Contents - SftColor Version 1.0

Introduction

Using Resource Editors

Programming

Reference

MFC

OWL

Support

This is an incorrectly licensed version of SftColor. 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.

SftColor Overview

SftColor is a custom control for the Windows operating system, offering an easy method for a user to make a color selection based on a predefined color-set or to define a custom color.

Color Control

SftColor offers easy to use color selection.

- Displays predefined sets of colors
- Quick, intuitive interface for color selection
- Optional edit button for custom colors
- Built-in default color list
- User-definable color list
- Support for default entry
- Easy to use API, superset of combo box
- Acts like a combo box
- Great for color selection on tool bars
- Several styles
- Built-in ChooseColor dialog access
- Support for SDK dialog editors, Visual C++, Resource Workshop, Borland C++
- Supports C, C++ with MFC and C++ with OWL

Source Code

The source code for the MFC and OWL C++ classes for color control access is included. The DLL source code (written in C) is available. Any application that you develop can use SftColor royalty-free as long as none of our source code is shipped with your application.

Languages Supported

SftColor supports C, C++ and other languages when using the standard SendMessage Windows API. The SftColor DLL 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, SftColor is shipped with class definitions which support the Microsoft Foundation Class Library (MFC) and the Borland ObjectWindows Library (OWL).

Environments Supported

SftColor supports Windows 3.1 (including Win32s), Windows NT and Windows 95 using the same easy to use API.

Licensing

SftColor is shipped under a single developer license, which allows one single developer to install the product and use all files included. The SftColor DLLs can be redistributed with an application royalty-free. If more than one developer needs access to the SftColor DLLs and/or any of the development files, such as header files (*.h), source files (*.c and *.cpp), etc., additional licenses have to be purchased. SftColor is also offered under a site license agreement. If you have any questions regarding licensing of our products, please contact Softel vdm.

Using SftColor

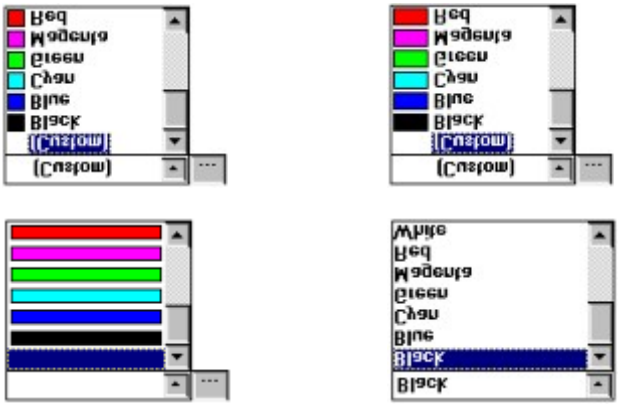
Depending on the programming language used, the steps necessary to add a color control to an application differ somewhat, but the following steps outline the basic method:

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

Once the color control has been created, the API functions documented in the topics Definitions and Structures and Messages and Functions can be used to add colors, define attributes, respond to events, etc.

Color Control Styles

The SftColor color control builds on the Windows combo box. The color control is always shown as a drop-down list style combo box. This means that the user cannot type the color name, a selection has to be made from the colors offered. Most styles offer support for an optional edit button. This edit button can be enabled under program control or can automatically invoke the ChooseColor Common Dialog.



The color control has support for a special color name *(Custom)*, which allows the user to invoke the ChooseColor Common Dialog to define a custom color.

Keyboard Interface

The color control implements the same interface as a standard combo box. If the color combo box has the input focus, the right arrow key will cause the input focus to shift to the edit button (if present). If the edit button has the input focus the left arrow key will cause the input focus to shift to the color combo box.

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 .....

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 November 3, 1995. Subject to change.

.....	SftColor for Windows, Windows NT and Windows 95 Single developer license, without DLL source code $59 each copy	$ .....
.....	SftColor for Windows, Windows NT and Windows 95 Single developer license, including DLL source code $99 each copy	$ .....
.....	Upgrade, SftColor DLL Source Code Only Single developer license $40 each copy Enter License # _____	$ .....
	6% Sales Tax (NJ residents only)	$ .....
	Shipping and Handling ($5.00 per copy, $7.00 Canada, $12.00 international)	$ .....
	Total	$ .....

Product Support

If you experience difficulties using SftColor, there are several methods to contact us, so we can help you resolve the problem. 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
Fax	(201) 366-3984
WWW	http://www.softelvdm.com Download up-to-date bug descriptions, solutions, samples
Internet	support@softelvdm.com
CompuServe	CIS 72724,2321
Mail	Softel vdm 11 Michigan Ave 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 the installation diskette label.

Rebuilding the DLLs

For information on how to link your application to the SftColor DLL, please see the programming sections.

Note: If you need to modify the SftColor DLL source code, please make sure to test the resulting DLL with the sample applications.

If you wish to rebuild the SftColor 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 SftColor DLL can be found in the directory C:\SFTCOLOR\SOURCE (unless changed during the installation).

Please note that you can only rebuild the DLLs if you have purchased the SftColor source code.

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	SFTCLR.DLL	SFTCL32**U**.DLL	SFTCL32.DLL
Required Source Files	BCT1CLR.C HELPER.C MCT1CLR.C SFTCLR.C CLRINIT.C SFTCLR.RC SFTCLR.DEF	 HELPER.C MCT2CLR.C SFTCLR.C CLRINIT.C SFTCLR.RC SFTCL32**U**.DEF	 HELPER.C MCT2CLR.C SFTCLR.C CLRINIT.C SFTCLR.RC SFTCL32.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 SftColor DLL. For certain error conditions, the SftColor 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 SftColor 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.

The project has to linked with COMMDLG.LIB and the COMMDLG.DLL has to be available at run-time.

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 SftColor (for 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 SFTCL32U.DLL, the non-UNICODE DLL is named SFTCL32.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	SFTCLR.LIB	SFTCL32V.LIB	SFTCL32B.LIB

C Programming

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

Building an Application

Every source program making use of a SftColor control must include the required header file SFTCOLOR.H by using the #include directive.

```
#include "sftclr.h"                /* SftColor required header file */
```

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

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

```
SftColor_RegisterApp(hInstance);    /* Use SftColor with this application */
```

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

```
SftColor_UnregisterApp(hInstance);  /* No longer use SftColor */
```

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 SftColor 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	SFTCLR.LIB	SFTCLR.LIB	SFTCLR.DLL
WIN32/s/c, all 32-bit environments incl. Windows NT (DLL w/o UNICODE support)	SFTCL32.LIB	SFTCL32B.LIB	SFTCL32.DLL
Windows NT (DLL with UNICODE support)	SFTCL32U.LIB	SFTCL32V.LIB	SFTCL32U.DLL

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

Adding a Color Control

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

- using dialog resources
- using CreateWindow(Ex)

Adding a color control using dialog resources is accomplished by using a resource editor to design a dialog. Once a color control is created, its window handle can be obtained by using the Windows GetDlgItem function.

