# Contents - SftTabs/DLL Version 2.1



## Introduction


What is SftTabs/DLL
Using SftTabs/DLL
SftTabs/DLL Demo
SftTabs/DLL Wizard Application
Ordering Additional Copies


## Using Resource Editors


Resource Workshop
Borland C++
AppStudio, Visual C++
Dialog Editor (Windows SDK)


## Programming


Using C
Using C++ and the Microsoft Foundation Class library
Using C++ and the ObjectWindows Library
Rebuilding the DLLs


## Reference


Definitions and Structures
Messages and Functions
Tab Control Styles
Notifications
Windows Messages
SoftelGrayDialog Dialog Class


## MFC


CSftTabs Class
CSftTabsDialog Class
CSftTabsPage Class
CSftTabsWindowSheet Class
CSftTabsWindowPage Class
Notifications


## OWL


TSftTabs Class
TSftTabsDialog Class
TSftTabsPage Class
TSftTabsWindowSheet Class
TSftTabsWindowPage Class
Notifications

## Support

Contacting Softel vdm

## Can't load file *filename*.  or  File *filename* does not contain tab definitions.

The SftTabs/DLL Wizard cannot open the specified file.  Make sure the file exists and that the file was created using the SftTabs/DLL Wizard.

## Can't save *filename*.

The SftTabs/DLL Wizard cannot save the specified file. Make sure the target disk/drive is not write protected and that there is sufficient space available.

## File *filename* has been modified. Do you want to save the changes?

The SftTabs/DLL Wizard is about to end. The current tab settings being edited have not yet been saved. Click *Yes* to save the current settings, *No* to skip saving the settings or *Cancel* to prevent the application from ending.

## This is an incorrectly licensed version of SftTabs/DLL. Please contact Softel vdm for a replacement.

The product you have received has been created incorrectly. The licensing information is missing. Please contact Softel vdm for a replacement.

## An unexpected application error has occurred.

The application has encountered a condition which was not anticipated. This could indicate a programming error or an error caused by an unsupported environment. An additional error message may precede this error, which explains the problem in more detail. If you need assistance resolving this problem, please contact Softel vdm.

# SftTabs/DLL Overview

SftTabs/DLL is a custom control for the Windows operating system, offering an alternative method of displaying multiple dialogs or cascading menu selections, by displaying tabs as found in a file cabinet or a notebook.

### Tab Control

SftTabs/DLL offers many features; from a simple, single-row tab control to a multi-row, multi-color tab control in a notebook style.

- SftTabs/DLL Wizard
- Single or multiple tab rows
- Different color tabs
- Icons or bitmaps on each tab
- Enabled/disabled tabs
- Single or multiline tab labels
- Selectable tab label text alignment (left, right, center)
- "Wizard" style dialogs
- Tabbed dialogs and tabbed windows
- Up to 16 rows of tabs
- Up to 128 tabs per tab control
- Multiple tab styles
- Tab colors can change based on tab status
- Fixed or variable width tabs
- Tab control with or without client area
- ToolTips (32-bit applications)
- Scrollable tabs
- User-customizable scroll button bitmaps
- All icons/bitmaps fully customizable
- Reduced menu complexity by using tab controls
- Support for SDK dialog editors, AppStudio, Resource Workshop
- Supports C, C++ with MFC and C++ with OWL
- Compatible with the standard Windows color scheme and the 3D look provided by CTL3DV2.DLL and CTL3D32.DLL

### SftTabs/DLL Wizard Application

The SftTabs/DLL Wizard allows you to design and test a tab control without any programming. Tab labels, the number of tab rows, tab colors, icon and bitmap locations are just a few of the items you can customize. Once you are satisfied with your tab control look, the SftTabs/DLL Wizard can even generate the required run-time code for C, C++ with MFC and C++ with OWL. So your programming effort is kept to a minimum.

### Source Code

The source code for the MFC and OWL C++ classes for tab control access and tabbed dialogs and tabbed windows are supplied. The DLL source code (written in C) is available. Any application that you develop can use SftTabs/DLL royalty-free as long as none of our source code is shipped with your application.

### Languages Supported

SftTabs/DLL supports C, C++ and other languages when using the standard SendMessage Windows API. The DLLs can be called using the definitions provided in the supplied header file. For languages other than C or C++, the user can translate these definitions. In addition, SftTabs/DLL is shipped with class definitions which support the Microsoft Foundation Class Library (MFC) and the Borland ObjectWindows Library (OWL).

### Environments Supported

SftTabs/DLL supports Windows 3.1, Windows NT and Windows 95 using the same easy to use API.  Special UNICODE support is also available when running on Windows NT.

## Using SftTabs/DLL

Depending on the programming language used, the steps necessary to add a tab control to an application differ somewhat, but the following steps outline the basic method:

First, a tab control is added to a dialog using a resource editor. The sections Resource Workshop, Borland C++, AppStudio, Visual C++ and Dialog Editor (Windows SDK) outline the process for each of the supported resource editors. When the dialog is later used in an application, the tab control is automatically created and can be accessed using the supplied API. A tab control can also be created outside of a dialog. This is documented in the language specific programming sections Using C, Using C++ and the Microsoft Foundation Class library and Using C++ and the ObjectWindows Library.

Once the tab control has been created, the API functions documented in section Messages and Functions can be used to add tabs, define attributes, respond to events, etc. The following samples create a very minimal tab control with three tabs as pictured below. This example can easily be extended by adding a few calls to define tab bitmaps or icons and change other tab components to alter the appearance of the tab control.

**Sample Code**

C Sample
C++/MFC Sample
C++/OWL Sample

## Using SftTabs/DLL - C Sample

The following code has been created using the SftTabs/DLL Wizard application.

```
/*- Tab Control Initialization Data ----------------------------------------*/

static const SFTTABS_CONTROL CtlInit = {
    SFTTABSSTYLE_SIMPLE,                /* tab style */
    1,                                  /* number of rows */
    0,                                  /* number of tabs per row (if fFixed) */
    10,                                 /* width of left margin */
    10,                                 /* width of right margin */
    FALSE,                              /* same width for all tabs */
    TRUE,                               /* Client area wanted */
    FALSE,                              /* allow multiline label text */
    TRUE,                               /* use with dialog */
    FALSE,                              /* use specified background color only for
text */
    FALSE,                              /* scrollable tabs */
    FALSE,                              /* hide scroll buttons */
    FALSE,                              /* bold font for active tab wanted */
    FALSE,                              /* fill rows completely */
    NULL,                               /* scroll button bitmap */
    NULL,                               /* Dialog data associated with active tab */
    NULL,                               /* Dialog window handle associated with ...
    NULL,                               /* Frame, used as client area */
    TRUE,                               /* Tooltips wanted */
    FALSE,                              /* drop text if it doesn't fit */
    FALSE,                              /* conditional scroll buttons */
};

static const SFTTABS_TAB Tab0 = {   /*&First */
    SFTTABS_NOCOLOR, SFTTABS_NOCOLOR,   /* background, foreground color */
    SFTTABS_NOCOLOR, SFTTABS_NOCOLOR,   /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },          /* Bitmap, Icon */
    TRUE,                               /* enabled/disabled */
    0,                                  /* userdata */
//  (DWORD) Page_callback               /* user supplied tab callback */
};
static const SFTTABS_TAB Tab1 = {   /*&Second */
    SFTTABS_NOCOLOR, SFTTABS_NOCOLOR,   /* background, foreground color */
    SFTTABS_NOCOLOR, SFTTABS_NOCOLOR,   /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },          /* Bitmap, Icon */
    TRUE,                               /* enabled/disabled */
    0,                                  /* userdata */
//  (DWORD) Page_callback               /* user supplied tab callback */
};
static const SFTTABS_TAB Tab2 = {   /*&Third */
    SFTTABS_NOCOLOR, SFTTABS_NOCOLOR,   /* background, foreground color */
    SFTTABS_NOCOLOR, SFTTABS_NOCOLOR,   /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },          /* Bitmap, Icon */
    TRUE,                               /* enabled/disabled */
    0,                                  /* userdata */
//  (DWORD) Page_callback               /* user supplied tab callback */
};

/* Replace "Page_callback" with the names of the actual        */
/* callback routines once they have been implemented and       */
/* remove the comment marks.                                   */

/*- Initialize Tab Control -------------------------------------------------*/
```

```
/*  Usually added to the WM_INITDIALOG or WM_CREATE message handler of the tab  */
/*  control's parent window (i.e., the tabbed dialog)                           */
/*------------------------------------------------------------------------------*/

    int index;
    HWND hwndTab;

    hwndTab = GetDlgItem(hwndParent, IDC_TAB);
                                        /* get the window handle */

    /* Initialization is faster if we set redraw off */
    SendMessage(hwndTab, WM_SETREDRAW, (WPARAM)FALSE, 0);

    /* We are using new features */
    SftTabs_SetVersion(hwndTab, SFTTABS_2_1);

    index = SftTabs_AddTab(hwndTab, TEXT("&First"));
    SftTabs_SetTabInfo(hwndTab, index, &Tab0);

    index = SftTabs_AddTab(hwndTab, TEXT("&Second"));
    SftTabs_SetTabInfo(hwndTab, index, &Tab1);

    index = SftTabs_AddTab(hwndTab, TEXT("&Third"));
    SftTabs_SetTabInfo(hwndTab, index, &Tab2);

    SftTabs_SetControlInfo(hwndTab, &CtlInit);

    // Make sure to turn redraw back on
    SendMessage(hwndTab, WM_SETREDRAW, (WPARAM)TRUE, 0);
    InvalidateRect(hwndTab, NULL, TRUE);

    // Activate current page.
    SftTabs_ActivatePage(hwndParent, hwndTab, NULL, TRUE);

    // Mark the window as a main, tabbed dialog (so accel. keys work) by registering
it.
    // Register the dialog AFTER activating the current page
    SftTabs_RegisterDialog(hwndParent);

    return FALSE;                       // WM_INITDIALOG, input focus already set
```

## Using SftTabs - C++/MFC Sample

The following code has been created using the SftTabs/DLL Wizard application.

```
/*- Tab Control Initialization Data ----------------------------------------*/

static const SFTTABS_CONTROL CtlInit = {
    SFTTABSSTYLE_SIMPLE,                /* tab style */
    1,                                  /* number of rows */
    0,                                  /* number of tabs per row (if fFixed) */
    10,                                 /* width of left margin */
    10,                                 /* width of right margin */
    FALSE,                              /* same width for all tabs */
    TRUE,                               /* Client area wanted */
    FALSE,                              /* allow multiline label text */
    TRUE,                               /* use with dialog */
    FALSE,                              /* use specified background color only for
text */
    FALSE,                              /* scrollable tabs */
    FALSE,                              /* hide scroll buttons */
    FALSE,                              /* bold font for active tab wanted */
    FALSE,                              /* fill rows completely */
    NULL,                               /* scroll button bitmap */
    NULL,                               /* Dialog data associated with active tab */
    NULL,                               /* Dialog window handle associated with ...
    NULL,                               /* Frame, used as client area */
    TRUE,                               /* Tooltips wanted */
    FALSE,                              /* drop text if it doesn't fit */
    FALSE,                              /* conditional scroll buttons */
};

static const SFTTABS_TAB Tab0 = {   /*&First */
    SFTTABS_NOCOLOR, SFTTABS_NOCOLOR,    /* background, foreground color */
    SFTTABS_NOCOLOR, SFTTABS_NOCOLOR,    /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },           /* Bitmap, Icon */
    TRUE,                                /* enabled/disabled */
    0                                    /* userdata */
};
static const SFTTABS_TAB Tab1 = {   /*&Second */
    SFTTABS_NOCOLOR, SFTTABS_NOCOLOR,    /* background, foreground color */
    SFTTABS_NOCOLOR, SFTTABS_NOCOLOR,    /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },           /* Bitmap, Icon */
    TRUE,                                /* enabled/disabled */
    0                                    /* userdata */
};
static const SFTTABS_TAB Tab2 = {   /*&Third */
    SFTTABS_NOCOLOR, SFTTABS_NOCOLOR,    /* background, foreground color */
    SFTTABS_NOCOLOR, SFTTABS_NOCOLOR,    /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },           /* Bitmap, Icon */
    TRUE,                                /* enabled/disabled */
    0                                    /* userdata */
};

/*---------------------------------------------------------------------------*/
/* This sample code can be used to initialize the tab control.               */
/* This code is usually used in an OnInitDialog (WM_INITDIALOG), OnCreate    */
/* (WM_CREATE) or OnInitialUpdate member function of the tab control's parent*/
/* dialog or window.                                                         */
/*---------------------------------------------------------------------------*/

    int index;
```

```
    /* Associate the tab control created from the dialog       */
    /* resource with the C++ object.                           */
    m_Tab.SubclassDlgItem(IDC_TAB, this /* parent window */);

    /* You could use DDX/DDV instead and add the following     */
    /* line to the DoDataExchange function of the tab          */
    /* control's parent window (remove the //).                */
    // DDX_Control(pDX, IDC_TAB, m_Tab);

    /* Initialization is faster if we set redraw off */
    m_Tab.SetRedraw(FALSE);

    /* We are using new features */
    m_Tab.SetVersion(SFTTABS_2_1);

    index = m_Tab.AddTab(_T("&First"));
    m_Tab.SetTabInfo(index, &Tab0);
    // If you don't want to attach a page to the tab, the following is optional
//  m_Tab.SetTabDialog(index, new an_object_based_on_CSftTabsPage(this)); // tab page

    index = m_Tab.AddTab(_T("&Second"));
    m_Tab.SetTabInfo(index, &Tab1);
    // If you don't want to attach a page to the tab, the following is optional
//  m_Tab.SetTabDialog(index, new an_object_based_on_CSftTabsPage(this)); // tab page

    index = m_Tab.AddTab(_T("&Third"));
    m_Tab.SetTabInfo(index, &Tab2);
    // If you don't want to attach a page to the tab, the following is optional
//  m_Tab.SetTabDialog(index, new an_object_based_on_CSftTabsPage(this)); // tab page

    m_Tab.SetControlInfo(&CtlInit);

    // Make sure to turn redraw back on
    m_Tab.SetRedraw(TRUE);
    m_Tab.InvalidateRect(NULL, TRUE);

    // If you are not using the sheet/page classes, remove ...
    // Initialize tab control
    InitializeTabControl(0, &m_Tab, NULL);
    // return FALSE; if this is a dialog's OnInitDialog member function
```

## Using SftTabs - C++/OWL Sample

The following code has been created using the SftTabs/DLL Wizard application.

```
/*- Tab Control Initialization Data ----------------------------------------*/

static const SFTTABS_CONTROL CtlInit = {
    SFTTABSSTYLE_SIMPLE,                /* tab style */
    1,                                  /* number of rows */
    0,                                  /* number of tabs per row (if fFixed) */
    10,                                 /* width of left margin */
    10,                                 /* width of right margin */
    FALSE,                              /* same width for all tabs */
    TRUE,                               /* Client area wanted */
    FALSE,                              /* allow multiline label text */
    TRUE,                               /* use with dialog */
    FALSE,                              /* use specified background color only for
text */
    FALSE,                              /* scrollable tabs */
    FALSE,                              /* hide scroll buttons */
    FALSE,                              /* bold font for active tab wanted */
    FALSE,                              /* fill rows completely */
    NULL,                               /* scroll button bitmap */
    NULL,                               /* Dialog data associated with active tab */
    NULL,                               /* Dialog window handle associated ...
    NULL,                               /* Frame, used as client area */
    TRUE,                               /* Tooltips wanted */
    FALSE,                              /* drop text if it doesn't fit */
    FALSE,                              /* conditional scroll buttons */
};

static const SFTTABS_TAB Tab0 = {   /*&First */
    SFTTABS_NOCOLOR, SFTTABS_NOCOLOR,    /* background, foreground color */
    SFTTABS_NOCOLOR, SFTTABS_NOCOLOR,    /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },           /* Bitmap, Icon */
    TRUE,                                /* enabled/disabled */
    0                                    /* userdata */
};
static const SFTTABS_TAB Tab1 = {   /*&Second */
    SFTTABS_NOCOLOR, SFTTABS_NOCOLOR,    /* background, foreground color */
    SFTTABS_NOCOLOR, SFTTABS_NOCOLOR,    /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },           /* Bitmap, Icon */
    TRUE,                                /* enabled/disabled */
    0                                    /* userdata */
};
static const SFTTABS_TAB Tab2 = {   /*&Third */
    SFTTABS_NOCOLOR, SFTTABS_NOCOLOR,    /* background, foreground color */
    SFTTABS_NOCOLOR, SFTTABS_NOCOLOR,    /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },           /* Bitmap, Icon */
    TRUE,                                /* enabled/disabled */
    0                                    /* userdata */
};

/*---------------------------------------------------------------------------*/
/* This sample code can be used to initialize the tab control.               */
/* This code is usually used in an EvInitDialog (WM_INITDIALOG) or EvCreate  */
/* member function of the tab control's parent dialog or window.             */
/*---------------------------------------------------------------------------*/

    int index;
```

```
    /* Initialization is faster if we set redraw off */
    pTab->SetRedraw(false);

    /* We are using new features */
    pTab->SetVersion(SFTTABS_2_1);

    index = pTab->AddTab(TEXT("&First"));
    pTab->SetTabInfo(index, &Tab0);
    // If you don't want to attach a page to the tab, the following is optional
//  pTab->SetTabDialog(index, new an_object_based_on_TSftTabsPage(this)); // tab page

    index = pTab->AddTab(TEXT("&Second"));
    pTab->SetTabInfo(index, &Tab1);
    // If you don't want to attach a page to the tab, the following is optional
//  pTab->SetTabDialog(index, new an_object_based_on_TSftTabsPage(this)); // tab page

    index = pTab->AddTab(TEXT("&Third"));
    pTab->SetTabInfo(index, &Tab2);
    // If you don't want to attach a page to the tab, the following is optional
//  pTab->SetTabDialog(index, new an_object_based_on_TSftTabsPage(this)); // tab page

    pTab->SetControlInfo(&CtlInit);

    // Make sure to turn redraw back on
    pTab->SetRedraw(true);
    pTab->Invalidate(true);

    // If you are not using the sheet/page classes, remove ...
    // Initialize tab control
    InitializeTabControl(0, pTab, NULL);
    // return false; if this is a dialog's OnInitDialog member function
```

## Order Form

When ordering by mail or fax, please use this order form. Print this help topic using the *File, Print Topic* menu command.


**Call:** (201) 366-9618 **Mail to:** Softel vdm
**or** 11 Michigan Ave
**FAX:** (201) 366-3984 Wharton, NJ 07885


Name .....

Company .....

Street .....

City, State, ZIP .....

Country .....

Payment Method [ ] Visa [ ] Mastercard [ ] American Express
[ ] Check enclosed

Card Number .....

Expiration Date .....

Signature .....

Phone Number .....

FAX Number .....

Email Address .....

Please include your phone number so we can contact you if there is a problem filling your order.

Site licensing available - please call for more information.

Prices as of October 10, 1996. Subject to change.

| | | |
|---|---|---|
| ..... | SftTabs/DLL 2.1, 16- & 32-bit, without DLL source code<br>$389 each Single developer license | $ ..... |
| ..... | SftTabs/DLL 2.1, 16- & 32-bit, including DLL source code<br>$578 each Single developer license | $ ..... |
| | 6% Sales Tax (NJ residents only) | $ ..... |
| | Shipping & Handling ($10.00 per copy, $15.00 international) | $ ..... |
| | Total | $ ..... |

## Tab Bitmap

The tab bitmap is an optional bitmap, displayed on a tab next to the tab text. The exact location of the tab bitmap depends on the selected picture location (defined using the SftTabs/DLL Wizard). If a tab bitmap is used, a tab icon cannot be displayed at the same time.

## Tab Icon

The tab icon is an optional icon, displayed on a tab next to the tab text. The exact location of the tab icon depends on the selected picture location (defined using the SftTabs/DLL Wizard). If a tab icon is used, a tab bitmap cannot be displayed at the same time.

## Tab Picture

The tab picture is the term used to refer to a tab's icon or a tab's bitmap.

## Tab Text

The text portion of the tab label.

## Tab Label

The tab label is the term used to refer to the collection of the tab picture and the tab text, the entire visual representation of a tab.

## Tab Rows

A tab control can have up to 16 rows of tabs.  The number of rows is defined using the SftTabs/DLL Wizard.

## Left/Right Margin

A tab control can have a left margin, which is the area between the left side of the tab control and the first tab. Setting the left margin to 0 positions the first tab at the left edge of the tab control. Some tab styles have a built-in left margin, so tabs cannot start at the left edge of the control. The SftTabs/DLL Wizard can be used to determine which tab styles support tabs at the left edge of the control. If tab rows are displayed vertically, the left margin defined is used as the top margin.

## Client Area

A tab control can optionally contain a client area.  This area is typically used to display dialogs or Windows controls, which are associated with a given tab.

## Scroll Button

A scrollable tab control offers scroll buttons. These scroll buttons are used by the user to make currently obscured tabs visible.

## Page

Each tab can have a page attached to it. A page is a window (or dialog) which is normally displayed in the tab control's client area.

## Active Tab

The active tab is the tab that is currently the selected tab, usually in the front row. The tab text also receives a focus rectangle when the tab control has the input focus.

## Mouse and Keyboard Interface

Tabs in a tab control can be activated using the left mouse button. By clicking on a currently inactive tab, that tab will become the active tab. This takes place under program control. The application receives a SFTTABSN_SWITCHING and SFTTABSN_SWITCHED WM_COMMAND notification.

Tabs can also be made active using the keyboard. Just as regular Windows controls, such as buttons respond to an Alt + *key* combination, the tab control responds to keyboard accelerators, if any one of the tab labels has been defined to support this. When defining tab text, the & character indicates that the following character is to be recognized by an Alt + *key* combination, e.g., the tab with the tab label "&First" will be made active when the user presses Alt-F. The & character will never be shown, the following character will be underlined instead. Switching tabs using Alt-*key* combinations is always supported when the tab control has the input focus. For cases where other windows have the input focus, the enclosing window or dialog has to be registered for keyboard accelerators to be recognized by the tab control (see SftTabs_RegisterDialog or SftTabs_RegisterWindow for more information).

Control + Tab key combinations and Control + Shift + Tab key combinations are used to switch to the next (or previous) tab. That tab will become the new active tab.

Control + *arrow key* key combinations can be used to scroll in a scrollable tab control. For Control + *arrow key* key combinations to have an effect, the tab control must be the active control, i.e., the control which currently has the input focus.

## "Wizard" Style Dialogs

With SftTabs/DLL it is very easy to implement wizard-style dialogs. SftTabs/DLL offers a special tab control style that does not display any tabs, but still manages all required attached pages. Because the tab control doesn't offer any tabs, an external event such as a button click has to be used to change the tab control's current tab (using SetCurrentTab). The application has to explicitly control the current tab.

During development of an application, it may be easier to use a regular tab style, so the programmer can test the pages simply by clicking on a tab. Once the pages have been debugged and tested, the tab control style can be changed to a wizard-style tab control.

# Demo Application


Click here to run the Demo application. If the application cannot be started, please reinstall SftTabs/DLL from the installation disks.

## Overview

During the installation of SftTabs/DLL, an icon for the demo application "DLL Demo" is installed in the Program Manager group *SftTabs 2.1*.

The source code for the demo application DEMO32.EXE is included with SftTabs/DLL. The source code can be found in the directory C:\SFTTABS\SAMPLES\C\DEMO.

# Product Support

### Before Contacting Product Support

A comprehensive user guide and on-line help can assist you in using SftTabs/DLL. If you have difficulties using SftTabs/DLL, please use the following sources of information:

- Obtain help using the on-line help files provided
- Review this user's guide

If this does not resolve your problem, please contact Softel vdm Product Support.

### Contacting Product Support

If you have reviewed the on-line help and your manual, please contact Softel vdm Product Support using any of the following methods:

| | |
|---|---|
| **Telephone** | (201) 366-9618<br>Make sure you have your license number ready when calling. |
| **Fax** | (201) 366-3984 |
| **WWW** | http://www.softelvdm.com<br>Download up-to-date bug descriptions, solutions, samples |
| **Internet** | support@softelvdm.com |
| **Mail** | Softel vdm<br>11 Michigan Ave<br>Wharton, NJ 07885-2540 |

Please include your license number in all cases. Without your license number, we will not be able to help you. Your license number is printed on your installation diskette label or can be found by using the *About SftTabs/DLL* menu command of the SftTabs/DLL Wizard application.

# SftTabs/DLL Wizard

Click here to run the SftTabs/DLL Wizard application. If the application cannot be started, please reinstall SftTabs/DLL from the installation disks.

## Overview

During the installation of SftTabs/DLL, an icon for the application "SftTabs/DLL Wizard" is installed in the program group *SftTabs 2.1*.

The SftTabs/DLL Wizard is a tool to assist the developer in creating a tab control layout and in populating the tab control with suitable tab labels. The SftTabs/DLL Wizard can generate the necessary run-time source code in C, C++ for MFC and C++ for OWL, which can then be copied into an application. The source code does require minor modifications once it has been copied into an application. A tab layout can be saved in tab files (*.TAB), which can be further edited at a later time using the SftTabs/DLL Wizard.

The SftTabs/DLL Wizard has access to all supported SftTabs/DLL styles and tab control features.

The SftTabs/DLL Wizard's main window can be used to select a tab style, change tab control attributes, add or delete tabs, set tab attributes and much more. Any changes made to a tab layout are immediately reflected in the Sample window.

The Sample window can be resized to adjust the size of the control. If a tab control doesn't provide a client area, the sample window cannot be resized. When switching between tab styles or changing tab control attributes, the SftTabs/DLL Wizard automatically resizes the sample window if necessary.

The SftTabs/DLL Wizard can be used to generate the source code for the current tab control attributes, by using the C, C++/MFC and C++/OWL tabs.

The SftTabs/DLL Wizard can also provide an immediate view of the main tab control structure SFTTABS_CONTROL. This control information is accessible by clicking on the *Control* button of the main window. All members of the SFTTABS_CONTROL structure are shown and are immediately updated when changes are made to the tab control.

When a new tab is created (using the *File, New* menu command or by clicking on the *Tab Gallery...* button), the SftTabs/DLL Style Gallery dialog is presented. This dialog offers a variety of tab styles to chose from as a basis for the tab control to be created. The developer can then refine the control attributes, add or delete tabs, change tab labels, etc.

## Rebuilding the DLLs

**Warning:** If you rebuild the DLL(s) and have made changes to the source code which could potentially make the resulting DLL(s) incompatible with the DLL(s) as supplied with SftTabs/DLL, you must rename the DLL(s). Make sure to also update the LIBRARY statement in the module definition file(s) to reflect the new DLL name(s).

**Note:** If you need to modify the SftTabs/DLL source code, please make sure to test the resulting DLL with the SftTabs/DLL Wizard.

If you wish to rebuild the DLL, please follow these steps. Use your development environment to create a new project and set desired project options. Make sure the target is a DLL (as opposed to an EXE). The source files for the DLL can be found in the directory C:\SFTTABS\SOURCE (unless changed during the installation).

The following files have to be added to your project:

| | ***DLL for Windows 3.1*** | ***DLL for Windows NT (with UNICODE support)*** | ***DLL for Windows 95, Win32s , Windows NT (incl. Windows NT without UNICODE support)*** |
|---|---|---|---|
| ***Target*** | SFTTB.DLL | SFTTB32**U**.DLL | SFTTB32.DLL |
| ***Required Source Files*** | BCT1TABS.C<br>BMBUTTON.C<br>HELPER.C<br>MCT1TABS.C<br><br>SFTTABS.C<br>STYLE*.C<br>(multiple)<br>TABSINIT.C<br>SFTTABS.RC<br>SFTTABS.DEF | <br>BMBUTTON.C<br>HELPER.C<br><br>MCT2TABS.C<br>SFTTABS.C<br>STYLE*.C<br>(multiple)<br>TABSINIT.C<br>SFTTABS.RC<br>SFTTB32**U**.DEF | <br>BMBUTTON.C<br>HELPER.C<br><br>MCT2TABS.C<br>SFTTABS.C<br>STYLE*.C<br>(multiple)<br>TABSINIT.C<br>SFTTABS.RC<br>SFTTB32.DEF |

**Note:** If you do not include a DEF file above, your DLL may be built correctly, but applications will fail to load or execute properly.

### Special Considerations

By defining the _DEBUG preprocessor symbol, tracing options are enabled for the DLL. For certain error conditions the SftTabs/DLL will send messages to a debugging terminal or the debugger using the OutputDebugString Windows API function. For more information, see the Windows OutputDebugString documentation. The DLLs shipped with SftTabs/DLL do not have this tracing facility enabled.

### Special Considerations for Windows 3.1

When rebuilding the Windows 3.1 version, choose the LARGE memory model.

When creating a debugging version for Windows 3.1, the project has to linked with TOOLHELP.LIB and the TOOLHELP.DLL has to be available at run-time.

### Special Considerations for Windows NT

To rebuild the UNICODE version of SftTabs/DLL (Windows NT only), make sure to define the

following preprocessor symbols:

UNICODE
_UNICODE

If these symbols are not defined, the resulting DLL will not support UNICODE. The DLL supporting UNICODE is named SFTTB32U.DLL, the non-UNICODE DLL is named SFTTB32.DLL.

## Special Considerations using Borland C++ 32-bit compiler

When creating a DLL, a LIB file is automatically created or can be created using the IMPLIB utility. The LIB files created by the Borland 32-bit compiler are incompatible with the LIB files created by the Microsoft compiler. For this reason, the LIB file created when using Borland C++ should be renamed according to the following table. The DLLs created with Borland C++ and Microsoft Visual C++ are interchangeable, however, the LIB files are not.

| ***Target*** | ***DLL for Windows 3.1*** | ***DLL for Windows NT (with UNICODE support)*** | ***DLL for WIN32/s/c (incl. Windows NT without UNICODE support)*** |
|---|---|---|---|
| ***LIB file*** | SFTTB.LIB | SFTTB32V.LIB | SFTTB32B.LIB |

# C Programming

This section describes how to use SftTabs/DLL with an application written using the C programming language.

## Building an Application

A) Every source program making use of a SftTabs/DLL control must include the required header file SFTTB.H by using the #include directive.

```
#include "sfttb.h"                /* SftTabs/DLL required header file */
```

This include statement should appear after the #include <windows.h> statement. The file is located in the directory C:\SFTTABS\INCLUDE (unless changed during the installation).

B) In order to use SftTabs/DLL controls, an application must call the SftTabs_RegisterApp function. The call to this function is required so that SftTabs/DLL window classes can be registered. This call has to be made before any SftTabs/DLL controls are created. Add the following statement to your source code, where your application registers its window classes (normally during application initialization):

```
SftTabs_RegisterApp(hInstance);    /* Use SftTabs/DLL with this application */
```

C) Once SftTabs/DLL controls are no longer needed, an application must call the SftTabs_UnregisterApp function. The call to this function is required so that SftTabs/DLL window classes can be unregistered and cleanup processing can take place. This call has to be made after all SftTabs/DLL controls have been destroyed (normally during application termination).

```
SftTabs_UnregisterApp(hInstance);  /* No longer use SftTabs/DLL */
```

D) The application's executable (EXE or DLL) must be linked with the correct LIB file, depending on the target environment and the compiler used. The DLL must be available and accessible at run-time for proper execution. The DLL used at run-time depends on the LIB file used at link time.

| Target Environment | LIB File for Applications developed using MS C or Visual C++ | LIB File for Applications developed using Borland C++ | DLL File Required at Run-Time |
|---|---|---|---|
| **Windows 3.1 16-bit applications** | SFTTB.LIB | SFTTB.LIB | SFTTB.DLL |
| **WIN32, all 32-bit environments including Windows NT (without UNICODE support)** | SFTTB32.LIB | SFTTB32B.LIB | SFTTB32.DLL |
| **Windows NT only (with UNICODE support** | SFTTB32U.LIB | SFTTB32V.LIB | SFTTB32U.DLL |

All required files can be found in the directory C:\SFTTABS\LIB and C:\SFTTABS\BIN, unless changed during the installation.

## Adding a Tab Control

There are two methods to add a tab control to an application:

- using dialog resources

• using CreateWindow(Ex)

Adding a tab control using dialog resources is accomplished by using a resource editor to design a dialog. Once a tab control is created, its window handle can be obtained by using the Windows GetDlgItem function. For more information on the different resource editors supported by SftTabs/DLL, see Resource Workshop, Borland C++, AppStudio, Visual C++ and Dialog Editor (Windows SDK).

Another method to create a tab control is by using the CreateWindow(Ex) Windows calls.

```
hwndTab = CreateWindow(SFTTABS_CLASS, "",          // Window class and caption
                       style,  x, y, cx, cy,       // location
                       hwndParent,                 // parent window
                       IDC_TABS,                   // tab control ID
                       hInst,                      // application instance
                       NULL);
```

For more information on the various parameters used, see the Windows API documentation. The tab control class is defined by the SFTTABS_CLASS constant (SFTTB.H). The window class is **SoftelTabControl** (Windows 3.1) or **SoftelTabControl32** (for Windows NT, 95, Win32s).

## Handling Notifications

As with standard Windows controls, applications must respond to events and messages to cause controls to respond to user requests. For additional information see Notifications.

### Switching Between Tabs

Switching between tabs is fully automatic, however, an application may wish to prevent a user from switching to another tab. By responding to the WM_COMMAND, SFTTABSN_SWITCHING notification, an application can prevent completion of the tab switch by sending a WM_CANCELMODE message to the tab control.

```
    case WM_COMMAND: {
        // Parameter packing differs between 16-bit and 32-bit Windows
#if defined(_WIN32) || defined(__WIN32__)
        HWND hwndCtl = (HWND) lParam;
        int id = LOWORD(wParam);
        int code = HIWORD(wParam);
#else
        HWND hwndCtl = (HWND) LOWORD(lParam);
        int id = (int) wParam;
        int code = HIWORD(lParam);
#endif
        if (hwndCtl) {
            switch (id) {
            case IDC_TAB:
                switch (code) {
                case SFTTABSN_SWITCHING:// we're about to switch away from
                    // the current page.  If you need to know what the new
                    // page will be use SftTabs_GetNextTab(hwndCtl).
                    if (!IsOKToSwitch())
                        SendMessage(hwndCtl, WM_CANCELMODE, 0, 0);
                    break;
                case SFTTABSN_SWITCHED:// we switched to a new page
                    SftTabs_ActivatePage(hwndParent, hwndCtl, NULL, FALSE);
                    break;
                }
                break;

            case IDOK:
            case IDCANCEL:
                if (code == BN_CLICKED)
                    SendMessage(hwndParent, WM_COMMAND, id, 0);
```

```
                break;
            }
        }
        break;
    }
```

An application has to make attached controls or dialogs visible when switching between tabs. The SftTabs/DLL API offers functions to manage dialogs and Windows controls that are attached to tabs. See Implementing Tabbed Dialogs and Implementing Tabbed Windows for more information.

## 3D and Colors

SftTabs/DLL offers many tab control styles which look best on a gray background. For dialogs, the gray background can be achieved using the SoftelGrayDialog or SoftelGrayDialog32 window class or using one of the following methods:

### CTL3DV2 or CTL3D32

The tab control can be used with CTL3DV2 (or CTL3D32). Any dialogs attached to a tab control can use the 3D display, if properly enabled (usually using Ctl3dAutoSubclass). For more information on CTL3DV2 or CTL3D32, see the Microsoft documentation.

### WM_CTLCOLOR, WM_CTLCOLORSTATIC

This message is generated by the tab control for compatibility with SftTabs 2.0 only. When developing new applications, please use SftTabs_SetCtlColors instead.

The background color of the tab control can be modified by handling the WM_CTLCOLOR message. The parent window can override the default window background color used by the tab control by handling the WM_CTLCOLOR message.

```
case WM_CTLCOLOR:
    if (HIWORD(lParam)== CTLCOLOR_STATIC) {
        SetBkColor(hdc, (HDC) wParam, RGB(192,192,192)); // on gray background
        return GetStockObject(LTGRAY_BRUSH);
    }
```

## Implementing Tabbed Dialogs

A tabbed dialog is created just like any other dialog. A tabbed dialog has a tab control with an available client area. In this client area, pages are displayed. Each tab has an attached page (although during development of a tabbed dialog, a tab doesn't require an attached page). As the user switches between tabs, the appropriate page is created, displayed and destroyed.

A tabbed dialog and each page have a dialog procedure. This makes conversion of existing dialogs and development of new pages very easy. Tabbed dialogs and pages are first designed using a resource editor. The sections Resource Workshop, Borland C++, AppStudio, Visual C++ and Dialog Editor (Windows SDK) describe how the necessary tabbed dialog and page (dialog) resources are created.

Once the necessary dialogs have been designed, the tab control layout can be defined using the SftTabs/DLL Wizard. The SftTabs/DLL Wizard also creates much of the code required to initialize the tab control. This code should be copied to the application (with possibly minor modifications).

