbDrawBarHandlesEvent cbDrawBarHandlesEvent`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdrawbarhandlesevent')`

<sup>K</sup>`cbDrawBarHandlesEvent`

<sup>S#+K!</sup>**cbDrawHintRectEvent::cbDrawHintRectEvent**

**cbDrawHintRectEvent(const wxRect& *rect*, bool *isInClient*, bool *eraseRect*, bool *lastTime*)<sup>K</sup>**

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

---

<sup>C</sup>bDrawHintRectEvent::cbDrawHintRectEvent

<sup>C</sup>bdrawhintrecteventcbdrawhintrectevent

<sup>b</sup>rowse00179

<sup>K</sup> cbDrawHintRectEvent cbDrawHintRectEvent

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbdrawhintrectevent')")

<sup>K</sup> cbDrawHintRectEvent

**`cbDrawPaneBkGroundEvent::cbDrawPaneBkGroundEvent`**

**`cbDrawPaneBkGroundEvent(wxDC& dc, cbDockPane* pPane)`**<sup>K</sup>

Constructor, taking device context and pane.

---

<sup>C</sup>`cbDrawPaneBkGroundEvent::cbDrawPaneBkGroundEvent`

<sup>C</sup>`cbdrawpanebkgroundeventcbdrawpanebkgroundevent`

<sup>b</sup>`rowse00181`

<sup>K</sup>`cbDrawPaneBkGroundEvent cbDrawPaneBkGroundEvent`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdrawpanebkgroundevent')`

<sup>K</sup>`cbDrawPaneBkGroundEvent`

**`cbDrawPaneDecorEvent::cbDrawPaneDecorEvent`**

**`cbDrawPaneDecorEvent(wxDC& dc, cbDockPane* pPane)`**<sup>K</sup>

Constructor, taking device context and pane.

---

<sup>C</sup>`cbDrawPaneDecorEvent::cbDrawPaneDecorEvent`

<sup>C</sup>`cbdrawpanedecoreventcbdrawpanedecorevent`

<sup>b</sup>`rowse00183`

<sup>K</sup>`cbDrawPaneDecorEvent cbDrawPaneDecorEvent`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdrawpanedecorevent')`

<sup>K</sup>`cbDrawPaneDecorEvent`

**`cbDrawRowBkGroundEvent::cbDrawRowBkGroundEvent`**

**`cbDrawRowBkGroundEvent(cbRowInfo* pRow, wxDC& dc, cbDockPane* pPane)`**<sup>K</sup>

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

---

`cbDrawRowBkGroundEvent::cbDrawRowBkGroundEvent`

`cbdrawrowbkgroundeventcbdrawrowbkgroundevent`

`rowse00185`

`cbDrawRowBkGroundEvent cbDrawRowBkGroundEvent`

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

`cbDrawRowBkGroundEvent`

**`cbDrawRowDecorEvent::cbDrawRowDecorEvent`**

**`cbDrawRowDecorEvent(cbRowInfo* pRow, wxDC& dc, cbDockPane* pPane)`**<sup>K</sup>

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

---

<sup>C</sup>`cbDrawRowDecorEvent::cbDrawRowDecorEvent`

<sup>C</sup>`cbdrawrowdecoreventcbdrawrowdecorevent`

<sup>b</sup>`rowse00187`

<sup>K</sup>`cbDrawRowDecorEvent cbDrawRowDecorEvent`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdrawrowdecorevent')`

<sup>K</sup>`cbDrawRowDecorEvent`

**`cbDrawRowHandlesEvent::cbDrawRowHandlesEvent`**

**`cbDrawRowHandlesEvent(cbRowInfo* pRow, wxDC& dc, cbDockPane* pPane)`**<sup>K</sup>

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

---

<sup>C</sup>`cbDrawRowHandlesEvent::cbDrawRowHandlesEvent`

<sup>C</sup>`cbdrawrowhandleseventcbdrawrowhandlesevent`

<sup>b</sup>`rowse00189`

<sup>K</sup>`cbDrawRowHandlesEvent cbDrawRowHandlesEvent`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbdrawrowhandlesevent')")`

<sup>K</sup>`cbDrawRowHandlesEvent`

**`cbDynToolBarDimHandler::OnChangeBarState`**

**`void OnChangeBarState(cbBarInfo* pBar, int newState)`**<sup>K</sup>

Called when the bar changes state.

---

<sup>C</sup>`cbDynToolBarDimHandler::OnChangeBarState`

<sup>C</sup>`bdyntoolbardimhandleronchangebarstate`

<sup>b</sup>`rowse00191`

<sup>K</sup>`cbDynToolBarDimHandler OnChangeBarState`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `bdyntoolbardimhandler')")`

<sup>K</sup>`OnChangeBarState`

`cbDynToolBarDimHandler::OnResizeBar`

`void OnResizeBar(cbBarInfo* pBar, const wxSize& given, wxSize& preferred)`<sup>K</sup>

Called when a bar is resized.

---

`cbDynToolBarDimHandler::OnResizeBar`

`cbdyntoolbardimhandleronresizebar`

`rowse00192`

`cbDynToolBarDimHandler OnResizeBar`

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

`OnResizeBar`

`##+K! cbFinishDrawInAreaEvent::cbFinishDrawInAreaEvent`

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

Constructor, taking rectangular area and pane.

---

`CbFinishDrawInAreaEvent::cbFinishDrawInAreaEvent`

`Cbfinishdrawinareaeventcbfinishdrawinareaevent`

`browse00194`

`K cbFinishDrawInAreaEvent cbFinishDrawInAreaEvent`

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

`K cbFinishDrawInAreaEvent`

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

**`cbFloatedBarWindow()`<sup>K</sup>**

Default constructor.

---

`C` `cbFloatedBarWindow::cbFloatedBarWindow`  
`C` `cbfloatedbarwindowcbfloatedbarwindow`  
`b` `rowse00196`  
`K` `cbFloatedBarWindow cbFloatedBarWindow`  
`E` `nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbfloatedbarwindow')")`  
`K` `cbFloatedBarWindow`

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

**`cbBarInfo* GetBar()`**<sup>K</sup>

Returns the bar information for this window.

---

<sup>C</sup>`bFloatedBarWindow::GetBar`

<sup>C</sup>`bfloatedbarwindowgetbar`

<sup>b</sup>`rowse00197`

<sup>K</sup>`cbFloatedBarWindow GetBar`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbfloatedbarwindow')")`

<sup>K</sup>`GetBar`

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

**wxSize GetPreferredSize(const wxSize& *given*)**<sup>K</sup>

Overridden function returning the preferred size.

---

<sup>C</sup>bFloatedBarWindow::GetPreferredSize

<sup>C</sup>bfloatedbarwindowgetpreferredsize

<sup>b</sup>rowse00198

<sup>K</sup> cbFloatedBarWindow GetPreferredSize

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbfloatedbarwindow')")

<sup>K</sup> GetPreferredSize

**`cbFloatedBarWindow::HandleTitleClick`**

**`bool HandleTitleClick(wxMouseEvent& event)`**<sup>K</sup>

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`

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

**`void OnDbIClick(wxMouseEvent& event)K`**

Responds to double-click mouse events.

---

`CbFloatedBarWindow::OnDbIClick`

`Cbfloatedbarwindowondbclick`

`browse00200`

`K cbFloatedBarWindow OnDbIClick`

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

`K OnDbIClick`

**`cbFloatedBarWindow::OnMiniButtonClicked`**

**`void OnMiniButtonClicked(int btnIdx)`**<sup>K</sup>

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!` `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.

---

`CbFloatedBarWindow::PositionFloatedWnd`  
`Cbfloatedbarwindowpositionfloatedwnd`  
`browse00202`  
`K cbFloatedBarWindow PositionFloatedWnd`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbfloatedbarwindow')")`  
`K PositionFloatedWnd`

**`$#+K! 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)`**<sup>K</sup>

Sets the layout for this window.

---

<sup>C</sup>`cbFloatedBarWindow::SetLayout`

<sup>C</sup>`cbfloatedbarwindowsetlayout`

<sup>b</sup>`rowse00204`

<sup>K</sup>`cbFloatedBarWindow SetLayout`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbfloatedbarwindow')")`

<sup>K</sup>`SetLayout`

**`cbGCUpdatesMgr::cbGCUpdatesMgr`**

**`cbGCUpdatesMgr()`**<sup>K</sup>

Default constructor.

**`cbGCUpdatesMgr(wxFrameLayout* pPanel)`**<sup>K</sup>

Constructor, taking a frame layout.

---

`cbGCUpdatesMgr::cbGCUpdatesMgr`  
`cbgcupdatesmgrcbgcupdatesmgr`  
`rowse00206`  
`cbGCUpdatesMgr cbGCUpdatesMgr`  
`enableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbgcupdatesmgr')")`  
`cbGCUpdatesMgr`  
`cbGCUpdatesMgr`

**`cbGCUpdatesMgr::AddItem`**

**`void AddItem(wxList& itemList, cbBarInfo* pBar, cbDockPane* pPane, wxRect& curBounds, wxRect& prevBounds)`**<sup>K</sup>

Internal function for repositioning items.

---

<sup>C</sup>`cbGCUpdatesMgr::AddItem`

<sup>C</sup>`bgcupdatesmgradditem`

<sup>b</sup>`rowse00207`

<sup>K</sup>`cbGCUpdatesMgr AddItem`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `bgcupdatesmgr')`

<sup>K</sup>`AddItem`

**`cbGCUpdatesMgr::DoRepositionItems`**

**`void DoRepositionItems(wxList& items)`**<sup>K</sup>

Internal function for repositioning items.

---

<sup>C</sup>`cbGCUpdatesMgr::DoRepositionItems`

<sup>C</sup>`cbgcupdatesmgrdorepositionitems`

<sup>b</sup>`rowse00208`

<sup>K</sup>`cbGCUpdatesMgr DoRepositionItems`

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

<sup>K</sup>`DoRepositionItems`

**`$#+K! cbGCUpdatesMgr::OnStartChanges`**

**`void OnStartChanges()`**<sup>K</sup>

Receives notifications from the frame layout.

---

<sup>C</sup>`bGCUpdatesMgr::OnStartChanges`

<sup>C</sup>`bgcupdatesmgronstartchanges`

<sup>b</sup>`rowse00209`

<sup>K</sup>`cbGCUpdatesMgr OnStartChanges`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `bgcupdatesmgr')")`

<sup>K</sup>`OnStartChanges`

**`$#+K! cbGCUpdatesMgr::UpdateNow`**

**`void UpdateNow()`**<sup>K</sup>

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

---

`c` bGCUpdatesMgr::UpdateNow  
`c` bgcupdatesmgrupdatenow  
`b` rowse00210  
`K` cbGCUpdatesMgr UpdateNow  
`E` nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `bgcupdatesmgr')")  
`K` UpdateNow

`$#+K!` **cbHintAnimationPlugin::cbHintAnimationPlugin**

**cbHintAnimationPlugin()**<sup>K</sup>

Default constructor.

**cbHintAnimationPlugin(wxFrameLayout\* pPanel, int paneMask = wxALL\_PANES)**<sup>K</sup>

Constructor, taking a layout panel and pane mask.

---

`C` bHintAnimationPlugin::cbHintAnimationPlugin  
`C` bhintanimationplugincbhintanimationplugin  
`b` rowse00212  
`K` cbHintAnimationPlugin cbHintAnimationPlugin  
`E` nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbhintanimationplugin')")  
`K` cbHintAnimationPlugin  
`K` cbHintAnimationPlugin

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

`~cbHintAnimationPlugin()K`

Destructor.

---

`CbHintAnimationPlugin::~cbHintAnimationPlugin`

`Cbhintanimationplugindtor`

`browse00213`

`K cbHintAnimationPlugin ~cbHintAnimationPlugin`

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

`K ~cbHintAnimationPlugin`

**`cbHintAnimationPlugin::DoDrawHintRect`**

**`void DoDrawHintRect(wxRect& rect, bool isInClientRect)`**<sup>K</sup>

Internal function for drawing a hint rectangle.

---

<sup>C</sup>`cbHintAnimationPlugin::DoDrawHintRect`

<sup>C</sup>`bhintanimationplugindodrawhintrect`

<sup>b</sup>`rowse00214`

<sup>K</sup>`cbHintAnimationPlugin DoDrawHintRect`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbhintanimationplugin')")`

<sup>K</sup>`DoDrawHintRect`

`$#+K!` **cbHintAnimationPlugin::DrawHintRect**

**void DrawHintRect(wxRect& rect, bool isInClientRect)<sup>K</sup>**

Internal function for drawing a hint rectangle.

---

<sup>C</sup>cbHintAnimationPlugin::DrawHintRect

<sup>C</sup>bhintanimationplugindrawhintrect

<sup>b</sup>rowse00215

<sup>K</sup> cbHintAnimationPlugin DrawHintRect

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbhintanimationplugin')")

<sup>K</sup> DrawHintRect

<sup>S#+K!</sup>**cbHintAnimationPlugin::EraseHintRect**

**void EraseHintRect(wxRect& *rect*, bool *isInClientRect*)<sup>K</sup>**

Internal function for erasing a hint rectangle.

---

<sup>C</sup>bHintAnimationPlugin::EraseHintRect

<sup>C</sup>bhintanimationpluginerasehintrect

<sup>b</sup>rowse00216

<sup>K</sup> cbHintAnimationPlugin EraseHintRect

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbhintanimationplugin')")

<sup>K</sup> EraseHintRect

**`cbHintAnimationPlugin::FinishTracking`**

**`void FinishTracking()`**

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)`**<sup>K</sup>

Event handler respoding to hint draw events.

---

<sup>C</sup>`cbHintAnimationPlugin::OnDrawHintRect`

<sup>C</sup>`cbhintanimationpluginondrawhintrect`

<sup>b</sup>`rowse00218`

<sup>K</sup>`cbHintAnimationPlugin OnDrawHintRect`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbhintanimationplugin')")`

<sup>K</sup>`OnDrawHintRect`

**`cbHintAnimationPlugin::RectToScr`**

**`void RectToScr(wxRect& frameRect, wxRect& scrRect)`**<sup>K</sup>

Internal function for translating coordinates.

---

<sup>C</sup>`cbHintAnimationPlugin::RectToScr`

<sup>C</sup>`cbhintanimationpluginrecttoscr`

<sup>b</sup>`rowse00219`

<sup>K</sup>`cbHintAnimationPlugin RectToScr`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbhintanimationplugin')")`

<sup>K</sup>`RectToScr`

**`cbHintAnimationPlugin::StartTracking`**

**`void StartTracking()`**<sup>K</sup>

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

---

<sup>C</sup>`cbHintAnimationPlugin::StartTracking`

<sup>C</sup>`cbhintanimationpluginstarttracking`

<sup>b</sup>`rowse00220`

<sup>K</sup>`cbHintAnimationPlugin StartTracking`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbhintanimationplugin')")`

<sup>K</sup>`StartTracking`

**`$#+K!` `cbInsertBarEvent::cbInsertBarEvent`**

**`cbInsertBarEvent(cbBarInfo* pBar, cbRowInfo* pIntoRow, cbDockPane* pPane)`<sup>K</sup>**

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

---

`C` `cbInsertBarEvent::cbInsertBarEvent`  
`C` `cbinsertbareventcbinsertbarevent`  
`b` `rowse00222`  
`K` `cbInsertBarEvent cbInsertBarEvent`  
`E` `nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbinsertbarevent')")`  
`K` `cbInsertBarEvent`

<sup>S#+K!</sup>**cbLayoutRowEvent::cbLayoutRowEvent**

**cbLayoutRowEvent(cbRowInfo\* *pRow*, cbDockPane\* *pPane*)<sup>K</sup>**

Constructor, taking row information and pane.

---

<sup>C</sup>bLayoutRowEvent::cbLayoutRowEvent

<sup>C</sup>blayoutroweventcblayoutrowevent

<sup>b</sup>rowse00224

<sup>K</sup> cbLayoutRowEvent cbLayoutRowEvent

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cblayoutrowevent')")

<sup>K</sup> cbLayoutRowEvent



`$#+K!` **cbLeftDClickEvent::cbLeftDClickEvent**

**cbLeftDClickEvent(const wxPoint& pos, cbDockPane\* pPane)<sup>K</sup>**

Constructor, taking mouse position and pane.

---

`^`cbLeftDClickEvent::cbLeftDClickEvent

`^`bleftdclিকেবentcbleftdclিকেবent

`b`rowse00228

`K` cbLeftDClickEvent cbLeftDClickEvent

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

`K` cbLeftDClickEvent

<sup>S#+K!</sup>**cbLeftDownEvent::cbLeftDownEvent**

**cbLeftDownEvent(const wxPoint& pos, cbDockPane\* pPane)<sup>K</sup>**

Constructor, taking mouse position and pane.

---

<sup>C</sup>bLeftDownEvent::cbLeftDownEvent

<sup>C</sup>bleftdowneventcbleftdownevent

<sup>b</sup>rowse00230

<sup>K</sup> cbLeftDownEvent cbLeftDownEvent

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbleftdownevent')")

<sup>K</sup> cbLeftDownEvent

**`cbLeftUpEvent::cbLeftUpEvent`**

**`cbLeftUpEvent(const wxPoint& pos, cbDockPane* pPane)`**<sup>K</sup>

Constructor, taking mouse position and pane.

---

<sup>C</sup>`cbLeftUpEvent::cbLeftUpEvent`

<sup>C</sup>`cbLeftUpEvent::cbLeftUpEvent`

<sup>b</sup>`rowse00232`

<sup>K</sup>`cbLeftUpEvent cbLeftUpEvent`

<sup>E</sup>`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbLeftUpEvent')")`

<sup>K</sup>`cbLeftUpEvent`

`$#+K!` **cbMiniButton::cbMiniButton**

**cbMiniButton()**<sup>K</sup>

Default constructor.

---

<sup>C</sup> **cbMiniButton::cbMiniButton**

<sup>C</sup> **cbminibuttoncbminibutton**

<sup>b</sup> rowse00234

<sup>K</sup> **cbMiniButton cbMiniButton**

<sup>E</sup> **nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbminibutton')")**

<sup>K</sup> **cbMiniButton**

`$#+K!` **cbMiniButton::Draw**

**void Draw(wxDC& dc)<sup>K</sup>**

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

---

`CbMiniButton::Draw  
Cbminibuttondraw  
browse00235  
K cbMiniButton Draw  
EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbminibutton')")  
K Draw`

`$#+K!` **cbMiniButton::Enable**

**void Enable**(bool *enable*)<sup>K</sup>

Enable or disable the button.

---

<sup>C</sup>bMiniButton::Enable

<sup>C</sup>bminibuttonenable

<sup>b</sup>rowse00236

<sup>K</sup> cbMiniButton Enable

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbminibutton')")

<sup>K</sup> Enable

**`$#+K!` `cbMiniButton::HitTest`**

**`bool HitTest(const wxPoint& pos)`<sup>K</sup>**

Returns TRUE if the given position was over the button.

---

<sup>C</sup>`cbMiniButton::HitTest`

<sup>C</sup>`bminibuttonhittest`

<sup>b</sup>`rowse00237`

<sup>K</sup>`cbMiniButton HitTest`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbminibutton')")`

<sup>K</sup>`HitTest`

**`$#+K!` `cbMiniButton::IsPressed`**

**`bool IsPressed()`<sup>K</sup>**

Returns TRUE if this button is pressed.

---

`^``cbMiniButton::IsPressed`  
`^``bminibuttonispressed`  
`^``rowse00238`  
`^``cbMiniButton IsPressed`  
`E``nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbminibutton')")`  
`^``IsPressed`

**`cbMiniButton::OnLeftDown`**

**`void OnLeftDown(const wxPoint& pos)`**<sup>K</sup>

Responds to a left down event.

---

<sup>C</sup>`cbMiniButton::OnLeftDown`

<sup>C</sup>`bminibuttononleftdown`

<sup>b</sup>`rowse00239`

<sup>K</sup>`cbMiniButton OnLeftDown`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbminibutton')")`

<sup>K</sup>`OnLeftDown`

`$#+K!` **cbMiniButton::OnLeftUp**

**void OnLeftUp(const wxPoint& pos)<sup>K</sup>**

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)`**<sup>K</sup>

Responds to a mouse move event.