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

```
hwndColor = CreateWindow(SFTCOLOR_CLASS, "",       // Window class and caption
                         style,  x, y, cx, cy,     // location
                         hwndParent,               // parent window
```

```
                        IDC_COLOR,              // color control ID
                        hInst,                  // application instance
                        NULL);
```

For more information on the various parameters used, see the Windows API documentation. The color control class is defined by the SFTCOLOR_CLASS constant (SFTCLR.H). The window class is **SoftelColor** (Windows 3.1) or **SoftelColor32** (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.

Handling Combo Box Notifications

The color control generates the same notifications as a drop-down list style combo box. The following sample code illustrates how a selection change notification (CBN_SELCHANGE) could be handled:

WIN16 (Windows 3.1, WFW, etc.):

```
switch (msg) {
case WM_COMMAND: {
    HWND hwndCtl = (HWND) LOWORD(lParam);
    int id = (int) wParam;
    int code = HIWORD(lParam);
    if (hwndCtl) {
        switch (id) {
        case IDC_COLOR:
            switch (code) {
            case CBN_SELCHANGE:    // just like a real combo box
                // implement your handler here
                break;
            }
            break;
        }
    }
    break;
 }
```

Win32 (Windows 95, Windows NT, Win32s):

```
switch (msg) {
case WM_COMMAND: {
    HWND hwndCtl = (HWND) lParam;
    int id = LOWORD(wParam);
    int code = HIWORD(wParam);
    if (hwndCtl) {
        switch (id) {
        case IDC_COLOR:
            switch (code) {
            case CBN_SELCHANGE:    // just like a real combo box
                // implement your handler here
                break;
            }
            break;
        }
    }
    break;
 }
```

Handling the Edit Button

If a color control is defined with the SFTCOLORSTYLE_CUSTOMEDIT style, it will generate WM_COMMAND, SFTCOLORN_EDIT notification messages when the user clicks the edit button. The following sample code illustrates how the notification could be handled:

WIN16 (Windows 3.1, WFW, etc.):

```
switch (msg) {
case WM_COMMAND: {
    HWND hwndCtl = (HWND) LOWORD(lParam);
    int id = (int) wParam;
    int code = HIWORD(lParam);
    if (hwndCtl) {
        switch (id) {
        case IDC_COLOR:
            switch (code) {
            case SFTCOLORN_EDIT:    // The user clicked the edit button
                // you could bring up your own ChooseColor dialog
                break;
            }
            break;
        }
    }
    break;
 }
```

Win32 (Windows 95, Windows NT, Win32s):

```
switch (msg) {
case WM_COMMAND: {
    HWND hwndCtl = (HWND) lParam;
    int id = LOWORD(wParam);
    int code = HIWORD(wParam);
    if (hwndCtl) {
        switch (id) {
        case IDC_COLOR:
            switch (code) {
            case SFTCOLORN_EDIT:    // The user clicked the edit button
                // you could bring up your own ChooseColor dialog
                break;
            }
            break;
        }
    }
    break;
 }
```

C++/MFC Programming

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

Building an Application

Every source program making use of a SftColor control must include the required header file SFTCLR.H by using the #include directive.

```
#include "sftclr.h"                    /* SftColor required header file */
```

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

One source program must include the CSftColor class implementation, using the #include directive.

```
#include "sftclrm.cpp"                 /* SftColor implementation */
```

This include statement should appear after the #include "sftclr.h" statement. This is the preferred method to include the implementation of the CSftColor class. Adding the file SFTCLRM.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:\SFTCOLOR\INCLUDE (unless changed during the installation).

In order to use SftColor controls, an application must call the CSftColor::RegisterApp function. The call to this function is required so that SftColor window classes can be registered. This call has to be made before any SftColor 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:

```
CSftColor::RegisterApp();          /* Use SftColor with this application */
```

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

```
CSftColor::UnregisterApp();        /* No longer use SftColor */
```

The application's executable (EXE or DLL) must be linked with the correct LIB file, depending on the target environment. The SftColor 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	SFTCLR.LIB	SFTCLR.DLL
WIN32/s/c, all 32-bit environments incl. Windows NT (DLL w/o UNICODE support)	SFTCL32.LIB	SFTCL32.DLL
Windows NT (DLL with UNICODE support)	SFTCL32U.LIB	SFTCL32U.DLL

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

Adding a Color Control

ClassWizard does not support new classes such as CSftColor, so any color 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 color control to an application:

- using dialog resources
- using CSftColor::Create

Adding a color 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 SftColor, see Resource Workshop, AppStudio, Visual C++, Dialog Editor (Windows SDK) and Borland C++. Once a color control is created, its CSftColor based object can be obtained by using the Windows GetDlgItem function or attached to a CSftColor object using SubclassWindow.

```
CSftColor * pClrControl;
pClrControl = (CSftColor *) GetDlgItem(IDC_COLOR);
```

```
CSftColor m_Clr;
m_Clr.SubclassWindow(::GetDlgItem(m_hWnd, IDC_COLOR));
CSftColor m_Color;
m_Color.SubclassDlgItem(IDC_COLOR, this);
```

Another method to create a color control is by using the CSftColor::Create member function.

```
CSftColor m_Color;
m_Color.Create(WS_CHILD|WS_VISIBLE|
      SFTCOLORSTYLE_STYLE1 | SFTCOLORSTYLE_VSCROLL |
      SFTCOLORSTYLE_SELTEXTONLY | SFTCOLORSTYLE_CUSTOMEDIT |
      SFTCOLORSTYLE_CUSTOMEDITAUTO,
      CRect(250, 200, 500, 240),
      pParentWnd,
      IDC_COLOR);
```

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 CSftColor, so any color 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.

Handling Combo Box Notifications

The color control generates the same notifications as a drop-down list style combo box. The following sample code illustrates how a selection change notification (CBN_SELCHANGE) could be handled:

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

// Event handler(s) added to message map
BEGIN_MESSAGE_MAP(CYourDialog, CDialog)
    ON_CBN_SELCHANGE(IDC_COLOR, OnSelChange)
END_MESSAGE_MAP()

// event handler implementation
void CYourDialog::OnSelChange()
{
    // your event handler
}
```

Handling the Edit Button

If a color control is defined with the SFTCOLORSTYLE_CUSTOMEDIT style, it will generate WM_COMMAND, SFTCOLORN_EDIT Notifications messages when the user clicks the edit button. The following sample code illustrates how the notification could be handled:

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

// Event handler(s) added to message map
BEGIN_MESSAGE_MAP(CYourDialog, CDialog)
```

```
    ON_SFTCOLORN_EDIT(IDC_COLOR, OnEditButtonClick)
END_MESSAGE_MAP()

// event handler implementation
void CYourDialog::OnEditButtonClick()
{
    // you could implement your own ChooseColor dialog
}
```

C++/OWL Programming

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

Building an Application

Every source program making use of a SftColor control must include the required header file SFTCLR.H by using the #include directive.

```
#include "sftclr.h"                    /* SftColor required header file */
```

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

One source program must include the TSftColor class implementation, using the #include directive.

```
#include "sftclrb.cpp"                 /* SftColor implementation */
```

This include statement should appear after the #include "sftclr.h" statement. This is the preferred method to include the implementation of the TSftColor class. Adding the file SFTCLRB.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:\SFTCOLOR\INCLUDE (unless changed during the installation).

