Contents - SftTree/DLL Demo Version 2.0

Introduction

Softel vdm

Samples

Using Resource Editors

Programming

Reference

C

MFC

OWL

Notifications

This is an incorrectly licensed version of SftTree/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.

SftTree/DLL Overview

Product Description

SftTree/DLL is a custom control for the Windows operating system, offering multi-line, multi-column, hierarchical data displays.

Tree Control

SftTree/DLL offers many features; from simple, graphical list box displays to complex hierarchical data displays.

- Hierarchical item display
- Single and multiple text lines per item
- Single and multiple selection built-in
- Drag & drop with automatic scrolling
- Drag & drop within and outside tree control
- Single and multiple roots
- Expand/collapse buttons
- No clumsy tabs, true multi-column support
- Column headers with column titles or buttons
- Column headers with bitmaps
- Resizable columns
- Last column open-ended or fixed-width
- Row headers with titles or buttons
- Row headers with bitmaps
- Row/column header title or button
- Row/column header with bitmaps
- Selectable column alignment (left, right, center)
- Sorting
- Item data editing (cell editing) using Windows controls
- Cell fonts, colors and bitmaps
- Standard look or 3D items
- Tree items with individual attributes
- All bitmaps fully customizable
- Tree lines fully customizable
- Support for SDK dialog editors, AppStudio, Resource Workshop
- Complete implementation, not a sub/superclassed list box
- 16,000 item capacity under Windows 3.1, no preset limit with Windows NT and Windows 95

Royalties

Any application that you develop can use SftTree/DLL royalty-free in run-time only mode, design-time features are not available. Each user (developer) who needs access to design-time features must license a copy of SftTree/DLL.

Source Code

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

Languages Supported

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

Environments Supported

SftTree/DLL supports Windows 3.1 (incl. WIN32s), Windows NT and Windows 95 using the same easy to use API. Special UNICODE support is also available when running on Windows NT.

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 May 31, 1996. Subject to change.

.....	SftTree/DLL 2.0 for Windows, Windows NT and Windows 95 Single developer license, without DLL source code $279 each copy	$ .....
.....	SftTree/DLL 2.0 for Windows, Windows NT and Windows 95 Single developer license, including DLL source code $389 each copy	$ .....
.....	Upgrade, SftTree/DLL 2.0 Source Code Only Single developer license $110 each copy Enter License # _____	$ .....
	6% Sales Tax (NJ residents only)	$ .....
	Shipping and Handling ($10.00 per copy, $15.00 international air mail)	$ .....
	Total	$ .....

Product Support

If you experience difficulties using SftTree/DLL, 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
Email	support@softelvdm.com
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.

Using SftTree

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

First, a tree control is added to a dialog using a resource editor. When the dialog is later used in an application, the tree control is automatically created and can be accessed using the supplied API. A tree control can also be created outside of a dialog. This is documented in the language specific programming sections C Programming, C++ using the Microsoft Foundation Class library and C++ using the ObjectWindows Library.

Once the tree control has been created, the API functions documented in the programming sections can be used to add items, define attributes, enable tree components, etc. The following samples create a very minimal tree control with three items as shown below. This example can easily be extended by adding a few calls to define item bitmaps and other tree components to change the appearance of the tree control.

Note: SftTree/DLL Tree Wizard should be used to design the tree control look. With the SftTree/DLL Tree Wizard, most source code is generated for you.

C Sample

```
      HWND hwndTree;
      int index;

      hwndTree = GetDlgItem(hwndDialog, IDC_TREE);     // Get the tree window handle
      SftTree_SetShow3D(hwndTree, TRUE);               // Use 3D display mode
      index = SftTree_AddString(hwndTree, "The First Item");
      index = SftTree_AddString(hwndTree, "The Second Item");
      SftTree_SetItemLevel(hwndTree, index, 1);
      index = SftTree_AddString(hwndTree, "The Third Item");
      SftTree_SetItemLevel(hwndTree, index, 2);
      SftTree_RecalcHorizontalExtent(hwndTree);        // For optimal horizontal
  scrolling
```

C++/MFC Sample

```
      CSftTree m_Tree;
      int index;

      m_Tree.SubclassDlgItem(IDC_TREE, this);           // Connect the C++ object to
  the window
      m_Tree.SetShow3D(TRUE);                          // Use 3D display mode
      index = m_Tree.AddString(The First Item);
      index = m_Tree.AddString(The Second Item);
      m_Tree.SetItemLevel(index, 1);
      index = m_Tree.AddString(The Third Item);
      m_Tree.SetItemLevel(index, 2);
      m_Tree.RecalcHorizontalExtent();                 // For optimal horizontal
  scrolling
```

C++/OWL Sample

```
      TSftTree* pTree = TSftTree(thisDialog, IDC_TREE);// Get the tree window object
      int index;

      pTree->SetShow3D(TRUE);                          // Use 3D display mode
      index = pTree->AddString("The First Item");
      index = pTree->AddString("The Second Item");
      pTree->SetItemLevel(index, 1);
      index = pTree->AddString("The Third Item");
      pTree->SetItemLevel(index, 2);
```

```
    pTree->RecalcHorizontalExtent();                // For optimal horizontal
scrolling
```

Tree Components

The following describes the individual components that are available in a SftTree/DLL tree control. All components, except for the first column displaying an item's text, are optional. By turning all optional components off, the SftTree/DLL tree control can visually act as a simple list box.

Component	*Description*
Cell Bitmap	The cell bitmap is an optional bitmap displayed in a cell as graphic component. The cell bitmap can be of any size, and any number of different bitmaps can be used, but all cell bitmaps used must be the same size.
Cell Text	Each item has one cell per column. The cell text may be set anytime after an item has been added to the tree control. Each column's cell text may be single-line text or multi-line text, containing new-line ('\r\n') characters to begin a new line. Each column's cell text may be left or right justified or centered within the column boundaries.
Cell	Each item has one cell per column. The cell can contain optional text (cell text) and an optional graphic component (cell bitmap).
Column Headers	The optional column header is available for single- and multi-column trees, with labeled buttons or just titles. Each header or button label may be left or right justified or centered within the column boundaries. Column headers can contain bitmaps.
Column Resizing	The individual columns may optionally be resized using the mouse. Resizing is possible by positioning the mouse cursor in the header between two columns, pressing the left mouse button and moving the cursor to the desired new location.
Expand/Collapse Button	Expand/collapse buttons are optional buttons displayed on the tree lines to allow users to expand and collapse tree sections by clicking a button. The button bitmaps can be modified and are shared between all items. Items on level 0 (the highest level or root level) have special level 0 expand/collapse buttons which can be enabled separately, but share the same bitmaps. Expand/collapse buttons (other than level 0 buttons) are only shown if tree lines are also shown.
Grid Lines	Optional grid lines can be shown to help separate items horizontally and vertically. Grid lines are thin lines drawn vertically between columns and horizontally between items.
Item Bitmap	The item bitmap is an optional bitmap displayed to the left of the cell of the first or only column. The item bitmaps can be of any size and any number of different bitmaps can be used, but all item bitmaps used must be the same size. Default and individual bitmaps can be assigned to items.
Item	An item (also called row) is the term used to refer to one element added to the tree control and includes the row header, all bitmaps, tree lines, expand/collapse buttons and all cells.
Label Bitmap	The label bitmap is an optional bitmap. The label bitmap areas of all items are always aligned, regardless of item levels, so all label bitmaps are displayed in one column, without indentation. The label bitmap can be of any size and any number of different bitmaps can be used, one for each item (all label bitmaps used must be the same size).
Plus/Minus Bitmap	The plus/minus bitmap is an optional bitmap, which indicates the current expand status of an item. Three bitmaps can be defined, one for a (non-expandable) leaf item, one for an expanded parent item and one for a non-expanded parent item, all the same size. All items share the same three bitmaps and based on the expand status, the appropriate graphic is displayed. Individual items cannot override the bitmaps used.
Row Headers	The optional row headers are available for single- and multi-column trees, with labeled buttons or just titles. The header or button labels may be left

	or right justified or centered within the available width. Row headers can contain bitmaps.
Row	A row (also called item) is the term used to refer to one element added to the tree control and includes the row header, all bitmaps, tree lines, expand/collapse buttons and all cells.
Row/Column Header	The optional row/column header is available if both row and column headers are available. The row/column header can be shown as a labeled button or just a title. The header or button label may be left or right justified or centered within the row header area boundaries. The row/column header can contain a bitmap.
Text Selection	SftTree/DLL offers two different display methods for selected items. The entire item may be highlighted or only the text portion.
Tree Lines	Tree lines are optional lines connecting individual items. Tree lines are required if expand/collapse buttons need to be displayed.

Cell Bitmap

The cell bitmap is an optional bitmap displayed in a cell as graphic component. The cell bitmap can be of any size, and any number of different bitmaps can be used, but all cell bitmaps used must be the same size.

Cell Text

Each item has one cell per column. The cell text may be set anytime after an item has been added to the tree control. Each column's cell text may be single-line text or multi-line text, containing new-line ('\r\n') characters to begin a new line. Each column's cell text may be left or right justified or centered within the column boundaries.

Cell

Each item has one cell per column. The cell can contain optional text (cell text) and an optional graphic component (cell bitmap).

Column Headers

The optional column header is available for single- and multi-column trees, with labeled buttons or just titles. Each header or button label may be left or right justified or centered within the column boundaries. Column headers can contain bitmaps.

Column Resizing

The individual columns may optionally be resized using the mouse. Resizing is possible by positioning the mouse cursor in the header between two columns, pressing the left mouse button and moving the cursor to the desired new location.

Expand/Collapse Buttons

Expand/collapse buttons are optional buttons displayed on the tree lines to allow users to expand and collapse tree sections by clicking a button. The button bitmaps can be modified and are shared between all items. Items on level 0 (the highest level or root level) have special level 0 expand/collapse buttons which can be enabled separately, but share the same bitmaps. Expand/collapse buttons (other than level 0 buttons) are only shown if tree lines are also shown.

Grid Lines

Optional grid lines can be shown to help separate items horizontally and vertically. Grid lines are thin lines drawn vertically between columns and horizontally between items.

Item Bitmap

The item bitmap is an optional bitmap displayed to the left of the cell of the first or only column. The item bitmaps can be of any size and any number of different bitmaps can be used, but all item bitmaps used must be the same size. Default and individual bitmaps can be assigned to items.

Item

An item (also called row) is the term used to refer to one element added to the tree control and includes the row header, all bitmaps, tree lines, expand/collapse buttons and all cells.

Label Bitmap

The label bitmap is an optional bitmap. The label bitmap areas of all items are always aligned, regardless of item levels, so all label bitmaps are displayed in one column, without indentation. The label bitmap can be of any size and any number of different bitmaps can be used, one for each item (all label bitmaps used must be the same size).

Plus/Minus Bitmap

The plus/minus bitmap is an optional bitmap, which indicates the current expand status of an item. Three bitmaps can be defined, one for a (non-expandable) leaf item, one for an expanded parent item and one for a non-expanded parent item, all the same size. All items share the same three bitmaps and based on the expand status, the appropriate graphic is displayed. Individual items cannot override the bitmaps used.

Row Headers

The optional row headers are available for single- and multi-column trees, with labeled buttons or just titles. The header or button labels may be left or right justified or centered within the available width. Row headers can contain bitmaps.

Row

A row (also called item) is the term used to refer to one element added to the tree control and includes the row header, all bitmaps, tree lines, expand/collapse buttons and all cells.

Row/Column Header

The optional row/column header is available if both row and column headers are available. The row/column header can be shown as a labeled button or just a title. The header or button label may be left or right justified or centered within the row header area boundaries. The row/column header can contain a bitmap.

Text Selection

SftTree/DLL offers two different display methods for selected items. The entire item may be highlighted or only the text portion.

Tree Lines

Tree lines are optional lines connecting individual items. Tree lines are required if expand/collapse buttons need to be displayed.

Tree Items

Items within tree controls have relationships to other items based on the level they are on. This determines how items are connected to each other. The following table defines the terms used throughout to identify relationships and the connecting lines drawn for each type:

Term	*Description*
Parent	A parent item can be located at any level. In order to become a parent item, an item has to have one or more dependents, which are items which immediately follow the parent item and are on a lower level. A parent item may be expanded and collapsed. Expanding a parent item means making its immediate dependents (dependents on the next lower level) visible. Collapsing a parent item means hiding all its dependents (on all lower levels). Parent items may have sibling and parent items.
Top Parent	A parent item is a top parent (or top level parent) if it has no parent of its own. Top parent items may have sibling items.
Leaf Item	A leaf item is an item which has no dependents, i.e., it is not a parent item. Leaf items may have sibling and parent items.
Dependents	A dependent is an item which has a parent. The item is said to be a dependent of the parent item. Dependents can be parent items themselves or leaf items. A dependent cannot be at the root level.
Siblings	A sibling is an item which precedes or follows another item on the same level with the same parent. An item can have zero or more sibling items. Sibling items can be parent and leaf items.
Root Level	Any item without parent is at the root level, usually level 0. Multiple items can be at the root level, parent items or leaf items.
Expand Status	The expand status indicates whether dependent items are visible or not. A parent item that has one or more visible dependents is considered expanded. If a parent item is expanded, all its immediate dependents, i.e., all dependents on the next lower level are visible. If a parent item is collapsed, no immediate dependents are visible.
Visibility Status	The visibility status indicates whether an item is visible or not. An item is considered visible if its immediate and all other parent items are expanded. An item is considered visible even if it isn't currently displayed in the window client area.

When adding items to a tree control, all that is required from an application is that the level numbers be set. The SftTree/DLL tree control automatically determines the correct relationships and draws connecting lines appropriately. Applications can then interrogate the tree control about relationships and don't have to manage these themselves. Because SftTree/DLL determines the proper relationships, it is impossible for an application to construct an invalid tree. Even when levels are set incorrectly, SftTree/DLL can build a correct tree (although it may not be the desired result).

Parent

A parent item can be located at any level. In order to become a parent item, an item has to have one or more dependents, which are items which immediately follow the parent item and are on a lower level. A parent item may be expanded and collapsed. Expanding a parent item means making its immediate dependents (dependents on the next lower level) visible. Collapsing a parent item means hiding all its dependents (on all lower levels). Parent items may have sibling and parent items.

Top Parent

A parent item is a top parent (or top level parent) if it has no parent of its own. Top parent items may have sibling items.

Leaf Item

A leaf item is an item which has no dependents, i.e., it is not a parent item. Leaf items may have sibling and parent items.

Dependents

A dependent is an item which has a parent. The item is said to be a dependent of the parent item. Dependents can be parent items themselves or leaf items. A dependent cannot be at the root level.

Siblings

A sibling is an item which precedes or follows another item on the same level with the same parent. An item can have zero or more sibling items. Sibling items can be parent and leaf items.

Root Level

Any item without parent is at the root level, usually level 0. Multiple items can be at the root level, parent items or leaf items.

Expand Status

The expand status indicates whether dependent items are visible or not. A parent item that has one or more visible dependents is considered expanded. If a parent item is expanded, all its immediate dependents, i.e., all dependents on the next lower level are visible. If a parent item is collapsed, no immediate dependents are visible.

Visibility Status

The visibility status indicates whether an item is visible or not. An item is considered visible if its immediate and all other parent items are expanded. An item is considered visible even if it isn't currently displayed in the window client area.

Expand/Collapse

SftTree/DLL supports expanding and collapsing tree items with minimal application intervention. Items in a tree control are managed as a linear list, or an array of items. The application can expand and collapse items without having to add or remove items. The tree control takes care of hiding and making items visible as needed. Under program control, an application responds to SFTTREEN_LBUTTON-type notifications and expands and collapses items using SetItemExpand.

Drag & Drop

SftTree/DLL supports a drag & drop protocol by sending WM_COMMAND messages to the parent window, if the tree control has the appropriate window style SFTTREESTYLE_DRAGDROP defined. When dragging within the tree control, the drop target will automatically be moved and the data will automatically start scrolling vertically when the mouse cursor moves into the areas close to the upper or lower window border. Once the mouse cursor goes beyond those areas, indicating a drag operation outside the tree control, scrolling stops automatically. No application program intervention is required for this to take place. A header or scroll bar is not required. This drag-scrolling support is identical to the drag-scrolling supported by OLE conventions.

Selections

A SftTree/DLL tree control supports single and multiple selection, based on the window style used when the tree control is created.

Single Selection

In single selection mode, only one item can be selected (highlighted) at a time. When a new item is selected, the previously selected item is then no longer selected. The current position (or caret location) is automatically selected, except while a drag & drop operation is in progress. The caret location indicates the drop target and the selected item is the source. Drag & drop can cause the caret location to be a deselected item.

Items can be selected using the mouse by clicking anywhere on the item to be selected, from row header (leftmost position), label bitmap to the right side of the window area. Moving the caret location, using the arrow keys, also automatically selects the item at the caret location.

Multiple Selection

In multiple selection mode, many items can be selected (highlighted) at the same time. The current position (or caret location) is automatically moved to the new location when clicking anywhere on an item. An item is selected when clicking on the text portion. Clicking on other components, such as the label bitmap, tree lines, etc., does not select an item, unless the application handles the event and causes the item to be selected.

Items can be selected using the mouse by clicking anywhere on the text portion of an item or by using arrow keys. The new selection replaces the previous selection. If the CONTROL key is used, the selection is added to previous selections. If the SHIFT key is used, the entire range of items is selected, from the first selected item to the newly selected item. If the SHIFT and the CONTROL keys are pressed, a range of items is added to the previous selection(s).

Columns

A SftTree/DLL tree control can define multiple columns, each with its individual text alignment (left, right, center). A tree control can also define a header with titles, each with its individual text alignment. Columns can be resized by the user without application program intervention (unless disabled by the application). By dragging the separator between titles (or buttons), users can adjust columns to their particular needs. An application could save these column widths in an INI file or the registry for future use.

Item Data Editing

SftTree/DLL supports editing of item data using a unique approach. SftTree/DLL will notify your application of certain events (such a double-clicking on a tree item). Your application can then create any Windows control at a location specified by SftTree/DLL. The Windows control is "attached" to the tree control, but your application receives all messages and notifications for the new control, so even owner-drawn controls or custom controls can be used. SftTree/DLL will notify your application when editing (using your control) should end, either with or without data validation.

Demo Application


Click here to run the Demo application. If the application cannot be started, please reinstall the SftTree/DLL demo.

Overview

During the installation of SftTree/DLL, an icon for the demo application "DLL Demo" is installed in the Program Manager group *SftTree 2.0*.

The source code for the demo application DEMO.EXE is included with SftTree/DLL. The demo application is written in C and was built with SftTabs 2.0, a tab control for Windows. To recompile the sample, the product SftTabs 2.0 is required. SftTabs 2.0 allows existing applications written in C or C++ to be easily upgraded to include tabbed dialogs and tabbed windows, without rewriting existing dialog procedures or dialog class implementations.

The source code can be found in the directory C:\SFTTREE\SAMPLES\C\DEMO.

Wizard Application

Click here to run the 16-bit Wizard application. If the application cannot be started, please reinstall the SftTree/DLL demo from the installation disks.

Overview

During the installation of SftTree/DLL, an icon for the application "SftTree/DLL Wizard" is installed in the Program Manager group *SftTree 2.0*.

This application can be used to generate most of the source code needed to interact with a tree control in a dialog or a window. It honors most SftTree/DLL attributes and should be used at design-time to build the necessary tree control initialization code.