---

`cbMiniButton::OnMotion`

`cbminibuttononmotion`

`rowse00241`

`cbMiniButton OnMotion`

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

`OnMotion`

**`$#+K!` cbMiniButton::Refresh**

**`void Refresh()`<sup>K</sup>**

Refreshes the button.

---

`^`cbMiniButton::Refresh

`^`bminibuttonrefresh

`^`rowse00242

`^`cbMiniButton Refresh

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

`^`Refresh

**`$#+K!` cbMiniButton::Reset**

**`void Reset()`<sup>K</sup>**

Reset the button.

---

`c` bMiniButton::Reset

`c` bminibuttonreset

`b` rowse00243

`K` cbMiniButton Reset

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

`K` Reset

`$#+K!` **cbMiniButton::SetPos**

**void SetPos(const wxPoint& pos)<sup>K</sup>**

Set the position of the button.

---

`CbMiniButton::SetPos`  
`Cbminibuttonsetpos`  
`browse00244`  
`K cbMiniButton SetPos`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbminibutton')")`  
`K SetPos`

**`$#+K! cbMiniButton::WasClicked`**

**`bool WasClicked()`**<sup>K</sup>

Returns TRUE if the button was clicked.

---

<sup>C</sup>`bMiniButton::WasClicked`

<sup>C</sup>`bminibuttonwasclicked`

<sup>b</sup>`rowse00245`

<sup>K</sup>`cbMiniButton WasClicked`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbminibutton')")`

<sup>K</sup>`WasClicked`

**`$#+K!` `cbMotionEvent::cbMotionEvent`**

**`cbMotionEvent(const wxPoint& pos, cbDockPane* pPane)`<sup>K</sup>**

Constructor, taking mouse position and pane.

---

<sup>C</sup>`cbMotionEvent::cbMotionEvent`

<sup>C</sup>`cbmotioneventcbmotionevent`

<sup>b</sup>`rowse00247`

<sup>K</sup>`cbMotionEvent cbMotionEvent`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbmotionevent')")`

<sup>K</sup>`cbMotionEvent`

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

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

Constructor taking frame layout pane and a pane mask.

**`cbPaneDrawPlugin()K`**

Default constructor.

---

<sup>C</sup>`cbPaneDrawPlugin::cbPaneDrawPlugin`

<sup>C</sup>`cbpanedrawplugincbpanedrawplugin`

<sup>b</sup>`rowse00249`

<sup>K</sup>`cbPaneDrawPlugin cbPaneDrawPlugin`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbpanedrawplugin')")`

<sup>K</sup>`cbPaneDrawPlugin`

<sup>K</sup>`cbPaneDrawPlugin`

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

**`~cbPaneDrawPlugin()`<sup>K</sup>**

Destructor.

---

<sup>C</sup>`bPaneDrawPlugin::~cbPaneDrawPlugin`

<sup>C</sup>`bpanedrawplugindtor`

<sup>b</sup>`rowse00250`

<sup>K</sup>`cbPaneDrawPlugin ~cbPaneDrawPlugin`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbpanedrawplugin')")`

<sup>K</sup>`~cbPaneDrawPlugin`

`$#+K!` **cbPaneDrawPlugin::Clone**

**cbPluginBase\* Clone()**<sup>K</sup>

Clone function, returning a new instance of this class.

---

<sup>C</sup>bPaneDrawPlugin::Clone

<sup>C</sup>bpanedrawpluginclone

<sup>b</sup>rowse00251

<sup>K</sup> cbPaneDrawPlugin Clone

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbpanedrawplugin')")

<sup>K</sup> Clone

<sup>S#+K!</sup>**cbPaneDrawPlugin::DrawBarInnerShadeRect**

**void DrawBarInnerShadeRect(cbBarInfo\* pBar, wxDC& dc)<sup>K</sup>**

Internal helper: draws the inner bar shading.

---

<sup>C</sup>bPaneDrawPlugin::DrawBarInnerShadeRect  
<sup>C</sup>bpanedrawplugindrawbarinnershaderect  
<sup>b</sup>rowse00252  
<sup>K</sup> cbPaneDrawPlugin DrawBarInnerShadeRect  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbpanedrawplugin')")  
<sup>K</sup> DrawBarInnerShadeRect

**`cbPaneDrawPlugin::DrawDraggedHandle`**

**`void DrawDraggedHandle(const wxPoint& pos, cbDockPane& pane)`**<sup>K</sup>

Internal helper: draws the dragged handle.

---

<sup>C</sup>`cbPaneDrawPlugin::DrawDraggedHandle`  
<sup>C</sup>`cbpanedrawplugindrawdraggedhandle`  
<sup>b</sup>`rowse00253`  
<sup>K</sup>`cbPaneDrawPlugin DrawDraggedHandle`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbpanedrawplugin')")`  
<sup>K</sup>`DrawDraggedHandle`

**`cbPaneDrawPlugin::DrawLowerRowHandle`**

**`void DrawLowerRowHandle(cbRowInfo* pRow, wxDC& dc)`**<sup>K</sup>

Internal helper: draws the lower row handle.

---

<sup>C</sup>`cbPaneDrawPlugin::DrawLowerRowHandle`  
<sup>C</sup>`cbpanedrawplugindrawlowerrowhandle`  
<sup>b</sup>`rowse00254`  
<sup>K</sup>`cbPaneDrawPlugin DrawLowerRowHandle`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbpanedrawplugin')")`  
<sup>K</sup>`DrawLowerRowHandle`

**`cbPaneDrawPlugin::DrawLowerRowShades`**

**`void DrawLowerRowShades(cbRowInfo* pRow, wxDC& dc, int level)`**<sup>K</sup>

Internal helper: draws the lower row shading.

---

<sup>C</sup>`cbPaneDrawPlugin::DrawLowerRowShades`  
<sup>C</sup>`cbpanedrawplugindrawlowerrowshades`  
<sup>b</sup>`rowse00255`  
<sup>K</sup>`cbPaneDrawPlugin DrawLowerRowShades`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbpanedrawplugin')")`  
<sup>K</sup>`DrawLowerRowShades`

`$#+K!` **cbPaneDrawPlugin::DrawPaneShade**

**void DrawPaneShade(wxDC& dc, int alignment)<sup>K</sup>**

Internal helper: draws the pane shading.

---

`CbPaneDrawPlugin::DrawPaneShade`  
`Cbpanedrawplugindrawpaneshade`  
`browse00256`  
`K cbPaneDrawPlugin DrawPaneShade`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbpanedrawplugin')")`  
`K DrawPaneShade`

**`cbPaneDrawPlugin::DrawPaneShadeForRow`**

**`void DrawPaneShadeForRow(cbRowInfo* pRow, wxDC& dc)`**<sup>K</sup>

Internal helper: draws the pane shading for a row.

---

<sup>C</sup>`cbPaneDrawPlugin::DrawPaneShadeForRow`  
<sup>C</sup>`cbpanedrawplugindrawpaneshadeforrow`  
<sup>b</sup>`rowse00257`  
<sup>K</sup>`cbPaneDrawPlugin DrawPaneShadeForRow`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbpanedrawplugin')")`  
<sup>K</sup>`DrawPaneShadeForRow`

`$$$K!cbPaneDrawPlugin::DrawShade`

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

Internal helper: draws shading.

---

`CbPaneDrawPlugin::DrawShade  
Cbpanedrawplugindrawshade  
browse00258  
KcbPaneDrawPlugin DrawShade  
EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbpanedrawplugin')")  
KDrawShade`

`$#+K!` **cbPaneDrawPlugin::DrawShade1**

**void DrawShade1**(int *level*, **wxRect&** *rect*, int *alignment*, **wxDC&** *dc*)<sup>K</sup>

Internal helper: draws shading.

---

<sup>C</sup>cbPaneDrawPlugin::DrawShade1  
<sup>C</sup>cbpanedrawplugindrawshade1  
<sup>b</sup>rowse00259  
<sup>K</sup> cbPaneDrawPlugin DrawShade1  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbpanedrawplugin')")  
<sup>K</sup> DrawShade1

**`cbPaneDrawPlugin::DrawUpperRowHandle`**

**`void DrawUpperRowHandle(cbRowInfo* pRow, wxDC& dc)`**<sup>K</sup>

Internal helper: draws the upper row handle.

---

<sup>C</sup>`cbPaneDrawPlugin::DrawUpperRowHandle`  
<sup>C</sup>`cbpanedrawplugindrawupperrowhandle`  
<sup>b</sup>`rowse00260`  
<sup>K</sup>`cbPaneDrawPlugin DrawUpperRowHandle`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbpanedrawplugin')")`  
<sup>K</sup>`DrawUpperRowHandle`

**`cbPaneDrawPlugin::DrawUpperRowShades`**

**`void DrawUpperRowShades(cbRowInfo* pRow, wxDC& dc, int level)`**<sup>K</sup>

Internal helper: draws the upper row shading.

---

<sup>C</sup>`cbPaneDrawPlugin::DrawUpperRowShades`  
<sup>C</sup>`cbpanedrawplugindrawupperrowshades`  
<sup>b</sup>`rowse00261`  
<sup>K</sup>`cbPaneDrawPlugin DrawUpperRowShades`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbpanedrawplugin')")`  
<sup>K</sup>`DrawUpperRowShades`

**`cbPaneDrawPlugin::OnDrawBarDecorations`**

**`void OnDrawBarDecorations(cbDrawBarDecorEvent& event)`**<sup>K</sup>

Handler for draw bar decorations events.

---

<sup>C</sup>`cbPaneDrawPlugin::OnDrawBarDecorations`

<sup>C</sup>`bpanedrawpluginondrawbardecorations`

<sup>b</sup>`rowse00262`

<sup>K</sup>`cbPaneDrawPlugin OnDrawBarDecorations`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbpanedrawplugin')")`

<sup>K</sup>`OnDrawBarDecorations`

**`cbPaneDrawPlugin::OnDrawBarHandles`**

**`void OnDrawBarHandles(cbDrawBarHandlesEvent& event)`**<sup>K</sup>

Handler for draw bar handles events.

---

<sup>C</sup>`cbPaneDrawPlugin::OnDrawBarHandles`  
<sup>C</sup>`bpanedrawpluginondrawbarhandles`  
<sup>b</sup>`rowse00263`  
<sup>K</sup>`cbPaneDrawPlugin OnDrawBarHandles`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbpanedrawplugin')")`  
<sup>K</sup>`OnDrawBarHandles`

**`cbPaneDrawPlugin::OnDrawPaneBackground`**

**`void OnDrawPaneBackground(cbDrawPaneBkGroundEvent& event)`**<sup>K</sup>

Handler for draw pane background events.

---

<sup>C</sup>`bPaneDrawPlugin::OnDrawPaneBackground`  
<sup>C</sup>`bpanedrawpluginondrawpanebackground`  
<sup>b</sup>`rowse00264`  
<sup>K</sup>`cbPaneDrawPlugin OnDrawPaneBackground`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbpanedrawplugin')")`  
<sup>K</sup>`OnDrawPaneBackground`

**`cbPaneDrawPlugin::OnDrawPaneDecorations`**

**`void OnDrawPaneDecorations(cbDrawPaneDecorEvent& event)`**<sup>K</sup>

Handler for draw pane decoration events.

---

<sup>C</sup>`cbPaneDrawPlugin::OnDrawPaneDecorations`

<sup>C</sup>`cbpanedrawpluginondrawpanedecorations`

<sup>b</sup>`rowse00265`

<sup>K</sup>`cbPaneDrawPlugin OnDrawPaneDecorations`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbpanedrawplugin')")`

<sup>K</sup>`OnDrawPaneDecorations`

**`cbPaneDrawPlugin::OnDrawRowBackground`**

**`void OnDrawRowBackground(cbDrawRowBkGroundEvent& event)`**<sup>K</sup>

Handler for draw row background events.

---

<sup>C</sup>`cbPaneDrawPlugin::OnDrawRowBackground`  
<sup>C</sup>`cbpanedrawpluginondrawrowbackground`  
<sup>b</sup>`rowse00266`  
<sup>K</sup>`cbPaneDrawPlugin OnDrawRowBackground`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbpanedrawplugin')")`  
<sup>K</sup>`OnDrawRowBackground`

<sup>S#+K!</sup> **cbPaneDrawPlugin::OnDrawRowDecorations**

**void OnDrawRowDecorations(cbDrawRowDecorEvent& *event*)<sup>K</sup>**

Handler for draw row decoration events.

---

<sup>C</sup>cbPaneDrawPlugin::OnDrawRowDecorations

<sup>C</sup>cbpanedrawpluginondrawrowdecorations

<sup>b</sup>rowse00267

<sup>K</sup>cbPaneDrawPlugin OnDrawRowDecorations

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbpanedrawplugin')")

<sup>K</sup>OnDrawRowDecorations

**<sup>S#+K!</sup>cbPaneDrawPlugin::OnDrawRowHandles**

**void OnDrawRowHandles(cbDrawRowHandlesEvent& *event*)<sup>K</sup>**

Handler for draw row handles events.

---

<sup>C</sup>bPaneDrawPlugin::OnDrawRowHandles

<sup>C</sup>bpanedrawpluginondrawrowhandles

<sup>b</sup>rowse00268

<sup>K</sup> cbPaneDrawPlugin OnDrawRowHandles

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbpanedrawplugin')")

<sup>K</sup> OnDrawRowHandles

**`cbPaneDrawPlugin::OnFinishDrawInArea`**

**`void OnFinishDrawInArea(cbFinishDrawInAreaEvent& event)`**<sup>K</sup>

Handler for finish draw in area events.

---

<sup>C</sup>`cbPaneDrawPlugin::OnFinishDrawInArea`  
<sup>C</sup>`bpanedrawpluginonfinishdrawinarea`  
<sup>b</sup>`rowse00269`  
<sup>K</sup>`cbPaneDrawPlugin OnFinishDrawInArea`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbpanedrawplugin')")`  
<sup>K</sup>`OnFinishDrawInArea`

**`cbPaneDrawPlugin::OnLButtonDown`**

**`void OnLButtonDown(cbLeftDownEvent& event)`**<sup>K</sup>

Handler for left mouse button down events.

---

<sup>C</sup>`cbPaneDrawPlugin::OnLButtonDown`  
<sup>C</sup>`cbpanedrawpluginonlbuttondown`  
<sup>b</sup>`rowse00270`  
<sup>K</sup>`cbPaneDrawPlugin OnLButtonDown`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbpanedrawplugin')")`  
<sup>K</sup>`OnLButtonDown`

**`$#+K!` `cbPaneDrawPlugin::OnLButtonUp`**

**`void OnLButtonUp(cbLeftUpEvent& event)`<sup>K</sup>**

Handler for left mouse button up events.

---

`^`cbPaneDrawPlugin::OnLButtonUp  
`^`bpanedrawpluginonlbuttonup  
`^`rowse00271  
`^`cbPaneDrawPlugin OnLButtonUp  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbpanedrawplugin')")  
`^` OnLButtonUp

**`cbPaneDrawPlugin::OnLDbIcIck`**

**`void OnLDbIcIck(cbLeftDcIckEvent& event)`**<sup>K</sup>

Handler for left double-click mouse button down events.

---

<sup>C</sup>`cbPaneDrawPlugin::OnLDbIcIck`  
<sup>C</sup>`bpanedrawpluginonldbclick`  
<sup>b</sup>`rowse00272`  
<sup>K</sup>`cbPaneDrawPlugin OnLDbIcIck`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbpanedrawplugin')")`  
<sup>K</sup>`OnLDbIcIck`

**`$#+K!` `cbPaneDrawPlugin::OnMouseMove`**

**`void OnMouseMove(cbMotionEvent& event)`<sup>K</sup>**

Handler for mouse move events.

---

`^`cbPaneDrawPlugin::OnMouseMove  
`^`cbpanedrawpluginonmousemove  
`^`rowse00273  
`^`cbPaneDrawPlugin OnMouseMove  
`^`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbpanedrawplugin')")  
`^` OnMouseMove

**`cbPaneDrawPlugin::OnRButtonUp`**

**`void OnRButtonUp(cbRightUpEvent& event)`**<sup>K</sup>

Handler for right mouse button up events.

---

<sup>C</sup>`cbPaneDrawPlugin::OnRButtonUp`  
<sup>C</sup>`cbpanedrawpluginonrbuttonup`  
<sup>b</sup>`rowse00274`  
<sup>K</sup>`cbPaneDrawPlugin OnRButtonUp`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbpanedrawplugin')")`  
<sup>K</sup>`OnRButtonUp`

**`$#+K!` `cbPaneDrawPlugin::OnSizeBarWindow`**

**`void OnSizeBarWindow(cbSizeBarWndEvent& event)K`**

Handler for bar size events.

---

`CcbPaneDrawPlugin::OnSizeBarWindow`

`Cbpanedrawpluginonsizebarwindow`

`browse00275`

`KcbPaneDrawPlugin OnSizeBarWindow`

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

`K OnSizeBarWindow`

**`$#+K!` cbPaneDrawPlugin::OnStartDrawInArea**

**`void OnStartDrawInArea(cbStartDrawInAreaEvent& event)`<sup>K</sup>**

Handler for start draw in area events.

---

<sup>C</sup>bPaneDrawPlugin::OnStartDrawInArea

<sup>C</sup>bpanedrawpluginonstartdrawinarea

<sup>b</sup>rowse00276

<sup>K</sup> cbPaneDrawPlugin OnStartDrawInArea

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbpanedrawplugin')")

<sup>K</sup> OnStartDrawInArea

**`$#+K! cbPaneDrawPlugin::SetDarkPixel`**

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

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

---

`CbPaneDrawPlugin::SetDarkPixel`  
`Cbpanedrawpluginsetdarkpixel`  
`browse00277`  
`K cbPaneDrawPlugin SetDarkPixel`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbpanedrawplugin')")`  
`K SetDarkPixel`

**`cbPaneDrawPlugin::SetLightPixel`**

**`void SetLightPixel(int x, int y, wxDC& dc)`**<sup>K</sup>

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

---

<sup>C</sup>`cbPaneDrawPlugin::SetLightPixel`  
<sup>C</sup>`cbpanedrawpluginsetlightpixel`  
<sup>b</sup>`rowse00278`  
<sup>K</sup>`cbPaneDrawPlugin SetLightPixel`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbpanedrawplugin')")`  
<sup>K</sup>`SetLightPixel`

`$$$K! 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`

**`$#+K! cbPluginBase::~~cbPluginBase`**

**`~cbPluginBase()`<sup>K</sup>**

Destructor. Destroys the whole plugin chain of connected plugins.

---

`c` bPluginBase::~~cbPluginBase

`c` bpluginbasedtor

`b` rowse00281

`K` cbPluginBase ~cbPluginBase

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

`K` ~cbPluginBase

**`$#+K!` cbPluginBase::GetPaneMask**

**`int GetPaneMask()`<sup>K</sup>**

Returns the pane mask.

---

<sup>C</sup>bPluginBase::GetPaneMask  
<sup>C</sup>bpluginbasegetpanemask  
<sup>b</sup>rowse00282  
<sup>K</sup> cbPluginBase GetPaneMask  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbpluginbase')")  
<sup>K</sup> GetPaneMask

`$#+K!` **cbPluginBase::IsReady**

**bool IsReady()**<sup>K</sup>

Returns TRUE if the plugin is ready to receive events.

---

`^`cbPluginBase::IsReady  
`^`cbpluginbaseisready  
`^`rowse00283  
`^`cbPluginBase IsReady  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbpluginbase)")  
`^`IsReady

`$$$K!cbPluginBase::OnInitPlugin`

`void OnInitPlugin()K`

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

`$#+K!` **cbPluginBase::ProcessEvent**

**bool ProcessEvent(wxEvent& event)**<sup>K</sup>

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).

---

`C` bPluginBase::ProcessEvent  
`C` bpluginbaseprocessevent  
`b` rowse00285  
`K` cbPluginBase ProcessEvent  
`E` nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbpluginbase')")  
`K` ProcessEvent

`##+K! cbPluginEvent::cbPluginEvent`

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

Constructor, taking event type and pane.

---

`CbPluginEvent::cbPluginEvent`

`Cbplugineventcbpluginevent`

`browse00287`

`K cbPluginEvent cbPluginEvent`

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

`K cbPluginEvent`

`$#+KKl` **cbPluginEvent::Clone**

**wxEvent\* Clone() const**

Not used, but required.