### Creating a Tabbed Dialog Dialog Procedure

The following code sample (from C:\SFTTABS\SAMPLES\CSAM1\CSAM1.C) shows a typical dialog procedure used for a tabbed dialog. Most of the code has been created using the SftTabs/DLL Wizard and then copied into the application.

```
/*****************************************************************/
/*                          Frame Dialog Proc                    */
```

```
/*********************************************************************/

/*- Tab Control Initialization Data ----------------------------------------*/

static const SFTTABS_CONTROL CtlInit = {
    SFTTABSSTYLE_MODERN_I,              /* tab style */
    2,                                  /* number of rows */
    0,                                  /* number of tabs per row (if fFixed) */
    0,                                  /* width of left margin */
    0,                                  /* width of right margin */
    FALSE,                              /* same width for all tabs */
    TRUE,                               /* Client area wanted */
    FALSE,                              /* allow multiline label text */
    TRUE,                               /* use with dialog */
    FALSE,                              /* use specified background color only for
text */
    FALSE,                              /* scrollable tabs */
    FALSE,                              /* hide scroll buttons */
    TRUE,                               /* bold font for active tab wanted */
    TRUE,                               /* fill rows completely */
    NULL,                               /* scroll button bitmap */
    NULL,                               /* Dialog data associated with active tab */
    NULL,                               /* Dialog window handle associated with ...
    NULL,                               /* Frame, used as client area */
    TRUE,                               /* Tooltips wanted */
    FALSE,                              /* drop text if it doesn't fit */
    FALSE,                              /* conditional scroll buttons */
};

static const SFTTABS_TAB Tab0 = {   /*The First One */
    SFTTABS_NOCOLOR, RGB(0,0,255),      /* background, foreground color */
    SFTTABS_NOCOLOR, RGB(0,0,255),      /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_LEFT, SFTTABS_GRAPH_BITMAP },
                                        /* Bitmap, Icon */
    TRUE,                               /* enabled/disabled */
    0,                                  /* userdata */
    (DWORD) Page1_Callback              /* user supplied tab callback */
};

static const SFTTABS_TAB Tab1 = {   /*&Second */
    SFTTABS_NOCOLOR, RGB(255,0,0),      /* background, foreground color */
    SFTTABS_NOCOLOR, RGB(255,0,0),      /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_RIGHT, SFTTABS_GRAPH_ICON },
                                        /* Bitmap, Icon */
    TRUE,                               /* enabled/disabled */
    0,                                  /* userdata */
    (DWORD) Page2_Callback              /* user supplied tab callback */
};

static const SFTTABS_TAB Tab2 = {   /*&Third */
    SFTTABS_NOCOLOR, RGB(128,128,0),    /* background, foreground color */
    SFTTABS_NOCOLOR, RGB(128,128,0),    /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },          /* Bitmap, Icon */
    TRUE,                               /* enabled/disabled */
    0,                                  /* userdata */
    (DWORD) Page3_Callback              /* user supplied tab callback */
};

static const SFTTABS_TAB Tab3 = {   /*F&ourth */
    SFTTABS_NOCOLOR, RGB(0,255,255),    /* background, foreground color */
    SFTTABS_NOCOLOR, RGB(0,255,255),    /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },          /* Bitmap, Icon */
```

```
    TRUE,                               /* enabled/disabled */
    0,                                  /* userdata */
    (DWORD) Page4_Callback              /* user supplied tab callback */
};

static const SFTTABS_TAB Tab4 = {   /*F&ifth */
    SFTTABS_NOCOLOR, RGB(0,0,128),      /* background, foreground color */
    SFTTABS_NOCOLOR, RGB(0,0,128),      /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },          /* Bitmap, Icon */
    TRUE,                               /* enabled/disabled */
    0,                                  /* userdata */
    (DWORD) Page5_Callback              /* user supplied tab callback */
};

static const SFTTABS_TAB Tab5 = {   /*Si&xth */
    SFTTABS_NOCOLOR, RGB(128,0,0),      /* background, foreground color */
    SFTTABS_NOCOLOR, RGB(128,0,0),      /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },          /* Bitmap, Icon */
    TRUE,                               /* enabled/disabled */
    0,                                  /* userdata */
    (DWORD) Page6_Callback              /* user supplied tab callback */
};
```

The dialog procedure shown here initializes the tab control in its WM_INITDIALOG message handler.

```
BOOL _export CALLBACK MainDialogProc(HWND hwndDlg, UINT msg, WPARAM wParam, LPARAM
lParam)
{
    switch (msg) {

    case WM_INITDIALOG: {
        int index;
        HWND hwndTab;
        SFTTABS_TAB Tab;

        // Center this dialog
        CenterWindow(hwndDlg);

        hwndTab = GetDlgItem(hwndDlg, IDC_TAB);
                                        /* get the window handle */
        /* load the bitmaps/icons */
        m_hSampleBitmap = LoadBitmap(g_hInst, MAKEINTRESOURCE(IDB_BITMAP));
        m_hSampleIcon = LoadIcon(g_hInst, MAKEINTRESOURCE(IDI_ICON));

        /* Initialization is faster if we set redraw off */
        SendMessage(hwndTab, WM_SETREDRAW, (WPARAM)FALSE, 0);

        /* We are using new features */
        SftTabs_SetVersion(hwndTab, SFTTABS_2_1);

        index = SftTabs_AddTab(hwndTab, TEXT("The First One"));
        SftTabs_SetToolTip(hwndTab, index, TEXT("Demonstrates tabbing into and ...
        Tab = Tab0;
        Tab.graph.item.hBitmap = m_hSampleBitmap;
        SftTabs_SetTabInfo(hwndTab, index, &Tab);

        index = SftTabs_AddTab(hwndTab, TEXT("&Second"));
        SftTabs_SetToolTip(hwndTab, index, TEXT("Demonstrates how an ...
        Tab = Tab1;
        Tab.graph.item.hIcon = m_hSampleIcon;
        SftTabs_SetTabInfo(hwndTab, index, &Tab);

        index = SftTabs_AddTab(hwndTab, TEXT("&Third"));
        SftTabs_SetToolTip(hwndTab, index, TEXT("This page is reset everytime ...
        SftTabs_SetTabInfo(hwndTab, index, &Tab2);
```

```
        index = SftTabs_AddTab(hwndTab, TEXT("F&ourth"));
        SftTabs_SetToolTip(hwndTab, index, TEXT("A page with private OK, ...
        SftTabs_SetTabInfo(hwndTab, index, &Tab3);

        index = SftTabs_AddTab(hwndTab, TEXT("F&ifth"));
        SftTabs_SetToolTip(hwndTab, index, TEXT("A page that has not ...
        SftTabs_SetTabInfo(hwndTab, index, &Tab4);

        index = SftTabs_AddTab(hwndTab, TEXT("Si&xth"));
        SftTabs_SetToolTip(hwndTab, index, TEXT("A page with nested tab ...
        SftTabs_SetTabInfo(hwndTab, index, &Tab5);

        SftTabs_SetControlInfo(hwndTab, &CtlInit);

        // Make sure to turn redraw back on
        SendMessage(hwndTab, WM_SETREDRAW, (WPARAM)TRUE, 0);
        InvalidateRect(hwndTab, NULL, TRUE);

        // tab 0 is automatically made the current tab by SftTabs_ActivatePage
        // Make a different tab active before calling SftTabs_ActivatePage
//      SftTabs_SetCurrentTab(hwndTab, 1);

        // Activate current page
        SftTabs_ActivatePage(hwndDlg, hwndTab, NULL, TRUE);

        // Mark the window as a main, tabbed dialog (so accel. keys work) by ...
        // Register the dialog AFTER activating the current page
        SftTabs_RegisterDialog(hwndDlg);

        return FALSE;                       // WM_INITDIALOG, input focus already set
    }
```

The call to SftTabs_ActivatePage makes the current page active. The SftTabs_RegisterDialog call registers the dialog for special tabbed dialog handling by the DLL. After registering the dialog, TAB and ESCAPE key handling, default button setting and keyboard accelerator keys (Alt-*x*) will be performed by SftTabs/DLL.

A tabbed dialog should always return FALSE when handling the WM_INITDIALOG message. The focus has been set already by SftTabs/DLL, so returning FALSE will prevent Windows from setting the focus (to the wrong control).

The following WM_DESTROY message handler shows the required cleanup calls to delete all bitmaps that the application has loaded, and also unregisters the dialog from the DLL.

```
    case WM_DESTROY:
        /* delete the bitmaps/icons */
        DeleteObject(m_hSampleBitmap);

        // Unregister, or the window properties used won't be removed
        SftTabs_UnregisterDialog(hwndDlg);

        // destroy all pages
        SftTabs_Destroy(hwndDlg, GetDlgItem(hwndDlg, IDC_TAB));
        break;
    }
```

The following WM_COMMAND message handler responds to notifications sent to the tabbed dialog by the tab control. When the user attempts to switch to another page by clicking a tab using the left mouse button or by using the keyboard, a SFTTABSN_SWITCHING notification is sent to the tabbed dialog. The dialog then calls SftTabs_DeactivatePage to test if the active page can be left. This call to SftTabs_DeactivatePage results in a WM_QUERYENDSESSION message being sent to the page's dialog procedure. Based on the dialog procedure's response, the page may or may not be deactivated. By handling the WM_QUERYENDSESSION message, a page can prevent a user from switching away from a tab.

If a new tab has become active, the tabbed dialog receives a SFTTABSN_SWITCHED

notification, so the associated page can be made active by the call to SftTabs_ActivatePage.

```
    case WM_COMMAND: {
        // Parameter packing differs between 16-bit and 32-bit Windows
#if defined(_WIN32) || defined(__WIN32__)
        HWND hwndCtl = (HWND) lParam;
        int id = LOWORD(wParam);
        int code = HIWORD(wParam);
#else
        HWND hwndCtl = (HWND) LOWORD(lParam);
        int id = (int) wParam;
        int code = HIWORD(lParam);
#endif

        if (hwndCtl) {
            switch (id) {
            case IDC_TAB:
                switch (code) {
                case SFTTABSN_SWITCHING:// we're about to switch away from
                    // the current page.  If you need to know what the new
                    // page will be use SftTabs_GetNextTab(hwndCtl).
                    if (!SftTabs_DeactivatePage(hwndDlg, hwndCtl))
                        // couldn't deactivate current page, so don't switch
                        SendMessage(hwndCtl, WM_CANCELMODE, 0, 0);
                    break;
                case SFTTABSN_SWITCHED:// we switched to a new page
                    SftTabs_ActivatePage(hwndDlg, hwndCtl, NULL, FALSE);
                    break;
                }
                break;
            case IDOK:
            case IDCANCEL:
                if (code == BN_CLICKED)
                    SendMessage(hwndDlg, WM_COMMAND, id, 0);
                break;
            }

        } else {
            switch (id) {
            case IDOK:
                // The currently active page will be called with a
                // WM_QUERYENDSESSION message to determined if it can be closed
                if (SftTabs_ClosePossible(hwndDlg, GetDlgItem(hwndDlg, IDC_TAB)))
                    EndDialog(hwndDlg, TRUE);
                break;
            case IDCANCEL:
                EndDialog(hwndDlg, FALSE);
                break;
            }
            // The above assumes that this is a modal dialog. If it is a modeless
            // don't use EndDialog, use DestroyWindow instead.
        }
        break;
     }
    }
```

At the end of the dialog procedure, a call to SftTabs_HandleDialogMessage should be made to allow SftTabs/DLL to process any messages.  Without this call, SftTabs/DLL may not be able to provide the tabbed dialog handling, such as TAB and ESCAPE key processing, etc.

```
    if (SftTabs_HandleDialogMessage(hwndDlg, msg, wParam, lParam))
        return TRUE;

    return FALSE;
}
```

The SFTTABS_TAB structures used to define each tab in the tab control, also define a callback routine of type SFTTABS_TABCALLBACK. This callback routine is responsible for creating and destroying the dialog which represents the page. SftTabs/DLL will hide and disable a dialog, but it is up to this callback routine to create and destroy the dialog. The following tab definition is repeated here so it can be shown next to the callback function:

```
static const SFTTABS_TAB Tab2 = {   /*&Third */
    SFTTABS_NOCOLOR, RGB(128,128,0),     /* background, foreground color */
    SFTTABS_NOCOLOR, RGB(128,128,0),     /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },           /* Bitmap, Icon */
    TRUE,                                /* enabled/disabled */
    0,                                   /* userdata */
    (DWORD) Page3_Callback,              /* create/destroy callback */
};

HWND _export CALLBACK Page3_Callback(BOOL fCreate, HWND hwndOwner,
                                     HWND hwndPage, HWND hwndTab)
{
    if (fCreate) {                      // creating a new page
        if (hwndPage) {
            // already created, we could do some initialization here.
            // this will be called every time the page becomes active
            // The WM_CREATE/WM_INITDIALOG/WM_DESTROY messages are also sent to
            // the page and could be used to determine activation/deactivation.
            // of the page.
            // optional, set the main window's title to the window title ...
            SftTabs_CopyWindowTitle(hwndPage, hwndOwner);
            return NULL;
        } else {
            // Create the page.
            // You can create and initialize any type of window here, not just
dialogs.
            // Use CreateWindow to create other windows. Don't specify WS_VISIBLE, but
            // make sure you use WS_TABSTOP.
            // When creating a non-dialog window, make sure to call
SftTabs_SetPageActive
            // after the page has been created.
            HWND hwnd = CreateDialogParam(g_hInst, MAKEINTRESOURCE(IDD_PAGE3),
                                                  hwndOwner, (DLGPROC)Page3_DialogProc,
                    (LPARAM)(UINT)hwndTab);// pass tab control as data
            // optional, set the main window's title to the window title defined ...
            SftTabs_CopyWindowTitle(hwnd, hwndOwner);
            return hwnd;
        }
    } else {                            // destroying page
        // We'll always destroy this page (to save resources)
        DestroyWindow(hwndPage);
        return NULL;
    }
}
```

This callback routine is called by SftTabs/DLL to perform certain functions, based on the parameters passed.

The callback routine creates the dialog (page) using CreateDialogParam. The window handle of the tab control is passed as the last parameter to CreateDialogParam. The WM_INITDIALOG processing of the page's dialog procedure needs access to the tab control's window handle for the call to SftTabs_SetPageActive. This is a convenient way to pass it to the dialog procedure. Of course, it could also be passed using other mechanisms.

### Creating a Page Dialog Procedure

Once the page is created, all Windows messages are sent to the page's dialog procedure. Any messages that the page doesn't handle should be passed to SftTabs/DLL by using a call to SftTabs_HandleDialogMessage.

```
BOOL _export  CALLBACK Page3_DialogProc(HWND hwndDlg, UINT msg,
                                        WPARAM wParam, LPARAM lParam)
{
    switch (msg) {
    case WM_INITDIALOG:
        SetWindowText(GetDlgItem(hwndDlg, IDC_P3_EDIT1), TEXT("Type Here"));
        // initialize page
        SftTabs_SetPageActive(hwndDlg, (HWND) lParam, NULL);
        return !SftTabs_IsRegisteredDialog(GetParent(hwndDlg));
    }

    if (SftTabs_HandleDialogMessage(hwndDlg, msg, wParam, lParam))
        return TRUE;

    return FALSE;
}
```

When the page is initialized and handles the WM_INITDIALOG message, it must call SftTabs_SetPageActive to notify SftTabs/DLL that this page is the active page. SftTabs/DLL will at this time resize the page to fit into the tab control's client area, override certain incompatible window styles and make the page visible.

When the very first page is initialized, the focus should be on the first control in that page. Otherwise, when other pages have been active before, the focus should remain on the control which currently has the focus (usually the tab control). The return value of the WM_INITDIALOG message handler determines the focus handling. GetParent below returns the window handle of the page's parent window, which should be the tabbed dialog. This window handle can be tested to find out if the dialog has been registered with SftTabs/DLL. When the very first page is activating, the tabbed dialog has not yet been registered, SftTabs_IsRegisteredDialog would return FALSE and the dialog procedure would subsequently return TRUE, allowing the focus to be set to the first control. Otherwise, if this is not the very first page, FALSE would be returned, and the focus would remain on the control that currently has the focus.

```
        return !SftTabs_IsRegisteredDialog(GetParent(hwndDlg));
```

## Implementing Tabbed Windows

A tabbed window is created just like a regular window. A tabbed window has at least one tab control as its child window (with or without a client area). Each tab has an attached page (although during development of a tabbed dialog, a tab doesn't require an attached page). As the user switches between tabs, the appropriate page is created, displayed and destroyed.

### Creating a Tabbed Window Window Procedure

The following code sample (from C:\SFTTABS\SAMPLES\CSAM2\CSAM2.C) shows a typical window procedure used for a tabbed window. Most of the code has been created using the SftTabs/DLL Wizard and then copied into the application.

```
/*********************************************************************/
/*                           Frame Window                            */
/*********************************************************************/

/*- Tab Control Initialization Data ----------------------------------------*/

static const SFTTABS_CONTROL CtlInit = {
    SFTTABSSTYLE_SIMPLE,                /* tab style */
    1,                                  /* number of rows */
    0,                                  /* number of tabs per row (if fFixed) */
    0,                                  /* width of left margin */
```

```
    0,                                  /* width of right margin */
    FALSE,                              /* same width for all tabs */
    FALSE,                              /* Client area wanted */
    FALSE,                              /* allow multiline label text */
    FALSE,                              /* use with dialog */
    FALSE,                              /* use specified background color only for
text */
    TRUE,                               /* scrollable tabs */
    FALSE,                              /* hide scroll buttons */
    TRUE,                               /* bold font for active tab wanted */
    FALSE,                              /* fill rows completely */
    NULL,                               /* scroll button bitmap */
    NULL,                               /* Dialog data associated with active tab */
    NULL,                               /* Dialog window handle associated with ...
    NULL,                               /* Frame, used as client area */
    TRUE,                               /* Tooltips wanted */
    FALSE,                              /* drop text if it doesn't fit */
    FALSE,                              /* conditional scroll buttons */
};

static const SFTTABS_TAB Tab0 = {   /*&Listbox */
    SFTTABS_NOCOLOR, SFTTABS_NOCOLOR,    /* background, foreground color */
    SFTTABS_NOCOLOR, RGB(0,0,255),       /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },           /* Bitmap, Icon */
    TRUE,                                /* enabled/disabled */
    0,                                   /* userdata */
    (DWORD) Page1_Callback,              /* create/destroy callback */
};

static const SFTTABS_TAB Tab1 = {   /*&Edit Control */
    SFTTABS_NOCOLOR, SFTTABS_NOCOLOR,    /* background, foreground color */
    SFTTABS_NOCOLOR, RGB(0,0,255),       /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },           /* Bitmap, Icon */
    TRUE,                                /* enabled/disabled */
    0,                                   /* userdata */
    (DWORD) Page2_Callback,              /* create/destroy callback */
};

static const SFTTABS_TAB Tab2 = {   /*&Dialog */
    SFTTABS_NOCOLOR, SFTTABS_NOCOLOR,    /* background, foreground color */
    SFTTABS_NOCOLOR, RGB(0,0,255),       /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },           /* Bitmap, Icon */
    TRUE,                                /* enabled/disabled */
    0,                                   /* userdata */
    (DWORD) Page3_Callback,              /* create/destroy callback */
};
```

The window procedure shown here initializes the tab control in its WM_CREATE message handler.

```
LRESULT EXP CALLBACK Frame_WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
{

       ... other source code removed

    switch (msg) {

    case WM_CREATE: {
        int index;

        /* This static window is just a filler window above the tab control */
        /* We only need it to get the right background color above the tab control */
        pfrm->hwndStatic = CreateWindow(/* Create a static window */
            TEXT("STATIC"),             /* Window Class */
```

```
        TEXT(""),                   /* Window Title (not used) */
        WS_CHILD|WS_VISIBLE,        /* Window Style */
        0, 0,                       /* x, y */
        0, 0,                       /* cx, cy */
        hwnd,                       /* Parent Window */
        NULL,                       /* control ID */
        g_hInst,                    /* Application Instance */
        NULL);                      /* creation data */
    if (pfrm->hwndStatic == NULL)   /* create failed */
        return -1;

    /* Create the tab control */
    pfrm->hwndTab = CreateWindow(   /* Create the tab control */
        TEXT(SFTTABS_CLASS),        /* Window Class */
        TEXT(""),                   /* Window Title (not used) */
        WS_CHILD|WS_VISIBLE|        /* Window Style */
        WS_CLIPCHILDREN|WS_TABSTOP|
        SFTTABSSTYLE_STANDARD,
        0, 0,                       /* x, y */
        0, 0,                       /* cx, cy */
        hwnd,                       /* Parent Window */
        (HMENU) IDC_TAB,            /* Tab control ID */
        g_hInst,                    /* Application Instance */
        NULL);                      /* creation data */
    if (pfrm->hwndTab == NULL)      /* create failed */
        return -1;

    /* Create a frame window. This frame window will be used by SftTabs/DLL */
    /* to resize the pages that are attached to the tabs */
    pfrm->hwndFrame = CreateWindow( /* Create a static window */
        TEXT("STATIC"),             /* Window Class */
        TEXT(""),                   /* Window Title (not used) */
        WS_CHILD,                   /* Window Style NOTE: IT'S NOT VISIBLE */
        0, 0,                       /* x, y */
        0, 0,                       /* cx, cy */
        hwnd,                       /* Parent Window */
        NULL,                       /* control ID */
        g_hInst,                    /* Application Instance */
        NULL);                      /* creation data */
    if (pfrm->hwndFrame == NULL)   /* create failed */
        return -1;

    // Initialization is faster if we set redraw off
    SendMessage(pfrm->hwndTab, WM_SETREDRAW, (WPARAM)FALSE, 0);

    /* We are using new features */
    SftTabs_SetVersion(pfrm->hwndTab, SFTTABS_2_1); // exploit 2.1 functions

    index = SftTabs_AddTab(pfrm->hwndTab, TEXT("&Listbox"));
    SftTabs_SetToolTip(pfrm->hwndTab, index, TEXT("A standard listbox is ...
    SftTabs_SetTabInfo(pfrm->hwndTab, index, &Tab0);

    index = SftTabs_AddTab(pfrm->hwndTab, TEXT("&Edit Control"));
    SftTabs_SetToolTip(pfrm->hwndTab, index, TEXT("A standard edit control ...
    SftTabs_SetTabInfo(pfrm->hwndTab, index, &Tab1);

    index = SftTabs_AddTab(pfrm->hwndTab, TEXT("&Dialog"));
    SftTabs_SetToolTip(pfrm->hwndTab, index, TEXT("A dialog is attached to this
tab"));
    SftTabs_SetTabInfo(pfrm->hwndTab, index, &Tab2);

    SftTabs_SetControlInfo(pfrm->hwndTab, &CtlInit);

    // Make sure to turn redraw back on
    SendMessage(pfrm->hwndTab, WM_SETREDRAW, (WPARAM)TRUE, 0);
    InvalidateRect(pfrm->hwndTab, NULL, TRUE);
```

```
        // Activate current page. Note the frame window is supplied in this example.
        // if your tab control has a client area (see fClientArea), you don't need a
        // frame window
        SftTabs_ActivatePage(hwnd, pfrm->hwndTab, pfrm->hwndFrame, TRUE);

        // Mark the window as a main, tabbed windows (so accel. keys work) by ...
        // Register the window AFTER activating the current page
        SftTabs_RegisterWindow(hwnd);


        return 0L;
     }
```

The call to SftTabs_ActivatePage makes the current page active. The SftTabs_RegisterWindow call registers the window for special tabbed window handling by the DLL. After registering the window keyboard accelerator keys (Alt-*x*) handling will be performed by SftTabs/DLL.

The following WM_SIZE message handler resizes the tabbed window's child windows.

```
    case WM_SIZE: {
        RECT rect;
        HWND hwndCtl;

        // resize all child windows

        // get frame window dimension
        GetClientRect(hwnd, &rect);

        // calculate position of tab control
        if (pfrm->hwndTab) {
            SFTTABS_CONTROL Ctl;
            // Get tab control info
            SftTabs_GetControlInfo(pfrm->hwndTab, &Ctl);
            // Ctl.naturalSize has best height for this tab control

            rect.top += 5+Ctl.naturalSize;
        }


        // reposition static control which serves as a filler window above the tab
control
        if (pfrm->hwndStatic)
            MoveWindow(pfrm->hwndStatic, rect.left, 0, rect.right-rect.left, 5, TRUE);

        // now reposition tab control
        if (pfrm->hwndTab)
            MoveWindow(pfrm->hwndTab, 0, 5, rect.right-rect.left, rect.top-5, TRUE);

        // reposition the frame window
        if (pfrm->hwndFrame)
            MoveWindow(pfrm->hwndFrame, rect.left, rect.top, rect.right-rect.left,
                                        rect.bottom-rect.top, TRUE);
        // now that the frame window has the right size, resize all pages
        if (pfrm->hwndTab)
            SftTabs_ResizePages(pfrm->hwndTab);

        return 0L;
     }
```

The following WM_DESTROY message handler shows the required cleanup calls and also unregisters the dialog from the DLL.

```
    case WM_DESTROY:
        // Unregister, or the window properties used won't be removed
        SftTabs_UnregisterWindow(hwnd);
        // destroy all pages (BEFORE destroying tab control)
        SftTabs_Destroy(hwnd, pfrm->hwndTab);
```

```
        if (pfrm->hwndStatic)
            DestroyWindow(pfrm->hwndStatic);
        if (pfrm->hwndFrame)
            DestroyWindow(pfrm->hwndFrame);
        if (pfrm->hwndTab)
            DestroyWindow(pfrm->hwndTab);
        break;
```

The following WM_COMMAND message handler responds to notifications sent to the tabbed window by the tab control. When the user attempts to switch to another page by clicking a tab using the left mouse button or by using the keyboard, a SFTTABSN_SWITCHING notification is sent to the tabbed window. The window then calls SftTabs_DeactivatePage to test if the active page can be left. This call to SftTabs_DeactivatePage results in a WM_QUERYENDSESSION message being sent to the page's dialog or window procedure. Based on the window procedure's response, the page may or may not be deactivated. By handling the WM_QUERYENDSESSION message, a page can prevent a user from switching away from a tab.

If a new tab has become active, the tabbed window receives a SFTTABSN_SWITCHED notification, so the associated page can be made active by the call to SftTabs_ActivatePage.

```
    case WM_COMMAND: {
#if defined(_WIN32) || defined(__WIN32__)
        HWND hwndCtl = (HWND) lParam;
        int id = LOWORD(wParam);
        int code = HIWORD(wParam);
#else
        HWND hwndCtl = (HWND) LOWORD(lParam);
        int id = (int) wParam;
        int code = HIWORD(lParam);
#endif
        if (hwndCtl) {
            switch (id) {
            case IDC_TAB:
                switch (code) {
                case SFTTABSN_SWITCHING:// we're about to switch away from
                    // the current page.  If you need to know what the new
                    // page will be use SftTabs_GetNextTab(hwndCtl).
                    if (!SftTabs_DeactivatePage(hwnd, hwndCtl))
                        // couldn't deactivate current page, so don't switch
                        SendMessage(hwndCtl, WM_CANCELMODE, 0, 0);
                    break;
                case SFTTABSN_SWITCHED:// we switched to a new page
                    SftTabs_ActivatePage(hwnd, hwndCtl, NULL, FALSE);
                    break;
                }
                break;
            case IDOK:
            case IDCANCEL:
                if (code == BN_CLICKED)
                    SendMessage(hwnd, WM_COMMAND, id, 0);
                break;
            }
        } else {
            switch (id) {
            case IDM_EXIT:
                // The currently active page will be called with a
                // WM_QUERYENDSESSION message to determine if it can be closed
                if (SftTabs_ClosePossible(hwnd, GetDlgItem(hwnd, IDC_TAB))) {
                    DestroyWindow(hwnd);
                    PostQuitMessage(0);
                }
                break;
```

```
            }
        }
        break;
     }
```

At the end of the window procedure, a call to SftTabs_HandleDialogMessage should be made to allow SftTabs/DLL to process any messages. Without this call, SftTabs/DLL may not be able to provide the tabbed dialog handling.

```
    // Call SftTabs/DLL to let it handle some messages (mostly for keyboard accel.
keys)
    {
        LRESULT lRes;
        if (SftTabs_HandleWindowMessage(hwnd, msg, wParam, lParam, &lRes))
            return lRes;
    }

    return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

The SFTTABS_TAB structures used to define each tab in the tab control, also define a callback routine of type SFTTABS_TABCALLBACK. This callback routine is responsible for creating and destroying the window which represents the page. SftTabs/DLL will hide and disable a window, but it is up to this callback routine to create and destroy the window. The following tab definition is repeated here so it can be shown next to the callback function:

```
static const SFTTABS_TAB Tab1 = {   /*&Edit Control */
    SFTTABS_NOCOLOR, SFTTABS_NOCOLOR,    /* background, foreground color */
    SFTTABS_NOCOLOR, RGB(0,0,255),       /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },           /* Bitmap, Icon */
    TRUE,                                /* enabled/disabled */
    0,                                   /* userdata */
    (DWORD) Page2_Callback,              /* create/destroy callback */
};

HWND _export CALLBACK Page2_Callback(BOOL fCreate, HWND hwndOwner,
                                HWND hwndPage, HWND hwndTab)
{
    if (fCreate) {                      // creating a new page
        if (hwndPage) {
            // already created, we could do some initialization here.
            // this will be called every time the page becomes active.
            // The WM_SHOWWINDOW message is also sent to the page and could
            // be used to determine activation/deactivation of the page.
            SetWindowText(hwndPage, TEXT("Thank you for coming back."));
            return NULL;                // return NULL, ignored
        } else {
            // create the window
            HWND hwnd;
            // Create the edit control
            // You can create and initialize any type of window here, not just
dialogs.
            // Use CreateWindow to create other windows. Don't specify WS_VISIBLE, but
            // make sure you use WS_TABSTOP.
            hwnd = CreateWindow(  /* Create the list box */
                "EDIT",                 /* Window Class */
                TEXT(""),               /* Window Title (not used) */
                WS_CHILD|               /* Window Style */
                WS_TABSTOP|ES_MULTILINE|ES_WANTRETURN,
                0, 0,                   /* x, y */
                0, 0,                   /* cx, cy */
                hwndOwner,              /* Parent Window */
                (HMENU) IDC_EDIT,       /* control ID */
                g_hInst,                /* Application Instance */
```

```
                NULL);                  /* creation data */
            if (hwnd == NULL)           /* create failed */
                return NULL;
            SetWindowText(hwnd, TEXT("This is an edit control.\r\nClick a ...
            SftTabs_SetPageActive(hwnd, hwndTab, NULL);
            return hwnd;
        }
    } else {                            // destroying page
        if (hwndOwner)                  // - because we're switching away
            return hwndPage;            //   keep the window handle, don't destroy it
        else {                          // - because we're closing the main dialog
            DestroyWindow(hwndPage);
            return NULL;
        }
    }
}
```

This callback routine is called by SftTabs/DLL to perform certain functions, based on the parameters passed.

The callback routine creates the window (page) using CreateWindow.

# C++/MFC Programming

This section describes how to use SftTabs with an application written using C++ and the Microsoft Foundation Class library (MFC).

## Building an Application

A) Every source program making use of a SftTabs/DLL control must include the required header file SFTTB.H by using the #include directive.

```
#include "sfttb.h"                      /* SftTabs/DLL required header file */
```

This include statement should appear after the #include <windows.h> statement. The file is located in the directory C:\SFTTABS\INCLUDE (unless changed during the installation).

B) One source program must include the CSftTabs class implementation, using the #include directive.

```
#include "sfttbm.cpp"                   /* SftTabs/DLL implementation */
```

This include statement should appear after the #include "sfttb.h" statement. This is the preferred method to include the implementation of the CSftTabs class. Adding the file SFTTBM.CPP to your project is not recommended because it will complicate the use of pre-compiled header files. The file is located in the directory C:\SFTTABS\INCLUDE (unless changed during the installation).

C) In order to use SftTabs/DLL controls, an application must call the CSftTabs::RegisterApp function. The call to this function is required so that SftTabs/DLL window classes can be registered. This call has to be made before any SftTabs/DLL controls are created. Add the following statement to your source code, the preferred location is the InitInstance member function of your CWinApp based application object:

```
CSftTabs::RegisterApp();            /* Use SftTabs/DLL with this application
*/
```

D) Once SftTabs/DLL controls are no longer needed, an application must call the CSftTabs::UnregisterApp function. The call to this function is required so that SftTabs/DLL window classes can be unregistered and cleanup processing can take place. This call has to be made after all SftTabs/DLL controls have been destroyed. The preferred location is the ExitInstance member function of your CWinApp based application object:

```
CSftTabs::UnregisterApp();          /* No longer use SftTabs/DLL */
```

E) The application's executable (EXE or DLL) must be linked with the correct LIB file, depending on the target environment. The DLL must be available and accessible at run-time for proper execution. The DLL used at run-time depends on the LIB file used at link time.

| Target Environment | LIB File Required when Linking | DLL File Required at Run-Time |
|---|---|---|
| **Windows 3.1, 16-bit applications** | SFTTB.LIB | SFTTB.DLL |
| **WIN32, all 32-bit environments including Windows NT (without UNICODE support** | SFTTB32.LIB | SFTTB32.DLL |

| | | |
|---|---|---|
| **Windows NT only (with UNICODE support** | SFTTB32U.LIB | SFTTB32U.DLL |

All required files can be found in the directory C:\SFTTABS\LIB and C:\SFTTABS\BIN, unless changed during the installation.

## Adding a Tab Control

ClassWizard does not support new classes such as CSftTabs, so any tab control instance variables, notification handlers, message map entries, etc., have to be added manually. To simplify this process, you can copy these items that are generated by ClassWizard for other "standard" Windows controls.

There are two methods to add a tab control to an application:

- using dialog resources
- using CSftTabs::Create

Adding a tab control using dialog resources is accomplished by using a resource editor to design a dialog. For more information on the different resource editors supported by SftTabs/DLL, see Resource Workshop, Borland C++, AppStudio, Visual C++ and Dialog Editor (Windows SDK). Once a tab control is created, its CSftTabs based object can be obtained by using the Windows GetDlgItem function or attached to a CSftTabs object using SubclassWindow.

```
CSftTabs * pTabControl;
pTabControl = (CSftTabs *) GetDlgItem(IDC_TAB);
```

or

```
CSftTabs m_Tab;
m_Tab.SubclassDlgItem(IDC_TABS, this);
```

Another method to create a tab control is by using the CSftTabs::Create member function.

```
CSftTabs m_Tab;
m_Tab.Create(WS_CHILD | WS_VISIBLE, CRect(250,200,400,700), pParentWnd, IDC_TABS);
```

For more information on the various parameters used, see the Create member function documentation.

## Handling Notifications

As with standard Windows controls, applications must respond to events and messages to cause controls to respond to user requests. For additional information see Notifications.

ClassWizard does not support new classes such as CSftTabs, so any tab control instance variables, notification handlers, message map entries, etc., have to be added manually. To simplify this process, you can copy these items that are generated by ClassWizard for other "standard" Windows controls.

### Switching Between Tabs

Switching between tabs is fully automatic, however, an application may wish to prevent a user from switching to another tab. By responding to the WM_COMMAND, SFTTABSN_SWITCHING notification, an application can prevent completion of the tab switch by sending a WM_CANCELMODE message to the tab control.

```
// Event handler prototype added to dialog/window class
afx_msg void OnTabSwitching();

// Event handler(s) added to message map
BEGIN_MESSAGE_MAP(CSampleView, CView)
    ON_SFTTABSN_SWITCHING(IDC_TAB, OnTabSwitching)
END_MESSAGE_MAP()

// Event handler implementation
```

```
void CSampleView::OnTabSwitching()
{
    if (YouDontLikeThisUser())
        m_Tab.SendMessage(WM_CANCELMODE);// cancel switching
}
```

An application has to make attached controls or dialogs visible when switching between tabs. The SftTabs/DLL API offers functions to manage dialogs and Windows controls that are attached to tabs. See Implementing Tabbed Dialogs and Implementing Tabbed Windows for more information.

## 3D and Colors

SftTabs/DLL offers many tab control styles which look best on a gray background, using one of the following methods:

### CTL3DV2 or CTL3D32

The tab control can be used with CTL3DV2 (or CTL3D32). Any dialogs attached to a tab control can use the 3D display, if properly enabled (usually using Ctl3dAutoSubclass). For more information on CTL3DV2 or CTL3D32, see the Microsoft documentation.

### WM_CTLCOLOR, WM_CTLCOLORSTATIC

| This message is generated by the tab control for compatibility with SftTabs 2.0 only. When developing new applications, please use CSftTabs::SetCtlColors instead. |
|---|

The appearance of the tab control can be modified by handling the WM_CTLCOLOR message. The parent window can override the background color used by the tab control by defining a WM_CTLCOLOR message handler.

```
// Event handler definition added to dialog class
afx_msg HBRUSH OnCtlColorTab( CDC* pDC, CWnd* pWnd, UINT nCtlColor );

// Event handler added to parent window message map
ON_WM_CTLCOLOR(OnCtlColorTab)

// Event handler implementation
HBRUSH CYourDialog::OnCtlColorTab(CDC* pDC, CWnd* pWnd, UINT nCtlColor)
{
    if (nCtlColor == CTLCOLOR_STATIC) {
        pDC->SetBkColor(RGB(192,192,192));      // on gray background
        return (HBRUSH)::GetStockObject(LTGRAY_BRUSH);
    }
}
```

## Implementing Tabbed Dialogs

A tabbed dialog is created just like any other dialog. A tabbed dialog has a tab control with an available client area. In this client area, pages are displayed. Each tab has an attached page (although during development of a tabbed dialog, a tab doesn't require an attached page). As the user switches between tabs, the appropriate page is created, displayed and destroyed.

A tabbed dialog and each page are based on the class CSftTabsDialog and CSftTabsPage, which are both derived from the MFC class CDialog. This makes conversion of existing dialogs and development of new pages very easy. Tabbed dialogs and pages are first designed using a resource editor. The sections Resource Workshop, Borland C++, AppStudio, Visual C++ and Dialog Editor (Windows SDK) describe how the necessary tabbed dialog and page (dialog) resources are created.

ClassWizard can be used to create the dialogs and pages. However, ClassWizard can only create new classes based on CDialog (not CSftTabsDialog or CSftTabsPage). After ClassWizard generates a new class derived from CDialog, you have to manually change

references to CDialog to the classes CSftTabsDialog and CSftTabsPage. When adding new member functions, make sure to call the CSftTabsDialog and CSftTabsPage base classes instead of CDialog.

Once the necessary dialogs have been designed, the tab control layout can be defined using the SftTabs/DLL Wizard. The SftTabs/DLL Wizard also creates much of the code required to initialize the tab control. This code should be copied to the application (with possibly minor modifications).

### Creating a Tabbed Dialog

The following code sample (from C:\SFTTABS\SAMPLES\MFCSAM1\MAINDLG.CPP) shows a typical implementation of a tabbed dialog. Most of the code has been created using the SftTabs/DLL Wizard and then copied into the application. The code that was copied is marked by a vertical line on the side. The majority of the code is used to initialize the tab control.

```
/*- Tab Control Initialization Data -----------------------------------------*/

static const SFTTABS_CONTROL CtlInit = {
    SFTTABSSTYLE_MODERN_I,              /* tab style */
    2,                                  /* number of rows */
    0,                                  /* number of tabs per row (if fFixed) */
    0,                                  /* width of left margin */
    0,                                  /* width of right margin */
    FALSE,                              /* same width for all tabs */
    TRUE,                               /* Client area wanted */
    FALSE,                              /* allow multiline label text */
    TRUE,                               /* use with dialog */
    FALSE,                              /* use specified background color only for
text */
    FALSE,                              /* scrollable tabs */
    FALSE,                              /* hide scroll buttons */
    TRUE,                               /* bold font for active tab wanted */
    TRUE,                               /* fill rows completely */
    NULL,                               /* scroll button bitmap */
    NULL,                               /* Dialog data associated with active tab */
    NULL,                               /* Dialog window handle associated with ...
    NULL,                               /* Frame, used as client area */
    TRUE,                               /* Tooltips wanted */
    FALSE,                              /* drop text if it doesn't fit */
    FALSE,                              /* conditional scroll buttons */
};

static const SFTTABS_TAB Tab0 = {   /*The First One */
    SFTTABS_NOCOLOR, RGB(0,0,255),      /* background, foreground color */
    SFTTABS_NOCOLOR, RGB(0,0,255),      /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_LEFT, SFTTABS_GRAPH_BITMAP },
                                        /* Bitmap, Icon */
    TRUE,                               /* enabled/disabled */
    0                                   /* userdata */
};

static const SFTTABS_TAB Tab1 = {   /*&Second */
    SFTTABS_NOCOLOR, RGB(255,0,0),      /* background, foreground color */
    SFTTABS_NOCOLOR, RGB(255,0,0),      /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_RIGHT, SFTTABS_GRAPH_ICON },
                                        /* Bitmap, Icon */
    TRUE,                               /* enabled/disabled */
    0                                   /* userdata */
};
```

```
static const SFTTABS_TAB Tab2 = {   /*&Third */
    SFTTABS_NOCOLOR, RGB(128,128,0),     /* background, foreground color */
    SFTTABS_NOCOLOR, RGB(128,128,0),     /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },           /* Bitmap, Icon */
    TRUE,                                /* enabled/disabled */
    0                                    /* userdata */
};

static const SFTTABS_TAB Tab3 = {   /*F&ourth */
    SFTTABS_NOCOLOR, RGB(0,255,255),     /* background, foreground color */
    SFTTABS_NOCOLOR, RGB(0,255,255),     /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },           /* Bitmap, Icon */
    TRUE,                                /* enabled/disabled */
    0                                    /* userdata */
};

static const SFTTABS_TAB Tab4 = {   /*F&ifth */
    SFTTABS_NOCOLOR, RGB(0,0,128),       /* background, foreground color */
    SFTTABS_NOCOLOR, RGB(0,0,128),       /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },           /* Bitmap, Icon */
    TRUE,                                /* enabled/disabled */
    0                                    /* userdata */
};

static const SFTTABS_TAB Tab5 = {   /*Si&xth */
    SFTTABS_NOCOLOR, RGB(128,0,0),       /* background, foreground color */
    SFTTABS_NOCOLOR, RGB(128,0,0),       /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },           /* Bitmap, Icon */
    TRUE,                                /* enabled/disabled */
    0                                    /* userdata */
};
```

The OnInitDialog message handler of the tabbed dialog initializes the tab control and associates CSftTabsPage based objects to each tab.