The SftTree/DLL Wizard application is used to design a tree control look. Most tree control attributes can be manipulated, except bitmaps. Sample bitmaps are provided by SftTree/DLL Wizard and cannot be changed. Of course, an application can freely define its own bitmaps. Once the desired look has been achieved, the run-time source code used to initialize the tree control can be generated for C, C++/MFC and C++/OWL by clicking on the corresponding tab. The generated source code contains step by step instructions on how to incorporate it into an application.

Rebuilding the DLLs

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

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 SftTree/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 SftTree/DLL source code, please make sure to test the resulting DLL with the sample applications.

In order to rebuild the source code, you must have purchased the optional DLL source code.

If you wish to rebuild DLLs shipped with SftTree/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 SftTree/DLL can be found in the directory C:\SFTTREE\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 WIN32/s/c (incl. Windows NT without UNICODE support)***
Target	SFTTREE.DLL	SFTTR32**U**.DLL	SFTTR32.DLL
Required Source Files	BCT1TREE.C HELPER.C MCT1TREE.C SFTTREE.C TREEINIT.C SFTTREE.RC SFTTREE.DEF	 HELPER.C MCT2TREE.C SFTTREE.C TREEINIT.C SFTTREE.RC SFTTR32**U**.DEF	 HELPER.C MCT2TREE.C SFTTREE.C TREEINIT.C SFTTREE.RC SFTTR32.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 SftTree/DLL. For certain error conditions, SftTree/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. 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. The DLLs shipped with SftTree/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.

Special Considerations using Visual C++ (16-bit compiler)

Due to the segmented architecture of 16-bit modules, the compiler switch /Gy has to be specified when compiling SftTree/DLL for 16-bit mode. Using the Visual C++ IDE, the compiler switch can be set by using the *Options, Project* menu command, click the *Compiler…* button, then select *Custom Options* and check the *Enable Function-Level Linking* option (must be checked).

If this compiler switch is not specified, the following compiler error will be issued:

```
fatal error C1050: 'DrawCaret' : code segment too large
```

The actual function name shown (DrawCaret in this example) may differ.

Special Considerations for Windows NT

To rebuild the UNICODE version of SftTree/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 SFTTR32U.DLL, the non-UNICODE DLL is named SFTTR32.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 32-bit 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	SFTTREE.LIB	SFTTR32V.LIB	SFTTR32B.LIB

C Programming

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

Building an Application

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

```
#include "sfttree.h"                    /* SftTree/DLL required header file */
```

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

B) In order to use SftTree/DLL controls, an application must call the SftTree_RegisterApp2 function. The call to this function is required so that SftTree/DLL window classes can be registered. This call has to be made before any SftTree/DLL controls are created. Add the following statement to your source code, where your application registers its window classes (normally during application initialization):

```
SftTree_RegisterApp2(hInstance);    /* Use SftTree/DLL with this application */
```

C) Once SftTree/DLL controls are no longer needed, an application must call the SftTree_UnregisterApp2 function. The call to this function is required so that SftTree/DLL window classes can be unregistered and cleanup processing can take place. This call has to be made after all SftTree/DLL controls have been destroyed (normally during application termination).

```
SftTree_UnregisterApp2(hInstance);  /* No longer use SftTree/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. SftTree/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 Visual C++***	***LIB File for Applications developed using Borland C++***	***DLL File Required at Run-Time***

Windows 3.1, 16-bit applications	SFTTREE.LIB	SFTTREE.LIB	SFTTREE.DLL
WIN32, all 32-bit environments including Windows NT (without UNICODE support)	SFTTR32.LIB	SFTTR32B.LIB	SFTTR32.DLL
Windows NT only (with UNICODE support)	SFTTR32U.LIB	SFTTR32V.LIB	SFTTR32U.DLL

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

Adding a Tree Control

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

- using dialog resources
- using CreateWindow(Ex)

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

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

```
hwndTree = CreateWindow(
    TEXT(SFTTREE_CLASS),              /* Window class */
    TEXT(""),                         /* Caption (none) */
    SFTTREESTYLE_NOTIFY |             /* Notify parent window */
    SFTTREESTYLE_DRAGDROP |           /* Drag & drop enabled */
    SFTTREESTYLE_LEFTBUTTONONLY |     /* Only respond to left mouse button */
    SFTTREESTYLE_SCROLL |             /* Honor WS_H/VSCROLL */
    SFTTREESTYLE_DISABLENOSCROLL |    /* Disable scrollbars instead of hiding */
    WS_HSCROLL | WS_VSCROLL |         /* Vertical and horizontal scrollbars */
    WS_VISIBLE | WS_CHILD,            /* Visible, child window */
    x, y, cx, cy,                     /* Location */
    hwndParent,                       /* Parent window */
    (HMENU) IDC_TREE,                 /* Tree control ID */
    app_instance,                     /* Application instance */
    NULL);
if (g_hwndTree == NULL)
    ; /* Error handling here */
```

For more information on the various parameters used, see the Windows API documentation. The tree control class is defined by the SFTTREE_CLASS constant (SFTTREE.H). The window class is **SoftelTreeControl** (Windows 3.1) or **SoftelTreeControl32** (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. Most applications may want to implement some of the notification handlers suggested here.

By using the SftTree/DLL Tree Wizard application, these event handlers are generated for you.

Expanding/Collapsing Items

A tree control will only send notification messages if its window style includes the SFTTREESTYLE_NOTIFY style.

By handling the appropriate notification, a tree control can respond to the mouse-button clicks on the small button bitmaps or (double-)clicks on other areas of the tree control. If an application does not implement an event handler, the tree items do not automatically expand and collapse. The following

code sample illustrates how the notifications could be handled:

WIN16:

```
switch (msg) {
case WM_COMMAND:                        /* WM_COMMAND message handler */
    if ((int) wParam == IDC_TREE) {
        switch(HIWORD(lParam)) {
        case SFTTREEN_LBUTTONDBLCLK_TEXT:
        case SFTTREEN_LBUTTONDOWN_BUTTON:
        case SFTTREEN_LBUTTONDBLCLK_BUTTON: {
            int index;
            BOOL fExpand, fControl;
            HWND hwndCtl = (HWND) LOWORD(lParam);
            /* Get current position */
            index = SftTree_GetCaretIndex(hwndCtl);/* Get caret location */
            /* Check if item is expanded */
            fExpand = SftTree_GetItemExpand(hwndCtl, index);
            /* If the CONTROL key is pressed, expand all dependent levels */
            fControl = (BOOL)(GetKeyState(VK_CONTROL)&0x8000);
            /* Do the opposite */
            SftTree_SetItemExpand(hwndCtl, index, !fExpand, fControl);
            break;
          }
        }
    }
    break;
}
```

WIN32:

```
switch (msg) {
case WM_COMMAND:                        /* WM_COMMAND message handler */
    if ((int) LOWORD(wParam) == IDC_TREE) {
        switch(HIWORD(wParam)) {
        case SFTTREEN_LBUTTONDBLCLK_TEXT:
        case SFTTREEN_LBUTTONDOWN_BUTTON:
        case SFTTREEN_LBUTTONDBLCLK_BUTTON: {
            int index;
            BOOL fExpand, fControl;
            HWND hwndCtl = (HWND) lParam;
            /* Get current position */
            index = SftTree_GetCaretIndex(hwndCtl);/* Get caret location */
            /* Check if item is expanded */
            fExpand = SftTree_GetItemExpand(hwndCtl, index);
            /* If the CONTROL key is pressed, expand all dependent levels */
            fControl = (BOOL)(GetKeyState(VK_CONTROL)&0x8000);
            /* Do the opposite */
            SftTree_SetItemExpand(hwndCtl, index, !fExpand, fControl);
            break;
          }
        }
    }
    break;
}
```

Drag & Drop

A tree control will only support drag & drop operations if its window style includes the SFTTREESTYLE_DRAGDROP style.

When a user initiates a drag & drop operation, a WM_COMMAND / SFTTREEN_BEGINDRAG notification is sent to the parent window. All items that are currently selected are part of the drag & drop operation. To abort the operation at this point, the application can clear all selections or send a WM_CANCELMODE message to the tree control. However, items may not be deleted or inserted during a drag & drop operation.

To find out more about the current drag & drop operation, an application can use SftTree_GetDragInfo, which makes a pointer to a SFTTREE_DRAGINFO structure available. This area is only valid while processing one WM_COMMAND notification and must be retrieved for each notification. The SFTTREE_DRAGINFO structure members are read/only unless otherwise indicated. SftTree_GetDragInfo should be used when processing SFTTREEN_BEGINDRAG, SFTTREEN_DRAGGING and SFTTREEN_ENDDRAG or SFTTREEN_CANCELDRAG notifications.

A user can abort a drag & drop operation by pressing the ESCAPE key, at which point an application will receive a SFTTREEN_CANCELDRAG notification.

For more information, see the SFTTREE_DRAGINFO structure.

Item Data Editing

A tree control will only generate the item data editing notifications if its window style includes the SFTTREESTYLE_NOTIFY style.

SftTree/DLL supports a very easy item data editing protocol. Unlike other custom controls, no new API has to be used to edit data in a SftTree/DLL tree control. Existing Windows controls can be used to edit item data, and because they are completely under your application's control, even owner-drawn controls and other custom controls can be used.

An application can "attach" a control to a SftTree/DLL control, by creating the control and defining the tree control as the control's parent. This control, usually used to edit item data, is completely under your application's control. The tree control forwards all messages for the control directly to your application. This control can be created in response to a mouse button click (or double-click), or any other reasonable event in your application. Any Windows control can be used (edit controls, combo boxes, etc.). SftTree_GetEditColumnRect can be used to determine the proper location for the control.

This example shows how an application can respond to a left mouse-button double-click event by creating an edit control on the current item:

```
/* These routines handle item data editing. In this        */
/* example, an edit control is used. Any Windows control   */
/* can be used, even custom controls.                      */

/* Start editing in response to a                          */
/* SFTTREEN_LBUTTONDBLCLK_TEXT notification.               */

int m_editIndex;                        /* Index of item being edited */
int m_editCol;                          /* Column # being edited */
HWND m_hwndEdit;                        /* Control used for editing */

static void StartEdit(void)
{
    char szBuffer[80];
    LPRECT lpRect;

    /* Get the item index where double-click happened */
    m_editIndex = SftTree_GetCaretIndex(g_hwndTree); /* Save item # */
    m_editCol = -1;                     /* Let SftTree/DLL provide column # */
    lpRect = SftTree_GetEditColumnRect(g_hwndTree, m_editIndex, &m_editCol);
    if (!lpRect)                        /* No column active */
        return;

    /* Make the column completely visible */
    SftTree_MakeColumnVisible(g_hwndTree, m_editCol);
    /* Make the item completely visible */
    SftTree_MakeRowVisible(g_hwndTree, m_editIndex);
    /* Get the location again (in case it scrolled) */
    lpRect = SftTree_GetEditColumnRect(g_hwndTree, m_editIndex, &m_editCol);
    if (!lpRect)                        /* No column active */
        return;

    /* Change selection style to not shown anything */
    SftTree_SetNoFocusStyle(g_hwndTree, SFTTREE_NOFOCUS_NOTHING);
```

```
    /* Create the edit control.*/
    /* Based on the tree control attributes and your preference, you may */
    /* have to adjust the rectangle used for the edit control. */
    m_hwndEdit = CreateWindow(TEXT("EDIT"), TEXT(""),
                                        /* Class, Title */
            WS_CHILD|WS_BORDER|ES_LEFT|ES_AUTOHSCROLL, /* Styles */
            lpRect->left, lpRect->top, lpRect->right-lpRect->left, lpRect->bottom-
lpRect->top,
            g_hwndTree,                 /* Tree control */
            (HMENU) IDC_EDIT_your_id,   /* <-- provide unique ID */
            app_instance,               /* <-- provide instance handle */
            NULL);
    if (!m_hwndEdit)                    /* Failed */
        return;

    /* Set some edit control attributes */
    /* Set the font defined for item data editing */
    SendMessage(m_hwndEdit, WM_SETFONT, (WPARAM)(UINT)m_hEditFont, 0L);
    /* Copy the text found in the tree control */
    SftTree_GetTextCol(g_hwndTree, m_editIndex, m_editCol, szBuffer);
    SetWindowText(m_hwndEdit, szBuffer);
    /* Select all text in the edit control and display it */
    SendMessage(m_hwndEdit, EM_SETSEL, TRUE, MAKELPARAM(0, -1));
    ShowWindow(m_hwndEdit, SW_SHOW);
    SetFocus(m_hwndEdit);               /* Set input focus to the control */
}
```

WIN16:

```
    switch (msg) {
    case WM_COMMAND:                    /* WM_COMMAND message handler */
        if ((int) wParam == IDC_TREE) {
            switch(HIWORD(lParam)) {
            case SFTTREEN_LBUTTONDBLCLK_TEXT:
                /* Edit the current cell */
                StartEdit();
                break;
            case SFTTREEN_QUITEDIT:
                /* Abandon editing */
                QuitEdit();
                break;
            case SFTTREEN_VALIDATEEDIT:
                /* Validate input data */
                ValidateEdit();
                break;
            }
        }
        break;
    }
```

WIN32:

```
    switch (msg) {
    case WM_COMMAND:                    /* WM_COMMAND message handler */
        if ((int) LOWORD(wParam) == IDC_TREE) {
            switch(HIWORD(wParam)) {
            case SFTTREEN_LBUTTONDBLCLK_TEXT:
                /* Edit the current cell */
                StartEdit();
                break;
            case SFTTREEN_QUITEDIT:
                /* Abandon editing */
                QuitEdit();
                break;
            case SFTTREEN_VALIDATEEDIT:
                /* Validate input data */
```

```
                ValidateEdit();
                break;
            }
        }
        break;
    }
```

Once a tree control has an attached child window, it generates the SFTTREEN_QUITEDIT and SFTTREEN_VALIDATEEDIT notifications, which signal the tree's parent window to abandon editing by destroying any associated controls, or to validate the input data, issue error messages and/or destroy the controls. When an application receives the SFTTREEN_QUITEDIT notification, it must unconditionally abort editing by destroying all child controls.

```
/* Quit editing in response to a SFTTREEN_QUITEDIT         */
/* notification.                                           */
static void QuitEdit(void)
{
    if (m_hwndEdit) {
        /* If the control has the focus, set the focus back to */
        /* tree control after destroying the control */
        BOOL fHadFocus = (GetFocus() == m_hwndEdit);
        DestroyWindow(m_hwndEdit);
        m_hwndEdit = NULL;
        /* Restore nofocus display method */
        SftTree_SetNoFocusStyle(g_hwndTree, SFTTREE_NOFOCUS_FRAME);
        if (fHadFocus)
            SetFocus(g_hwndTree);        /* Back to tree control */
    }
}

/* Validate edit data in response to a                     */
/* SFTTREEN_VALIDATEEDIT notification.                     */

static void ValidateEdit(void)
{
    if (m_hwndEdit) {
        char szBuffer[80];
        /* Get the text from the edit control */
        GetWindowText(m_hwndEdit, szBuffer, sizeof(szBuffer));
        /* Validate the data */
        if (lstrcmp(TEXT(""), szBuffer) == 0) {
            MessageBox(NULL, TEXT("Just to demonstrate data input validation, ")
                             TEXT("this example rejects empty cells.  Please ")
                             TEXT("enter some data."), TEXT("SftTree/DLL"),
                             MB_OK|MB_TASKMODAL|MB_ICONSTOP);
            SetFocus(m_hwndEdit);
        } else {
            /* Save the data in the tree control */
            SftTree_SetTextCol(g_hwndTree, m_editIndex, m_editCol, szBuffer);
            DestroyWindow(m_hwndEdit);
            m_hwndEdit = NULL;
            /* Restore nofocus display method */
            SftTree_SetNoFocusStyle(g_hwndTree, SFTTREE_NOFOCUS_FRAME);
        }
    }
}
```

While editing item data using a control, the user may abort editing by pressing the ESCAPE key. This generates a SFTTREEN_QUITEDIT notification.

Additional Considerations

When creating controls for item data editing, a suitable font may have to be used. If a control is too small to display the data, the data may not only be clipped, but even be completely suppressed. This is particularly noticeable with edit controls.

If item data editing is started under program control during a WM_INITDIALOG message or anytime the tree control has not yet been painted, the tree control has to be painted explicitly before a child control can be attached. This can be accomplished using the *UpdateWindow* call.

3D and Colors

WM_CTLCOLOR

This message is generated by the tree control for compatibility with SftTree 1.0 only. When developing new applications, please use SftTree_SetCtlColor instead.

The appearance of the tree control can be modified by handling the WM_CTLCOLOR message. The parent window can override the default window colors used by the tree control by handling the WM_CTLCOLOR message. Even though the tree control is not a list box, it generates WM_CTLCOLOR messages as a list box would.

```
case WM_CTLCOLOR:
    if (HIWORD(lParam)== CTLCOLOR_LISTBOX) {
        SetTextColor((HDC) wParam, RGB(255,0,0));        // red text
        SetBkColor((HDC) wParam, RGB(192,192,192)); // on gray background
        return GetStockObject(LTGRAY_BRUSH);
    }
```

CTL3DV2 or CTL3D32

The tree control can be used with CTL3DV2 (or CTL3D32). More recent versions of CTL3DV2.DLL and CTL3D32.DLL allow user controls to receive a 3D border by using the Ctl3dSubclassCtlEx function.

```
Ctl3dSubclassCtlEx(hwndTreeCtl, CTL3D_LISTBOX_CTL);
```

This call is required even if Ctl3dAutoSubclass is used. For more information on CTL3DV2 or CTL3D32, see the Microsoft documentation.

When running under Windows 95 or Windows NT 4.0 and above, the window style WS_EX_CLIENTEDGE can be specified for the tree control, eliminating the need for CTL3DV2.DLL and CTL3D32.DLL.

C++/MFC Programming

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

Building an Application

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

```
    #include "sfttree.h"                        /* SftTree/DLL required header file
*/
```

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

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

```
    #include "sfttreem.cpp"                     /* SftTree implementation */
```

This include statement should appear after the #include "sfttree.h" statement. This is the preferred method to include the implementation of the CSftTree class. Adding the file SFTTREEM.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:\SFTTREE\INCLUDE (unless changed during the installation).

C) In order to use SftTree/DLL controls, an application must call the CSftTree::RegisterApp function. The call to this function is required so that SftTree/DLL window classes can be registered. This call has to be made before any SftTree/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:

```
    CSftTree::RegisterApp();              /* Use SftTree/DLL with this application
*/
```

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

```
    CSftTree::UnregisterApp();            /* No longer use SftTree/DLL */
```

E) The application's executable (EXE or DLL) must be linked with the correct LIB file, depending on the target environment. The SftTree 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	SFTTREE.LIB	SFTTREE.DLL
WIN32, all 32-bit environments including Windows NT (without UNICODE support)	SFTTR32.LIB	SFTTR32.DLL
Windows NT only (with UNICODE support)	SFTTR32U.LIB	SFTTR32U.DLL

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

Adding a Tree Control

ClassWizard does not support new classes such as CSftTree. So any tree 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 (such as a list box).

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

- using dialog resources
- using CSftTree::Create

Adding a tree control using dialog resources is accomplished by using a resource editor to design a dialog. Once a tree control is created, its CSftTree based object can be obtained by using the Windows GetDlgItem function or attached to a CSftTree object using SubclassWindow or SubclassDlgItem.