---

`^`cbPluginEvent::Clone  
`^`cbplugineventclone  
`b`rowse00288  
`K` cbPluginEvent Clone  
`K` Clone  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbpluginevent')")

**`cbRemoveBarEvent::cbRemoveBarEvent`**

**`cbRemoveBarEvent(cbBarInfo* pBar, cbDockPane* pPane)`<sup>K</sup>**

Constructor, taking bar information and pane.

---

<sup>C</sup>`cbRemoveBarEvent::cbRemoveBarEvent`

<sup>C</sup>`bremovebareventcbremovebarevent`

<sup>b</sup>`rowse00290`

<sup>K</sup>`cbRemoveBarEvent cbRemoveBarEvent`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbremovebarevent')")`

<sup>K</sup>`cbRemoveBarEvent`

**`$#+K!` `cbResizeBarEvent::cbResizeBarEvent`**

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

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

---

`CbResizeBarEvent::cbResizeBarEvent`

`Cbresizebareventcbresizebarevent`

`browse00292`

`KcbResizeBarEvent cbResizeBarEvent`

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

`KcbResizeBarEvent`

`##+K! cbResizeRowEvent::cbResizeRowEvent`

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

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

---

`CcbResizeRowEvent::cbResizeRowEvent`

`Cbresizeroweentcbresizeroweent`

`browse00294`

`KcbResizeRowEvent cbResizeRowEvent`

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

`KcbResizeRowEvent`

`##+K! cbRightDownEvent::cbRightDownEvent`

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

Constructor, taking mouse position and pane.

---

`^cbRightDownEvent::cbRightDownEvent  
^brightdowneventcbrightdownevent  
^rowse00296  
^cbRightDownEvent cbRightDownEvent  
^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbrightdownevent')")  
^cbRightDownEvent`

**`cbRightUpEvent::cbRightUpEvent`**

**`cbRightUpEvent(const wxPoint& pos, cbDockPane* pPane)`**<sup>K</sup>

Constructor, taking mouse position and pane.

---

<sup>C</sup> bRightUpEvent::cbRightUpEvent  
<sup>C</sup> brightupeventcbrightupevent  
<sup>b</sup> rowse00298  
<sup>K</sup> cbRightUpEvent cbRightUpEvent  
<sup>E</sup> nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbrightupevent')")  
<sup>K</sup> cbRightUpEvent

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

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

Constructor, taking paren layout frame and pane mask.

**`cbRowDragPlugin()K`**

Default constructor.

---

`CbRowDragPlugin::cbRowDragPlugin`

`Cbrowdragpluginbrowdragplugin`

`browse00300`

`K cbRowDragPlugin cbRowDragPlugin`

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

`K cbRowDragPlugin`

`K cbRowDragPlugin`

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

`~cbRowDragPlugin()`<sup>K</sup>

Destructor.

---

`C_bRowDragPlugin::~~cbRowDragPlugin`

`C_browdragplugindtor`

`b_rowse00301`

`K cbRowDragPlugin ~cbRowDragPlugin`

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

`K ~cbRowDragPlugin`

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

**wxBitmap\* CaptureDCArea(wxDC& dc, wxRect& area)<sup>K</sup>**

Helper for drag and drop.

---

`^`cbRowDragPlugin::CaptureDCArea

`^`cbrowdragplugincapturedcare

`b`rowse00302

`K` cbRowDragPlugin CaptureDCArea

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

`K` CaptureDCArea

**`cbRowDragPlugin::CheckPrevItemInFocus`**

**`void CheckPrevItemInFocus(cbRowInfo* pRow, int iconIdx)`**

Helper for drag and drop.

---

`cbRowDragPlugin::CheckPrevItemInFocus`  
`browdragplugincheckpreviteminfocus`  
`rowse00303`  
`cbRowDragPlugin CheckPrevItemInFocus`  
`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbrowdragplugin')")`  
`CheckPrevItemInFocus`

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

**cbPluginBase\* Clone()**<sup>K</sup>

Clone function, returning a new instance of this class.

---

`^`cbRowDragPlugin::Clone  
`^`browdragpluginclone  
`b`rowse00304  
`K` cbRowDragPlugin Clone  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")  
`K` Clone

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

**`void CollapseRow(cbRowInfo* pRow)`<sup>K</sup>**

Helper for drag and drop.

---

`^`cbRowDragPlugin::CollapseRow  
`^`browdragplugincollapserow  
`b`rowse00305  
`K` cbRowDragPlugin CollapseRow  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")  
`K` CollapseRow

**`cbRowDragPlugin::Draw3DPattern`**

**`void Draw3DPattern(wxRect& inRect, wxDC& dc)`**<sup>K</sup>

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

---

<sup>C</sup>`cbRowDragPlugin::Draw3DPattern`

<sup>C</sup>`browdragplugindraw3dpattern`

<sup>b</sup>`rowse00306`

<sup>K</sup>`cbRowDragPlugin Draw3DPattern`

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

<sup>K</sup>`Draw3DPattern`

**`cbRowDragPlugin::Draw3DRect`**

**`void Draw3DRect(wxRect& inRect, wxDC& dc, wxBrush& bkBrush)`**<sup>K</sup>

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

---

<sup>C</sup>`bRowDragPlugin::Draw3DRect`

<sup>C</sup>`browdragplugindraw3direct`

<sup>b</sup>`rowse00307`

<sup>K</sup>`cbRowDragPlugin Draw3DRect`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")`

<sup>K</sup>`Draw3DRect`

**`cbRowDragPlugin::DrawCollapsedRowIcon`**

**`void DrawCollapsedRowIcon(int index, wxDC& dc, bool isHighlighted)`**<sup>K</sup>

Draws collapsed row icon (appearance-dependent).

---

<sup>C</sup>`cbRowDragPlugin::DrawCollapsedRowIcon`

<sup>C</sup>`browdragplugindrawcollapsedrowicon`

<sup>b</sup>`rowse00308`

<sup>K</sup>`cbRowDragPlugin DrawCollapsedRowIcon`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")`

<sup>K</sup>`DrawCollapsedRowIcon`

**`cbRowDragPlugin::DrawCollapsedRowsBorder`**

**`void DrawCollapsedRowsBorder(wxDC& dc)`**<sup>K</sup>

Draws collapsed rows border (appearance-dependent).

---

<sup>C</sup>`cbRowDragPlugin::DrawCollapsedRowsBorder`

<sup>C</sup>`browdragplugindrawcollapsedrowsborder`

<sup>b</sup>`rowse00309`

<sup>K</sup>`cbRowDragPlugin DrawCollapsedRowsBorder`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")`

<sup>K</sup>`DrawCollapsedRowsBorder`

**`cbRowDragPlugin::DrawEmptyRow`**

**`void DrawEmptyRow(wxDC& dc, wxRect& rowBounds)`**<sup>K</sup>

Draws empty row (appearance-dependent).

---

<sup>C</sup>`bRowDragPlugin::DrawEmptyRow`  
<sup>C</sup>`browdragplugindrawemptyrow`  
<sup>b</sup>`rowse00310`  
<sup>K</sup>`cbRowDragPlugin DrawEmptyRow`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbrowdragplugin')")`  
<sup>K</sup>`DrawEmptyRow`

**`cbRowDragPlugin::DrawOrtoRomb`**

**`void DrawOrtoRomb(wxRect& inRect, wxDC& dc, wxBrush& bkBrush)`**<sup>K</sup>

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

---

<sup>C</sup>`bRowDragPlugin::DrawOrtoRomb`

<sup>C</sup>`browdragplugindrawortoromb`

<sup>b</sup>`rowse00311`

<sup>K</sup>`cbRowDragPlugin DrawOrtoRomb`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")`

<sup>K</sup>`DrawOrtoRomb`

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

**void DrawRectShade(wxRect& *inRect*, wxDC& *dc*, int *level*, wxPen& *upperPen*, wxPen& *lowerPen*)<sup>K</sup>**

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)`**<sup>K</sup>

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

---

<sup>C</sup>`bRowDragPlugin::DrawRomb`  
<sup>C</sup>`browdragplugindrawromb`  
<sup>b</sup>`rowse00313`  
<sup>K</sup>`cbRowDragPlugin DrawRomb`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbrowdragplugin)")`  
<sup>K</sup>`DrawRomb`

**`$#+K! 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)`**<sup>K</sup>

Draws row drag hint (appearance-dependent).

---

<sup>C</sup>`cbRowDragPlugin::DrawRowDragHint`

<sup>C</sup>`browdragplugindrawrowdraghint`

<sup>b</sup>`rowse00315`

<sup>K</sup>`cbRowDragPlugin DrawRowDragHint`

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

<sup>K</sup>`DrawRowDragHint`

**`cbRowDragPlugin::DrawRowsDragHintsBorder`**

**`void DrawRowsDragHintsBorder(wxDC& dc)`**<sup>K</sup>

Draws rows drag hints border (appearance-dependent).

---

<sup>C</sup>`bRowDragPlugin::DrawRowsDragHintsBorder`

<sup>C</sup>`browdragplugindrawrowsdraghintsborder`

<sup>b</sup>`rowse00316`

<sup>K</sup>`cbRowDragPlugin DrawRowsDragHintsBorder`

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

<sup>K</sup>`DrawRowsDragHintsBorder`

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

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

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

---

`CbRowDragPlugin::DrawTrianDown`  
`Cbrowdragplugindrawtriandown`  
`browse00317`  
`K cbRowDragPlugin DrawTrianDown`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")`  
`K DrawTrianDown`

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

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

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

---

`CbRowDragPlugin::DrawTrianRight`  
`Cbrowdragplugindrawtrianright`  
`browse00318`  
`K cbRowDragPlugin DrawTrianRight`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbrowdragplugin)")`  
`K DrawTrianRight`

**`$#+K! 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*)<sup>K</sup>

Helper for drag and drop.

---

<sup>C</sup>bRowDragPlugin::ExpandRow  
<sup>C</sup>browdragpluginexpandrow  
<sup>b</sup>rowse00320  
<sup>K</sup> cbRowDragPlugin ExpandRow  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")  
<sup>K</sup> ExpandRow

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

**`void FinishOnScreenDraw()`<sup>K</sup>**

Helper for drag and drop.

---

`c` bRowDragPlugin::FinishOnScreenDraw  
`c` browdragpluginfinishonscreendraw  
`b` rowse00321  
`K` cbRowDragPlugin FinishOnScreenDraw  
`E` nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")  
`K` FinishOnScreenDraw

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

**`int GetCollapsedIconsPos()`<sup>K</sup>**

Helper for drag and drop.

---

`^`cbRowDragPlugin::GetCollapsedIconsPos  
`^`browdragplugingetcollapsediconspos  
`b`rowse00322  
`K` cbRowDragPlugin GetCollapsedIconsPos  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbrowdragplugin)")  
`K` GetCollapsedIconsPos

`##+K! cbRowDragPlugin::GetCollapsedInconRect`

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

Helper for drag and drop.

---

`CbRowDragPlugin::GetCollapsedInconRect`  
`Cbrowdragplugingetcollapsedinconrect`  
`browse00323`  
`K cbRowDragPlugin GetCollapsedInconRect`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbrowdragplugin')")`  
`K GetCollapsedInconRect`

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

**`int GetCollapsedRowIconHeight()`**<sup>K</sup>

Gets the collapsed row icon height.

---

<sup>C</sup>`bRowDragPlugin::GetCollapsedRowIconHeight`  
<sup>C</sup>`browdragplugingetcollapsedrowiconheight`  
<sup>b</sup>`rowse00324`  
<sup>K</sup>`cbRowDragPlugin GetCollapsedRowIconHeight`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbrowdragplugin)")`  
<sup>K</sup>`GetCollapsedRowIconHeight`

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

**`cbRowInfo*` `GetFirstRow()`<sup>K</sup>**

Helper for drag and drop.

---

`^``cbRowDragPlugin::GetFirstRow`  
`^``cbrowdragpluggingetfirstrow`  
`^``rowse00325`  
`^``cbRowDragPlugin GetFirstRow`  
`^``nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbrowdragplugin')")`  
`^``GetFirstRow`

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

**`int GetHRowCountForPane(cbDockPane* pPane)`<sup>K</sup>**

Helper for drag and drop.

---

`^``cbRowDragPlugin::GetHRowCountForPane`  
`^``cbrowdragpluggingethrowscountforpane`  
`^``rowse00326`  
`^``cbRowDragPlugin GetHRowCountForPane`  
`^``nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")`  
`^``GetHRowCountForPane`

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

**`int GetRowDragHintWidth()`**<sup>K</sup>

Gets the row drag hint width.

---

<sup>C</sup>`bRowDragPlugin::GetRowDragHintWidth`  
<sup>C</sup>`browdragplugingetrowdraghintwidth`  
<sup>b</sup>`rowse00327`  
<sup>K</sup>`cbRowDragPlugin GetRowDragHintWidth`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbrowdragplugin')")`  
<sup>K</sup>`GetRowDragHintWidth`

**`cbRowDragPlugin::GetRowHintRect`**

**`void GetRowHintRect(cbRowInfo* pRow, wxRect& rect)`**<sup>K</sup>

Helper for drag and drop.

---

<sup>C</sup>`bRowDragPlugin::GetRowHintRect`  
<sup>C</sup>`browdragpluggingetrowhintrect`  
<sup>b</sup>`rowse00328`  
<sup>K</sup>`cbRowDragPlugin GetRowHintRect`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")`  
<sup>K</sup>`GetRowHintRect`

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

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

Test for the collapsed row icon position.

---

`CbRowDragPlugin::HitTestCollapsedRowIcon`  
`Cbrowdragpluginhittestcollapsedrowicon`  
`browse00329`  
`K cbRowDragPlugin HitTestCollapsedRowIcon`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")`  
`K HitTestCollapsedRowIcon`

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

**`bool HitTestRowDragHint(cbRowInfo* pRow, const wxPoint& pos)`<sup>K</sup>**

Test for the row drag hint position.

---

`C` `cbRowDragPlugin::HitTestRowDragHint`  
`C` `browdragpluginhittestrowdraghint`  
`b` `rowse00330`  
`K` `cbRowDragPlugin HitTestRowDragHint`  
`E` `nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")`  
`K` `HitTestRowDragHint`

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

**`void InsertDraggedRowBefore(cbRowInfo* pBeforeRow)`<sup>K</sup>**

Helper for drag and drop.

---

`^``cbRowDragPlugin::InsertDraggedRowBefore`  
`^``browdragplugininsertdraggedrowbefore`  
`b``rowse00331`  
`K``cbRowDragPlugin InsertDraggedRowBefore`  
`E``nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbrowdragplugin')")`  
`K``InsertDraggedRowBefore`

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

**bool ItemIsInFocus()**<sup>K</sup>

Helper for drag and drop.

---

<sup>C</sup>bRowDragPlugin::ItemIsInFocus  
<sup>C</sup>browdragpluginitemisinfocus  
<sup>b</sup>rowse00332  
<sup>K</sup> cbRowDragPlugin ItemIsInFocus  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")  
<sup>K</sup> ItemIsInFocus

**<sup>S#+K!</sup>cbRowDragPlugin::OnDrawPaneBackground**

**void OnDrawPaneBackground(cbDrawPaneDecorEvent& *event*)<sup>K</sup>**

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

---

<sup>C</sup>bRowDragPlugin::OnDrawPaneBackground  
<sup>C</sup>browdragpluginondrawpanebackground  
<sup>b</sup>rowse00333  
<sup>K</sup> cbRowDragPlugin OnDrawPaneBackground  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbrowdragplugin)")  
<sup>K</sup> OnDrawPaneBackground

**cbRowDragPlugin::OnInitPlugin**

**void OnInitPlugin()**

Called to initialize this plugin.

---

**cbRowDragPlugin::OnInitPlugin**  
**cbrowdragpluginoninitplugin**  
**rowse00334**  
**cbRowDragPlugin OnInitPlugin**  
**nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbrowdragplugin')")**  
**OnInitPlugin**

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

**`void OnLButtonDown(cbLeftDownEvent& event)K`**

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

---

`CbRowDragPlugin::OnLButtonDown  
Cbrowdragpluginonlbuttondown  
browse00335  
K cbRowDragPlugin OnLButtonDown  
EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")  
K OnLButtonDown`

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

**void OnLButtonUp(cbLeftUpEvent& *event*)**<sup>K</sup>

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

---

<sup>C</sup>bRowDragPlugin::OnLButtonUp  
<sup>C</sup>browdragpluginonlbuttonup  
<sup>b</sup>rowse00336  
<sup>K</sup> cbRowDragPlugin OnLButtonUp  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")  
<sup>K</sup> OnLButtonUp

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

**`void OnMouseMove(cbMotionEvent& event)`<sup>K</sup>**

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

---

`^``cbRowDragPlugin::OnMouseMove`  
`^``cbrowdragpluginonmousemove`  
`^``rowse00337`  
`^``cbRowDragPlugin OnMouseMove`  
`E``nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")`  
`^``OnMouseMove`

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

**`void PrepareForRowDrag()`<sup>K</sup>**

Helper for drag and drop.

---

`^`cbRowDragPlugin::PrepareForRowDrag  
`^`browdragpluginprepareforrowdrag  
`b`rowse00338  
`K` cbRowDragPlugin PrepareForRowDrag  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")  
`K` PrepareForRowDrag

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

**void SetMouseCapture**(`bool captureOn`)<sup>K</sup>

Helper for drag and drop.

---

<sup>C</sup>bRowDragPlugin::SetMouseCapture  
<sup>C</sup>browdragpluginsetmousecapture  
<sup>b</sup>rowse00339  
<sup>K</sup> cbRowDragPlugin SetMouseCapture  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")  
<sup>K</sup> SetMouseCapture

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

**`void SetPaneMargins()`**<sup>K</sup>

Sets the pane margins.

---

<sup>C</sup>`cbRowDragPlugin::SetPaneMargins`

<sup>C</sup>`browdragpluginsetpanemargins`

<sup>b</sup>`rowse00340`

<sup>K</sup>`cbRowDragPlugin SetPaneMargins`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")`

<sup>K</sup>`SetPaneMargins`

**`$$$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`

**`cbRowDragPlugin::ShowPanelImage`**

**`void ShowPanelImage()`**

Helper for drag and drop.

---

`cbRowDragPlugin::ShowPanelImage`  
`cbrowdragpluginshowpaneimage`  
`rowse00342`  
`cbRowDragPlugin ShowPanelImage`  
`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbrowdragplugin')")`  
`ShowPanelImage`

**`cbRowDragPlugin::UnhighlightItemInFocus`**

**`void UnhighlightItemInFocus()`**<sup>K</sup>

Helper for drag and drop.

---

`cbRowDragPlugin::UnhighlightItemInFocus`  
`browdragpluginunhighlightiteminfocus`  
`rowse00343`  
`cbRowDragPlugin UnhighlightItemInFocus`  
`enableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowdragplugin')")`  
`UnhighlightItemInFocus`

`$#+K!` **cbRowInfo::cbRowInfo**

**cbRowInfo()**<sup>K</sup>

Constructor.

---

`^` **cbRowInfo::cbRowInfo**

`^` **browinfocbrowinfo**

`b` rowse00345

`K` **cbRowInfo** **cbRowInfo**

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

`K` **cbRowInfo**

**`$#+K!cbRowInfo::~~cbRowInfo`**

**`~cbRowInfo()`**<sup>K</sup>

Destructor.

---

`C`<sub>1</sub>**`bRowInfo::~~cbRowInfo`**

`C`<sub>1</sub>**`browinfodtor`**

`b`<sub>1</sub>**`rowse00346`**

`K`<sub>1</sub>**`cbRowInfo ~cbRowInfo`**

`E`<sub>1</sub>**`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbrowinfo')`**

`K`<sub>1</sub>**`~cbRowInfo`**

**`$#+K!` `cbRowInfo::GetFirstBar`**

**`cbBarInfo* GetFirstBar()`<sup>K</sup>**

Returns the first bar.

---

<sup>C</sup>`bRowInfo::GetFirstBar`

<sup>C</sup>`browinfogetfirstbar`

<sup>b</sup>`rowse00347`

<sup>K</sup>`cbRowInfo GetFirstBar`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbrowinfo')`

<sup>K</sup>`GetFirstBar`

**`cbRowLayoutPlugin::cbRowLayoutPlugin`**

**`cbRowLayoutPlugin(wxFrameLayout* pPanel, int paneMask = wxALL_PANES)`**<sup>K</sup>

Constructor taking frame layout pane and pane mask.

**`cbRowLayoutPlugin()`**<sup>K</sup>

Default constructor.