In order to use SftColor controls, an application must call the TSftColor::RegisterApp function. The call to this function is required so that SftColor window classes can be registered. This call has to be made before any SftColor 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:

```
TSftColor::RegisterApp();          /* Use SftColor with this application */
```

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

```
TSftColor::UnregisterApp();        /* No longer use SftColor */
```

The application's executable (EXE or DLL) must be linked with the correct LIB file, depending on the target environment. The SftColor 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	SFTCLR.LIB	SFTCLR.DLL
WIN32/s/c, all 32-bit environments incl. Windows NT (DLL w/o UNICODE support)	SFTCL32B.LIB	SFTCL32.DLL
Windows NT (DLL with UNICODE support)	SFTCL32V.LIB	SFTCL32U.DLL

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

Adding a Color Control

ClassExpert does not support new classes such as TSftColor, so any color 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 color control to an application:

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

Adding a color 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 SftColor, see Resource Workshop, AppStudio, Visual C++, Dialog Editor (Windows SDK) and Borland C++. Once a color control is created by creating the dialog, the TSftColor based object can be constructed by using the TSftColor constructor.

```
pColor = new TSftColor(this, IDC_COLOR);
```

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

```
pColor = new TSftColor(parentWindow, IDC_COLOR, 250,200,500,240);
pColor->Attr.Style |= WS_CHILD|WS_VISIBLE|
      SFTCOLORSTYLE_STYLE1 | SFTCOLORSTYLE_VSCROLL |
      SFTCOLORSTYLE_SELTEXTONLY | SFTCOLORSTYLE_CUSTOMEDIT |
      SFTCOLORSTYLE_CUSTOMEDITAUTO;
pColor->Create();
```

The constructor creates the color control object. The arguments define the position of the color control window once it is created using the Create function. For more information on the various parameters used, see section topics Definitions and Structures and Messages and Functions.

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 TSftColor, so any color 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.

Handling Combo Box Notifications

The color control generates the same notifications as a drop-down list style combo box. The following sample code illustrates how a selection change notification (CBN_SELCHANGE) could be handled:

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

// Response table
DEFINE_RESPONSE_TABLE1(TMainDlg, TDialog)
    EV_CBN_SELCHANGE(IDC_COLOR, EvSelChange),
END_RESPONSE_TABLE;

// Event handler implementation
void TMainDlg::EvSelChange()
{
    // your event handler
}
```

Handling the Edit Button

If a color control is defined with the SFTCOLORSTYLE_CUSTOMEDIT style, it will generate WM_COMMAND, SFTCOLORN_EDIT notification messages when the user clicks the edit button. The following sample code illustrates how the notification could be handled:

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

// Response table
DEFINE_RESPONSE_TABLE1(TMainDlg, TDialog)
    EV_SFTCOLORN_EDIT(IDC_COLOR, EvEditButtonClick),
END_RESPONSE_TABLE;
```

```
// Event handler implementation
void TMainDlg::EvEditButtonClick()
{
    // you could implement your own ChooseColor dialog
}
```

Resource Workshop

First Time

In order to make SftColor available to Resource Workshop, use its *File, Install control library...* menu command to define SftColor to Resource Workshop. This has to be done only once. Locate the SftColor DLL using the dialog shown. The files SFTCLR.DLL and SFTCL32.DLL can be found in the directory C:\SFTCOLOR\BIN (unless changed during the installation). The 16-bit version of Borland C++ Resource Workshop can use the 16-bit SFTCLR.DLL, later 32-bit versions of Borland's IDE may also support the 32-bit DLL SFTCL32.DLL. The resource script generated can be compiled using 32-bit (or 16-bit) tools and linked with the appropriate 32-bit or 16-bit version of SftColor.

New Project

Whenever you create a project which is to include a SftColor control, make sure to add the C and C++ header file SFTCLR.H to your project. This file can be found in the directory C:\SFTCOLOR\INCLUDE (unless changed during the installation). Use the *File, Add to project...* menu command to display the *Add file to project* dialog. Adding the SftColor header file insures that your resource definitions for the SftColor control can be compiled correctly. The SFTCLR.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 SftColor control.

Adding a Color Control to a Dialog

To add a SftColor control to a dialog, use the SftColor toolbar button. Click on the button and then on the dialog being designed to add a control. The height of the color control is dynamically determined at run-time, based on the font used. The height of the control as defined in the dialog resource is ignored. Once a SftColor 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.

SftColor Control Styles Dialog

The *SftColor Styles* dialog allows you to manipulate the following color 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_COLOR=201. In both cases the value 201 is assigned to the control as control ID, the second example also defines IDC_COLOR 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.
Vertical scroll bar	The *Vertical scroll bar* check box determines whether the control displays a vertical scroll bar, if scrolling is possible. Equivalent to the SFTCOLORSTYLE_VSCROLL style.
Disable no-scroll	The *Disable no-scroll* check box determines whether the vertical scroll bar is disabled or hidden if scrolling is not possible. If the option is not selected, the vertical scroll bar will be hidden when scrolling is not possible. Equivalent to the SFTCOLORSTYLE_DISABLENOSCROLL style.
Highlight text only	The *Highlight text only* check box determines whether an item's text portion only will be highlighted when the item is the selected item. Otherwise the entire item (including color sample) will be highlighted. Under Windows 3.1 and Windows NT, this option is ignored, items will always be highlighted in their entirety (including color sample). This option applies to Windows 95 only. Equivalent to the SFTCOLORSTYLE_SELTEXTONLY style.
Sort items by color name	The *Sort items by color name* check box determines whether items are sorted. If this option is selected, items added to the color combo box are automatically sorted by name, otherwise their order is not changed. Equivalent

to the SFTCOLORSTYLE_SORT style.

Solid colors only The *Solid colors only* check box determines whether solid colors only are displayed. If this option is selected, the color samples will always be displayed using the (closest) solid color (not dithered colors) and the ChooseColor dialog invoked using the edit button will not allow custom colors. Equivalent to the SFTCOLORSTYLE_SOLID style.