```
BOOL CMainDlg::OnInitDialog()
{
    int index;
    SFTTABS_TAB Tab;
    /* Associate the tab control created from the dialog       */
    /* resource with the C++ object.                           */
    m_Tab.SubclassDlgItem(IDC_TAB, this /* parent window */);
    /* You could use DDX/DDV instead and add the following     */
    /* line to the DoDataExchange function of the tab          */
    /* control's parent window (remove the //).                */
    // DDX_Control(pDX, IDC_TAB, m_Tab);

    /* Initialization is faster if we set redraw off */
    m_Tab.SetRedraw(FALSE);

    /* We are using new features */
    m_Tab.SetVersion(SFTTABS_2_1);

    index = m_Tab.AddTab(_T("The First One"));
    m_Tab.SetToolTip(index, _T("Demonstrates tabbing into and out of the tab page"));
    Tab = Tab0;
    Tab.graph.item.hBitmap = (HBITMAP) m_SampleBitmap.m_hObject;
    m_Tab.SetTabInfo(index, &Tab);
    m_Tab.SetTabDialog(index, new CPage1(this)); /* tab page */

    index = m_Tab.AddTab(_T("&Second"));
    m_Tab.SetToolTip(index, _T("Demonstrates how an application can ...
    Tab = Tab1;
```

```
    Tab.graph.item.hIcon = m_hSampleIcon;
    m_Tab.SetTabInfo(index, &Tab);
    m_Tab.SetTabDialog(index, new CPage2(this)); /* tab page */

    index = m_Tab.AddTab(_T("&Third"));
    m_Tab.SetToolTip(index, _T("This page is reset everytime you switch to it"));
    m_Tab.SetTabInfo(index, &Tab2);
    m_Tab.SetTabDialog(index, new CPage3(this)); /* tab page */

    index = m_Tab.AddTab(_T("F&ourth"));
    m_Tab.SetToolTip(index, _T("A page with private OK, Cancel, Next and ...
    m_Tab.SetTabInfo(index, &Tab3);
    m_Tab.SetTabDialog(index, new CPage4(this)); /* tab page */

    index = m_Tab.AddTab(_T("F&ifth"));
    m_Tab.SetToolTip(index, _T("A page that has not yet been implemented"));
    m_Tab.SetTabInfo(index, &Tab4);
    // If you don't want to attach a page to the tab, the following is optional
//  m_Tab.SetTabDialog(index, new an_object_based_on_CSftTabsPage(this)); /* tab page
*/

    index = m_Tab.AddTab(_T("Si&xth"));
    m_Tab.SetToolTip(index, _T("A page with nested tab controls and pages"));
    m_Tab.SetTabInfo(index, &Tab5);
    m_Tab.SetTabDialog(index, new CPage6(this)); /* tab page */

    m_Tab.SetControlInfo(&CtlInit);

    // Make sure to turn redraw back on
    m_Tab.SetRedraw(TRUE);
    m_Tab.InvalidateRect(NULL, TRUE);

    // If you are not using the sheet/page classes, remove the call ...
    // Initialize tab control
    InitializeTabControl(0, &m_Tab, NULL);
    return FALSE; // if this is a dialog's OnInitDialog member function
}
```

Each call to CSftTabs::SetTabDialog associates a CSftTabsPage based dialog to a tab. Cleanup of all resources, including the dynamically allocated pages is done automatically when the tabbed dialog is destroyed.

The call to InitializeTabControl starts the tabbed dialog handling and creates the current page.

A tabbed dialog should always return FALSE from the OnInitDialog member function. The input focus has already been set by SftTabs/DLL, so returning FALSE will prevent Windows from setting the focus (to the wrong control).

### Creating a Page

The implementation of a page is identical to a regular dialog, except that the base class is CSftTabsPage instead of CDialog.

## Implementing Tabbed Windows

A tabbed window is created just like a regular window. A tabbed window has at least one tab control as its child window (with or without a client area). Each tab has an attached page (although during development of a tabbed window, a tab doesn't require an attached page). As the user switches between tabs, the appropriate page is created, displayed and destroyed.

A tabbed window and each page are based on the class CWnd or any of its derived classes, such as CView, CFormView, etc. Using multiple inheritance, a window can inherit the required support to make into a tabbed window or a page. This makes conversion of existing windows and development of new pages very easy.

ClassWizard can be used to create the tabbed window and the pages initially. By using multiple inheritance, the classes CSftTabsWindowSheet and CSftTabsWindowPage are used to add tabbed window and page support to the new classes.

### Creating a Tabbed Window

Most CWnd based classes are suitable to be used as a tabbed window. By using multiple inheritance, a class can be used as a tabbed window by inheriting the required support from the class CSftTabsWindowSheet.

```
    class CSampleView : public CView, public CSftTabsWindowSheet
    {
        ... class definitions
    }
```

The following code sample (from C:\SFTTABS\SAMPLES\MFCSAM2\SAMPLVW.CPP) shows a typical implementation of a tabbed window. Most of the code has been created using the SftTabs/DLL Wizard and then copied into the application.

```
/*- Tab Control Initialization Data ----------------------------------------*/

static const SFTTABS_CONTROL CtlInit = {
    SFTTABSSTYLE_SIMPLE,                /* tab style */
    1,                                  /* number of rows */
    0,                                  /* number of tabs per row (if fFixed) */
    5,                                  /* width of left margin */
    5,                                  /* width of right margin */
    FALSE,                              /* same width for all tabs */
#if defined(TAB_CONTROL_WITH_CLIENTAREA)
    TRUE,                               /* Client area wanted */
#else
    FALSE,                              /* Client area not wanted */
#endif
    FALSE,                              /* allow multiline label text */
    TRUE,                               /* use with dialog */
    FALSE,                              /* use specified background color only for
text */
    TRUE,                               /* scrollable tabs */
    FALSE,                              /* hide scroll buttons */
    TRUE,                               /* bold font for active tab wanted */
    FALSE,                              /* fill rows completely */
    NULL,                               /* scroll button bitmap */
    NULL,                               /* Dialog data associated with active tab */
    NULL,                               /* Dialog window handle associated with ...
    NULL,                               /* Frame, used as client area */
    TRUE,                               /* Tooltips wanted */
    FALSE,                              /* drop text if it doesn't fit */
    TRUE,                               /* conditional scroll buttons */
};

static const SFTTABS_TAB Tab0 = {   /*&Listbox */
    SFTTABS_NOCOLOR, SFTTABS_NOCOLOR,   /* background, foreground color */
    SFTTABS_NOCOLOR, RGB(0,0,255),      /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },          /* Bitmap, Icon */
    TRUE,                               /* enabled/disabled */
    0                                   /* userdata */
};

static const SFTTABS_TAB Tab1 = {   /*&Edit Control */
    SFTTABS_NOCOLOR, SFTTABS_NOCOLOR,   /* background, foreground color */
    SFTTABS_NOCOLOR, RGB(0,0,255),      /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },          /* Bitmap, Icon */
    TRUE,                               /* enabled/disabled */
```

```
    0                                   /* userdata */
};

static const SFTTABS_TAB Tab2 = {   /*&Other Listbox */
    SFTTABS_NOCOLOR, SFTTABS_NOCOLOR,    /* background, foreground color */
    SFTTABS_NOCOLOR, RGB(0,0,255),       /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },           /* Bitmap, Icon */
    TRUE,                                /* enabled/disabled */
    0                                    /* userdata */
};
```

The OnCreate message handler of the tabbed window initializes the tab control and associates CSftTabsWindowPage based objects to each tab.

```
int CSampleView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CView::OnCreate(lpCreateStruct) == -1)
        return -1;

    // Create a static control that we can place above the tab control.
    // This is just used to cover the parent window in that area.
    if (!m_Gap.Create(_T(""), SS_SIMPLE | WS_VISIBLE | WS_CHILD,
                CRect(0, 0, 0, 0),           /* position */
                this))
        return -1;

#if !defined(TAB_CONTROL_WITH_CLIENTAREA)
    // Create a static control that we can use as a frame window for the tab control's
    // pages.  This window is not visible and is just used to indicate the page
position
    if (!m_Frame.Create(_T(""), SS_SIMPLE | WS_CHILD,
                CRect(0, 0, 0, 0),           /* position */
                this))
        return -1;
#endif

    // Create the tab control
    if (!m_Tab.Create(
            WS_VISIBLE | WS_CHILD |          /* Visible, child window */
                WS_CLIPCHILDREN | WS_TABSTOP |
                                             /* Clip child windows, tabstop */
                WS_GROUP,                    /* Group */
                CRect(0, 0, 0, 0),           /* position */
            this,                            /* Parent window */
            IDC_TAB))                        /* tab control ID */
        return -1;

    int index;

    /* Initialization is faster if we set redraw off */
    m_Tab.SetRedraw(FALSE);

    /* Create the font used for the tab control.               */
    /* Fonts are owned by the application and have to remain   */
    /* valid as long as the tab control uses the font.         */

    int height;                          /* Height in pixels */
    HDC hDC;                             /* Device context */

    /* Create the font to be used for the tab control.         */
    hDC = ::GetDC(NULL);                 /* Get a device context */
    height = MulDiv(12, ::GetDeviceCaps(hDC, LOGPIXELSY), 72);/* Convert point-...
    m_Font.CreateFont(-height, 0, 0, 0, FW_NORMAL, 0, 0, 0, 0, 0, 0, 0, 0,
_T("Arial"));
    ::ReleaseDC(NULL, hDC);              /* Release device context */
    m_Tab.SetFont(&m_Font, FALSE);       /* Set tab control font */
```

```
    /* We are using new features */
    m_Tab.SetVersion(SFTTABS_2_1);

    index = m_Tab.AddTab(_T("&Listbox"));
    m_Tab.SetToolTip(index, _T("ToolTip for the ListBox tab"));
    m_Tab.SetTabInfo(index, &Tab0);
    m_Tab.SetTabWindowPage(index, &m_ListBox); /* tab page */

    index = m_Tab.AddTab(_T("&Edit Control"));
    m_Tab.SetToolTip(index, _T("ToolTip for the Edit Control tab"));
    m_Tab.SetTabInfo(index, &Tab1);
    m_Tab.SetTabWindowPage(index, &m_Edit); /* tab page */

    index = m_Tab.AddTab(_T("&Other Listbox"));
    m_Tab.SetToolTip(index, _T("ToolTip for the Other ListBox tab"));
    m_Tab.SetTabInfo(index, &Tab2);
    m_Tab.SetTabWindowPage(index, &m_OtherListBox); /* tab page */

    m_Tab.SetControlInfo(&CtlInit);

    // Make sure to turn redraw back on
    m_Tab.SetRedraw(TRUE);
    m_Tab.InvalidateRect(NULL, TRUE);

    // If you are not using the sheet/page classes, remove the call ...
#if defined(TAB_CONTROL_WITH_CLIENTAREA)
    // Initialize tab control
    InitializeTabControl(this, 0, &m_Tab, NULL);
#else
    // Initialize tab control.  An invisible, disabled frame window is used ...
    InitializeTabControl(this, 0, &m_Tab, &m_Frame);
#endif

    // Mark the view as a main, tabbed window (so accel. keys work) by registering it.
    SftTabs_RegisterWindow(m_hWnd);

    return 0;
}
```

Each call to CSftTabs::SetTabWindowPage associates a CSftTabsWindowPage based window with a tab.

The call to CSftTabsWindowSheet::InitializeTabControl starts the tabbed window handling and creates the current page.

Cleanup of all resources is accomplished by the call to CSftTabsWindowSheet::TerminateTabControl.

```
void CSampleView::OnDestroy()
{
    // Remove all pages from the tab control
    TerminateTabControl(this, &m_Tab);
    // Unregister, or the window properties used won't be removed
    SftTabs_UnregisterWindow(m_hWnd);

    CView::OnDestroy();
```

The following WM_SIZE message handler OnSize resizes the tabbed window's child windows.

```
void CSampleView::OnSize(UINT nType, int cx, int cy)
{
    CView::OnSize(nType, cx, cy);

    // get frame window dimension
    RECT rect;
    GetClientRect(&rect);

    // Reposition the gap window, just to cover the 5 pixels above the tab control.
```

```
    m_Gap.MoveWindow(0, 0, rect.right-rect.left, 5, TRUE);

#if defined(TAB_CONTROL_WITH_CLIENTAREA)
    // Use the whole space if the tab control has a client area
    // Reposition the tab control, this will also resize any attached dialog(s)
    m_Tab.MoveWindow(0, 5, rect.right-rect.left, rect.bottom-rect.top-5, TRUE);

#else
    // There is no client area
    // Use the space that is left over for the frame window

    SFTTABS_CONTROL Ctl;
    // Get tab control info
    m_Tab.GetControlInfo(&Ctl);

    // Reposition the tab control
    m_Tab.MoveWindow(0, 5, rect.right-rect.left, Ctl.naturalSize, TRUE);

    // Reposition the frame window
    // Ctl.naturalSize has best height for this tab control
    m_Frame.MoveWindow(0, 5+Ctl.naturalSize, rect.right-rect.left,
                            rect.bottom-rect.top-Ctl.naturalSize-5, TRUE);
    m_Tab.ResizePages();                // let the tab control know to resize all
pages

#endif
}
```

As the main tabbed window receives notifications from the tab control that the user is switching to another page, it has to handle these notifications using the CSftTabsWindowSheet::TabSwitching and CSftTabsWindowSheet::TabSwitched member functions.

By calling the CSftTabsWindowSheet::TabSwitching member function, the class implementation then calls CSftTabsWindowPage::AllowSwitch to determine if the current page can be left, giving the application the opportunity to cancel tab switching.

```
// Switching away from the current tab

void CSampleView::OnTabSwitching()
{
    // You have to handle this event, so the tab pages are switched
    TabSwitching(this, &m_Tab);
}

// Switching to a new tab

void CSampleView::OnTabSwitched()
{
    // You have to handle this event, so the tab pages are switched
    TabSwitched(this, &m_Tab);
}
```

### Creating a Page

Most CWnd based classes are suitable to become a page in a tabbed window. By using multiple inheritance, a class can be used as a page by inheriting the required support from the class CSftTabsWindowPage.

```
    class CSampleListBox : public CListBox, public CSftTabsWindowPage
    {
        ... class definitions
    }
```

The following code sample (from C:\SFTTABS\SAMPLES\MFCSAM2\LISTBOX.CPP) shows a typical implementation of a page. Most of the code has been created using SftTabs/DLL Wizard.

```
// create/attach window

BOOL CSampleListBox::ActivatePage(CWnd* pParent, CSftTabs* pTabCtl)
{
    if (!m_hWnd) {
        // The window doesn't exist, create it now
        if (!Create(WS_CHILD|                    /* Window Style */
                WS_TABSTOP|
                LBS_NOTIFY|LBS_NOINTEGRALHEIGHT,
                CRect(0,0,0,0),               /* location */
                pParent,                      /* Parent Window */
                IDC_LIST))                    /* control ID */
            return FALSE;
        AddString(_T("Item 1"));
        AddString(_T("Item 2"));
        AddString(_T("Item 3"));
        AddString(_T("Item 4"));
        AddString(_T("Item 5"));
        AddString(_T("Item 6"));
        AddString(_T("This is a listbox."));
        AddString(_T("Click a tab or use Alt-xxx to"));
        AddString(_T("switch to another tab."));
        SetCurSel(0);
    } else {
        // The user switched back to this page
    }

    // This page is now active
    SftTabs_SetPageActive(m_hWnd, pTabCtl->m_hWnd, NULL);
    // Enable + show it, its size is 0,0,0,0, it will be resized by the tab control
    EnableWindow(TRUE);
    ShowWindow(SW_SHOW);

    return TRUE;
}

// destroy/detach window

void CSampleListBox::DeactivatePage(CWnd* pParent, CSftTabs* pTabCtl, BOOL fFinal)
{
    if (fFinal)
        DestroyWindow();
    else {
        // hide the page
        ShowWindow(SW_HIDE);
        EnableWindow(FALSE);
    }
    // clear associated page in tab's control structure
    SftTabs_SetPageInactive(pTabCtl->m_hWnd);
}
```

The CSftTabsWindowPage::ActivatePage and CSftTabsWindowPage::DeactivatePage member functions must be implemented by a page. SftTabs/DLL Wizard generates the required sample code.

These functions allow the application to do initialization and termination processing for each page.

# C++/OWL Programming

This section describes how to use SftTabs/DLL with an application written using C++ and the Borland ObjectWindows Library (OWL).

## Building an Application

A) Every source program making use of a SftTabs/DLL control must include the required header file SFTTB.H by using the #include directive.

```
#include "sfttb.h"                      /* SftTabs/DLL required header file */
```

This include statement should appear after any OWL- and Windows-related #include statements. The file is located in the directory C:\SFTTABS\INCLUDE (unless changed during the installation).

B) One source program must include the TSftTabs class implementation, using the #include directive.

```
#include "sfttbb.cpp"                   /* SftTabs/DLL implementation */
```

This include statement should appear after the #include "sfttb.h" statement. This is the preferred method to include the implementation of the TSftTabs class. Adding the file SFTTBB.CPP to your project is not recommended because it will complicate the use of pre-compiled header files. The file is located in the directory C:\SFTTABS\INCLUDE (unless changed during the installation).

C) In order to use SftTabs/DLL controls, an application must call the TSftTabs::RegisterApp function. The call to this function is required so that SftTabs/DLL window classes can be registered. This call has to be made before any SftTabs/DLL controls are created. Add the following statement to your source code, the preferred location is the InitInstance member function of your TApplication based application object:

```
TSftTabs::RegisterApp();            /* Use SftTabs/DLL with this application
*/
```

D) Once SftTabs/DLL controls are no longer needed, an application must call the TSftTabs::UnregisterApp function. The call to this function is required so that SftTabs/DLL window classes can be unregistered and cleanup processing can take place. This call has to be made after all SftTabs/DLL controls have been destroyed. The preferred location is the TermInstance member function of your TApplication based application object:

```
TSftTabs::UnregisterApp();          /* No longer use SftTabs/DLL */
```

E) The application's executable (EXE or DLL) must be linked with the correct LIB file, depending on the target environment. The DLL must be available and accessible at run-time for proper execution. The DLL used at run-time depends on the LIB file used at link time.

| Target Environment | LIB File Required when Linking | DLL File Required at Run-Time |
|---|---|---|
| **Windows 3.1, 16-bit applications** | SFTTB.LIB | SFTTB.DLL |
| **WIN32, all 32-bit environments including Windows NT (without UNICODE support** | SFTTB32B.LIB | SFTTB32.DLL |
| **Windows NT only** | SFTTB32V.LIB | SFTTB32U.DLL |

**(with UNICODE support**

All required files can be found in the directories C:\SFTTABS\LIB and C:\SFTTABS\BIN, unless changed during the installation.

## Adding a Tab Control

ClassExpert does not support new classes such as TSftTabs, so any tab control instance variables, notification handlers, message map entries, etc., have to be added manually. To simplify this process, you can copy these items that are generated by ClassExpert for other "standard" Windows controls (such as a list box).

There are two methods to add a tab control to an application:

- using dialog resources
- using the TWindow::Create function

Adding a tab control using dialog resources is accomplished by using a resource editor to design a dialog. For more information on the different resource editors supported by SftTabs/DLL, see Resource Workshop, Borland C++, AppStudio, Visual C++ and Dialog Editor (Windows SDK). Once a tab control is created by creating the dialog, the TSftTabs based object can be constructed by using the TSftTabs constructor.

```
pTab = new TSftTabs(this, IDC_TAB);
```

Another method to create a tab control is by using the TSftTabs constructor and the TWindow::Create function:

```
pTab = new TSftTabs(parentWindow, IDC_TAB, 250,200,400,400);
pTab->Create();
```

The constructor creates the tab control object. The arguments define the position of the tab control window once it is created using the Create function.

## Handling Notifications

As with standard Windows controls, applications must respond to events and messages to cause controls to respond to user requests. For additional information see Notifications.

ClassExpert does not support new classes such as TSftTabs, so any tab control instance variables, notification handlers, message map entries, etc., have to be added manually. To simplify this process, you can copy these items that are generated by ClassExpert for other "standard" Windows controls.

### Switching Between Tabs

Switching between tabs is fully automatic, however, an application may wish to prevent a user from switching to another tab. By responding to the WM_COMMAND, SFTTABSN_SWITCHING notification, an application can prevent completion of the tab switch by sending a WM_CANCELMODE message to the tab control.

```
// Event handler prototype added to dialog/window class
void EvTabSwitching();

// Response table
DEFINE_RESPONSE_TABLE1(TMainWin, TLayoutWindow)
    EV_SFTTABSN_SWITCHING(IDC_TAB, EvTabSwitching),
END_RESPONSE_TABLE;

// Event handler implementation
void TMainWin::EvTabSwitching()
{
    if (YouDontLikeThisUser())
        pTabCtl->SendMessage(WM_CANCELMODE);// cancel switching
```

```
}
```

An application has to make attached controls or dialogs visible when switching between tabs. The SftTabs/DLL API offers functions to manage dialogs and Windows controls that are attached to tabs. See Implementing Tabbed Dialogs and Implementing Tabbed Windows for more information.

## 3D and Colors

SftTabs/DLL offers many tab control styles which look best on a gray background. For dialogs, the gray background can be achieved using the SoftelGrayDialog or SoftelGrayDialog32 window class or using one of the following methods:

### CTL3DV2 or CTL3D32

The tab control can be used with CTL3DV2 (or CTL3D32). Any dialogs attached to a tab control can use the 3D display, if properly enabled (usually using Ctl3dAutoSubclass). For more information on CTL3DV2 or CTL3D32, see the Microsoft documentation. For more information on CTL3DV2 or CTL3D32, see the documentation supplied with Borland C++.

### WM_CTLCOLOR, WM_CTLCOLORSTATIC

| This message is generated by the tab control for compatibility with SftTabs 2.0 only. When developing new applications, please use TSftTabs::SetCtlColors instead. |
|---|

The appearance of the tab control can be modified by handling the WM_CTLCOLOR message. The parent window can override the background color used by the tab control by defining a WM_CTLCOLOR message handler.

```
// Pointer to tab object added to dialog class
TSftTabs* pTab;
// Event handler definition added to dialog class
HBRUSH EvCtlColorTab (HDC dc, HWND hWndChild, uint ctlType);
// Event handler added to parent window response table
EV_WM_CTLCOLOR,

// Event handler implementation
HBRUSH TYourDialog::EvCtlColorTab(HDC dc, HWND hWndChild, uint ctlType)
{
    if (ctlType == CTLCOLOR_STATIC) {
        ::SetBkColor(dc, RGB(192,192,192));      // on gray background
        return (HBRUSH)::GetStockObject(LTGRAY_BRUSH);
    }
}
```

## Implementing Tabbed Dialogs

A tabbed dialog is created just like any other dialog. A tabbed dialog has a tab control with an available client area. In this client area, pages are displayed. Each tab has an attached page (although during development of a tabbed dialog, a tab doesn't require an attached page). As the user switches between tabs, the appropriate page is created, displayed and destroyed.

A tabbed dialog and each page are based on the class TSftTabsDialog and TSftTabsPage, which are both derived from the OWL class TDialog. This makes conversion of existing dialogs and development of new pages very easy. Tabbed dialogs and pages are first designed using a resource editor. The sections Resource Workshop, Borland C++, AppStudio, Visual C++ and Dialog Editor (Windows SDK) describe how the necessary tabbed dialog and page (dialog) resources are created.

ClassExpert can be used to create the dialogs and pages. However, ClassExpert can only create new classes based on TDialog (not TSftTabsDialog or TSftTabsPage). After ClassExpert generates a new class derived from TDialog, you have to manually change references to TDialog to the classes TSftTabsDialog and TSftTabsPage. When adding new

member functions, make sure to call the TSftTabsDialog and TSftTabsPage base classes instead of TDialog. The Rescan function of the Borland IDE no longer recognizes classes which have been manually edited to TSftTabsDialog and TSftTabsPage. The samples included with SftTabs/DLL show how using '#define TDialog TSftTabsDialog' and '#undef TDialog' in the header file of a dialog can bypass this problem. See MAINDLG.H and PAGE1.H of the sample found in the directory C:\SFTTABS\SAMPLES\OWLSAM1.

Once the necessary dialogs have been designed, the tab control layout can be defined using the SftTabs/DLL Wizard. The SftTabs/DLL Wizard also creates much of the code required to initialize the tab control. This code should be copied to the application (with possibly minor modifications).

### Creating a Tabbed Dialog

The following code sample (from C:\SFTTABS\SAMPLES\OWLSAM1\MAINDLG.CPP) shows a typical implementation of a tabbed dialog. Most of the code has been created using the SftTabs/DLL Wizard and then copied into the application. The code that was copied is marked by a vertical line on the side. The majority of the code is used to initialize the tab control.

```
/*- Tab Control Initialization Data -----------------------------------------*/

static const SFTTABS_CONTROL CtlInit = {
    SFTTABSSTYLE_MODERN_I,              /* tab style */
    2,                                  /* number of rows */
    0,                                  /* number of tabs per row (if fFixed) */
    0,                                  /* width of left margin */
    0,                                  /* width of right margin */
    FALSE,                              /* same width for all tabs */
    TRUE,                               /* Client area wanted */
    FALSE,                              /* allow multiline label text */
    TRUE,                               /* use with dialog */
    FALSE,                              /* use specified background color only for
text */
    FALSE,                              /* scrollable tabs */
    FALSE,                              /* hide scroll buttons */
    TRUE,                               /* bold font for active tab wanted */
    TRUE,                               /* fill rows completely */
    NULL,                               /* scroll button bitmap */
    NULL,                               /* Dialog data associated with active tab */
    NULL,                               /* Dialog window handle associated with ...
    NULL,                               /* Frame, used as client area */
    TRUE,                               /* Tooltips wanted */
    FALSE,                              /* drop text if it doesn't fit */
    FALSE,                              /* conditional scroll buttons */
};

static const SFTTABS_TAB Tab0 = {   /*The First One */
    SFTTABS_NOCOLOR, RGB(0,0,255),      /* background, foreground color */
    SFTTABS_NOCOLOR, RGB(0,0,255),      /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_LEFT, SFTTABS_GRAPH_BITMAP },
                                        /* Bitmap, Icon */
    TRUE,                               /* enabled/disabled */
    0                                   /* userdata */
};

static const SFTTABS_TAB Tab1 = {   /*&Second */
    SFTTABS_NOCOLOR, RGB(255,0,0),      /* background, foreground color */
    SFTTABS_NOCOLOR, RGB(255,0,0),      /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_RIGHT, SFTTABS_GRAPH_ICON },
                                        /* Bitmap, Icon */
    TRUE,                               /* enabled/disabled */
```

```
    0                                   /* userdata */
};

static const SFTTABS_TAB Tab2 = {   /*&Third */
    SFTTABS_NOCOLOR, RGB(128,128,0),    /* background, foreground color */
    SFTTABS_NOCOLOR, RGB(128,128,0),    /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },          /* Bitmap, Icon */
    TRUE,                               /* enabled/disabled */
    0                                   /* userdata */
};

static const SFTTABS_TAB Tab3 = {   /*F&ourth */
    SFTTABS_NOCOLOR, RGB(0,255,255),    /* background, foreground color */
    SFTTABS_NOCOLOR, RGB(0,255,255),    /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },          /* Bitmap, Icon */
    TRUE,                               /* enabled/disabled */
    0                                   /* userdata */
};

static const SFTTABS_TAB Tab4 = {   /*F&ifth */
    SFTTABS_NOCOLOR, RGB(0,0,128),      /* background, foreground color */
    SFTTABS_NOCOLOR, RGB(0,0,128),      /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },          /* Bitmap, Icon */
    TRUE,                               /* enabled/disabled */
    0                                   /* userdata */
};

static const SFTTABS_TAB Tab5 = {   /*Si&xth */
    SFTTABS_NOCOLOR, RGB(128,0,0),      /* background, foreground color */
    SFTTABS_NOCOLOR, RGB(128,0,0),      /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },          /* Bitmap, Icon */
    TRUE,                               /* enabled/disabled */
    0                                   /* userdata */
};
```

The EvInitDialog message handler of the tabbed dialog initializes the tab control and associates TSftTabsPage based objects to each tab.

```
bool TMainDialog::EvInitDialog (HWND hWndFocus)
{
    TSftTabsDialog::EvInitDialog(hWndFocus);

    int index;
    SFTTABS_TAB Tab;

    /* Initialization is faster if we set redraw off */
    pTab->SetRedraw(false);

    /* We are using new features */
    pTab->SetVersion(SFTTABS_2_1);

    index = pTab->AddTab(TEXT("The First One"));
    pTab->SetToolTip(index, TEXT("Demonstrates tabbing into and out of the tab
page"));
    Tab = Tab0;
    Tab.graph.item.hBitmap = *m_pSampleBitmap;
    pTab->SetTabInfo(index, &Tab);
    pTab->SetTabDialog(index, new TPage1(this));

    index = pTab->AddTab(TEXT("&Second"));
    pTab->SetToolTip(index, TEXT("Demonstrates how an application can ...
    Tab = Tab1;
    Tab.graph.item.hIcon = *m_pSampleIcon;
    pTab->SetTabInfo(index, &Tab);
```

```
    pTab->SetTabDialog(index, new TPage2(this));

    index = pTab->AddTab(TEXT("&Third"));
    pTab->SetToolTip(index, TEXT("This page is reset everytime you switch to it"));
    pTab->SetTabInfo(index, &Tab2);
    pTab->SetTabDialog(index, new TPage3(this));

    index = pTab->AddTab(TEXT("F&ourth"));
    pTab->SetToolTip(index, TEXT("A page with private OK, Cancel, Next and ...
    pTab->SetTabInfo(index, &Tab3);
    pTab->SetTabDialog(index, new TPage4(this));

    index = pTab->AddTab(TEXT("F&ifth"));
    pTab->SetToolTip(index, TEXT("A page that has not yet been implemented"));
    pTab->SetTabInfo(index, &Tab4);
    // no page attached

    index = pTab->AddTab(TEXT("Si&xth"));
    pTab->SetToolTip(index, TEXT("A page with nested tab controls and pages"));
    pTab->SetTabInfo(index, &Tab5);
    pTab->SetTabDialog(index, new TPage6(this));

    pTab->SetControlInfo(&CtlInit);

    // Make sure to turn redraw back on
    pTab->SetRedraw(true);
    pTab->Invalidate(true);

    // If you are not using the sheet/page classes, remove the ...
    // Initialize tab control
    InitializeTabControl(0, pTab, NULL);
    return false;
}
```

Each call to TSftTabs::SetTabDialog associates a TSftTabsPage based dialog to a tab. Cleanup of all resources, including the dynamically allocated pages is done automatically when the tabbed dialog is destroyed.

The call to TSftTabsDialog::InitializeTabControl starts the tabbed dialog handling and creates the current page.

A tabbed dialog should always return FALSE from the EvInitDialog member function. The input focus has already been set by SftTabs/DLL, so returning FALSE will prevent Windows from setting the focus (to the wrong control).

### Creating a Page

The implementation of a page is identical to a regular dialog, except that the base class is TSftTabsPage instead of TDialog.

## Implementing Tabbed Windows

A tabbed window is created just like a regular window. A tabbed window has at least one tab control as its child window (with or without a client area). Each tab has an attached page (although during development of a tabbed window, a tab doesn't require an attached page). As the user switches between tabs, the appropriate page is created, displayed and destroyed.

A tabbed window and each page are based on the class TWindow or any of its derived classes, such as TListBox, TEdit, etc. Using multiple inheritance, a window can inherit the required support to make into a tabbed window or a page. This makes conversion of existing windows and development of new pages very easy.

ClassExpert can be used to create the tabbed window and the pages initially. By using multiple inheritance, the classes TSftTabsWindowSheet and TSftTabsWindowPage are used to add tabbed window and page support to the new classes.

### Creating a Tabbed Window

Most TWindow based classes are suitable to be used as a tabbed window. By using multiple inheritance, a class can be used as a tabbed window by inheriting the required support from the class TSftTabsWindowSheet.

```
    class TMainWin : public TLayoutWindow, public TSftTabsWindowSheet {
        ... class definitions
    };
```

The following code sample (from C:\SFTTABS\SAMPLES\OWLSAM2\MAINWIN.CPP) shows a typical implementation of a tabbed window. Most of the code has been created using the SftTabs/DLL Wizard and then copied into the application.

```
/*- Tab Control Initialization Data ----------------------------------------*/

static const SFTTABS_CONTROL CtlInit = {
    SFTTABSSTYLE_SIMPLE,                /* tab style */
    1,                                  /* number of rows */
    0,                                  /* number of tabs per row (if fFixed) */
    0,                                  /* width of left margin */
    0,                                  /* width of right margin */
    FALSE,                              /* same width for all tabs */
#if defined(TAB_CONTROL_WITH_CLIENTAREA)
    TRUE,                               /* Client area wanted */
#else
    FALSE,                              /* Client area not wanted */
#endif
    FALSE,                              /* allow multiline label text */
    FALSE,                              /* use with dialog */
    FALSE,                              /* use specified background color only for
text */
    TRUE,                               /* scrollable tabs */
    FALSE,                              /* hide scroll buttons */
    TRUE,                               /* bold font for active tab wanted */
    FALSE,                              /* fill rows completely */
    NULL,                               /* scroll button bitmap */
    NULL,                               /* Dialog data associated with active tab */
    NULL,                               /* Dialog window handle associated with ...
    NULL,                               /* Frame, used as client area */
    TRUE,                               /* Tooltips wanted */
    FALSE,                              /* drop text if it doesn't fit */
    TRUE,                               /* conditional scroll buttons */
};

static const SFTTABS_TAB Tab0 = {   /*&Listbox */
    RGB(0,0,255), RGB(0,255,255),       /* background, foreground color */
    RGB(0,0,255), RGB(0,255,255),       /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },          /* Bitmap, Icon */
    TRUE,                               /* enabled/disabled */
    0                                   /* userdata */
};

static const SFTTABS_TAB Tab1 = {   /*&Edit Control */
    RGB(255,255,0), RGB(0,0,0),         /* background, foreground color */
    RGB(255,255,0), RGB(0,0,0),         /* background, foreground color (when
selected) */
    { SFTTABS_GRAPH_NONE, 0 },          /* Bitmap, Icon */
    TRUE,                               /* enabled/disabled */
    0                                   /* userdata */
};

static const SFTTABS_TAB Tab2 = {   /*&Dialog */
    RGB(255,0,0), RGB(255,255,0),       /* background, foreground color */
    RGB(255,0,0), RGB(255,255,0),       /* background, foreground color (when
```

```
selected) */
    { SFTTABS_GRAPH_NONE, 0 },          /* Bitmap, Icon */
    TRUE,                               /* enabled/disabled */
    0                                   /* userdata */
};
```

The EvCreate message handler of the tabbed window initializes the tab control and associates TSftTabsWindowPage based objects to each tab.

```
int TMainWin::EvCreate (CREATESTRUCT far& createStruct)
{
    if (TLayoutWindow::EvCreate(createStruct) != 0)
        return -1;

    // Create the tab control
    pTab = new TSftTabs(this,           // 'this' is the parent window
        IDC_TAB,                        // tab control ID
        0, 0,                           /* x, y */
        0, 0);                          /* width, height */
    pTab->Attr.Style |= WS_CLIPCHILDREN | WS_TABSTOP | WS_GROUP |
        WS_VISIBLE | WS_CHILD;          // Visible, child window
    if (!pTab->Create())
        return -1;

#if !defined(TAB_CONTROL_WITH_CLIENTAREA)
    // Create the frame window (which will hold the pages)
    m_pFrame = new TStatic(this, -1, TEXT(""), 0, 0, 0, 0);
    m_pFrame->Attr.Style &= ~WS_VISIBLE;     // not visible
    m_pFrame->Attr.Style |= WS_DISABLED;     //  and disabled
    // Create a static control that we can use as a frame window for the tab control's
    // pages.  This window is not visible and is just used to indicate the page
position
    if (!m_pFrame->Create())
        return -1;
#endif

    int index;

    /* Initialization is faster if we set redraw off */
    pTab->SetRedraw(false);

    /* We are using new features */
    pTab->SetVersion(SFTTABS_2_1);

    index = pTab->AddTab(TEXT("&Listbox"));
    pTab->SetToolTip(index, TEXT("A standard listbox is attached to this tab"));
    pTab->SetTabInfo(index, &Tab0);
    // If you don't want to attach a page to the tab, the following is optional
    pTab->SetTabWindowPage(index, m_pList); // tab page

    index = pTab->AddTab(TEXT("&Edit Control"));
    pTab->SetToolTip(index, TEXT("A standard edit control is attached to this tab"));
    pTab->SetTabInfo(index, &Tab1);
    pTab->SetTabWindowPage(index, m_pEdit); // tab page

    index = pTab->AddTab(TEXT("&Dialog"));
    pTab->SetToolTip(index, TEXT("A dialog is attached to this tab"));
    pTab->SetTabInfo(index, &Tab2);
    pTab->SetTabWindowPage(index, m_pDlg); // tab page

    pTab->SetControlInfo(&CtlInit);

    // Make sure to turn redraw back on
    pTab->SetRedraw(true);
    pTab->Invalidate(true);

    // If you are not using the sheet/page classes, remove the call ...
#if defined(TAB_CONTROL_WITH_CLIENTAREA)
```

```
    // Initialize tab control
    InitializeTabControl(this, 0, pTab, NULL);
#else
    // Initialize tab control.  An invisible, disabled frame window is used to ...
    InitializeTabControl(this, 0, pTab, m_pFrame);
#endif
    // Mark the view as a main, tabbed window (so accel. keys work) by registering it.
    SftTabs_RegisterWindow(HWindow);

    return 0;
}
```

Each call to TSftTabs::SetTabWindowPage associates a TSftTabsWindowPage based window with a tab.

The call to TSftTabsWindowSheet::InitializeTabControl starts the tabbed window handling and creates the current page.

Cleanup of all resources is accomplished by the call to TSftTabsWindowSheet::TerminateTabControl.

```
void TMainWin::EvDestroy ()
{
    TLayoutWindow::EvDestroy();

    // Remove all pages from the tab control
    TerminateTabControl(this, pTab);
    // Unregister, or the window properties used won't be removed
    SftTabs_UnregisterWindow(HWindow);
}
```

The following WM_SIZE message handler EvSize resizes the tabbed window's child windows.

```
void TMainWin::EvSize (uint sizeType, TSize& size)
{
    TLayoutWindow::EvSize(sizeType, size);

    TRect rect;
    // resize all child windows.  You could also use TLayoutMetrics and Layout().
    // Here we "manually" calculate the window sizes.

    // get frame window dimension
    rect = GetClientRect();

#if defined(TAB_CONTROL_WITH_CLIENTAREA)
    // Use the whole space if the tab control has a client area
    // Reposition the tab control, this will also resize any attached dialog(s)
    pTab->MoveWindow(0, 0, rect.right-rect.left, rect.bottom, TRUE);

#else
    // There is no client area
    // Use the space that is left over for the frame window

    // calculate position of tab control
    SFTTABS_CONTROL Ctl;
    // Get tab control info
    pTab->GetControlInfo(&Ctl);

    // Reposition the tab control
    pTab->MoveWindow(0, 0, rect.right-rect.left, Ctl.naturalSize, TRUE);

    // Reposition the frame window
    // Ctl.naturalSize has best height for this tab control
    m_pFrame->MoveWindow(0, Ctl.naturalSize, rect.right-rect.left,
                                    rect.bottom-rect.top-Ctl.naturalSize, TRUE);
    pTab->ResizePages();                // let the tab control know to resize all
pages
#endif
}
```

As the main tabbed window receives notifications from the tab control that the user is switching to another page, it has to handle these notifications using the TSftTabsWindowSheet::TabSwitching and TSftTabsWindowSheet::TabSwitched member functions.

By calling the TSftTabsWindowSheet::TabSwitching member function, the class implementation then calls TSftTabsWindowPage::AllowSwitch to determine if the current page can be left, giving the application the opportunity to cancel tab switching.

```
void TMainWin::EvTabSwitching()
{
    TabSwitching(this, pTab);
}

void TMainWin::EvTabSwitched()
{
    TabSwitched(this, pTab);
}
```

### Creating a Page

Most TWindow based classes are suitable to become a page in a tabbed window. By using multiple inheritance, a class can be used as a page by inheriting the required support from the class TSftTabsWindowPage.

```
    class TSampleList : public TListBox, public TSftTabsWindowPage {
        ... class definitions
    }
```

As a page is allocated by the tabbed window, it is important that TWindow::DisableAutoCreate is called for each page before the tabbed window is created. Without the call to DisableAutoCreate, all pages would be created automatically as soon as the tabbed window is created, which interferes with the processing offered by SftTabs/DLL and the C++ class implementations.

```
TMainWin::TMainWin (TWindow* parent, const char far* title, TModule* module):
    TLayoutWindow(parent, title, module)
{
    pTab = NULL;                          // tab control

    // Construct an edit control (one of the pages)
    m_pEdit = new TSampleEdit (this, IDC_EDIT, TEXT(""), 0, 0, 0, 0, 0, true);
    // Make sure the control/window isn't automatically created when the parent is
created
    m_pEdit->DisableAutoCreate();

    // Construct a listbox
    m_pList = new TSampleList(this, IDC_LIST, 0, 0, 0, 0, 0);
    // Make sure the control/window isn't automatically created when the parent is
created.
    m_pList->DisableAutoCreate();

    // Construct a dialog
    m_pDlg = new TLastDialog(this);
    // Make sure the control/window isn't automatically created when the parent is
created.
    m_pDlg->DisableAutoCreate();
}
```

The following code sample (from C:\SFTTABS\SAMPLES\OWLSAM2\LISTBOX.CPP) shows a typical implementation of a page. Most of the code has been created using SftTabs/DLL Wizard.