---

<sup>C</sup>`cbRowLayoutPlugin::cbRowLayoutPlugin`

<sup>C</sup>`browlayoutpluginbrowlayoutplugin`

<sup>b</sup>`rowse00349`

<sup>K</sup>`cbRowLayoutPlugin cbRowLayoutPlugin`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowlayoutplugin')")`

<sup>K</sup>`cbRowLayoutPlugin`

<sup>K</sup>`cbRowLayoutPlugin`

**`cbRowLayoutPlugin::AdjustLengthOfInserted`**

**`void AdjustLengthOfInserted(cbRowInfo* pRow, cbBarInfo* pTheBar)`**<sup>K</sup>

Internal helper relating to not-fixed-bars layout.

---

`cbRowLayoutPlugin::AdjustLengthOfInserted`  
`cbrowlayoutpluginadjustlengthofinserted`  
`rowse00350`  
`cbRowLayoutPlugin AdjustLengthOfInserted`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowlayoutplugin')")`  
`AdjustLengthOfInserted`

**`$#+K!` `cbRowLayoutPlugin::ApplyLengthRatios`**

**`void ApplyLengthRatios(cbRowInfo* pRow)`<sup>K</sup>**

Internal helper relating to not-fixed-bars layout.

---

`C` `cbRowLayoutPlugin::ApplyLengthRatios`  
`C` `cbrowlayoutpluginapplylengthratios`  
`b` `rowse00351`  
`K` `cbRowLayoutPlugin ApplyLengthRatios`  
`E` `nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowlayoutplugin')")`  
`K` `ApplyLengthRatios`

`$#+K!` **cbRowLayoutPlugin::CalcRowHeight**

**int CalcRowHeight(cbRowInfo& row)<sup>K</sup>**

Row layout helper simulating bar 'friction'.

---

`CbRowLayoutPlugin::CalcRowHeight  
Cbrowlayoutpluginrowheight  
browse00352  
K cbRowLayoutPlugin CalcRowHeight  
EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowlayoutplugin')")  
K CalcRowHeight`

**`$#+K!` `cbRowLayoutPlugin::CheckIfAtTheBoundary`**

**`void CheckIfAtTheBoundary(cbBarInfo* pTheBar, cbRowInfo& rowInfo)`<sup>K</sup>**

Internal helper relating to not-fixed-bars layout.

---

<sup>C</sup>`bRowLayoutPlugin::CheckIfAtTheBoundary`

<sup>C</sup>`browlayoutplugincheckifattheboundary`

<sup>b</sup>`rowse00353`

<sup>K</sup>`cbRowLayoutPlugin CheckIfAtTheBoundary`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowlayoutplugin')")`

<sup>K</sup>`CheckIfAtTheBoundary`

**`$#+K!` `cbRowLayoutPlugin::DetectBarHandles`**

**`void DetectBarHandles(cbRowInfo* pRow)`<sup>K</sup>**

Internal helper relating to not-fixed-bars layout.

---

<sup>C</sup>`bRowLayoutPlugin::DetectBarHandles`

<sup>C</sup>`browlayoutplugindetectbarhandles`

<sup>b</sup>`rowse00354`

<sup>K</sup>`cbRowLayoutPlugin DetectBarHandles`

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

<sup>K</sup>`DetectBarHandles`

**`cbRowLayoutPlugin::DoInsertBar`**

**`void DoInsertBar(cbBarInfo* pTheBar, cbRowInfo& row)`**<sup>K</sup>

Insert the bar before the given row.

---

<sup>C</sup>`bRowLayoutPlugin::DoInsertBar`

<sup>C</sup>`browlayoutplugindoinsertbar`

<sup>b</sup>`rowse00355`

<sup>K</sup>`cbRowLayoutPlugin DoInsertBar`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowlayoutplugin')")`

<sup>K</sup>`DoInsertBar`

**`$#+K!` `cbRowLayoutPlugin::ExpandNotFixedBars`**

**`void ExpandNotFixedBars(cbRowInfo* pRow)K`**

Internal helper relating to not-fixed-bars layout.

---

`^cbRowLayoutPlugin::ExpandNotFixedBars`

`^cbrowlayoutpluginexpandnotfixedbars`

`^rowse00356`

`^cbRowLayoutPlugin ExpandNotFixedBars`

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

`^ExpandNotFixedBars`

**`$#+K!` cbRowLayoutPlugin::FitBarsToRange**

**`void FitBarsToRange(int from, int till, cbBarInfo* pTheBar, cbRowInfo* pRow)`**<sup>K</sup>

Internal helper relating to not-fixed-bars layout.

---

<sup>C</sup>bRowLayoutPlugin::FitBarsToRange

<sup>C</sup>browlayoutpluginfitbarstorage

<sup>b</sup>rowse00357

<sup>K</sup> cbRowLayoutPlugin FitBarsToRange

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowlayoutplugin')")

<sup>K</sup> FitBarsToRange

**`$#+K!` `cbRowLayoutPlugin::GetRowFreeSpace`**

**`int GetRowFreeSpace(cbRowInfo* pRow)`<sup>K</sup>**

Internal helper relating to not-fixed-bars layout.

---

<sup>C</sup>`bRowLayoutPlugin::GetRowFreeSpace`

<sup>C</sup>`browlayoutpluggingetrowfreespace`

<sup>b</sup>`rowse00358`

<sup>K</sup>`cbRowLayoutPlugin GetRowFreeSpace`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowlayoutplugin')")`

<sup>K</sup>`GetRowFreeSpace`

<sup>S#+K!</sup>**cbRowLayoutPlugin::InsertBefore**

**void InsertBefore(cbBarInfo\* pBeforeBar, cbBarInfo\* pTheBar, cbRowInfo& row)<sup>K</sup>**

Insert the bar before the given row.

---

<sup>C</sup>bRowLayoutPlugin::InsertBefore

<sup>C</sup>browlayoutplugininsertbefore

<sup>b</sup>rowse00359

<sup>K</sup> cbRowLayoutPlugin InsertBefore

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

<sup>K</sup> InsertBefore

**`cbRowLayoutPlugin::LayoutItemsVertically`**

**`void LayoutItemsVertically(cbRowInfo& row)`**<sup>K</sup>

Row layout helper simulating bar 'friction'.

---

<sup>C</sup>`cbRowLayoutPlugin::LayoutItemsVertically`  
<sup>C</sup>`cbrowlayoutpluginlayoutitemsvertically`  
<sup>b</sup>`rowse00360`  
<sup>K</sup>`cbRowLayoutPlugin LayoutItemsVertically`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowlayoutplugin')")`  
<sup>K</sup>`LayoutItemsVertically`

**`cbRowLayoutPlugin::MinimizeNotFixedBars`**

**`void MinimizeNotFixedBars(cbRowInfo* pRow, cbBarInfo* pBarToPreserve)`**<sup>K</sup>

Internal helper relating to not-fixed-bars layout.

---

<sup>C</sup>`bRowLayoutPlugin::MinimizeNotFixedBars`

<sup>C</sup>`browlayoutpluginminimizenotfixedbars`

<sup>b</sup>`rowse00361`

<sup>K</sup>`cbRowLayoutPlugin MinimizeNotFixedBars`

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

<sup>K</sup>`MinimizeNotFixedBars`

`##+K! cbRowLayoutPlugin::OnInsertBar`

`void OnInsertBar(cbInsertBarEvent& event)K`

Responds to bar insertion event.

---

`cbRowLayoutPlugin::OnInsertBar`

`cbrowlayoutpluginoninsertbar`

`rowse00362`

`K cbRowLayoutPlugin OnInsertBar`

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

`K OnInsertBar`

**`cbRowLayoutPlugin::OnLayoutRow`**

**`void OnLayoutRow(cbLayoutRowEvent& event)`**<sup>K</sup>

Responds to row layout event.

---

<sup>C</sup>`bRowLayoutPlugin::OnLayoutRow`  
<sup>C</sup>`browlayoutpluginonlayoutrow`  
<sup>b</sup>`rowse00363`  
<sup>K</sup> `cbRowLayoutPlugin OnLayoutRow`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbrowlayoutplugin')")`  
<sup>K</sup> `OnLayoutRow`

**`cbRowLayoutPlugin::OnLayoutRows`**

**`void OnLayoutRows(cbLayoutRowsEvent& event)`**<sup>K</sup>

Responds to rows layout event.

---

<sup>C</sup>`bRowLayoutPlugin::OnLayoutRows`  
<sup>C</sup>`browlayoutpluginonlayoutrows`  
<sup>b</sup>`rowse00364`  
<sup>K</sup>`cbRowLayoutPlugin OnLayoutRows`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowlayoutplugin')")`  
<sup>K</sup>`OnLayoutRows`

**`cbRowLayoutPlugin::OnRemoveBar`**

**`void OnRemoveBar(cbRemoveBarEvent& event)`**<sup>K</sup>

Responds to bar removal event.

---

<sup>C</sup>`bRowLayoutPlugin::OnRemoveBar`  
<sup>C</sup>`browlayoutpluginonremovebar`  
<sup>b</sup>`rowse00365`  
<sup>K</sup>`cbRowLayoutPlugin OnRemoveBar`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbrowlayoutplugin')")`  
<sup>K</sup>`OnRemoveBar`

**`cbRowLayoutPlugin::OnResizeRow`**

**`void OnResizeRow(cbResizeRowEvent& event)`**<sup>K</sup>

Responds to row resize event.

---

<sup>C</sup>`bRowLayoutPlugin::OnResizeRow`

<sup>C</sup>`browlayoutpluginonresizerow`

<sup>b</sup>`rowse00366`

<sup>K</sup>`cbRowLayoutPlugin OnResizeRow`

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

<sup>K</sup>`OnResizeRow`

**`cbRowLayoutPlugin::RecalcLengthRatios`**

**`void RecalcLengthRatios(cbRowInfo* pRow)`**<sup>K</sup>

Internal helper relating to not-fixed-bars layout.

---

<sup>C</sup>`cbRowLayoutPlugin::RecalcLengthRatios`

<sup>C</sup>`browlayoutpluginrecalclengthratios`

<sup>b</sup>`rowse00367`

<sup>K</sup>`cbRowLayoutPlugin RecalcLengthRatios`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowlayoutplugin')")`

<sup>K</sup>`RecalcLengthRatios`

**`$#+K!` `cbRowLayoutPlugin::RelayoutNotFixedBarsAround`**

**`void RelayoutNotFixedBarsAround(cbBarInfo* pTheBar, cbRowInfo* pRow)K`**

Internal helper relating to not-fixed-bars layout.

---

<sup>C</sup>`bRowLayoutPlugin::RelayoutNotFixedBarsAround`

<sup>C</sup>`browlayoutpluginrelayoutnotfixedbarsaround`

<sup>b</sup>`rowse00368`

<sup>K</sup>`cbRowLayoutPlugin RelayoutNotFixedBarsAround`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowlayoutplugin')")`

<sup>K</sup>`RelayoutNotFixedBarsAround`

**`cbRowLayoutPlugin::ShiftLeftTrashold`**

**`void ShiftLeftTrashold(cbBarInfo* pTheBar, cbRowInfo& row)`**<sup>K</sup>

Row layout helper simulating bar 'friction'.

---

<sup>C</sup>`bRowLayoutPlugin::ShiftLeftTrashold`

<sup>C</sup>`browlayoutpluginshiftlefttrashold`

<sup>b</sup>`rowse00369`

<sup>K</sup>`cbRowLayoutPlugin ShiftLeftTrashold`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowlayoutplugin')")`

<sup>K</sup>`ShiftLeftTrashold`

**`cbRowLayoutPlugin::ShiftRightTrashold`**

**`void ShiftRightTrashold(cbBarInfo* pTheBar, cbRowInfo& row)`**<sup>K</sup>

Row layout helper simulating bar 'friction'.

---

`cbRowLayoutPlugin::ShiftRightTrashold`

`cbrowlayoutpluginshiftrighttrashold`

`rowse00370`

`cbRowLayoutPlugin ShiftRightTrashold`

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

`ShiftRightTrashold`

**`cbRowLayoutPlugin::SlideLeftSideBars`**

**`void SlideLeftSideBars(cbBarInfo* pTheBar)`**<sup>K</sup>

Row layout helper simulating bar 'friction'.

---

<sup>C</sup>`bRowLayoutPlugin::SlideLeftSideBars`

<sup>C</sup>`browlayoutpluginslideleftsidebars`

<sup>b</sup>`rowse00371`

<sup>K</sup>`cbRowLayoutPlugin SlideLeftSideBars`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbrowlayoutplugin')")`

<sup>K</sup>`SlideLeftSideBars`

**`cbRowLayoutPlugin::SlideRightSideBars`**

**`void SlideRightSideBars(cbBarInfo* pTheBar)`**<sup>K</sup>

Row layout helper simulating bar 'friction'.

---

<sup>C</sup>`cbRowLayoutPlugin::SlideRightSideBars`

<sup>C</sup>`browlayoutpluginsliderightsidebars`

<sup>b</sup>`rowse00372`

<sup>K</sup>`cbRowLayoutPlugin SlideRightSideBars`

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

<sup>K</sup>`SlideRightSideBars`

**`cbRowLayoutPlugin::StickRightSideBars`**

**`void StickRightSideBars(cbBarInfo* pToBar)`**<sup>K</sup>

Row layout helper simulating bar 'friction'.

---

<sup>C</sup>`bRowLayoutPlugin::StickRightSideBars`

<sup>C</sup>`browlayoutpluginstickrightsidebars`

<sup>b</sup>`rowse00373`

<sup>K</sup>`cbRowLayoutPlugin StickRightSideBars`

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

<sup>K</sup>`StickRightSideBars`

`$$$K!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`  
`^bsimplecustomizationplugin^cbSimpleCustomizationPlugin`  
`^rowse00375`  
`^cbSimpleCustomizationPlugin cbSimpleCustomizationPlugin`  
`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbSimpleCustomizationPlugin')")`  
`^cbSimpleCustomizationPlugin`  
`^cbSimpleCustomizationPlugin`

**`cbSimpleCustomizationPlugin::OnCustomizeBar`**

**`void OnCustomizeBar(cbCustomizeBarEvent& event)`**<sup>K</sup>

Plugin event handler for cbCustomizeBarEvent.

---

<sup>C</sup>`bSimpleCustomizationPlugin::OnCustomizeBar`

<sup>C</sup>`bsimplecustomizationpluginoncustomizebar`

<sup>b</sup>`rowse00376`

<sup>K</sup>`cbSimpleCustomizationPlugin OnCustomizeBar`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbsimplecustomizationplugin)")`

<sup>K</sup>`OnCustomizeBar`

**`cbSimpleCustomizationPlugin::OnCustomizeLayout`**

**`void OnCustomizeLayout(cbCustomizeLayoutEvent& event)`**<sup>K</sup>

Plugin event handler for cbCustomizeLayoutEvent.

---

<sup>C</sup>`bSimpleCustomizationPlugin::OnCustomizeLayout`  
<sup>C</sup>`bsimplecustomizationpluginoncustomizelayout`  
<sup>b</sup>`rowse00377`  
<sup>K</sup> `cbSimpleCustomizationPlugin OnCustomizeLayout`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbsimplecustomizationplugin')`)  
<sup>K</sup> `OnCustomizeLayout`

**`cbSimpleCustomizationPlugin::OnMenuItemSelected`**

**`void OnMenuItemSelected(wxCommandEvent& event)`**<sup>K</sup>

Menu event handler.

---

<sup>C</sup>`bSimpleCustomizationPlugin::OnMenuItemSelected`

<sup>C</sup>`bsimplecustomizationpluginonmenuitemselected`

<sup>b</sup>`rowse00378`

<sup>K</sup>`cbSimpleCustomizationPlugin OnMenuItemSelected`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbsimplecustomizationplugin')")`

<sup>K</sup>`OnMenuItemSelected`

`$#+K!` **cbSimpleUpdatesMgr::cbSimpleUpdatesMgr**

**cbSimpleUpdatesMgr()**<sup>K</sup>

Default constructor.

**cbSimpleUpdatesMgr(wxFrameLayout\* *pPanel*)**<sup>K</sup>

Constructor taking frame layout panel.

---

```
^cbSimpleUpdatesMgr::cbSimpleUpdatesMgr  
^bsimpleupdatesmgrcbsimpleupdatesmgr  
^rowse00380  
K cbSimpleUpdatesMgr cbSimpleUpdatesMgr  
E nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbsimpleupdatesmgr)")  
K cbSimpleUpdatesMgr  
K cbSimpleUpdatesMgr
```

**`cbSimpleUpdatesMgr::OnBarWillChange`**

**`void OnBarWillChange(cbBarInfo* pBar, cbRowInfo* pInRow, cbDockPane* pInPane)`**<sup>K</sup>

Notification received from Frame Layout in the order in which they would usually be invoked.

---

<sup>C</sup>bSimpleUpdatesMgr::OnBarWillChange  
<sup>C</sup>bsimpleupdatesmgronbarwillchange  
<sup>b</sup>rowse00381  
<sup>K</sup> cbSimpleUpdatesMgr OnBarWillChange  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbsimpleupdatesmgr)")  
<sup>K</sup> OnBarWillChange

**`$#+K! cbSimpleUpdatesMgr::OnFinishChanges`**

**`void OnFinishChanges()`**<sup>K</sup>

Notification received from Frame Layout in the order in which they would usually be invoked.

---

`^cbSimpleUpdatesMgr::OnFinishChanges`  
`^bsimpleupdatesmgronfinishchanges`  
`^rowse00382`  
`^cbSimpleUpdatesMgr OnFinishChanges`  
`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbsimpleupdatesmgr)")`  
`^ OnFinishChanges`

**`cbSimpleUpdatesMgr::OnPaneMarginsWillChange`**

**`void OnPaneMarginsWillChange(cbDockPane* pPane)`**<sup>K</sup>

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`

**`cbSimpleUpdatesMgr::OnPaneWillChange`**

**`void OnPaneWillChange(cbDockPane* pPane)`**<sup>K</sup>

Notification received from Frame Layout in the order in which they would usually be invoked.

---

<sup>C</sup>`bSimpleUpdatesMgr::OnPaneWillChange`  
<sup>C</sup>`bsimpleupdatesmgronpanewillchange`  
<sup>b</sup>`rowse00384`  
<sup>K</sup>`cbSimpleUpdatesMgr OnPaneWillChange`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbsimpleupdatesmgr')")`  
<sup>K</sup>`OnPaneWillChange`

**`##+K! 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  
browse00385  
K cbSimpleUpdatesMgr OnRowWillChange  
EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbsimpleupdatesmgr')")  
K OnRowWillChange`

**`cbSimpleUpdatesMgr::OnStartChanges`**

**`void OnStartChanges()`**<sup>K</sup>

Notification received from Frame Layout in the order in which they would usually be invoked.

---

`cbSimpleUpdatesMgr::OnStartChanges`  
`cbSimpleUpdatesMgrOnStartChanges`  
`rowse00386`  
`cbSimpleUpdatesMgr OnStartChanges`  
`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbSimpleUpdatesMgr')")`  
`OnStartChanges`

`$#+K!` **cbSimpleUpdatesMgr::UpdateNow**

**void UpdateNow()**<sup>K</sup>

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

---

`^`cbSimpleUpdatesMgr::UpdateNow

`^`bsimpleupdatesmgrupdatenow

`^`rowse00387

`^`cbSimpleUpdatesMgr UpdateNow

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

`^` UpdateNow

`$#+K!` **cbSimpleUpdatesMgr::WasChanged**

**bool WasChanged(cbUpdateMgrData& data, wxRect& currentBounds)<sup>K</sup>**

Helper function.

---

`^`cbSimpleUpdatesMgr::WasChanged  
`^`bsimpleupdatesmgrwaschanged  
`^`rowse00388  
`^`cbSimpleUpdatesMgr WasChanged  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbsimpleupdatesmgr)")  
`^`WasChanged

**`$#+K!` `cbSizeBarWndEvent::cbSizeBarWndEvent`**

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

Constructor, taking bar information and pane.