Edit button The *Edit button* check box determines whether the color edit button is available. If this option is selected, the edit button is displayed. The edit button is only enabled once the user selects *(Custom)* as the current color, unless *Always enabled* is selected also (see below). Equivalent to the SFTCOLORSTYLE_CUSTOMEDIT style.

Use built-in ChooseColor dialog The *Use built-in ChooseColor dialog* check box determines whether the color edit button automatically invokes the ChooseColor dialog when clicked. If this option is selected, the ChooseColor dialog is automatically displayed when the user clicks the color edit button. Once the user makes a selection in the dialog, a SFTCOLORN_EDIT WM_COMMAND notification is sent to the color control's parent window. If this option is not selected, a SFTCOLORN_EDIT WM_COMMAND notification is sent to the color control's parent window when the edit button is clicked. Equivalent to the SFTCOLORSTYLE_CUSTOMEDITAUTO style.

Always enabled The *Always enabled* check box determines whether the color edit button is always enabled. If this option is selected, the edit button is always enabled and can be disabled by the application. Otherwise the edit button is only enabled if the *(Custom)* color is selected. Equivalent to the SFTCOLORSTYLE_EDITALWAYS 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 the tab key to access this control. Equivalent to the WS_TABSTOP style.

OK Click the *OK* button to accept all style settings and end the *SftColor Styles* dialog.

Cancel Click the *Cancel* button to abandon all (modified) style settings and end the *SftColor Styles* dialog.

Help Click the *Help* button for on-line help information on the *SftColor Styles* dialog.

Test Mode

In the dialog test mode offered by Resource Workshop, a SftColor control will be displayed in the location specified with the attributes defined using the *SftColor Styles* dialog.

Borland C++

Borland C++ does not support custom control DLLs (Resource Workshop, shipped with Borland C++ 4.x fully supports custom controls). It is still possible to use SftColor with Borland C++, but the easy design-time 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 SftColor control, make sure to include the C and C++ header file SFTCLR.H. This insures that your resource definitions for the SftColor control can be compiled correctly. Add the following statement to your resource script:

```
#include sftclr.h                    // SftColor header file (for style bits)
```

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

Adding a Color Control to a Dialog

To add a SftColor control to a dialog, use the *Dialog, Insert New Control* menu command. Enter the class **SoftelColor** (Windows 3.1) or **SoftelColor32** (for Windows NT, 95, Win32s) in the *New Control* dialog shown.

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.

SftColor Control Styles

To enter a SftColor window style in the *Control Properties* dialog, use the following list to add the desired style values and enter the resulting hexadecimal value in the field marked *Style*.

Style

Value	Description
SFTCOLORSTYLE_CUSTOMEDIT	
0x00001000	Add a color edit button to the color control. The edit button is only enabled once the user selects *(Custom)* as the current color, unless SFTCOLORSTYLE_EDITALWAYS is also used (see below).
SFTCOLORSTYLE_CUSTOMEDITAUTO	
0x00002000	The color edit button automatically invokes the ChooseColor dialog when clicked. Once the user makes a selection in the dialog, a SFTCOLORN_EDIT WM_COMMAND notification is sent to the color controls parent window. If this style is not given, a SFTCOLORN_EDIT WM_COMMAND notification is sent to the color controls parent window when the edit button is clicked. This style is ignored if SFTCOLORSTYLE_CUSTOMEDIT is not specified.
SFTCOLORSTYLE_DISABLENOSCROLL	
0x00008000	Disable the scroll bar if scrolling is not possible. If this style is used, SFTCOLORSTYLE_VSCROLL must also be used.
SFTCOLORSTYLE_EDITALWAYS	
0x00004000	The edit button is always enabled and can be disabled by the application. This style is ignored if SFTCOLORSTYLE_CUSTOMEDIT is not specified.
SFTCOLORSTYLE_SELTEXTONLY	
0x00000100	Highlight only the text portion (not the color sample) of a selected item. This style is ignored under Window 3.1 and Windows NT and only applies to Windows 95.
SFTCOLORSTYLE_SOLID	
0x00000400	Display the color samples using the (closest) solid color (not dithered colors) and do not allow custom colors to be defined using the ChooseColor dialog invoked using the edit button.
SFTCOLORSTYLE_SORT	
0x00000200	Sort items by color name.
SFTCOLORSTYLE_STYLE1	

0x00000000	A color control with a color sample and color name. Cannot be used in conjunction with any other SFTCOLORSTYLE_STYLE... value.
SFTCOLORSTYLE_STYLE2	
0x00000001	A color control with a wide color sample and color name. Cannot be used in conjunction with any other SFTCOLORSTYLE_STYLE... value.
SFTCOLORSTYLE_STYLE3	
0x00000002	A color control with a color sample only. Cannot be used in conjunction with any other SFTCOLORSTYLE_STYLE... value.
SFTCOLORSTYLE_STYLE4	
0x00000003	A color control with color name only. Cannot be used in conjunction with any other SFTCOLORSTYLE_STYLE... value.
SFTCOLORSTYLE_VSCROLL	
0x00000080	Add a vertical scroll bar to the color control.
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 color control that is initially disabled. A disabled color 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 color control that is initially visible.
WS_VSCROLL	
0x00200000	Add a vertical scroll bar to the color control.

ExStyle

Value	**Description**
WS_EX_CLIENTEDGE	
0x00000200	Specifies that a window has a border with a sunken edge (Windows 95, NT only).

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. It is still possible to use SftColor 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 SftColor control, make sure to include the C and C++ header file SFTCLR.H. This insures that your resource definitions for the SftColor control can be compiled correctly. Add the following statement to your resource script:

```
#include "sftclr.h"                    // SftColor header file (for style bits)
```

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

Adding a Color control to a Dialog

To add a SftColor control to a dialog, use the custom control toolbar button. Click on the button and then the dialog being designed to add a control. The height of the color control is dynamically determined at run-time, based on the font used. The height of the control as defined in the dialog resource is ignored.

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 SftColor control, enter the class **SoftelColor** (Windows 3.1) or **SoftelColor32** (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.

SftColor Control Styles

To enter a SftColor 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 Color control Styles.

Note: After entering the style value, make sure to select the desired *Visible*, *Group*, *Disabled* and *Tabstop* options again. These are also controlled by the style value entered, but are not listed here, because it is easier to maintain these options using the check boxes.

Style/Value Description

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

SFTCOLORSTYLE_CUSTOMEDITAUTO / 0x00002000
: The color edit button automatically invokes the ChooseColor dialog when clicked. Once the user makes a selection in the dialog, a SFTCOLORN_EDIT WM_COMMAND notification is sent to the color control's parent window. If this style is not given, a SFTCOLORN_EDIT WM_COMMAND notification is sent to the color control's parent window when the edit button is clicked. This style is ignored if SFTCOLORSTYLE_CUSTOMEDIT is not specified.