```
bool TSampleList::ActivatePage(TWindow* pParent, TSftTabs* pTabCtl)
{
    // This is called when the user switches to a page
```

```
    if (!HWindow) {
        // The window doesn't exist, create it now. Make sure it's NOT VISIBLE.
        // You can modify this to create another type of window instead.
        Attr.Style &= ~(WS_BORDER|WS_VISIBLE); // turn these off
        // you may need to add/remove additional styles
        Attr.Style |= WS_TABSTOP; // turn these styles on
        if (!Create())
            return false;
        // Additional initialization if desired
        AddString("Item 1");
        AddString("Item 2");
        AddString("Item 3");
        AddString("Item 4");
        AddString("Item 5");
        AddString("Item 6");
        AddString("This is a listbox.");
        AddString("Click a tab or use Alt-xxx to");
        AddString("switch to another tab.");
        SetSelIndex(0);
    } else {
        // The user switched back to this page
    }

    // This page is now active
    SftTabs_SetPageActive(HWindow, pTabCtl->HWindow, NULL);
    // Enable + show it, its size is 0,0,0,0, it will be resized by the tab control
    EnableWindow(true);
    ShowWindow(SW_SHOW);

    return true;
}

void TSampleList::DeactivatePage(TWindow* pParent, TSftTabs* pTabCtl, bool fFinal)
{
    if (fFinal) {
        // You must destroy the window, the tabbed window (parent) is going away
        Destroy();
    } else {
        // Hide the page. If you want, you could use Destroy here too.
        // In that case you save resources and the window will be recreated
        // when the user switches back to this page
        ShowWindow(SW_HIDE);
        EnableWindow(false);
    }
    // clear associated page in tab's control structure
    SftTabs_SetPageInactive(pTabCtl->HWindow);
}
```

The TSftTabsWindowPage::ActivatePage and TSftTabsWindowPage::DeactivatePage member functions must be implemented by a page. SftTabs/DLL Wizard generates the required sample code.

These functions allow the application to do initialization and termination processing for each page.

# Resource Workshop

Resource Workshop is part of Borland C++ 4.5. If you are using Borland C++ 5.0, see section Borland C++ for more information.

## First Time

In order to make SftTabs/DLL available to Resource Workshop, use its *File, Install control library...* menu command to define SftTabs/DLL to Resource Workshop. This has to be done only once. Locate the DLL using the dialog shown. The file SFTTB.DLL can be found in the directory C:\SFTTABS\BIN (unless changed during the installation). Do not install the 32-bit version of the DLL. Borland C++ Resource Workshop does not support 32-bit custom control DLLs, even when running in a 32-bit environment. By installing the 16-bit version of SftTabs/DLL, Resource Workshop will be able to display and modify SftTabs/DLL attributes as expected. The resource script can be compiled using 32-bit (or 16-bit) tools and linked with the appropriate 32-bit or 16-bit version of SftTabs/DLL.

## New Project

Whenever you create a project which is to include a SftTabs/DLL control, make sure to add the C and C++ header file SFTTB.H to your project. This file can be found in the directory C:\SFTTABS\INCLUDE (unless changed during the installation). Use the *File, Add to project...* menu command to display the *Add file to project* dialog. Adding the SftTabs/DLL header file insures that your resource definitions for the SftTabs/DLL control can be compiled correctly. The SFTTB.H header file has to be accessible to Resource Workshop and the resource compiler. If the header file is not added to your project you will get the error message "Resource Workshop 197: Compile Error, Expecting control window style" when editing a dialog containing a SftTabs/DLL control.

## Adding a Tab Control to a Dialog

To add a SftTabs/DLL control to a dialog, use the SftTabs/DLL toolbar button. Click on the button and then on the dialog being designed to add a control. Once a SftTabs/DLL control has been added to a dialog, you can edit the window styles by double-clicking anywhere within the control, or by using the *Control, Style...* menu command. This dialog can only be used to manipulate a few very basic styles. More styles are available through the C and C++ API.

## SftTabs/DLL Control Styles Dialog

The *SftTabs/DLL Styles* dialog allows you to manipulate the following tab control attributes:

| Item | Description |
|---|---|
| Control ID | Enter the control's identifier in the *Control ID* input box. Control IDs can be a short integer such as 201, or an integer expression, such as IDC_TABS=201. In both cases the value 201 is assigned to the control as control ID, the second example also defines IDC_TABS as an alphanumeric identifier. If you enter an alphanumeric identifier, Resource Workshop checks to see if a #define or a constant declaration has already been created for that identifier. If not, Resource Workshop will create the identifier. |
| Visible | The *Visible* check box determines whether the control is visible when the dialog box is first displayed. If the option is not checked, the control does not appear. The application can call the ShowWindow function at run-time to make the control appear. Equivalent to the WS_VISIBLE style. |
| Disabled | The *Disabled* check box disables the control by graying it. This prevents the control from responding to user input. Equivalent to the WS_DISABLED style. |
| Border | Turn the *Border* check box on to draw a border around the control. The border is a dark line. Equivalent to the WS_BORDER style. |

| | |
|---|---|
| Group | Turn the *Group* check box on to indicate the first control within a group of controls. The user can then press the arrow keys to access all controls in the group. Equivalent to the WS_GROUP style. |
| Tab Stop | Turn the *Tab Stop* check box on if you want the user to be able to press Tab to access this control. Equivalent to the WS_TABSTOP style. |
| OK | Click the *OK* button to accept all style settings and end the *SftTabs/DLL Styles* dialog. |
| Cancel | Click the *Cancel* button to abandon all (modified) style settings and end the *SftTabs/DLL Styles* dialog. |
| Help | Click the *Help* button for on-line help information on the *SftTabs/DLL Styles* dialog. |
| Design | Click Click the *Design* button to open or create a tab layout file (*.TAB). This invokes the SftTabs/DLL Wizard application, which allows you to define a tab layout and generates the required C or C++ run-time source code. |

## Designing a Tabbed Dialog

A tabbed dialog is designed just like a regular dialog. The only difference is the tab control, which is added to make it a tabbed dialog. The tab control has to be designed using the SftTabs/DLL Wizard application and the resulting source code can then be copied to your application.

A tab control used with a tabbed dialog must provide a client area or a frame window has to be defined when calling SftTabs_SetControlInfo, CSftTabsDialog::InitializeTabControl or TSftTabsDialog::InitializeTabControl.

The size of the tab control's client area must be large enough to accommodate all pages. Differences in font sizes and display resolutions may cause problems if the client area is not large enough. A page will always be sized as large as the client area allows. To make tabbed dialog versus page size consistent, make sure to always specify the same font to be used for all dialogs.

## Designing a Page

A page is a modeless dialog which is placed in the tab control's client area.

The size of the tab control's client area must be large enough to accommodate a page. Differences in font sizes and display resolutions may cause problems if the client area is not large enough. A page will always be sized as large as the client area allows.

A tab page is designed just like a regular dialog with a few minor restrictions. A page is always a modeless dialog which has a tabbed dialog as its parent window. For that reason, the window style of a page has to be defined as "Child". The dialog should be defined without frame, border or other decoration. A system menu may be defined so a dialog caption can be entered. Both the system menu and the dialog caption will be removed by SftTabs before the page is shown. The page's caption will be used as the dialog caption for the main, tabbed dialog. This offers an easy way to change the tabbed dialog's window title when switching between pages.

To make tabbed dialog versus page size consistent, make sure to always specify the same font to be used for all dialogs.

## Test Mode

In the dialog test mode offered by Resource Workshop, a SftTabs/DLL control will be displayed in the location specified. The tab control will not reflect any settings you may have defined using the SftTabs/DLL Wizard application. The tab control shown will always be of the "Standard" style with four tabs, labeled 1,2,3 and 4.

# Borland C++

Borland C++ 5.x does not support custom control DLLs (Resource Workshop, shipped with Borland C++ 4.x fully supports custom controls). It is still possible to use SftTabs/DLL with Borland C++, but the easy design-time interface that is provided by other resource editors is not available.

## Adding a Tab Control to a Dialog

To add a SftTabs/DLL control to a dialog, use the *Dialog, Insert New Control* menu command. Enter the class **SoftelTabControl** (Windows 3.1) or **SoftelTabControl32** (for Windows NT, 95, Win32s) in the *New Control* dialog.

Once a custom control has been added to a dialog, you can edit the control properties by double-clicking anywhere within the control. A window caption is not necessary, so the edit field marked *Caption* can be left blank.

## SftTabs/DLL Control Styles

To enter a SftTabs/DLL window style in the *User Control Properties* dialog, use the following list to add the desired style values and enter the resulting hexadecimal value in the field marked *Style*. For detailed information, see Tab Control Styles.

**Style**

| | Value | Description |
|---|---|---|
| WS_BORDER | 0x00800000 | Draw a border around the control. The border is a dark line. |
| WS_CHILD | 0x40000000 | Create a child window. |
| WS_DISABLED | 0x08000000 | Create a tab control that is initially disabled. A disabled tab control cannot receive input from the user. |
| WS_GROUP | 0x00020000 | Specifies the first control of a group of controls. All controls defined with the WS_GROUP style after the first control belong to the same group. The next control with the WS_GROUP style ends the group and starts the next group. |
| WS_TABSTOP | 0x00010000 | Specifies a control that can receive the keyboard focus when the user presses the TAB key. Pressing the TAB key changes the keyboard focus to the next control with the WS_TABSTOP style. |
| WS_VISIBLE | 0x10000000 | Create a tab control that is initially visible. |

## Designing a Tabbed Dialog

A tabbed dialog is designed just like a regular dialog. The only difference is the tab control, which is added to make it a tabbed dialog. The tab control has to be designed using the SftTabs/DLL Wizard and the resulting source code can then be copied to your application..

A tab control used with a tabbed dialog must provide a client area or a frame window has to be defined when calling SftTabs_ActivatePage, CSftTabsDialog::InitializeTabControl or TSftTabsDialog::InitializeTabControl.

The size of the tab control's client area must be large enough to accommodate all pages. Differences in font sizes and display resolutions may cause problems if the client area is not large enough. A page will always be sized as large as the client area allows. To make tabbed dialog versus page size consistent, make sure to always specify the same font to be used for all dialogs.

## Designing a Page

A page is a modeless dialog which is placed in the tab control's client area.

The size of the tab control's client area must be large enough to accommodate a page. Differences in font sizes and display resolutions may cause problems if the client area is not large enough. A page will always be sized as large as the client area allows.

A tab page is designed just like a regular dialog with a few minor restrictions. A page is always a modeless dialog which has a tabbed dialog as its parent window. For that reason, the window style of a page has to be defined as "Child". The dialog should be defined without frame, border or other decoration. A system menu may be defined so a dialog caption can be entered. Both the system menu and the dialog caption will be removed by SftTabs/DLL before the page is shown. The page's caption will be used as the dialog caption for the main, tabbed dialog. This offers an easy way to change the tabbed dialog's window title when switching between pages.

To make tabbed dialog versus page size consistent, make sure to always specify the same font to be used for all dialogs.

## Test Mode

The dialog test mode offered by Borland C++ does not support custom controls.

# AppStudio, Visual C++

AppStudio and Visual C++ do not support custom control DLLs (AppStudio supports only VBX controls). It is still possible to use SftTabs/DLL with AppStudio or Visual C++, but the easy interface that is provided by other resource editors is not available.

## New Project

Whenever you create a resource script (*.RC) with dialogs which are to include a SftTabs/DLL control, make sure to include the C and C++ header file SFTTB.H. This insures that your resource definitions for the SftTabs/DLL control can be compiled correctly. Add the following statement to your resource script:

```
#include "sfttb.h"                      // SftTabs/DLL header file (for style bits)
```

The SFTTB.H header file has to be accessible to the resource compiler. This file can be found in the directory C:\SFTTABS\INCLUDE (unless changed during the installation).

## Adding a Tab Control to a Dialog

To add a SftTabs/DLL control to a dialog, use the custom control toolbar button. Click on the button and then the dialog being designed to add a control.

Once a custom control has been added to a dialog, you can edit the control properties by double-clicking anywhere within the control, or by using the *Resource, Properties...* menu command. To define a SftTabs/DLL control, enter the class **SoftelTabControl** (Windows 3.1) or **SoftelTabControl32** (for Windows NT, 95, Win32s) in the edit field labeled *Class*. A window caption is not necessary, so the edit field marked *Caption* can be left blank.

## SftTabs/DLL Control Styles

To enter a SftTabs/DLL window style in the *User Control Properties* dialog, use the following list to add the desired style values and enter the resulting hexadecimal value in the field marked *Style*. For detailed information, see Tab Control Styles.

**Style/Value Description**

WS_BORDER / 0x00800000
Draw a border around the control. The border is a dark line.

The tab control can be customized using run-time code, which can be created using the SftTabs/DLL Wizard application.

## Designing a Tabbed Dialog

A tabbed dialog is designed just like a regular dialog. The only difference is the tab control, which is added to make it a tabbed dialog. The tab control has to be designed using the SftTabs/DLL Wizard application and the resulting source code can then be copied to your application.

A tab control used with a tabbed dialog must provide a client area or a frame window has to be defined when calling SftTabs_SetControlInfo, CSftTabsDialog::InitializeTabControl or TSftTabsDialog::InitializeTabControl.

The size of the tab control's client area must be large enough to accommodate all pages. Differences in font sizes and display resolutions may cause problems if the client area is not large enough. A page will always be sized as large as the client area allows. To make tabbed dialog versus page size consistent, make sure to always specify the same font to be used for all dialogs.

## Designing a Page

A page is a modeless dialog which is placed in the tab control's client area.

The size of the tab control's client area must be large enough to accommodate a page. Differences in font sizes and display resolutions may cause problems if the client area is not

large enough. A page will always be sized as large as the client area allows.

A tab page is designed just like a regular dialog with a few minor restrictions. A page is always a modeless dialog which has a tabbed dialog as its parent window. For that reason, the window style of a page has to be defined as "Child". The dialog should be defined without frame, border or other decoration. A system menu and title bar may be defined so a dialog caption can be entered. The system menu, title bar and the dialog caption will be removed by SftTabs/DLL before the page is shown. The page's caption will be used as the dialog caption for the main, tabbed dialog. This offers an easy way to change the tabbed dialog's window title when switching between pages.

To make tabbed dialog versus page size consistent, make sure to always specify the same font to be used for all dialogs.

## Test Mode

In the dialog test mode offered by AppStudio and Visual C++, the SftTabs/DLL control will not be displayed. Instead, a gray box will show the location of the control. When using the tab key to test the tab stops, the simulated SftTabs/DLL control will not receive the input focus and appear not to have a tab stop defined.

# SDK Dialog Editor

This section applies to the Windows SDK dialog editor for Windows 3.1, Windows 95 and the Windows SDK dialog editor for Windows NT.

## First Time

In order to make SftTabs/DLL available to the dialog editor, use its *File, Open Custom...* menu command to define SftTabs/DLL to the dialog editor. This has to be done only once.

Locate the DLL using the dialog shown. The DLL can be found in the directory C:\SFTTABS\BIN (unless changed during the installation). Use the following table to select the correct DLL.

| Dialog Editor Environment | DLL Required |
|---|---|
| Windows 3.1, Windows 95 | SFTTB.DLL |
| Windows NT (DLL w/o UNICODE support) | SFTTB32.DLL |
| Windows NT (DLL with UNICODE support) | SFTTB32U.DLL |

**Note:** Do not install the 32-bit version in a 16-bit dialog editor or vice versa.

Once the DLL is installed, the SftTabs/DLL control is installed and SftTabs/DLL controls can now be added to your dialogs just like a standard Windows control.

## New Project

Whenever you create a resource script (*.RC) with dialogs which are to include a SftTabs/DLL control, make sure to include the C and C++ header file SFTTB.H. This insures that your resource definitions for the SftTabs/DLL control can be compiled correctly. Add the following statement to your resource script:

```
#include "sfttb.h"                    // SftTabs/DLL header file (for style bits)
```

The SFTTB.H header file has to be accessible to the resource compiler. This file can be found in the directory C:\SFTTABS\INCLUDE (unless changed during the installation).

## Adding a Tab Control to a Dialog

To add a SftTabs/DLL control to a dialog, use the custom control toolbar button. Click on the button and then on the dialog being designed to add a control. Once a SftTabs/DLL control has been added to a dialog, you can edit the window styles by double-clicking anywhere within the control, or by using the *Edit, Styles...* menu command. This dialog can only be used to manipulate a few very basic styles. More styles are available through the C and C++ API.

## SftTabs/DLL Control Styles

The *SftTabs/DLL Styles* dialog allows you to manipulate the following tab control attributes.

| Item | Description |
|---|---|
| Border | Turn the *Border* check box on to draw a border around the control. The border is a dark line. Equivalent to the WS_BORDER style. |
| Visible | The *Visible* check box determines whether the control is visible when the dialog box is first displayed. If the option is not checked, the control does not appear. The application can call the ShowWindow function at run-time to make the control appear. Equivalent to the WS_VISIBLE style. |
| Disabled | The *Disabled* check box disables the control by graying it. This prevents the control from responding to user input. Equivalent to the WS_DISABLED style. |
| Group | Turn the *Group* check box on to indicate the first control within a group of controls. The user can then press the arrow keys to access all controls in the group. Equivalent to the WS_GROUP style. |

| | |
|---|---|
| Tab Stop | Turn the *Tab Stop* check box on if you want the user to be able to press Tab to access this control. Equivalent to the WS_TABSTOP style. |
| OK | Click the *OK* button to accept all style settings and end the *SftTabs/DLL Styles* dialog. |
| Cancel | Click the *Cancel* button to abandon all (modified) style settings and end the *SftTabs/DLL Styles* dialog. |
| Help | Click the *Help* button for on-line help information on the *SftTabs/DLL Styles* dialog. |
| Design | Click the *Design* button to open or create a tab layout file (*.TAB). This invokes the SftTabs/DLL Wizard application, which allows you to define a tab layout and generates the required C or C++ run-time source code. |

## Designing a Tabbed Dialog

A tabbed dialog is designed just like a regular dialog. The only difference is the tab control, which is added to make it a tabbed dialog. The tab control has to be designed using the SftTabs/DLL Wizard application and the resulting source code can then be copied to your application.

A tab control used with a tabbed dialog must provide a client area or a frame window has to be defined when calling SftTabs_ActivatePage, CSftTabsDialog::InitializeTabControl or TSftTabsDialog::InitializeTabControl.

The size of the tab control's client area must be large enough to accommodate all pages. Differences in font sizes and display resolutions may cause problems if the client area is not large enough. A page will always be sized as large as the client area allows. To make tabbed dialog versus page size consistent, make sure to always specify the same font to be used for all dialogs.

## Designing a Page

A page is a modeless dialog which is placed in the tab control's client area.

The size of the tab control's client area must be large enough to accommodate a page. Differences in font sizes and display resolutions may cause problems if the client area is not large enough. A page will always be sized as large as the client area allows.

A tab page is designed just like a regular dialog with a few minor restrictions. A page is always a modeless dialog which has a tabbed dialog as its parent window. For that reason, the window style of a page has to be defined as "Child". The dialog should be defined without frame, border or other decoration. A system menu and caption may be defined. The system menu, title bar and the dialog caption will be removed by SftTabs/DLL before the page is shown. The page's caption will be used as the dialog caption for the main, tabbed dialog. This offers an easy way to change the tabbed dialog's window title when switching between pages.

To make tabbed dialog versus page size consistent, make sure to always specify the same font to be used for all dialogs.

## Test Mode

In the dialog test mode offered by the dialog editor, a SftTabs/DLL control will be displayed in the location specified. The tab control will not reflect any settings you may have defined using the SftTabs/DLL Wizard application. The tab control shown will always be of the "Standard" style with four tabs.

# SoftelGrayDialog Dialog Class

Most of the tab controls offered with SftTabs/DLL look best on a gray background, rather than the standard window background (usually white). While Windows 95 offers a 3D look and applications running on Windows 3.1 and Windows NT can use CTL3DV2.DLL or CTL3D32.DLL to achieve a similar result, these may not always be available to provide the gray background.

In order to achieve a gray background, dialogs can be defined using the included SoftelGrayDialog and SoftelGrayDialog32 window classes. Please note that these are window classes, not C++ classes. These window classes can be used when defining a dialog resource. At run-time, the dialog's background will automatically be gray, without the use of CTL3DV2 or other DLLs. Other than the background color, the SoftelGrayDialog(32) class behaves just like a regular dialog.

The SoftelGrayDialog and SoftelGrayDialog32 classes are only available if the DLL is linked to an application. RegisterApp has to be called to insure that the DLL is linked, even if the application doesn't use a tab control.

The class SoftelGrayDialog is used for Windows 3.1 (16-bit) applications. The class SoftelGrayDialog32 is used for all 32-bit applications (Windows NT, Windows 95 and Win32s).

## Using AppStudio, Visual C++

The Microsoft Foundation Class library uses a gray background color for dialogs if enabled using CWinApp::SetDialogBkColor. In that case there is no need to use the SoftelGrayDialog(32) class. When developing non-MFC applications, dialogs can be defined using the window class SoftelGrayDialog or SoftelGrayDialog32. The application itself does not need any changes. A dialog can use the SoftelGrayDialog or SoftelGrayDialog32 class by entering the class name in the *Dialog Properties* dialog. The dialog can be invoked by double-clicking on the background of the dialog being edited or by using the *Resource, Properties...* menu command.

Note: If a resource file of an MFC project is edited, the *Registered Class* edit control is disabled and the SoftelGrayDialog(32) window class cannot be used.

## Using Resource Workshop

When developing applications, dialogs can be defined using the window classes SoftelGrayDialog and SoftelGrayDialog32. The application itself does not need any changes. A dialog can use the SoftelGrayDialog(32) class by entering the class name in the *Window Style* dialog. The dialog can be invoked by double-clicking on the border of the dialog being edited or by using the *Control, Properties...* menu command.

**Note:** Resource Workshop also offers "bordlg" and "BorDlg_Gray" as window class names. Neither is compatible with SftTabs/DLL. SftTabs/DLL doesn't support dialogs with a dithered background color (only solid colors are supported).

## Using the Windows SDK Dialog Editor

When developing applications, dialogs can be defined using the window classes SoftelGrayDialog and SoftelGrayDialog32. The application itself does not need any changes. A dialog can use the SoftelGrayDialog(32) class by entering the class name in the *Dialog Styles* dialog. The dialog can be invoked by double-clicking on the background of the dialog being edited or by using the *Edit, Styles...* menu command.

## Tab Control Styles

The following tab control window styles are available in addition to the standard window styles (such as WS_BORDER, WS_TABSTOP, etc.). The tab control styles can be retrieved using GetWindowLong. It is not possible to set the styles using SetWindowLong. These styles are used to describe the features available with the current tab control. SetControlInfo can be used to change tab control attributes.

Once a tab style has been defined (using SetControlInfo, by supplying a *style* value in the SFTTABS_CONTROL structure, the tab control sets the window style bits listed below, to indicate what features the current tab control supports. The actual tab style (SFTTABSSTYLE_*xxx*) can also be found in the window's style information. For an up-to-date list of style values, see the header file SFTTB.H in the directory C:\SFTTABS\INCLUDE (unless changed during installation).

### SFTTABSSTYLE_CLIENTAREA (0x2000L)

This style bit is used to determine if the current tab control supports a client area. If this style bit is on, the tab control can be defined as having a client area (using SFTTABS_CONTROL, *fClientArea*).

### SFTTABSSTYLE_HORIZONTAL (0x1000L)

This style bit is used to determine the basic orientation of tab rows. If this style bit is on, the tab control's tabs are arranged horizontally within one row, otherwise they are arranged vertically.

### SFTTABSSTYLE_MARGIN (0x0400L)

The tab control supports left and right margins between the tab control border and the first tab.

### SFTTABSSTYLE_MULTILINE (0x0100L)

The tab control supports multiline tab labels if this style bit is on, otherwise only single line labels are available.

### SFTTABSSTYLE_MULTIROW (0x0800L)

The tab control supports more than one row of tabs if this style bit is on.

### SFTTABSSTYLE_SCROLLABLE (0x0200L)

The tab control supports scrollable tabs if this style bit is on.

## Notifications

The parent window of a tab control can receive the following event notifications using the WM_COMMAND message.

**Note:** The WM_COMMAND message parameter packing is environment specific.

**WIN16:**

*NotifyCode = HIWORD(lParam);*
*idItem = wParam;*
*hwndCtl = (HWND) LOWORD(lParam);*

**WIN32:**

*NotifyCode = HIWORD(wParam);*
*idItem = LOWORD(wParam);*
*hwndCtl = (HWND) lParam;*

| NotifyCode | Description |
|---|---|
| SFTTABSN_KILLFOCUS | The tab control lost the input focus. |
| SFTTABSN_SETFOCUS | The tab control received the input focus. |
| SFTTABSN_SWITCHING | The user has initiated a switch to another tab. This notification signals that the tab control is about to switch away from the current tab to a new tab. The application can cancel switching to the new tab by sending a WM_CANCELMODE message to the tab control. If the application doesn't cancel the switching, the new tab will be activated and a SFTTABSN_SWITCHED notification sent to the parent window. |
| SFTTABSN_SWITCHED | The tab control has been switched to a new tab, which is now active. |
| SFTTABSN_SCROLLED | The user has caused scrolling of the tabs shown, by pressing a scroll button or by using the keyboard interface. |
| SFTTABSN_MOUSEMOVE | The tab control received a WM_MOUSEMOVE message. |
| SFTTABSN_SIZECHANGED | The tab control received a WM_SIZE message or the tab control's client area has been resized. |
| SFTTABSN_MBUTTONDOWN | The tab control received a WM_MBUTTONDOWN message which it doesn't process. This notification is only generated if the mouse cursor is located on a tab. |
| SFTTABSN_MBUTTONDBLCLK | The tab control received a WM_MBUTTONDBLCLK message which it doesn't process. This notification is only generated if the mouse cursor is located on a tab. |
| SFTTABSN_RBUTTONDOWN | The tab control received a WM_RBUTTONDOWN message which it doesn't process. This notification is only generated if the mouse cursor is located on a tab. |
| SFTTABSN_RBUTTONDBLCLK | The tab control received a WM_RBUTTONDBLCLK message which it doesn't process. This notification is only generated if the mouse cursor is located on a tab. |
| SFTTABSN_TTSHOW | The tab control is about to display a tooltip for a tab. |
| SFTTABSN_TTPOP | The tab control is about to hide the tooltip currently shown for a tab. |

# MFC and Notifications

Notifications can be handled by a tab control's parent window or directly by the tab control itself (in a derived class).

### Parent Window

If you want to handle Windows notification messages sent by a tab control to its parent (usually a class derived from CDialog), add a message-map entry and a message-handler member function to the parent class for each notification.

Message-map entries take the following form:

```
ON_Notification( id, memberFxn )
```

The parent's function prototype is as follows:

```
afx_msg void memberFxn( );
```

*Notification* specifies one of the available notification codes listed in Notifications. *id* specifies the child window ID of the control sending the notification and *memberFxn* is the name of the parent member function in your application which handles the notification.

**Example:**

```
// Event handler prototype added to dialog/window class
afx_msg void OnTabSwitching();

// Event handler(s) added to message map
BEGIN_MESSAGE_MAP(CSampleView, CView)
    ON_SFTTABSN_SWITCHING(IDC_TAB, OnTabSwitching)
END_MESSAGE_MAP()

// Event handler implementation
void CSampleView::OnTabSwitching()
{
    if (!SwitchingOK())
        m_Tab.SendMessage(WM_CANCELMODE);// cancel switching
}
```

### Derived Objects

By overriding the OnChildNotify function of an object derived from CSftTabs, you can handle messages in the object's class. The parameters are as documented in Notifications. Please see the Visual C++ documentation for additional information regarding the OnChildNotify function.

Starting with Visual C++ 4.0, MFC defines the ON_CONTROL_REFLECT macro which allows adding notifications directly to the message map. OnChildNotify doesn't have to be used any longer. SftTabs/DLL implements all required macros based on ON_CONTROL_REFLECT. See the MFC documentation for more information on message reflection.

Message-map entries take the following form:

```
ON_Notification_REFLECT( memberFxn )
```

The function prototype is as follows:

```
afx_msg void memberFxn( );
```

*Notification* specifies one of the available notification codes listed in Notifications. *memberFxn* is the name of the member function in your object's class which handles the notification.

```
BEGIN_MESSAGE_MAP(CYourTabControl, CSftTabs)
```

```
    //{{AFX_MSG_MAP(CYourTabControl)
    ON_WM_CREATE()
    ON_SFTTABSN_SWITCHING REFLECT(OnTabSwitching)
    ON_SFTTABSN_SWITCHED REFLECT(OnTabSwitched)
END_MESSAGE_MAP()
```

# OWL and Notifications

Notifications can be handled by a tab control's parent window or directly by the tab control itself (in a derived class).

### Parent Window

If you want to handle Windows notification messages sent by a tab control to its parent (usually a class derived from TDialog), add a response table entry and a response function to the parent class for each notification.

Response table entries take the following form:

```
EV_Notification( id, memberFxn ),
```

The parent's response function (event handler) prototype is as follows:

```
void memberFxn( );
```

*Notification* specifies one of the available notification codes listed in Notifications. *id* specifies the child window ID of the control sending the notification and *memberFxn* is the name of the parent response function in your application which handles the notification.

**Example:**

```
// Event handler prototype added to dialog/window class
void EvTabSwitching();

// Response table
DEFINE_RESPONSE_TABLE1(TMainWin, TLayoutWindow)
    EV_SFTTABSN_SWITCHING(IDC_TAB, EvTabSwitching),
END_RESPONSE_TABLE;

// Event handler implementation
void TMainWin::EvTabSwitching()
{
    if (YouDontLikeThisUser())
        pTabCtl->SendMessage(WM_CANCELMODE);// cancel switching
}
```

### Derived Objects

OWL defines the EV_NOTIFY_AT_CHILD macro which allows adding notifications directly to the response table of an object derived from TSftTabs. SftTabs/DLL implements all required macros based on EV_NOTIFY_AT_CHILD. See the OWL documentation for more information on event handlers.

Response table entries take the following form:

```
EV_Notification_AT_CHILD( memberFxn ),
```

The response function (event handler) prototype is as follows:

```
void memberFxn( );
```

*Notification* specifies one of the available notification codes listed in Notifications. *memberFxn* is the name of the member function in your object's class which handles the notification.

```
// response table
DEFINE_RESPONSE_TABLE1(TYourTabControl, TSftTabs)
    EV_SFTTABSN_SWITCHING_AT_CHILD(EvTabSwitching),
    EV_SFTTABSN_SWITCHED_AT_CHILD(EvTabSwitched),
END_RESPONSE_TABLE;
```

# Windows Messages

## WM_CONTEXTMENU

The WM_CONTEXTMENU message notifies a window that the user clicked the right mouse button in the tab control.

**Parameters:**

*hwnd = (HWND) wParam;*
Window handle of the tab control.

*xPos = LOWORD(lParam);*
Horizontal position of the cursor, in screen coordinates, at the time of the mouse click.

*yPos = HIWORD(lParam);*
Vertical position of the cursor, in screen coordinates, at the time of the mouse click.

**Comments**

A window can process this message by displaying a context menu using the TrackPopupMenu or TrackPopupMenuEx function.

The WM_CONTEXTMENU message is only generated by Windows 95, Windows NT 3.51 and above.

## WM_CTLCOLOR, WM_CTLCOLORSTATIC

The WM_CTLCOLOR (or WM_CTLCOLORSTATIC) message is sent to the parent window of a tab control. A parent window can set the tab control's background color as described in the programming sections Using C, Using C++ and the Microsoft Foundation Class library and Using C++ and the ObjectWindows Library.

Using GetCtlColors and SetCtlColors is the preferred method to change color attributes. Although a tab control generates WM_CTLCOLOR messages, the WM_CTLCOLOR message handling is provided for compatibility with SftTabs 2.0 only.

## WM_QUERYENDSESSION

The WM_QUERYENDSESSION message is sent to a page of a tabbed dialog, when the user chooses to switch to another page or to end the tabbed dialog.

**Returns**

The return value specifies what action is to be taken. Return TRUE to prevent the tab control from switching to another page, or return FALSE to allow switching to another tab.

**Comments**

This message is only used for tabbed dialogs implemented using the C API and the techniques shown in Implementing Tabbed Dialogs. The C++ implementation of tabbed dialogs does not generate or use this message.

If a page (or dialog procedure) doesn't handle this message, tab switching is automatic and always possible.

# MFC/C++ SftTabs/DLL Classes

| *Class* | *Description* |
|---|---|
| CSftTabs | Tab Control |
| CSftTabsDialog | Tabbed dialogs |
| CSftTabsPage | Page of a tabbed dialog |
| CSftTabsWindowSheet | Tabbed window |
| CSftTabsWindowPage | Page of a tabbed window |

### CSftTabs Class, Member Functions

CSftTabs is derived from CWnd.

| | |
|---|---|
| AddTab | Adds a tab |
| AdjustClientRect | Calculates the tab control size based on client area size |
| Create | Creates a tab control |
| CSftTabs | Constructor |
| DeleteTab | Deletes a tab |
| GetControlInfo | Retrieves all attributes of a tab control |
| GetCount | Retrieves the number of tabs in a tab control |
| GetCtlColors | Retrieves the tab control's color attributes |
| GetCurrentTab | Retrieves the index of the currently active tab |
| GetNextTab | Retrieves the index of the next tab about to become active |
| GetTabDialog | Retrieves the page object attached to a tab |
| GetTabInfo | Retrieves all attributes of a tab |
| GetTabLabel | Retrieves a tab's text |
| GetTabLabelLen | Retrieves a tab's text length |
| GetTabText | Retrieves a tab's text |
| GetTabWindowPage | Retrieves the page object attached to a tab |
| GetToolTip | Retrieves a tab's tooltip text |
| GetToolTipLen | Retrieves a tab's tooltip text length |
| InsertTab | Inserts a tab |
| QueryChar | Tests if a character is processed by the tab control |
| RegisterApp | Registers an application with SftTabs/DLL |
| ResetContent | Deletes all tabs |
| ResizePages | Resizes attached pages when using a frame window |
| ScrollTabs | Scrolls tabs in a scrollable tab control |
| SetControlInfo | Sets all attributes of a tab control |
| SetCtlColors | Sets the tab control's color attributes |
| SetCurrentTab | Sets the index of the currently active tab |
| SetTabDialog | Sets a page object attached to a tab |
| SetTabInfo | Retrieves all attributes of a tab |
| SetTabLabel | Sets a tab's text |
| SetTabWindowPage | Sets a page object attached to a tab |
| SetToolTip | Sets a tab's tooltip text |
| SetVersion | Sets the SftTabs/DLL version an application requires |
| UnregisterApp | Unregisters an application from SftTabs/DLL |

### CSftTabsDialog Class, Member Functions

The class CSftTabsDialog describes a main, tabbed dialog. A CSftTabsDialog based dialog is created using a dialog resource defined using a resource editor such as AppStudio, Resource Workshop or other dialog editors. A CSftTabsDialog based dialog contains at least one tab control (CSftTabs based) and optionally buttons, such as OK, Cancel, and other Windows controls.

CSftTabsDialog is derived from CDialog.

| | |
|---|---|
| ClosePossible | Tests if a tabbed dialog can be closed |
| CSftTabsDialog | Constructor |
| GetModified | Returns the current data modification flag |
| InitializeTabControl | Initializes a tab control and activates the current page |

| | |
|---|---|
| OnCancel | Called for Cancel button handling |
| OnOK | Called for OK button handling |
| SetClose | Signals that data has been permanently altered |
| SetModified | Sets the current data modification flag |

### CSftTabsPage Class, Member Functions

The class CSftTabsPage describes a dialog (called page) attached to a tab control, which is embedded in a CSftTabsDialog based dialog. A CSftTabsPage based dialog is created using a dialog resource defined using a resource editor such as AppStudio, Resource Workshop or other dialog editors. A CSftTabsPage based dialog contains Windows controls and may optionally also include a tab control with nested CSftTabsPage objects attached to the tab control.

CSftTabsPage is derived from CDialog.

| | |
|---|---|
| AllowDestroy | Tests if a page can be destroyed |
| AllowSwitch | Tests if a page can be left |
| ClosePossible | Tests if a page can be closed |
| CSftTabsPage | Constructor |
| GetModified | Returns the current data modification flag |
| GetParentDialog | Returns the parent's CSftTabsDialog based object |
| InitializeTabControl | Initializes a tab control and activates the current page |
| OnCancel | Called for Cancel button handling |
| OnOK | Called for OK button handling |
| SetClose | Signals that data has been permanently altered |
| SetModified | Sets the current data modification flag |

### CSftTabsWindowSheet Class, Member Functions

The class CSftTabsWindowSheet describes the support necessary for a tabbed, main window. A tabbed window is usually created dynamically using the CWnd::Create function. A tabbed window contains at least one tab control (CSftTabs based) and optionally other Windows controls.

The class CSftTabsWindowSheet is used to add tabbed window support to most CWnd-derived classes. This is accomplished using multiple inheritance. You supply the CWnd-derived class, and through multiple inheritance, the class can then be used as a tabbed window, containing one or more tab controls with attached pages.

| | |
|---|---|
| ClosePossible | Tests if a tabbed window can be closed |
| CSftTabsWindowSheet | Constructor |
| InitializeTabControl | Initializes a tab control and activates the current page |
| TabSwitched | Called by application to handle the SFTTABSN_SWITCHED notification |
| TabSwitching | Called by application to handle the SFTTABSN_SWITCHING notification |
| TerminateTabControl | Terminates a tab control and deactivates all pages |

### CSftTabsWindowPage Class, Member Functions

The class CSftTabsWindowPage describes the support necessary for a window to be used as a page in a tabbed window. A CSftTabsWindowPage based window is typically created dynamically (at run-time) when the user switches to a tab.

The class CSftTabsWindowPage is used to add support to most CWnd-derived classes so they can be used as pages in a tabbed window. This is accomplished using multiple inheritance. You supply the CWnd-derived class, and through multiple inheritance, the class can then be used as a page in a tabbed window.

| | |
|---|---|
| ActivatePage | Creates or activates a page |
| AllowSwitch | Tests if a page can be left |
| CSftTabsWindowPage | Constructor |
| DeactivatePage | Deactivates or destroys a page |

# OWL/C++ SftTabs/DLL Classes

| *Class* | *Description* |
|---|---|
| TSftTabs | Tab Control |
| TSftTabsDialog | Tabbed dialogs |
| TSftTabsPage | Page of a tabbed dialog |
| TSftTabsWindowSheet | Tabbed window |
| TSftTabsWindowPage | Page of a tabbed window |

### TSftTabs Class, Member Functions

TSftTabs is derived from TControl.

| | |
|---|---|
| AddTab | Adds a tab |
| AdjustClientRect | Calculates the tab control size based on client area size |
| DeleteTab | Deletes a tab |
| GetControlInfo | Retrieves all attributes of a tab control |
| GetCount | Retrieves the number of tabs in a tab control |
| GetCtlColors | Retrieves the tab control's color attributes |
| GetCurrentTab | Retrieves the index of the currently active tab |
| GetNextTab | Retrieves the index of the next tab about to become active |
| GetTabDialog | Retrieves the page object attached to a tab |
| GetTabInfo | Retrieves all attributes of a tab |
| GetTabLabel | Retrieves a tab's text |
| GetTabLabelLen | Retrieves a tab's text length |
| GetTabWindowPage | Retrieves the page object attached to a tab |
| GetToolTip | Retrieves a tab's tooltip text |
| GetToolTipLen | Retrieves a tab's tooltip text length |
| InsertTab | Inserts a tab |
| QueryChar | Tests if a character is processed by the tab control |
| RegisterApp | Registers an application with SftTabs/DLL |
| ResetContent | Deletes all tabs |
| ResizePages | Resizes attached pages when using a frame window |
| ScrollTabs | Scrolls tabs in a scrollable tab control |
| SetControlInfo | Sets all attributes of a tab control |
| SetCtlColors | Sets the tab control's color attributes |
| SetCurrentTab | Sets the index of the currently active tab |
| SetTabDialog | Sets a page object attached to a tab |
| SetTabInfo | Retrieves all attributes of a tab |
| SetTabLabel | Sets a tab's text |
| SetTabWindowPage | Sets a page object attached to a tab |
| SetToolTip | Sets a tab's tooltip text |
| SetVersion | Sets the SftTabs/DLL version an application requires |
| TSftTabs | Constructor |
| UnregisterApp | Unregisters an application from SftTabs/DLL |

### TSftTabsDialog Class, Member Functions

The class TSftTabsDialog describes a main, tabbed dialog. A TSftTabsDialog based dialog is created using a dialog resource defined using a resource editor such as Resource Workshop, AppStudio or other dialog editors. A TSftTabsDialog based dialog contains at least one tab control (TSftTabs based) and optionally buttons, such as OK, Cancel, and other Windows controls.

TSftTabsDialog is derived from TDialog.

| | |
|---|---|
| CanClose | Tests if a dialog can be closed |
| GetModified | Returns the current data modification flag |
| InitializeTabControl | Initializes a tab control and activates the current page |
| SetClose | Signals that data has been permanently altered |
| SetModified | Sets the current data modification flag |
| TSftTabsDialog | Constructor |

### TSftTabsPage Class, Member Functions

The class TSftTabsPage describes a dialog (called page) attached to a tab control, which is embedded in a TSftTabsDialog based dialog. A TSftTabsPage based dialog is created using a dialog resource defined using a resource editor such as Resource Workshop, AppStudio or other dialog editors. A TSftTabsPage based dialog contains Windows controls and may optionally also include a tab control with nested TSftTabsPage objects attached to the tab control.

TSftTabsPage is derived from TDialog.

| | |
|---|---|
| AllowDestroy | Tests if a page can be destroyed |
| CanClose | Tests if a dialog can be closed |
| CloseWindow | Close the tabbed dialog |
| CmCancel | Called for Cancel button handling |
| CmOk | Called for OK button handling |
| GetModified | Returns the current data modification flag |
| GetParentDialog | Returns the parent's TSftTabsDialog based object |
| InitializeTabControl | Initializes a tab control and activates the current page |
| SetClose | Signals that data has been permanently altered |
| SetModified | Sets the current data modification flag |
| TSftTabsPage | Constructor |

### TSftTabsWindowSheet Class, Member Functions

The class TSftTabsWindowSheet describes the support necessary for a tabbed, main window. A tabbed window is usually created dynamically using the Create function. A tabbed window contains at least one tab control (TSftTabs based) and optionally other Windows controls.

The class TSftTabsWindowSheet is used to add tabbed window support to most TWindow-derived classes. This is accomplished using multiple inheritance. You supply the TWindow-derived class, and through multiple inheritance, the class can then be used as a tabbed window, containing one or more tab controls with attached pages.

| | |
|---|---|
| CanClose | Tests if a tabbed window can be closed |
| InitializeTabControl | Initializes a tab control and activates the current page |
| TabSwitched | Called by application to handle the SFTTABSN_SWITCHED notification |
| TabSwitching | Called by application to handle the SFTTABSN_SWITCHING notification |
| TerminateTabControl | Terminates a tab control and deactivates all pages |
| TSftTabsWindowSheet | Constructor |

### TSftTabsWindowPage Class, Member Functions

The class TSftTabsWindowPage describes the support necessary for a window to be used as a page in a tabbed window. A TSftTabsWindowPage based window is typically created dynamically (at run-time) when the user switches to a tab.

The class TSftTabsWindowPage is used to add support to most TWindow-derived classes so they can be used as pages in a tabbed window. This is accomplished using multiple inheritance. You supply the TWindow-derived class, and through multiple inheritance, the class can then be used as a page in a tabbed window.

| | |
|---|---|
| ActivatePage | Creates or activates a page |
| AllowSwitch | Tests if a page can be left |
| DeactivatePage | Deactivates or destroys a page |
| TSftTabsWindowPage | Constructor |

# C, C++ API

An application communicates with the SftTabs/DLL tab control by sending messages using the Windows SendMessage function. To simplify the process, SftTabs/DLL offers not only the direct SendMessage interface, but also a predefined "message-cracker" macro for each message. This eliminates the casting of parameters when using SendMessage, and is more efficient than a SendMessage call, because the macro expands into a direct call to SftTabs/DLL.

### Definitions and Structures

| | |
|---|---|
| SFTTABS_CLASS | Tab control window class name |
| SFTTABS_COLORS | Color information |
| SFTTABS_CONTROL | Tab control structure describing tab layout and attributes |
| SFTTABS_GRAPH | Structure used to describe a tab's picture component |
| SFTTABS_GRAYDIALOGCLASS | Window class name supporting gray dialog background color |
| SFTTABS_MAXROWS | Maximum number of tab rows |
| SFTTABS_MAXTABS | Maximum number of tabs per tab control |
| SFTTABS_STYLETABLEA | Tab control style table entry for resource editors |
| SFTTABS_TAB | Structure describing one tab |
| SFTTABS_TABCALLBACK | C callback function associated with a tab, manages an attached page (dialog) |

### Messages and Functions

| | |
|---|---|
| SftTabs_ActivatePage | Activates a page |
| SftTabs_AddTab | Adds a tab |
| SftTabs_AdjustClientRect | Calculates the tab control size based on client area size |
| SftTabs_ClosePossible | Tests if a page can be closed |
| SftTabs_CopyWindowTitle | Copies the window caption of a window to another window |
| SftTabs_DeactivatePage | Deactivates the current page |
| SftTabs_DeleteTab | Deletes a tab |
| SftTabs_Destroy | Cleanup processing for tabbed dialogs and windows |
| SftTabs_GetControlInfo | Retrieves all attributes of a tab control |
| SftTabs_GetCount | Retrieves the number of tabs in a tab control |
| SftTabs_GetCtlColors | Retrieves the tab control's color attributes |
| SftTabs_GetCurrentTab | Retrieves the index of the currently active tab |
| SftTabs_GetNextTab | Retrieves the index of the next tab about to become active |
| SftTabs_GetStyleTable | Retrieves the style table for use by resource editors |
| SftTabs_GetTabControlFromPage | Returns the tab control window handle associated with a given page |
| SftTabs_GetTabInfo | Retrieves all attributes of a tab |
| SftTabs_GetTabLabel | Retrieves a tab's text |
| SftTabs_GetTabLabelLen | Retrieves a tab's text length |
| SftTabs_GetToolTip | Retrieves a tab's tooltip text |
| SftTabs_GetToolTipHandle | Retrieves the tooltip control window handle |
| SftTabs_GetToolTipLen | Retrieves a tab's tooltip text length |
| SftTabs_HandleDialogMessage | Message handling for tabbed dialogs |
| SftTabs_HandleWindowMessage | Message handling for tabbed windows |
| SftTabs_InsertTab | Inserts a tab |
| SftTabs_IsRegisteredDialog | Tests if a dialog is a registered tabbed dialog |
| SftTabs_IsRegisteredWindow | Tests if a dialog is a registered tabbed window |
| SftTabs_IsTabControl | Tests if a window is a tab control |
| SftTabs_IsTabControlWithDialog | Tests if a window is a tab control with an attached page |
| SftTabs_IsTabControlWithPage | Tests if a window is a tab control with an attached page |
| SftTabs_QueryChar | Tests if a character is processed by the tab control |
| SftTabs_RegisterApp | Registers an application with SftTabs/DLL |
| SftTabs_RegisterDialog | Registers a dialog as a tabbed dialog |

| | |
|---|---|
| SftTabs_RegisterWindow | Registers a window as a tabbed window |
| SftTabs_ResetContent | Deletes all tabs |
| SftTabs_ResizePages | Resizes attached pages when using a frame window |
| SftTabs_ScrollTabs | Scrolls tabs in a scrollable tab control |
| SftTabs_SetControlInfo | Sets all attributes of a tab control |
| SftTabs_SetCtlColors | Sets the tab control's color attributes |
| SftTabs_SetCurrentTab | Sets the index of the currently active tab |
| SftTabs_SetPageActive | Notifies tabbed dialog that a page is active |
| SftTabs_SetPageInactive | Notifies tabbed dialog that a page is no longer active |
| SftTabs_SetTabInfo | Sets all attributes of a tab |
| SftTabs_SetTabLabel | Sets a tab's text |
| SftTabs_SetToolTip | Sets a tab's tooltip text |
| SftTabs_SetVersion | Sets the SftTabs/DLL version an application requires |
| SftTabs_UnregisterApp | Unregisters an application from SftTabs/DLL |
| SftTabs_UnregisterDialog | Unregisters a registered tabbed dialog |
| SftTabs_UnregisterWindow | Unregisters a registered tabbed window |

# SFTTABS_CLASS

**WIN16**
```
#define SFTTABS_CLASS "SoftelTabControl"
```
**WIN32**
```
#define SFTTABS_CLASS "SoftelTabControl32"
```

The SFTTABS_CLASS constant can be used when the SftTabs/DLL control class name is required.

# SFTTABS_COLORS