```
CSftTree * pTreeControl;
pTreeControl = (CSftTree *) GetDlgItem(IDC_TREE);
```

or

```
CSftTree m_Tree;
m_Tree.SubclassDlgItem(IDC_TREE, this);
```

Another method to create a tree control is by using the CSftTree::Create member function.

```
CSftTree m_Tree;
if (!m_Tree.Create(
    SFTTREESTYLE_NOTIFY |            /* Notify parent window */
    SFTTREESTYLE_DRAGDROP |          /* Drag & drop enabled */
    SFTTREESTYLE_LEFTBUTTONONLY |    /* Only respond to left mouse button */
    SFTTREESTYLE_SCROLL |            /* Honor WS_H/VSCROLL */
    SFTTREESTYLE_DISABLENOSCROLL |   /* Disable scrollbars instead of hiding */
    WS_HSCROLL | WS_VSCROLL |        /* Vertical and horizontal scrollbars */
    WS_VISIBLE | WS_CHILD,           /* Visible, child window */
    CRect(x, y, x+cx, y+cy),         /* Location */
    this,                            /* Parent window */
    IDC_TREE))                       /* Tree control ID */
        ; /* Error handling here */
```

Handling Notifications

As with standard Windows controls, applications must respond to events and messages to cause controls to respond to user requests. Most applications may want to implement some of the notification handlers suggested here.

By using the SftTree/DLL Tree Wizard application, these event handlers are generated for you.

ClassWizard does not support new classes such as CSftTree. So any tree 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 (such as a list box).

Expanding/Collapsing Items

A tree control will only send notification messages if its window style includes the SFTTREESTYLE_NOTIFY style.

By handling the appropriate notification, a tree control can respond to the mouse-button clicks on the small button bitmaps or (double-)clicks on other areas of the tree control. If an application does not implement an event handler, the tree items do not automatically expand and collapse. The following code sample illustrates how the notifications could be handled:

```
/* Add the following to your parent window class.          */
afx_msg void OnLButtonExpandCollapse();

/* Add the following to the parent window message map.     */
ON_SFTTREEN_LBUTTONDOWN_BUTTON(IDC_TREE, OnLButtonExpandCollapse)
ON_SFTTREEN_LBUTTONDBLCLK_BUTTON(IDC_TREE, OnLButtonExpandCollapse)

void CYourDialog::OnLButtonExpandCollapse()
{
```

```
    /* get index of item to expand/collapse */
    int index = m_Tree.GetCaretIndex();
    /* get current expand/collapsed status */
    BOOL fExpanded = m_Tree.GetItemExpand(index);
    /* if control key is used we'll expand all dependents */
    BOOL fDepth = (::GetKeyState(VK_CONTROL)&0x8000);
    m_Tree.SetItemExpand(index, !fExpanded, fDepth);
}
```

Drag & Drop

A tree control will only support drag & drop operations if its window style includes the SFTTREESTYLE_DRAGDROP style.

When a user initiates a drag & drop operation, a WM_COMMAND / SFTTREEN_BEGINDRAG notification is sent to the parent window. All items that are currently selected are part of the drag & drop operation. To abort the operation at this point, the application can clear all selections or send a WM_CANCELMODE message to the tree control. However, items may not be deleted or inserted during a drag & drop operation.

To find out more about the current drag & drop operation, an application can use CSftTree::GetDragInfo, which makes a pointer to a SFTTREE_DRAGINFO structure available. This area is only valid while processing one notification and must be retrieved for each notification. The SFTTREE_DRAGINFO structure fields are read/only unless otherwise indicated. CSftTree::GetDragInfo should be used when processing SFTTREEN_BEGINDRAG, SFTTREEN_DRAGGING and SFTTREEN_ENDDRAG or SFTTREEN_CANCELDRAG notifications.

A user can abort a drag & drop operation by pressing the ESCAPE key, at which point an application will receive a SFTTREEN_CANCELDRAG notification.

For more information, see the SFTTREE_DRAGINFO structure.

Item Data Editing

A tree control will only generate the item data editing notifications if its window style includes the SFTTREESTYLE_NOTIFY style.

SftTree/DLL supports a very easy item data editing protocol. Unlike other custom controls, no new API has to be used to edit data in a SftTree/DLL tree control. Existing Windows controls can be used to edit item data, and because they are completely under your application's control, even owner-drawn controls and other custom controls can be used.

An application can "attach" a control to a SftTree/DLL control, by creating the control and defining the tree control as the control's parent. This control, usually used to edit item data, is completely under your application's control. The tree control forwards all messages for the control directly to your application. This control can be created in response to a mouse button click (or double-click), or any other reasonable event in your application. Any Windows control can be used (edit controls, combo boxes, etc.). CSftTree::GetEditColumnRect can be used to determine the proper location for the control.

This example shows how an application can respond to a left mouse-button double-click event by creating an edit control on the current item:

```
/* These routines handle item data editing. In this        */
/* example, an edit control is used. Any Windows control   */
/* can be used, even custom controls.                      */

    /* Add the following to your parent window class.          */
    int m_editIndex;                    /* Index of item being edited */
    int m_editCol;                      /* Column # being edited */
    CEdit* m_pEdit;                     /* Control used for editing */

    afx_msg void OnStartEdit();         /* Start item data editing */
    afx_msg void OnQuitEdit();          /* Abandon item data editing */
    afx_msg void OnValidateEdit();      /* Validate input */

    /* Add the following to the parent window message map.     */
```

```
    ON_SFTTREEN_LBUTTONDBLCLK_TEXT(IDC_TREE, OnStartEdit)
    ON_SFTTREEN_QUITEDIT(IDC_TREE, OnQuitEdit)
    ON_SFTTREEN_VALIDATEEDIT(IDC_TREE, OnValidateEdit)

    /* Add the following to your parent window class           */
    /* constructor.                                            */
    m_pEdit = NULL;                     /* No item data editing */

/* Start editing in response to a                             */
/* SFTTREEN_LBUTTONDBLCLK_TEXT notification.                  */
void CYourDialog::OnStartEdit()
{
    CString str;
    LPCRECT lpRect;

    /* Get the item index where double-click happened */
    m_editIndex = m_Tree.GetCaretIndex(); /* Save item # */
    m_editCol = -1;                     /* Let SftTree/DLL provide column # */
    lpRect = m_Tree.GetEditColumnRect(m_editIndex, &m_editCol);
    if (!lpRect)                        /* No column active */
        return;

    /* Make the column completely visible */
    m_Tree.MakeColumnVisible(m_editCol);
    /* Make the item completely visible */
    m_Tree.MakeRowVisible(m_editIndex);
    /* Get the location again (in case it scrolled) */
    lpRect = m_Tree.GetEditColumnRect(m_editIndex, &m_editCol);
    if (!lpRect)                        /* No column active */
        return;

    /* Change selection style to not shown anything */
    m_Tree.SetNoFocusStyle(SFTTREE_NOFOCUS_NOTHING);

    /* Create the edit control.*/
    /* Based on the tree control attributes and your preference, you may */
    /* have to adjust the rectangle used for the edit control. */
    m_pEdit = new CEdit;
    m_pEdit->Create(
            WS_CHILD|WS_BORDER|ES_LEFT|ES_AUTOHSCROLL, /* Styles */
            (CRect) lpRect,
            &m_Tree,                    /* Tree control */
            IDC_EDIT_your_id);          /* <-- provide unique ID */

    /* Set some edit control attributes */
    /* Set the font defined for item data editing */
    m_pEdit->SetFont(&m_EditFont, FALSE);
    /* Copy the text found in the tree control */
    m_Tree.GetText(m_editIndex, m_editCol, str);
    m_pEdit->SetWindowText(str);
    /* Select all text in the edit control and display it */
    m_pEdit->SetSel(0, -1, TRUE);
    m_pEdit->ShowWindow(SW_SHOW);
    m_pEdit->SetFocus();                /* Set input focus to the control */
}
```

Once a tree control has one or more attached child windows, it generates the SFTTREEN_QUITEDIT and SFTTREEN_VALIDATEEDIT notifications, which signal the tree's parent window to abandon editing by destroying any associated controls, or to validate the input data, issue error messages and/or destroy the controls. When an application receives the SFTTREEN_QUITEDIT notification, it must unconditionally abort editing by destroying all child controls.

```
/* Quit editing in response to a SFTTREEN_QUITEDIT         */
/* notification.                                           */

void CYourDialog::OnQuitEdit()
{
```

```
    if (m_pEdit) {
        /* If the control has the focus, set the focus back to */
        /* tree control after destroying the control */
        BOOL fHadFocus = (GetFocus() == m_pEdit);
        m_pEdit->DestroyWindow();
        delete m_pEdit;
        m_pEdit = NULL;
        /* Restore nofocus display method */
        m_Tree.SetNoFocusStyle(SFTTREE_NOFOCUS_FRAME);
        if (fHadFocus)
            m_Tree.SetFocus();          /* Back to tree control */
    }
}

/* Validate edit data in response to a                    */
/* SFTTREEN_VALIDATEEDIT notification.                    */

void CYourDialog::OnValidateEdit()
{
    if (m_pEdit) {
        CString str;
        /* Get the text from the edit control */
        m_pEdit->GetWindowText(str);
        /* Validate the data */
        if (str == _T("")) {
            AfxMessageBox(_T("Just to demonstrate data input validation, ")
                          _T("this example rejects empty cells.  Please ")
                          _T("enter some data."));
            m_pEdit->SetFocus();
        } else {
            /* Save the data in the tree control */
            m_Tree.SetText(m_editIndex, m_editCol, str);
            m_pEdit->DestroyWindow();
            delete m_pEdit;
            m_pEdit = NULL;
            /* Restore nofocus display method */
            m_Tree.SetNoFocusStyle(SFTTREE_NOFOCUS_FRAME);
        }
    }
}
```

While editing item data using a control, the user may abort editing by pressing the ESCAPE key. This generates a SFTTREEN_QUITEDIT notification.

Additional Considerations

When creating controls for item data editing, a suitable font may have to be used. If a control is too small to display the data, the data may not only be clipped, but even be completely suppressed. This is particularly noticeable with edit controls.

If item data editing is started under program control during a WM_INITDIALOG message in the OnInitDialog member function or anytime the tree control has not yet been painted, the tree control has to be painted explicitly before a child control can be attached. This can be accomplished using the *UpdateWindow* call.

3D and Colors

WM_CTLCOLOR

This message is generated by the tree control for compatibility with SftTree 1.0 only. When developing new applications, please use CSftTree::SetCtlColor instead.

The appearance of the tree control can be modified by handling the WM_CTLCOLOR message. The parent window can override the default window colors used by the tree control by defining a WM_CTLCOLOR message handler. Even though the tree control is not a list box, it generates

WM_CTLCOLOR messages as a list box would.

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

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

// Event handler implementation
HBRUSH CYourDialog::OnCtlColorTree(CDC* pDC, CWnd* pWnd, UINT nCtlColor)
{
    if (nCtlColor == CTLCOLOR_LISTBOX) {
        pDC->SetTextColor(RGB(255,0,0));        // red text
        pDC->SetBkColor(RGB(192,192,192));      // on gray background
        return (HBRUSH)::GetStockObject(LTGRAY_BRUSH);
    }
    return CDialog::OnCtlColor(pDC, pWnd, nCtlColor);
}
```

CTL3DV2 or CTL3D32

The tree control can be used with CTL3DV2 (or CTL3D32). More recent versions of CTL3DV2.DLL and CTL3D32.DLL allow user controls to receive a 3D border by using the Ctl3dSubclassCtlEx function.

```
::Ctl3dSubclassCtlEx(pTree->m_hWnd, CTL3D_LISTBOX_CTL);
```

This call is required even if Ctl3dAutoSubclass is used. For more information on CTL3DV2 or CTL3D32, see the Microsoft documentation.

When running under Windows 95 or Windows NT 4.0 and above, the window style WS_EX_CLIENTEDGE can be specified for the tree control, eliminating the need for CTL3DV2.DLL and CTL3D32.DLL.

C++/OWL Programming

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

Building an Application

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

```
    #include "sfttree.h"                    /* SftTree/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:\SFTTREE\INCLUDE (unless changed during the installation).

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

```
    #include "sfttreeb.cpp"                 /* SftTree/DLL implementation */
```

This include statement should appear after the #include "sfttree.h" statement. This is the preferred method to include the implementation of the TSftTree class. Adding the file SFTTREEB.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:\SFTTREE\INCLUDE (unless changed during the installation).

C) In order to use SftTree/DLL controls, an application must call the TSftTree::RegisterApp function. The call to this function is required so that SftTree/DLL window classes can be registered. This call has to be made before any SftTree/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:

```
    TSftTree::RegisterApp();            /* Use SftTree/DLL with this application
*/
```

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

```
    TSftTree::UnregisterApp();          /* No longer use SftTree/DLL */
```

E) The application's executable (EXE or DLL) must be linked with the correct LIB file, depending on the target environment. SftTree/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	SFTTREE.LIB	SFTTREE.DLL
WIN32, all 32-bit environments including Windows NT (without UNICODE support)	SFTTR32B.LIB	SFTTR32.DLL
Windows NT (with UNICODE support)	SFTTR32V.LIB	SFTTR32U.DLL

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

Adding a Tree Control

ClassExpert does not support new classes such as TSftTree. So any tree 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 tree control to an application:

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

Adding a tree control using dialog resources is accomplished by using a resource editor to design a dialog. Once a tree control is created by creating the dialog, the TSftTree based object can be constructed by using the TSftTree constructor.

```
pTree = new TSftTree(this, IDC_TREE);
```

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

```
pTree = new TSftTree(this,              /* 'this' is the parent window */
    IDC_TREE,                           /* tree control ID */
    x, y, cx, cy);                      /* Location */
pTree->Attr.Style |=
    SFTTREESTYLE_NOTIFY |               /* Notify parent window */
    SFTTREESTYLE_DRAGDROP |             /* Drag & drop enabled */
    SFTTREESTYLE_LEFTBUTTONONLY |       /* Only respond to left mouse button */
    SFTTREESTYLE_SCROLL |               /* Honor WS_H/VSCROLL */
    SFTTREESTYLE_DISABLENOSCROLL |      /* Disable scrollbars instead of hiding */
    WS_HSCROLL | WS_VSCROLL |           /* Vertical and horizontal scrollbars */
    WS_VISIBLE | WS_CHILD;              /* Visible, child window */
if (!pTree->Create())
        ; /* Error handling here */
```

The constructor creates the tree control object. The arguments define the position of the tree control window once it is created using the Create function. For more information on the various parameters used, see the TSftTree::TSftTree constructor.

Handling Notifications

As with standard Windows controls, applications must respond to events and messages to cause controls to respond to user requests. Most applications may want to implement some of the notification handlers suggested here.

By using the SftTree/DLL Tree Wizard application, these event handlers are generated for you.

ClassExpert does not support new classes such as TSftTree. So any tree 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).

Expanding/Collapsing Items

A tree control will only send notification messages if its window style includes the SFTTREESTYLE_NOTIFY style.

By handling the appropriate notification, a tree control can respond to the mouse-button clicks on the small button bitmaps or (double-)clicks on other areas of the tree control. If an application does not implement an event handler, the tree items do not automatically expand and collapse. The following code sample illustrates how the notifications could be handled:

```
/* Respond to expand/collapse requests as the user clicks  */
/* on different tree components.  The events handled here  */
/* can be changed to suit your application.                */

    /* Add the following to your parent window class.          */
    void EvLButtonExpandCollapse();

    /* Add the following to the parent window message map.     */
    EV_SFTTREEN_LBUTTONDOWN_BUTTON(IDC_TREE, EvLButtonExpandCollapse),
    EV_SFTTREEN_LBUTTONDBLCLK_BUTTON(IDC_TREE, EvLButtonExpandCollapse),
```

```
void TYourDialog::EvLButtonExpandCollapse()
{
    /* get index of item to expand/collapse */
    int index = pTree->GetCaretIndex();
    /* get current expand/collapsed status */
    BOOL fExpanded = pTree->GetItemExpand(index);
    /* if control key is used we'll expand all dependents */
    BOOL fDepth = (::GetKeyState(VK_CONTROL)&0x8000);
    pTree->SetItemExpand(index, !fExpanded, fDepth);
}
```

Drag & Drop

A tree control will only support drag & drop operations if its window style includes the SFTTREESTYLE_DRAGDROP style.

When a user initiates a drag & drop operation, a WM_COMMAND / SFTTREEN_BEGINDRAG notification is sent to the parent window. All items that are currently selected are part of the drag & drop operation. To abort the operation at this point, the application can clear all selections or send a WM_CANCELMODE message to the tree control. However, items may not be deleted during a drag & drop operation.

To find out more about the current drag & drop operation, an application can use TSftTree::GetDragInfo, which makes a pointer to a SFTTREE_DRAGINFO structure available. This area is only valid while processing one notification and must be retrieved for each notification. The SFTTREE_DRAGINFO structure fields are read/only unless otherwise indicated. The TSftTree::GetDragInfo function should be used when processing SFTTREEN_BEGINDRAG, SFTTREEN_DRAGGING and SFTTREEN_ENDDRAG or SFTTREEN_CANCELDRAG notifications.

A user can abort a drag & drop operation by pressing the ESCAPE key, at which point an application will receive a SFTTREEN_CANCELDRAG notification.

For more information, see the SFTTREE_DRAGINFO structure.

Item Data Editing

A tree control will only generate the item data editing notifications if its window style includes the SFTTREESTYLE_NOTIFY style.

SftTree/DLL supports a very easy item data editing protocol. Unlike other custom controls, no new API has to be used to edit data in a SftTree/DLL tree control. Existing Windows controls can be used to edit item data, and because they are completely under your application's control, even owner-drawn controls and other custom controls can be used.

An application can "attach" a control to a SftTree/DLL control, by creating the control and defining the tree control as the control's parent. This control, usually used to edit item data, is completely under your application's control. The tree control forwards all messages for the control directly to your application. This control can be created in response to a mouse button click (or double-click), or any other reasonable event in your application. Any Windows control can be used (edit controls, combo boxes, etc.). TSftTree::GetEditColumnRect can be used to determine the proper location for the control.

This example shows how an application can respond to a left mouse-button double-click event by creating an edit control on the current item:

```
/* These routines handle item data editing. In this        */
/* example, an edit control is used. Any Windows control   */
/* can be used, even custom controls.                      */

    /* Add the following to your parent window class.          */
    int m_editIndex;                    /* Index of item being edited */
    int m_editCol;                      /* Column # being edited */
    TEdit* m_pEdit;                     /* Control used for editing */

    void EvStartEdit();                 /* Start item data editing */
    void EvQuitEdit();                  /* Abandon item data editing */
    void EvValidateEdit();              /* Validate input */
```

```
    /* Add the following to the parent window message map.     */
    EV_SFTTREEN_LBUTTONDBLCLK_TEXT(IDC_TREE, EvStartEdit),
    EV_SFTTREEN_QUITEDIT(IDC_TREE, EvQuitEdit),
    EV_SFTTREEN_VALIDATEEDIT(IDC_TREE, EvValidateEdit),

    /* Add the following to your parent window class           */
    /* constructor.                                            */
    m_pEdit = NULL;                      /* No item data editing */

/* Start editing in response to a                             */
/* SFTTREEN_LBUTTONDBLCLK_TEXT notification.                  */
void TYourDialog::EvStartEdit()
{
    char szBuffer[80];
    LPCRECT lpRect;

    /* Get the item index where double-click happened */
    m_editIndex = pTree->GetCaretIndex(); /* Save item # */
    m_editCol = -1;                      /* Let SftTree/DLL provide column # */
    lpRect = pTree->GetEditColumnRect(m_editIndex, &m_editCol);
    if (!lpRect)                         /* No column active */
        return;

    /* Make the column completely visible */
    pTree->MakeColumnVisible(m_editCol);
    /* Make the item completely visible */
    pTree->MakeRowVisible(m_editIndex);
    /* Get the location again (in case it scrolled) */
    lpRect = pTree->GetEditColumnRect(m_editIndex, &m_editCol);
    if (!lpRect)                         /* No column active */
        return;

    /* Change selection style to not shown anything */
    pTree->SetNoFocusStyle(SFTTREE_NOFOCUS_NOTHING);

    /* Create the edit control.*/
    /* Based on the tree control attributes and your preference, you may */
    /* have to adjust the rectangle used for the edit control. */
    m_pEdit = new TEdit(pTree, IDC_EDIT_your_id, TEXT(""),
         lpRect->left, lpRect->top, lpRect->right-lpRect->left, lpRect->bottom-
lpRect->top);
    m_pEdit->Attr.Style |= WS_CHILD|WS_BORDER|ES_AUTOHSCROLL|ES_LEFT; /* Styles */
    m_pEdit->Create();

    /* Set some edit control attributes */
    /* Set the font defined for item data editing */
    m_pEdit->SetWindowFont(*m_pEditFont, FALSE);
    /* Copy the text found in the tree control */
    pTree->GetString(szBuffer, m_editIndex, m_editCol);
    m_pEdit->SetWindowText(str);
    /* Select all text in the edit control and display it */
    m_pEdit->SetSelection(0, -1);
    m_pEdit->ShowWindow(SW_SHOW);
    m_pEdit->SetFocus();                 /* Set input focus to the control */
}
```

Once a tree control has one or more attached child windows, it generates the SFTTREEN_QUITEDIT and SFTTREEN_VALIDATEEDIT notifications, which signal the tree's parent window to abandon editing by destroying any associated controls, or to validate the input data, issue error messages and/or destroy the controls. When an application receives the SFTTREEN_QUITEDIT notification, it must unconditionally abort editing by destroying all child controls.