SFTCOLORSTYLE_DISABLENOSCROLL / 0x00008000
: Disable the scroll bar if scrolling is not possible. If this style is used, SFTCOLORSTYLE_VSCROLL must also be used.

SFTCOLORSTYLE_EDITALWAYS / 0x00004000
: The edit button is always enabled and can be disabled by the application. This style is ignored if SFTCOLORSTYLE_CUSTOMEDIT is not specified.

SFTCOLORSTYLE_CUSTOMEDIT / 0x00001000
: Add a color edit button to the color control. The edit button is only enabled once the user selects *(Custom)* as the current color, unless SFTCOLORSTYLE_EDITALWAYS is also used (see below).

SFTCOLORSTYLE_SELTEXTONLY / 0x00000100
: Highlight only the text portion (not the color sample) of a selected item. This style is ignored under Window 3.1 and Windows NT and only applies to Windows 95.

SFTCOLORSTYLE_SOLID / 0x00000400
: Display the color samples using the (closest) solid color (not dithered colors) and do not

allow custom colors to be defined using the ChooseColor dialog invoked using the edit button.

SFTCOLORSTYLE_SORT / 0x00000200
: Sort items by color name.

SFTCOLORSTYLE_STYLE1 / 0x00000000
: A color control with a color sample and color name. Cannot be used in conjunction with any other SFTCOLORSTYLE_STYLE... value.

SFTCOLORSTYLE_STYLE2 / 0x00000001
: A color control with a wide color sample and color name. Cannot be used in conjunction with any other SFTCOLORSTYLE_STYLE... value.

SFTCOLORSTYLE_STYLE3 / 0x00000002
: A color control with a color sample only. Cannot be used in conjunction with any other SFTCOLORSTYLE_STYLE... value.

SFTCOLORSTYLE_STYLE4 / 0x00000003
: A color control with color name only. Cannot be used in conjunction with any other SFTCOLORSTYLE_STYLE... value.

SFTCOLORSTYLE_VSCROLL / 0x00000080 Add a vertical scroll bar to the color control.

Test Mode

In the dialog test mode offered by Visual C++, the SftColor 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 SftColor 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 SftColor available to the dialog editor, use its *File, Open Custom...* menu command to define SftColor to the dialog editor. This has to be done only once.

Locate the SftColor DLL using the dialog shown. The DLL can be found in the directory C:\SFTCOLOR\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	SFTCLR.DLL
Windows NT (DLL w/o UNICODE support)	SFTCL32.DLL
Windows NT (DLL with UNICODE support)	SFTCL32U.DLL

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

New Project

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

```
#include "sftclr.h"                    // SftColor header file (for style bits)
```

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

Adding a Color control to a Dialog

To add a SftColor 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 SftColor 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.

When designing a dialog using a 16-bit dialog editor, the generated resource script will always use the color control class **SoftelColor**. If your application is a 32-bit application for Windows NT or Window 95, make sure to change the color control class to **SoftelColor32**. This has to be done manually by editing the resource script with a text editor. If the class is not changed, the dialog cannot be created by Windows at run-time.

SftColor Control Styles

The *SftColor Styles* dialog allows you to manipulate the following color control attributes.

Item	Description
Vertical scroll bar	The *Vertical scroll bar* check box determines whether the control displays a vertical scroll bar, if scrolling is possible. Equivalent to the SFTCOLORSTYLE_VSCROLL style.
Disable no-scroll	The *Disable no-scroll* check box determines whether the vertical scroll bar is disabled or hidden if scrolling is not possible. If the option is not selected, the vertical scroll bar will be hidden when scrolling is not possible. Equivalent to the SFTCOLORSTYLE_DISABLENOSCROLL style.
Highlight text only	The *Highlight text only* check box determines whether an item's text portion only will be highlighted when the item is the selected item. Otherwise the entire item (including color sample) will be highlighted. Under Windows 3.1 and Windows NT, this option is ignored, items will always be highlighted in their entirety (including color sample). This option applies to Windows 95 only. Equivalent to the

	SFTCOLORSTYLE_SELTEXTONLY style.
Sort items by color name	The *Sort items by color name* check box determines whether items are sorted. If this option is selected, items added to the color combo box are automatically sorted by name, otherwise their order is not changed. Equivalent to the SFTCOLORSTYLE_SORT style.
Solid colors only	The *Solid colors only* check box determines whether solid colors only are displayed. If this option is selected, the color samples will always be displayed using the (closest) solid color (not dithered colors) and the ChooseColor dialog invoked using the edit button will not allow custom colors. Equivalent to the SFTCOLORSTYLE_SOLID style.
Edit button	The *Edit button* check box determines whether the color edit button is available. If this option is selected, the edit button is displayed. The edit button is only enabled once the user selects *(Custom)* as the current color, unless Always enabled is selected also (see below). Equivalent to the SFTCOLORSTYLE_CUSTOMEDIT style.
Use built-in ChooseColor dialog	The *Use built-in ChooseColor dialog* check box determines whether the color edit button automatically invokes the ChooseColor dialog when clicked. If this option is selected, the ChooseColor dialog is automatically displayed when the user clicks the color edit button. Once the user makes a selection in the dialog, a SFTCOLORN_EDIT WM_COMMAND notification is sent to the color control's parent window. If this option is not selected, a SFTCOLORN_EDIT WM_COMMAND notification is sent to the color control's parent window when the user clicks the edit button. Equivalent to the SFTCOLORSTYLE_CUSTOMEDITAUTO style.
Always enabled	The *Always enabled* check box determines whether the color edit button is always enabled. If this option is selected, the edit button is always enabled and can be disabled by the application. Otherwise the edit button is only enabled if the *(Custom)* color is selected. Equivalent to the SFTCOLORSTYLE_EDITALWAYS 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 *SftColor Styles* dialog.
Cancel	Click the *Cancel* button to abandon all (modified) style settings and end the *SftColor Styles* dialog.
Help	Click the *Help* button for on-line help information on the *SftColor Styles* dialog.

Test Mode

In the dialog test mode offered by the dialog editor, a SftColor control will be displayed in the location specified with the attributes defined using the *SftColor Styles* dialog.

Color Control Styles

The following color control window styles are available in addition to the standard window styles (such as WS_BORDER, WS_TABSTOP, etc.).

SFTCOLORSTYLE_STYLE1 (0x0000L)

SFTCOLORSTYLE_STYLE2 (0x0001L)

SFTCOLORSTYLE_STYLE3 (0x0002L)

SFTCOLORSTYLE_STYLE4 (0x0003L)

The color control takes on a different appearance based on which style value is used. STYLE1 displays a color sample and text, STYLE2 displays a wide color sample and text, STYLE3 displays a color sample only and STYLE4 displays text only. Only one style value can be used. These values cannot be combined.