```
typedef struct tagTabsColors {
    COLORREF colorBg;                   /* background color */
    COLORREF colorFg;                   /* foreground color */
    COLORREF color1;                    /* usually used for black border */
    COLORREF color2;                    /* usually used for shadow lines */
    COLORREF color3;                    /* usually used for highlight lines */
    COLORREF color4;                    /* usually used for somewhat highlight'ed
lines */
    DWORD res1, res2, res3, res4;       /* reserved */
    DWORD res5, res6, res7, res8;
    DWORD res9, res10, res11, res12;
    DWORD res13, res14, res15, res16;
} SFTTABS_COLORS, FAR * LPSFTTABS_COLORS;
```

The SFTTABS_COLORS structure is used with GetCtlColors and SetCtlColors to retrieve and set a tab control's color attributes.

**Members**

*colorBg*
The default background color used for the tab control. Tabs can override the default background color using SFTTABS_TAB *colorBg* or *colorBgSel*.

*colorFg*
The default foreground color used to draw tab label text. Tabs can override the default foreground color using SFTTABS_TAB *colorFg* or *colorFgSel*.

*color1*
The color used to draw the lines indicating the tab control border, preferably the darkest color.

*color2*
The color used to draw the lines away from the light source, indicating a shadow, preferably a dark color.

*color3*
The color used to draw the lines directly exposed to the light source, indicating a highlight, preferably a bright color.

*color4*
The color used to draw the lines somewhat exposed to the light source, preferably a bright color, but of lesser intensity than *color3*. This color value is not used by all tab styles.

**Comments**

Not all color values are used by all the tab styles. Some tab styles do not honor *color1* - *color4* settings. To determine support for a particular color setting, use the SftTabs/DLL Wizard application.

**Example**

This example changes the tab control's foreground and background colors.

**C**

```
    SFTTABS_COLORS Colors;
    SftTabs_GetCtlColors(hwndTab, &Colors);   /* Get current color settings */
    Colors.colorBg = RGB(0,255,255);          /* Background color */
    Colors.colorFg = RGB(0,0,128);            /* Foreground color */
    SftTab_SetCtlColors(hwndTab, &Colors);   /* Set new colors */
```

**C++/MFC**

```
    SFTTABS_COLORS Colors;
    m_Tab.GetCtlColors(&Colors);     /* Get current color settings */
    Colors.colorBg = RGB(0,255,255); /* Background color */
    Colors.colorFg = RGB(0,0,128);   /* Foreground color */
    m_Tab.SetCtlColors(&Colors);     /* Set new colors */
```

**C++/OWL**

```
    SFTTABS_COLORS Colors;
```

```
pTab->GetCtlColors(&Colors);     /* Get current color settings */
Colors.colorBg = RGB(0,255,255); /* Background color */
Colors.colorFg = RGB(0,0,128);   /* Foreground color */
pTab->SetCtlColors(&Colors);     /* Set new colors */
```

# SFTTABS_CONTROL

```
typedef struct tagSftTabsControl {

    /* Modifiable fields */
    int style;                          /* tab style */
    int nRows;                          /* number of rows */
    int nRowTabs;                       /* number of tabs per row (if fFixed) */
    int leftMargin;                     /* width of left margin */
    int rightMargin;                    /* width of right margin */
    BOOL fFixed;                        /* same width for all tabs */
    BOOL fClientArea;                   /* Client area wanted */
    BOOL fMultiline;                    /* allow multiline label text */
    BOOL fDialog;                       /* use with dialog */
    BOOL fTextOnly;                     /* use specified background color only for
text */
    BOOL fScrollable;                   /* scrollable tabs */
    BOOL fHideScrollButtons;            /* hide scroll buttons */
    BOOL fBoldFont;                     /* use bold font for active tab */
    BOOL fFillComplete;                 /* fill rows completely */
    HBITMAP hButtonBitmap;              /* Scroll Button bitmap */
    LPVOID lpTabData;                   /* Data/Dialog associated with active tab */
    HWND hwndSubDlg;                    /* Subdialog associated with active tab */
    HWND hwndFrame;                     /* Frame, used as client area */
    short fToolTips;                    /* Tooltips wanted (formerly DWORD res1) */
    short fDropText;                    /* drop text if it doesn't fit ...
    short fCondScrollButtons;           /* conditional scroll buttons ...
    short res2s;                        /* reserved (formerly DWORD res2) */

    DWORD res3, res4, res5, res6;

    /* read/only fields */
    int nTabs;                          /* number of tabs */
    RECT ClientRect;                    /* Area useable by application */
    BOOL fLeftButton, fRightButton;     /* TRUE if scrolling in that direction
possible */
    int visibleLeftTab;                 /* leftmost tab in first row (if fScrollable)
*/
    int naturalSize;                    /* Best height/width depending on tab style */

    DWORD res11, res12, res13, res14, res15, res16;

} SFTTABS_CONTROL, FAR * LPSFTTABS_CONTROL;
```

The SFTTABS_CONTROL structure is used to describe a tab control's layout and attributes.

**Members**

*style*
The tab control style. This value defines the basic look of the tab control. This value can be modified using SetControlInfo.

*nRows*
The number of tab rows. This value can be modified using SetControlInfo.

*nRowTabs*
if *fFixed* is TRUE, *nRowTabs* is used to set the number of tabs per row. If fewer tabs are available than specified in *nRowTabs*, the remainder of each row is left blank (*fFillComplete* == FALSE) or filled with disabled tabs (*fFillComplete* == TRUE), if *fFixed* is FALSE, this field should be set to 0. This value can be modified using SetControlInfo.

*leftMargin*
The number of pixels reserved for the left margin. This value can be modified using SetControlInfo.

*rightMargin*

The number of pixels reserved for the right margin. This value can be modified using SetControlInfo.

*fFixed*

The width of all tabs. If this set to TRUE, all tabs will be of the same width (or height for vertical rows). If set to FALSE, tabs will be sized proportionally to their text and picture size and the available space. This value can be modified using SetControlInfo.

*fClientArea*

The availability of a client area, normally used for pages or Windows controls attached to tabs. If this is set to TRUE, a client area is available. The client area size can be found in *clientRect*. If this field is set to FALSE, a client area is not available. Not all tab styles support a client area. The SftTabs/DLL Wizard can be used to determine which tab styles support a client area. This value can be modified using SetControlInfo.

*fMultiline*

TRUE if multiline tab text is available, other FALSE. When this field is TRUE, tab text can contain newline characters ("\r\n") to signal a new line. Not all tab styles support multiline tab text. The SftTabs/DLL Wizard can be used to determine which tab styles support multiline tab text. This value can be modified using SetControlInfo.

*fDialog*

TRUE if the tab control is used in a dialog, instead of a window. This field determines some of the colors used for the tab control. Setting this field to FALSE and still using the tab control in a dialog doesn't cause any adverse effects. This field only determines the colors used by the tab control. If a tab control will be placed on a background that is not gray (see SoftelGrayDialog Dialog Class), this field should be set to FALSE. The SftTabs/DLL Wizard can be used to determine the effect of this value. This value can be modified using SetControlInfo.

*fTextOnly*

TRUE if the tab background colors are used for the tab text only, otherwise the tab background colors (*colorBg* and *colorBgSel* in SFTTABS_TAB) are used to fill the entire tab. This value can be modified using SetControlInfo.

*fScrollable*

TRUE if the tab control offers scrollable tabs (and is restricted to one row of tabs). This value can be modified using SetControlInfo.

*fHideScrollButtons*

TRUE if the buttons used for tab scrolling are to be made invisible. If the scroll buttons are hidden, the only method to scroll the tabs is by using the keyboard interface or under program control. This value can be modified using SetControlInfo.

*fBoldFont*

TRUE if the tab text of the currently active tab should be bold, FALSE if the same font should be used for active and inactive tabs. The tab text font can be set using the Windows WM_SETFONT message. if *fBoldFont* is TRUE and the default font for the tab control is already bold, the weight of the font used for inactive tabs will be reduced. This value can be modified using SetControlInfo.

*fFillComplete*

TRUE if the tab control should attempt to fill each tab row completely so the left and right margins are minimized. For fixed width tabs (*fFixed* is TRUE), additional blank, disabled tabs are added, for variable width tabs (*fFixed* is FALSE), the available space is distributed equally among all tabs so they grow (or shrink) proportionally. This value can be modified using SetControlInfo.

*hButtonBitmap*

A bitmap handle. The bitmap is used to display the graphics of the left and right (or up and down) scroll buttons. The bitmap should contain two equal-sized images, arranged horizontally, so the height of the bitmap is the height of a button's bitmap and the width of the supplied bitmap is twice the width of a button's bitmap. The button size is automatically determined based on the bitmap size. The top left pixel of

each button bitmap must contain the background color. This color will be replaced by the actual window background color when the bitmap is displayed. This parameter may be NULL. Default scroll button bitmaps are provided by SftTabs/DLL and can be seen using the SftTabs/DLL Wizard. This value can be modified using SetControlInfo.

*lpTabData*

Stores a 32-bit pointer, used for the C++ implementation of tabbed dialogs. For C, the member is not used, but reserved. For C++, the pointer points to the C++ object based on CSftTabsPage or TSftTabsPage. This value can be modified using SftTabs_SetPageActive or SetControlInfo.

*hwndSubDlg*

Stores a window handle, used for the C and C++ implementation of tabbed dialogs. The window handle describes the page attached to the active tab. This value can be modified using SetTabInfo.

*hwndFrame*

Stores a window handle, used by SftTabs/DLL as client area for pages attached to the tab control. SftTabs/DLL uses this window's client area size and location as a replacement for the tab control's client area. The window described by *hwndFrame* may be hidden and/or disabled. If an application resizes or moves the frame window, the dependent page or Windows control also has to be resized by the application. The dependent page can be found in *hwndSubDlg*. Using this frame window handle, the client area of a tab control can be located anywhere in relation to the tab control, even on a different dialog or window. This value can be modified using SftTabs_ActivatePage, SetControlInfo, CSftTabsDialog::InitializeTabControl, TSftTabsDialog::InitializeTabControl, CSftTabsWindowSheet::InitializeTabControl or TSftTabsWindowSheet::InitializeTabControl.

*res2s, res3 - res6*

Reserved. Not Used.

*fToolTips*

TRUE if the tab control should display tooltips. Tooltip text for each tab has to be added using SetToolTip. This value can be modified using SetControlInfo.

*fDropText*

TRUE if the tab control should drop tab labels, if the labels don't fit vertically and/or horizontally. This option should only be used if all tabs have a graphic component. If the tab label of any tab would be clipped, all tabs will drop the tab label temporarily. This is most useful with resizable tab controls where the user could potentially make the tab control very small, so that tab labels would be clipped. This value can be modified using SetControlInfo.

*fToolTips*

TRUE if the tab control should hide the scroll buttons, when scrolling isn't possible because all tabs are visible. This is most useful with resizable tab controls where the user could potentially make the tab control very large, so that all tab labels can be displayed. This value can be modified using SetControlInfo.

*nTabs*

The currently defined number of tabs.

*ClientRect*

The location of the client area in tab control coordinates, where the top left corner of the tab control is at 0,0. if *fClientArea* is FALSE, an empty rectangle is returned.

*fLeftButton*

TRUE if the tab control supports scrolling and scrolling left (or up in a vertical tab control) is currently possible. This member can be used when an application provides its own scrolling mechanism, to query the tab control if scrolling is possible in this direction.

*fRightButton*

TRUE if the tab control supports scrolling and scrolling right (or down in a vertical tab control) is currently possible. This member can be used when an application provides

its own scrolling mechanism, to query the tab control if scrolling is possible in this direction.

*visibleLeftTab*

The index of the leftmost tab (or topmost in a vertical tab control) currently visible in a scrollable tab control.

*naturalSize*

The ideal height of a tab control (or width of a vertical tab control). This field is only valid for tab controls that do not offer a client area (*fClientArea* == FALSE). This value can be used to determine the best height for a tab control with horizontal tab rows (or width for vertical rows).

*res11 - res16*

Reserved. Not Used.

**Comments**

The SFTTABS_CONTROL structure can be defined using the SftTabs/DLL Wizard.

# SFTTABS_GRAPH

```
typedef struct tagSftTabsGraph {
    WORD location;
    WORD type;
    union {
        HICON hIcon;
        HBITMAP hBitmap;
    } item;
} SFTTABS_GRAPH, FAR * LPSFTTABS_GRAPH;
```

The SFTTABS_GRAPH structure is used to describe a tab's picture component and its location.

**Members**

*location*

Describes the location of the bitmap, relative to the tab text. The following values can be used:

| | |
|---|---|
| SFTTABS_GRAPH_NONE | No bitmap or icon, text is centered horizontally and vertically. |
| SFTTABS_LEFTVALIGN | No bitmap or icon, text is left aligned. Use this option for tab controls with a vertical orientation to line up the tab text on the left side. The text is centered vertically. |
| SFTTABS_RIGHTVALIGN | No bitmap or icon, text is right aligned. Use this option for tab controls with a vertical orientation to line up the tab text on the right side. The text is centered vertically. |
| SFTTABS_GRAPH_LEFTVALIGN | Bitmap or icon, tab picture is left aligned, followed by the tab text. Use this option for tab controls with a vertical orientation to line up the tab picture on the left side. The text and picture are centered vertically. |
| SFTTABS_GRAPH_RIGHTVALIGN | Bitmap or icon, tab picture is right aligned, the tab text is located to the left of the tab picture. Use this option for tab controls with a vertical orientation to line up the tab picture on the right side. The text and picture are centered vertically. |
| SFTTABS_GRAPH_TOP | Bitmap or icon, tab picture is above the tab text. The text and picture are centered horizontally. |
| SFTTABS_GRAPH_BOTTOM | Bitmap or icon, tab picture is below the tab text. The text and picture are centered horizontally. |
| SFTTABS_GRAPH_LEFT | Bitmap or icon, tab picture is on the left of the tab text. The text and picture are centered horizontally and vertically. |
| SFTTABS_GRAPH_RIGHT | Bitmap or icon, tab picture is on the right of the tab text. The text and picture are centered horizontally and vertically. |

If a rotated font is used, the orientation of the font's base line is used to determine the actual location of the bitmap or icon.

*type*

Describes the type of tab picture. The following values can be used:

| | |
|---|---|
| 0 | No tab picture |
| SFTTABS_GRAPH_ICON | Tab picture is an icon. The *hIcon* field of the *item* union contains a valid icon handle. |
| SFTTABS_GRAPH_BITMAP | Tab picture is a bitmap. The *hBitmap* field of the *item* union contains a valid bitmap handle. |

*hIcon*

An icon handle, used as tab picture if *type* is defined as SFTTABS_GRAPH_ICON.

*hBitmap*

A bitmap handle, used as tab picture if *type* is defined as SFTTABS_GRAPH_BITMAP. The top, left pixel of the bitmap must contain the image's background color. This color will be replaced by the actual window background color when the bitmap is displayed.

**Comments**

There are no default tab bitmaps or icons. Bitmap and icon handles are owned by the application. The handles have to remain valid until they are no longer used by the tab control, usually until the tab control is destroyed. The application is responsible for deleting the handles when they are no longer used.

# SFTTABS_GRAYDIALOGCLASS

**WIN16**

```
#define SFTTABS_GRAYDIALOGCLASS "SoftelGrayDialog"
```

**WIN32**

```
#define SFTTABS_GRAYDIALOGCLASS "SoftelGrayDialog32"
```

The SFTTABS_GRAYDIALOGCLASS constant can be used when the dialog class supplied with SftTabs/DLL is required. This dialog class implements a special dialog class, which offers a gray window background, which provides a more pleasing look for many of SftTabs' tab styles. See SoftelGrayDialog Dialog Class for more information.

# SFTTABS_MAXROWS

```
#define SFTTABS_MAXROWS      16
```

SFTTABS_MAXROWS defines the theoretical maximum number of tab rows supported by any tab control. Some tab styles may restrict the maximum rows to a lower number. The SftTabs/DLL Wizard can be used to determine the maximum based on the tab style selected.

# SFTTABS_MAXTABS

```
#define SFTTABS_MAXTABS     128
```

SFTTABS_MAXTABS defines the theoretical maximum number of tabs supported by any tab control. Some tab styles may restrict the maximum number of tabs to a lower number. The SftTabs/DLL Wizard can be used to determine the maximum based on the tab style selected.

# SFTTABS_STYLETABLEA

```
typedef struct tagSftTabsStyleTableA {
    /* Notice, these strings are always ANSI strings */
    LPCSTR lpszDesc;                    /* Style description */
    LPCSTR lpszStyle;                   /* style ID */
    DWORD style;
    short fAvailable;                   /* True if available */
    short res1s;                        /* reserved */
} SFTTABS_STYLETABLEA, FAR * LPSFTTABS_STYLETABLEA;
```

The SFTTABS_STYLETABLEA structure describes each available tab style.

**Members**

*lpszDesc*

The text description of the tab style. This is always an ANSI string, even when using the SftTabs/DLL DLL with UNICODE support.

*lpszStyle*

The text literal of the selected tab style. This can be used by a resource editor to translate the tab control style into a textual representation of the value, using the predefined symbols for tab styles.

*style*

The supported attributes of the tab style. Once a tab control is created, this style information can also be retrieved using GetWindowWord(hwnd, GWL_STYLE).

*fAvailable*

TRUE if the style described by the current entry is available. When using the DLLs as shipped with SftTabs/DLL, this field is always TRUE for all styles.

*res1s*

Reserved. Not Used.

**Comments**

The SFTTABS_STYLETABLEA structure is used by the SftTabs/DLL Wizard to display all available tab styles. The style table can be retrieved using the SftTabs_GetStyleTable function.

# SFTTABS_TAB

```
typedef struct tagSftTabsTab {
    /* modifiable fields */
    COLORREF colorBg, colorFg;          /* color */
    COLORREF colorBgSel, colorFgSel;
    SFTTABS_GRAPH graph;                /* graphics */
    BOOL fEnabled;                      /* enabled/disabled status */

    DWORD userData;                     /* userdata */

    DWORD lpTabData;                    /* reserved for C, C++ class implementation */
    HWND hwndSubDlg;                    /* reserved for C, C++ class implementation */

    DWORD res1, res2;

    /* read/only information */
    int x, y;                           /* position (top left corner) */
    int cx, cy;                         /* width and height */
    int cxVis, cyVis;                   /* width and height of visible portion */

#if defined(UNICODE) || defined(_UNICODE)
    LPWSTR lpszText;                    /* label text */
    LPWSTR lpszToolTip;                 /* tool tip text (formerly res10) */
#else
    LPSTR lpszText;                     /* label text */
    LPSTR lpszToolTip;                  /* tool tip text */
#endif

    DWORD res10, res11;

} SFTTABS_TAB, FAR * LPSFTTABS_TAB;
```

The SFTTABS_TAB structure is used to describe one tab label, including its colors, picture and text components.

**Members**

*colorBg*
The background color of the tab, when the tab is not the active tab. Use any solid color or SFTTABS_NOCOLOR, the default window background color. If *fTextOnly* is TRUE (see SFTTABS_CONTROL), this background color is only used as background color of the tab text, the remainder of the tab will be filled with the tab control's background color. This value can be modified using SetTabInfo.

*colorFg*
The foreground, text color of the tab, when the tab is not the active tab. Use any solid color or SFTTABS_NOCOLOR, the default window text color. This value can be modified using SetTabInfo.

*colorBgSel*
The background color of the tab, when the tab is the active tab. Use any solid color or SFTTABS_NOCOLOR, the default window background color. If *fTextOnly* is TRUE (see SFTTABS_CONTROL), this background color is only used as background color of the tab text, the remainder of the tab will be filled with the tab control's background color. This value can be modified using SetTabInfo.

*colorFgSel*
The foreground, text color of the tab, when the tab is the active tab. Use any solid color or SFTTABS_NOCOLOR, the default window text color. This value can be modified using SetTabInfo.

*graph*
The picture component. See SFTTABS_GRAPH for more information. This value can be modified using SetTabInfo.

*fEnabled*

The tab status. Set to TRUE to enable the tab or FALSE to disable. A disabled tab will be shown with its picture bitmap component drawn in a "grayed" fashion, icons are always drawn with their original colors, never grayed. The text portion will be shown grayed if the default colors (SFTTABS_NOCOLOR) are defined, otherwise the specified colors will be used. This value can be modified using SetTabInfo.

*userData*

A 32-bit application defined value associated with the tab. This value can be modified using SetTabInfo.

*lpTabData*

Stores a 32-bit pointer, used for the C and C++ implementation of tabbed dialogs. For C, the pointer points to a function of type SFTTABS_TABCALLBACK. This callback routine is called by SftTabs/DLL to create and destroy the page associated with this tab. For C++, the pointer points to the C++ object based on CSftTabsPage or TSftTabsPage. This value can be modified using SetTabInfo, SetTabDialog or SetTabWindowPage.

*hwndSubDlg*

Stores a window handle, used for the C and C++ implementation of tabbed dialogs. The window handle describes the page attached to the tab. This value can be modified using SetTabInfo.

*res1, res2*

Reserved. Not Used.

*x, y*

Current location of the tab.

*cx, cy*

Current theoretical width and height of the tab. A tab may be truncated in a scrollable tab control. In this case the *cx* and *cy* members hold the full, untruncated size of the tab.

*cxVis, cyVis*

Current actual width and height of the tab. A tab may be truncated in a scrollable tab control. In this case the *cxVis* and *cyVis* members hold the size of the visible portion of the tab.

*lpszText*

The tab text. This value can be modified using SetTabLabel.

*lpszToolTip*

The tabs tooltip text. This value can be modified using SetToolTip.

*res10, res11*

Reserved. Not Used.

**Comments**

The SFTTABS_TAB structure can be defined using the SftTabs/DLL Wizard.

# SFTTABS_TABCALLBACK

**C, not used with C++**

```
typedef HWND (CALLBACK* SFTTABS_TABCALLBACK)(BOOL fCreate, HWND hwndOwner,
                                             HWND hwndPage, HWND hwndTab);
```

Defines the callback function associated with a tab. This callback routine is called by SftTabs/DLL to create and destroy the page associated with a tab.

**Parameters**

*fCreate*

TRUE if creating a new page. If *hwndPage* is NULL, the page is created for the first time, otherwise the page already exists. When creating a new page, the application should create a modeless dialog. *fCreate* is FALSE when destroying a page or switching away from a page.

*hwndOwner*

The window handle of the tab control's parent window. This window should be the owner of any pages created by this callback function. If *fCreate* is FALSE, *hwndOwner* can be NULL, in this case the page *hwndPage* should be destroyed unconditionally. If *hwndOwner* is not NULL, the window may be left intact, the tab control is merely switching away from the current page. By returning the page's window handle, the callback can indicate that the window wasn't destroyed. Returning NULL indicates that the page was destroyed.

*hwndPage*

The window handle of the page to create or destroy. *hwndPage* may be NULL when the page is created for the first time.

*hwndTab*

The window handle of the tab control.

**Returns**

The return value is the new page's window handle if *fCreate* is TRUE. If *fCreate* is FALSE, *hwndOwner* is not NULL and the callback hasn't destroyed the page, the return value is the window handle of the page. Otherwise NULL should be returned.

**Comments**

For more information on tabbed dialogs and windows, see Implementing Tabbed Dialogs and Implementing Tabbed Windows

Under Windows 3.1, the function has to be **_export**ed. If certain compiler switches and/or (older) compiler versions are used, the function must also be listed in the EXPORTS section of the application's module definition file (*.DEF) and the MakeProcInstance function call may have to be used.

**Example**

This example supports a page, which is kept when switching away from the active tab:

**C**

```
HWND _export CALLBACK Page_Callback(BOOL fCreate, HWND hwndOwner, HWND hwndPage,
                                    HWND hwndTab)
{
    if (fCreate) {                      // creating a new page
        if (hwndPage) {
            // already created, we could do some initialization here.
            // this will be called every time the page becomes active.
            // The WM_SHOWWINDOW message is also sent to the page and could
            // be used to determine activation/deactivation of the page.

            // optional, set the main window's title to the window title defined ...
            SftTabs_CopyWindowTitle(hwndPage, hwndOwner);
            return NULL;                // return NULL, ignored
        } else {
            // Create the page.
```

```
            // You can create and initialize any type of window here, not just
dialogs.
            // Use CreateWindow to create other windows. Don't specify WS_VISIBLE, but
            // make sure you use WS_TABSTOP.
            // When creating a non-dialog window, make sure to call
SftTabs_SetPageActive
            // after the page has been created.
            HWND hwnd = CreateDialogParam(g_hInst,
MAKEINTRESOURCE(IDD_your_dialog_ID),
                                    hwndOwner, (DLGPROC)Page_yourDialogProc,
                                    (LPARAM)(UINT)hwndTab);// pass tab control as data
            // optional, set the main window's title to the window title defined ...
            SftTabs_CopyWindowTitle(hwnd, hwndOwner);
            return hwnd;
        }
    } else {                            // destroying page
        if (hwndOwner)                  // - because we're switching away
            return hwndPage;            //   keep the window handle, don't destroy it
        else {                          // - because we're closing the main dialog
            DestroyWindow(hwndPage);
            return NULL;
        }
    }
}
```

This example supports a page, which is destroyed every time when switching away from the active tab:

**C**

```
HWND _export CALLBACK Page_Callback(BOOL fCreate, HWND hwndOwner, HWND hwndPage,
                                                    HWND hwndTab)
{
    if (fCreate) {                      // creating a new page
        if (hwndPage) {
            // already created, we could do some initialization here.
            // this will be called every time the page becomes active
            // The WM_CREATE/WM_INITDIALOG/WM_DESTROY messages are also sent to
            // the page and could be used to determine activation/deactivation.
            // of the page.

            // optional, set the main window's title to the window title defined ...
            SftTabs_CopyWindowTitle(hwndPage, hwndOwner);
            return NULL;
        } else {
            // Create the page.
            // You can create and initialize any type of window here, not just
dialogs.
            // Use CreateWindow to create other windows. Don't specify WS_VISIBLE, but
            // make sure you use WS_TABSTOP.
            // When creating a non-dialog window, make sure to call
SftTabs_SetPageActive
            // after the page has been created.
            HWND hwnd = CreateDialogParam(g_hInst,
MAKEINTRESOURCE(IDD_your_dialog_ID),
                                    hwndOwner, (DLGPROC)Page_yourDialogProc,
                                    (LPARAM)(UINT)hwndTab);// pass tab control as data
            // optional, set the main window's title to the window title defined ...
            SftTabs_CopyWindowTitle(hwnd, hwndOwner);
            return hwnd;
        }
    } else {                            // destroying page
        // We'll always destroy this page (to save resources)
        DestroyWindow(hwndPage);
        return NULL;
```

```
        }
    }
```

# SftTabs_ActivatePage

```
BOOL WINAPI SftTabs_ActivatePage(HWND hwndParent, HWND hwndTab,
                 HWND hwndFrame, BOOL fInitializing);
```

The parent window of a tab control calls the SftTabs_ActivatePage function after a new page has been activated or to activate the initial page.

**Parameters**

*hwndParent*
The window handle of the tab control's parent window.

*hwndTab*
The window handle of the tab control.

*hwndFrame*
The window handle of a window to be used by SftTabs/DLL as client area for tabbed dialogs. This parameter should be NULL to use a tab control's built-in client area. If a window handle is specified, SftTabs/DLL uses the client area size and location as a replacement for the tab control's client area. The window described by *hwndFrame* may be hidden and/or disabled. If an application resizes or moves the frame window, the dependent page or windows control also has to be resized by using the ResizePages function. Using this frame window handle, the client area of a tab control can be located anywhere in relation to the tab control, even on a different dialog or window.

*fInitializing*
Set to TRUE when the tabbed dialog is being created and is not yet visible (usually during WM_INITDIALOG or WM_CREATE message handling), set to FALSE when the tab control is already visible.

**Comments**

The SftTabs_ActivatePage function causes the tab callback routine SFTTABS_TABCALLBACK, responsible for the current tab, to be called to create or initialize the new page.

If a page is already active, SftTabs_DeactivatePage should be used first to deactivate that page, before calling SftTabs_ActivatePage.

**Example**

This C example shows the end of a typical tabbed dialog WM_INITDIALOG message handler:

```
      ... additional initialization code ...

  index = SftTabs_AddTab(hwndTab, TEXT("&Six"));
  SftTabs_SetTabInfo(hwndTab, index, &Tab5);

  SftTabs_SetControlInfo(hwndTab, &CtlInit);

  // Make sure to turn redraw back on
  SendMessage(hwndTab, WM_SETREDRAW, (WPARAM)TRUE, 0);
  InvalidateRect(hwndTab, NULL, TRUE);

  // Activate current page.
  SftTabs_ActivatePage(hwndParent, hwndTab, NULL, TRUE);

  // Mark the window as a main, tabbed dialog (so accel. keys work) by registering
it.
  // Register the dialog AFTER activating the current page
  SftTabs_RegisterDialog(hwndParent);

  return FALSE;                        // WM_INITDIALOG, input focus already set
```

# AddTab

**C, WIN16, SFTTABSM_ADDTAB**

```
int SftTabs_AddTab(HWND hwnd, LPCSTR lpsz);
```

**C, WIN32, SFTTABSM_ADDTAB, _A, _W**

```
int SftTabs_AddTab(HWND hwnd, LPCTSTR lpsz);
int SftTabs_AddTab_A(HWND hwnd, LPCSTR lpsz);
int SftTabs_AddTab_W(HWND hwnd, LPCWSTR lpsz);
```

**C++, WIN16, CSftTabs::AddTab, TSftTabs::AddTab**

```
int AddTab(LPCSTR lpsz);
```

**C++, WIN32, CSftTabs::AddTab, TSftTabs::AddTab**

```
int AddTab(LPCTSTR lpsz);
```

Adds a new tab to a tab control. The new tab will be added as the last tab.

**Parameters**

*lpsz*

Points to the null-terminated string that is to be used as text for the tab label.

**Returns**

The return value is the zero-based index of the newly added tab. The return value is -1 if an error occurred.

**Comments**

The tab control creates a copy of the string supplied.

The WM_SETREDRAW Windows message can be used to suppress the tab control from being redrawn when many tabs are added.

Tabs can be deleted using DeleteTab. New tabs can be inserted at a specific location using InsertTab.

**Example**

This example adds a tab with the text "A Test" to a tab control:

**C**

```
index = SftTabs_AddTab(hwndTab, "A Test");
```

**C++/MFC**

```
index = m_Tab.AddTab("A Test");
```

**C++/OWL**

```
index = pTab->AddTab("A Test");
```

# AdjustClientRect

**C, SFTTABSM_ADJUSTCLIENTRECT**

```
BOOL SftTabs_AdjustClientRect(HWND hwnd, LPRECT lpRect);
```

**C++, CSftTabs::AdjustClientRect, TSftTabs::AdjustClientRect**

```
BOOL AdjustClientRect(LPRECT lpRect);
```

Calculates a tab control size, which will provide a client area of the given size.

**Parameters**

*lpRect*

Points to a RECT structure containing the desired client area size. The values in the RECT structure will be updated to contain the required tab control size to accommodate the desired client area size.

**Returns**

The return value is TRUE if the function was successful, otherwise FALSE is returned.

**Comments**

This function can only be used with tab control styles that provide a client area (*fClientArea* of the SFTTABS_CONTROL structure is TRUE).

**Example**

This example calculates the tab control size necessary to fit a client area size of 100 pixels in width and 50 pixels in height:

**C**

```
RECT rect;
SetRect(&rect, 0, 0, 100, 50);
SftTabs_AdjustClientRect(hwndTab, &rect);
```

**C++/MFC**

```
CRect rect(0,0,100,50);
m_Tab.AdjustClientRect(&rect);
```

**C++/OWL**

```
RECT rect;
SetRect(&rect, 0, 0, 100, 50);
pTab->AdjustClientRect(&rect);
```

# SftTabs_ClosePossible

```
BOOL WINAPI SftTabs_ClosePossible(HWND hwndParent, HWND hwndTab);
```

The parent window of a tab control calls the SftTabs_ClosePossible function to test if a page can be ended.

**Parameters**

*hwndParent*

The window handle of the tab control's parent window.

*hwndTab*

The window handle of the tab control.

**Returns**

The return value is TRUE if the current page can be ended, otherwise the return value is FALSE.

**Comments**

The SftTabs_ClosePossible function sends a WM_QUERYENDSESSION message to the currently active tab and its associated page, to determine if the page can be deactivated.

**Example**

This C example shows the end of a typical tabbed dialog WM_COMMAND message handler:

```
    case WM_COMMAND: {
        // Parameter packing differs between 16-bit and 32-bit Windows
#if defined(_WIN32) || defined(__WIN32__)
        HWND hwndCtl = (HWND) lParam;
        int id = LOWORD(wParam);
        int code = HIWORD(wParam);
#else
        HWND hwndCtl = (HWND) LOWORD(lParam);
        int id = (int) wParam;
        int code = HIWORD(lParam);
#endif
        if (hwndCtl) {
            switch (id) {
            case IDC_TAB:
                switch (code) {
                case SFTTABSN_SWITCHING:// we're about to switch away from
                    // the current page.  If you need to know what the new
                    // page will be use SftTabs_GetNextTab(hwndCtl).
                    if (!SftTabs_DeactivatePage(hwndParent, hwndCtl))
                        // couldn't deactivate current page, so don't switch
                        SendMessage(hwndCtl, WM_CANCELMODE, 0, 0);
                    break;
                case SFTTABSN_SWITCHED:// we switched to a new page
                    SftTabs_ActivatePage(hwndParent, hwndCtl, NULL, FALSE);
                    break;
                }
                break;
            case IDOK:
            case IDCANCEL:
                if (code == BN_CLICKED)
                    SendMessage(hwndParent, WM_COMMAND, id, 0);
                break;
            }
        } else {
            switch (id) {
            case IDOK:
                // The currently active page will be called with a
                // WM_QUERYENDSESSION message to determine if it can be closed
                if (SftTabs_ClosePossible(hwndParent, GetDlgItem(hwndParent,
```

```
IDC_TAB)))
                    EndDialog(hwndParent, TRUE);
                break;
            case IDCANCEL:
                EndDialog(hwndParent, FALSE);
                break;
                // The above assumes that this is a modal dialog. If it is a modeless
                // don't use EndDialog, use DestroyWindow instead.
            }
        }
        break;
       }
```

# SftTabs_CopyWindowTitle

```
void _WINAPI SftTabs_CopyWindowTitle(HWND hwndFrom, HWND hwndTo);
```

Copies the window caption of a window to another window.

#### Parameters

*hwndFrom*

The window handle of the window whose caption is to be copied to *hwndTo*.

*hwndTo*

The window handle of the window which is to receive the window caption copied from *hwndFrom*.

#### Comments

SftTabs_CopyWindowTitle is typically used in a SFTTABS_TABCALLBACK function to copy a page's caption to the enclosing dialog.

If the window caption described by *hwndFrom* is an empty string, the caption of the window described by *hwndTo* is not changed.

# Create

**C++, MFC only, CSftTabs::Create**

```
BOOL Create(DWORD dwStyle, const RECT& rect, CWnd* pParentWnd, UINT nID);
```

Creates a tab control window and attaches it to the CSftTabs object.

**Parameters**

*dwStyle*
Specifies the window style of the tab control.

*rect*
Specifies the tab control size and location. Can be a CRect object or a RECT structure.

*pParentWnd*
Specifies the tab control's parent window (usually a CDialog or a CView object).

*nID*
Specifies the tab control's ID.

**Returns**

The return value is TRUE if the function was successful, otherwise FALSE is returned.

**Comments**

A CSftTabs object is created in two steps. First the constructor CSftTabs is called, then the CSftTabs::Create member function, which initializes the tab control window and attaches it to the CSftTabs object.

**Example**

This example creates a tab control:

**C++/MFC**

```
CSftTabs Tab;
Tab.Create(WS_CHILD|WS_VISIBLE|SFTTABSSTYLE_STANDARD_LEFT,
      CRect(250, 200, 400, 700),
      pParentWnd,
      IDC_TAB);
```

# CSftTabs::CSftTabs

**C++, MFC only, CSftTabs::CSftTabs**

```
CSftTabs();
```

Standard constructor.

**Comments**

A CSftTabs object is created in two steps. First the constructor CSftTabs is called, then the CSftTabs::Create member function, which initializes the tab control window and attaches it to the CSftTabs object.

# SftTabs_DeactivatePage

```
BOOL WINAPI SftTabs_DeactivatePage(HWND hwndParent, HWND hwndTab)
```

The parent window of a tab control calls the SftTabs_DeactivatePage function to signal that a page is no longer the active page.

**Parameters**

*hwndParent*

The window handle of the tab control's parent window.

*hwndTab*

The window handle of the tab control.

**Returns**

The return value is TRUE if the current page was deactivated, otherwise the return value is FALSE.

**Comments**

The SftTabs_DeactivatePage function sends a WM_QUERYENDSESSION message to the currently active tab and its associated dialog procedure, to determine if the page can be deactivated. It also causes the tab callback routine SFTTABS_TABCALLBACK responsible for the current tab to be called to destroy the page.

**Example**

This C example shows a typical tabbed dialog WM_COMMAND message handler:

```
    case WM_COMMAND: {
        // Parameter packing differs between 16-bit and 32-bit Windows
#if defined(_WIN32) || defined(__WIN32__)
        HWND hwndCtl = (HWND) lParam;
        int id = LOWORD(wParam);
        int code = HIWORD(wParam);
#else
        HWND hwndCtl = (HWND) LOWORD(lParam);
        int id = (int) wParam;
        int code = HIWORD(lParam);
#endif
        if (hwndCtl) {
            switch (id) {
            case IDC_TAB:
                switch (code) {
                case SFTTABSN_SWITCHING:// we're about to switch away from
                    // the current page.  If you need to know what the new
                    // page will be use SftTabs_GetNextTab(hwndCtl).
                    if (!SftTabs_DeactivatePage(hwndParent, hwndCtl))
                        // couldn't deactivate current page, so don't switch
                        SendMessage(hwndCtl, WM_CANCELMODE, 0, 0);
                    break;
                case SFTTABSN_SWITCHED:// we switched to a new page
                    SftTabs_ActivatePage(hwndParent, hwndCtl, NULL, FALSE);
                    break;
                }
                break;
            case IDOK:
            case IDCANCEL:
                if (code == BN_CLICKED)
                    SendMessage(hwndParent, WM_COMMAND, id, 0);
                break;
            }
        } else {
            switch (id) {
            case IDOK:
                // The currently active page will be called with a
                // WM_QUERYENDSESSION message to determine if it can be closed
```

```
                if (SftTabs_ClosePossible(hwndParent, GetDlgItem(hwndParent,
IDC_TAB)))
                    EndDialog(hwndParent, TRUE);
                break;
            case IDCANCEL:
                EndDialog(hwndParent, FALSE);
                break;
                // The above assumes that this is a modal dialog. If it is a modeless
                // don't use EndDialog, use DestroyWindow instead.
            }
        }
        break;
       }
```

# DeleteTab

**C, SFTTABSM_DELETETAB**

```
int SftTabs_DeleteTab(HWND hwnd, int iTab);
```

**C++, CSftTabs::DeleteTab, TSftTabs::DeleteTab**

```
int DeleteTab(int iTab);
```

Deletes a tab from the tab control.

**Parameters**

*iTab*

Specifies the zero-based index of the tab to be deleted.

**Returns**

The return value is the number of tabs remaining in the tab control. The return value is -1 if an error occurred.

**Comments**

The WM_SETREDRAW Windows message can be used to suppress the tab control from being redrawn when many tabs are deleted.

Deleting an active tab can cause unpredictable results. Switch to another tab first using SetCurrentTab.

**Example**

This example deletes the tenth tab from the tab control:

**C**

```
total = SftTabs_DeleteTab(hwndTab, 9);
```

**C++/MFC**

```
total = m_Tab.DeleteTab(9);
```

**C++/OWL**

```
total = pTab->DeleteTab(9);
```

# SftTabs_Destroy

```
BOOL WINAPI SftTabs_Destroy(HWND hwndParent, HWND hwndTab);
```

The parent window of a tab control calls SftTabs_Destroy when the parent window is about to be destroyed.

#### Parameters

*hwndParent*

The window handle of the tab control's parent window.

*hwndTab*

The window handle of the tab control.

#### Returns

The return value is TRUE if the function was successful.

#### Comments

The SftTabs_Destroy function ends and destroys all pages that may still exist (even though not active) by calling the tab callback routines SFTTABS_TABCALLBACK responsible for each tab page.

#### Example

This C example shows a typical tabbed dialog WM_DESTROY message handler:

```
/* Unregister, or the window properties used won't be removed */
SftTabs_UnregisterDialog(hwndParent);
/* destroy all pages */
SftTabs_Destroy(hwndParent, GetDlgItem(hwndParent, IDC_TAB));
```

# GetControlInfo

**C, SFTTABSM_GETCONTROLINFO**

```
BOOL SftTabs_GetControlInfo(HWND hwnd, LPSFTTABS_CONTROL lpCtl);
```

**C++, CSftTabs::GetControlInfo, TSftTabs::GetControlInfo**

```
BOOL GetControlInfo(LPSFTTABS_CONTROL lpCtl) const;
```

Retrieves tab control attributes.

**Parameters**

*lpCtl*

A pointer to a SFTTABS_CONTROL structure. This structure will be updated with the current tab control attributes.

**Returns**

The return value is TRUE if the function was successful, otherwise FALSE is returned.

**Comments**

Some of the structure values returned can be modified and updated using SetControlInfo. See SFTTABS_CONTROL for more information.

**Example**

This example retrieves the current tab control attributes and modifies the number of tab rows:

**C**

```
SFTTABS_CONTROL Ctl;
SftTabs_GetControlInfo(hwndTab, &Ctl);
Ctl.nRows = 1;
SftTabs_SetControlInfo(hwndTab, &Ctl);
```

**C++/MFC**

```
SFTTABS_CONTROL Ctl;
m_Tab.GetControlInfo(&Ctl);
Ctl.nRows = 1;
m_Tab.SetControlInfo(&Ctl);
```

**C++/OWL**

```
SFTTABS_CONTROL Ctl;
pTab->GetControlInfo(&Ctl);
Ctl.nRows = 1;
pTab->SetControlInfo(&Ctl);
```

# GetCount

**C, SFTTABSM_GETCOUNT**

```
int SftTabs_GetCount(HWND hwnd);
```

**C++, CSftTabs::GetCount, TSftTabs::GetCount**

```
int GetCount() const;
```

Retrieves the number of tabs in a tab control.

**Returns**

The return value is the number of tabs defined in the tab control. The return value is -1 if an error occurred.

**Example**

This example retrieves the number of tabs:

**C**

```
    total = SftTabs_GetCount(hwndTab);
```

**C++/MFC**

```
    total = m_Tab.GetCount();
```

**C++/OWL**

```
    total = pTab->GetCount();
```

# GetCtlColors

**C, SFTTABSM_GETCTLCOLORS**

```
void SftTabs_GetCtlColors(HWND hwnd, LPSFTTABS_COLORS lpColors);
```

**C++, CSftTabs::GetCtlColors, TSftTabs::GetCtlColors**

```
void GetCtlColors(LPSFTTABS_COLORS lpColors) const;
```

Returns or sets the tab control's color attributes.

**Parameters**

*lpColors*

A pointer to a SFTTABS_COLORS structure containing the color definitions. GetCtlColors uses this structure to return the current color settings.

**Comments**

Using GetCtlColors and SetCtlColors is the preferred method to change color attributes. Although a tab control generates WM_CTLCOLOR messages, the WM_CTLCOLOR message handling is provided for compatibility with SftTabs 2.0 only.

**Example**

This example changes the tab control's foreground and background colors.

**C**

```
SFTTABS_COLORS Colors;
SftTabs_GetCtlColors(hwndTab, &Colors);   /* Get current color settings */
Colors.colorBg = RGB(0,255,255);          /* Background color */
Colors.colorFg = RGB(0,0,128);            /* Foreground color */
SftTabs_SetCtlColors(hwndTab, &Colors);   /* Set new colors */
```

**C++/MFC**

```
SFTTABS_COLORS Colors;
m_Tabs.GetCtlColors(&Colors);    /* Get current color settings */
Colors.colorBg = RGB(0,255,255); /* Background color */
Colors.colorFg = RGB(0,0,128);   /* Foreground color */
m_Tabs.SetCtlColors(&Colors);    /* Set new colors */
```

**C++/OWL**

```
SFTTABS_COLORS Colors;
pTab->GetCtlColors(&Colors);     /* Get current color settings */
Colors.colorBg = RGB(0,255,255); /* Background color */
Colors.colorFg = RGB(0,0,128);   /* Foreground color */
pTab->SetCtlColors(&Colors);     /* Set new colors */
```

# GetCurrentTab

**C, SFTTABSM_GETCURRENTTAB**

```
int SftTabs_GetCurrentTab(HWND hwnd);
```

**C++, CSftTabs::GetCurrentTab, TSftTabs::GetCurrentTab**

```
int GetCurrentTab() const;
```

Retrieves the index of the currently active tab.

**Returns**

The return value is the index of the currently active tab, otherwise -1 is returned.

**Comments**

The currently active tab can be set using SetCurrentTab.

**Example**

This example retrieves the index of the current tab:

**C**

```
iTab = SftTabs_GetCurrentTab(hwndTab);
```

**C++/MFC**

```
iTab = m_Tab.GetCurrentTab();
```

**C++/OWL**

```
iTab = pTab->GetCurrentTab();
```

# GetNextTab

**C, SFTTABSM_GETNEXTTAB**

```
int SftTabs_GetNextTab(HWND hwnd);
```

**C++, CSftTabs::GetNextTab, TSftTabs::GetNextTab**

```
int GetNextTab() const;
```

Retrieves the index of the next tab about to become active.

**Returns**

The return value is the index of the next tab about to become active.

**Comments**

GetNextTab returns the index of the tab about to become active while processing a SFTTABSN_SWITCHING notification. The application can prevent the new tab from becoming active by sending a WM_CANCELMODE message to the tab control.

GetNextTab can also be used in a CSftTabsPage::AllowSwitch, CSftTabsWindowPage::AllowSwitch, TSftTabsDialog::CanClose, TSftTabsPage::CanClose or TSftTabsWindowPage::AllowSwitch function, because these are called by the C++ class implementations while a SFTTABSN_SWITCHING notification is being processed.

GetNextTab can only be used while processing a SFTTABSN_SWITCHING notification.

**Example**

This example prevents the user from switching to tab 0:

**C**

```
    case WM_COMMAND: {
        // Parameter packing differs between 16-bit and 32-bit Windows
#if defined(_WIN32) || defined(__WIN32__)
        HWND hwndCtl = (HWND) lParam;
        int id = LOWORD(wParam);
        int code = HIWORD(wParam);
#else
        HWND hwndCtl = (HWND) LOWORD(lParam);
        int id = (int) wParam;
        int code = HIWORD(lParam);
#endif
        if (hwndCtl) {
            switch (id) {
            case IDC_TAB:
                switch (code) {
                case SFTTABSN_SWITCHING:// we're about to switch away from
                    // the current page.  If you need to know what the new
                    // page will be use SftTabs_GetNextTab(hwndCtl).
                    if (SftTabs_GetNextTab(hwndCtl) == 0)
                        SendMessage(hwndCtl, WM_CANCELMODE, 0, 0);
                    if (!SftTabs_DeactivatePage(hwndParent, hwndCtl))
                        // couldn't deactivate current page, so don't switch
                        SendMessage(hwndCtl, WM_CANCELMODE, 0, 0);
                    break;
                case SFTTABSN_SWITCHED:// we switched to a new page
                    SftTabs_ActivatePage(hwndParent, hwndCtl, NULL, FALSE);
                    break;
                }
                break;
           }
        }
        break;
       }
```

**C++/MFC**

```
BOOL CPage2::AllowSwitch()
{
```

```
    if (m_pTabCtl->GetNextTab() == 0))
        return FALSE;
    return TRUE;                        // Allow switching away from this page
}
```

**C++/OWL**

```
bool TPage2::CanClose ()
{
    if (m_pTabCtl->GetNextTab() == 0))
        return false;
    return true;                        // Allow switching away from this page
}
```

## SftTabs_GetStyleTable

```
LPSFTTABS_STYLETABLEA WINAPI SftTabs_GetStyleTable(void);
```

This function returns a pointer to the first entry in the SftTabs DLL's style table. The style table describes all available tab styles, suitable for use by a resource editor.

**Returns**

The return value is a pointer to the first entry in the SftTabs DLL's style table. Each entry is of type SFTTABS_STYLETABLEA. The *lpszDesc* member of the last entry in the table is NULL.

## SftTabs_GetTabControlFromPage

```
HWND WINAPI SftTabs_GetTabControlFromPage(HWND hwndPage);
```

This function returns the tab control window handle, given the window handle of a window attached to a tab.

**Parameters**

*hwndPage*

The window handle of the window attached to a tab.

**Returns**

The return value is the window handle of the tab control if successful, otherwise NULL is returned.

# GetTabDialog

**C++, MFC, CSftTabs::GetTabDialog**

```
CSftTabsPage* GetTabDialog(int iTab = -1) const;
```

**C++, OWL, TSftTabs::GetTabDialog**

```
TSftTabsPage* GetTabDialog(int iTab = -1) const;
```

Retrieves the CSftTabsPage or TSftTabsPage based object attached to the specified tab.

**Parameters**

*iTab*

The zero-based index of the tab for which information is to be retrieved. If -1 is specified, the information for the currently active tab is retrieved.

**Returns**

The return value is a pointer to the CSftTabsPage or TSftTabsPage based object, attached to the specified tab or NULL if no page is attached. The CSftTabsPage or TSftTabsPage based object is set using SetTabDialog.

# GetTabInfo

**C, SFTTABSM_GETTABINFO**

```
BOOL SftTabs_GetTabInfo(HWND hwnd, int iTab, LPSFTTABS_TAB lpTab);
```

**C++, CSftTabs::GetTabInfo, TSftTabs::GetTabInfo**

```
BOOL GetTabInfo(int iTab, LPSFTTABS_TAB lpTab) const;
```

Retrieves tab control attributes.

**Parameters**

*iTab*

The zero-based index of the tab for which information is to be retrieved.

*lpTab*

A pointer to a SFTTABS_TAB structure. This structure will be updated with the specified tab's attributes.

**Returns**

The return value is TRUE if the function was successful, otherwise FALSE is returned.

**Comments**

Some of the structure values returned can be modified and updated using SetTabInfo. See SFTTABS_TAB for more information.

**Example**

This example retrieves the tab attributes for the third tab and modifies the background color:

**C**

```
SFTTABS_TAB Tab;
SftTabs_GetTabInfo(hwndTab, 2, &Tab);
Tab.colorBg = RGB(255, 0, 0);
SftTabs_SetTabInfo(hwndTab, 2, &Tab);
```

**C++/MFC**

```
SFTTABS_TAB Tab;
m_Tab.GetTabInfo(2, &Tab);
Tab.colorBg = RGB(255, 0, 0);
m_Tab.SetTabInfo(2, &Tab);
```

**C++/OWL**

```
SFTTABS_TAB Tab;
pTab->GetTabInfo(2, &Tab);
Tab.colorBg = RGB(255, 0, 0);
pTab->SetTabInfo(2, &Tab);
```

# GetTabLabel

**C, WIN16, SFTTABSM_GETTABLABEL**

```
int SftTabs_GetTabLabel(HWND hwnd, int iTab, LPSTR lpsz);
```

**C, WIN32, SFTTABSM_GETTABLABEL, _A, _W**

```
int SftTabs_GetTabLabel(HWND hwnd, int iTab, LPTSTR lpsz);
int SftTabs_GetTabLabel_A(HWND hwnd, int iTab, LPSTR lpsz);
int SftTabs_GetTabLabel_W(HWND hwnd, int iTab, LPWSTR lpsz);
```

**C++, WIN16, CSftTabs::GetTabLabel, TSftTabs::GetTabLabel**

```
int GetTabLabel(int iTab, LPSTR lpsz) const;
```

**C++, WIN32, CSftTabs::GetTabLabel, TSftTabs::GetTabLabel**

```
int GetTabLabel(int iTab, LPTSTR lpsz) const;
```

Retrieves a tab's text.

**Parameters**

*iTab*

The zero-based index of the tab for which information is to be retrieved.

*lpsz*

A pointer to a buffer where the tab's text will be returned.

**Returns**

The return value is the number of characters returned in the buffer, not including the terminating '\0'. The buffer must be large enough to receive the complete text. GetTabLabelLen can be used to determine the buffer length needed. -1 is returned if an error occurred.

**Comments**

A tab's text can be changed using SetTabLabel.

**Example**

This example retrieves the text of the second tab:

**C**

```
char szBuffer[80];
SftTabs_GetTabLabel(hwndTab, 1, szBuffer);
```

**C++/MFC**

```
char szBuffer[80];
m_Tab.GetTabLabel(1, szBuffer);
```

**C++/OWL**

```
char szBuffer[80];
pTab->GetTabLabel(hwndTab, 1, szBuffer);
```

# GetTabLabelLen

**C, SFTTABSM_GETTABLABELLEN**

```
int SftTabs_GetTabLabelLen(HWND hwnd, int iTab);
```

**C++, CSftTabs::GetTabLabelLen, TSftTabs::GetTabLabelLen**

```
int GetTabLabelLen(int iTab) const;
```

Retrieves the length of a tab's text.

**Parameters**

*iTab*

The zero-based index of the tab for which the text length is to be retrieved.

**Returns**

The return value is the length of the tab's text, not including the terminating '\0' or -1 if an error occurred.

**Comments**

When using the UNICODE-enabled DLL SFTTB32U.DLL, the number returned is the number of wide characters, not the number of bytes.

**Example**

This example retrieves the length of the tenth tab's text:

**C**

```
len = SftTabs_GetTabLabelLen (hwndTab, 9);
```

**C++/MFC**

```
len = m_Tab.GetTabLabelLen(9);
```

**C++/OWL**

```
len = pTab->GetTabLabelLen(9);
```

# GetTabText

**C++, MFC only, CSftTabs::GetTabText**

```
void GetTabText(int iTab, CString& string) const;
```

Retrieves a tab's text.

**Parameters**

*iTab*

The zero-based index of the tab for which information is to be retrieved.

*string*

A reference to a CString object, where the text will be returned.

**Comments**

A tab's text can be changed using SetTabLabel.

**Example**

This example retrieves the text of the second tab:

**C++/MFC**

```
CString str;
m_Tab.GetTabText(1, &str);
```

# GetTabWindowPage

**C++, MFC, CSftTabs::GetTabWindowPage**

```
CSftTabsWindowPage* GetTabWindowPage(int iTab = -1) const;
```

**C++, OWL, TSftTabs::GetTabWindowPage**

```
TSftTabsWindowPage* GetTabWindowPage(int iTab = -1) const;
```

Retrieves the CSftTabsWindowPage or TSftTabsWindowPage based object attached to the specified tab.

**Parameters**

*iTab*

The zero-based index of the tab for which information is to be retrieved. If -1 is specified, the information for the currently active tab is retrieved.

**Returns**

The return value is a pointer to the CSftTabsWindowPage or TSftTabsWindowPage based object, attached to the specified tab or NULL if no page is attached. The CSftTabsWindowPage or TSftTabsWindowPage based object is set using SetTabWindowPage.

# GetToolTip

**C, WIN16, SFTTABSM_GETTOOLTIP**

```
int SftTabs_GetToolTip(HWND hwnd, int iTab, LPSTR lpsz);
```

**C, WIN32, SFTTABSM_GETTOOLTIP, _A, _W**

```
int SftTabs_GetToolTip(HWND hwnd, int iTab, LPTSTR lpsz);
int SftTabs_GetToolTip_A(HWND hwnd, int iTab, LPSTR lpsz);
int SftTabs_GetToolTip_W(HWND hwnd, int iTab, LPWSTR lpsz);
```

**C++, MFC only, CSftTabs::GetToolTip**

```
void GetToolTip(int iTab, CString& string) const;
```

**C++, WIN16, CSftTabs::GetToolTip, TSftTabs::GetToolTip**

```
int GetToolTip(int iTab, LPSTR lpsz) const;
```

**C++, WIN32, CSftTabs::GetToolTip, TSftTabs::GetToolTip**

```
int GetToolTip(int iTab, LPTSTR lpsz) const;
```

Retrieves a tab's tooltip text.

**Parameters**

*iTab*

The zero-based index of the tab for which information is to be retrieved.

*lpsz*

A pointer to a buffer where the tab's tooltip text will be returned.

**Returns**

The return value is the number of characters returned in the buffer, not including the terminating '\0'. The buffer must be large enough to receive the complete text. GetToolTipLen can be used to determine the buffer length needed. -1 is returned if an error occurred.

**Comments**

A tab's tooltip text can be changed using SetToolTip.

**Example**

This example retrieves the text of the second tab's tooltip:

**C**

```
char szBuffer[80];
SftTabs_GetToolTip(hwndTab, 1, szBuffer);
```

**C++/MFC**

```
char szBuffer[80];
m_Tab.GetToolTip(1, szBuffer);
```

**C++/OWL**

```
char szBuffer[80];
pTab->GetToolTip(hwndTab, 1, szBuffer);
```

# SftTabs_GetToolTipHandle

**C, SFTTABSM_GETTOOLTIPHANDLE**

```
HWND SftTabs_GetToolTipHandle(HWND hwnd);
```

Retrieves the tooltip control window handle.

**Returns**

The return value is the window handle of the tooltip control which was created by the tab control to display tooltips for each tab..  NULL is returned if no tooltip control has been created.

**Comments**

Tooltips are only available in 32-bit applications running on Windows 95 or Windows NT 3.51 and higher.

**Example**

This example retrieves the window handle of the tooltip control:

**C**

```
HWND hwndCtl;
hwndCtl = SftTabs_GetToolTipHandle(hwndTab);
```

# GetToolTipLen

**C, SFTTABSM_GETTOOLTIPLEN**

```
int SftTabs_GetToolTipLen(HWND hwnd, int iTab);
```

**C++, CSftTabs::GetToolTipLen, TSftTabs::GetToolTipLen**

```
int GetToolTipLen(int iTab) const;
```

Retrieves the length of a tab's tooltip text.

**Parameters**

*iTab*

The zero-based index of the tab for which the tooltip text length is to be retrieved.

**Returns**

The return value is the length of the tab's tooltip text, not including the terminating '\0' or -1 if an error occurred.

**Comments**

When using the UNICODE-enabled DLL SFTTB32U.DLL, the number returned is the number of wide characters, not the number of bytes.

**Example**

This example retrieves the length of the tenth tab's tooltip text:

**C**

```
len = SftTabs_GetToolTipLen (hwndTab, 9);
```

**C++/MFC**

```
len = m_Tab.GetToolTipLen(9);
```

**C++/OWL**

```
len = pTab->GetToolTipLen(9);
```

# SftTabs_HandleDialogMessage

```
BOOL WINAPI SftTabs_HandleDialogMessage(HWND hwnd, UINT msg,
                                        WPARAM wParam, LPARAM lParam);
```

The parent dialog window of a tab control and the dialogs used as pages of a tab control call SftTabs_HandleDialogMessage to pass messages on to SftTabs/DLL so they can be processed.

**Parameters**

*hwnd*
The window handle of the destination window.

*msg*
Message ID.

*wParam, lParam*
Message parameters.

**Returns**

The return value is TRUE if the message was processed by SftTabs/DLL, otherwise FALSE.

**Comments**

If this function is not called, certain features of SftTabs/DLL may not appear to be working correctly, such as accelerator keys, tab switching, ESCAPE and TAB key handling, etc. For windows (as opposed to dialogs) with tab controls, use the SftTabs_HandleWindowMessage function instead.

**Example**

This C example shows a dialog procedure for a dialog page:

```
BOOL _export CALLBACK Page1_DialogProc(HWND hwndDlg, UINT msg,
                                       WPARAM wParam, LPARAM lParam)
{
    switch (msg) {
    case WM_INITDIALOG:
        SetWindowText(GetDlgItem(hwndDlg, IDC_P1_EDIT1), TEXT("Click another tab"));
        SendMessage(GetDlgItem(hwndDlg, IDC_P1_CHECK1), BM_SETCHECK, 1, 0);
        // initialize page
        SftTabs_SetPageActive(hwndDlg, (HWND) lParam, NULL);
        return !SftTabs_IsRegisteredDialog(GetParent(hwndDlg));
    }
    // Any message your dialog procedure doesn't handle, must come here
    if (SftTabs_HandleDialogMessage(hwndDlg, msg, wParam, lParam))
        return TRUE;

    return FALSE;
}
```

## SftTabs_HandleWindowMessage

```
BOOL WINAPI SftTabs_HandleWindowMessage(HWND hwnd, UINT message,
                                WPARAM wParam, LPARAM lParam, LRESULT FAR* lplResult);
```

The parent window of a tab control and the windows or dialogs used as pages of a tab control call SftTabs_HandleWindowMessage to pass messages on to SftTabs/DLL so they can be processed.

**Parameters**

*hwnd*
The window handle of the destination window.

*msg*
Message ID.

*wParam, lParam*
Message parameters.

*lplResult*
Pointer to an LRESULT value. This field will be set to the result of the processed message.

**Returns**

The return value is TRUE if the message was processed by SftTabs/DLL, otherwise FALSE.

**Comments**

If this function is not called, certain features of SftTabs/DLL may not appear to be working correctly, such as accelerator keys, tab switching, ESCAPE and TAB key handling, etc. For dialogs (as opposed to windows) with tab controls, use the SftTabs_HandleDialogMessage function instead.

**Example**

This C++/MFC example handles unprocessed messages for a CView based window containing a tab control:

```
LRESULT CSampleView::WindowProc(UINT message, WPARAM wParam, LPARAM lParam)
{
    LRESULT lRes;

    if (SftTabs_HandleWindowMessage(m_hWnd, message, wParam, lParam, &lRes))
        return lRes;
    // call base class
    return CView::WindowProc(message, wParam, lParam);
}
```

# InsertTab

**C, WIN16, SFTTABSM_INSERTTAB**

```
int SftTabs_InsertTab(HWND hwnd, int iTab, LPCSTR lpsz);
```

**C, WIN32, SFTTABSM_INSERTTAB, _A, _W**

```
int SftTabs_InsertTab(HWND hwnd, int iTab, LPCTSTR lpsz);
int SftTabs_InsertTab_A(HWND hwnd, int iTab, LPCSTR lpsz);
int SftTabs_InsertTab_W(HWND hwnd, int iTab, LPCWSTR lpsz);
```

**C++, WIN16, CSftTabs::InsertTab, TSftTabs::InsertTab**

```
int InsertTab(int iTab, LPCSTR lpszText);
```

**C++, WIN32, CSftTabs::InsertTab, TSftTabs::InsertTab**

```
int InsertTab(int iTab, LPCTSTR lpszText);
```

Adds a new tab at the specified position.

**Parameters**

*iTab*

The zero-based index of the tab to be added. If -1 is specified, the tab will be added at the end.

*lpsz*

Points to the null-terminated string that is to be used as text for the tab label.

**Returns**

The return value is the zero-based index of the newly added tab. The return value is -1 if an error occurred.

**Comments**

The tab control creates a copy of the string supplied.

The WM_SETREDRAW Windows message can be used to suppress the tab control from being redrawn when many tabs are added.

Tabs can be deleted using DeleteTab. Tabs can be added using AddTab.

**Example**

This example inserts a tab with the text "A Test" at the third position:

**C**

```
index = SftTabs_InsertTab(hwndTab, 2, "A Test");
```

**C++/MFC**

```
index = m_Tab.InsertTab(2, "A Test");
```

**C++/OWL**

```
index = pTab->InsertTab(2, "A Test");
```

# SftTabs_IsRegisteredDialog / -Window

```
BOOL WINAPI SftTabs_IsRegisteredDialog(HWND hwndDialog);
BOOL WINAPI SftTabs_IsRegisteredWindow(HWND hwndWnd);
```

This function determines if a given window or dialog is registered with SftTabs/DLL for special tabbed dialog or tabbed window handling, such as accelerator key handling, ESCAPE and TAB key processing, etc.

**Parameters**

*hwndDialog, hwndWnd*

The window handle of the window or dialog to be tested.

**Returns**

The return value is TRUE if the window is registered with SftTabs/DLL for special tabbed dialog or window handling, otherwise FALSE is returned.

**Comments**

A main tabbed dialog or window containing a tab control is registered using SftTabs_RegisterWindow or SftTabs_RegisterDialog. Windows and dialogs based on the C++ classes CSftTabsDialog and TSftTabsDialog are automatically registered.

## SftTabs_IsTabControl

```
BOOL WINAPI SftTabs_IsTabControl(HWND hwndCtl);
```

This function determines if a given window is a tab control.

**Parameters**

*hwndCtl*

The window handle of the window to be tested.

**Returns**

The return value is TRUE if the window is a tab control, otherwise FALSE is returned.

## SftTabs_IsTabControlWithDialog / -Page

```
BOOL WINAPI SftTabs_IsTabControlWithDialog(HWND hwndCtl);
BOOL WINAPI SftTabs_IsTabControlWithPage(HWND hwndCtl);
```

This function determines if a given window is a tab control with an attached page.

**Parameters**

*hwndCtl*

The window handle of the window to be tested.

**Returns**

The return value is TRUE if the window *hwndCtl* is a tab control with an attached page (*hwndSubDlg* in SFTTABS_CONTROL is not NULL), otherwise FALSE is returned.

**Comments**

SftTabs_IsTabControlWithPage is a synonym for SftTabs_IsTabControlWithDialog and works the same way for tab controls in a tabbed dialog or a tabbed window.

# QueryChar

**C, WIN16, SFTTABSM_QUERYCHAR**

```
BOOL SftTabs_QueryChar(HWND hwnd, int ch);
```

**C, WIN32, SFTTABSM_QUERYCHAR, _A, _W**

```
BOOL SftTabs_QueryChar(HWND hwnd, TCHAR ch);
BOOL SftTabs_QueryChar_A(HWND hwnd, int ch);
BOOL SftTabs_QueryChar_W(HWND hwnd, WCHAR ch);
```

**C++, WIN16, CSftTabs::QueryChar, TSftTabs::QueryChar**

```
int QueryChar(int ch) const;
```

**C++, WIN32, CSftTabs::QueryChar, TSftTabs::QueryChar**

```
int QueryChar(TCHAR ch) const;
```

Tests if a tab control responds to the specified character, i.e. the character is an accelerator key which the tab control processes.

**Parameters**

*ch*

The character to be tested.

**Returns**

The return value is TRUE if the tab control responds to the specified character, otherwise FALSE.

# RegisterApp

**C, SftTabs_RegisterApp**

```
BOOL WINAPI SftTabs_RegisterApp(HINSTANCE hInst);
```

**C++, CSftTabs::RegisterApp, TSftTabs::RegisterApp**

```
static BOOL RegisterApp();
```

An application calls this function to register the application for use of SftTabs/DLL controls.

**Parameters**

*hInst*

The instance handle of the application, which will use SftTabs/DLL controls.

**Returns**

The return value is TRUE if SftTabs/DLL has been initialized for this application, otherwise FALSE is returned.

**Comments**

This call allows SftTabs/DLL to register all required window classes for the calling application. This call has to be made before any SftTabs/DLL controls are created.

An application should call UnregisterApp once the application no longer uses SftTabs/DLL controls.

The call to this function should be made during application initialization.

**Example**

This example registers an application with SftTabs/DLL:

**C**

```
int PASCAL WinMain(HINSTANCE hinst, HINSTANCE hinstPrev, LPSTR Cmd, int cmdShow)
{
    SftTabs_RegisterApp(hinst);         // Register application

    .... application message loop

    SftTabs_UnregisterApp(hinst);       // Unregister application
    return msg.wParam;
}
```

**C++/MFC**

```
BOOL CSampleApp::InitInstance()   // based on CWinApp
{
    CSftTabs::RegisterApp();      // Register to use SftTabs/DLL
    .... other initialization
    return TRUE;
}
```

**C++/OWL**

```
void TApp::InitInstance()     // based on TApplicaton
{
    // Register to use SftTabs/DLL
    TSftTabs::RegisterApp();
    // call base class
    TApplication::InitInstance();
}
```

# SftTabs_RegisterDialog / -Window

```
BOOL WINAPI SftTabs_RegisterDialog(HWND hwndDialog);
BOOL WINAPI SftTabs_RegisterWindow(HWND hwndWnd);
```

An application can call this function to register a window or dialog containing a tab control. Once registered, SftTabs/DLL will perform special tabbed dialog handling, such as accelerator key handling, ESCAPE and TAB key processing, etc.

**Parameters**

*hwndDialog, hwndWnd*

The window handle of the window or dialog to be registered.

**Returns**

The return value is TRUE if the window is successfully registered with SftTabs/DLL.

**Comments**

If this function is not called, certain features of SftTabs/DLL may not appear to be working correctly, such as accelerator keys, tab switching, ESCAPE key handling, etc.

A window or dialog registered using this function, must also be unregistered using SftTabs_UnregisterDialog or SftTabs_UnregisterWindow.

**Example**

This C example shows the end of a typical tabbed dialog WM_INITDIALOG message handler:

```
      ... additional initialization code ...

    index = SftTabs_AddTab(hwndTab, TEXT("&Six"));
    SftTabs_SetTabInfo(hwndTab, index, &Tab5);

    SftTabs_SetControlInfo(hwndTab, &CtlInit);

    // Make sure to turn redraw back on
    SendMessage(hwndTab, WM_SETREDRAW, (WPARAM)TRUE, 0);
    InvalidateRect(hwndTab, NULL, TRUE);

    // Activate current page.
    SftTabs_ActivatePage(hwndParent, hwndTab, NULL, TRUE);

    // Mark the window as a main, tabbed dialog (so accel. keys work) by registering
it.
    // Register the dialog AFTER activating the current page
    SftTabs_RegisterDialog(hwndParent);

    return FALSE;                       // WM_INITDIALOG, input focus already set
```

# ResetContent

**C, SFTTABSM_RESETCONTENT**

```
void SftTabs_ResetContent(HWND hwnd);
```

**C++, CSftTabs::ResetContent, TSftTabs::ResetContent**

```
void ResetContent();
```

Removes all tabs from a tab control.

**Comments**

A tab control without tabs no longer paints a tab border or client area and becomes transparent.

# ResizePages

**C, SFTTABSM_RESIZEPAGES**

```
void SftTabs_ResizePages(HWND hwnd);
```

**C++, CSftTabs::ResizePages, TSftTabs::ResizePages**

```
void ResizePages();
```

Resizes attached pages when using a frame window.

**Comments**

ResizePages should be used whenever a frame window has been resized. A frame window is used when a tab control does not have a client area. That frame window "holds" the pages that are attached to a tab control. If the user or the application resizes this frame window, ResizePages must be called so the tab control can adjust the sizes of all attached pages.

A frame window is defined using SftTabs_ActivatePage, SetControlInfo, CSftTabsDialog::InitializeTabControl, TSftTabsDialog::InitializeTabControl, CSftTabsWindowSheet::InitializeTabControl or TSftTabsWindowSheet::InitializeTabControl.

# ScrollTabs

**C, SFTTABSM_SCROLLTABS**

```
int SftTabs_ScrollTabs(HWND hwnd, BOOL fUpOrLeft);
```

**C++, CSftTabs::ScrollTabs, TSftTabs::ScrollTabs**

```
int ScrollTabs(BOOL fUpOrLeft);
```

Scrolls tabs in the direction specified.

**Parameters**

*fUpOrLeft*

TRUE to scroll left (or up in a vertical tab control), FALSE to scroll right or down.

**Returns**

The return value is the index of the new leftmost (topmost) tab visible, or -1 if an error occurred.

**Comments**

SftTabs_ScrollTabs can only be used with scrollable tab controls. The members *fLeftButton* and *fRightButton* of the SFTTABS_CONTROL structure can be tested to see if scrolling in either direction is currently possible.

**Example**

This example scrolls all tabs left by one position:

**C**

```
SftTabs_ScrollTabs(hwndTab, TRUE);
```

**C++/MFC**

```
m_Tab.ScrollTabs(TRUE);
```

**C++/OWL**

```
pTab->ScrollTabs(TRUE);
```

# SetControlInfo

## SetControlInfo

**C, SFTTABSM_SETCONTROLINFO**

```
BOOL SftTabs_SetControlInfo(HWND hwnd, LPCSFTTABS_CONTROL lpCtl);
```

**C++, CSftTabs::SetControlInfo, TSftTabs::SetControlInfo**