```
/* Quit editing in response to a SFTTREEN_QUITEDIT            */
/* notification.                                              */

void TYourDialog::EvQuitEdit()
{
```

```
    if (m_pEdit) {
        /* If the control has the focus, set the focus back to */
        /* tree control after destroying the control */
        BOOL fHadFocus = (GetFocus() == m_pEdit->HWindow);
        m_pEdit->Destroy();
        delete m_pEdit;
        m_pEdit = NULL;
        /* Restore nofocus display method */
        pTree->SetNoFocusStyle(SFTTREE_NOFOCUS_FRAME);
        if (fHadFocus)
            pTree->SetFocus();          /* Back to tree control */
    }
}

/* Validate edit data in response to a                    */
/* SFTTREEN_VALIDATEEDIT notification.                    */

void TYourDialog::EvValidateEdit()
{
    if (m_pEdit) {
        char szBuffer[80];
        /* Get the text from the edit control */
        m_pEdit->GetWindowText(szBuffer, sizeof(szBuffer));
        /* Validate the data */
        if (lstrcmp(TEXT(""), szBuffer) == 0) {
            MessageBox(TEXT("Just to demonstrate data input validation, ")
                       TEXT("this example rejects empty cells.  Please enter some
data."),
                       TEXT("SftTree/DLL"), MB_OK|MB_TASKMODAL|MB_ICONSTOP);
            m_pEdit->SetFocus();
        } else {
            /* Save the data in the tree control */
            pTree->SetString(szBuffer, m_editIndex, m_editCol);
            m_pEdit->Destroy();
            delete m_pEdit;
            m_pEdit = NULL;
            /* Restore nofocus display method */
            pTree->SetNoFocusStyle(SFTTREE_NOFOCUS_FRAME);
        }
    }
}
```

While editing item data using a control, the user may abort editing by pressing the ESCAPE key. This generates a SFTTREEN_QUITEDIT notification.

Additional Considerations

When creating controls for item data editing, a suitable font may have to be used. If a control is too small to display the data, the data may not only be clipped, but even be completely suppressed. This is particularly noticeable with edit controls.

If item data editing is started under program control during a WM_INITDIALOG message in the EvInitDialog member function or anytime the tree control has not yet been painted, the tree control has to be painted explicitly before a child control can be attached. This can be accomplished using the *UpdateWindow* call.

3D and Colors

WM_CTLCOLOR

This message is generated by the tree control for compatibility with SftTree 1.0 only. When developing new applications, please use TSftTree::SetCtlColor instead.

The appearance of the tree control can be modified by handling the WM_CTLCOLOR message. The parent window can override the default window colors used by the tree control by defining a

WM_CTLCOLOR message handler. Even though the tree control is not a list box, it generates WM_CTLCOLOR messages as a list box would.

```
// Pointer to tree object added to dialog class
TSftTree* pTree;

// Event handler definition added to dialog class
HBRUSH EvCtlColorTree (HDC dc, HWND hWndChild, uint ctlType);

// Event handler added to parent window response table
EV_WM_CTLCOLOR,

// Event handler implementation
HBRUSH TYourDialog::EvCtlColorTree(HDC dc, HWND hWndChild, uint ctlType)
{
    if (ctlType == CTLCOLOR_LISTBOX) {
        ::SetTextColor(dc, RGB(255,0,0));        // red text
        ::SetBkColor(dc, RGB(192,192,192));      // on gray background
        return (HBRUSH)::GetStockObject(LTGRAY_BRUSH);
    }
    return TDialog::EvCtlColor(dc, hWndChild, ctlType);
}
```

CTL3DV2 or CTL3D32

The tree control can be used with CTL3DV2 (or CTL3D32). More recent versions of CTL3DV2.DLL and CTL3D32.DLL allow user controls to receive a 3D border by using the Ctl3dSubclassCtlEx function.

```
::Ctl3dSubclassCtlEx(pTree->HWindow, CTL3D_LISTBOX_CTL);
```

This call is required even if Ctl3dAutoSubclass is used. For more information on CTL3DV2 or CTL3D32, see the documentation supplied with Borland C++.

When running under Windows 95 or Windows NT 4.0 and above, the window style WS_EX_CLIENTEDGE can be specified for the tree control, eliminating the need for CTL3DV2.DLL and CTL3D32.DLL.

Resource Workshop

Resource Workshop is part of Borland C++ 4.5. If you are using Borland C++ 5.0, see Using Borland C++.

First Time

In order to make SftTree/DLL available to Resource Workshop, use its *File, Install control library...* menu command to define SftTree/DLL to Resource Workshop. This has to be done only once.

Locate SftTree/DLL using the dialog shown. The file SFTTREE.DLL can be found in the directory C:\SFTTREE\BIN (unless changed during the installation). Do not install the 32-bit version of SftTree/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 SftTree/DLL, Resource Workshop will be able to display and modify SftTree/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 SftTree/DLL.

Once the DLL is installed by clicking the *OK* button in the dialog shown, the SftTree/DLL toolbar button is installed, and SftTree/DLL controls can be added to your dialogs just like a standard Windows control.

New Project

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

Adding a Tree Control to a Dialog

To add a SftTree/DLL control to a dialog, use the SftTree/DLL toolbar button. Click on the button and then on the dialog being designed to add a control. Once a SftTree/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. Only a few styles can be manipulated using this dialog. More styles are available through the C and C++ API. See the programming sections for more information.

SftTree Control Styles Dialog

The *SftTree/DLL Styles* dialog allows you to manipulate the following tree 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_TREE=201. In both cases the value 201 is assigned to the control as control ID, the second example also defines IDC_TREE 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.
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.
Notify Parent	Turn on the *Notify Parent* check box so the SftTree/DLL control will send WM_COMMAND messages to the parent window for special event notification. Equivalent to the SFTTREESTYLE_NOTIFY style.
Vertical Scroll Bar	The *Vertical Scroll Bar* check box adds a vertical scroll bar to

	the tree control. Equivalent to the WS_VSCROLL style.
Horizontal Scroll Bar	The *Horizontal Scroll Bar* check box adds a horizontal scroll bar to the tree control. Equivalent to the WS_HSCROLL style.
Disable No-Scroll	The *Disable No-Scroll* check box prevents the scroll bars from being hidden when scrolling is not possible. If this check box is on, the scroll bars are disabled when scrolling is not possible. Equivalent to the SFTTREESTYLE_DISABLENOSCROLL style.
Don't Add Scroll Bars Automatically	Turn on the *Don't Add Scroll Bars Automatically* check box to keep the original window styles WS_HSCROLL and WS_VSCROLL given when the tree control is created. Otherwise scroll bars are automatically added to the tree control when needed. E.g., to prevent a vertical scroll bar from being added to the tree control, select this option and do not select *Vertical Scroll Bar*. Equivalent to the SFTTREESTYLE_SCROLL style.
Want Keyboard Input	Turn on the *Want Keyboard Input* to have WM_VKEYTOITEM messages sent to the tree control's parent window for keyboard input processing. Equivalent to the SFTTREESTYLE_WANTKEYBOARDINPUT style.
Multiple Selection	Turn on the *Multiple Selection* check box to enable multiple tree items to be selected at the same time. The mouse, the SHIFT and CONTROL keys can be used to select multiple items. Equivalent to the SFTTREESTYLE_MULTIPLESEL style.
Drag & Drop	Turn on the *Drag & Drop* check box to enable drag & drop processing for the tree control. WM_COMMAND messages are then sent to the tree control's parent window for drag & drop processing. Equivalent to the SFTTREESTYLE_DRAGDROP style.
Left Button Only	Turn on the *Left Button Only* check box to ignore the middle and right mouse buttons. No notifications will be sent to the parent window when the middle or right mouse buttons are clicked. WM_CONTEXTMENU messages are generated instead on Windows 95, Windows NT 3.51 and above. Equivalent to the SFTTREESTYLE_LEFTBUTTONONLY 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.
Client Edge	Turn on the *Client Edge* check box to specify that the tree control has a border with a sunken edge (Windows 95, NT 4.0 and above only). This option is not available to 16-bit

	resource editors such as Resource Workshop and is disabled.
OK	Click the *OK* button to accept all style settings and end the *SftTree/DLL Styles* dialog.
Cancel	Click the *Cancel* button to abandon all (modified) style settings and end the *SftTree/DLL Styles* dialog.
Help	Click the *Help* button for on-line help information on the *SftTree/DLL Styles* dialog.
Design…	Click the *Design…* button to start the SftTree/DLL Wizard application to open or create a tree design and to generate the required C and C++ run-time initialization code for the tree control.

Test Mode

In the dialog test mode offered by Resource Workshop, a SftTree/DLL control will be displayed in the location specified with the styles selected. When using the tab key to test the tab stops, the SftTree/DLL control will display a caret when it receives the input focus.

Using Borland C++

Borland C++ 5.0 does not support custom control DLLs (Resource Workshop, shipped with Borland C++ 4.x fully supports custom controls). It is still possible to use SftTree/DLL 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 SftTree/DLL control, make sure to include the C and C++ header file SFTTREE.H. This insures that your resource definitions for the SftTree/DLL control can be compiled correctly. Add the following statement to your resource script:

```
#include "sfttree.h"                    // SftTree/DLL header file (for style
bits)
```

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

Adding a Tree Control to a Dialog

To add a SftTree/DLL control to a dialog, use the *Dialog, Insert New Control* menu command. Enter the class **SoftelTreeControl** (Windows 3.1) or **SoftelTreeControl32** (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.

SftTree Control Styles

To enter a SftTree/DLL 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

Style

Value	*Description*
SFTTREESTYLE_DISABLENOSCROLL	
0x00000001	Prevent the scroll bars from being hidden when scrolling is not possible. The scroll bars are disabled when scrolling is not possible.
SFTTREESTYLE_DRAGDROP	
0x00000010	Enable drag & drop processing for the tree control. WM_COMMAND messages are then sent to the tree control's parent window for drag & drop processing.
SFTTREESTYLE_LEFTBUTTONONLY	
0x00000020	When this style is selected, the tree control will ignore the middle and right mouse buttons. No notifications will be sent to the parent window when the middle or right mouse buttons are clicked. WM_CONTEXTMENU messages are generated instead on Windows 95, Windows NT 3.51 and above.
SFTTREESTYLE_MULTIPLESEL	
0x00000008	Enable multiple tree items to be selected at the same time. The mouse, the SHIFT and CONTROL keys can be used to select multiple items.
SFTTREESTYLE_NOTIFY	
0x00000004	The SftTree/DLL control will send WM_COMMAND messages to the parent window for special event notification.
SFTTREESTYLE_SCROLL	
0x00000040	When this style is selected, the window styles WS_HSCROLL and WS_VSCROLL given when the tree control is created determine if scroll

bars are present. If this style is not selected, scroll bars are automatically added to the tree control when needed. E.g., to prevent a vertical scroll bar from being added to the tree control, define the SFTTREESTYLE_SCROLL style and do not add the WS_VSCROLL style.

SFTTREESTYLE_WANTKEYBOARDINPUT
0x00000002 The SftTree/DLL control will send WM_VKEYTOITEM messages to the parent window for keyboard input processing.

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 tree control that is initially disabled. A disabled tree 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_HSCROLL
0x00100000 Add a horizontal scroll bar to the tree control.

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 tree control that is initially visible.

WS_VSCROLL
0x00200000 Add a vertical scroll bar to the tree control.

ExStyle

ExStyle

Value	*Description*
WS_EX_CLIENTEDGE	
0x00000200	Specifies that a window has a border with a sunken edge (Windows 95, NT 4.0 and above 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 (AppStudio only supports VBX controls). It is still possible to use SftTree/DLL with AppStudio or Visual 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 SftTree/DLL control, make sure to include the C and C++ header file SFTTREE.H. This insures that your resource definitions for the SftTree/DLL control can be compiled correctly. Add the following statement to your resource script:

```
#include "sfttree.h"                    // SftTree/DLL header file (for style
bits)
```

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

Adding a Tree Control to a Dialog

To add a SftTree/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 SftTree/DLL control, enter the class **SoftelTreeControl** (Windows 3.1) or **SoftelTreeControl32** (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.

SftTree Control Styles

To enter a SftTree/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*.

Style

Value	***Description***
SFTTREESTYLE_DISABLENOSCROLL	
0x00000001	Prevent the scroll bars from being hidden when scrolling is not possible. The scroll bars are disabled when scrolling is not possible.
SFTTREESTYLE_DRAGDROP	
0x00000010	Enable drag & drop processing for the tree control. WM_COMMAND messages are then sent to the tree control's parent window for drag & drop processing.
SFTTREESTYLE_LEFTBUTTONONLY	
0x00000020	When this style is selected, the tree control will ignore the middle and right mouse buttons. No notifications will be sent to the parent window when the middle or right mouse buttons are clicked. WM_CONTEXTMENU messages are generated instead on Windows 95, Windows NT 3.51 and above.
SFTTREESTYLE_MULTIPLESEL	
0x00000008	Enable multiple tree items to be selected at the same time. The mouse, the SHIFT and CONTROL keys can be used to select multiple items.
SFTTREESTYLE_NOTIFY	
0x00000004	The SftTree/DLL control will send WM_COMMAND messages to the parent window for special event notification.
SFTTREESTYLE_SCROLL	
0x00000040	When this style is selected, the window styles WS_HSCROLL and WS_VSCROLL given when the tree control is created determine if scroll bars are present. If this style is not selected, scroll bars are automatically added to the tree control when needed. E.g., to prevent a vertical scroll bar

from being added to the tree control, define the SFTTREESTYLE_SCROLL style and do not add the WS_VSCROLL style.

SFTTREESTYLE_WANTKEYBOARDINPUT

0x00000002 The SftTree/DLL control will send WM_VKEYTOITEM messages to the parent window for keyboard input processing.

WS_BORDER

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

WS_HSCROLL

0x00100000 Add a horizontal scroll bar to the tree control.

WS_VSCROLL

0x00200000 Add a vertical scroll bar to the tree control.

Test Mode

In the dialog test mode offered by AppStudio and Visual C++, the SftTree/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 SftTree/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 Windows NT.

First Time

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

Locate SftTree/DLL using the dialog shown. The DLL can be found in the directory C:\SFTTREE\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	SFTTREE.DLL
Windows NT (DLL w/o UNICODE support)	SFTTR32.DLL
Windows NT (DLL with UNICODE support)	SFTTR32U.DLL

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

Once the DLL is installed by clicking the *OK* button in the dialog, the SftTree/DLL control is installed and SftTree/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 SftTree/DLL control, make sure to include the C and C++ header file SFTTREE.H. This insures that your resource definitions for the SftTree/DLL control can be compiled correctly. Add the following statement to your resource script:

```
#include "sfttree.h"                    // SftTree/DLL header file (for style
bits)
```

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

Adding a Tree Control to a Dialog

To add a SftTree/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 SftTree/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 *Tools, Custom Control...* menu command. Only a few styles can be manipulated using this dialog. More styles are available through the C and C++ API. See the programming sections for more information.

SftTree Control Styles

The *SftTree/DLL Styles* dialog allows you to manipulate the following tree 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.
Notify Parent	Turn on the *Notify Parent* check box so the SftTree/DLL control will send WM_COMMAND messages to the parent window for special event notification. Equivalent to the SFTTREESTYLE_NOTIFY style.
Vertical Scroll Bar	The *Vertical Scroll Bar* check box adds a vertical scroll bar to the tree control. Equivalent to the WS_VSCROLL style.
Horizontal Scroll Bar	The *Horizontal Scroll Bar* check box adds a horizontal scroll bar to the tree control. Equivalent to the WS_HSCROLL style.
Disable No-Scroll	The *Disable No-Scroll* check box prevents the scroll bars from being hidden when scrolling is not possible. If this check box is on, the scroll bars are disabled when scrolling is not possible. Equivalent to the

SFTTREESTYLE_DISABLENOSCROLL style.

Don't Add Scroll Bars Automatically Turn on the *Don't Add Scroll Bars Automatically* check box to keep the original window styles WS_HSCROLL and WS_VSCROLL given when the tree control is created. Otherwise scroll bars are automatically added to the tree control when needed. E.g., to prevent a vertical scroll bar from being added to the tree control, select this option and do not select *Vertical Scroll Bar*. Equivalent to the SFTTREESTYLE_SCROLL style.

Want Keyboard Input Turn on the *Want Keyboard Input* to have WM_VKEYTOITEM messages sent to the tree control's parent window for keyboard input processing. Equivalent to the SFTTREESTYLE_WANTKEYBOARDINPUT style.

Multiple Selection Turn on the *Multiple Selection* check box to enable multiple tree items to be selected at the same time. The mouse, the SHIFT and CONTROL keys can be used to select multiple items. Equivalent to the SFTTREESTYLE_MULTIPLESEL style.

Drag & Drop Turn on the *Drag & Drop* check box to enable drag & drop processing for the tree control. WM_COMMAND messages are then sent to the tree control's parent window for drag & drop processing. Equivalent to the SFTTREESTYLE_DRAGDROP style.

Left Button Only Turn on the *Left Button Only* check box to ignore the middle and right mouse buttons. No notifications will be sent to the parent window when the middle or right mouse buttons are clicked. WM_CONTEXTMENU messages are generated instead on Windows 95, Windows NT 3.51 and above. Equivalent to the SFTTREESTYLE_LEFTBUTTONONLY 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.

Client Edge Turn on the *Client Edge* check box to specify that the tree control has a border with a sunken edge (Windows 95, NT 4.0 and above only). This option is not available to 16-bit resource editors and is disabled.

OK Click the *OK* button to accept all style settings and end the *SftTree/DLL Styles* dialog.

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

Help Click the *Help* button for on-line help information on the *SftTree/DLL Styles* dialog.

Design… Click the *Design…* button to start the SftTree/DLL Wizard

application to open or create a tree design and to generate the required C and C++ run-time initialization code for the tree control.

Test Mode

In the dialog test mode offered by the dialog editor, a SftTree/DLL control will be displayed in the location specified with the styles selected. When using the tab key to test the tab stops, the SftTree/DLL control will display a caret when it receives the input focus.

Tree Styles

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

SFTTREESTYLE_DISABLENOSCROLL (0x0001L)
When this style is selected, the scroll bars are disabled when scrolling is not possible. Without this style, scroll bars are hidden when scrolling is not possible. This style should not be changed after the tree control is created. Use SetDisableNoScroll instead.

SFTTREESTYLE_DRAGDROP (0x0010L)
When this style is selected, the tree control allows the user to drag & drop items within and outside the tree control. It is up to the control's parent to handle the notification messages to process a drag & drop operation and to set appropriate mouse cursors. This style can be changed using SetWindowLong(hwnd, GWL_STYLE, style), as long as no drag & drop operation is in progress.

SFTTREESTYLE_LEFTBUTTONONLY (0x0020L)
When this style is selected, the tree control will ignore the middle and right mouse buttons. No notifications will be sent to the parent window when the middle or right mouse buttons are clicked. WM_CONTEXTMENU messages are generated instead on Windows 95, Windows NT 3.51 and above.

SFTTREESTYLE_MULTIPLESEL (0x0008L)
When this style is selected, the tree control will allow multiple items to be selected at the same time. One or more items can be selected using the mouse. Using the CONTROL key causes additional items to be selected without removing previous selections. Using the SHIFT key causes ranges of items to be selected, starting at the last position. Without this style, only one item can be selected at a time and the CONTROL and SHIFT keys have no effect whatsoever. This style can be changed using SetWindowLong(hwnd, GWL_STYLE, style), however all selections have to be cleared first.

SFTTREESTYLE_NOTIFY (0x0004L)
When this style is selected, the tree control will send WM_COMMAND messages to the parent window for event notification. This style can be changed using SetWindowLong(hwnd, GWL_STYLE, style).

SFTTREESTYLE_SCROLL (0x0040L)
When this style is selected, the window styles WS_HSCROLL and WS_VSCROLL given when the tree control is created, determine if scroll bars are present. If this style is not selected, scroll bars are automatically added to the tree control when needed. E.g., to prevent a vertical scroll bar from being added to the tree control, define the SFTTREESTYLE_SCROLL style and do not add the WS_VSCROLL style.

SFTTREESTYLE_WANTKEYBOARDINPUT (0x0002L)
When this style is selected, the tree control will send WM_VKEYTOITEM messages to the parent window for keyboard input processing. Without this style, the parent window will not receive WM_VKEYTOITEM messages from the tree control. This style can be changed using SetWindowLong(hwnd, GWL_STYLE, style).

Notifications

The parent window of a tree 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

SFTTREEN_BEGINDRAG
The user is starting to drag one or several items. All items currently selected participate in the drag operation. An application can set the mouse cursor in response to this event or cancel the drag operation. The drag operation can be aborted by sending a WM_CANCELMODE message to the tree control or by clearing all currently selected items. The drag operation is described by the SFTTREE_DRAGINFO structure. The SFTTREESTYLE_DRAGDROP style has to be defined to receive this notification.

SFTTREEN_CANCELDRAG
The user canceled the drag operation. The SFTTREESTYLE_DRAGDROP style has to be defined to receive this notification.

SFTTREEN_CARETCHANGE
The caret location has changed. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification.

SFTTREEN_COLUMNSIZE
The user resized the columns. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification.

SFTTREEN_DRAGGING
The user is moving the mouse cursor to drag one or several items. All items currently selected participate in the drag operation. The drag operation is described by the SFTTREE_DRAGINFO structure. An application can set the mouse cursor in response to this event. The SFTTREESTYLE_DRAGDROP style has to be defined to receive this notification.

SFTTREEN_ENDDRAG
The user released the mouse button. All items currently selected participate in the drag operation. The drag operation is described by the SFTTREE_DRAGINFO structure. An application should process the dropped items in response to this event. The SFTTREESTYLE_DRAGDROP style has to be defined to receive this notification.

SFTTREEN_KILLFOCUS
The tree control lost the input focus.

SFTTREEN_LBUTTONDBLCLK_BUTTON
The left mouse button was double-clicked on the expand/collapse button of the current entry (caret location). Use GetCaretIndex to determine the item index where the mouse button was clicked. Use SetItemExpand to expand/collapse the item. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification.

SFTTREEN_LBUTTONDBLCLK_ITEM
The left mouse button was double-clicked on the item bitmap of the current entry (caret location). The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification.

SFTTREEN_LBUTTONDBLCLK_LABEL

The left mouse button was double-clicked on the label bitmap. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification.

SFTTREEN_LBUTTONDBLCLK_PLUSMIN

The left mouse button was double-clicked on the plus/minus bitmap of the current entry (caret location). The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification.

SFTTREEN_LBUTTONDBLCLK_TEXT

The left mouse button was double-clicked on the text of the current entry (caret location) or the RETURN key was pressed. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification for mouse button clicks. It is always sent for the RETURN key.

SFTTREEN_LBUTTONDBLCLK_TREE

The left mouse button was double-clicked in the area of the current entry (caret location) where the connecting tree lines are drawn. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification.

SFTTREEN_LBUTTONDOWN_BUTTON

The left mouse button was clicked on the expand/collapse button of the current entry (caret location). Use GetCaretIndex to determine the item index where the mouse button was clicked. Use SetItemExpand to expand/collapse the item. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification.

SFTTREEN_LBUTTONDOWN_COLUMN

The left mouse button was clicked on the column header. Use GetHeaderButton to determine the header button pressed and use SetHeaderButton to reset the button (if desired). The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification.

SFTTREEN_LBUTTONDOWN_ITEM

The left mouse button was clicked on the item bitmap of the current entry (caret location). The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification.

SFTTREEN_LBUTTONDOWN_LABEL

The left mouse button was clicked on the label bitmap. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification.

SFTTREEN_LBUTTONDOWN_PLUSMIN

The left mouse button was clicked on the plus/minus bitmap of the current entry (caret location). The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification.

SFTTREEN_LBUTTONDOWN_ROW

The left mouse button was clicked on the row header of the current item. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification.

SFTTREEN_LBUTTONDOWN_ROWCOLUMN

The left mouse button was clicked on the row/column header. Use the SftTree_SetRowColHeaderButton function to reset the button (if desired). The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification.

SFTTREEN_LBUTTONDOWN_TEXT

The left mouse button was clicked on the text (any column) of the current entry (caret location). The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification.

SFTTREEN_LBUTTONDOWN_TREE

The left mouse button was clicked in the area of the current entry (caret location) where the connecting tree lines are drawn. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification.

SFTTREEN_MBUTTONDBLCLK_BUTTON

The middle mouse button was double-clicked on the expand/collapse button of the current entry (caret location). Use GetCaretIndex to determine the item index where the mouse button was clicked. Use SetItemExpand to expand/collapse the item. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. This notification is not generated if the tree control has the

SFTTREESTYLE_LEFTBUTTONONLY style.

SFTTREEN_MBUTTONDBLCLK_ITEM

The middle mouse button was double-clicked on the item bitmap of the current entry (caret location). The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. This notification is not generated if the tree control has the SFTTREESTYLE_LEFTBUTTONONLY style.

SFTTREEN_MBUTTONDBLCLK_LABEL

The middle mouse button was double-clicked on the label bitmap. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. This notification is not generated if the tree control has the SFTTREESTYLE_LEFTBUTTONONLY style.

SFTTREEN_MBUTTONDBLCLK_PLUSMIN

The middle mouse button was double-clicked on the plus/minus bitmap of the current entry (caret location). The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. This notification is not generated if the tree control has the SFTTREESTYLE_LEFTBUTTONONLY style.

SFTTREEN_MBUTTONDBLCLK_TEXT

The middle mouse button was double-clicked on the text of the current entry (caret location) or the RETURN key was pressed. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. This notification is not generated if the tree control has the SFTTREESTYLE_LEFTBUTTONONLY style.

SFTTREEN_MBUTTONDBLCLK_TREE

The middle mouse button was double-clicked in the area of the current entry (caret location) where the connecting tree lines are drawn. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. This notification is not generated if the tree control has the SFTTREESTYLE_LEFTBUTTONONLY style.

SFTTREEN_MBUTTONDOWN_BUTTON

The middle mouse button was clicked on the expand/collapse button of the current entry (caret location). Use GetCaretIndex to determine the item index where the mouse button was clicked. Use SetItemExpand to expand/collapse the item. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. This notification is not generated if the tree control has the SFTTREESTYLE_LEFTBUTTONONLY style.

SFTTREEN_MBUTTONDOWN_COLUMN

The middle mouse button was clicked on the column header. Use GetHeaderButton to determine the header button pressed and use SetHeaderButton to reset the button (if desired). The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. This notification is not generated if the tree control has the SFTTREESTYLE_LEFTBUTTONONLY style.

SFTTREEN_MBUTTONDOWN_ITEM

The middle mouse button was clicked on the item bitmap of the current entry (caret location). The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. This notification is not generated if the tree control has the SFTTREESTYLE_LEFTBUTTONONLY style.

SFTTREEN_MBUTTONDOWN_LABEL

The middle mouse button was clicked on the label bitmap. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. This notification is not generated if the tree control has the SFTTREESTYLE_LEFTBUTTONONLY style.

SFTTREEN_MBUTTONDOWN_PLUSMIN

The middle mouse button was clicked on the plus/minus bitmap of the current entry (caret location). The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. This notification is not generated if the tree control has the SFTTREESTYLE_LEFTBUTTONONLY style.

SFTTREEN_MBUTTONDOWN_ROW

The middle mouse button was clicked on the row header of the current item. The

SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. This notification is not generated if the tree control has the SFTTREESTYLE_LEFTBUTTONONLY style.

SFTTREEN_MBUTTONDOWN_ROWCOLUMN

The middle mouse button was clicked on the row/column header. Use the SftTree_SetRowColHeaderButton function to reset the button (if desired). The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. This notification is not generated if the tree control has the SFTTREESTYLE_LEFTBUTTONONLY style.

SFTTREEN_MBUTTONDOWN_TEXT

The middle mouse button was clicked on the text (any column) of the current entry (caret location). The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. This notification is not generated if the tree control has the SFTTREESTYLE_LEFTBUTTONONLY style.

SFTTREEN_MBUTTONDOWN_TREE

The middle mouse button was clicked in the area of the current entry (caret location) where the connecting tree lines are drawn. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. This notification is not generated if the tree control has the SFTTREESTYLE_LEFTBUTTONONLY style.

SFTTREEN_MOUSEMOVE

The tree control received a WM_MOUSEMOVE message.

SFTTREEN_OFFSETCHANGE

The horizontal scrolling offset has changed. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. While processing this notification, no tree control and item attributes should be altered.

SFTTREEN_OVERHEADCHANGED

Due to the variable number of levels and the resulting hierarchical display, the width of the first column is always treated as a minimum width. The text portion of the first column will always be at least of the width specified using SetColumns, no matter what level the item is on. This can result in the first column being much wider than the defined width. If more levels are added to a hierarchy, the value returned by GetOverheadWidth increases. When the overhead width changes, this notification is sent to the tree control's parent window. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification.

SFTTREEN_QUITEDIT

This notification is used to notify the tree control's parent, that any editing of tree item data should be abandoned now. No data validation should take place and no modifications should be made to the tree control. This notification is only sent if the tree control currently has a child window. The tree control generates this notification, when the size or position of the child window is changing, when modifications to the tree control occur or when a menu or menu selection is about to become active. The parent should destroy all controls associated with item data editing. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification.

SFTTREEN_RBUTTONDBLCLK_BUTTON

The right mouse button was double-clicked on the expand/collapse button of the current entry (caret location). Use GetCaretIndex to determine the item index where the mouse button was clicked. Use SetItemExpand to expand/collapse the item. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. This notification is not generated if the tree control has the SFTTREESTYLE_LEFTBUTTONONLY style.

SFTTREEN_RBUTTONDBLCLK_ITEM

The right mouse button was double-clicked on the item bitmap of the current entry (caret location). The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. This notification is not generated if the tree control has the SFTTREESTYLE_LEFTBUTTONONLY style.

SFTTREEN_RBUTTONDBLCLK_LABEL

The right mouse button was double-clicked on the label bitmap. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. This notification is not generated if the tree control has the SFTTREESTYLE_LEFTBUTTONONLY style.

SFTTREEN_RBUTTONDBLCLK_PLUSMIN

The right mouse button was double-clicked on the plus/minus bitmap of the current entry (caret location). The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. This notification is not generated if the tree control has the SFTTREESTYLE_LEFTBUTTONONLY style.

SFTTREEN_RBUTTONDBLCLK_TEXT

The right mouse button was double-clicked on the text of the current entry (caret location) or the RETURN key was pressed. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. This notification is not generated if the tree control has the SFTTREESTYLE_LEFTBUTTONONLY style.

SFTTREEN_RBUTTONDBLCLK_TREE

The right mouse button was double-clicked in the area of the current entry (caret location) where the connecting tree lines are drawn. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. This notification is not generated if the tree control has the SFTTREESTYLE_LEFTBUTTONONLY style.

SFTTREEN_RBUTTONDOWN_BUTTON

The right mouse button was clicked on the expand/collapse button of the current entry (caret location). Use GetCaretIndex to determine the item index where the mouse button was clicked. Use SetItemExpand to expand/collapse the item. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. This notification is not generated if the tree control has the SFTTREESTYLE_LEFTBUTTONONLY style.

SFTTREEN_RBUTTONDOWN_COLUMN

The right mouse button was clicked on the column header. Use GetHeaderButton to determine the header button pressed and use SetHeaderButton to reset the button (if desired). The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. This notification is not generated if the tree control has the SFTTREESTYLE_LEFTBUTTONONLY style.

SFTTREEN_RBUTTONDOWN_ITEM

The right mouse button was clicked on the item bitmap of the current entry (caret location). The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. This notification is not generated if the tree control has the SFTTREESTYLE_LEFTBUTTONONLY style.

SFTTREEN_RBUTTONDOWN_LABEL

The right mouse button was clicked on the label bitmap. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. This notification is not generated if the tree control has the SFTTREESTYLE_LEFTBUTTONONLY style.

SFTTREEN_RBUTTONDOWN_PLUSMIN

The right mouse button was clicked on the plus/minus bitmap of the current entry (caret location). The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. This notification is not generated if the tree control has the SFTTREESTYLE_LEFTBUTTONONLY style.

SFTTREEN_RBUTTONDOWN_ROW

The right mouse button was clicked on the row header of the current item. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. This notification is not generated if the tree control has the SFTTREESTYLE_LEFTBUTTONONLY style.

SFTTREEN_RBUTTONDOWN_ROWCOLUMN

The right mouse button was clicked on the row/column header. Use the SftTree_SetRowColHeaderButton function to reset the button (if desired). The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. This notification is not generated if the tree control has the

SFTTREESTYLE_LEFTBUTTONONLY style.

SFTTREEN_RBUTTONDOWN_TEXT

The right mouse button was clicked on the text (any column) of the current entry (caret location). The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. This notification is not generated if the tree control has the SFTTREESTYLE_LEFTBUTTONONLY style.

SFTTREEN_RBUTTONDOWN_TREE

The right mouse button was clicked in the area of the current entry (caret location) where the connecting tree lines are drawn. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification. This notification is not generated if the tree control has the SFTTREESTYLE_LEFTBUTTONONLY style.

SFTTREEN_SELCANCEL

The user canceled a selection. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification.

SFTTREEN_SELCHANGE

The user has changed the current selection. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification.

SFTTREEN_SETFOCUS

The tree control received the input focus.

SFTTREEN_TOPCHANGE

The first item displayed by the control has changed. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification.

SFTTREEN_VALIDATEEDIT

If item data is being edited, the parent should now validate the data entered. The parent can display an error message and then set the input focus back to the control used for item data editing. In this case, the event that caused input validation to occur, is ignored. If the parent accepts the input data, the child control(s) should now be destroyed, the tree control updated and the input focus set to the tree control. If the child control is not destroyed, the tree control assumes that input validation failed and item data editing continues. This notification is only sent if the tree control currently has a child window. The tree control generates this notification, when the user moves away from the child control using the mouse buttons. The SFTTREESTYLE_NOTIFY style has to be defined to receive this notification.

MFC and Notifications

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

A tree control will only send notification messages if its window style includes the SFTTREESTYLE_NOTIFY style.

Parent Window

If you want to handle Windows notification messages sent by a tree 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 class
afx_msg void OnLButtonExpandCollapse();

// Event handler(s) added to message map
BEGIN_MESSAGE_MAP(CYourDialog, CDialog)
    ON_SFTTREEN_LBUTTONDOWN_BUTTON(IDC_TREE, OnLButtonExpandCollapse)
    ON_SFTTREEN_LBUTTONDBLCLK_BUTTON(IDC_TREE, OnLButtonExpandCollapse)
    ON_SFTTREEN_LBUTTONDBLCLK_TEXT(IDC_TREE, OnLButtonExpandCollapse)
END_MESSAGE_MAP()

// event handler implementation
void CYourDialog::OnLButtonExpandCollapse()
{
    // get index of item to expand/collapse
    int index = m_Tree.GetCaretIndex();
    // get current expand/collapsed status
    BOOL fExpanded = m_Tree.GetItemExpand(index);
    // if control key is used we'll expand all dependents
    BOOL fDepth = (::GetKeyState(VK_CONTROL)&0x8000);
    m_Tree.SetItemExpand(index, !fExpanded, fDepth);
}
```

Derived Objects

By overriding the OnChildNotify function of an object derived from CSftTree, you can handle messages in the object's class. 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. SftTree/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. *memberFxn* is the name of the member function in your object's class which handles the notification.

```
BEGIN_MESSAGE_MAP(CYourTree, CSftTree)
    //{{AFX_MSG_MAP(CYourTree)
    ON_WM_CREATE()
    ON_SFTTREEN_SELCHANGE_REFLECT(OnSelChangeReflect)
    ON_SFTTREEN_CARETCHANGE_REFLECT(OnCaretChangeReflect)
END_MESSAGE_MAP()
```

OWL and Notifications

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

A tree control will only send notification messages if its window style includes the SFTTREESTYLE_NOTIFY style.