SFTCOLORSTYLE_VSCROLL (0x0080L)

When this style is selected, a vertical scroll bar is added to the color control.

SFTCOLORSTYLE_SELTEXTONLY (0x0100L)

When this style is selected, only the text portion (not the color sample) of a selected item is highlighted. This style is ignored under Window 3.1 and Windows NT and only applies to Windows 95.

SFTCOLORSTYLE_SORT (0x0200L)

When this style is selected, items are automatically sorted by color name as they are added to the color control.

SFTCOLORSTYLE_SOLID (0x0400L)

When this style is selected, the color samples are displayed using the (closest) solid color (not dithered colors) and the ChooseColor dialog invoked using the edit button does not allow custom colors to be defined.

SFTCOLORSTYLE_CUSTOMEDIT (0x1000L)

When this style is selected, an edit button is added to the color control. The edit button is only enabled once the user selects *(Custom)* as the current color, unless SFTCOLORSTYLE_EDITALWAYS is also used.

SFTCOLORSTYLE_CUSTOMEDITAUTO (0x2000L)

When this style is selected, the edit button automatically invokes the ChooseColor dialog when clicked. Once the user makes a selection in the dialog, a SFTCOLORN_EDIT WM_COMMAND notification is sent to the color control's parent window. If this style is not given, a SFTCOLORN_EDIT WM_COMMAND notification is sent to the color control's parent window when the edit button is clicked. This style is ignored if SFTCOLORSTYLE_CUSTOMEDIT is not specified.

SFTCOLORSTYLE_EDITALWAYS (0x4000L)

When this style is selected, the edit button is always enabled and can be disabled by the application. This style is ignored if SFTCOLORSTYLE_CUSTOMEDIT is not specified.

SFTCOLORSTYLE_DISABLENOSCROLL (0x8000L)

When this style is selected, the scroll bar is disabled instead of hidden, if scrolling is not possible. This style is ignored if SFTCOLORSTYLE_VSCROLL is not specified.

Notifications

The parent window of a color 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
SFTCOLORN_EDIT	The user has clicked the color edit button. If the color control has the SFTCOLORSTYLE_CUSTOMEDITAUTO style, this notification is sent after the user chooses a color using the ChooseColor dialog. Otherwise the notification is sent when the user clicks the edit button and the application can respond to the event.
SFTCOLORN_KILLFOCUS	The color control lost the input focus.
SFTCOLORN_SETFOCUS	The color control received the input focus.

The SftColor control also generates the following events. These are equivalent to the notifications sent by a regular combo box. Please see the Windows API, MFC or OWL help files for more information.

NotifyCode	Description
CBN_CLOSEUP	The drop-down list of the combo box has been closed.
CBN_DROPDOWN	The drop-down list of the combo box dropped down.
CBN_ERRSPACE	The color control is out of memory.
CBN_SELCHANGE	The user's selection is changing.
CBN_SELENDCANCEL	The user's selection is canceled.
CBN_SELENDOK	The user's selection is valid.

MFC and Notifications

If you want to handle Windows notification messages sent by a color 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. *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 OnSelChange();

// Event handler(s) added to message map
BEGIN_MESSAGE_MAP(CSampleView, CView)
    ON_CBN_SELCHANGE(IDC_COLOR, OnSelChange)
END_MESSAGE_MAP()

// Event handler implementation
void CSampleView::OnSelChange()
{
    // The selection is changing
}
```

OWL and Notifications

If you want to handle Windows notification messages sent by a color 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. *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 EvSelChange();

// Response table
DEFINE_RESPONSE_TABLE1(TMainDlg, TDialog)
    EV_CBN_SELCHANGE(IDC_COLOR, EvSelChange),
END_RESPONSE_TABLE;

// Event handler implementation
void TMainDlg::EvSelChange()
{
    // The selection is changing
}
```

MFC/C++ SftColor Classes

CSftColor Class, Member Functions

CSftColor is derived from CWnd.

~CSftColor	Destructor.
Create	Creates a color control.
CSftColor	Constructor.
FillColorList	Adds a list of colors to the color control.
FindColor	Locates a color in the color control.
GetButton	Returns the edit button window handle.
GetComboBox	Returns the combo box window handle.
GetCtlColor	Returns the current control colors (text, background) used.
GetCustomColor	Returns the current custom entry's color value.
GetDefaultColor	Returns the current default entry's color value.
RegisterApp	Registers an application with SftColor.
SetCtlColor	Sets the current control colors (text, background) used.
SetCustomColor	Sets the custom entry's current color value.
SetDefaultColor	Sets the default entry's current color value.
UnregisterApp	Unregisters an application from SftColor.

The color control also implements the following functions which are identical to the functions in the CComboBox class. Some functions are not available in Windows 3.1 and Window NT. Please see the MFC reference material and help files for more information.

AddString	Adds an item. The string used should be a color name. The RGB value of the color has to be added using SetItemData. Use FillColorList to add many items using a list
DeleteString	Deletes an item.
FindString	Returns the index of the first item that begins with the specified color name (to find a color by RGB value use FillColorList).
FindStringExact	Returns the index of the first item exactly matching the specified color name (to find a color by RGB value use FillColorList).
GetCount	Returns the number of items.
GetCurSel	Returns the index of the currently selected item.
GetDroppedControlRect	Fills the specified rectangle structure with the screen coordinates of the drop-down list.
GetDroppedState	Returns TRUE if the drop-down list is open, otherwise FALSE is returned.
GetExtendedUI	Returns TRUE if the extended user-interface is used, otherwise FALSE is returned.
GetItemData	Returns the RGB value associated with the specified item.
GetItemHeight	Returns the height (pixels) of the specified owner-drawn item.
GetLBText	Copies the specified item's color name to the specified buffer.
GetLBTextLen	Returns the length in characters of the specified item's color name.
GetLocale	Returns the current locale for the list.
InsertString	Inserts a list item at the specified position. The string used should be a color name. The RGB value of the color has to be added using SetItemData. Use FillColorList to add many items using a list.
ResetContent	Removes the contents of a color control.
SelectString	Selects the first item that begins with the characters in the specified text.
SetCurSel	Sets the current selection.
SetExtendedUI	Sets or clears the extended user-interface flag. Changes the keys that open and close the list in the color control.
SetItemData	Sets the RGB value associated with the specified item.
SetItemHeight	Sets the height of the specified owner-drawn list item.
SetLocale	Sets the current locale for the list.
ShowDropDown	Shows or hides the drop-down list.

OWL/C++ SftColor Classes

TSftColor Class, Member Functions

TSftColor is derived from TControl.