---

`CcbSizeBarWndEvent::cbSizeBarWndEvent`

`Cbsizebarwndeventcbsizebarwndevent`

`browse00390`

`KcbSizeBarWndEvent cbSizeBarWndEvent`

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

`KcbSizeBarWndEvent`

**`$#+K!` `cbStartBarDraggingEvent::cbStartBarDraggingEvent`**

**`cbStartBarDraggingEvent(cbBarInfo* pBar, const wxPoint& pos, cbDockPane* pPane)K`**

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

---

`CcbStartBarDraggingEvent::cbStartBarDraggingEvent`  
`Cbstartbardraggingeventcbstartbardraggingevent`  
`browse00392`  
`KcbStartBarDraggingEvent cbStartBarDraggingEvent`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbstartbardraggingevent')")`  
`KcbStartBarDraggingEvent`

**cbStartDrawInAreaEvent::cbStartDrawInAreaEvent**

**cbStartDrawInAreaEvent(const wxRect& area, wxDC\*\* ppDCForArea, cbDockPane\* pPane)<sup>K</sup>**

to the obtained buffer-context should be placed Constructor, taking rectangular area, device context pointer to a pointer, and pane.

---

<sup>C</sup>bStartDrawInAreaEvent::cbStartDrawInAreaEvent  
<sup>C</sup>bstartdrawinareaeventcbstartdrawinareaevent  
<sup>b</sup>rowse00394  
<sup>K</sup> cbStartDrawInAreaEvent cbStartDrawInAreaEvent  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbstartdrawinareaevent')")  
<sup>K</sup> cbStartDrawInAreaEvent

`$#+K! cbUpdateMgrData::cbUpdateMgrData`

`cbUpdateMgrData()`<sup>K</sup>

Default constructor. Is-dirty flag is set TRUE initially.

---

`^cbUpdateMgrData::cbUpdateMgrData`  
`^bupdatemgrdatacbupdatemgrdata`  
`^rowse00396`  
`^cbUpdateMgrData cbUpdateMgrData`  
`^nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbupdatemgrdata')")`  
`^cbUpdateMgrData`

**`$#+K! cbUpdateMgrData::IsDirty`**

**`bool IsDirty()`**<sup>K</sup>

Returns the is-dirty flag.

---

`^cbUpdateMgrData::IsDirty`  
`^bupdatemgrdataisdirty`  
`^rowse00397`  
`^K cbUpdateMgrData IsDirty`  
`^EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbupdatemgrdata)")`  
`^K IsDirty`

**`$#+K!` `cbUpdateMgrData::SetCustomData`**

**`void SetCustomData(wxObject* pCustomData)`<sup>K</sup>**

Set custom data.

---

<sup>C</sup>`bUpdateMgrData::SetCustomData`

<sup>C</sup>`bupdatemgrdatasetcustomdata`

<sup>b</sup>`rowse00398`

<sup>K</sup>`cbUpdateMgrData SetCustomData`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbupdatemgrdata')")`

<sup>K</sup>`SetCustomData`

`$#+K!` **cbUpdateMgrData::SetDirty**

**void SetDirty**(bool *isDirty* = *TRUE*)<sup>K</sup>

Set the dirty flag.

---

<sup>C</sup>bUpdateMgrData::SetDirty  
<sup>C</sup>bupdatemgrdatasetdirty  
<sup>b</sup>rowse00399  
<sup>K</sup> cbUpdateMgrData SetDirty  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbupdatemgrdata')")  
<sup>K</sup> SetDirty

`$#+K!` **cbUpdateMgrData::StoreItemState**

**void StoreItemState(const wxRect& *boundsInParent*)<sup>K</sup>**

Store the item state.

---

`CbUpdateMgrData::StoreItemState  
Cbupdatemgrdatastoreitemstate  
browse00400  
K cbUpdateMgrData StoreItemState  
EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbupdatemgrdata')")  
K StoreItemState`

**`$#+K! cbUpdatesManagerBase::cbUpdatesManagerBase`**

**`cbUpdatesManagerBase(wxFrameLayout* pPanel)K`**

Constructor taking layout panel.

**`cbUpdatesManagerBase()K`**

Default constructor

---

`CbUpdatesManagerBase::cbUpdatesManagerBase  
Cbupdatesmanagerbasecbupdatesmanagerbase  
browse00402  
K cbUpdatesManagerBase cbUpdatesManagerBase  
EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbupdatesmanagerbase')")  
K cbUpdatesManagerBase  
K cbUpdatesManagerBase`

**`$#+K!cbUpdatesManagerBase::~~cbUpdatesManagerBase`**

**`~cbUpdatesManagerBase()`<sup>K</sup>**

Destructor.

---

`c`<sub>1</sub>`bUpdatesManagerBase::~~cbUpdatesManagerBase`

`c`<sub>1</sub>`bupdatesmanagerbasedtor`

`b`<sub>1</sub>`rowse00403`

`K`<sub>1</sub>`cbUpdatesManagerBase ~cbUpdatesManagerBase`

`E`<sub>1</sub>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbupdatesmanagerbase')")`

`K`<sub>1</sub>`~cbUpdatesManagerBase`

## `$$$K!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`  
`browse00404`  
`K cbUpdatesManagerBase OnBarWillChange`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbupdatesmanagerbase')")`  
`K OnBarWillChange`

## `$#+K!` **cbUpdatesManagerBase::OnFinishChanges**

**void OnFinishChanges()**<sup>K</sup>

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::OnFinishChanges  
`^`bupdatesmanagerbaseonfinishchanges  
`^`rowse00405  
`^` cbUpdatesManagerBase OnFinishChanges  
`^`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbupdatesmanagerbase')")  
`^` OnFinishChanges

**`cbUpdatesManagerBase::OnPaneMarginsWillChange`**

**`void OnPaneMarginsWillChange(cbDockPane* pPane)`**<sup>K</sup>

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.

---

<sup>C</sup>bUpdatesManagerBase::OnPaneMarginsWillChange  
<sup>C</sup>bupdatesmanagerbaseonpanemarginswillchange  
<sup>b</sup>rowse00406  
<sup>K</sup> cbUpdatesManagerBase OnPaneMarginsWillChange  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbupdatesmanagerbase')")  
<sup>K</sup> OnPaneMarginsWillChange

## `cbUpdatesManagerBase::OnPaneWillChange`

`void OnPaneWillChange(cbDockPane* pPane)`

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::OnPaneWillChange`  
`cbupdatesmanagerbaseonpanewillchange`  
`rowse00407`  
`cbUpdatesManagerBase OnPaneWillChange`  
`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbupdatesmanagerbase')")`  
`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`

## `cbUpdatesManagerBase::OnStartChanges`

`void OnStartChanges()`

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::OnStartChanges`  
`cbupdatesmanagerbaseonstartchanges`  
`rowse00409`  
`cbUpdatesManagerBase OnStartChanges`  
`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbupdatesmanagerbase')")`  
`OnStartChanges`

**`cbUpdatesManagerBase::SetLayout`**

**`void SetLayout(wxFrameLayout* pLayout)`**<sup>K</sup>

Sets the associated layout.

---

<sup>C</sup>`bUpdatesManagerBase::SetLayout`

<sup>C</sup>`bupdatesmanagerbasesetlayout`

<sup>b</sup>`rowse00410`

<sup>K</sup>`cbUpdatesManagerBase SetLayout`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `cbupdatesmanagerbase')")`

<sup>K</sup>`SetLayout`

**cbUpdatesManagerBase::UpdateNow**

**void UpdateNow()**

Refreshes parts of the frame layout that need an update.

---

`cbUpdatesManagerBase::UpdateNow`  
`bupdatesmanagerbaseupdatenow`  
`rowse00411`  
`cbUpdatesManagerBase UpdateNow`  
`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `cbupdatesmanagerbase')")`  
`UpdateNow`

`$#+K!` **wxDynamicToolBar::wxDynamicToolBar**

**wxDynamicToolBar()**<sup>K</sup>

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)**<sup>K</sup>

Constructor: see the documentation for wxToolBar for details.

---

`wxDynamicToolBar::wxDynamicToolBar`  
`wxdynamictoolbarwxdynamictoolbar`  
`rowse00413`  
`K wxDynamicToolBar wxDynamicToolBar`  
`E nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxdynamictoolbar')")`  
`K wxDynamicToolBar`  
`K wxDynamicToolBar`

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

`~wxDynamicToolBar()`<sup>K</sup>

Destructor.

---

`w_xDynamicToolBar::~wxDynamicToolBar`

`w_xdynamictoolbarctor`

`b_rowse00414`

`K wxDynamicToolBar ~wxDynamicToolBar`

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

`K ~wxDynamicToolBar`

`wxDynamicToolBar::AddSeparator`

`void AddSeparator(wxWindow* pSeparatorWnd = NULL)`

Adds a separator. See the documentation for `wxToolBar` for details.

---

`wxDynamicToolBar::AddSeparator`

`wxdynamictoolbaraddseparator`

`rowse00415`

`wxDynamicToolBar AddSeparator`

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

`AddSeparator`

## `wxDynamicToolBar::AddTool`

**void AddTool**(int *toolIndex*, **wxWindow\*** *pToolWindow*, **const wxSize&** *size* = *wxDefaultSize*)<sup>K</sup>

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*)<sup>K</sup>

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*)<sup>K</sup>

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* = "")<sup>K</sup>

Method from `wxToolBarBase` (for compatibility), only the first two arguments are valid. See the documentation for `wxToolBar` for details.

---

<sup>w</sup>`wxDynamicToolBar::AddTool`

<sup>w</sup>`wxdynamictoolbaraddtool`

<sup>b</sup>`rowse00416`

<sup>K</sup>`wxDynamicToolBar AddTool`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxdynamictoolbar')")`

<sup>K</sup>`AddTool`

<sup>K</sup>`AddTool`

<sup>K</sup>`AddTool`

<sup>K</sup>`AddTool`

**`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)`**<sup>K</sup>

Creation function: see the documentation for `wxToolBar` for details.

---

<sup>w</sup>`wxDynamicToolBar::Create`  
<sup>w</sup>`wxdynamictoolbarcreate`  
<sup>b</sup>`rowse00417`  
<sup>K</sup> `wxDynamicToolBar Create`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxdynamictoolbar')")`  
<sup>K</sup> `Create`

**`$#+K!` wxDynamicToolBar::CreateDefaultLayout**

**LayoutManagerBase\* CreateDefaultLayout()**<sup>K</sup>

Creates the default layout (BagLayout).

---

`w`xDynamicToolBar::CreateDefaultLayout  
`w`xdynamictoolbarcreatedefaultlayout  
`b`rowse00418  
`K` wxDynamicToolBar CreateDefaultLayout  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxdynamictoolbar)")  
`K` CreateDefaultLayout

**`$#+K!` wxDynamicToolBar::CreateTool**

**`wxToolBarToolBase*` CreateTool(`wxControl*` control)<sup>K</sup>**

Creates a toolbar tool.

**`wxToolBarToolBase*` CreateTool(`int id`, `const wxBitmap& bitmap1`, `const wxBitmap& bitmap2`, `bool toggle`, `wxObject*` clientData, `const wxString& shortHelpString`, `const wxString& longHelpString`)<sup>K</sup>**

Creates a toolbar tool.

---

`wxDynamicToolBar::CreateTool`

`wxdynamictoolbarcreatetool`

`rowse00419`

`wxDynamicToolBar CreateTool`

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

`CreateTool`

`CreateTool`

`wxDynamicToolBar::DoDeleteTool`

`bool DoDeleteTool(size_t pos, wxToolBarToolBase* tool)`

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`  
`rowse00420`  
`wxDynamicToolBar DoDeleteTool`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxdynamictoolbar")`  
`DoDeleteTool`

**`wxDynamicToolBar::DoEnableTool`**

**`void DoEnableTool(wxToolBarToolBase* tool, bool enable)`**<sup>K</sup>

Called when the tools enabled flag changes.

---

<sup>w</sup>`DynamicToolBar::DoEnableTool`  
<sup>w</sup>`dynamictoolbarenabletool`  
<sup>b</sup>`rowse00421`  
<sup>K</sup> `wxDynamicToolBar DoEnableTool`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxdynamictoolbar")`  
<sup>K</sup> `DoEnableTool`

**`wxDynamicToolBar::DoInsertTool`**

**`bool DoInsertTool(size_t pos, wxToolBarToolBase* tool)`**<sup>K</sup>

Inserts a tool at the given position.

---

<sup>w</sup>`wxDynamicToolBar::DoInsertTool`  
<sup>w</sup>`wxdynamictoolbarinserttool`  
<sup>b</sup>`rowse00422`  
<sup>K</sup> `wxDynamicToolBar DoInsertTool`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxdynamictoolbar")`  
<sup>K</sup> `DoInsertTool`

**`wxDynamicToolBar::DoSetToggle`**

**`void DoSetToggle(wxToolBarToolBase* tool, bool toggle)`**<sup>K</sup>

Called when the tools 'can be toggled' flag changes.

---

<sup>w</sup>xDynamicToolBar::DoSetToggle  
<sup>w</sup>xdynamictoolbarosettoggle  
<sup>b</sup>rowse00423  
<sup>K</sup> wxDynamicToolBar DoSetToggle  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxdynamictoolbar)")  
<sup>K</sup> DoSetToggle

`wxDynamicToolBar::DoToggleTool`

`void DoToggleTool(wxToolBarToolBase* tool, bool toggle)`

Called when the tool is toggled.

---

`wxDynamicToolBar::DoToggleTool`  
`wxdynamictoolbardotoggletool`  
`rowse00424`  
`wxDynamicToolBar DoToggleTool`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxdynamictoolbar")")`  
`DoToggleTool`

**`wxDynamicToolBar::DrawSeparator`**

**`void DrawSeparator(wxDynToolInfo& info, wxDC& dc)`**<sup>K</sup>

Draws a separator. The default implementation draws a shaded line.

---

<sup>w</sup>`wxDynamicToolBar::DrawSeparator`

<sup>w</sup>`wxdynamictoolbardrawseparator`

<sup>b</sup>`rowse00425`

<sup>K</sup>`wxDynamicToolBar DrawSeparator`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxdynamictoolbar')")`

<sup>K</sup>`DrawSeparator`

`wxDynamicToolBar::EnableTool`

`void EnableTool(const int toolIndex, const bool enable = TRUE)K`

Enables or disables the given tool.

---

`wxDynamicToolBar::EnableTool`

`wxdynamictoolbarenabletool`

`rowse00426`

`K wxDynamicToolBar EnableTool`

`E nableButton("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.

---

**w**xDynamicToolBar::FindToolForPosition

**w**xdynamictoolbarfindtoolforposition

**b**rowse00427

**K** wxDynamicToolBar FindToolForPosition

**K** FindToolForPosition

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

`$#+K!` **wxDynamicToolBar::GetPreferredDim**

**void GetPreferredDim(const wxSize& *givenDim*, wxSize& *prefDim*)<sup>K</sup>**

Returns the preferred dimension, taking the given dimension and a reference to the result.

---

<sup>w</sup>xDynamicToolBar::GetPreferredDim  
<sup>w</sup>xdynamicoolbargetpreferredim  
<sup>b</sup>rowse00428  
<sup>K</sup> wxDynamicToolBar GetPreferredDim  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxdynamictoolbar)")  
<sup>K</sup> GetPreferredDim

**`$#+K!` wxDynamicToolBar::GetToolInfo**

**`wxDynToolInfo*` GetToolInfo(`int toolIndex`)<sup>K</sup>**

Returns tool information for the given tool index.

---

<sup>w</sup>xDynamicToolBar::GetToolInfo

<sup>w</sup>xdynamictoolbar::gettoolinfo

<sup>b</sup>rowse00429

<sup>K</sup> wxDynamicToolBar GetToolInfo

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxdynamictoolbar)")

<sup>K</sup> GetToolInfo

`wxDynamicToolBar::Layout`

`bool Layout()`

Performs layout. See definitions of orientation types.

---

`wxDynamicToolBar::Layout`

`wxdynamictoolbarlayout`

`rowse00430`

`wxDynamicToolBar Layout`

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

`Layout`

`##+K! wxDynamicToolBar::OnEraseBackground`

`void OnEraseBackground(wxEraseEvent& event)K`

Responds to background erase events. Currently does nothing.

---

`wxDynamicToolBar::OnEraseBackground`

`wxdynamictoolbaronerasebackground`

`browse00431`

`K wxDynamicToolBar OnEraseBackground`

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

`K OnEraseBackground`

**`wxDynamicToolBar::OnPaint`**

**`void OnPaint(wxPaintEvent& event)`**<sup>K</sup>

Responds to paint events, drawing separators.

---

<sup>w</sup>`wxDynamicToolBar::OnPaint`

<sup>w</sup>`wxdynamictoolbaronpaint`

<sup>b</sup>`rowse00432`

<sup>K</sup>`wxDynamicToolBar OnPaint`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxdynamictoolbar")`

<sup>K</sup>`OnPaint`

`$#+K!` **wxDynamicToolBar::OnSize**

**void OnSize(wxSizeEvent& event)**<sup>K</sup>

Responds to size events, calling Layout.

---

<sup>w</sup>xDynamicToolBar::OnSize  
<sup>w</sup>xdynamictoolbaronsize  
<sup>b</sup>rowse00433  
<sup>K</sup> wxDynamicToolBar OnSize  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxdynamictoolbar)")  
<sup>K</sup> OnSize

`$#+K! wxDynamicToolBar::Realize`

`bool Realize()K`

Overriden from wxToolBarBase; does nothing.

---

`wxDynamicToolBar::Realize`

`wxdynamictoolbarrealize`

`browse00434`

`K wxDynamicToolBar Realize`

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

`K Realize`

`$#+K!` **wxDynamicToolBar::RemveTool**

**void RemveTool(int *toolIndex*)<sup>K</sup>**

Removes the given tool. Misspelt in order not to clash with a similar function in the base class.

---

`w`xDynamicToolBar::RemveTool  
`w`xdynamictoolbarremvetool  
`b`rowse00435  
`K` wxDynamicToolBar RemveTool  
`E`nableButton("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`

`rowse00436`

`K wxDynamicToolBar SetLayout`

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

`K SetLayout`

`$#+K!` **wxDynamicToolBar::SizeToolWindows**

**void SizeToolWindows()**<sup>K</sup>

Internal function for sizing tool windows.

---

<sup>w</sup>wxDynamicToolBar::SizeToolWindows

<sup>w</sup>wxdynamictoolbarstoolwindows

<sup>b</sup>rowse00437

<sup>K</sup> wxDynamicToolBar SizeToolWindows

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxdynamictoolbar)")

<sup>K</sup> SizeToolWindows

**`wxFrameLayout::wxFrameLayout`**

**`wxFrameLayout(wxWindow* pParentFrame, wxWindow* pFrameClient = NULL, bool activateNow = TRUE)`**<sup>K</sup>

Constructor, taking parent window, the (MDI) client of the parent if there is one, and flag specifying whether to activate the layout.

**`wxFrameLayout()`**<sup>K</sup>

Default constructor, used only for serialization.

---

<sup>w</sup>`wxFrameLayout::wxFrameLayout`  
<sup>w</sup>`wxframelayouwxframelayout`  
<sup>b</sup>`rowse00440`  
<sup>K</sup> `wxFrameLayout wxFrameLayout`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")`  
<sup>K</sup> `wxFrameLayout`  
<sup>K</sup> `wxFrameLayout`

`$#+K1 wxFrameLayout::~wxFrameLayout`

`~wxFrameLayout()`<sup>K</sup>

Destructor. It does not destroy the bar windows.

---

`wxFrameLayout::~wxFrameLayout`

`wxframelayouthtor`

`rowse00441`

`K wxFrameLayout ~wxFrameLayout`

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

`K ~wxFrameLayout`

`$#+K!` **wxFrameLayout::Activate**

**void Activate()**<sup>K</sup>

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.

---

`w`xFrameLayout::Activate  
`w`xframelayouctivate  
`b`rowse00442  
`K` wxFrameLayout Activate  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")  
`K` Activate

## `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`.

---

<sup>w</sup>`wxFrameLayout::AddBar`

<sup>w</sup>`wxframelayouaddbar`

<sup>b</sup>`rowse00443`

<sup>K</sup>`wxFrameLayout AddBar`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxframelayout')`

<sup>K</sup>`AddBar`

`$$$K1 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`  
`wxframelayouaddplugin`  
`rowse00444`  
`K wxFrameLayout AddPlugin`  
`E nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")`  
`K AddPlugin`

## `wxFrameLayout::AddPluginBefore`

**void AddPluginBefore(wxClassInfo\* pNextPInfo, wxClassInfo\* pPInfo, int paneMask = wxALL\_PANES)**<sup>K</sup>

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.

