User Manual for Tab Classes Version 1.0

Julian Smart
Artificial Intelligence Applications Institute
University of Edinburgh
EH1 1HN

April 24th 1996

Contents

Copyright notice

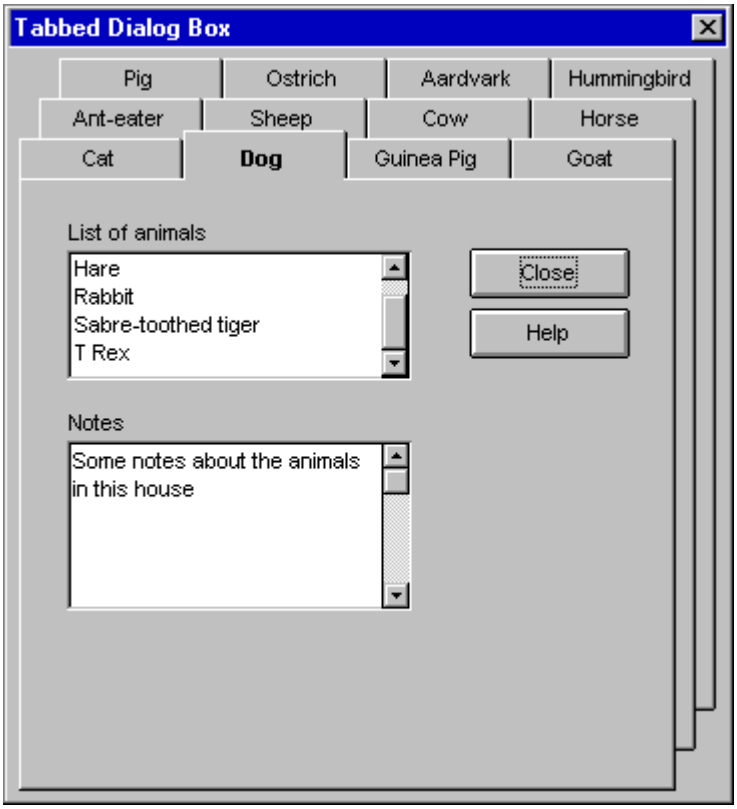
1. Introduction

The tab classes provides a way to display rows of tabs (like file divider tabs), which can be used to switch between panels or other information. Tabs are most commonly used in dialog boxes where the number of options is too great to fit on one dialog.

This code compiles and runs on MS Windows and Motif, but limitations of XView (panel refresh, no nested subwindows) prevent it from being used on that platform.

1.1. The appearance and behaviour of a wxTabbedDialogBox

The following screenshot shows the appearance of the sample tabbed dialog application.



By clicking on the tabs, the user can display a different set of controls. In the example, the Close and Help buttons remain constant. These two buttons are children of the main dialog box, whereas the other controls are children of panels which are shown and hidden according to which tab is active.

A tabbed dialog may have several layers (rows) of tabs, each being offset vertically and horizontally from the previous. Tabs work in columns, in that when a tab is pressed, it swaps place with the tab on the first row of the same column, in order to give the effect of displaying that tab. All tabs must be of the same width. This is a constraint of the implementation, but it also means that the user will find it easier to find tabs since there are distinct tab columns. On some tabbed dialog implementations, tabs jump around seemingly randomly because tabs have different widths. In this implementation, a tab can always be found on the same column.

Tabs are always drawn along the top of the view area; the implementation does not allow for

vertical tabs or any other configuration.

2. Files

The class library comprises the following files, plus makefiles:

- wxtab.cpp: implementation
- wxtab.h: header
- test.h: test application header
- test.cpp: test application implementation

3. Alphabetical class reference

Tab classes overview (page 18)

3.1. wxPanelTabView: wxObject

wxTabView overview (page 20)

wxPanelTabView::wxPanelTabView

void wxPanelTabView(**wxPanel ***_panel_, **long** _style_ = _wxTAB_STYLE_DRAW_BOX_ | _wxTAB_STYLE_COLOUR_INTERIOR_)

Constructor. *panel* should be a wxTabbedPanel or wxTabbedDialogBox: the type will be checked by the view at run time.

style may be a bit list of the following:

wxTAB_STYLE_DRAW_BOX Draw a box around the view area. Most commonly used for dialogs.
wxTAB_STYLE_COLOUR_INTERIOR Draw tab backgrounds in the specified colour. Omitting this style will ensure that the tab background matches the dialog background.