~TSftColor	Destructor.
FillColorList	Adds a list of colors to the color control.
FindColor	Locates a color in the color control.
GetButton	Returns the edit button window handle.
GetComboBox	Returns the combo box window handle.
GetCtlColor	Returns the current control colors (text, background) used.
GetCustomColor	Returns the current custom entry's color value.
GetDefaultColor	Returns the current default entry's color value.
RegisterApp	Registers an application with SftColor.
SetCtlColor	Sets the current control colors (text, background) used.
SetCustomColor	Sets the custom entry's current color value.
SetDefaultColor	Sets the default entry's current color value.
TSftColor	Constructor.
UnregisterApp	Unregisters an application from SftColor.

The color control also implements the following functions which are identical to the functions in the TComboBox class. Some functions are not available in Windows 3.1 and Window NT. Please see the OWL reference material and help files for more information.

AddString	Adds an item. The string used should be a color name. The RGB value of the color has to be added using SetItemData. Use FillColorList to add many items using a list
ClearList	Removes the contents of a color control.
DeleteString	Deletes an item.
FindString	Returns the index of the first item that begins with the specified color name (to find a color by RGB value use FillColorList).
GetCount	Returns the number of items.
GetDroppedControlRect	Fills the specified rectangle structure with the screen coordinates of the drop-down list.
GetDroppedState	Returns TRUE if the drop-down list is open, otherwise FALSE is returned.
GetExtendedUI	Returns TRUE if the extended user-interface is used, otherwise FALSE is returned.
GetItemData	Returns the RGB value associated with the specified item.
GetItemHeight	Returns the height (pixels) of the specified owner-drawn item.
GetSelIndex	Returns the index of the currently selected item.
GetString	Copies the specified item's color name to the specified buffer.
GetStringLen	Returns the length in characters of the specified item's color name.
HideList	Hides the drop-down list.
InsertString	Inserts a list item at the specified position. The string used should be a color name. The RGB value of the color has to be added using SetItemData. Use FillColorList to add many items using a list.
SetExtendedUI	Sets or clears the extended user-interface flag. Changes the keys that open and close the list in the color control.
SetItemData	Sets the RGB value associated with the specified item.
SetItemHeight	Sets the height of the specified owner-drawn list item.
SetSelIndex	Sets the current selection.
SetSelString	Selects the first item that begins with the characters in the specified text.
ShowList	Shows or hides the drop-down list.

C, C++ API

An application communicates with the SftColor color control by sending messages using the Windows SendMessage function. To simplify the process, SftColor offers not only the direct SendMessage interface, but for C programmers also a predefined "message-cracker" macro for each message. With C++, class member functions are offered for each message. This eliminates the casting of parameters when using SendMessage, but is just as efficient as a SendMessage call, because the macros and member functions expand into a SendMessage call. All samples shown in the reference section use these macros and member functions. The *wParam* and *lParam* message parameters are also documented, mainly for use with other programming languages, which cannot use the message-cracker macros.

Definitions and Structures

SFTCOLOR_CLASS	Color control window class name.
SFTCOLOR_LIST	Structure used to describe a list of colors and color names.
SFTCOLOR_PARM	Structure used to describe the color control's text and background color attributes.

Messages and Functions

The SendMessage Windows API function can be used to send messages to a color control.

SftColor_FillColorList	Adds a list of colors to the color control.
SftColor_FindColor	Locates a color in the color control.
SftColor_GetButton	Returns the edit button window handle.
SftColor_GetComboBox	Returns the combo box window handle.
SftColor_GetCtlColor	Returns the current control colors (text, background) used.
SftColor_GetCustomColor	Returns the current custom entry's color value.
SftColor_GetDefaultColor	Returns the current default entry's color value.
SftColor_RegisterApp	Registers an application with SftColor.
SftColor_SetCtlColor	Sets the current control colors (text, background) used.
SftColor_SetCustomColor	Sets the custom entry's current color value.
SftColor_SetDefaultColor	Sets the default entry's current color value.
SftColor_UnregisterApp	Unregisters an application from SftColor.

The color control also respond to the following combo box messages. Some messages are not available in Windows 3.1 and Window NT. Please see the Windows API reference and help files for more information.

CB_ADDSTRING	Adds an item. The string used should be a color name. The RGB value of the color has to be added using CB_SETITEMDATA. Use SftColor_FillColorList to add many items using a list.
CB_DELETESTRING	Deletes an item.
CB_FINDSTRING	Returns the index of the first item that begins with the specified color name (to find a color by RGB value use SftColor_FindColor).
CB_FINDSTRINGEXACT	Returns the index of the first item exactly matching the specified color name (to find a color by RGB value use SftColor_FindColor).
CB_GETCOUNT	Returns the number of items.
CB_GETCURSEL	Returns the index of the currently selected item.
CB_GETDROPPEDCONTROLRECT	Fills the specified rectangle structure with the screen coordinates of the drop-down list.
CB_GETDROPPEDSTATE	Returns TRUE if the drop-down list is open, otherwise FALSE is returned.
CB_GETDROPPEDWIDTH	Returns the minimum allowable width (pixels) of the drop down list.
CB_GETEXTENDEDUI	Returns TRUE if the extended user-interface is used, otherwise FALSE is returned.
CB_GETITEMDATA	Returns the RGB value associated with the specified item.

CB_GETITEMHEIGHT	Returns the height (pixels) of the specified owner-drawn item.
CB_GETLBTEXT	Copies the specified item's color name to the specified buffer.
CB_GETLBTEXTLEN	Returns the length in characters of the specified item's color name.
CB_GETLOCALE	Returns the current locale for the list.
CB_GETTOPINDEX	Returns the index of the first visible item in the drop down list.
CB_INSERTSTRING	Inserts a list item at the specified position. The string used should be a color name. The RGB value of the color has to be added using CB_SETITEMDATA. Use SftColor_FillColorList to add many items using a list.
CB_RESETCONTENT	Removes the contents of a color control.
CB_SELECTSTRING	Selects the first item that begins with the characters in the specified text.
CB_SETCURSEL	Sets the current selection.
CB_SETDROPPEDWIDTH	Sets the minimum allowable width (pixels) of the drop down list.
CB_SETEXTENDEDUI	Sets or clears the extended user-interface flag. Changes the keys that open and close the list in the color control.
CB_SETITEMDATA	Sets the RGB value associated with the specified item.
CB_SETITEMHEIGHT	Sets the height of the specified owner-drawn list item.
CB_SETLOCALE	Sets the current locale for the list.
CB_SETTOPINDEX	Scrolls the drop down list so the specified item is at the top of the visible range.
CB_SHOWDROPDOWN	Shows or hides the drop-down list.

SFTCOLOR_CLASS

WIN16
```
#define SFTCOLOR_CLASS "SoftelColor"
```
WIN32
```
#define SFTCOLOR_CLASS "SoftelColor32"
```

The SFTCOLOR_CLASS constant can be used when the SftColor control class name is required.