---

<sup>w</sup> wxFrameLayout::AddPluginBefore  
<sup>w</sup> wxframelayouaddpluginbefore  
<sup>b</sup> rowse00445  
<sup>K</sup> wxFrameLayout AddPluginBefore  
<sup>E</sup> nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")  
<sup>K</sup> AddPluginBefore

`$#+K!` **wxFrameLayout::ApplyBarProperties**

**void ApplyBarProperties(cbBarInfo\* pBar)**<sup>K</sup>

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.

---

`w`xFrameLayout::ApplyBarProperties  
`w`xframelayoutapplybarproperties  
`b`rowse00446  
`K` wxFrameLayout ApplyBarProperties  
`E`nableButton("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`  
`wxframeLayoutcanreparent`  
`rowse00447`  
`K wxFrameLayout CanReparent`  
`E nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxframeLayout')")`  
`K CanReparent`

`$#+K!` **wxFrameLayout::CaptureEventsForPane**

**void CaptureEventsForPane(cbDockPane\* toPane)<sup>K</sup>**

Called by plugins; also captures the mouse in the parent frame.

---

`w`xFrameLayout::CaptureEventsForPane  
`w`xframelayoucaptureeventsforpane  
`b`rowse00448  
`K` wxFrameLayout CaptureEventsForPane  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxframelayout')")  
`K` CaptureEventsForPane

**`wxFrameLayout::CaptureEventsForPlugin`**

**`void CaptureEventsForPlugin(cbPluginBase* pPlugin)`**<sup>K</sup>

Captures user input events for the given plugin. Input events are: mouse movement, mouse clicks, keyboard input.

---

<sup>w</sup>`wxFrameLayout::CaptureEventsForPlugin`  
<sup>w</sup>`wxframelayoucaptureeventsforplugin`  
<sup>b</sup>`rowse00449`  
<sup>K</sup> `wxFrameLayout CaptureEventsForPlugin`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxframelayout')")`  
<sup>K</sup> `CaptureEventsForPlugin`

`$#+K! wxFrameLayout::CreateCursors`

`void CreateCursors()K`

Creates the cursors.

---

`w_xFrameLayout::CreateCursors`  
`w_xframelayoucreatecursors`  
`b_rowse00450`  
`K wxFrameLayout CreateCursors`  
`E_nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")`  
`K CreateCursors`

**`$#+K!` wxFrameLayout::CreateUpdatesManager**

**`cbUpdatesManagerBase* CreateUpdatesManager()`<sup>K</sup>**

Returns a new cbGCUpdatesMgr object.

---

<sup>w</sup>xFrameLayout::CreateUpdatesManager

<sup>w</sup>xframelayoucreateupdatesmanager

<sup>b</sup>rowse00451

<sup>K</sup> wxFrameLayout CreateUpdatesManager

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")

<sup>K</sup> CreateUpdatesManager

`$#+K! wxFrameLayout::Deactivate`

`void Deactivate()K`

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.

---

`w_xFrameLayout::Deactivate`  
`w_xframelayoutdeactivate`  
`b_rowse00452`  
`K wxFrameLayout Deactivate`  
`E_nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")`  
`K Deactivate`

**`wxFrameLayout::DestroyBarWindows`**

**`void DestroyBarWindows()`**<sup>K</sup>

Destroys the bar windows.

---

<sup>w</sup>`wxFrameLayout::DestroyBarWindows`  
<sup>w</sup>`wxframelayoutdestroybarwindows`  
<sup>b</sup>`rowse00453`  
<sup>K</sup> `wxFrameLayout DestroyBarWindows`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")`  
<sup>K</sup> `DestroyBarWindows`

`$#+K!` **wxFrameLayout::DoSetBarState**

**void DoSetBarState(cbBarInfo\* pBar)<sup>K</sup>**

Applies the state to the window objects.

---

`w` wxFrameLayout::DoSetBarState  
`w` wxframelayouthdosetbarstate  
`b` rowse00454  
`K` wxFrameLayout DoSetBarState  
`E` nableButton("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`  
`E nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")`  
`K EnableFloating`

<sup>S#+K!</sup>**wxFramelayout::FindBarByName**

**cbBarInfo\*** FindBarByName(const wxString& *name*)<sup>K</sup>

Finds the bar in the framelayout, by name.

---

<sup>w</sup>xFrameLayout::FindBarByName

<sup>w</sup>xframelayoutfindbarbyname

<sup>b</sup>rowse00456

<sup>K</sup> wxFrameLayout FindBarByName

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")

<sup>K</sup> FindBarByName

**`$#+K!` wxFrameLayout::FindBarByWindow**

**`cbBarInfo* FindBarByWindow(const wxWindow* pWnd)`<sup>K</sup>**

Finds the bar in the framelayout, by window.

---

<sup>w</sup>xFrameLayout::FindBarByWindow

<sup>w</sup>xframelayoutfindbarbywindow

<sup>b</sup>rowse00457

<sup>K</sup> wxFrameLayout FindBarByWindow

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")

<sup>K</sup> FindBarByWindow

**`$#+K!` wxFrameLayout::FindPlugin**

**`cbPluginBase*` FindPlugin(`wxClassInfo*` *pPInfo*)<sup>K</sup>**

Finds a plugin with the given class, or returns NULL if a plugin of the given class is not hooked.

---

<sup>w</sup>xFrameLayout::FindPlugin

<sup>w</sup>xframeLayoutfindplugin

<sup>b</sup>rowse00458

<sup>K</sup> wxFrameLayout FindPlugin

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframeLayout')")

<sup>K</sup> FindPlugin

`$#+K!` **wxFrameLayout::FirePluginEvent**

**void FirePluginEvent(cbPluginEvent& event)<sup>K</sup>**

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.

---

`w`xFrameLayout::FirePluginEvent  
`w`xframelayoufirepluginevent  
`b`rowse00459  
`K` wxFrameLayout FirePluginEvent  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")  
`K` FirePluginEvent

**`$#+K!` wxFrameLayout::ForwardMouseEvent**

**`void ForwardMouseEvent(wxMouseEvent& event, cbDockPane* pToPane, int eventType)`<sup>K</sup>**

Delegated from "bar-spy".

---

<sup>w</sup>xFrameLayout::ForwardMouseEvent

<sup>w</sup>xframelayouforwardmouseevent

<sup>b</sup>rowse00460

<sup>K</sup> wxFrameLayout ForwardMouseEvent

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")

<sup>K</sup> ForwardMouseEvent

**`$#+K!` wxFrameLayout::GetBarPane**

**`cbDockPane*` GetBarPane(`cbBarInfo*` *pBar*)<sup>K</sup>**

Returns the pane to which the given bar belongs.

---

`w`xFrameLayout::GetBarPane

`w`xframelayougetbarpane

`b`rowse00461

`K` wxFrameLayout GetBarPane

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

`K` GetBarPane

`$$$K! wxFrameLayout::GetBars`

`BarArrayT& GetBars()`<sup>K</sup>

Gets an array of bars.

---

`wxFrameLayout::GetBars`

`wxframelayougetbars`

`browse00462`

`K wxFrameLayout GetBars`

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

`K GetBars`

`wxFrameLayout::GetClientHeight`

`int GetClientHeight()`

Returns the client height.

---

`wxFrameLayout::GetClientHeight`

`wxframeLayoutGetClientHeight`

`rowse00463`

`wxFrameLayout GetClientHeight`

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

`GetClientHeight`

`$#+K1 wxFrameLayout::GetClientRect`

`wxRect& GetClientRect()`<sup>K</sup>

Returns the client's rectangle.

---

<sup>w</sup>xFrameLayout::GetClientRect

<sup>w</sup>xframelayougetclientrect

<sup>b</sup>rowse00464

<sup>K</sup> wxFrameLayout GetClientRect

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")

<sup>K</sup> GetClientRect

`$#+K!` **wxFrameLayout::GetClientWidth**

**int GetClientWidth()**<sup>K</sup>

Returns the client width.

---

<sup>w</sup>wxFrameLayout::GetClientWidth

<sup>w</sup>wxframeLayoutGetClientWidth

<sup>b</sup>rowse00465

<sup>K</sup>wxFrameLayout GetClientWidth

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframeLayout')")

<sup>K</sup>GetClientWidth

**`$#+K!` wxFrameLayout::GetFrameClient**

**`wxWindow*` GetFrameClient()<sup>K</sup>**

Returns the frame client, or NULL if not present.

---

<sup>w</sup>xFrameLayout::GetFrameClient

<sup>w</sup>xframelayougetframeclient

<sup>b</sup>rowse00466

<sup>K</sup> wxFrameLayout GetFrameClient

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")

<sup>K</sup> GetFrameClient

`$#+K!` **wxFrameLayout::GetPane**

**cbDockPane\*** **GetPane**(int *alignment*)<sup>K</sup>

Returns a pane for the given alignment. See pane alignment types.

---

<sup>w</sup>xFrameLayout::GetPane

<sup>w</sup>xframelayougetpane

<sup>b</sup>rowse00467

<sup>K</sup> wxFrameLayout GetPane

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")

<sup>K</sup> GetPane

`$#+K!` **wxFrameLayout::GetPaneProperties**

**void GetPaneProperties**(**cbCommonPaneProperties&** *props*, **int** *alignment* = *FL\_ALIGN\_TOP*)<sup>K</sup>

Gets the pane properties for the given alignment.

---

<sup>w</sup>xFrameLayout::GetPaneProperties

<sup>w</sup>xframeLayoutGetPaneProperties

<sup>b</sup>rowse00468

<sup>K</sup> wxFrameLayout GetPaneProperties

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframeLayout')")

<sup>K</sup> GetPaneProperties

`$#+K!` **wxFrameLayout::GetPanelsArray**

**cbDockPane\*\* GetPanelsArray()**<sup>K</sup>

Returns an array of panes. Used by update managers.

---

<sup>w</sup>xFrameLayout::GetPanelsArray

<sup>w</sup>xframelayougetpanesarray

<sup>b</sup>rowse00469

<sup>K</sup> wxFrameLayout GetPanelsArray

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")

<sup>K</sup> GetPanelsArray

**`wxFrameLayout::GetParentFrame`**

**`wxWindow& GetParentFrame()`**<sup>K</sup>

Returns the parent frame.

---

<sup>w</sup>`wxFrameLayout::GetParentFrame`  
<sup>w</sup>`wxframelayougetparentframe`  
<sup>b</sup>`rowse00470`  
<sup>K</sup> `wxFrameLayout GetParentFrame`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxframelayout')")`  
<sup>K</sup> `GetParentFrame`

`$#+K!` **wxFrameLayout::GetPrevClientRect**

**wxRect& GetPrevClientRect()**<sup>K</sup>

Returns the previous client window rectangle.

---

<sup>w</sup>xFrameLayout::GetPrevClientRect

<sup>w</sup>xframelayougetprevclientrect

<sup>b</sup>rowse00471

<sup>K</sup> wxFrameLayout GetPrevClientRect

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")

<sup>K</sup> GetPrevClientRect

`$#+K!` **wxFrameLayout::GetTopPlugin**

**cbPluginBase& GetTopPlugin()**<sup>K</sup>

Returns the current top-level plugin (the one that receives events first, except if input events are currently captured by some other plugin).

---

`w`xFrameLayout::GetTopPlugin  
`w`xframelayougettopplugin  
`b`rowse00472  
`K` wxFrameLayout GetTopPlugin  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")  
`K` GetTopPlugin

**`wxFrameLayout::GetUpdatesManager`**

**`cbUpdatesManagerBase& GetUpdatesManager()`**<sup>K</sup>

Returns a reference to the updates manager. Note: in future, the updates manager will become a normal plugin.

---

<sup>w</sup>`wxFrameLayout::GetUpdatesManager`  
<sup>w</sup>`wxframelayougetupdatesmanager`  
<sup>b</sup>`rowse00473`  
<sup>K</sup> `wxFrameLayout GetUpdatesManager`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")`  
<sup>K</sup> `GetUpdatesManager`

`$#+K!` **wxFrameLayout::HasTopPlugin**

**bool HasTopPlugin()**<sup>K</sup>

Returns true if there is a top plugin.

---

<sup>w</sup>xFrameLayout::HasTopPlugin  
<sup>w</sup>xframelayouthastopplugin  
<sup>b</sup>rowse00474  
<sup>K</sup> wxFrameLayout HasTopPlugin  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")  
<sup>K</sup> HasTopPlugin

**wxFrameLayout::HideBarWindows**

**void HideBarWindows()**

Hides the bar windows, and also the client window if present.

---

wxFrameLayout::HideBarWindows  
wxframelayouthidebarwindows  
browse00475  
K wxFrameLayout HideBarWindows  
E nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxframelayout')")  
K HideBarWindows

`$#+K!` **wxFrameLayout::HitTestPane**

**bool HitTestPane(cbDockPane\* pPane, int x, int y)<sup>K</sup>**

Returns TRUE if the position is within the given pane.

---

`w`xFrameLayout::HitTestPane

`w`xframelayouthittestpane

`b`rowse00476

`K` wxFrameLayout HitTestPane

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

`K` HitTestPane

`$#+K!` **wxFrameLayout::HitTestPanes**

**cbDockPane\*** HitTestPanes(const wxRect& *rect*, **cbDockPane\*** *pCurPane*)<sup>K</sup>

Returns the pane for which the rectangle hit test succeeds, giving preference to the given pane if supplied.

---

`w`xFrameLayout::HitTestPanes  
`w`xframelayouthittestpanes  
`b`rowse00477  
`K` wxFrameLayout HitTestPanes  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")  
`K` HitTestPanes

`$#+K!` **wxFrameLayout::HookUpToFrame**

**void HookUpToFrame()**<sup>K</sup>

Hooks the layout up to the frame (pushes the layout onto the frame's event handler stack).

---

`w`xFrameLayout::HookUpToFrame  
`w`xframelayouthookuptoframe  
`b`rowse00478  
`K` wxFrameLayout HookUpToFrame  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")  
`K` HookUpToFrame

**wxFrameLayout::InverseVisibility**

**void InverseVisibility(cbBarInfo\* pBar)**

Toggles the bar between visible and hidden.

---

wxFrameLayout::InverseVisibility  
wxframelayouinversevisibility  
rowse00479  
K wxFrameLayout InverseVisibility  
E nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxframelayout')")  
K InverseVisibility

<sup>S#+K!</sup>**wxFrameLayout::LocateBar**

**bool LocateBar(cbBarInfo\* pBarInfo, cbRowInfo\*\* ppRow, cbDockPane\*\* ppPane)<sup>K</sup>**

The purpose of this function is unknown.

---

<sup>w</sup>wxFrameLayout::LocateBar

<sup>w</sup>wxframelayoulocatebar

<sup>b</sup>rowse00480

<sup>K</sup>wxFrameLayout LocateBar

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxframelayout')")

<sup>K</sup>LocateBar

`$#+K!` **wxFrameLayout::OnActivate**

**void OnActivate(wxActivateEvent& event)<sup>K</sup>**

Handles activation events. Currently does nothing.

---

`w`xFrameLayout::OnActivate  
`w`xframelayouonactivate  
`b`rowse00481  
`K` wxFrameLayout OnActivate  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxframelayout')")  
`K` OnActivate

`$#+K!` **wxFrameLayout::OnEraseBackground**

**void OnEraseBackground(wxEraseEvent& event)<sup>K</sup>**

Handles background erase events. Currently does nothing.

---

`w`xFrameLayout::OnEraseBackground  
`w`xframelayoutonerasebackground  
`b`rowse00482  
`K` wxFrameLayout OnEraseBackground  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")  
`K` OnEraseBackground

`$#+K!` **wxFrameLayout::OnIdle**

**void OnIdle(wxIdleEvent& event)**<sup>K</sup>

Handles idle events.

---

<sup>w</sup>xFrameLayout::OnIdle

<sup>w</sup>xframelayoutonidle

<sup>b</sup>rowse00483

<sup>K</sup> wxFrameLayout OnIdle

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")

<sup>K</sup> OnIdle

`##+K! wxFrameLayout::OnKillFocus`

`void OnKillFocus(wxFocusEvent& event)K`

Handles focus kill events. Currently does nothing.

---

`w_xFrameLayout::OnKillFocus`

`w_xframelayoutonkillfocus`

`b_rowse00484`

`K wxFrameLayout OnKillFocus`

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

`K OnKillFocus`

`wxFrameLayout::OnLButtonDown`

`void OnLButtonDown(wxMouseEvent& event)`

Event handler for a left down button event.

---

`wxFrameLayout::OnLButtonDown`  
`wxframelayouonlbuttondown`  
`rowse00485`  
`wxFrameLayout OnLButtonDown`  
`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxframelayout')")`  
`OnLButtonDown`

`wxFrameLayout::OnLButtonUp`

`void OnLButtonUp(wxMouseEvent& event)`

Event handler for a left button up event.

---

`wxFrameLayout::OnLButtonUp`  
`wxframelayouonlbuttonup`  
`rowse00486`  
`wxFrameLayout OnLButtonUp`  
`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxframelayout')")`  
`OnLButtonUp`

**`wxFrameLayout::OnLDbIcIck`**

**`void OnLDbIcIck(wxMouseEvent& event)`**<sup>K</sup>

Event handler for a left doubleclick button event.

---

<sup>w</sup>xFrameLayout::OnLDbIcIck  
<sup>w</sup>xframelayouonldbIckIck  
<sup>b</sup>rowse00487  
<sup>K</sup> wxFrameLayout OnLDbIcIck  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxframelayout')")  
<sup>K</sup> OnLDbIcIck

`$#+K!` **wxFrameLayout::OnMouseMove**

**void OnMouseMove(wxMouseEvent& event)**<sup>K</sup>

Event handler for a mouse move event.

---

`w`xFrameLayout::OnMouseMove  
`w`xframelayouonmousemove  
`b`rowse00488  
`K` wxFrameLayout OnMouseMove  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")  
`K` OnMouseMove

`$#+K1 wxFrameLayout::OnPaint`

`void OnPaint(wxPaintEvent& event)K`

Handles paint events, calling PaintPane for each pane.

---

`w_xFrameLayout::OnPaint`

`w_xframelayouonpaint`

`b_rowse00489`

`K wxFrameLayout OnPaint`

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

`K OnPaint`

`$#+K!` **wxFrameLayout::OnRButtonDown**

**void OnRButtonDown(wxMouseEvent& *event*)<sup>K</sup>**

Event handler for a right button down event.

---

`w`xFrameLayout::OnRButtonDown  
`w`xframelayouonrbuttondown  
`b`rowse00490  
`K` wxFrameLayout OnRButtonDown  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")  
`K` OnRButtonDown

`wxFrameLayout::OnRButtonUp`

`void OnRButtonUp(wxMouseEvent& event)`

Event handler for a right button up event.

---

`wxFrameLayout::OnRButtonUp`  
`wxframelayoutrbuttonup`  
`rowse00491`  
`wxFrameLayout OnRButtonUp`  
`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxframelayout')")`  
`OnRButtonUp`

`##+K! wxFrameLayout::OnSetFocus`

`void OnSetFocus(wxFocusEvent& event)K`

Handles focus set events. Currently does nothing.

---

`wxFrameLayout::OnSetFocus`

`wxframelayoutonsetfocus`

`rowse00492`

`K wxFrameLayout OnSetFocus`

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

`K OnSetFocus`

`wxFrameLayout::OnSize`

`void OnSize(wxSizeEvent& event)`

Event handler for a size event.

---

`wxFrameLayout::OnSize`  
`wxframelayousize`  
`rowse00493`  
`wxFrameLayout OnSize`  
`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxframelayout')")`  
`OnSize`

`$#+K! wxFrameLayout::PopAllPlugins`

`void PopAllPlugins()`<sup>K</sup>

Pop all plugins.

---

`w` wxFrameLayout::PopAllPlugins

`w` wxframelayouthpopallplugins

`b` rowse00494

`K` wxFrameLayout PopAllPlugins

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

`K` PopAllPlugins

`$$$K! wxFrameLayout::PopPlugin`

`void PopPlugin()K`

Similar to wxWindow's "push/pop-event-handler" methods, except that the plugin is deleted upon "popping".

---

`wxFrameLayout::PopPlugin`  
`wxframeLayoutpopplugin`  
`rowse00495`  
`K wxFrameLayout PopPlugin`  
`E nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxframeLayout')")`  
`K PopPlugin`

**`$#+K!` wxFrameLayout::PositionClientWindow**

**`void PositionClientWindow()`<sup>K</sup>**

Called to apply the calculated layout to window objects.