Parent Window

If you want to handle Windows notification messages sent by a tree 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:

```
// Pointer to tree object added to dialog class
TSftTree* pTree;

//  Event handler prototype added to dialog class
void EvLButtonExpandCollapse();

// Allocate tree object in dialog constructor
pTree = new TSftTree(this, IDC_TREE);

// response table
DEFINE_RESPONSE_TABLE1(TYourDialog, TDialog)
  EV_SFTTREEN_LBUTTONDOWN_BUTTON(IDC_TREE, EvLButtonExpandCollapse),
  EV_SFTTREEN_LBUTTONDBLCLK_BUTTON(IDC_TREE, EvLButtonExpandCollapse),
  EV_SFTTREEN_LBUTTONDBLCLK_TEXT(IDC_TREE, EvLButtonExpandCollapse),
END_RESPONSE_TABLE;

// event handler implementation
void TYourDialog::EvLButtonExpandCollapse()
{
    // get index of item to expand/collapse
    int index = pTree->GetCaretIndex();
    // get current expand/collapsed status
    BOOL fExpanded = pTree->GetItemExpand(index);
    // if control key is used we'll expand all dependents
    BOOL fDepth = (::GetKeyState(VK_CONTROL)&0x8000);

    pTree->SetItemExpand(index, !fExpanded, fDepth);
}
```

Derived Objects

OWL defines the EV_NOTIFY_AT_CHILD macro which allows adding notifications directly to the response table of an object derived from TSftTree. SftTree/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. *memberFxn* is the name of the member function in your object's class which handles the notification.

```
// response table
DEFINE_RESPONSE_TABLE1(TYourTree, TSftTree)
    EV_SFTTREEN_SELCHANGE_AT_CHILD(EvSelChangeAtChild)
    EV_SFTTREEN_CARETCHANGE_AT_CHILD(EvCaretChangeAtChild)
END_RESPONSE_TABLE;
```

Parent Messages

The parent window of a tree control can receive the following messages:

WM_CONTEXTMENU

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

Parameters:

hwnd = (HWND) wParam;
Window handle of the tree 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

This message is only generated if the right and middle mouse button are not handled by the tree control (see SFTTREESTYLE_LEFTBUTTONONLY).

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_VKEYTOITEM

The WM_VKEYTOITEM message is sent by a tree control with the SFTTREESTYLE_WANTKEYBOARDINPUT style to its parent in response to a WM_KEYDOWN message.

Parameters, WIN16:

wVkey = wParam
The virtual-key code of the key that the user pressed.
hwnd = (HWND) LOWORD(lParam)
Window handle of the tree control.
nCaretPos = HIWORD(lParam)
Caret location.

Parameters, WIN32:

wVkey = LOWORD(wParam)
The virtual-key code of the key that the user pressed.
hwnd = (HWND) lParam
Window handle of the tree control.
nCaretPos = HIWORD(wParam)
Caret location (if more than 32K items are added to the tree control, use GetCaretIndex to retrieve a valid caret location).

Returns

The return value specifies the action that the application performed in response to the message. A return value of -2 indicates that the application handled all aspects of selecting the item and requires no further action by the tree control. A return value of -1 indicates that the tree control should perform the default action in response to the keystroke. A return value of 0 or greater specifies the zero-based index of an item in the tree control and indicates that the tree control should perform the default action for the keystroke on the given item.

Comments

A tree control will only support WM_VKEYTOITEM messages if its window style includes the

SFTTREESTYLE_WANTKEYBOARDINPUT style.

C, Functions

The SendMessage Windows API function can be used to send messages to a tree control window. Macro definitions are provided for easy use of SftTree/DLL messages, so parameter packing is transparent to the developer.

Function/Macro	*Description*
SftTree_AddString	Adds a new item at the end of the tree control at level 0 using a string.
SftTree_AddString_A	Adds a new item at the end of the tree control at level 0 using an ANSI string.
SftTree_AddString_W	Adds a new item at the end of the tree control at level 0 using a UNICODE string.
SftTree_CalcIndexFromPoint	Calculates the item index number given a point in tree control client area coordinates.
SftTree_CalcIndexFromPointEx	Calculates the item index number given a point in tree control client area coordinates.
SftTree_CalcOptimalColumnWidth	Calculates a column's optimal width so text and bitmaps are not clipped.
SftTree_CalcOptimalRowHeaderWidth	Calculates the optimal width for row headers so text and bitmaps are not clipped.
SftTree_CopyItem	Copies an item to a new location.
SftTree_DeleteDependents	Deletes an item's dependent items.
SftTree_DeleteString	Deletes an item.
SftTree_FindItem	Searches item data.
SftTree_FindString	Searches a string.
SftTree_FindString_A	Searches an ANSI string.
SftTree_FindString_W	Searches a UNICODE string.
SftTree_FindStringCol	Searches a string.
SftTree_FindStringColExact	Searches a string.
SftTree_FindStringExact	Searches a string.
SftTree_FindStringExact_A	Searches an ANSI string.
SftTree_FindStringExact_W	Searches a UNICODE string.
SftTree_GetAccessColumn	Returns the current column accessed by column specific messages/functions.
SftTree_GetCaretColumn	Returns the column where a mouse button was last clicked.
SftTree_GetCaretIndex	Returns the current item (caret location).
SftTree_GetCellInfo	Returns a cell's attributes.
SftTree_GetColumns	Returns column information and the number of columns defined.
SftTree_GetColumnsEx	Returns column information and the number of columns defined.
SftTree_GetCount	Returns the number of items in the tree control.
SftTree_GetCtlColors	Returns the tree control's color attributes.
SftTree_GetCurSel	Returns the index of the selected item (single selection tree control).
SftTree_GetDependent	Returns dependent item information for a parent item.
SftTree_GetDependentCount	Returns the number of dependents for a parent item.
SftTree_GetDisableNoScroll	Returns the scroll bar handling status.
SftTree_GetDragBitmaps	Returns the drag & drop starting location attribute.
SftTree_GetDragInfo	Returns drag & drop operation information in a SFTTREE_DRAGINFO structure.
SftTree_GetDragType	Returns the drag & drop detection attribute.
SftTree_GetDropHighlight	Returns the current drag & drop target location.
SftTree_GetDropHighlightStyle	Returns the display attribute of the current drag & drop target item.
SftTree_GetEditColumnRect	Returns an item's column location in client area coordinates.

SftTree_GetForwardChildMsgs	Returns the child window message handling status.
SftTree_GetGridStyle	Returns the grid line display style.
SftTree_GetHeader	Returns the current column's header text.
SftTree_GetHeader_A	Returns the current column's header text as an ANSI string.
SftTree_GetHeader_W	Returns the current column's header text as a UNICODE string.
SftTree_GetHeaderButton	Returns the column number of the currently pressed header button.
SftTree_GetHeaderCol	Returns a column's header text.
SftTree_GetHeaderColLength	Returns the length of a column's header text.
SftTree_GetHeaderFont	Returns the font used for header text display.
SftTree_GetHeaderLength	Returns the length of a column's header text.
SftTree_GetHeaderRect	Returns the dimensions of the header area.
SftTree_GetHorizontalExtent	Returns the last defined horizontal extent (in pixels) of the displayable area.
SftTree_GetHorizontalOffset	Returns the current horizontal offset (in pixels) of the displayed area.
SftTree_GetItemBitmap	Returns an item's item bitmap.
SftTree_GetItemData	Returns an item's application-specific 32-bit data value.
SftTree_GetItemExpand	Returns an item's expand status.
SftTree_GetItemHeight	Returns the current height (in pixels) of each item.
SftTree_GetItemLabel	Returns an item's label bitmap information.
SftTree_GetItemLevel	Returns an item's level number.
SftTree_GetItemLines	Returns the number of text lines used for item height calculation.
SftTree_GetItemRect	Returns an item's location in client area coordinates.
SftTree_GetItemShown	Returns an item's visibility status.
SftTree_GetItemsShown	Returns the number of items displayable in the tree control's client area, including partial items.
SftTree_GetItemsShownComplete	Returns the number of items displayable in the tree control's client area, not including partial items.
SftTree_GetItemStatus	Returns an item's enabled/disabled status.
SftTree_GetNoFocusStyle	Returns the display style of selected items when the tree control does not have the input focus.
SftTree_GetOverheadWidth	Returns the width of the area added to the first column for hierarchical graphics components.
SftTree_GetParent	Returns an item's parent index.
SftTree_GetResizeHeader	Returns a value indicating if columns are resizable by the user.
SftTree_GetRowColBitmap	Returns the bitmap displayed in the row/column header.
SftTree_GetRowColBitmapStyle	Returns the position of the bitmap displayed in the row/column header.
SftTree_GetRowColHeaderButton	Returns a value indicating if the row/column header button is currently down (pressed).
SftTree_GetRowColHeaderRect	Returns the dimensions of the row/column header area.
SftTree_GetRowColHeaderStyle	Returns the position of the text displayed in the row/column header.
SftTree_GetRowColText	Returns the row/column header text.
SftTree_GetRowColText_A	Returns the row/column header text as an ANSI string.
SftTree_GetRowColText_W	Returns the row/column header text as a UNICODE string.
SftTree_GetRowColTextLength	Returns the row/column header text length.
SftTree_GetRowHeaderFont	Returns the font used for row header text display.
SftTree_GetRowHeaderRect	Returns the dimensions of the row header area.
SftTree_GetRowHeaderStyle	Returns the position of text displayed in the row headers.
SftTree_GetRowHeaderWidth	Returns the width of the row header area.
SftTree_GetRowInfo	Returns an item's row header attributes.

SftTree_GetRowText	Returns an item's row header text.
SftTree_GetRowText_A	Returns an item's row header text as an ANSI string.
SftTree_GetRowText_W	Returns an item's row header text as a UNICODE string.
SftTree_GetRowTextLength	Returns an item's row header text length.
SftTree_GetSel	Returns the selection status of an item (multiple selection tree only).
SftTree_GetSelCount	Returns the number of currently selected items (multiple selection tree only).
SftTree_GetSelItems	Fills an array with the index numbers of currently selected items (multiple selection tree only).
SftTree_GetSelTextOnly	Returns a value indicating the display style of selected items.
SftTree_GetShow3D	Returns a value indicating the current display method used for items.
SftTree_GetShowBitmaps	Returns a value indicating the presence of item bitmaps.
SftTree_GetShowButton0	Returns a value indicating the presence of level 0 expand/collapse buttons.
SftTree_GetShowButtons	Returns a value indicating the presence of expand/collapse buttons (other than level 0).
SftTree_GetShowGrid	Returns a value indicating the presence of grid lines.
SftTree_GetShowHeader	Returns a value indicating the presence of column headers.
SftTree_GetShowHeaderButtons	Returns a value indicating the presence of column header buttons.
SftTree_GetShowLabels	Returns a value indicating the presence of label bitmaps.
SftTree_GetShowPlusMinus	Returns a value indicating the presence of plus/minus bitmaps.
SftTree_GetShowRowColHeaderButton	Returns the row/column header's display style.
SftTree_GetShowRowHeader	Returns the row header display style.
SftTree_GetShowTreeLines	Returns a value indicating the presence of tree lines.
SftTree_GetShowTruncated	Returns a value that determines whether text is clipped or truncated using trailing "...".
SftTree_GetSibling	Returns an item's sibling information.
SftTree_GetTabKeyIntercept	Returns the status of TAB key handling while editing item data.
SftTree_GetText	Returns an item's text.
SftTree_GetText_A	Returns an item's text as an ANSI string.
SftTree_GetText_W	Returns an item's text as a UNICODE string.
SftTree_GetTextCol	Returns an item's text.
SftTree_GetTextColLength	Returns the length of an item's text.
SftTree_GetTextLength	Returns the length of an item's text.
SftTree_GetTopIndex	Returns the index number of the item currently shown at the top of the tree control.
SftTree_GetTopParent	Returns the highest level parent index number for an item.
SftTree_InsertString	Inserts a new item using a string.
SftTree_InsertString_A	Inserts a new item using an ANSI string.
SftTree_InsertString_W	Inserts a new item using a UNICODE string.
SftTree_MakeColumnOptimal	Sets the optimal column width so that the text and bitmaps of all items can be displayed.
SftTree_MakeColumnVisible	Horizontally scrolls the specified column into view so that it is displayed in the tree control's client area.
SftTree_MakeIntegralHeight	Vertically resizes the tree control so visible items are displayed in their entirety.
SftTree_MakeRowHeaderOptimal	Sets the optimal row header width so that the text and bitmaps of all row headers can be displayed.
SftTree_MakeRowVisible	Vertically scrolls the specified item into view so that it is displayed in the tree control's client area.
SftTree_MoveItem	Moves an item to a new position in the tree control.

SftTree_RecalcHorizontalExtent	Recalculates the optimal horizontal scrolling extent.
SftTree_RegisterApp2	Registers an application with SftTree/DLL.
SftTree_ResetContent	Removes all items from the tree control.
SftTree_SelItemRange	Selects or deselects a range of items (multiple selection tree only).
SftTree_SetAccessColumn	Sets the current column accessed by column specific messages/functions.
SftTree_SetBitmaps	Registers the size and sets the default item bitmaps used for all items.
SftTree_SetButtons	Registers the size and sets the bitmaps used for expand/collapse buttons.
SftTree_SetCaretIndex	Sets the current item (caret location).
SftTree_SetCellInfo	Sets a cell's attributes.
SftTree_SetColumns	Sets the number of columns and their attributes.
SftTree_SetColumnsEx	Sets the number of columns and their attributes.
SftTree_SetCtlColors	Sets the tree control's color attributes.
SftTree_SetCurSel	Sets the currently selected item (single selection tree only).
SftTree_SetDeleteCallback	Sets a deletion callback routine, called when items are deleted.
SftTree_SetDisableNoScroll	Sets the scroll bar handling status.
SftTree_SetDragBitmaps	Sets the drag & drop starting location attribute.
SftTree_SetDragType	Sets the drag & drop detection attribute.
SftTree_SetDrawInfoCallback	Sets a drawing callback routine. Applications can override certain drawing defaults using this callback.
SftTree_SetDropHighlight	Sets the current drag & drop target location.
SftTree_SetDropHighlightStyle	Sets the display attribute of the current drag & drop target item.
SftTree_SetForwardChildMsgs	Sets the child window message handling status.
SftTree_SetGridStyle	Sets the grid line display style.
SftTree_SetHeader	Sets the current column's header text.
SftTree_SetHeader_A	Sets the current column's header text using an ANSI string.
SftTree_SetHeader_W	Sets the current column's header text using a UNICODE string.
SftTree_SetHeaderButton	Sets the column number of the currently pressed header button.
SftTree_SetHeaderCol	Sets a column's header text.
SftTree_SetHeaderFont	Sets the font used for header text display.
SftTree_SetHorizontalExtent	Sets the horizontal extent (in pixels) of the displayable area.
SftTree_SetHorizontalOffset	Sets the horizontal offset (in pixels) of the displayed area.
SftTree_SetItemBitmap	Sets an item's item bitmap.
SftTree_SetItemData	Sets an item's application-specific 32-bit data value.
SftTree_SetItemExpand	Sets an item's expand status.
SftTree_SetItemLabel	Sets an item's label bitmap information.
SftTree_SetItemLevel	Sets an item's level number.
SftTree_SetItemLines	Sets the number of text lines used for item height calculation.
SftTree_SetItemShown	Sets an item's visibility status.
SftTree_SetItemStatus	Sets an item's enabled/disabled status.
SftTree_SetNoFocusStyle	Sets the display style of selected items when the tree control does not have the input focus.
SftTree_SetPlusMinus	Set the plus/minus bitmaps used for all items.
SftTree_SetResizeHeader	Sets a value indicating if columns are resizable by the user.
SftTree_SetRowColBitmap	Sets the bitmap displayed in the row/column header.
SftTree_SetRowColBitmapStyle	Sets the position of the bitmap displayed in the row/column header.
SftTree_SetRowColHeaderButton	Sets a value indicating if the row/column header button is currently down (pressed).

SftTree_SetRowColHeaderStyle	Sets the position of the text displayed in the row/column header.
SftTree_SetRowColText	Sets the row/column header text.
SftTree_SetRowColText_A	Sets the row/column header text using an ANSI string.
SftTree_SetRowColText_W	Sets the row/column header text using a UNICODE string.
SftTree_SetRowHeaderFont	Sets the font used for row header text display.
SftTree_SetRowHeaderStyle	Sets the position of text displayed in the row headers.
SftTree_SetRowHeaderWidth	Sets the width of the row header area.
SftTree_SetRowInfo	Sets an item's row header attributes.
SftTree_SetRowText	Sets an item's row header text.
SftTree_SetRowText_A	Sets an item's row header text using an ANSI string.
SftTree_SetRowText_W	Sets an item's row header text using a UNICODE string.
SftTree_SetSel	Selects or deselects an item (multiple selection tree only).
SftTree_SetSelTextOnly	Sets the display style of selected items.
SftTree_SetShow3D	Sets the current display method used for items.
SftTree_SetShowButton0	Displays or hides level 0 expand/collapse buttons.
SftTree_SetShowButtons	Displays or hides expand/collapse buttons (other than level 0).
SftTree_SetShowGrid	Displays or hides the grid lines.
SftTree_SetShowHeader	Displays or hides the header.
SftTree_SetShowHeaderButtons	Sets the header's display style.
SftTree_SetShowRowColHeaderButton	Sets the row/column header's display style.
SftTree_SetShowRowHeader	Sets the row header display style.
SftTree_SetShowTreeLines	Displays or hides the tree lines.
SftTree_SetShowTruncated	Sets a value that determines whether text is clipped or truncated using trailing "...".
SftTree_SetTabKeyIntercept	Sets the handling of the TAB key while editing item data.
SftTree_SetText	Sets an item's text.
SftTree_SetText_A	Sets an item's text using an ANSI string.
SftTree_SetText_W	Sets an item's text using a UNICODE string.
SftTree_SetTextCol	Sets an item's text.
SftTree_SetTopIndex	Sets the index number of the item shown at the top of the tree control.
SftTree_SortColDependentsEx	Sorts items.
SftTree_SortDependents	Sorts items.
SftTree_SortDependentsEx	Sorts items.
SftTree_UnregisterApp2	Unregisters an application from SftTree/DLL.

C, Function Groups

The SendMessage Windows API function can be used to send messages to a tree control window. Macro definitions are provided for easy use of SftTree/DLL messages, so parameter packing is transparent to the developer.

Application

SftTree_RegisterApp2	Registers an application with SftTree/DLL.
SftTree_UnregisterApp2	Unregisters an application from SftTree/DLL.

Callback Routines

SftTree_SetDeleteCallback	Sets a deletion callback routine, called when items are deleted.
SftTree_SetDrawInfoCallback	Sets a drawing callback routine. Applications can override certain drawing defaults using this callback.