wxPanelTabView::~wxPanelTabView

void ~wxPanelTabView(**void**)

Destructor. This destructor deletes all the panels associated with the view. If you do not wish this to happen, call ClearWindows with argument FALSE before the view is likely to be destroyed. This will clear the list of windows, without deleting them.

wxPanelTabView::AddTabWindow

void AddTabPanel(**int** _id_, **wxWindow ***_window_)

Adds a window to the view. The window is associated with the tab identifier, and will be shown or hidden as the tab is selected or deselected.

wxPanelTabView::ClearWindows

void ClearWindows(**Bool** _deleteWindows = TRUE_)

Removes the child windows from the view. If *deleteWindows* is TRUE, the windows will be deleted.

wxPanelTabView::GetCurrentWindow

wxPanel * GetCurrentWindow(**void**)

Returns the child window currently being displayed on the tabbed panel or dialog box.

wxPanelTabView::GetTabWindow

wxWindow * GetTabWindow(**int** *id*)

Returns the window associated with the tab identifier.

wxPanelTabView::ShowWindowForTab

void ShowWindowForTab(**int** *id*)

Shows the child window corresponding to the tab identifier, and hides the previously shown window.

3.2. wxTabbedDialogBox: wxDialogBox

Tab classes overview (page 18)

wxTabbedDialogBox::wxTabbedDialogBox

void wxTabbedDialogBox(**wxWindow ***parent*, **char ***title*, **Bool** *modal*, **int** *x*, **int** *y*, **int** *width*, **int** *height*, **long** *style=wxDEFAULT_DIALOG_STYLE*, **char ***name="dialogBox"*)

Constructor.

wxTabbedDialogBox::~wxTabbedDialogBox

void ~wxTabbedDialogBox(**void**)

Destructor. This destructor deletes the tab view associated with the dialog box. If you do not wish this to happen, set the tab view to NULL before destruction (for example, in the OnClose member).

wxTabbedDialogBox::SetTabView

void SetTabView(**wxTabView ***view*)

Sets the tab view associated with the dialog box.

wxTabbedDialogBox::GetTabView

wxTabView * GetTabView(**void**)

Returns the tab view associated with the dialog box.

3.3. wxTabbedPanel: wxPanel

Tab classes overview (page 18)

wxTabbedPanel::wxTabbedPanel

void wxTabbedPanel(**wxWindow ***_parent_, **int** _x_, **int** _y_, **int** _width_, **int** _height_, **long** _style=0_, **char ***_name="panel"_)

Constructor.

wxTabbedPanel::SetTabView

void SetTabView(**wxTabView ***_view_)

Sets the tab view associated with the panel.

wxTabbedPanel::GetTabView

wxTabView * GetTabView(**void**)

Returns the tab view associated with the panel.

3.4. wxTabControl: wxObject

Tab classes overview (page 18)

You will rarely need to use this class directly.

wxTabControl::wxTabControl

void wxTabControl(**wxTabView ***_view = NULL_)

Constructor.

wxTabControl::GetColPosition

int GetColPosition(**void**)

Returns the position of the tab in the tab column.

wxTabControl::GetFont

wxFont * GetFont(**void**)

Returns the font to be used for this tab.

wxTabControl::GetHeight

int GetHeight(**void**)

Returns the tab height.

wxTabControl::GetId

int GetId(**void**)

Returns the tab identifier.

wxTabControl::GetLabel

wxString GetLabel(**void**)

Returns the tab label.

wxTabControl::GetRowPosition

int GetRowPosition(**void**)

Returns the position of the tab in the layer or row.

wxTabControl::GetSelected

Bool GetSelected(**void**)

Returns the selected flag.

wxTabControl::GetWidth

int GetWidth(**void**)

Returns the tab width.

wxTabControl::GetX

int GetX(**void**)

Returns the x offset from the top-left of the view area.

wxTabControl::GetY

int GetY(**void**)

Returns the y offset from the top-left of the view area.

wxTabControl::HitTest

Bool HitTest(**int** *x*, **int** *y*)

Returns TRUE if the point x, y is within the tab area.

wxTabControl::OnDraw

void OnDraw(**wxDC ***dc*, **Bool** *lastInRow*)

Draws the tab control on the given device context.

wxTabControl::SetColPosition

void SetColPosition(**int** *pos*)

Sets the position in the column.

wxTabControl::SetFont

void SetFont(**wxFont ***font*)