---

`w`xFrameLayout::PositionClientWindow  
`w`xframelayoupositionclientwindow  
`b`rowse00496  
`K` wxFrameLayout PositionClientWindow  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")  
`K` PositionClientWindow

`$#+K!` **wxFrameLayout::PositionPanes**

**void PositionPanes()**<sup>K</sup>

Called to apply the calculated layout to window objects.

---

<sup>w</sup>wxFrameLayout::PositionPanes  
<sup>w</sup>wxframelayouthpositionpanes  
<sup>b</sup>rowse00497  
<sup>K</sup>wxFrameLayout PositionPanes  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")  
<sup>K</sup>PositionPanes

## `$#+K!` **wxFrameLayout::PushDefaultPlugins**

### **void PushDefaultPlugins()**<sup>K</sup>

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.

---

`w`xFrameLayout::PushDefaultPlugins  
`w`xframelayouthpushdefaultplugins  
`b`rowse00498  
`K` wxFrameLayout PushDefaultPlugins  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")  
`K` PushDefaultPlugins

`$$$K! wxFrameLayout::PushPlugin`

`void PushPlugin(cbPluginBase* pPugin)K`

Similar to wxWindow's "push/pop-event-handler" methods, except that the plugin is deleted upon "popping".

---

`wxFrameLayout::PushPlugin`  
`wxframelayouthpushplugin`  
`rowse00499`  
`K wxFrameLayout PushPlugin`  
`E nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxframelayout')")`  
`K PushPlugin`

`$#+K!` **wxFrameLayout::RecalcLayout**

**void RecalcLayout**(**bool** *repositionBarsNow* = *FALSE*)<sup>K</sup>

Recalculates the layout of panes, and all bars/rows in each pane.

---

`w`xFrameLayout::RecalcLayout  
`w`xframelayoutrcalclayout  
`b`rowse00500  
`K` wxFrameLayout RecalcLayout  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")  
`K` RecalcLayout

`$#+K1` **wxFrameworkLayout::RedockBar**

**bool RedockBar**(**cbBarInfo\*** *pBar*, **const wxRect&** *shapeInParent*, **cbDockPane\*** *pToPane* = *NULL*, **bool** *updateNow* = *TRUE*)<sup>K</sup>

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`  
`wxfameworklayoutredockbar`  
`browse00501`  
`K wxFrameworkLayout RedockBar`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxfameworklayout')")`  
`K RedockBar`

`$#+K!` **wxFrameLayout::RefreshNow**

**void RefreshNow**(**bool** *recalcLayout* = *TRUE*)<sup>K</sup>

Recalculates layout and performs on-screen update of all panes.

---

<sup>w</sup>xFrameLayout::RefreshNow

<sup>w</sup>xframelayourefreshnow

<sup>b</sup>rowse00502

<sup>K</sup> wxFrameLayout RefreshNow

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")

<sup>K</sup> RefreshNow

**`wxFrameLayout::ReleaseEventsFromPane`**

**`void ReleaseEventsFromPane(cbDockPane* fromPane)`**<sup>K</sup>

Called by plugins; also releases mouse in the parent frame.

---

<sup>w</sup>`wxFrameLayout::ReleaseEventsFromPane`  
<sup>w</sup>`wxframelayoureleaseeventsfrompane`  
<sup>b</sup>`rowse00503`  
<sup>K</sup> `wxFrameLayout ReleaseEventsFromPane`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxframelayout')")`  
<sup>K</sup> `ReleaseEventsFromPane`

**`wxFrameLayout::ReleaseEventsFromPlugin`**

**`void ReleaseEventsFromPlugin(cbPluginBase* pPlugin)`**<sup>K</sup>

Releases user input events for the given plugin. Input events are: mouse movement, mouse clicks, keyboard input

---

<sup>w</sup>`wxFrameLayout::ReleaseEventsFromPlugin`  
<sup>w</sup>`wxframelayoureleaseeventsfromplugin`  
<sup>b</sup>`rowse00504`  
<sup>K</sup> `wxFrameLayout ReleaseEventsFromPlugin`  
<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxframelayout')")`  
<sup>K</sup> `ReleaseEventsFromPlugin`

`$#+K!` **wxFrameLayout::RemoveBar**

**void RemoveBar**(`cbBarInfo*` *pBar*)<sup>K</sup>

Removes the bar from the layout permanently, and hides its corresponding window if present.

---

`w`xFrameLayout::RemoveBar  
`w`xframelayouremovebar  
`b`rowse00505  
`K` wxFrameLayout RemoveBar  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")  
`K` RemoveBar

**`wxFrameLayout::RemovePlugin`**

**`void RemovePlugin(wxClassInfo* pPInfo)`**

Checks if the plugin of the given class is hooked, and removes it if found.

---

`wxFrameLayout::RemovePlugin`  
`wxframelayouremoveplugin`  
`rowse00506`  
`wxFrameLayout RemovePlugin`  
`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxframelayout')")`  
`RemovePlugin`

`$$$K! wxFrameLayout::ReparentWindow`

`void ReparentWindow(wxWindow* pChild, wxWindow* pNewParent)K`

Reparents pChild to have parent pNewParent.

---

`wxFrameLayout::ReparentWindow`

`wxframelayourepaintwindow`

`rowse00507`

`K wxFrameLayout ReparentWindow`

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

`K ReparentWindow`

**`wxFrameLayout::RepositionFloatedBar`**

**`void RepositionFloatedBar(cbBarInfo* pBar)`**<sup>K</sup>

Applies the calculated layout to a floating bar.

---

<sup>w</sup>`wxFrameLayout::RepositionFloatedBar`

<sup>w</sup>`wxframelayoutrositionfloatedbar`

<sup>b</sup>`rowse00508`

<sup>K</sup>`wxFrameLayout RepositionFloatedBar`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxframelayout')")`

<sup>K</sup>`RepositionFloatedBar`

**`wxFrameLayout::RouteMouseEvent`**

**`void RouteMouseEvent(wxMouseEvent& event, int pluginEvtType)`**<sup>K</sup>

Routes the mouse event to the appropriate pane.

---

<sup>w</sup>xFrameLayout::RouteMouseEvent  
<sup>w</sup>xframelayoutroutemouseevent  
<sup>b</sup>rowse00509  
<sup>K</sup> wxFrameLayout RouteMouseEvent  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")  
<sup>K</sup> RouteMouseEvent

<sup>S#+K!</sup>**wxFrameLayout::SetBarState**

**void SetBarState**(**cbBarInfo\*** *pBar*, **int** *newState*, **bool** *updateNow*)<sup>K</sup>

Changes the bar's docking state (see possible control bar states).

---

<sup>w</sup>wxFrameLayout::SetBarState

<sup>w</sup>wxframelayousetbarstate

<sup>b</sup>rowse00510

<sup>K</sup>wxFrameLayout SetBarState

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxframelayout')")

<sup>K</sup>SetBarState

<sup>S#+K!</sup>**wxFrameLayout::SetFrameClient**

**void SetFrameClient**(wxWindow\* *pFrameClient*)<sup>K</sup>

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.

---

<sup>w</sup>wxFrameLayout::SetFrameClient

<sup>w</sup>wxframelayousetframeclient

<sup>b</sup>rowse00511

<sup>K</sup>wxFrameLayout SetFrameClient

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")

<sup>K</sup>SetFrameClient

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

**void SetMargins(int top, int bottom, int left, int right, int paneMask = wxALL\_PANES)**<sup>K</sup>

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.

---

`w`xFrameLayout::SetMargins  
`w`xframelayousetmargins  
`b`rowse00512  
`K` wxFrameLayout SetMargins  
`E`nableButton("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`  
`rowse00513`  
`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.

---

`w_xFrameLayout::SetPaneProperties`  
`w_xframelayoutsetpaneproperties`  
`b_rowse00514`  
`K wxFrameLayout SetPaneProperties`  
`E_nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxframelayout')")`  
`K SetPaneProperties`

`$$$K! 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 secenario is very inconvenient and "low-level", so use the `Add/Push/PopPlugin` methods instead.

---

`wxFrameLayout::SetTopPlugin`  
`wxframeLayoutsettopplugin`  
`rowse00515`  
`K wxFrameLayout SetTopPlugin`  
`E nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxframeLayout')")`  
`K SetTopPlugin`

**`wxFrameLayout::SetUpdatesManager`**

**`void SetUpdatesManager(cbUpdatesManagerBase* pUMgr)`**<sup>K</sup>

Destroys the previous manager if any, and sets the new one.

---

<sup>w</sup>xFrameLayout::SetUpdatesManager  
<sup>w</sup>xframelayousetupdatesmanager  
<sup>b</sup>rowse00516  
<sup>K</sup> wxFrameLayout SetUpdatesManager  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframelayout')")  
<sup>K</sup> SetUpdatesManager

**`$#+K!` wxFrameLayout::ShowFloatedWindows**

**`void ShowFloatedWindows(bool show)`<sup>K</sup>**

Shows all floated windows.

---

<sup>w</sup> wxFrameLayout::ShowFloatedWindows

<sup>w</sup> wxframelayouthowfloatedwindows

<sup>b</sup> rowse00517

<sup>K</sup> wxFrameLayout ShowFloatedWindows

<sup>E</sup> nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxframelayout')")

<sup>K</sup> ShowFloatedWindows

**wxFrameLayout::UnhookFromFrame**

**void UnhookFromFrame()**

Unhooks the layout from the frame.

---

wxFrameLayout::UnhookFromFrame  
wxframelayouthookfromframe  
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  
E nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframemanager)")  
K wxFrameManager

**\$#+K! wxFrameManager::~wxFrameManager**

**~wxFrameManager()<sup>K</sup>**

---

<sup>w</sup>xFrameManager::~wxFrameManager

<sup>w</sup>xframemanagerdtor

<sup>b</sup>rowse00521

<sup>K</sup> wxFrameManager ~wxFrameManager

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxframemanager)")

<sup>K</sup> ~wxFrameManager

**`$#+K!` wxFrameManager::ActivateView**

**`void ActivateView(wxFrameView* pFrmView)`<sup>K</sup>**

**`void ActivateView(int viewNo)`<sup>K</sup>**

---

<sup>w</sup>xFrameManager::ActivateView

<sup>w</sup>xframemanageractivateview

<sup>b</sup>rowse00522

<sup>K</sup> wxFrameManager ActivateView

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframemanager)")

<sup>K</sup> ActivateView

<sup>K</sup> ActivateView

**`$#+K!` wxFrameManager::AddView**

**`void AddView(wxFrameView* pFrmView)K`**

---

`w`xFrameManager::AddView  
`w`xframemanageraddview  
`b`rowse00523  
`K` wxFrameManager AddView  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframemanager')")  
`K` AddView

**\$#+K! wxFrameManager::DeactivateCurrentView**

**void DeactivateCurrentView()<sup>K</sup>**

---

<sup>w</sup>xFrameManager::DeactivateCurrentView  
<sup>w</sup>xframemanagerdeactivatecurrentview  
<sup>b</sup>rowse00524  
<sup>K</sup> wxFrameManager DeactivateCurrentView  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxframemanager)")  
<sup>K</sup> DeactivateCurrentView

`$#+K! wxFrameManager::DestroyViews`

`void DestroyViews()K`

---

`w_xFrameManager::DestroyViews`

`w_xframemanagerdestroyviews`

`browse00525`

`K wxFrameManager DestroyViews`

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

`K DestroyViews`

`$#+K! wxFrameManager::DoSerialize`

`void DoSerialize(wxObjectStorage& store)K`

---

`wxFrameManager::DoSerialize`

`wxframemanagerdoserialize`

`rowse00526`

`K wxFrameManager DoSerialize`

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

`K DoSerialize`

`$#+K!` **wxFrameManager::EnableMenusForView**

**void EnableMenusForView(wxFrameView\* pView, bool enable)<sup>K</sup>**

---

`w`xFrameManager::EnableMenusForView

`w`xframemanagereenablemenusforview

`b`rowse00527

`K` wxFrameManager EnableMenusForView

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

`K` EnableMenusForView

**\$#+K! wxFrameManager::GetActiveView**

**wxFrameView\* GetActiveView()<sup>K</sup>**

---

<sup>w</sup>xFrameManager::GetActiveView  
<sup>w</sup>xframemanagergetactiveview  
<sup>b</sup>rowse00528  
<sup>K</sup> wxFrameManager GetActiveView  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframemanager')")  
<sup>K</sup> GetActiveView

**\$#+K! wxFrameManager::GetActiveViewNo**

**int GetActiveViewNo()K**

---

w\_xFrameManager::GetActiveViewNo  
w\_xframemanagergetactiveviewno  
b\_rowse00529  
K wxFrameManager GetActiveViewNo  
E\_nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframemanager)")  
K GetActiveViewNo

**`$#+K!` wxFrameManager::GetActiveViewNode**

**`wxNode*` GetActiveViewNode()<sup>K</sup>**

---

`w`xFrameManager::GetActiveViewNode  
`w`xframemanagergetactiveviewnode  
`b`rowse00530  
`K` wxFrameManager GetActiveViewNode  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframemanager')")  
`K` GetActiveViewNode

**\$#+K! wxFrameManager::GetClientWindow**

**wxWindow\* GetClientWindow()<sup>K</sup>**

---

<sup>w</sup>xFrameManager::GetClientWindow  
<sup>w</sup>xframemanagergetclientwindow  
<sup>b</sup>rowse00531  
<sup>K</sup> wxFrameManager GetClientWindow  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframemanager')")  
<sup>K</sup> GetClientWindow

**`$#+K!` wxFrameManager::GetObjectStore**

**`wxObjectStorage& GetObjectStore()`<sup>K</sup>**

---

<sup>w</sup>xFrameManager::GetObjectStore

<sup>w</sup>xframemanagergetobjectstore

<sup>b</sup>rowse00532

<sup>K</sup> wxFrameManager GetObjectStore

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframemanager')")

<sup>K</sup> GetObjectStore

`$#+K!` **wxFrameManager::GetParentFrame**

**wxFrame\*** GetParentFrame()<sup>K</sup>

synonyms

---

<sup>w</sup>xFrameManager::GetParentFrame

<sup>w</sup>xframemanagergetparentframe

<sup>b</sup>rowse00533

<sup>K</sup> wxFrameManager GetParentFrame

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframemanager')")

<sup>K</sup> GetParentFrame

**`$#+K!` wxFrameManager::GetParentWindow**

**`wxWindow*` GetParentWindow()<sup>K</sup>**

---

`w`xFrameManager::GetParentWindow  
`w`xframemanagergetparentwindow  
`b`rowse00534  
`K` wxFrameManager GetParentWindow  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframemanager')")  
`K` GetParentWindow

**`$#+K!` wxFrameManager::GetView**

**`wxFrameView* GetView(int viewNo)`<sup>K</sup>**

---

`w`xFrameManager::GetView  
`w`xframemanagergetview  
`b`rowse00535  
`K` wxFrameManager GetView  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframemanager')")  
`K` GetView

`$#+K! wxFrameManager::GetViewNo`

`int GetViewNo(wxFrameView* pView)K`

---

`w_xFrameManager::GetViewNo`

`w_xframemanagergetviewno`

`browse00536`

`K wxFrameManager GetViewNo`

`E_nableButton("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

---

`w_xFrameManager::Init`

`w_xframemanagerinit`

`b_rowse00537`

`K wxFrameManager Init`

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

`K Init`

`$#+K!` **wxFrameManager::ReloadViews**

**bool ReloadViews()**<sup>K</sup>

---

<sup>w</sup>wxFrameManager::ReloadViews  
<sup>w</sup>wxframemanagerreloadviews  
<sup>b</sup>rowse00538  
<sup>K</sup>wxFrameManager ReloadViews  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframemanager')")  
<sup>K</sup>ReloadViews

**`$#+K!` wxFrameManager::RemoveView**

**`void RemoveView(wxFrameView* pFrmView)K`**

---

`w`xFrameManager::RemoveView

`w`xframemanagerremoveview

`b`rowse00539

`K` wxFrameManager RemoveView

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

`K` RemoveView

**\$#+K! wxFrameManager::SaveViewsNow**

**void SaveViewsNow()<sup>K</sup>**

---

<sup>w</sup>xFrameManager::SaveViewsNow  
<sup>w</sup>xframemanagersaveviewsnow  
<sup>b</sup>rowse00540  
<sup>K</sup> wxFrameManager SaveViewsNow  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframemanager')")  
<sup>K</sup> SaveViewsNow

**`$#+K!` wxFrameManager::SetClnetWindow**

**`void SetClnetWindow(wxWindow* pFrameClient)K`**

---

`w`xFrameManager::SetClnetWindow

`w`xframemanagersetclinetwindow

`b`rowse00541

`K` wxFrameManager SetClnetWindow

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

`K` SetClnetWindow

**`$#+K!` wxFrameManager::SyncAllMenus**

**`void SyncAllMenus()`<sup>K</sup>**

---

<sup>w</sup>xFrameManager::SyncAllMenus

<sup>w</sup>xframemanagersyncallmenus

<sup>b</sup>rowse00542

<sup>K</sup> wxFrameManager SyncAllMenus

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframemanager)")

<sup>K</sup> SyncAllMenus

**\$#+K! wxFrameManager::ViewsAreLoaded**

**bool ViewsAreLoaded()<sup>K</sup>**

---

<sup>w</sup>xFrameManager::ViewsAreLoaded  
<sup>w</sup>xframemanagerviewsareloaded  
<sup>b</sup>rowse00543  
<sup>K</sup> wxFrameManager ViewsAreLoaded  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxframemanager)")  
<sup>K</sup> ViewsAreLoaded

**`$#+K!` GarbageCollector::GarbageCollector**

**`GarbageCollector()`<sup>K</sup>**

Default constructor.

---

`G`arbageCollector::GarbageCollector  
`g`arbagecollectorgarbagecollector  
`b`rowse00545  
`K` GarbageCollector GarbageCollector  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `garbagecollector')")  
`K` GarbageCollector

**\$#+K! GarbageCollector::~~GarbageCollector**

**~GarbageCollector()K**

Destructor.

---

G arbageCollector::~~GarbageCollector  
g arbagecollectordtor  
b rowse00546  
K GarbageCollector ~GarbageCollector  
E nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `garbagecollector')")  
K ~GarbageCollector

**`GarbageCollector::AddDependency`**

**`void AddDependency(void* pObj, void* pDependsOnObj)`**<sup>K</sup>

Prepare data for garbage collection.

---

<sup>G</sup> `GarbageCollector::AddDependency`

<sup>g</sup> `garbagecollectoradddependency`

<sup>b</sup> `rowse00547`

<sup>K</sup> `GarbageCollector AddDependency`

<sup>E</sup> `enableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `garbagecollector')")`

<sup>K</sup> `AddDependency`

**`$#+K!` GarbageCollector::AddObject**

**`void AddObject(void* pObj, int refCnt = 1)K`**

Prepare data for garbage collection.

---

`G`arbageCollector::AddObject

`g`arbagecollectoraddobject

`b`rowse00548

`K` GarbageCollector AddObject

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

`K` AddObject

**`$#+K!` GarbageCollector::ArrangeCollection**

**`void ArrangeCollection()`<sup>K</sup>**

Executes garbage collection algorithm.

---

<sup>G</sup> GarbageCollector::ArrangeCollection  
<sup>g</sup> garbagecollectorarrangecollection  
<sup>b</sup> rowse00549  
<sup>K</sup> GarbageCollector ArrangeCollection  
<sup>E</sup> nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `garbagecollector')")  
<sup>K</sup> ArrangeCollection

**`$#+K!` GarbageCollector::DestroyItemList**

**`void DestroyItemList(wxList& lst)K`**

Destroys a list of items.

---

`G`arbageCollector::DestroyItemList  
`g`arbagecollectordestroyitelist  
`b`rowse00550  
`K` GarbageCollector DestroyItemList  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `garbagecollector')")  
`K` DestroyItemList

**`$#+K!` GarbageCollector::FindItemNode**

**`wxNode* FindItemNode(void* pForObj)`**<sup>K</sup>

Internal method for finding a node.

---

<sup>G</sup> GarbageCollector::FindItemNode

<sup>g</sup> garbagecollectorfinditemnode

<sup>b</sup> rowse00551

<sup>K</sup> GarbageCollector FindItemNode

<sup>E</sup> enableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `garbagecollector')")

<sup>K</sup> FindItemNode

**`$#+K!` GarbageCollector::FindReferenceFreeItemNode**

**`wxNode* FindReferenceFreeItemNode()`**<sup>K</sup>

Internal method for findind and freeing a node.