```
BOOL SetControlInfo(LPCSFTTABS_CONTROL lpCtl);
```

Sets tab control attributes.

**Parameters**

*lpCtl*

A pointer to a SFTTABS_CONTROL structure. This structure will be used to define the new tab control attributes.

**Returns**

The return value is TRUE if the function was successful, otherwise FALSE is returned.

**Comments**

If the SFTTABS_CONTROL structure defines a tab control that is invalid, the current tab control settings remain unchanged and the function returns FALSE. The following validations take place to insure that a tab control is valid:

- the style specified in *style* has to be valid
- *nRows* must be at least 1 and less or equal to the number of tabs
- if *nRows* is > 1, the new tab style must support multiple rows of tabs
- if *leftMargin* or *rightMargin* is > 1, the new tab style must support margins
- if a specific number of tabs per row is defined (*nRowTabs*), tabs must be defined as fixed width tabs (*fFixed*)
- if a client area is requested (*fClientArea*), the new tab style must support a client area
- if a scrollable tab control is requested (*fScrollable*), the new tab style must support scrollable tabs
- if a scrollable tab control is requested (*fScrollable*), the number of tab rows requested must be one (*nRows*)
- if a scrollable tab control is requested (*fScrollable*), the tab rows cannot be filled completely (*fFillComplete*)
- if a scroll button bitmap handle is supplied (*hButtonBitmap*), the handle must be valid
- if hidden scroll buttons are requested (*fHideScrollButtons*), the tab control must be defined as scrollable (*fScrollable*)
- if multiline tab text is requested, the new tab style must support multiline tab text
- if conditional scroll buttons are requested (*fCondScrollButtons*), the tab control must be defined as scrollable (*fScrollable*)
- conditional scroll buttons (*fCondScrollButtons*) and hidden scroll buttons (*fHideScrollButtons*) are mutually exclusive

**Example**

This example retrieves the current tab control attributes and modifies the number of tab rows:

**C**

```
SFTTABS_CONTROL Ctl;
SftTabs_GetControlInfo(hwndTab, &Ctl);
Ctl.nRows = 1;
SftTabs_SetControlInfo(hwndTab, &Ctl);
```

**C++/MFC**

```
SFTTABS_CONTROL Ctl;
m_Tab.GetControlInfo(&Ctl);
Ctl.nRows = 1;
m_Tab.SetControlInfo(&Ctl);
```

**C++/OWL**

```
SFTTABS_CONTROL Ctl;
pTab->GetControlInfo(&Ctl);
```

```
Ctl.nRows = 1;
pTab->SetControlInfo(&Ctl);
```

# SetCtlColors

**C, SFTTABSM_SETCTLCOLORS**

```
void SftTabs_SetCtlColors(HWND hwnd, LPCSFTTABS_COLORS lpColors);
```

**C++, CSftTabs::SetCtlColors, TSftTabs::SetCtlColors**

```
void SetCtlColors(LPCSFTTABS_COLORS lpColors) const;
```

Sets the tab control's color attributes.

**Parameters**

*lpColors*

A pointer to a SFTTABS_COLORS structure containing the color definitions.

**Comments**

Using GetCtlColors and SetCtlColors is the preferred method to change color attributes. Although a tab control generates WM_CTLCOLOR messages, the WM_CTLCOLOR message handling is provided for compatibility with SftTabs 2.0 only.

**Example**

This example changes the tab control's foreground and background colors.

**C**

```
SFTTABS_COLORS Colors;
SftTabs_GetCtlColors(hwndTab, &Colors);   /* Get current color settings */
Colors.colorBg = RGB(0,255,255);          /* Background color */
Colors.colorFg = RGB(0,0,128);            /* Foreground color */
SftTabs_SetCtlColors(hwndTab, &Colors);   /* Set new colors */
```

**C++/MFC**

```
SFTTABS_COLORS Colors;
m_Tabs.GetCtlColors(&Colors);    /* Get current color settings */
Colors.colorBg = RGB(0,255,255); /* Background color */
Colors.colorFg = RGB(0,0,128);   /* Foreground color */
m_Tabs.SetCtlColors(&Colors);    /* Set new colors */
```

**C++/OWL**

```
SFTTABS_COLORS Colors;
pTab->GetCtlColors(&Colors);     /* Get current color settings */
Colors.colorBg = RGB(0,255,255); /* Background color */
Colors.colorFg = RGB(0,0,128);   /* Foreground color */
pTab->SetCtlColors(&Colors);     /* Set new colors */
```

# SetCurrentTab

**C, SFTTABSM_SETCURRENTTAB**

```
int SftTabs_SetCurrentTab(HWND hwnd, int iTab);
```

**C++, CSftTabs::SetCurrentTab, TSftTabs::SetCurrentTab**

```
int SetCurrentTab(int iTab);
```

Makes the specified tab the new active tab.

**Returns**

The return value is the index of the new active tab, otherwise -1 is returned.

**Comments**

When using this function to activate a new tab, the normal tab switching mechanism takes place, such as calling the SFTTABS_TABCALLBACK tab callback function, the CSftTabsPage::AllowSwitch, CSftTabsWindowPage::AllowSwitch, TSftTabsPage::CanClose or TSftTabsWindowPage::AllowSwitch member functions of the C++ based implementation of tabbed dialog.

**Example**

This example makes the third tab the new active tab:

**C**

```
SftTabs_SetCurrentTab(hwndTab, 2);
```

**C++/MFC**

```
m_Tab.SetCurrentTab(2);
```

**C++/OWL**

```
pTab->SetCurrentTab(2);
```

# SftTabs_SetPageActive

```
void WINAPI SftTabs_SetPageActive(HWND hwndPage, HWND hwndTab, LPVOID lpTabData);
```

An application calls the SftTabs_SetPageActive function to notify SftTabs/DLL that the page attached to the currently active tab has been activated.

**Parameters**

*hwndPage*

The window handle of the page attached to the currently active tab. This value is saved in the *hwndSubDlg* member of the SFTTABS_CONTROL structure.

*hwndTab*

The window handle of the tab control.

*lpTabData*

An application defined 32-bit value. This value is saved in the *lpTabData* member of the SFTTABS_CONTROL structure. For the C++ tabbed dialog and window implementation, this is the page object, otherwise this parameter should be NULL.

**Comments**

The page is automatically resized to fit inside the tab control's client area (or a supplied frame window, see SFTTABS_CONTROL), certain incompatible window styles are changed and the page is made visible. The call to SftTabs_SetPageActive should always be performed in the page's WM_INITDIALOG message handler. For C++, the call is automatic when using the supplied classes.

**Example**

This C example shows a page (subdialog) about to become active:

```
switch (msg) {
    case WM_INITDIALOG: {
        HWND hwndTab = (HWND) lParam;// get the associated tab control
        // initialize page
        SftTabs_SetPageActive(hwndDlg, (HWND) lParam, NULL);
        return FALSE;
    }
}
```

## SftTabs_SetPageInactive

```
void WINAPI SftTabs_SetPageInactive(HWND hwndTab);
```

An application calls the SftTabs_SetPageInactive function to notify SftTabs/DLL that the page attached to the currently active tab has been deactivated.

**Parameters**

*hwndTab*

The window handle of the tab control.

# SetTabDialog

**C++, MFC, CSftTabs::SetTabDialog**

```
void SetTabDialog(int iTab, CSftTabsPage* pPage);
```

**C++, OWL, TSftTabs::SetTabDialog**

```
void SetTabDialog(int iTab, TSftTabsPage* pPage);
```

Sets the CSftTabsPage or TSftTabsPage based object pointer attached to the specified tab.

**Parameters**

*iTab*

The zero-based index of the tab for which information is to be set.

*pPage*

A pointer to the CSftTabsPage or TSftTabsPage based object representing the page attached to the tab specified.

**Comments**

This function is used by the CSftTabsDialog and CSftTabsPage, TSftTabsDialog and TSftTabsPage class implementation. When a tab is made the active tab, the CSftTabsPage or TSftTabsPage based dialog is created or made visible.

# SetTabInfo

**C, SFTTABSM_SETTABINFO**

```
BOOL SftTabs_SetTabInfo(HWND hwnd, int iTab, LPSFTTABS_TAB lpTab);
```

**C++, CSftTabs::SetTabInfo, TSftTabs::SetTabInfo**

```
BOOL SetTabInfo(int iTab, LPCSFTTABS_TAB lpTab);
```

Sets the tab information for the specified tab.

**Parameters**

*iTab*

The zero-based index of the tab for which attributes are to be defined.

*lpTab*

A pointer to a SFTTABS_TAB structure. This structure will be used to define the tab attributes.

**Returns**

The return value is TRUE if the function was successful, otherwise FALSE is returned.

**Comments**

If the SFTTABS_TAB structure defines invalid tab attributes, the current tab settings remain unchanged and the function returns FALSE.

**Example**

This example retrieves the tab attributes for the third tab and modifies the background color:

**C**

```
SFTTABS_TAB Tab;
SftTabs_GetTabInfo(hwndTab, 2, &Tab);
Tab.colorBg = RGB(255, 0, 0);
SftTabs_SetTabInfo(hwndTab, 2, &Tab);
```

**C++/MFC**

```
SFTTABS_TAB Tab;
m_Tab.GetTabInfo(2, &Tab);
Tab.colorBg = RGB(255, 0, 0);
m_Tab.SetTabInfo(2, &Tab);
```

**C++/OWL**

```
SFTTABS_TAB Tab;
pTab->GetTabInfo(2, &Tab);
Tab.colorBg = RGB(255, 0, 0);
pTab->SetTabInfo(2, &Tab);
```

# SetTabLabel

**C, WIN16, SFTTABSM_SETTABLABEL**

```
BOOL SftTabs_SetTabLabel(HWND hwnd, int iTab, LPCSTR lpsz);
```

**C, WIN32, SFTTABSM_SETTABLABEL, _A, _W**

```
BOOL SftTabs_SetTabLabel(HWND hwnd, int iTab, LPCTSTR lpsz);
BOOL SftTabs_SetTabLabel_A(HWND hwnd, int iTab, LPCSTR lpsz);
BOOL SftTabs_SetTabLabel_W(HWND hwnd, int iTab, LPCWSTR lpsz);
```

**C++, WIN16, CSftTabs::SetTabLabel, TSftTabs::SetTabLabel**

```
BOOL SetTabLabel(int iTab, LPCSTR lpsz);
```

**C++, WIN32, CSftTabs::SetTabLabel, TSftTabs::SetTabLabel**

```
BOOL SetTabLabel(int iTab, LPCTSTR lpsz);
```

Sets a tab's text.

**Parameters**

*iTab*

The zero-based index of the tab for which the tab text is to be set.

*lpsz*

A pointer to a buffer containing the tab's text.

**Returns**

The return value is TRUE if the function was successful, otherwise FALSE is returned.

**Comments**

The tab control creates a copy of the string supplied.

A tab's text can be retrieved using GetTabLabel.

**Example**

This example sets the text of the second tab:

**C**

```
SftTabs_SetTabLabel(hwndTab, 1, "New Text");
```

**C++/MFC**

```
m_Tab.SetTabLabel(1, "New Text");
```

**C++/OWL**

```
pTab->SetTabLabel(1, "New Text");
```

# SetTabWindowPage

**C++, MFC, CSftTabs::SetTabWindowPage**

```
void SetTabWindowPage(int iTab, CSftTabsWindowPage* pPage, HWND hwnd = NULL);
```

**C++, OWL, TSftTabs::SetTabWindowPage**

```
void SetTabWindowPage(int iTab, TSftTabsWindowPage* pPage, HWND hwnd = NULL);
```

Sets the CSftTabsWindowPage or TSftTabsWindowPage based object pointer attached to the specified tab.

**Parameters**

*iTab*

The zero-based index of the tab for which information is to be set.

*pPage*

A pointer to the CSftTabsWindowPage or TSftTabsWindowPage based object representing the page attached to the tab specified.

**Comments**

This function is used by the CSftTabsWindowSheet and CSftTabsWindowPage, TSftTabsWindowSheet and TSftTabsWindowPage class implementation. When a tab is made the active tab, the CSftTabsWindowPage or TSftTabsWindowPage based dialog is created or made visible.

# SetToolTip

**C, WIN16, SFTTABSM_SETTOOLTIP**

```
BOOL SftTabs_SetToolTip(HWND hwnd, int iTab, LPCSTR lpsz);
```

**C, WIN32, SFTTABSM_SETTOOLTIP, _A, _W**

```
BOOL SftTabs_SetToolTip(HWND hwnd, int iTab, LPCTSTR lpsz);
BOOL SftTabs_SetToolTip_A(HWND hwnd, int iTab, LPCSTR lpsz);
BOOL SftTabs_SetToolTip_W(HWND hwnd, int iTab, LPCWSTR lpsz);
```

**C++, WIN16, CSftTabs::SetToolTip, TSftTabs::SetToolTip**

```
BOOL SetToolTip(int iTab, LPCSTR lpsz);
```

**C++, WIN32, CSftTabs::SetToolTip, TSftTabs::SetToolTip**

```
BOOL SetToolTip(int iTab, LPCTSTR lpsz);
```

Sets a tab's tooltip text.

**Parameters**

*iTab*

The zero-based index of the tab for which the tooltip text is to be set.

*lpsz*

A pointer to a buffer containing the tab's tooltip text.

**Returns**

The return value is TRUE if the function was successful, otherwise FALSE is returned.

**Comments**

The tab control creates a copy of the string supplied.

A tab's text can be retrieved using GetToolTip.

**Example**

This example sets the text of the second tab's tooltip:

**C**

```
SftTabs_SetToolTip(hwndTab, 1, "New Text");
```

**C++/MFC**

```
m_Tab.SetToolTip(1, "New Text");
```

**C++/OWL**

```
pTab->SetToolTip(1, "New Text");
```

# SetVersion

**C, WIN16, SFTTABSM_SETVERSION**

```
BOOL SftTabs_SetVersion(HWND hwnd, int version);
```

**C++, CSftTabs::SetVersion, TSftTabs::SetVersion**

```
BOOL SetVersion(int version);
```

Sets a SftTabs/DLL version an application requires.

**Parameters**

*version*

A value indicating for which SftTabs/DLL version the application was developed.

| | |
|---|---|
| SFTTABS_2_0 | The application was developed for use with SftTabs 2.0 |
| SFTTABS_2_1 | The application was developed for use with SftTabs/DLL 2.1 |

**Returns**

The return value is TRUE if the function was successful, otherwise FALSE is returned.

**Comments**

When developing new applications, always use SetVersion(SFTTABS_2_1) to be compatible with SftTabs/DLL as documented in this reference.

If SetVersion is not used, compatibility with version 2.0 is the default.

Most features that were introduced with version 2.1 are available even if SetVersion(SFTTABS_2_0) is used. SetVersion cannot be used to disable new features, its purpose is to make sure that certain API calls have the correct result. Certain features have been modified between 2.0 and 2.1 so the calls actually behave differently.

New behavior with SFTTABS_2_1:

- If the tab control is resized, pages attached to the tab control are resized even if they are controlled via a frame window (see SftTabs_ActivatePage, SetControlInfo, CSftTabsDialog::InitializeTabControl, TSftTabsDialog::InitializeTabControl, CSftTabsWindowSheet::InitializeTabControl or TSftTabsWindowSheet::InitializeTabControl for more information on frame windows).
- The SFTTABS_CONTROL structure members *fToolTips*, *fDropText* and *fCondScrollButtons* are honored.
- Ctrl+Tab and Ctrl+Shift+Tab select the next or previous tab.
- SftTabs_SetPageActive no longer removes the WS_TABSTOP style from an attached page.
- SftTabs_SetPageActive no longer copies a page's window caption to the enclosing window.

**Example**

This example sets version 2.1 compatibility:

**C**

```
SftTabs_SetVersion(hwndTab, SFTTABS_2_1);
```

**C++/MFC**

```
m_Tab.SetVersion(SFTTABS_2_1);
```

**C++/OWL**

```
pTab->SetVersion(SFTTABS_2_1);
```

# TSftTabs::TSftTabs

**C++, OWL only, TSftTabs::TSftTabs**

**Syntax 1:**

```
TSftTabs(TWindow* parent, int id, int x, int y, int w, int h, TModule* module = 0);
```

Standard constructor. Creates a tab control object.

**Parameters**

*parent*
Specifies the parent window.

*id*
Specifies the tab control's ID.

*x, y, w, h*
Specifies the tab control's position (x and y coordinates) and its size (width and height).

*module*
Specifies the application module.

**Example**

This example creates a tab control:

```
pTab = new TSftTabs(parentWindow, IDC_TAB, 250, 200, 400, 400);
pTab->Create();
```

**Syntax 2:**

```
TSftTabs(TWindow* parent, int resourceId, TModule* module = 0);
```

Standard constructor. Creates a tab control object based on an already existing window.

**Parameters**

*parent*
Specifies the parent window.

*resourceId*
Specifies the tab control's ID.

*module*
Specifies the application module.

**Comments**

If a tab control is part of a dialog, it is created from a Windows resource definition. This form of the TSftTabs constructor "connects" an existing control to the TSftTabs object.

# UnregisterApp

**C, SftTabs_UnregisterApp**

```
void WINAPI SftTabs_UnregisterApp(HINSTANCE hInst);
```

**C++, CSftTabs::UnregisterApp, TSftTabs::UnregisterApp**

```
static void UnregisterApp();
```

An application calls this function to unregister the application, once SftTabs/DLL controls are no longer used.

**Parameters**

*hInst*

The instance handle of the application.

**Comments**

This call allows SftTabs/DLL to unregister all window classes used and perform cleanup processing. This call has to be made after all SftTabs/DLL controls have been destroyed.

The call to this function should be made during application termination.

**Example**

This example unregisters an application from SftTabs/DLL:

**C**

```
int PASCAL WinMain(HINSTANCE hinst, HINSTANCE hinstPrev, LPSTR Cmd, int cmdShow)
{
    SftTabs_RegisterApp(hinst);         // Register application

    .... application message loop

    SftTabs_UnregisterApp(hinst);       // Unregister application
    return msg.wParam;
}
```

**C++/MFC**

```
int CSampleApp::ExitInstance()   // based on CWinApp
{
    // Unregister from SftTabs/DLL
    CSftTabs::UnregisterApp();
    // call base class
    return CWinApp::ExitInstance();
}
```

**C++/OWL**

```
int TSampleApp::TermInstance (int status)     // based on TApplicaton
{
    // Unregister from SftTabs/DLL
    TSftTabs::UnregisterApp();
    // call base class
    return TApplication::TermInstance (status);
}
```

# SftTabs_UnregisterDialog / -Window

```
BOOL WINAPI SftTabs_UnregisterDialog(HWND hwndDialog);
BOOL WINAPI SftTabs_UnregisterWindow(HWND hwndWnd);
```

An application can call this function to unregister a window, which has been previously registered using SftTabs_RegisterDialog or SftTabs_RegisterWindow.

**Parameters**

*hwndDialog, hwndWnd*

The window handle of the window or dialog to be unregistered.

**Returns**

The return value is TRUE if the window is successfully unregistered with SftTabs/DLL.

**Comments**

If this function is not called, resource leaks may be experienced.

**Example**

This C example shows a typical tabbed dialog WM_DESTROY message handler:

```
case WM_DESTROY: {
    // Unregister, or the window properties used won't be removed
    SftTabs_UnregisterDialog(hwndDlg);
    // destroy all pages
    SftTabs_Destroy(hwndDlg, GetDlgItem(hwndDlg, IDC_TAB));
    break;
}
```

# CSftTabsDialog::ClosePossible

```
virtual BOOL ClosePossible()
```

An application can call this function to determine if a tabbed dialog can be closed.

**Returns**

The return value is TRUE if the dialog can be closed, otherwise FALSE is returned.

**Comments**

This function calls the CSftTabsPage::AllowSwitch member function of the currently active tab page. If the currently active page can be closed, the entire tabbed dialog can also be closed. An application can override this function to perform additional tests, such as input validation, to determine if the dialog can be closed.

# CSftTabsDialog::CSftTabsDialog

```
    CSftTabsDialog(UINT IDD, CWnd* pParent = NULL);
```
**WIN16:**
```
    CSftTabsDialog(LPCSTR lpszTemplate, CWnd* pParent = NULL);
```
**WIN32:**
```
    CSftTabsDialog(LPCTSTR lpszTemplate, CWnd* pParent = NULL);
protected:
    CSftTabsDialog();
```

Standard constructor.

**Parameters**

*IDD*

ID of the dialog resource used to create the dialog.

*lpszTemplate*

A null-terminated string containing the name of the dialog resource used to create the dialog.

*pParent*

A pointer to the parent window's CWnd based object. This parameter may be NULL, if the tabbed dialog doesn't have a parent window.

**Comments**

A tabbed dialog is created in two steps. First call the constructor CSftTabsDialog, then use DoModal to create a modal dialog or call Create to create a modeless tabbed dialog. Override the OnInitDialog member function to initialize the tab control and associate CSftTabsPage objects to tabs.

**Example**

This example invokes a modal tabbed dialog:

```
CMainDlg MainDlg;         // tabbed dialog
MainDlg.DoModal();
```

# CSftTabsDialog::GetModified

```
virtual BOOL GetModified() const;
```

Used to retrieve the current data modification flag for the tabbed dialog.

**Returns**

The return value is TRUE if data has been modified, otherwise FALSE is returned.

**Comments**

The maintenance of the data modification flag is up to the application. When input data is altered, the tabbed dialog or page should use CSftTabsDialog::SetModified or CSftTabsPage::SetModified to mark data as modified. There is only one data modification flag for a tabbed dialog. When using CSftTabsPage::SetModified (a page), the tabbed dialogs modification flag is updated, so a subsequent CSftTabsPage::GetModified (by another page attached to the same tab control), will return the value of the tabbed dialog's modification flag.

An application could override the CSftTabsDialog::SetModified member function to visually notify the user that data has been modified. SetModified could be implemented to change the OK button's caption to "Save".

# CSftTabsDialog::InitializeTabControl

```
BOOL InitializeTabControl(int iTab, CSftTabs* pTabCtl, CWnd* pFrame = NULL);
```

Used to initialize a tab control in a tabbed dialog. Activates the specified tab and the associated page.

**Parameters**

*iTab*

The zero-based index of the tab to be made the active tab.

*pTabCtl*

A pointer to the tab control's CSftTabs based object.

*pFrame*

A pointer to a window's CWnd based object. This window will be used by SftTabs/DLL as client area for pages attached to the tab control. SftTabs/DLL uses this window's client area size and location as a replacement for the tab control's client area. The window described by *pFrame* may be hidden and/or disabled. If an application resizes or moves the frame window, the dependent page or Windows control also has to be resized by using the ResizePages function. Using this frame window, the client area of a tab control can be located anywhere in relation to the tab control, even on a different dialog or window. This parameter may be NULL, in which case the tab control's client area is used for attached pages.

**Returns**

The return value is TRUE if the function was successful, otherwise FALSE is returned.

**Comments**

A tabbed dialog's tab control has to be initialized, which creates the page attached to the currently active tab. This is typically done in the OnInitDialog member function of the tabbed dialog.

When a tabbed dialog is destroyed, all attached CSftTabsPage objects are automatically destroyed and deleted (using the C++ delete operator).

**Example**

This example initializes the tab control of a tabbed dialog and activates the second tab:

```
////////////////////////////////////////////////////////////////////////////
// CMainDlg message handlers

BOOL CMainDlg::OnInitDialog()
{
    // call base class
    CSftTabsDialog::OnInitDialog();

    int index;
    /* Associate the tab control created from the dialog       */
    /* resource with the C++ object.                           */
    m_Tab.SubclassDlgItem(IDC_TAB, this /* parent window */);

    /* Initialization is faster if we set redraw off */
    m_Tab.SetRedraw(FALSE);

    /* We are using new features */
    m_Tab.SetVersion(SFTTABS_2_1);

       ... additional tab initialization ...

    index = m_Tab.AddTab(_T("&Six"));
    m_Tab.SetTabInfo(index, &Tab5);
    // If you don't want to attach a page to the tab, the following is optional
//  m_Tab.SetTabDialog(index, new an_object_based_on_CSftTabsPage(this)); // tab page

    m_Tab.SetControlInfo(&CtlInit);

    // Make sure to turn redraw back on
```

```
    m_Tab.SetRedraw(TRUE);
    m_Tab.InvalidateRect(NULL, TRUE);

    // If you are not using the sheet/page classes, remove the ...
    // Initialize tab control
    InitializeTabControl(1, &m_Tab, NULL);

    return FALSE;
}
```

## CSftTabsDialog::OnCancel

```
virtual void OnCancel();
```

Called when the user hits the ESCAPE key or clicks the Cancel button (the button with an ID of IDCANCEL).

**Comments**

Override this member function to perform Cancel button action. The default implementation terminates a modal dialog box by calling EndDialog and causes DoModal to return IDCANCEL.

If you implement the Cancel button in a modeless tabbed dialog, you must override the OnCancel member function and call DestroyWindow. Don't call the base-class member function because it calls EndDialog, which does not destroy a modeless dialog.

# CSftTabsDialog::OnOK

```
virtual void OnOK();
```

Called when the user clicks the OK button (the button with an ID of IDOK).

**Comments**

Override this member function to perform the OK button action.

The default implementation of this member function calls CSftTabsDialog::ClosePossible to make sure that the currently active page can be closed. Then any automatic data validation and exchange for the tabbed dialog takes place.

If you implement the OK button in a modeless tabbed dialog, you must override the OnOK member function and call DestroyWindow from within it. Don't call the base-class member function because it calls EndDialog, which does not destroy a modeless dialog.

# CSftTabsDialog::SetClose

```
virtual void SetClose(BOOL fClose = TRUE);
```

Called to signal that data has been changed permanently and the tabbed dialog can no longer be Cancel'ed.

**Comments**

When input data is altered permanently, the tabbed dialog or page should use CSftTabsDialog::SetClose or CSftTabsPage::SetClose. An application could override the CSftTabsDialog::SetClose member function to visually notify the user that data has been permanently altered. SetClose could be implemented to change the Cancel button's caption to "Close".

# CSftTabsDialog::SetModified

```
virtual void SetModified(BOOL fModified = TRUE);
```

Used to set the current data modification flag for the tabbed dialog.

**Parameters**

*fModified*

The new value to be saved as the data modification flag. TRUE if data has been modified, FALSE otherwise.

**Comments**

The maintenance of the data modification flag is up to the application. When input data is altered, the tabbed dialog or page should use CSftTabsDialog::SetModified or CSftTabsPage::SetModified to mark data as modified. There is only one data modification flag for a tabbed dialog. When using CSftTabsPage::SetModified (a page), the tabbed dialogs modification flag is updated, so a subsequent CSftTabsPage::GetModified (by another page attached to the same tab control), will return the value of the tabbed dialog's modification flag.

An application could override the CSftTabsDialog::SetModified member function to visually notify the user that data has been modified. SetModified could be implemented to change the OK button's caption to "Save".

# CSftTabsPage::AllowDestroy

```
virtual BOOL AllowDestroy();
```

This member function is called to determine if a page should be destroyed when it is no longer visible, because the associated tab is no longer the currently active tab.

**Returns**

If TRUE is returned, the page will be destroyed, otherwise the page is disabled and hidden.

**Comments**

The default implementation of this member function returns FALSE, after performing automatic data validation and exchange for the tabbed dialog.

If FALSE is returned, a page and all its associated controls will not be destroyed, which can cause considerable Windows resources to be allocated to these pages, however, the data stored in these controls will remain intact and can be accessed until the entire tabbed dialog is finally destroyed. It is up to the developer to weigh the benefits of data persistance against additional resource usage. Switching between tabs is also faster if pages aren't destroyed immediately, because the pages don't have to be recreated from the dialog resources every time they become active.

**Example**

This example causes a page to destroyed when the page is no longer the active page:

```
BOOL CSubPg6::AllowDestroy()        // Allow window to be destroyed
{
    return TRUE;
}
```

# CSftTabsPage::AllowSwitch

```
virtual BOOL AllowSwitch();
```

This member function is called to determine if a currently active page can be left, and a new page be activated.

**Returns**

If TRUE is returned, the current page will be deactivated and another page will become active, otherwise the current page (and associated tab) cannot be changed.

**Comments**

The default implementation of this member function returns TRUE, after performing automatic data validation and exchange for the page.

An application can override this function to perform additional tests, such as input validation, to determine if the page can be left.

# CSftTabsPage::ClosePossible

```
virtual BOOL ClosePossible();
```

An application can call this function to determine if a page can be closed.

**Returns**

The return value is TRUE if the page can be closed, otherwise FALSE is returned.

**Comments**

An application can override this function to perform additional tests, such as input validation, to determine if the dialog can be closed.

The default implementation also tests any nested tab controls and pages. The main dialog has to be tested using CSftTabsDialog::ClosePossible.

# CSftTabsPage::CSftTabsPage

```
CSftTabsPage(UINT IDD, CWnd* pParent);
```

**WIN16:**

```
CSftTabsPage(LPCSTR lpszTemplate, CWnd* pParent);
```

**WIN32:**

```
CSftTabsPage(LPCTSTR lpszTemplate, CWnd* pParent);
```

Standard constructor.

**Parameters**

*IDD*

ID of the dialog resource used to create the dialog.

*lpszTemplate*

A null-terminated string containing the name of the dialog resource used to create the dialog.

*pParent*

A pointer to the parent window's CWnd based object. This parameter may be not be NULL. The parent window must be an object derived from CSftTabsPage or CSftTabsDialog.

**Comments**

A page attached to a tab control is created automatically by SftTabs/DLL in response to user input or under program control, by calls such as CSftTabsDialog::InitializeTabControl or SetCurrentTab. All pages created by SftTabs/DLL are created as modeless dialogs.

**Example**

This example creates several CSftTabsPage objects which are attached to the tab control:

```
BOOL CMainDlg::OnInitDialog()
{
    int index;
    SFTTABS_TAB Tab;
    /* Associate the tab control created from the dialog       */
    /* resource with the C++ object.                           */
    m_Tab.SubclassDlgItem(IDC_TAB, this /* parent window */);
    /* You could use DDX/DDV instead and add the following     */
    /* line to the DoDataExchange function of the tab          */
    /* control's parent window (remove the //).                */
    // DDX_Control(pDX, IDC_TAB, m_Tab);

    /* Initialization is faster if we set redraw off */
    m_Tab.SetRedraw(FALSE);

    /* We are using new features */
    m_Tab.SetVersion(SFTTABS_2_1);

    index = m_Tab.AddTab(_T("The First One"));
    m_Tab.SetToolTip(index, _T("Demonstrates tabbing into and out of the tab page"));
    Tab = Tab0;
    Tab.graph.item.hBitmap = (HBITMAP) m_SampleBitmap.m_hObject;
    m_Tab.SetTabInfo(index, &Tab);
    m_Tab.SetTabDialog(index, new CPage1(this)); /* tab page */

       ... additional tab initialization ...

    index = m_Tab.AddTab(_T("Si&xth"));
    m_Tab.SetToolTip(index, _T("A page with nested tab controls and pages"));
    m_Tab.SetTabInfo(index, &Tab5);
    m_Tab.SetTabDialog(index, new CPage6(this)); /* tab page */

    m_Tab.SetControlInfo(&CtlInit);

    // Make sure to turn redraw back on
    m_Tab.SetRedraw(TRUE);
    m_Tab.InvalidateRect(NULL, TRUE);
```

```
    // If you are not using the sheet/page classes, remove the ...
    // Initialize tab control
    InitializeTabControl(0, &m_Tab, NULL);
    return FALSE; // if this is a dialog's OnInitDialog member function
}
```

# CSftTabsPage::GetModified

```
virtual BOOL GetModified() const;
```

Used to retrieve the current data modification flag for the tabbed dialog.

**Returns**

The return value is TRUE if data has been modified, otherwise FALSE is returned.

**Comments**

The maintenance of the data modification flag is up to the application. When input data is altered, the tabbed dialog or page should use CSftTabsDialog::SetModified or CSftTabsPage::SetModified to mark data as modified. There is only one data modification flag for a tabbed dialog. When using CSftTabsPage::SetModified (a page), the tabbed dialogs modification flag is updated, so a subsequent CSftTabsPage::GetModified (by another page attached to the same tab control), will return the value of the tabbed dialog's modification flag.

An application could override the CSftTabsDialog::SetModified member function to visually notify the user that data has been modified. SetModified could be implemented to change the OK button's caption to "Save".

# CSftTabsPage::GetParentDialog

```
CSftTabsDialog* GetParentDialog() const;
```

Used to retrieve the page's parent dialog object.

**Returns**

The return value is a pointer to the CSftTabsDialog based object, which is the parent window of the page.

**Comments**

GetParentDialog retrieves the top-most enclosing CSftTabsDialog based object, in case of nested tab controls with attached pages.

**Example**

This example shows an OnOK member function of a CSftTabsPage based object. The page implements its own OK button. To process the OK button, it calls the parent dialog's OnOK member function.

```
void CPage4::OnOK()
{
    // Send OK to parent
    GetParentDialog()->OnOK();
}
```

# CSftTabsPage::InitializeTabControl

```
BOOL InitializeTabControl(int iTab, CSftTabs* pTabCtl, CWnd* pFrame = NULL);
```

Used to initialize a tab control in a page. Activates the specified tab and the associated page.

**Parameters**

*iTab*

The zero-based index of the tab to be made the active tab.

*pTabCtl*

A pointer to the tab control's CSftTabs based object.

*pFrame*

A pointer to a window's CWnd based object. This window will be used by SftTabs/DLL as client area for pages attached to the tab control. SftTabs/DLL uses this window's client area size and location as a replacement for the tab control's client area. The window described by *pFrame* may be hidden and/or disabled. If an application resizes or moves the frame window, the dependent page or Windows control also has to be resized by using the ResizePages function. Using this frame window, the client area of a tab control can be located anywhere in relation to the tab control, even on a different dialog or window. This parameter may be NULL, in which case the tab control's client area is used for attached pages.

**Returns**

The return value is TRUE if the function was successful, otherwise FALSE is returned.

**Comments**

A page's tab control has to be initialized, which creates the page attached to the currently active tab. This is typically done in the OnInitDialog member function of the page.

This function is only used for pages which contain tab controls. A main tabbed dialog would use CSftTabsDialog::InitializeTabControl instead.

When a tabbed dialog is destroyed, all attached CSftTabsPage objects are automatically destroyed and deleted (using the C++ delete operator).

**Example**

This example initializes the tab control of a page and activates the second tab:

```
//////////////////////////////////////////////////////////////////////////
// CPageDlg message handlers

BOOL CPageDlg::OnInitDialog()
{
    // call base class
    CSftTabsPage::OnInitDialog();

    int index;
    SFTTABS_TAB Tab;

    // Attach the tab control window to the CSftTabs object
    m_Tab1.SubclassDlgItem(IDC_P6_TAB1, this /* parent window */);

       ... additional tab initialization ...

    index = m_Tab.AddTab(_T("Si&xth"));
    m_Tab.SetTabInfo(index, &Tab5);
    m_Tab.SetTabDialog(index, new CPage6(this));/* tab page */
    m_Tab.SetControlInfo(&CtlInit);

    // Initialize tab control
    InitializeTabControl(1, &m_Tab, NULL);
    return FALSE;
}
```

# CSftTabsPage::OnCancel

```
virtual void OnCancel();
```

Called when the user clicks the Cancel button (the button with an ID of IDCANCEL).

**Comments**

The default implementation of this member function doesn't respond to the button. It is up to the application to override this function to do any processing. Usually, the Cancel button is located on the parent dialog, not on a page attached to a tab, so the CSftTabsDialog::OnCancel member function would process the Cancel button event.

**Example**

This example shows an OnCancel member function of a CSftTabsPage based object. The page implements its own Cancel button. To process the Cancel button, it calls the parent dialog's OnCancel member function.

```
void CPage4::OnCancel()
{
    // Send Cancel to parent
    GetParentDialog()->OnCancel();
}
```

# CSftTabsPage::OnOK

```
virtual void OnOK();
```

Called when the user clicks the OK button (the button with an ID of IDOK).

**Comments**

Override this member function to perform the OK button action.

The default implementation of this member function doesn't respond to the button. It is up to the application to override this function to do any processing. Usually, the OK button is located on the parent dialog, not on a page attached to a tab, so the CSftTabsDialog::OnOK member function would process the OK button event.

**Example**

This example shows an OnOK member function of a CSftTabsPage based object. The page implements its own OK button. To process the OK button, it calls the parent dialog's OnOK member function.