Columns and Column Headers

SftTree_GetAccessColumn	Returns the current column accessed by column specific messages/functions.
SftTree_GetColumns	Returns column information and the number of columns defined.
SftTree_GetColumnsEx	Returns column information and the number of columns defined.
SftTree_GetHeader	Returns the current column's header text.
SftTree_GetHeader_A	Returns the current column's header text as an ANSI string.
SftTree_GetHeader_W	Returns the current column's header text as a UNICODE string.
SftTree_GetHeaderButton	Returns the column number of the currently pressed header button.
SftTree_GetHeaderCol	Returns a column's header text.
SftTree_GetHeaderColLength	Returns the length of a column's header text.
SftTree_GetHeaderFont	Returns the font used for header text display.
SftTree_GetHeaderLength	Returns the length of a column's header text.
SftTree_GetHeaderRect	Returns the dimensions of the header area.
SftTree_GetOverheadWidth	Returns the width of the area added to the first column for hierarchical graphics components.
SftTree_GetResizeHeader	Returns a value indicating if columns are resizable by the user.
SftTree_GetShowHeader	Returns a value indicating the presence of column headers.
SftTree_GetShowHeaderButtons	Returns a value indicating the presence of column header

	buttons.
SftTree_SetAccessColumn	Sets the current column accessed by column specific messages/functions.
SftTree_SetColumns	Sets the number of columns and their attributes.
SftTree_SetColumnsEx	Sets the number of columns and their attributes.
SftTree_SetHeader	Sets the current column's header text.
SftTree_SetHeader_A	Sets the current column's header text using an ANSI string.
SftTree_SetHeader_W	Sets the current column's header text using a UNICODE string.
SftTree_SetHeaderButton	Sets the column number of the currently pressed header button.
SftTree_SetHeaderCol	Sets a column's header text.
SftTree_SetHeaderFont	Sets the font used for header text display.
SftTree_SetResizeHeader	Sets a value indicating if columns are resizable by the user.
SftTree_SetShowHeader	Displays or hides the header.
SftTree_SetShowHeaderButtons	Sets the header's display style.
Current Item	
SftTree_GetCaretColumn	Returns the column where a mouse button was last clicked.
SftTree_GetCaretIndex	Returns the current item (caret location).
SftTree_SetCaretIndex	Sets the current item (caret location).
Drag & Drop	
SftTree_GetDragBitmaps	Returns the drag & drop starting location attribute.
SftTree_GetDragInfo	Returns drag & drop operation information in a SFTTREE_DRAGINFO structure.
SftTree_GetDragType	Returns the drag & drop detection attribute.
SftTree_GetDropHighlight	Returns the current drag & drop target location.
SftTree_GetDropHighlightStyle	Returns the display attribute of the current drag & drop target item.
SftTree_SetDragBitmaps	Sets the drag & drop starting location attribute.
SftTree_SetDragType	Sets the drag & drop detection attribute.
SftTree_SetDropHighlight	Sets the current drag & drop target location.
SftTree_SetDropHighlightStyle	Sets the display attribute of the current drag & drop target item.
Horizontal Scrolling	
SftTree_GetHorizontalExtent	Returns the last defined horizontal extent (in pixels) of the displayable area.
SftTree_GetHorizontalOffset	Returns the current horizontal offset (in pixels) of the displayed area.
SftTree_RecalcHorizontalExtent	Recalculates the optimal horizontal scrolling extent.
SftTree_SetHorizontalExtent	Sets the horizontal extent (in pixels) of the displayable area.
SftTree_SetHorizontalOffset	Sets the horizontal offset (in pixels) of the displayed area.
Item Attributes	
SftTree_GetCellInfo	Returns a cell's attributes.
SftTree_GetEditColumnRect	Returns an item's column location in client area coordinates.
SftTree_GetItemBitmap	Returns an item's item bitmap.
SftTree_GetItemData	Returns an item's application-specific 32-bit data value.
SftTree_GetItemExpand	Returns an item's expand status.
SftTree_GetItemLabel	Returns an item's label bitmap information.
SftTree_GetItemLevel	Returns an item's level number.
SftTree_GetItemRect	Returns an item's location in client area coordinates.
SftTree_GetItemShown	Returns an item's visibility status.
SftTree_GetItemStatus	Returns an item's enabled/disabled status.
SftTree_GetText	Returns an item's text.
SftTree_GetText_A	Returns an item's text as an ANSI string.

SftTree_GetText_W	Returns an item's text as a UNICODE string.
SftTree_GetTextCol	Returns an item's text.
SftTree_GetTextColLength	Returns the length of an item's text.
SftTree_GetTextLength	Returns the length of an item's text.
SftTree_SetCellInfo	Sets a cell's attributes.
SftTree_SetItemBitmap	Sets an item's item bitmap.
SftTree_SetItemData	Sets an item's application-specific 32-bit data value.
SftTree_SetItemExpand	Sets an item's expand status.
SftTree_SetItemLabel	Sets an item's label bitmap information.
SftTree_SetItemLevel	Sets an item's level number.
SftTree_SetItemShown	Sets an item's visibility status.
SftTree_SetItemStatus	Sets an item's enabled/disabled status.
SftTree_SetText	Sets an item's text.
SftTree_SetText_A	Sets an item's text using an ANSI string.
SftTree_SetText_W	Sets an item's text using a UNICODE string.
SftTree_SetTextCol	Sets an item's text.

Item Hit Testing

SftTree_CalcIndexFromPoint	Calculates the item index number given a point in tree control client area coordinates.
SftTree_CalcIndexFromPointEx	Calculates the item index number given a point in tree control client area coordinates.

Items

SftTree_AddString	Adds a new item at the end of the tree control at level 0 using a string.
SftTree_AddString_A	Adds a new item at the end of the tree control at level 0 using an ANSI string.
SftTree_AddString_W	Adds a new item at the end of the tree control at level 0 using a UNICODE string.
SftTree_CopyItem	Copies an item to a new location.
SftTree_DeleteDependents	Deletes an item's dependent items.
SftTree_DeleteString	Deletes an item.
SftTree_FindItem	Searches item data.
SftTree_FindString	Searches a string.
SftTree_FindString_A	Searches an ANSI string.
SftTree_FindString_W	Searches a UNICODE string.
SftTree_FindStringCol	Searches a string.
SftTree_FindStringColExact	Searches a string.
SftTree_FindStringExact	Searches a string.
SftTree_FindStringExact_A	Searches an ANSI string.
SftTree_FindStringExact_W	Searches a UNICODE string.
SftTree_GetCount	Returns the number of items in the tree control.
SftTree_InsertString	Inserts a new item using a string.
SftTree_InsertString_A	Inserts a new item using an ANSI string.
SftTree_InsertString_W	Inserts a new item using a UNICODE string.
SftTree_MoveItem	Moves an item to a new position in the tree control.
SftTree_ResetContent	Removes all items from the tree control.

Positioning & Resizing

SftTree_CalcOptimalColumnWidth	Calculates a column's optimal width so text and bitmaps are not clipped.
SftTree_CalcOptimalRowHeaderWidth	Calculates the optimal width for row headers so text and bitmaps are not clipped.
SftTree_GetTopIndex	Returns the index number of the item currently shown at the top of the tree control.
SftTree_MakeColumnOptimal	Sets the optimal column width so that the text and bitmaps of

	all items can be displayed.
SftTree_MakeColumnVisible	Horizontally scrolls the specified column into view so that it is displayed in the tree control's client area.
SftTree_MakeIntegralHeight	Vertically resizes the tree control so visible items are displayed in their entirety.
SftTree_MakeRowHeaderOptimal	Sets the optimal row header width so that the text and bitmaps of all row headers can be displayed.
SftTree_MakeRowVisible	Vertically scrolls the specified item into view so that it is displayed in the tree control's client area.
SftTree_SetTopIndex	Sets the index number of the item shown at the top of the tree control.

Relationships

SftTree_GetDependent	Returns dependent item information for a parent item.
SftTree_GetDependentCount	Returns the number of dependents for a parent item.
SftTree_GetParent	Returns an item's parent index.
SftTree_GetSibling	Returns an item's sibling information.
SftTree_GetTopParent	Returns the highest level parent index number for an item.

Row/Column Header

SftTree_GetRowColBitmap	Returns the bitmap displayed in the row/column header.
SftTree_GetRowColBitmapStyle	Returns the position of the bitmap displayed in the row/column header.
SftTree_GetRowColHeaderButton	Returns a value indicating if the row/column header button is currently down (pressed).
SftTree_GetRowColHeaderRect	Returns the dimensions of the row/column header area.
SftTree_GetRowColHeaderStyle	Returns the position of the text displayed in the row/column header.
SftTree_GetRowColText	Returns the row/column header text.
SftTree_GetRowColText_A	Returns the row/column header text as an ANSI string.
SftTree_GetRowColText_W	Returns the row/column header text as a UNICODE string.
SftTree_GetRowColTextLength	Returns the row/column header text length.
SftTree_GetShowRowColHeaderButton	Returns the row/column header's display style.
SftTree_SetRowColBitmap	Sets the bitmap displayed in the row/column header.
SftTree_SetRowColBitmapStyle	Sets the position of the bitmap displayed in the row/column header.
SftTree_SetRowColHeaderButton	Sets a value indicating if the row/column header button is currently down (pressed).
SftTree_SetRowColHeaderStyle	Sets the position of the text displayed in the row/column header.
SftTree_SetRowColText	Sets the row/column header text.
SftTree_SetRowColText_A	Sets the row/column header text using an ANSI string.
SftTree_SetRowColText_W	Sets the row/column header text using a UNICODE string.
SftTree_SetShowRowColHeaderButton	Sets the row/column header's display style.

Row Header

SftTree_GetRowHeaderFont	Returns the font used for row header text display.
SftTree_GetRowHeaderRect	Returns the dimensions of the row header area.
SftTree_GetRowHeaderStyle	Returns the position of text displayed in the row headers.
SftTree_GetRowHeaderWidth	Returns the width of the row header area.
SftTree_GetRowInfo	Returns an item's row header attributes.
SftTree_GetRowText	Returns an item's row header text.
SftTree_GetRowText_A	Returns an item's row header text as an ANSI string.
SftTree_GetRowText_W	Returns an item's row header text as a UNICODE string.
SftTree_GetRowTextLength	Returns an item's row header text length.
SftTree_GetShowRowHeader	Returns the row header display style.
SftTree_SetRowHeaderFont	Sets the font used for row header text display.

SftTree_SetRowHeaderStyle	Sets the position of text displayed in the row headers.
SftTree_SetRowHeaderWidth	Sets the width of the row header area.
SftTree_SetRowInfo	Sets an item's row header attributes.
SftTree_SetRowText	Sets an item's row header text.
SftTree_SetRowText_A	Sets an item's row header text using an ANSI string.
SftTree_SetRowText_W	Sets an item's row header text using a UNICODE string.
SftTree_SetShowRowHeader	Sets the row header display style.

Scrolling

SftTree_GetDisableNoScroll	Returns the scroll bar handling status.
SftTree_SetDisableNoScroll	Sets the scroll bar handling status.

Selection

SftTree_GetCurSel	Returns the index of the selected item (single selection tree control).
SftTree_GetSel	Returns the selection status of an item (multiple selection tree only).
SftTree_GetSelCount	Returns the number of currently selected items (multiple selection tree only).
SftTree_GetSelItems	Fills an array with the index numbers of currently selected items (multiple selection tree only).
SftTree_GetSelTextOnly	Returns a value indicating the display style of selected items.
SftTree_SelItemRange	Selects or deselects a range of items (multiple selection tree only).
SftTree_SetCurSel	Sets the currently selected item (single selection tree only).
SftTree_SetSel	Selects or deselects an item (multiple selection tree only).
SftTree_SetSelTextOnly	Sets the display style of selected items.

Sorting

SftTree_SortColDependentsEx	Sorts items.
SftTree_SortDependents	Sorts items.
SftTree_SortDependentsEx	Sorts items.

Tree Attributes

SftTree_GetCtlColors	Returns the tree control's color attributes.
SftTree_GetForwardChildMsgs	Returns the child window message handling status.
SftTree_GetGridStyle	Returns the grid line display style.
SftTree_GetItemHeight	Returns the current height (in pixels) of each item.
SftTree_GetItemLines	Returns the number of text lines used for item height calculation.
SftTree_GetItemsShown	Returns the number of items displayable in the tree control's client area, including partial items.
SftTree_GetItemsShownComplete	Returns the number of items displayable in the tree control's client area, not including partial items.
SftTree_GetNoFocusStyle	Returns the display style of selected items when the tree control does not have the input focus.
SftTree_GetShow3D	Returns a value indicating the current display method used for items.
SftTree_GetShowBitmaps	Returns a value indicating the presence of item bitmaps.
SftTree_GetShowButton0	Returns a value indicating the presence of level 0 expand/collapse buttons.
SftTree_GetShowButtons	Returns a value indicating the presence of expand/collapse buttons (other than level 0).
SftTree_GetShowGrid	Returns a value indicating the presence of grid lines.
SftTree_GetShowLabels	Returns a value indicating the presence of label bitmaps.
SftTree_GetShowPlusMinus	Returns a value indicating the presence of plus/minus bitmaps.
SftTree_GetShowTreeLines	Returns a value indicating the presence of tree lines.

SftTree_GetShowTruncated	Returns a value that determines whether text is clipped or truncated using trailing "...".
SftTree_GetTabKeyIntercept	Returns the status of TAB key handling while editing item data.
SftTree_SetBitmaps	Registers the size and sets the default item bitmaps used for all items.
SftTree_SetButtons	Registers the size and sets the bitmaps used for expand/collapse buttons.
SftTree_SetCtlColors	Sets the tree control's color attributes.
SftTree_SetForwardChildMsgs	Sets the child window message handling status.
SftTree_SetGridStyle	Sets the grid line display style.
SftTree_SetItemLines	Sets the number of text lines used for item height calculation.
SftTree_SetNoFocusStyle	Sets the display style of selected items when the tree control does not have the input focus.
SftTree_SetPlusMinus	Set the plus/minus bitmaps used for all items.
SftTree_SetShow3D	Sets the current display method used for items.
SftTree_SetShowButton0	Displays or hides level 0 expand/collapse buttons.
SftTree_SetShowButtons	Displays or hides expand/collapse buttons (other than level 0).
SftTree_SetShowGrid	Displays or hides the grid lines.
SftTree_SetShowTreeLines	Displays or hides the tree lines.
SftTree_SetShowTruncated	Sets a value that determines whether text is clipped or truncated using trailing "...".
SftTree_SetTabKeyIntercept	Sets the handling of the TAB key while editing item data.

MFC/C++ CSftTree Class

The CSftTree class defines all necessary functions to communicate with a tree control. CSftTree is derived from the class CWnd.

Function	*Description*
~CSftTree	Standard destructor.
AddString	Adds a new item at the end of the tree control at level 0 using a string.
CalcIndexFromPoint	Calculates the item index number given a point in tree control client area coordinates.
CalcIndexFromPointEx	Calculates the item index number given a point in tree control client area coordinates.
CalcOptimalColumnWidth	Calculates a column's optimal width so text and bitmaps are not clipped.
CalcOptimalRowHeaderWidth	Calculates the optimal width for row headers so text and bitmaps are not clipped.
CopyItem	Copies an item to a new location.
Create	Creates a tree control window.
CreateEx	Creates a tree control window.
CSftTree	Standard constructor.
DeleteDependents	Deletes an item's dependent items.
DeleteString	Deletes an item.
FindItem	Searches item data.
FindString	Searches a string.
FindStringExact	Searches a string.
GetAccessColumn	Returns the current column accessed by column specific messages/functions.
GetCaretColumn	Returns the column where a mouse button was last clicked.
GetCaretIndex	Returns the current item (caret location).
GetCellInfo	Returns a cell's attributes.
GetColumns	Returns column information and the number of columns defined.
GetCount	Returns the number of items in the tree control.
GetCtlColors	Returns the tree control's color attributes.
GetCurSel	Returns the index of the selected item (single selection tree control).
GetDependent	Returns dependent item information for a parent item.
GetDependentCount	Returns the number of dependents for a parent item.
GetDisableNoScroll	Returns the scroll bar handling status.
GetDragBitmaps	Returns the drag & drop starting location attribute.
GetDragInfo	Returns drag & drop operation information in a SFTTREE_DRAGINFO structure.
GetDragType	Returns the drag & drop detection attribute.
GetDropHighlight	Returns the current drag & drop target location.
GetDropHighlightStyle	Returns the display attribute of the current drag & drop target item.
GetEditColumnRect	Returns an item's column location in client area coordinates.
GetForwardChildMsgs	Returns the child window message handling status.
GetGridStyle	Returns the grid line display style.
GetHeader	Returns a column's header text.
GetHeaderButton	Returns the column number of the currently pressed header button.
GetHeaderFont	Returns the font used for header text display.
GetHeaderLen	Returns the length of a column's header text.
GetHeaderRect	Returns the dimensions of the header area.
GetHorizontalExtent	Returns the last defined horizontal extent (in pixels) of the displayable area.
GetHorizontalOffset	Returns the current horizontal offset (in pixels) of the displayed area.
GetItemBitmap	Returns an item's item bitmap.
GetItemData	Returns an item's application-specific 32-bit data value.

GetItemDataPtr	Returns an item's application-specific 32-bit data value as a pointer.
GetItemExpand	Returns an item's expand status.
GetItemHeight	Returns the current height (in pixels) of each item.
GetItemLabel	Returns an item's label bitmap information.
GetItemLevel	Returns an item's level number.
GetItemLines	Returns the number of text lines used for item height calculation.
GetItemRect	Returns an item's location in client area coordinates.
GetItemShown	Returns an item's visibility status.
GetItemsShown	Returns the number of items displayable in the tree control's client area, including partial items.
GetItemsShownComplete	Returns the number of items displayable in the tree control's client area, not including partial items.
GetItemStatus	Returns an item's enabled/disabled status.
GetNoFocusStyle	Returns the display style of selected items when the tree control does not have the input focus.
GetOverheadWidth	Returns the width of the area added to the first column for hierarchical graphics components.
GetParent	Returns an item's parent index.
GetResizeHeader	Returns a value indicating if columns are resizable by the user.
GetRowColBitmap	Returns the bitmap displayed in the row/column header.
GetRowColBitmapStyle	Returns the position of the bitmap displayed in the row/column header.
GetRowColHeaderButton	Returns a value indicating if the row/column header button is currently down (pressed).
GetRowColHeaderRect	Returns the dimensions of the row/column header area.
GetRowColHeaderStyle	Returns the position of the text displayed in the row/column header.
GetRowColText	Returns the row/column header text.
GetRowColTextLen	Returns the row/column header text length.
GetRowHeaderFont	Returns the font used for row header text display.
GetRowHeaderRect	Returns the dimensions of the row header area.
GetRowHeaderStyle	Returns the position of text displayed in the row headers.
GetRowHeaderWidth	Returns the width of the row header area.
GetRowInfo	Returns an item's row header attributes.
GetRowText	Returns an item's row header text.
GetRowTextLen	Returns an item's row header text length.
GetSel	Returns the selection status of an item (multiple selection tree only).
GetSelCount	Returns the number of currently selected items (multiple selection tree only).
GetSelItems	Fills an array with the index numbers of currently selected items (multiple selection tree only).
GetSelTextOnly	Returns a value indicating the display style of selected items.
GetShow3D	Returns a value indicating the current display method used for items.
GetShowBitmaps	Returns a value indicating the presence of item bitmaps.
GetShowButton0	Returns a value indicating the presence of level 0 expand/collapse buttons.
GetShowButtons	Returns a value indicating the presence of expand/collapse buttons (other than level 0).
GetShowGrid	Returns a value indicating the presence of grid lines.
GetShowHeader	Returns a value indicating the presence of column headers.
GetShowHeaderButtons	Returns a value indicating the presence of column header buttons.
GetShowLabels	Returns a value indicating the presence of label bitmaps.
GetShowPlusMinus	Returns a value indicating the presence of plus/minus bitmaps.
GetShowRowColHeaderButton	Returns the row/column header's display style.
GetShowRowHeader	Returns the row header display style.
GetShowTreeLines	Returns a value indicating the presence of tree lines.
GetShowTruncated	Returns a value that determines whether text is clipped or truncated

	using trailing "...".
GetSibling	Returns an item's sibling information.
GetTabKeyIntercept	Returns the status of TAB key handling while editing item data.
GetText	Returns an item's text.
GetTextLen	Returns the length of an item's text.
GetTopIndex	Returns the index number of the item currently shown at the top of the tree control.
GetTopParent	Returns the highest level parent index number for an item.
InsertString	Inserts a new item using a string.
MakeColumnOptimal	Sets the optimal column width so that the text and bitmaps of all items can be displayed.
MakeColumnVisible	Horizontally scrolls the specified column into view so that it is displayed in the tree control's client area.
MakeIntegralHeight	Vertically resizes the tree control so visible items are displayed in their entirety.
MakeRowHeaderOptimal	Sets the optimal row header width so that the text and bitmaps of all row headers can be displayed.
MakeRowVisible	Vertically scrolls the specified item into view so that it is displayed in the tree control's client area.
MoveItem	Moves an item to a new position in the tree control.
RecalcHorizontalExtent	Recalculates the optimal horizontal scrolling extent.
RegisterApp	Registers an application with SftTree/DLL.
ResetContent	Removes all items from the tree control.
SelItemRange	Selects or deselects a range of items (multiple selection tree only).
SetAccessColumn	Sets the current column accessed by column specific messages/functions.
SetBitmaps	Registers the size and sets the default item bitmaps used for all items.
SetButtons	Registers the size and sets the bitmaps used for expand/collapse buttons.
SetCaretIndex	Sets the current item (caret location).
SetCellInfo	Sets a cell's attributes.
SetColumns	Sets the number of columns and their attributes.
SetCtlColors	Sets the tree control's color attributes.
SetCurSel	Sets the currently selected item (single selection tree only).
SetDeleteCallback	Sets a deletion callback routine, called when items are deleted.
SetDisableNoScroll	Sets the scroll bar handling status.
SetDragBitmaps	Sets the drag & drop starting location attribute.
SetDragType	Sets the drag & drop detection attribute.
SetDrawInfoCallback	Sets a drawing callback routine. Applications can override certain drawing defaults using this callback.
SetDropHighlight	Sets the current drag & drop target location.
SetDropHighlightStyle	Sets the display attribute of the current drag & drop target item.
SetForwardChildMsgs	Sets the child window message handling status.
SetGridStyle	Sets the grid line display style.
SetHeader	Sets the current column's header text.
SetHeaderButton	Sets the column number of the currently pressed header button.
SetHeaderFont	Sets the font used for header text display.
SetHorizontalExtent	Sets the horizontal extent (in pixels) of the displayable area.
SetHorizontalOffset	Sets the horizontal offset (in pixels) of the displayed area.
SetItemBitmap	Sets an item's item bitmap.
SetItemData	Sets an item's application-specific 32-bit data value.
SetItemDataPtr	Sets an item's application-specific 32-bit data value using a pointer value.
SetItemExpand	Sets an item's expand status.
SetItemLabel	Sets an item's label bitmap information.