---

<sup>G</sup> GarbageCollector::FindReferenceFreeItemNode

<sup>g</sup> garbagecollectorfindreferencefreeitemnode

<sup>b</sup> rowse00552

<sup>K</sup> GarbageCollector FindReferenceFreeItemNode

<sup>E</sup> nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `garbagecollector')")

<sup>K</sup> FindReferenceFreeItemNode

**`$#+K!` GarbageCollector::GetCycledObjects**

**`wxList& GetCycledObjects()`<sup>K</sup>**

Get cycled objects.

---

<sup>G</sup> GarbageCollector::GetCycledObjects

<sup>g</sup> garbagecollectorgetcycledobjects

<sup>b</sup> rowse00553

<sup>K</sup> GarbageCollector GetCycledObjects

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `garbagecollector')")

<sup>K</sup> GetCycledObjects

**`$#+K!` GarbageCollector::GetRegularObjects**

**`wxList& GetRegularObjects()`<sup>K</sup>**

Accesses the results of the algorithm.

---

<sup>G</sup> garbageCollector::GetRegularObjects  
<sup>g</sup> garbagecollectorgetregularobjects  
<sup>b</sup> rowse00554  
<sup>K</sup> GarbageCollector GetRegularObjects  
<sup>E</sup> nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `garbagecollector')")  
<sup>K</sup> GetRegularObjects

**GarbageCollector::RemoveReferencesToNode**

**void RemoveReferencesToNode(wxNode\* pItemNode)**

Remove references to this node.

---

G GarbageCollector::RemoveReferencesToNode

g garbagecollectorremovereferencestonode

browse00555

K GarbageCollector RemoveReferencesToNode

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

K RemoveReferencesToNode

**`$#+K!` GarbageCollector::Reset**

**`void Reset()`<sup>K</sup>**

Removes all data from the garbage collector.

---

<sup>G</sup> GarbageCollector::Reset

<sup>g</sup> garbagecollectorreset

<sup>b</sup> rowse00556

<sup>K</sup> GarbageCollector Reset

<sup>E</sup> enableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `garbagecollector')")

<sup>K</sup> Reset

**`$#+K!` GarbageCollector::ResolveReferences**

**`void ResolveReferences()`<sup>K</sup>**

Internal method for resolving references.

---

<sup>G</sup> GarbageCollector::ResolveReferences  
<sup>g</sup> garbagecollectorresolverefereces  
<sup>b</sup> rowse00557  
<sup>K</sup> GarbageCollector ResolveReferences  
<sup>E</sup> nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `garbagecollector')")  
<sup>K</sup> ResolveReferences

**\$#+K! LayoutManagerBase::~~LayoutManagerBase**

**~LayoutManagerBase()<sup>K</sup>**

Destructor.

---

L\_ayoutManagerBase::~~LayoutManagerBase

l\_ayoutmanagerbasedtor

b\_rowse00559

K\_LayoutManagerBase ~LayoutManagerBase

E\_nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `layoutmanagerbase)")

K\_~LayoutManagerBase

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

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

Constructor.

---

`L` LayoutManagerBase::Layout

`l` layoutmanagerbaselayout

`b` rowse00560

`K` LayoutManagerBase Layout

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

`K` Layout

`$#+K!` **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)<sup>K</sup>**

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)<sup>K</sup>**

Use this constructor if buttons have to be persistent

---

<sup>w</sup> wxNewBitmapButton::wxNewBitmapButton

<sup>w</sup> wxnewbitmapbuttonwxnewbitmapbutton

<sup>b</sup> rowse00562

<sup>K</sup> wxNewBitmapButton wxNewBitmapButton

<sup>E</sup> nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxnewbitmapbutton')")

<sup>K</sup> wxNewBitmapButton

<sup>K</sup> wxNewBitmapButton

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

`~wxNewBitmapButton()`<sup>K</sup>

Destructor.

---

`wxNewBitmapButton::~wxNewBitmapButton`

`wxnewbitmapbuttonondtor`

`rowse00563`

`K wxNewBitmapButton ~wxNewBitmapButton`

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

`K ~wxNewBitmapButton`

`$#+K!` **wxNewBitmapButton::DestroyLabels**

**void DestroyLabels()**<sup>K</sup>

Internal function for destroying labels.

---

`w`xNewBitmapButton::DestroyLabels  
`w`xnewbitmapbuttondestroylabels  
`b`rowse00564  
`K` wxNewBitmapButton DestroyLabels  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxnewbitmapbutton')")  
`K` DestroyLabels

**`wxNewBitmapButton::DrawDecorations`**

**`void DrawDecorations(wxDC& dc)`**<sup>K</sup>

Draws the decorations.

---

<sup>w</sup>`wxNewBitmapButton::DrawDecorations`

<sup>w</sup>`wxnewbitmapbuttondrawdecorations`

<sup>b</sup>`rowse00565`

<sup>K</sup>`wxNewBitmapButton DrawDecorations`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxnewbitmapbutton')")`

<sup>K</sup>`DrawDecorations`

`$#+K!` **wxNewBitmapButton::DrawLabel**

**void DrawLabel(wxDC& dc)**<sup>K</sup>

Draws the label.

---

<sup>w</sup>xNewBitmapButton::DrawLabel

<sup>w</sup>xnewbitmapbuttondrawlabel

<sup>b</sup>rowse00566

<sup>K</sup> wxNewBitmapButton DrawLabel

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxnewbitmapbutton')")

<sup>K</sup> DrawLabel

`$#+K!` **wxNewBitmapButton::DrawShade**

**void DrawShade(int *outerLevel*, wxDC& *dc*, wxPen& *upperLeftSidePen*, wxPen& *lowerRightSidePen*)<sup>K</sup>**

Draws shading on the button.

---

`w`xNewBitmapButton::DrawShade  
`w`xnewbitmapbuttondrawshade  
`b`rowse00567  
`K` wxNewBitmapButton DrawShade  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxnewbitmapbutton')")  
`K` DrawShade

**`$#+K!` wxNewBitmapButton::GetStateLabel**

**`wxBitmap*` GetStateLabel()<sup>K</sup>**

Returns the label that matches the current button state.

---

`w`xNewBitmapButton::GetStateLabel  
`w`xnewbitmapbuttongetstatelabel  
`b`rowse00568  
`K` wxNewBitmapButton GetStateLabel  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxnewbitmapbutton')")  
`K` GetStateLabel

`$#+K!` **wxNewBitmapButton::IsInWindow**

**bool IsInWindow(int x, int y)<sup>K</sup>**

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

`wxNewBitmapButton::OnEraseBackground`

`void OnEraseBackground(wxEraseEvent& event)`

Responds to an erase background event.

---

`wxNewBitmapButton::OnEraseBackground`  
`wxnewbitmapbuttononerasebackground`  
`rowse00570`  
`wxNewBitmapButton OnEraseBackground`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxnewbitmapbutton')")`  
`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

`$#+K!` **wxNewBitmapButton::OnLButtonDown**

**void OnLButtonDown(wxMouseEvent& event)<sup>K</sup>**

Responds to a left mouse button down event.

---

`w`xNewBitmapButton::OnLButtonDown

`w`xnewbitmapbuttononlbuttondown

`b`rowse00572

`K` wxNewBitmapButton OnLButtonDown

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

`K` OnLButtonDown

`$#+K!` **wxNewBitmapButton::OnLButtonUp**

**void OnLButtonUp(wxMouseEvent& *event*)**<sup>K</sup>

Responds to a left mouse button up event.

---

<sup>w</sup>xNewBitmapButton::OnLButtonUp  
<sup>w</sup>xnewbitmapbuttononlbuttonup  
<sup>b</sup>rowse00573  
<sup>K</sup> wxNewBitmapButton OnLButtonUp  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxnewbitmapbutton')")  
<sup>K</sup> OnLButtonUp

`$#+K!` **wxNewBitmapButton::OnMouseMove**

**void OnMouseMove(wxMouseEvent& event)<sup>K</sup>**

Responds to a mouse move event.

---

`w` wxNewBitmapButton::OnMouseMove

`w` wxnewbitmapbuttononmousemove

`b` rowse00574

`K` wxNewBitmapButton OnMouseMove

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

`K` OnMouseMove

`$#+K!` **wxNewBitmapButton::OnPaint**

**void OnPaint(wxPaintEvent& *event*)**<sup>K</sup>

Responds to a paint event.

---

<sup>w</sup>xNewBitmapButton::OnPaint

<sup>w</sup>xnewbitmapbuttononpaint

<sup>b</sup>rowse00575

<sup>K</sup> wxNewBitmapButton OnPaint

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxnewbitmapbutton')")

<sup>K</sup> OnPaint

`$#+K!` **wxNewBitmapButton::OnSize**

**void OnSize(wxSizeEvent& event)**<sup>K</sup>

Responds to a size event.

---

<sup>w</sup>xNewBitmapButton::OnSize  
<sup>w</sup>xnewbitmapbuttononsize  
<sup>b</sup>rowse00576  
<sup>K</sup> wxNewBitmapButton OnSize  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxnewbitmapbutton')")  
<sup>K</sup> OnSize

**`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)<sup>K</sup>**

Renders the label image.

---

`w`xNewBitmapButton::RenderLabelImage  
`w`xnewbitmapbuttonrenderlabelimage  
`b`rowse00578  
`K` wxNewBitmapButton RenderLabelImage  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxnewbitmapbutton')")  
`K` RenderLabelImage

`$#+K!` **wxNewBitmapButton::RenderLabelImages**

**void RenderLabelImages()**<sup>K</sup>

Renders label images.

---

<sup>w</sup>wxNewBitmapButton::RenderLabelImages

<sup>w</sup>wxnewbitmapbuttonrenderlabelimages

<sup>b</sup>rowse00579

<sup>K</sup>wxNewBitmapButton RenderLabelImages

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxnewbitmapbutton')")

<sup>K</sup>RenderLabelImages

`$#+K!` **wxNewBitmapButton::Reshape**

**void Reshape()**<sup>K</sup>

This function should be called after Create. It renders the labels, having reloaded the button image if necessary.

---

`w`xNewBitmapButton::Reshape  
`w`xnewbitmapbuttonreshape  
`b`rowse00580  
`K` wxNewBitmapButton Reshape  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxnewbitmapbutton')")  
`K` Reshape

**`wxNewBitmapButton::SetAlignments`**

**`void SetAlignments(int alignText = NB_ALIGN_TEXT_BOTTOM, int marginX = 2, int marginY = 2, int textToLabelGap = 2)`**<sup>K</sup>

Sets the text alignment and margins.

---

`wxNewBitmapButton::SetAlignments`  
`wxnewbitmapbuttonsetalignments`  
`rowse00581`  
`wxNewBitmapButton SetAlignments`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxnewbitmapbutton')")`  
`SetAlignments`

`$#+K!` **wxNewBitmapButton::SetLabel**

**void SetLabel(const wxBitmap& *labelBitmap*, const wxString& *labelText* = "")**<sup>K</sup>

Sets the label and optionally label text.

---

<sup>w</sup>xNewBitmapButton::SetLabel

<sup>w</sup>xnewbitmapbuttonsetlabel

<sup>b</sup>rowse00582

<sup>K</sup> wxNewBitmapButton SetLabel

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxnewbitmapbutton')")

<sup>K</sup> SetLabel

`$#+K!` **wxToolWindow::wxToolWindow**

**wxToolWindow()**<sup>K</sup>

Default constructor.

---

`w`xToolWindow::wxToolWindow  
`w`xtoolwindowwxtoolwindow  
`b`rowse00585  
`K` wxToolWindow wxToolWindow  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxtoolwindow')")  
`K` wxToolWindow

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

`~wxToolWindow()K`

Destructor.

---

`wxToolWindow::~~wxToolWindow`

`wxtoolwindowdctor`

`rowse00586`

`K wxToolWindow ~wxToolWindow`

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

`K ~wxToolWindow`

`$#+K!` **wxToolWindow::AddMiniButton**

**void AddMiniButton(cbMiniButton\* pBtn)<sup>K</sup>**

Adds a button. Buttons are added in right-to-left order.

---

`w`xToolWindow::AddMiniButton

`w`xtoolwindowadminibutton

`b`rowse00587

`K` wxToolWindow AddMiniButton

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

`K` AddMiniButton

`$#+K!` **wxToolWindow::AdjustRectPos**

**void AdjustRectPos(const wxRect& *original*, const wxSize& *newDim*, wxRect& *newRect*)<sup>K</sup>**

Helper function.

---

`w`xToolWindow::AdjustRectPos  
`w`xtoolwindowadjustrectpos  
`b`rowse00588  
`K` wxToolWindow AdjustRectPos  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxtoolwindow')")  
`K` AdjustRectPos

**`wxToolWindow::CalcResizedRect`**

**`void CalcResizedRect(wxRect& rect, wxPoint& delta, const wxSize& minDim)`**<sup>K</sup>

Calculate resized rectangle.

---

<sup>w</sup>`wxToolWindow::CalcResizedRect`

<sup>w</sup>`xtoolwindowcalresizedrect`

<sup>b</sup>`rowse00589`

<sup>K</sup>`wxToolWindow CalcResizedRect`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxtoolwindow')")`

<sup>K</sup>`CalcResizedRect`

`wxToolWindow::DrawHintRect`

`void DrawHintRect(const wxRect& r)`

Draws the hint rectangle.

---

`wxToolWindow::DrawHintRect`

`wxtoolwindowdrawhintrect`

`browser00590`

`wxToolWindow DrawHintRect`

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

`DrawHintRect`

`$#+K!` **wxToolWindow::GetClient**

**wxWindow\*** **GetClient()**<sup>K</sup>

Returns the client window.

---

<sup>w</sup>xToolWindow::GetClient

<sup>w</sup>xtoolwindowgetclient

<sup>b</sup>rowse00591

<sup>K</sup> wxToolWindow GetClient

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxtoolwindow')")

<sup>K</sup> GetClient

`$#+K!` **wxToolWindow::GetMinimalWndDim**

**wxSize GetMinimalWndDim()**<sup>K</sup>

Helper function.

---

`w`xToolWindow::GetMinimalWndDim  
`w`xtoolwindowgetminimalwnddim  
`b`rowse00592  
`K` wxToolWindow GetMinimalWndDim  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxtoolwindow')")  
`K` GetMinimalWndDim

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

**wxSize GetPreferredSize(const wxSize& *given*)**<sup>K</sup>

Returns the preferred size for the window.

---

<sup>w</sup>xToolWindow::GetPreferredSize

<sup>w</sup>xtoolwindowgetpreferredsize

<sup>b</sup>rowse00593

<sup>K</sup> wxToolWindow GetPreferredSize

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxtoolwindow')")

<sup>K</sup> GetPreferredSize

`$#+K!` **wxToolWindow::GetScrMousePos**

**void GetScrMousePos(wxMouseEvent& event, wxPoint& pos)<sup>K</sup>**

Gets the mouse position in screen coordinates.

---

`w`xToolWindow::GetScrMousePos

`w`xtoolwindowgetscrmousepos

`b`rowse00594

`K` wxToolWindow GetScrMousePos

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

`K` GetScrMousePos

**wxToolWindow::GetScrWindowRect**

**void GetScrWindowRect(wxRect& r)**

Maps client coordinates to screen coordinates.

---

wxToolWindow::GetScrWindowRect  
wxtoolwindowgetscrwindowrect  
browse00595  
K wxToolWindow GetScrWindowRect  
E nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxtoolwindow)")  
K GetScrWindowRect

`$#+K!` **wxToolWindow::HandleTitleClick**

**bool HandleTitleClick(wxMouseEvent& event)<sup>K</sup>**

Handles clicking on the title. By default, does nothing.

---

`w`xToolWindow::HandleTitleClick  
`w`xtoolwindowhandletitleclick  
`b`rowse00596  
`K` wxToolWindow HandleTitleClick  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxtoolwindow')")  
`K` HandleTitleClick

`$#+K!` **wxToolWindow::HitTestWindow**

**int HitTestWindow(wxMouseEvent& event)<sup>K</sup>**

Tests if the mouse position is in this window.

---

`w`xToolWindow::HitTestWindow

`w`xtoolwindowhittestwindow

`b`rowse00597

`K` wxToolWindow HitTestWindow

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

`K` HitTestWindow

`wxToolWindow::LayoutMiniButtons`

`void LayoutMiniButtons()`

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)`**<sup>K</sup>

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)`<sup>K</sup>

Responds to a mouse left down event.

---

`wxToolWindow::OnLeftDown`

`wxtoolwindowonleftdown`

`browser00600`

`K wxToolWindow OnLeftDown`

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

`K OnLeftDown`

`wxToolWindow::OnLeftUp`

`void OnLeftUp(wxMouseEvent& event)`<sup>K</sup>

Responds to a mouse left up event.

---

`wxToolWindow::OnLeftUp`  
`wxtoolwindowonleftup`  
`rowse00601`  
`wxToolWindow OnLeftUp`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxtoolwindow)")`  
`OnLeftUp`<sup>K</sup>

**`wxToolWindow::OnMiniButtonClicked`**

**`void OnMiniButtonClicked(int btnIdx)`**<sup>K</sup>

Called when a mini button is clicked. By default, does nothing.

---

`wxToolWindow::OnMiniButtonClicked`  
`wxtoolwindowonminibuttonclicked`  
`rowse00602`  
`wxToolWindow OnMiniButtonClicked`  
`EnableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxtoolwindow)")`  
`OnMiniButtonClicked`

`$#+K!` **wxToolWindow::OnMotion**

**void OnMotion(wxMouseEvent& event)<sup>K</sup>**

Responds to a mouse move event.

---

`w`xToolWindow::OnMotion  
`w`xtoolwindowonmotion  
`b`rowse00603  
`K` wxToolWindow OnMotion  
`E`nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxtoolwindow')")  
`K` OnMotion

**`wxToolWindow::OnPaint`**

**`void OnPaint(wxPaintEvent& event)`**<sup>K</sup>

Responds to a paint event.

---

<sup>w</sup>`wxToolWindow::OnPaint`

<sup>w</sup>`xtoolwindowonpaint`

<sup>b</sup>`rowse00604`

<sup>K</sup>`wxToolWindow OnPaint`

<sup>E</sup>`nableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxtoolwindow)")`

<sup>K</sup>`OnPaint`

`wxToolWindow::OnSize`

`void OnSize(wxSizeEvent& event)`

Responds to a size event.

---

`wxToolWindow::OnSize`  
`wxtoolwindow::OnSize`  
`rowse00605`  
`wxToolWindow OnSize`  
`enableButton("Up");ChangeButtonBinding("Up", "JumpId(^fl.hlp', `wxtoolwindow)")`  
`OnSize`

`wxToolWindow::SetClient`

`void SetClient(wxWindow* pWnd)`

Sets the client for this tool window.

---

`wxToolWindow::SetClient`

`wxtoolwindowsetclient`

`rowse00606`

`wxToolWindow SetClient`

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

`SetClient`

`$#+K!` **wxToolWindow::SetHintCursor**

**void SetHintCursor(int *type*)**<sup>K</sup>

Sets the hint cursor.

---

<sup>w</sup>xToolWindow::SetHintCursor  
<sup>w</sup>xtoolwindowsethintcursor  
<sup>b</sup>rowse00607  
<sup>K</sup> wxToolWindow SetHintCursor  
<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxtoolwindow')")  
<sup>K</sup> SetHintCursor

**wxToolWindow::SetTitleFont**

**void SetTitleFont(wxFont& font)**<sup>K</sup>

Sets the title font.

---

<sup>w</sup>xToolWindow::SetTitleFont

<sup>w</sup>xtoolwindowsettitlefont

<sup>b</sup>rowse00608

<sup>K</sup> wxToolWindow SetTitleFont

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `wxtoolwindow')")

<sup>K</sup> SetTitleFont

## **<sup>##+K!</sup> 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);
}
```

---

<sup>A</sup> row of all non-fixed bars don't position properly

<sup>t</sup>opic3

<sup>b</sup>rowse00614

<sup>K</sup> A row of all non-fixed bars don't position properly

<sup>E</sup>nableButton("Up");ChangeButtonBinding("Up", "JumpId(`fl.hlp', `faq')")