Sets the font to be used for this tab.

wxTabControl::SetId

void SetId(**int** *id*)

Sets the tab identifier.

wxTabControl::SetLabel

void SetLabel(**const wxString&** *str*)

Sets the label for the tab.

wxTabControl::SetPosition

void SetPosition(**int** *x*, **int** *y*)

Sets the x and y offsets for this tab, measured from the top-left of the view area.

wxTabControl::SetRowPosition

void SetRowPosition(**int** *pos*)

Sets the position on the layer (row).

wxTabControl::SetSelected

void SetSelected(**Bool** *selected*)

Sets the selection flag for this tab (does not set the current tab for the view; use wxTabView::SetSelectedTab for that).

wxTabControl::SetSize

void SetSize(**int** *width*, **int** *height*)

Sets the width and height for this tab.

3.5. wxTabView: wxObject

wxTabView overview (page 20)

wxTabView::wxTabView

void wxTabView(**long** *style = wxTAB_STYLE_DRAW_BOX | wxTAB_STYLE_COLOUR_INTERIOR*)

Constructor.

style may be a bit list of the following:

wxTAB_STYLE_DRAW_BOX Draw a box around the view area. Most commonly used for dialogs.
wxTAB_STYLE_COLOUR_INTERIOR Draw tab backgrounds in the specified colour. Omitting this style will ensure that the tab background matches the dialog background.

wxTabView::AddTab

wxTabControl * AddTab(**int** *id*, **const wxString&** *label*, **wxTabControl ***existingTab=NULL*)

Adds a tab to the view.

id is the application-chosen identifier for the tab, which will be used in subsequent tab operations.

label is the label to give the tab.

existingTab maybe NULL to specify a new tab, or non-NULL to indicate that an existing tab should be used.

A new layer (row) is started when the current layer has been filled up with tabs.

wxTabView::CalculateTabWidth

int CalculateTabWidth(**int** *noTabs*, **Bool** *adjustView = FALSE*)

The application can specify the tab width using this function, in terms of the number of tabs per layer (row) which will fit the view area, which should have been set previously with SetViewRect.

noTabs is the number of tabs which should take up the full width of the view area.

adjustView can be set to TRUE in order to readjust the view width to exactly fit the given number of tabs.

The new tab width is returned.

wxTabView::ClearTabs

void ClearTabs(**Bool** *deleteTabs=TRUE*)

Clears the tabs, deleting them if *deleteTabs* is TRUE.

wxTabView::Draw

void Draw(**void**)

Draws the tab and (optionally) a box around the view area.

wxTabView::FindTabControlForId

wxTabControl * FindTabControlForId(**int** *id*)

Finds the wxTabControl corresponding to *id*.

wxTabView::FindTabControlForPosition

wxTabControl * FindTabControlForPosition(**int** *layer*, **int** *position*)

Finds the wxTabControl at layer *layer*, position in layer *position*, both starting from zero. Note that tabs change layer as they are selected or deselected.

wxTabView::GetBackgroundBrush

wxBrush * GetBackgroundBrush(**void**)

Returns the brush used to draw in the background colour. It is set when SetBackgroundColour is called.

wxTabView::GetBackgroundColour

wxColour GetBackgroundColour(**void**)

Returns the colour used for each tab background. By default, this is light grey. To ensure a match with the dialog or panel background, omit the wxTAB_STYLE_COLOUR_INTERIOR flag from the wxTabView constructor.

wxTabView::GetBackgroundPen

wxPen * GetBackgroundPen(**void**)

Returns the pen used to draw in the background colour. It is set when SetBackgroundColour is called.

wxTabView::GetDC

wxDC * GetDC(**void**)

Returns the current device context for the view.

wxTabView::GetHighlightColour

wxColour GetHighlightColour(**void**)

Returns the colour used for bright highlights on the left side of '3D' surfaces. By default, this is white.

wxTabView::GetHighlightPen

wxPen * GetHighlightPen(**void**)

Returns the pen used to draw 3D effect highlights. This is set when SetHighlightColour is called.

wxTabView::GetHorizontalTabOffset

int GetHorizontalTabOffset(**void**)

Returns the horizontal spacing by which each tab layer is offset from the one below.

wxTabView::GetNumberOfLayers

int GetNumberOfLayers(**void**)

Returns the number of layers (rows of tabs).

wxTabView::GetSelectedTabFont

wxFont * GetSelectedTabFont(**void**)