```
void CPage4::OnOK()
{
    // Send OK to parent
    GetParentDialog()->OnOK();
}
```

# CSftTabsPage::SetClose

```
virtual void SetClose(BOOL fClose = TRUE);
```

Called to signal that data has been changed permanently and the tabbed dialog can no longer be Cancel'ed.

**Comments**

When input data is altered permanently, the tabbed dialog or page should use CSftTabsDialog::SetClose or CSftTabsPage::SetClose. An application could override the CSftTabsDialog::SetClose member function to visually notify the user that data has been permanently altered. SetClose could be implemented to change the Cancel button's caption to "Close".

# CSftTabsPage::SetModified

```
virtual void SetModified(BOOL fModified = TRUE);
```

Used to set the current data modification flag for the tabbed dialog.

**Parameters**

*fModified*

The new value to be saved as the data modification flag. TRUE if data has been modified, FALSE otherwise.

**Comments**

The maintenance of the data modification flag is up to the application. When input data is altered, the tabbed dialog or page should use CSftTabsDialog::SetModified or CSftTabsPage::SetModified to mark data as modified. There is only one data modification flag for a tabbed dialog. When using CSftTabsPage::SetModified (a page), the tabbed dialogs modification flag is updated, so a subsequent CSftTabsPage::GetModified (by another page attached to the same tab control), will return the value of the tabbed dialog's modification flag.

An application could override the CSftTabsDialog::SetModified member function to visually notify the user that data has been modified. SetModified could be implemented to change the OK button's caption to "Save".

# CSftTabsWindowSheet::ClosePossible

```
virtual BOOL ClosePossible()
```

An application can call this function to determine if a tabbed window can be closed.

**Returns**

The return value is TRUE if the window can be closed, otherwise FALSE is returned.

**Comments**

This function calls the CSftTabsWindowPage::AllowSwitch member function of the currently active tab page. If the currently active page can be closed, the entire tabbed window can also be closed. An application can override this function to perform additional tests, such as input validation, to determine if the window can be closed.

# CSftTabsWindowSheet::CSftTabsWindowSheet

```
CSftTabsWindowSheet();
```

Standard constructor.

**Comments**

The class CSftTabsWindowSheet is never used by itself. It is used to add tabbed window support to a CWnd-based class.

To avoid problems usually found with MFC and Windows messaging when using multiple inheritance, the class CSftTabsWindowSheet must be defined as the "right-most" class.

**Example**

This example adds tabbed window support to the CSampleView class by using multiple inheritance:

```
class CSampleView : public CView, public CSftTabsWindowSheet
{
    ... class definition
};
```

# CSftTabsWindowSheet::InitializeTabControl

```
BOOL InitializeTabControl(CWnd* pWnd, int iTab, CSftTabs* pTabCtl,
                                                CWnd* pFrame = NULL);
```

Used to initialize a tab control in a tabbed window. Activates the specified tab and the associated page.

**Parameters**

*pWnd*

The CWnd based object describing the tab control's parent window (usually *this*).

*iTab*

The zero-based index of the tab to be made the active tab.

*pTabCtl*

A pointer to the tab control's CSftTabs based object.

*pFrame*

A pointer to a window's CWnd based object. This window will be used by SftTabs/DLL as client area for pages attached to the tab control. SftTabs/DLL uses this window's client area size and location as a replacement for the tab control's client area. The window described by *pFrame* may be hidden and/or disabled. If an application resizes or moves the frame window, the dependent page or Windows control also has to be resized by using the ResizePages function. Using this frame window, the client area of a tab control can be located anywhere in relation to the tab control, even on a different dialog or window. This parameter may be NULL, in which case the tab control's client area is used for attached pages.

**Returns**

The return value is TRUE if the function was successful, otherwise FALSE is returned.

**Comments**

A tabbed window's tab control has to be initialized, which creates the page attached to the currently active tab. This is typically done in the OnCreate member function of the tabbed window.

When a tabbed window is destroyed, all attached CSftTabsWindowPage objects are automatically destroyed. However, any dynamically allocated CSftTabsWindowPage derived objects must be deleted (using the C++ delete operator) by the application.

**Example**

This example initializes the tab control of a tabbed window and activates the first tab:

```
int CSampleView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CView::OnCreate(lpCreateStruct) == -1)
        return -1;

    // Create a static control that we can place above the tab control.
    // This is just used to cover the parent window in that area.
    if (!m_Gap.Create(_T(""), SS_SIMPLE | WS_VISIBLE | WS_CHILD,
                CRect(0, 0, 0, 0),           /* position */
                this))
        return -1;

#if !defined(TAB_CONTROL_WITH_CLIENTAREA)
    // Create a static control that we can use as a frame window for the tab control's
    // pages.  This window is not visible and is just used to indicate the page
position
    if (!m_Frame.Create(_T(""), SS_SIMPLE | WS_CHILD,
                CRect(0, 0, 0, 0),           /* position */
                this))
        return -1;
#endif

    // Create the tab control
```

```
    if (!m_Tab.Create(
            WS_VISIBLE | WS_CHILD |          /* Visible, child window */
                WS_CLIPCHILDREN | WS_TABSTOP |
                                             /* Clip child windows, tabstop */
                WS_GROUP,                    /* Group */
                CRect(0, 0, 0, 0),           /* position */
            this,                            /* Parent window */
            IDC_TAB))                        /* tab control ID */
        return -1;

    int index;

    /* Initialization is faster if we set redraw off */
    m_Tab.SetRedraw(FALSE);
    /* Create the font used for the tab control.               */
    /* Fonts are owned by the application and have to remain   */
    /* valid as long as the tab control uses the font.         */
    int height;                          /* Height in pixels */
    HDC hDC;                             /* Device context */

    /* Create the font to be used for the tab control.         */
    hDC = ::GetDC(NULL);                 /* Get a device context */
    height = MulDiv(12, ::GetDeviceCaps(hDC, LOGPIXELSY), 72);/* Convert ...
    m_Font.CreateFont(-height, 0, 0, 0, FW_NORMAL, 0, 0, 0, 0, 0, 0, 0, 0,
_T("Arial"));
    ::ReleaseDC(NULL, hDC);              /* Release device context */
    m_Tab.SetFont(&m_Font, FALSE);       /* Set tab control font */

    /* We are using new features */
    m_Tab.SetVersion(SFTTABS_2_1);

    index = m_Tab.AddTab(_T("&Listbox"));
    m_Tab.SetToolTip(index, _T("ToolTip for the ListBox tab"));
    m_Tab.SetTabInfo(index, &Tab0);
    m_Tab.SetTabWindowPage(index, &m_ListBox); /* tab page */

    index = m_Tab.AddTab(_T("&Edit Control"));
    m_Tab.SetToolTip(index, _T("ToolTip for the Edit Control tab"));
    m_Tab.SetTabInfo(index, &Tab1);
    m_Tab.SetTabWindowPage(index, &m_Edit); /* tab page */

    index = m_Tab.AddTab(_T("&Other Listbox"));
    m_Tab.SetToolTip(index, _T("ToolTip for the Other ListBox tab"));
    m_Tab.SetTabInfo(index, &Tab2);
    m_Tab.SetTabWindowPage(index, &m_OtherListBox); /* tab page */

    m_Tab.SetControlInfo(&CtlInit);

    // Make sure to turn redraw back on
    m_Tab.SetRedraw(TRUE);
    m_Tab.InvalidateRect(NULL, TRUE);

#if defined(TAB_CONTROL_WITH_CLIENTAREA)
    // Initialize tab control
    InitializeTabControl(this, 0, &m_Tab, NULL);
#else
    // Initialize tab control.  An invisible, disabled frame window is used to ...
    InitializeTabControl(this, 0, &m_Tab, &m_Frame);
#endif

    // Mark the view as a main, tabbed window (so accel. keys work) by registering it.
    SftTabs_RegisterWindow(m_hWnd);
    return 0;
}
```

# CSftTabsWindowSheet::TabSwitched

```
void TabSwitched(CWnd* pParent, CSftTabs* pTabCtl);
```

Called by an application to handle the SFTTABSN_SWITCHED notification.

**Parameters**

*pParent*

The CWnd based object describing the tab control's parent window (usually *this*).

*pTabCtl*

A pointer to the tab control's CSftTabs based object.

**Comments**

A tabbed window must call this function to process the SFTTABSN_SWITCHED notification that is generated by the tab control to switch between pages.

Message map entries must also be added to the tab control's parent window.

**Example**

This example implements the suggested OnTabSwitched function that calls TabSwitched to switch between pages:

```
// Add the following definitions to the tab control's parent window
// class CYourSheet

    afx_msg void OnTabSwitching();
    afx_msg void OnTabSwitched();

// Add the following to the parent window's message map

    ON_SFTTABSN_SWITCHING(IDC_TAB, OnTabSwitching)
    ON_SFTTABSN_SWITCHED(IDC_TAB, OnTabSwitched)

// Implement the following functions in CYourSheet

void CYourSheet::OnTabSwitching()
{
    TabSwitching(this, &m_Tab);
}

void CYourSheet::OnTabSwitched()
{
    TabSwitched(this, &m_Tab);
}
```

# CSftTabsWindowSheet::TabSwitching

```
void TabSwitching(CWnd* pParent, CSftTabs* pTabCtl);
```

Called by an application to handle the SFTTABSN_SWITCHING notification.

**Parameters**

*pParent*

The CWnd based object describing the parent window (usually *this*).

*pTabCtl*

A pointer to the tab control's CSftTabs based object.

**Comments**

A tabbed window must call this function to process the SFTTABSN_SWITCHING notification that is generated by the tab control to switch between pages.

TabSwitching calls the CSftTabsWindowPage::AllowSwitch function of the current page to determine if the next page can be activated. GetNextTab returns the index of the next tab about to become active. By sending a WM_CANCELMODE message, an application can prevent the tab control from activating the next page.

Message map entries must also be added to the tab control's parent window.

**Example**

This example implements the suggested OnTabSwitching function that calls TabSwitching to switch between pages:

```
// Add the following definitions to the tab control's parent window
// class CYourSheet

    afx_msg void OnTabSwitching();
    afx_msg void OnTabSwitched();

// Add the following to the parent window's message map

    ON_SFTTABSN_SWITCHING(IDC_TAB, OnTabSwitching)
    ON_SFTTABSN_SWITCHED(IDC_TAB, OnTabSwitched)

// Implement the following functions in CYourSheet

void CYourSheet::OnTabSwitching()
{
    TabSwitching(this, &m_Tab);
}

void CYourSheet::OnTabSwitched()
{
    TabSwitched(this, &m_Tab);
}
```

# CSftTabsWindowSheet::TerminateTabControl

```
void TerminateTabControl(CWnd* pWnd, CSftTabs* pTabCtl);
```

Terminates a tab control and deactivates all pages.

**Parameters**

*pWnd*

The CWnd based object describing the tab control's parent window (usually *this*).

*pTabCtl*

A pointer to the tab control's CSftTabs based object.

**Comments**

A tabbed window's tab control has to be terminated, which deactivates and destroys all pages attached to the tab control. This is typically done in the OnDestroy member function of the tabbed window.

When a tabbed window is destroyed, all attached CSftTabsWindowPage objects are automatically destroyed. However, any dynamically allocated CSftTabsWindowPage derived objects must be deleted (using the C++ delete operator) by the application.

**Example**

This example terminates the tab control of a tabbed window:

```
void CSampleView::OnDestroy()
{
    // Remove all pages from the tab control
    TerminateTabControl(this, &m_Tab);
    // Unregister, or the window properties used won't be removed
    SftTabs_UnregisterWindow(m_hWnd);

    CView::OnDestroy();
}
```

# CSftTabsWindowPage::ActivatePage

```
virtual BOOL ActivatePage(CWnd* pParent, CSftTabs* pTabCtl) = 0;
```

Called to create or activate a page.

**Parameters**

*pParent*

The CWnd based object describing the tab control's parent window.

*pTabCtl*

A pointer to the tab control's CSftTabs based object.

**Returns**

If TRUE is returned, the page was successfully created and activated, otherwise return FALSE.

**Comments**

The CSftTabsWindowSheet class implementation calls this member function to create the window associated with a page or to make the page visible.

Your CWnd based class must implement ActivatePage.

**Example**

This example shows the suggested implementation of the ActivatePage function:

```
BOOL CYourPage::ActivatePage(CWnd* pParent, CSftTabs* pTabCtl)
{
    // This is called when the user switches to a page
    if (!m_hWnd) {
        // The window doesn't exist, create it now. Make sure it's NOT VISIBLE
        // You can modify this to create another type of window instead.
        // The exact syntax of the Create function used depends on the base
        // class used.
        if (!Create(....                // Create the window
                WS_TABSTOP|             // Tabstop style is important
                other_styles,
                CRect(0,0,0,0),         // location
                pParent,                // Parent Window
                a_control_id))          // control ID
                // make sure the above control ID does not collide with
                // IDs used by other pages or by the tab control itself
            return FALSE;
        // Additional initialization if desired
    } else {
        // The user switched back to this page
    }

    // This page is now active
    SftTabs_SetPageActive(m_hWnd, pTabCtl->m_hWnd, NULL);
    // Enable + show it, its size is 0,0,0,0, it will be resized by the tab control
    EnableWindow(TRUE);
    ShowWindow(SW_SHOW);

    return TRUE;
}
```

# CSftTabsWindowPage::AllowSwitch

```
virtual BOOL AllowSwitch();
```

This member function is called to determine if a currently active page can be left, and a new page be activated.

**Returns**

If TRUE is returned, the current page will be deactivated and another page will become active, otherwise the current page (and associated tab) cannot be changed.

**Comments**

The default implementation of this member function returns TRUE.

An application can override this function to perform additional tests, such as input validation, to determine if the page can be left.

# CSftTabsWindowPage::CSftTabsWindowPage

```
CSftTabsWindowPage();
```

Standard constructor.

**Comments**

The class CSftTabsWindowPage is never used by itself. It is used to add the support necessary to a CWnd-based class as a page.

To avoid problems usually found with MFC and Windows messaging when using multiple inheritance, the class CSftTabsWindowSheet must be defined as the "right-most" class.

**Example**

This example adds tabbed window support to the CSampleView class by using multiple inheritance:

```
class CSampleListBox : public CListBox, public CSftTabsWindowPage
{
    ... class definition
};
```

# CSftTabsWindowPage::DeactivatePage

```
virtual void DeactivatePage(CWnd* pParent, CSftTabs* pTabCtl, BOOL fFinal) = 0;
```

Called to deactivate or destroy a page.

**Parameters**

*pParent*
The CWnd based object describing the tab control's parent window.

*pTabCtl*
A pointer to the tab control's CSftTabs based object.

*fFinal*
TRUE if the page must be destroyed, FALSE if the page can be hidden or destroyed.

**Comments**

The CSftTabsWindowSheet class implementation calls this member function to destroy the window associated with a page or to make the page invisible.

If the page is destroyed, the page must be recreated when the user switches back to this page (see CSftTabsWindowPage::ActivatePage). This does save resources but may cause excessive wait times. It is up to your application to chose the most suitable method.

Not all CWnd derived classes are suitable to be destroyed multiple times while using the same C++ object. Some classes (once constructed) assume that an attached window is only created once, not multiple times as it could happen with SftTabs/DLL. If a class doesn't support multiple creation of its window, you have to use ShowWindow when the user switches away from the tab page (as shown in the example below).

Your CWnd based class must implement DeactivatePage.

**Example**

This example shows the suggested implementation of the DeactivatePage function:

```
void CYourPage::DeactivatePage(CWnd* pParent, CSftTabs* pTabCtl, BOOL fFinal)
{
    if (fFinal) {
        // You must destroy the window, the tabbed window (parent) is going away
        DestroyWindow();
    } else {
        // Hide the page. If you want, you could use DestroyWindow here too.
        // In that case you save resources and the window will be recreated
        // when the user switches back to this page
        ShowWindow(SW_HIDE);
        EnableWindow(FALSE);
    }
    // clear associated page in tab's control structure
    SftTabs_SetPageInactive(pTabCtl->m_hWnd);
}
```

# TSftTabsDialog::CanClose

```
virtual bool CanClose();
```

This function is called to determine if the tabbed dialog can be closed.

**Returns**

The return value is TRUE if the dialog can be closed, otherwise FALSE is returned.

**Comments**

This function calls the CanClose member function of the currently active tab page. If the currently active page can be closed, the entire tabbed dialog can also be closed. An application can override this function to perform additional tests, such as input validation, to determine if the dialog can be closed.

# TSftTabsDialog::GetModified

```
virtual BOOL GetModified() const;
```

Used to retrieve the current data modification flag for the tabbed dialog.

**Returns**

The return value is TRUE if data has been modified, otherwise FALSE is returned.

**Comments**

The maintenance of the data modification flag is up to the application. When input data is altered, the tabbed dialog or page should use TSftTabsDialog::SetModified or TSftTabsPage::SetModified to mark data as modified. There is only one data modification flag for a tabbed dialog. When using TSftTabsPage::SetModified (a page), the tabbed dialogs modification flag is updated, so a subsequent TSftTabsPage::GetModified (by another page attached to the same tab control), will return the value of the tabbed dialog's modification flag.

An application could override the TSftTabsDialog::SetModified member function to visually notify the user that data has been modified. SetModified could be implemented to change the OK button's caption to "Save".

# TSftTabsDialog::InitializeTabControl

```
BOOL InitializeTabControl(int iTab, TSftTabs* pTabCtl, TWindow* pFrame = NULL);
```

Used to initialize a tab control in a tabbed dialog. Activates the specified tab and the associated page.

**Parameters**

*iTab*

The zero-based index of the tab to be made the active tab.

*pTabCtl*

A pointer to the tab control's TSftTabs based object.

*pFrame*

A pointer to a window's TWindow based object. This window will be used by SftTabs/DLL as client area for pages attached to the tab control. SftTabs/DLL uses this window's client area size and location as a replacement for the tab control's client area. The window described by *pFrame* may be hidden and/or disabled. If an application resizes or moves the frame window, the dependent page or Windows control also has to be resized using the ResizePages function. Using this frame window, the client area of a tab control can be located anywhere in relation to the tab control, even on a different dialog or window. This parameter may be NULL, in which case the tab control's client area is used for attached pages.

**Returns**

The return value is TRUE if the function was successful, otherwise FALSE is returned.

**Comments**

A tabbed dialog's tab control has to be initialized, which creates the page attached to the currently active tab. This is typically done in the EvInitDialog member function of the tabbed dialog.

When a tabbed dialog is destroyed, all attached TSftTabsPage objects are automatically destroyed and deleted (using the C++ delete operator).

**Example**

This example initializes the tab control of a tabbed dialog and activates the second tab:

```
bool TMainDialog::EvInitDialog (HWND hWndFocus)
{
    TSftTabsDialog::EvInitDialog(hWndFocus);

    int index;

    /* Initialization is faster if we set redraw off */
    pTab->SetRedraw(false);

    /* We are using new features */
    pTab->SetVersion(SFTTABS_2_1);

    index = pTab->AddTab(TEXT("&One"));
    pTab->SetTabInfo(index, &Tab0);
    // If you don't want to attach a page to the tab, the following is optional
    pTab->SetTabDialog(index, new TPage0(this));

    ... additional tab initialization here ...

    index = pTab->AddTab(TEXT("&Six"));
    pTab->SetTabInfo(index, &Tab5);
    // If you don't want to attach a page to the tab, the following is optional
    pTab->SetTabDialog(index, new TPage6(this));

    pTab->SetControlInfo(&CtlInit);

    // Make sure to turn redraw back on
    pTab->SetRedraw(true);
    pTab->Invalidate(true);
```

```
    // If you are not using the sheet/page classes, remove the ...
    // Initialize tab control
    InitializeTabControl(1, pTab, NULL);
    return false;
}
```

# TSftTabsDialog::SetClose

```
virtual void SetClose(BOOL fClose = TRUE);
```

Called to signal that data has been changed permanently and the tabbed dialog can no longer be Cancel'ed.

**Comments**

When input data is altered permanently, the tabbed dialog or page should use TSftTabsDialog::SetClose or TSftTabsPage::SetClose. An application could override the TSftTabsDialog::SetClose member function to visually notify the user that data has been permanently altered. SetClose could be implemented to change the Cancel button's caption to "Close".

# TSftTabsDialog::SetModified

```
virtual void SetModified(BOOL fModified = TRUE);
```

Used to set the current data modification flag for the tabbed dialog.

**Parameters**

*fModified*

The new value to be saved as the data modification flag. TRUE if data has been modified, FALSE otherwise.

**Comments**

The maintenance of the data modification flag is up to the application. When input data is altered, the tabbed dialog or page should use TSftTabsDialog::SetModified or TSftTabsPage::SetModified to mark data as modified. There is only one data modification flag for a tabbed dialog. When using TSftTabsPage::SetModified (a page), the tabbed dialogs modification flag is updated, so a subsequent TSftTabsPage::GetModified (by another page attached to the same tab control), will return the value of the tabbed dialog's modification flag.

An application could override the TSftTabsDialog::SetModified member function to visually notify the user that data has been modified. SetModified could be implemented to change the OK button's caption to "Save".

# TSftTabsDialog::TSftTabsDialog

```
TSftTabsDialog(TWindow* parent, TResId resId, TModule* module = 0);
```

Standard constructor.

#### Parameters

*pParent*

A pointer to the parent window's TWindow based object. This parameter may be NULL, if the tabbed dialog doesn't have a parent window.

*resid*

The description of the dialog resource used to create the dialog.

*module*

Specifies the application's module, where the dialog resource can be found.

#### Comments

A tabbed dialog is created by invoking the constructor TSftTabsDialog. Override the EvInitDialog member function to initialize the tab control and associate TSftTabsPage objects to tabs.

#### Example

This example invokes a modal tabbed dialog:

```
CMainDlg MainDlg;          // tabbed dialog
MainDlg.DoModal();
```

# TSftTabsPage::AllowDestroy

```
virtual BOOL AllowDestroy();
```

This member function is called to determine if a page should be destroyed when it is no longer visible, because the associated tab is no longer the currently active tab.

**Returns**

If TRUE is returned, the page will be destroyed, otherwise the page is disabled and hidden.

**Comments**

The default implementation of this member function returns FALSE.

If FALSE is returned, a page and all its associated controls will not be destroyed, which can cause considerable Windows resources to be allocated to these pages, however, the data stored in these controls will remain intact and can be accessed until the entire tabbed dialog is finally destroyed. It is up to the developer to weigh the benefits of data persistance against additional resource usage. Switching between tabs is also faster if pages aren't destroyed immediately, because the pages don't have to be recreated from the dialog resources every time they become active.

**Example**

This example causes a page to destroyed when the page is no longer the active page:

```
BOOL TSubPg6::AllowDestroy()        // Allow window to be destroyed
{
    return TRUE;
}
```

# TSftTabsPage::CanClose

```
virtual bool CanClose();
```

An application can call this function to determine if a page can be closed or made inactive.

**Returns**

The return value is TRUE if the page can be closed, otherwise FALSE is returned.

**Comments**

An application can override this function to perform additional tests, such as input validation, to determine if the dialog can be closed.

The default implementation also tests any nested tab controls and pages. The main dialog has to be tested using TSftTabsDialog::CanClose.

# TSftTabsPage::CloseWindow

```
virtual void CloseWindow(int retValue = IDCANCEL);
```

An application can call this function to close the entire tabbed dialog.

**Parameters**

*retValue*

The value passed to ::EndDialog if the tabbed dialog is a modal dialog.

**Comments**

An application can override this function to perform additional tests, such as input validation, to determine if the dialog can be closed.

The default implementation tests any nested tab controls and pages and the main tabbed dialog using CanClose. If CanClose returns TRUE, the tabbed dialog will be closed.

# TSftTabsPage::CmCancel

```
void CmCancel();
```

Called when the user clicks the Cancel button (the button with an ID of IDCANCEL).

**Comments**

The default implementation of this member function automatically forwards this notification to the parent tabbed dialog, by calling the parent dialog's CmCancel member function. Usually, the Cancel button is located on the parent dialog, not on a page attached to a tab, so the TSftTabsDialog::CmCancel member function would process the Cancel button event.

This function is only called if a page has a button control with an ID of IDCANCEL. This function will not be called for a button with the same ID which is located on the tabbed dialog (i.e., belongs to the TSftTabsDialog object).

# TSftTabsPage::CmOk

```
void CmOk();
```

Called when the user clicks the OK button (the button with an ID of IDOK).

**Comments**

The default implementation of this member function automatically forwards this notification to the parent tabbed dialog, by calling the parent dialog's CmOk member function  Usually, the OK button is located on the parent dialog, not on a page attached to a tab, so the TSftTabsDialog::CmOk member function would process the OK button event.

This function is only called if a page has a button control with an ID of IDOK.  This function will not be called for a button with the same ID which is located on the tabbed dialog (i.e., belongs to the TSftTabsDialog object).

# TSftTabsPage::GetModified

```
virtual BOOL GetModified() const;
```

Used to retrieve the current data modification flag for the tabbed dialog.

**Returns**

The return value is TRUE if data has been modified, otherwise FALSE is returned.

**Comments**

The maintenance of the data modification flag is up to the application. When input data is altered, the tabbed dialog or page should use TSftTabsDialog::SetModified or TSftTabsPage::SetModified to mark data as modified. There is only one data modification flag for a tabbed dialog. When using TSftTabsPage::SetModified (a page), the tabbed dialogs modification flag is updated, so a subsequent TSftTabsPage::GetModified (by another page attached to the same tab control), will return the value of the tabbed dialog's modification flag.

An application could override the TSftTabsDialog::SetModified member function to visually notify the user that data has been modified. SetModified could be implemented to change the OK button's caption to "Save".

# TSftTabsPage::GetParentDialog

```
TSftTabsDialog* GetParentDialog() const;
```

Used to retrieve the page's parent dialog object.

**Returns**

The return value is a pointer to the TSftTabsDialog based object, which is the parent window of the page.

**Comments**

GetParentDialog is used to retrieve the top-most enclosing TSftTabsDialog based object, in case of nested tab controls with attached pages.

**Example**

This example shows a member function of a TSftTabsPage based object, which responds to a button click. The page implements its own OK button. To process the OK button, it calls the parent dialog's CmOk member function.

```
void TPage4::OKClicked ()
{
    // Forward to enclosing dialog
    GetParentDialog()->CmOk;
}
```

# TSftTabsPage::InitializeTabControl

```
BOOL InitializeTabControl(int iTab, TSftTabs* pTabCtl, TWindow* pFrame);
```

Used to initialize a tab control in a page. Activates the specified tab and the associated page.

**Parameters**

*iTab*

The zero-based index of the tab to be made the active tab.

*pTabCtl*

A pointer to the tab control's TSftTabs based object.

*pFrame*

A pointer to a window's TWindow based object. This window will be used by SftTabs/DLL as client area for pages attached to the tab control. SftTabs/DLL uses this window's client area size and location as a replacement for the tab control's client area. The window described by *pFrame* may be hidden and/or disabled. If an application resizes or moves the frame window, the dependent page or Windows control also has to be resized using the ResizePages function. Using this frame window, the client area of a tab control can be located anywhere in relation to the tab control, even on a different dialog or window. This parameter may be NULL, in which case the tab control's client area is used for attached pages.

**Returns**

The return value is TRUE if the function was successful, otherwise FALSE is returned.

**Comments**

A page's tab control has to be initialized, which creates the page attached to the currently active tab. This is typically done in the EvInitDialog member function of the page.

This function is only used for pages which contain tab controls. A main tabbed dialog would use TSftTabsDialog::InitializeTabControl instead.

When a tabbed dialog is destroyed, all attached TSftTabsPage objects are automatically destroyed and deleted (using the C++ delete operator).

**Example**

This example initializes the tab control of a page and activates the second tab:

```
bool TMainPage::EvInitDialog (HWND hWndFocus)
{
    TSftTabsPage::EvInitDialog(hWndFocus);

    int index;

    /* Initialization is faster if we set redraw off */
    pTab->SetRedraw(false);

    /* We are using new features */
    pTab->SetVersion(SFTTABS_2_1);

    index = pTab->AddTab(TEXT("&One"));
    pTab->SetTabInfo(index, &Tab0);
    // If you don't want to attach a page to the tab, the following is optional
    pTab->SetTabDialog(index, new TSubPage0(this));

    ... additional tab initialization here ...

    index = pTab->AddTab(TEXT("&Six"));
    pTab->SetTabInfo(index, &Tab5);
    // If you don't want to attach a page to the tab, the following is optional
    pTab->SetTabDialog(index, new TSubPage6(this));

    pTab->SetControlInfo(&CtlInit);

    // Make sure to turn redraw back on
    pTab->SetRedraw(true);
```

```
    pTab->Invalidate(true);

    // If you are not using the sheet/page classes, remove the ...
    // Initialize tab control
    InitializeTabControl(1, pTab, NULL);
    return false;
}
```

# TSftTabsPage::SetClose

```
virtual void SetClose(BOOL fClose = TRUE);
```

Called to signal that data has been changed permanently and the tabbed dialog can no longer be Cancel'ed.

**Comments**

When input data is altered permanently, the tabbed dialog or page should use TSftTabsDialog::SetClose or TSftTabsPage::SetClose. An application could override the TSftTabsDialog::SetClose member function to visually notify the user that data has been permanently altered. SetClose could be implemented to change the Cancel button's caption to "Close".

# TSftTabsPage::SetModified

```
virtual void SetModified(BOOL fModified = TRUE);
```

Used to set the current data modification flag for the tabbed dialog.

**Parameters**

*fModified*

The new value to be saved as the data modification flag. TRUE if data has been modified, FALSE otherwise.

**Comments**

The maintenance of the data modification flag is up to the application. When input data is altered, the tabbed dialog or page should use TSftTabsDialog::SetModified or TSftTabsPage::SetModified to mark data as modified. There is only one data modification flag for a tabbed dialog. When using TSftTabsPage::SetModified (a page), the tabbed dialogs modification flag is updated, so a subsequent TSftTabsPage::GetModified (by another page attached to the same tab control), will return the value of the tabbed dialog's modification flag.

An application could override the TSftTabsDialog::SetModified member function to visually notify the user that data has been modified. SetModified could be implemented to change the OK button's caption to "Save".

# TSftTabsPage::TSftTabsPage

```
TSftTabsPage(TWindow* pParent, TResId resId, TModule* module = 0);
```

Standard constructor.

**Parameters**

*pParent*

A pointer to the parent window's TWindow based object. This parameter may be not be NULL. The parent window must be an object derived from TSftTabsPage or TSftTabsDialog.

*resid*

The description of the dialog resource used to create the dialog.

*module*

Specifies the application's module, where the dialog resource can be found.

**Comments**

A page attached to a tab control is created automatically by SftTabs/DLL in response to user input or under program control, by calls such as TSftTabsDialog::InitializeTabControl or SetCurrentTab. All pages created by SftTabs/DLL are created as modeless dialogs.

**Example**

This example creates several TSftTabsPage objects which are attached to the tab control:

```
bool TMainDialog::EvInitDialog (HWND hWndFocus)
{
    TSftTabsDialog::EvInitDialog(hWndFocus);

    int index;

    /* Initialization is faster if we set redraw off */
    pTab->SetRedraw(false);

    /* We are using new features */
    pTab->SetVersion(SFTTABS_2_1);

    index = pTab->AddTab(TEXT("&One"));
    pTab->SetTabInfo(index, &Tab0);
    // If you don't want to attach a page to the tab, the following is optional
    pTab->SetTabDialog(index, new TPage0(this));

    ... additional tab initialization here ...

    index = pTab->AddTab(TEXT("&Six"));
    pTab->SetTabInfo(index, &Tab5);
    // If you don't want to attach a page to the tab, the following is optional
    pTab->SetTabDialog(index, new TPage6(this));

    pTab->SetControlInfo(&CtlInit);

    // Make sure to turn redraw back on
    pTab->SetRedraw(true);
    pTab->Invalidate(true);

    // If you are not using the sheet/page classes, remove the ...
    // Initialize tab control
    InitializeTabControl(1, pTab, NULL);
    return false;
}
```

# TSftTabsWindowSheet::CanClose

```
virtual bool CanClose();
```

This function is called to determine if the tabbed window can be closed.

**Returns**

The return value is TRUE if the window can be closed, otherwise FALSE is returned.

**Comments**

This function calls the TSftTabsWindowPage::AllowSwitch member function of the currently active tab page. If the currently active page can be closed, the entire tabbed window can also be closed. An application can override this function to perform additional tests, such as input validation, to determine if the window can be closed.

# TSftTabsWindowSheet::InitializeTabControl

```
BOOL InitializeTabControl(TWindow* pWnd, int iTab, TSftTabs* pTabCtl,
                                                   TWindow* pFrame = NULL);
```

Used to initialize a tab control in a tabbed window. Activates the specified tab and the associated page.

**Parameters**

*pWnd*

The TWindow based object describing the tab control's parent window (usually *this*).

*iTab*

The zero-based index of the tab to be made the active tab.

*pTabCtl*

A pointer to the tab control's TSftTabs based object.

*pFrame*

A pointer to a window's TWindow based object. This window will be used by SftTabs/DLL as client area for pages attached to the tab control. SftTabs/DLL uses this window's client area size and location as a replacement for the tab control's client area. The window described by *pFrame* may be hidden and/or disabled. If an application resizes or moves the frame window, the dependent page or Windows control also has to be resized by using the ResizePages function. Using this frame window, the client area of a tab control can be located anywhere in relation to the tab control, even on a different dialog or window. This parameter may be NULL, in which case the tab control's client area is used for attached pages.

**Returns**

The return value is TRUE if the function was successful, otherwise FALSE is returned.

**Comments**

A tabbed window's tab control has to be initialized, which creates the page attached to the currently active tab. This is typically done in the EvCreate member function of the tabbed window.

When a tabbed window is destroyed, all attached TSftTabsWindowPage objects are automatically destroyed. However, any dynamically allocated TSftTabsWindowPage derived objects must be deleted (using the C++ delete operator) by the application.

**Example**

This example initializes the tab control of a tabbed window and activates the first tab:

```
int TMainWin::EvCreate (CREATESTRUCT far& createStruct)
{
    if (TLayoutWindow::EvCreate(createStruct) != 0)
        return -1;

    // Create the tab control
    pTab = new TSftTabs(this,            // 'this' is the parent window
        IDC_TAB,                         // tab control ID
        0, 0,                            /* x, y */
        0, 0);                           /* width, height */
    pTab->Attr.Style |= WS_CLIPCHILDREN | WS_TABSTOP | WS_GROUP |
        WS_VISIBLE | WS_CHILD;           // Visible, child window
    if (!pTab->Create())
        return -1;

#if !defined(TAB_CONTROL_WITH_CLIENTAREA)
    // Create the frame window (which will hold the pages)
    m_pFrame = new TStatic(this, -1, TEXT(""), 0, 0, 0, 0);
    m_pFrame->Attr.Style &= ~WS_VISIBLE;     // not visible
    m_pFrame->Attr.Style |= WS_DISABLED;     //  and disabled
    // Create a static control that we can use as a frame window for the tab control's
    // pages.  This window is not visible and is just used to indicate the page
```

```
position
    if (!m_pFrame->Create())
        return -1;
#endif

    int index;

    /* Initialization is faster if we set redraw off */
    pTab->SetRedraw(false);

    /* We are using new features */
    pTab->SetVersion(SFTTABS_2_1);

    index = pTab->AddTab(TEXT("&Listbox"));
    pTab->SetToolTip(index, TEXT("A standard listbox is attached to this tab"));
    pTab->SetTabInfo(index, &Tab0);
    // If you don't want to attach a page to the tab, the following is optional
    pTab->SetTabWindowPage(index, m_pList); // tab page

    index = pTab->AddTab(TEXT("&Edit Control"));
    pTab->SetToolTip(index, TEXT("A standard edit control is attached to this tab"));
    pTab->SetTabInfo(index, &Tab1);
    pTab->SetTabWindowPage(index, m_pEdit); // tab page

    index = pTab->AddTab(TEXT("&Dialog"));
    pTab->SetToolTip(index, TEXT("A dialog is attached to this tab"));
    pTab->SetTabInfo(index, &Tab2);
    pTab->SetTabWindowPage(index, m_pDlg); // tab page

    pTab->SetControlInfo(&CtlInit);

    // Make sure to turn redraw back on
    pTab->SetRedraw(true);
    pTab->Invalidate(true);

    // If you are not using the sheet/page classes, remove the call to ...
#if defined(TAB_CONTROL_WITH_CLIENTAREA)
    // Initialize tab control
    InitializeTabControl(this, 0, pTab, NULL);
#else
    // Initialize tab control.  An invisible, disabled frame window is used to ...
    InitializeTabControl(this, 0, pTab, m_pFrame);
#endif
    // Mark the view as a main, tabbed window (so accel. keys work) by registering it.
    SftTabs_RegisterWindow(HWindow);

    return 0;
}
```

# TSftTabsWindowSheet::TabSwitched

```
void TabSwitched(TWindow* pParent, TSftTabs* pTabCtl);
```

Called by an application to handle the SFTTABSN_SWITCHED notification.

**Parameters**

*pParent*

The TWindow based object describing the tab control's parent window (usually *this*).

*pTabCtl*

A pointer to the tab control's TSftTabs based object.

**Comments**

A tabbed window must call this function to process the SFTTABSN_SWITCHED notification that is generated by the tab control to switch between pages.

Response table entries must also be added to the tab control's parent window.

**Example**

This example implements the suggested EvTabSwitched function that calls TabSwitched to switch between pages:

```
// Add the following definitions to the tab control's parent window
// class TYourSheet

    void EvTabSwitching();
    void EvTabSwitched();

// Add the following to the parent window's response table
    EV_SFTTABSN_SWITCHING(IDC_TAB, EvTabSwitching),
    EV_SFTTABSN_SWITCHED(IDC_TAB, EvTabSwitched),

// Implement the following functions in TYourSheet

void TYourSheet::EvTabSwitching()
{
    TabSwitching(this, pTab);
}

void TYourSheet::EvTabSwitched()
{
    TabSwitched(this, pTab);
}
```

# TSftTabsWindowSheet::TabSwitching

```
void TabSwitching(TWindow* pParent, TSftTabs* pTabCtl);
```

Called by an application to handle the SFTTABSN_SWITCHING notification.

**Parameters**

*pParent*

The TWindow based object describing the parent window (usually *this*).

*pTabCtl*

A pointer to the tab control's TSftTabs based object.

**Comments**

A tabbed window must call this function to process the SFTTABSN_SWITCHING notification that is generated by the tab control to switch between pages.

TabSwitching calls the TSftTabsWindowPage::AllowSwitch function of the current page to determine if the next page can be activated. GetNextTab returns the index of the next tab about to become active. By sending a WM_CANCELMODE message, an application can prevent the tab control from activating the next page.

Message map entries must also be added to the tab control's parent window.

**Example**

This example implements the suggested EvTabSwitching function that calls TabSwitching to switch between pages:

```
// Add the following definitions to the tab control's parent window
// class TYourSheet

    void EvTabSwitching();
    void EvTabSwitched();

// Add the following to the parent window's response table
    EV_SFTTABSN_SWITCHING(IDC_TAB, EvTabSwitching),
    EV_SFTTABSN_SWITCHED(IDC_TAB, EvTabSwitched),

// Implement the following functions in TYourSheet

void TYourSheet::EvTabSwitching()
{
    TabSwitching(this, pTab);
}

void TYourSheet::EvTabSwitched()
{
    TabSwitched(this, pTab);
}
```

# TSftTabsWindowSheet::TerminateTabControl

```
void TerminateTabControl(TWindow* pWnd, TSftTabs* pTabCtl);
```

Terminates a tab control and deactivates all pages.

**Parameters**

*pWnd*

The TWindow based object describing the tab control's parent window (usually *this*).

*pTabCtl*

A pointer to the tab control's TSftTabs based object.

**Comments**

A tabbed window's tab control has to be terminated, which deactivates and destroys all pages attached to the tab control. This is typically done in the EvDestroy member function of the tabbed window.

When a tabbed window is destroyed, all attached TSftTabsWindowPage objects are automatically destroyed. However, any dynamically allocated TSftTabsWindowPage derived objects must be deleted (using the C++ delete operator) by the application.

**Example**

This example terminates the tab control of a tabbed window:

```
void TMainWin::EvDestroy ()
{
    TLayoutWindow::EvDestroy();

    // Remove all pages from the tab control
    TerminateTabControl(this, pTab);
    // Unregister, or the window properties used won't be removed
    SftTabs_UnregisterWindow(HWindow);
}
```

# TSftTabsWindowSheet::TSftTabsWindowSheet

```
TSftTabsWindowSheet();
```

Standard constructor.

**Comments**

The class TSftTabsWindowSheet is never used by itself. It is used to add tabbed window support to a TWindow-based class.

**Example**

This example adds tabbed window support to the TMainWin class by using multiple inheritance:

```
class TMainWin : public TLayoutWindow, public TSftTabsWindowSheet {
    ... class definition
};
```

# TSftTabsWindowPage::ActivatePage

```
virtual bool ActivatePage(TWindow* pParent, TSftTabs* pTabCtl) = 0;
```

Called to create or activate a page.

**Parameters**

*pParent*

The TWindow based object describing the tab control's parent window.

*pTabCtl*

A pointer to the tab control's TSftTabs based object.

**Returns**

If true is returned, the page was successfully created and activated, otherwise return false.

**Comments**

The TSftTabsWindowSheet class implementation calls this member function to create the window associated with a page or to make the page visible.

Your TWindow based class must implement ActivatePage.

**Example**

This example shows the suggested implementation of the ActivatePage function:

```
bool TYourPage::ActivatePage(TWindow* pParent, TSftTabs* pTabCtl)
{
    // This is called when the user switches to a page
    if (!HWindow) {
        // The window doesn't exist, create it now. Make sure it's NOT VISIBLE.
        // You can modify this to create another type of window instead.
        Attr.Style &= ~(WS_BORDER|WS_VISIBLE); // turn these off
        // you may need to add/remove additional styles
        Attr.Style |= WS_TABSTOP; // turn these styles on
        if (!Create())
            return false;
        // Additional initialization if desired
    } else {
        // The user switched back to this page
    }

    // This page is now active
    SftTabs_SetPageActive(HWindow, pTabCtl->HWindow, NULL);
    // Enable + show it, its size is 0,0,0,0, it will be resized by the tab control
    EnableWindow(true);
    ShowWindow(SW_SHOW);

    return true;
}
```

# TSftTabsWindowPage::AllowSwitch

```
virtual bool AllowSwitch();
```

This member function is called to determine if a currently active page can be left, and a new page be activated.

**Returns**

If true is returned, the current page will be deactivated and another page will become active, otherwise the current page (and associated tab) cannot be changed.

**Comments**

The default implementation of this member function returns true.

An application can override this function to perform additional tests, such as a call to TWindow::CanClose to determine if the page can be left.

# TSftTabsWindowPage::DeactivatePage

```
virtual void DeactivatePage(TWindow* pParent, TSftTabs* pTabCtl, bool fFinal) = 0;
```

Called to deactivate or destroy a page.

**Parameters**

*pParent*
The TWindow based object describing the tab control's parent window.

*pTabCtl*
A pointer to the tab control's TSftTabs based object.

*fFinal*
true if the page must be destroyed, false if the page can be hidden or destroyed.

**Comments**

The TSftTabsWindowSheet class implementation calls this member function to destroy the window associated with a page or to make the page invisible.

If the page is destroyed, the page must be recreated when the user switches back to this page (see TSftTabsWindowPage::ActivatePage). This does save resources but may cause excessive wait times. It is up to your application to chose the most suitable method.

Not all TWindow derived classes are suitable to be destroyed multiple times while using the same C++ object. Some classes (once constructed) assume that an attached window is only created once, not multiple times as it could happen with SftTabs/DLL. If a class doesn't support multiple creation of its window, you have to use ShowWindow when the user switches away from the tab page (as shown in the example below).

Your TWindow based class must implement DeactivatePage.

**Example**

This example shows the suggested implementation of the DeactivatePage function:

```
void TYourPage::DeactivatePage(TWindow* pParent, TSftTabs* pTabCtl, bool fFinal)
{
    if (fFinal) {
        // You must destroy the window, the tabbed window (parent) is going away
        Destroy();
    } else {
        // Hide the page. If you want, you could use Destroy here too.
        // In that case you save resources and the window will be recreated
        // when the user switches back to this page
        ShowWindow(SW_HIDE);
        EnableWindow(false);
    }
    // clear associated page in tab's control structure
    SftTabs_SetPageInactive(pTabCtl->HWindow);
}
```

# TSftTabsWindowPage::TSftTabsWindowPage

```
TSftTabsWindowPage();
```

Standard constructor.

**Comments**

The class TSftTabsWindowPage is never used by itself. It is used to add the support necessary to a TWindow-based class as a page.

OWL usually creates child windows automatically once a parent window is created. This would however interfere with the TSftTabsWindowSheet and TSftTabsWindowPage implementations. Make sure the control/window isn't automatically created when the parent is created by using the DisableAutoCreate function (see OWL reference for more information).

```
pPage->DisableAutoCreate();
```

**Example**

This example adds tabbed window support to the TSampleListBox class by using multiple inheritance:

```
class TSampleListBox : public TListBox, public TSftTabsWindowPage {
    ... class definition
};
```