SFTCOLOR_LIST

```
typedef struct tagColorList {
    HINSTANCE hInst;                    /* instance handle of application */
#if defined(UNICODE) || defined(_UNICODE)
    LPCTSTR lpszNames;                  /* String with comma delimited color names */
#else
    LPCSTR lpszNames;                   /* String with comma delimited color names */
#endif
    COLORREF FAR* lpColors;             /* Array of color values */
} FAR* LPSFTCOLOR_LIST, SFTCOLOR_LIST;

typedef const SFTCOLOR_LIST FAR* LPCSFTCOLOR_LIST;
```

The SFTCOLOR_LIST structure is used to describe the color entries to be added to the color control.

Members

hInst

The instance handle of the application. This instance handle is only used if *lpszNames* is a string resource identifier. The instance handle is used to load the string containing the color names.

lpszNames

A string identifier or a string pointer. The string described by *lpszNames* should contain comma-delimited color names. These color names will be used as the text portion of the color control. The number of comma-delimited names must match the number of color entries described by *lpColors* exactly. When using a string identifier, MAKEINTRESOURCE must be used.

lpColors

A pointer to one or more COLORREF fields, each containing a color to be added to the color control. The number of color entries must match the number of comma-delimited names described by *lpszNames* exactly. In addition to the RGB values, two special colors have been predefined using the following values:

SFTCOLOR_CUSTOM	A customizable color entry. A color combo box entry added using this color value (and a suitable string) can be used to represent a user customizable color. If the style SFTCOLORSTYLE_CUSTOMEDIT is given, this custom color entry is automatically added to the color control when it is created. The edit button is enabled when the user selects the custom color entry. The actual RGB value of the custom color entry can be set using SftColor_SetCustomColor, CSftColor::SetCustomColor and TSftColor::SetCustomColor.
SFTCOLOR_DEFAULT	The default color. A color combo box entry added using this color value (and a suitable string) can be used to represent an application's predefined color value. The actual RGB value of the default color entry can be set using SftColor_SetDefaultColor, CSftColor::SetDefaultColor and TSftColor::SetDefaultColor.

Comments

SftColor_FillColorList, CSftColor::FillColorList or TSftColor::FillColorList are used to add color entries to a color control.

Example

This example fills a color control with four colors and a *(Default)* color entry:

C:

```
    SFTCOLOR_LIST List;
    COLORREF aColors[] = {
        /*Default*/    SFTCOLOR_DEFAULT,
        /*Red*/        RGB(255,  0,  0),
        /*White*/      RGB(255,255,255),
        /*Blue*/       RGB(  0,  0,255),
        /*Yellow*/     RGB(255,255,  0),
```

```
    };
    HWND hwndCtl = GetDlgItem(hwndDlg, IDC_COLOR2);

    List.hInst = hInst;
    List.lpszNames = "(Default),Red,White,Blue,Yellow";
    List.lpColors = aColors;
    SftColor_FillColorList(hwndCtl, &List, TRUE);
```

C++/MFC:

```
    SFTCOLOR_LIST List;
    COLORREF aColors[] = {
        /*Default*/    SFTCOLOR_DEFAULT,
        /*Red*/        RGB(255,  0,  0),
        /*White*/      RGB(255,255,255),
        /*Blue*/       RGB(  0,  0,255),
        /*Yellow*/     RGB(255,255,  0),
    };

    List.hInst = AfxGetResourceHandle(); // where strings are
    List.lpszNames = "(Default),Red,White,Blue,Yellow";
    List.lpColors = aColors;
    m_Color.FillColorList(&List, TRUE);
```

C++/OWL:

```
    SFTCOLOR_LIST List;
    COLORREF aColors[] = {
        /*Default*/    SFTCOLOR_DEFAULT,
        /*Red*/        RGB(255,  0,  0),
        /*White*/      RGB(255,255,255),
        /*Blue*/       RGB(  0,  0,255),
        /*Yellow*/     RGB(255,255,  0),
    };

    List.hInst = *GetApplication(); // where strings are
    List.lpszNames = "(Default),Red,White,Blue,Yellow";
    List.lpColors = aColors;
    m_pColor2->FillColorList(&List, TRUE);
    m_pColor2->SetSelIndex(0);
```

SFTCOLOR_PARM

```
typedef struct tagColorParm {
    COLORREF textColor;                 /* Text color */
    COLORREF backColor;                 /* Background color */
    HBRUSH hbr;                         /* Background brush */
} FAR* LPSFTCOLOR_PARM, SFTCOLOR_PARM;

typedef const SFTCOLOR_PARM FAR* LPCSFTCOLOR_PARM;
```

The SFTCOLOR_PARM structure is used to describe the color control's color attributes, used to paint the color control.

Members

textColor

The color used to paint the color control's text.

backColor

The color used to paint the color control's background.

hbr

A brush handle used to paint the color control's background. This should be a solid brush based on the same color as *backColor*.

Comments

The color control does not send WM_CTLCOLOR messages to the parent window. An application can set the color control's attributes using SftColor_SetCtlColor, CSftColorSetCtlColor and TSftColor::SetCtlColor instead.

Due to a combo box limitation under Windows 3.1 and Windows NT, the drop-down list will use the standard window background color (COLOR_WINDOW) if the list is not completely filled with entries.

The brush *hbr* has to remain valid until the color control no longer uses the supplied brush.

Example

This example sets the color control's colors to red text and light gray background:

C:

```
    SFTCOLOR_PARM Parm;
    Parm.textColor = RGB(255, 0, 0);      // red
    Parm.backColor = RGB(192, 192, 192);  // lt gray
    Parm.hbr = GetStockObject(LTGRAY_BRUSH);
    SftColor_SetCtlColor(hwndCtl, &Parm);
```

C++/MFC:

```
    SFTCOLOR_PARM Parm;
    Parm.textColor = RGB(255, 0, 0);      // red
    Parm.backColor = RGB(192, 192, 192);  // lt gray
    Parm.hbr = (HBRUSH) GetStockObject(LTGRAY_BRUSH);
    m_Color1.SetCtlColor(&Parm);
```

C++/OWL:

```
    SFTCOLOR_PARM Parm;
    Parm.textColor = RGB(255, 0, 0);      // red
    Parm.backColor = RGB(192, 192, 192);  // lt gray
    Parm.hbr = (HBRUSH) GetStockObject(LTGRAY_BRUSH);
    m_pColor1->SetCtlColor(&Parm);
```

Topic not available

This help topic is not available with the demo version of SftColor. You will receive a 65+ page manual and complete on-line help when purchasing SftColor.

SftColor supports C, C++ and other DLL-call capable languages. C++ class implementations for MFC and OWL are included. The source code for the SftColor DLL is also available.