Returns the font to be used for the selected tab label.

wxTabView::GetShadowColour

wxColour GetShadowColour(**void**)

Returns the colour used for shadows on the right-hand side of '3D' surfaces. By default, this is dark grey.

wxTabView::GetTabHeight

int GetTabHeight(**void**)

Returns the tab default height.

wxTabView::GetTabFont

wxFont * GetTabFont(**void**)

Returns the tab label font.

wxTabView::GetTabSelectionHeight

int GetTabSelectionHeight(**void**)

Returns the height to be used for the currently selected tab; normally a few pixels higher than the other tabs.

wxTabView::GetTabStyle

long GetTabStyle(**void**)

Returns the tab style. See constructor documentation for details of valid styles.

wxTabView::GetTabWidth

int GetTabWidth(**void**)

Returns the tab default width.

wxTabView::GetTextColour

wxColour GetTextColour(**void**)

Returns the colour used to draw label text. By default, this is black.

wxTabView::GetTopMargin

int GetTopMargin(**void**)

Returns the height between the top of the view area and the bottom of the first row of tabs.

wxTabView::GetShadowPen

wxPen * GetShadowPen(**void**)

Returns the pen used to draw 3D effect shadows. This is set when SetShadowColour is called.

wxTabView::GetViewRect

wxRectangle GetViewRect(**void**)

Returns the rectangle specifying the view area (above which tabs are placed).

wxTabView::GetVerticalTabTextSpacing

int GetVerticalTabTextSpacing(**void**)

Returns the vertical spacing between the top of an unselected tab, and the tab label.

wxTabView::OnCreateTabControl

wxTabControl * OnCreateTabControl(**void**)

Creates a new tab control. By default, this returns a wxTabControl object, but the application may wish to define a derived class, in which case the tab view should be subclassed and this function overridden.

wxTabView::Layout

void Layout(**void**)

Recalculates the positions of the tabs, and adjusts the layer of the selected tab if necessary.

You may want to call this function if the view width has changed (for example, from an OnSize handler).

wxTabView::OnEvent

Bool OnEvent(**wxMouseEvent&** *event*)

Processes mouse events sent from the panel or dialog. Returns TRUE if the event was processed, FALSE otherwise.

wxTabView::OnTabActivate

void OnTabActivate(**int** *activateId*, **int** *deactivateId*)

Called when a tab is activated, with the new active tab id, and the former active tab id.

wxTabView::OnTabPreActivate

Bool OnTabPreActivate(**int** *activateId*, **int** *deactivateId*)

Called just before a tab is activated, with the new active tab id, and the former active tab id.

If the function returns FALSE, the tab is not activated.

wxTabView::SetBackgroundColour

void SetBackgroundColour(**const wxColour&** *col*)

Sets the colour to be used for each tab background. By default, this is light grey. To ensure a match with the dialog or panel background, omit the wxTAB_STYLE_COLOUR_INTERIOR flag from the wxTabView constructor.

wxTabView::SetDC

void SetDC(**wxDC ***dc*)

Set the device context that the tab view will use for drawing onto. You must specify this before drawing takes place (automatically set by wxTabbedDialogBox and wxTabbedPanel).

wxTabView::SetHighlightColour

void SetHighlightColour(**const wxColour&** *col*)

Sets the colour to be used for bright highlights on the left side of '3D' surfaces. By default, this is white.

wxTabView::SetHorizontalTabOffset

void SetHorizontalTabOffset(**int** *offset*)

Sets the horizontal spacing by which each tab layer is offset from the one below.

wxTabView::SetSelectedTabFont

void SetSelectedTabFont(**wxFont ***font*)

Sets the font to be used for the selected tab label.

wxTabView::SetShadowColour

void SetShadowColour(**const wxColour&** *col*)

Sets the colour to be used for shadows on the right-hand side of '3D' surfaces. By default, this is dark grey.

wxTabView::SetTabFont

void SetTabFont(**wxFont ***font*)

Sets the tab label font.

wxTabView::SetTabStyle

void SetTabStyle(**long** *tabStyle*)

Sets the tab style. See constructor documentation for details of valid styles.

wxTabView::SetTabSize

void SetTabSize(**int** *width*, **int** *height*)

Sets the tab default width and height.

wxTabView::SetTabSelectionHeight

void SetTabSelectionHeight(**int** *height*)

Sets the height to be used for the currently selected tab; normally a few pixels higher than the other tabs.

wxTabView::SetTabSelection

void SetTabSelection(**int** *sel*, **Bool** *activateTool=TRUE*)

Sets the selected tab, calling the application's OnTabActivate function.

If *activateTool* is FALSE, OnTabActivate will not be called.

wxTabView::SetTextColour

void SetTextColour(**const wxColour&** *col*)

Sets the colour to be used to draw label text. By default, this is black.

wxTabView::SetTopMargin

void SetTopMargin(**int** *margin*)

Sets the height between the top of the view area and the bottom of the first row of tabs.

wxTabView::SetVerticalTabTextSpacing

void SetVerticalTabTextSpacing(**int** *spacing*)

Sets the vertical spacing between the top of an unselected tab, and the tab label.

wxTabView::SetViewRect

void SetViewRect(**const wxRectangle&** *rect*)

Sets the rectangle specifying the view area (above which tabs are placed). This must be set by the application.

4. Classes by category

A classification of tab classes by category.

4.1. View classes

- *wxTabView* (page 9)
- *wxPanelTabView* (page 4)
- *wxTabControl* (page 6)

4.2. Window classes

- *wxTabbedDialogBox* (page 5)
- *wxTabbedPanel* (page 5)

5. Topic overviews

This chapter contains a selection of topic overviews.

5.1. Tab classes overview

Classes: *wxTabView* (page 9), *wxPanelTabView* (page 4), *wxTabbedPanel* (page 5), *wxTabbedDialogBox* (page 5), *wxTabControl* (page 6)

The tab classes provide facilities for switching between contexts by means of 'tabs', which look like file divider tabs.

To use them, include the wxtab.h header file and link with the wxtab.lib (or libwxtab_motif.a) library.

You must create both a *view* to handle the tabs, and a *window* to display the tabs and related information. The wxTabbedDialogBox and wxTabbedPanel classes are provided for convenience, but you could equally well construct your own window class and derived tab view.

If you wish to display a tabbed dialog - the most common use - you should follow these steps.

1. Create a new wxTabbedDialogBox class, and any buttons you wish always to be displayed (regardless of which tab is active).
2. Create a new wxPanelTabView, passing the dialog as the first argument.
3. Set the view rectangle with *wxTabView::SetViewRect* (page 16), to specify the area in which child panels will be shown. The tabs will sit on top of this view rectangle.
4. Call *wxTabView::CalculateTabWidth* (page 9) to calculate the width of the tabs based on the view area. This is optional if, for example, you have one row of tabs which does not extend the full width of the view area.
5. Call *wxTabView::AddTab* (page 9) for each of the tabs you wish to create, passing a unique identifier and a tab label.
6. Construct a number of windows, one for each tab, and call *wxPanelTabView::AddTabWindow* (page 4) for each of these, passing a tab identifier and the window.
7. Set the tab selection.
8. Show the dialog.

Under Motif, you may also need to size the dialog just before setting the tab selection, for unknown reasons.

Some constraints you need to be aware of:

- All tabs must be of the same width.
- Omit the wxTAB_STYLE_COLOUR_INTERIOR flag to ensure that the dialog background and tab backgrounds match.

5.1.1. Example

The following fragment is taken from the file test.cpp.

```
void MyFrame::TestTabbedDialog(void)
{
  int dialogWidth = 365;
```

```
  int dialogHeight = 400;

  wxTabbedDialogBox *dialog =
    new wxTabbedDialogBox(this, "Tabbed Dialog Box", TRUE, -1, -1, 365,
400);

  wxButton *okButton = new wxButton(dialog, (wxFunction)GenericOk,
"Close",
    230, 100, 80, 25);
  wxButton *cancelButton = new wxButton(dialog, NULL, "Help", 230, 130,
80, 25);

  // Note, omit the wxTAB_STYLE_COLOUR_INTERIOR, so we will guarantee a
match
  // with the panel background, and save a bit of time.
  wxPanelTabView *view = new wxPanelTabView(dialog,
wxTAB_STYLE_DRAW_BOX);

  wxRectangle rect;
  rect.x = 5;
  rect.y = 70;
  // Could calculate the view width from the tab width and spacing,
  // as below, but let's assume we have a fixed view width.
//  rect.width = view->GetTabWidth()*4 + 3*view-
>GetHorizontalTabSpacing();
  rect.width = 326;
  rect.height = 300;

  view->SetViewRect(rect);

  // Calculate the tab width for 4 tabs, based on a view width of 326
and
  // the current horizontal spacing. Adjust the view width to exactly
fit
  // the tabs.
  view->CalculateTabWidth(4, TRUE);

  if (!view->AddTab(TEST_TAB_CAT,        wxString("Cat")))
    return;

  if (!view->AddTab(TEST_TAB_DOG,        wxString("Dog")))
    return;
  if (!view->AddTab(TEST_TAB_GUINEAPIG,  wxString("Guinea Pig")))
    return;
  if (!view->AddTab(TEST_TAB_GOAT,       wxString("Goat")))
    return;
  if (!view->AddTab(TEST_TAB_ANTEATER,   wxString("Ant-eater")))
    return;
  if (!view->AddTab(TEST_TAB_SHEEP,      wxString("Sheep")))
    return;
  if (!view->AddTab(TEST_TAB_COW,        wxString("Cow")))
    return;
  if (!view->AddTab(TEST_TAB_HORSE,      wxString("Horse")))
    return;
  if (!view->AddTab(TEST_TAB_PIG,        wxString("Pig")))
    return;
  if (!view->AddTab(TEST_TAB_OSTRICH,    wxString("Ostrich")))
```

```
    return;
  if (!view->AddTab(TEST_TAB_AARDVARK,   wxString("Aardvark")))
    return;
  if (!view->AddTab(TEST_TAB_HUMMINGBIRD,wxString("Hummingbird")))
    return;

  // Add some panels
  wxPanel *panel1 = new wxPanel(dialog, rect.x + 20, rect.y + 10, 200,
250);
  (void)new wxButton(panel1, NULL, "Press me");
  panel1->NewLine();
  (void)new wxText(panel1, NULL, "Input:", "1234", -1, -1, 120);

  view->AddTabWindow(TEST_TAB_CAT, panel1);

  wxPanel *panel2 = new wxPanel(dialog, rect.x + 20, rect.y + 10, 200,
250);
  panel2->SetLabelPosition(wxVERTICAL);

  char *animals[] = { "Fox", "Hare", "Rabbit", "Sabre-toothed tiger",
"T Rex" };
  (void)new wxListBox(panel2, NULL, "List of animals", wxSINGLE,
    5, 5, 170, 80, 5, animals);

  (void)new wxMultiText(panel2, NULL, "Notes",
    "Some notes about the animals in this house", 5, 100, 170, 100);

  view->AddTabWindow(TEST_TAB_DOG, panel2);

  // Don't know why this is necessary under Motif...
#ifdef wx_motif
  dialog->SetSize(dialogWidth, dialogHeight-20);
#endif

  view->SetTabSelection(TEST_TAB_CAT);

  dialog->Show(TRUE);
}
```

5.2. wxTabView overview

Classes: *wxTabView* (page 9), *wxPanelTabView* (page 4)

A wxTabView manages and draws a number of tabs. Because it is separate from the tabbed window implementation, it can be reused in a number of contexts. This library provides tabbed dialog and panel classes to use with the wxPanelTabView class, but an application could derive other kinds of view from wxTabView.

For example, a help application might draw a representation of a book on a canvas, with a row of tabs along the top. The new tab view class might be called wxCanvasTabView, for example, with the wxBookCanvas posting the OnEvent function to the wxCanvasTabView before processing further, application-specific event processing.

A window class designed to work with a view class must call the view's OnEvent and Draw

functions at appropriate times.

6. Change log

June 3rd 1997, Version 1.2

- Fixed bug which drew some tabs incorrectly.
- Altered sample to put buttons below tabs, as per standard Windows conventions.
- Added improvements from Hitachi Europe Limited: draws correctly on Motif and Windows, and tabs are now rounded - much nicer.

April 29th 1996, Version 1.1

- Added SetHorizontalTabOffset, SetHorizontalTabSpacing.
- Corrected bug in colouring tabs (1 pixel out).
- Corrected bug in adding tabs: last tab on first row could overlap right-hand edge.
- Added Layout function to allow resizing of the view rectangle and subsequent redrawing of the tabs.
- Added WXTAB_VERSION symbol.
- Fixed bug in SetTabSelection which did not move the selected tab to the first row.
- Added argument in SetTabSelection to optionally avoid calling activation code.
- Changed wxPanelTabView API to allow use of any window, not just a panel, in a tab.

April 24th 1996, Version 1.0

- First release.

Index