SetItemLevel	Sets an item's level number.
SetItemLines	Sets the number of text lines used for item height calculation.
SetItemShown	Sets an item's visibility status.
SetItemStatus	Sets an item's enabled/disabled status.
SetNoFocusStyle	Sets the display style of selected items when the tree control does not have the input focus.
SetPlusMinus	Set the plus/minus bitmaps used for all items.
SetResizeHeader	Sets a value indicating if columns are resizable by the user.
SetRowColBitmap	Sets the bitmap displayed in the row/column header.
SetRowColBitmapStyle	Sets the position of the bitmap displayed in the row/column header.
SetRowColHeaderButton	Sets a value indicating if the row/column header button is currently down (pressed).
SetRowColHeaderStyle	Sets the position of the text displayed in the row/column header.
SetRowColText	Sets the row/column header text.
SetRowHeaderFont	Sets the font used for row header text display.
SetRowHeaderStyle	Sets the position of text displayed in the row headers.
SetRowHeaderWidth	Sets the width of the row header area.
SetRowInfo	Sets an item's row header attributes.
SetRowText	Sets an item's row header text.
SetSel	Selects or deselects an item (multiple selection tree only).
SetSelTextOnly	Sets the display style of selected items.
SetShow3D	Sets the current display method used for items.
SetShowButton0	Displays or hides level 0 expand/collapse buttons.
SetShowButtons	Displays or hides expand/collapse buttons (other than level 0).
SetShowGrid	Displays or hides the grid lines.
SetShowHeader	Displays or hides the header.
SetShowHeaderButtons	Sets the header's display style.
SetShowRowColHeaderButton	Sets the row/column header's display style.
SetShowRowHeader	Sets the row header display style.
SetShowTreeLines	Displays or hides the tree lines.
SetShowTruncated	Sets a value that determines whether text is clipped or truncated using trailing "...".
SetTabKeyIntercept	Sets the handling of the TAB key while editing item data.
SetText	Sets an item's text.
SetTopIndex	Sets the index number of the item shown at the top of the tree control.
SortDependents	Sorts items.
UnregisterApp	Unregisters an application from SftTree/DLL.

MFC/C++ CSftTree Class

The CSftTree class defines all necessary functions to communicate with a tree control. CSftTree is derived from the class CWnd.

Application

~CSftTree	Standard destructor.
Create	Creates a tree control window.
CreateEx	Creates a tree control window.
CSftTree	Standard constructor.
RegisterApp	Registers an application with SftTree/DLL.
UnregisterApp	Unregisters an application from SftTree/DLL.

Callback Routines

SetDeleteCallback	Sets a deletion callback routine, called when items are deleted.
SetDrawInfoCallback	Sets a drawing callback routine. Applications can override certain drawing defaults using this callback.

Columns and Column Headers

GetAccessColumn	Returns the current column accessed by column specific messages/functions.
GetColumns	Returns column information and the number of columns defined.
GetHeader	Returns a column's header text.
GetHeaderButton	Returns the column number of the currently pressed header button.
GetHeaderFont	Returns the font used for header text display.
GetHeaderLen	Returns the length of a column's header text.
GetHeaderRect	Returns the dimensions of the header area.
GetOverheadWidth	Returns the width of the area added to the first column for hierarchical graphics components.
GetResizeHeader	Returns a value indicating if columns are resizable by the user.
GetShowHeader	Returns a value indicating the presence of column headers.
GetShowHeaderButtons	Returns a value indicating the presence of column header buttons.
SetAccessColumn	Sets the current column accessed by column specific messages/functions.
SetColumns	Sets the number of columns and their attributes.
SetHeader	Sets the current column's header text.
SetHeaderButton	Sets the column number of the currently pressed header button.
SetHeaderFont	Sets the font used for header text display.
SetResizeHeader	Sets a value indicating if columns are resizable by the user.
SetShowHeader	Displays or hides the header.

SetShowHeaderButtons	Sets the header's display style.
Current Item	
GetCaretColumn	Returns the column where a mouse button was last clicked.
GetCaretIndex	Returns the current item (caret location).
SetCaretIndex	Sets the current item (caret location).
Drag & Drop	
GetDragBitmaps	Returns the drag & drop starting location attribute.
GetDragInfo	Returns drag & drop operation information in a SFTTREE_DRAGINFO structure.
GetDragType	Returns the drag & drop detection attribute.
GetDropHighlight	Returns the current drag & drop target location.
GetDropHighlightStyle	Returns the display attribute of the current drag & drop target item.
SetDragBitmaps	Sets the drag & drop starting location attribute.
SetDragType	Sets the drag & drop detection attribute.
SetDropHighlight	Sets the current drag & drop target location.
SetDropHighlightStyle	Sets the display attribute of the current drag & drop target item.
Horizontal Scrolling	
GetHorizontalExtent	Returns the last defined horizontal extent (in pixels) of the displayable area.
GetHorizontalOffset	Returns the current horizontal offset (in pixels) of the displayed area.
RecalcHorizontalExtent	Recalculates the optimal horizontal scrolling extent.
SetHorizontalExtent	Sets the horizontal extent (in pixels) of the displayable area.
SetHorizontalOffset	Sets the horizontal offset (in pixels) of the displayed area.
Item Attributes	
GetCellInfo	Returns a cell's attributes.
GetEditColumnRect	Returns an item's column location in client area coordinates.
GetItemBitmap	Returns an item's item bitmap.
GetItemData	Returns an item's application-specific 32-bit data value.
GetItemDataPtr	Returns an item's application-specific 32-bit data value as a pointer.
GetItemExpand	Returns an item's expand status.
GetItemLabel	Returns an item's label bitmap information.
GetItemLevel	Returns an item's level number.
GetItemRect	Returns an item's location in client area coordinates.
GetItemShown	Returns an item's visibility status.
GetItemStatus	Returns an item's enabled/disabled status.
GetText	Returns an item's text.
GetTextLen	Returns the length of an item's text.
SetCellInfo	Sets a cell's attributes.
SetItemBitmap	Sets an item's item bitmap.
SetItemData	Sets an item's application-specific 32-bit data value.
SetItemDataPtr	Sets an item's application-specific 32-bit data value using a pointer value.
SetItemExpand	Sets an item's expand status.
SetItemLabel	Sets an item's label bitmap information.
SetItemLevel	Sets an item's level number.
SetItemShown	Sets an item's visibility status.
SetItemStatus	Sets an item's enabled/disabled status.
SetText	Sets an item's text.
Item Hit Testing	
CalcIndexFromPoint	Calculates the item index number given a point in tree control client area coordinates.
CalcIndexFromPointEx	Calculates the item index number given a point in tree control client area coordinates.

Items

AddString	Adds a new item at the end of the tree control at level 0 using a string.
CopyItem	Copies an item to a new location.
DeleteDependents	Deletes an item's dependent items.
DeleteString	Deletes an item.
FindItem	Searches item data.
FindString	Searches a string.
FindStringExact	Searches a string.
GetCount	Returns the number of items in the tree control.
InsertString	Inserts a new item using a string.
MoveItem	Moves an item to a new position in the tree control.
ResetContent	Removes all items from the tree control.

Positioning & Resizing

CalcOptimalColumnWidth	Calculates a column's optimal width so text and bitmaps are not clipped.
CalcOptimalRowHeaderWidth	Calculates the optimal width for row headers so text and bitmaps are not clipped.
GetTopIndex	Returns the index number of the item currently shown at the top of the tree control.
MakeColumnOptimal	Sets the optimal column width so that the text and bitmaps of all items can be displayed.
MakeColumnVisible	Horizontally scrolls the specified column into view so that it is displayed in the tree control's client area.
MakeIntegralHeight	Vertically resizes the tree control so visible items are displayed in their entirety.
MakeRowHeaderOptimal	Sets the optimal row header width so that the text and bitmaps of all row headers can be displayed.
MakeRowVisible	Vertically scrolls the specified item into view so that it is displayed in the tree control's client area.
SetTopIndex	Sets the index number of the item shown at the top of the tree control.

Relationships

GetDependent	Returns dependent item information for a parent item.
GetDependentCount	Returns the number of dependents for a parent item.
GetParent	Returns an item's parent index.
GetSibling	Returns an item's sibling information.
GetTopParent	Returns the highest level parent index number for an item.

Row/Column Header

GetRowColBitmap	Returns the bitmap displayed in the row/column header.
GetRowColBitmapStyle	Returns the position of the bitmap displayed in the row/column header.
GetRowColHeaderButton	Returns a value indicating if the row/column header button is currently down (pressed).
GetRowColHeaderRect	Returns the dimensions of the row/column header area.
GetRowColHeaderStyle	Returns the position of the text displayed in the row/column header.
GetRowColText	Returns the row/column header text.
GetRowColTextLen	Returns the row/column header text length.
GetShowRowColHeaderButton	Returns the row/column header's display style.
SetRowColBitmap	Sets the bitmap displayed in the row/column header.
SetRowColBitmapStyle	Sets the position of the bitmap displayed in the row/column header.
SetRowColHeaderButton	Sets a value indicating if the row/column header button is currently down (pressed).
SetRowColHeaderStyle	Sets the position of the text displayed in the row/column header.

SetRowColText	Sets the row/column header text.
SetShowRowColHeaderButton	Sets the row/column header's display style.

Row Header

GetRowHeaderFont	Returns the font used for row header text display.
GetRowHeaderRect	Returns the dimensions of the row header area.
GetRowHeaderStyle	Returns the position of text displayed in the row headers.
GetRowHeaderWidth	Returns the width of the row header area.
GetRowInfo	Returns an item's row header attributes.
GetRowText	Returns an item's row header text.
GetRowTextLen	Returns an item's row header text length.
GetShowRowHeader	Returns the row header display style.
SetRowHeaderFont	Sets the font used for row header text display.
SetRowHeaderStyle	Sets the position of text displayed in the row headers.
SetRowHeaderWidth	Sets the width of the row header area.
SetRowInfo	Sets an item's row header attributes.
SetRowText	Sets an item's row header text.
SetShowRowHeader	Sets the row header display style.

Scrolling

GetDisableNoScroll	Returns the scroll bar handling status.
SetDisableNoScroll	Sets the scroll bar handling status.

Selection

GetCurSel	Returns the index of the selected item (single selection tree control).
GetSel	Returns the selection status of an item (multiple selection tree only).
GetSelCount	Returns the number of currently selected items (multiple selection tree only).
GetSelItems	Fills an array with the index numbers of currently selected items (multiple selection tree only).
GetSelTextOnly	Returns a value indicating the display style of selected items.
SelItemRange	Selects or deselects a range of items (multiple selection tree only).
SetCurSel	Sets the currently selected item (single selection tree only).
SetSel	Selects or deselects an item (multiple selection tree only).
SetSelTextOnly	Sets the display style of selected items.

Sorting

SortDependents	Sorts items.

Tree Attributes

GetCtlColors	Returns the tree control's color attributes.
GetForwardChildMsgs	Returns the child window message handling status.
GetGridStyle	Returns the grid line display style.
GetItemHeight	Returns the current height (in pixels) of each item.
GetItemLines	Returns the number of text lines used for item height calculation.
GetItemsShown	Returns the number of items displayable in the tree control's client area, including partial items.
GetItemsShownComplete	Returns the number of items displayable in the tree control's client area, not including partial items.
GetNoFocusStyle	Returns the display style of selected items when the tree control does not have the input focus.
GetShow3D	Returns a value indicating the current display method used for items.
GetShowBitmaps	Returns a value indicating the presence of item bitmaps.
GetShowButton0	Returns a value indicating the presence of level 0 expand/collapse buttons.
GetShowButtons	Returns a value indicating the presence of expand/collapse buttons (other than level 0).
GetShowGrid	Returns a value indicating the presence of grid lines.

GetShowLabels	Returns a value indicating the presence of label bitmaps.
GetShowPlusMinus	Returns a value indicating the presence of plus/minus bitmaps.
GetShowTreeLines	Returns a value indicating the presence of tree lines.
GetShowTruncated	Returns a value that determines whether text is clipped or truncated using trailing "...".
GetTabKeyIntercept	Returns the status of TAB key handling while editing item data.
SetBitmaps	Registers the size and sets the default item bitmaps used for all items.
SetButtons	Registers the size and sets the bitmaps used for expand/collapse buttons.
SetCtlColors	Sets the tree control's color attributes.
SetForwardChildMsgs	Sets the child window message handling status.
SetGridStyle	Sets the grid line display style.
SetItemLines	Sets the number of text lines used for item height calculation.
SetNoFocusStyle	Sets the display style of selected items when the tree control does not have the input focus.
SetPlusMinus	Set the plus/minus bitmaps used for all items.
SetShow3D	Sets the current display method used for items.
SetShowButton0	Displays or hides level 0 expand/collapse buttons.
SetShowButtons	Displays or hides expand/collapse buttons (other than level 0).
SetShowGrid	Displays or hides the grid lines.
SetShowTreeLines	Displays or hides the tree lines.
SetShowTruncated	Sets a value that determines whether text is clipped or truncated using trailing "...".
SetTabKeyIntercept	Sets the handling of the TAB key while editing item data.

OWL/C++ TSftTree Class

The TSftTree class defines all necessary functions to communicate with a tree control. TSftTree is derived from the class TControl.

Function	*Description*
~TSftTree	Standard destructor.
AddString	Adds a new item at the end of the tree control at level 0 using a string.
CalcIndexFromPoint	Calculates the item index number given a point in tree control client area coordinates.
CalcIndexFromPointEx	Calculates the item index number given a point in tree control client area coordinates.
CalcOptimalColumnWidth	Calculates a column's optimal width so text and bitmaps are not clipped.
CalcOptimalRowHeaderWidth	Calculates the optimal width for row headers so text and bitmaps are not clipped.
ClearList	Removes all items from the tree control.
CopyItem	Copies an item to a new location.
DeleteDependents	Deletes an item's dependent items.
DeleteString	Deletes an item.
FindItem	Searches item data.
FindString	Searches a string.
FindStringExact	Searches a string.
GetAccessColumn	Returns the current column accessed by column specific messages/functions.
GetCaretColumn	Returns the column where a mouse button was last clicked.
GetCaretIndex	Returns the current item (caret location).
GetCellInfo	Returns a cell's attributes.
GetColumns	Returns column information and the number of columns defined.
GetCount	Returns the number of items in the tree control.
GetCtlColors	Returns the tree control's color attributes.
GetDependent	Returns dependent item information for a parent item.
GetDependentCount	Returns the number of dependents for a parent item.
GetDisableNoScroll	Returns the scroll bar handling status.
GetDragBitmaps	Returns the drag & drop starting location attribute.
GetDragInfo	Returns drag & drop operation information in a SFTTREE_DRAGINFO structure.
GetDragType	Returns the drag & drop detection attribute.
GetDropHighlight	Returns the current drag & drop target location.
GetDropHighlightStyle	Returns the display attribute of the current drag & drop target item.
GetEditColumnRect	Returns an item's column location in client area coordinates.
GetForwardChildMsgs	Returns the child window message handling status.
GetGridStyle	Returns the grid line display style.
GetHeader	Returns a column's header text.
GetHeaderButton	Returns the column number of the currently pressed header button.
GetHeaderFont	Returns the font used for header text display.
GetHeaderLen	Returns the length of a column's header text.
GetHeaderRect	Returns the dimensions of the header area.
GetHorizontalExtent	Returns the last defined horizontal extent (in pixels) of the displayable area.
GetHorizontalOffset	Returns the current horizontal offset (in pixels) of the displayed area.
GetItemBitmap	Returns an item's item bitmap.
GetItemData	Returns an item's application-specific 32-bit data value.
GetItemExpand	Returns an item's expand status.
GetItemHeight	Returns the current height (in pixels) of each item.
GetItemLabel	Returns an item's label bitmap information.

GetItemLevel	Returns an item's level number.
GetItemLines	Returns the number of text lines used for item height calculation.
GetItemRect	Returns an item's location in client area coordinates.
GetItemShown	Returns an item's visibility status.
GetItemsShown	Returns the number of items displayable in the tree control's client area, including partial items.
GetItemsShownComplete	Returns the number of items displayable in the tree control's client area, not including partial items.
GetItemStatus	Returns an item's enabled/disabled status.
GetNoFocusStyle	Returns the display style of selected items when the tree control does not have the input focus.
GetOverheadWidth	Returns the width of the area added to the first column for hierarchical graphics components.
GetParent	Returns an item's parent index.
GetResizeHeader	Returns a value indicating if columns are resizable by the user.
GetRowColBitmap	Returns the bitmap displayed in the row/column header.
GetRowColBitmapStyle	Returns the position of the bitmap displayed in the row/column header.
GetRowColHeaderButton	Returns a value indicating if the row/column header button is currently down (pressed).
GetRowColHeaderRect	Returns the dimensions of the row/column header area.
GetRowColHeaderStyle	Returns the position of the text displayed in the row/column header.
GetRowColText	Returns the row/column header text.
GetRowColTextLen	Returns the row/column header text length.
GetRowHeaderFont	Returns the font used for row header text display.
GetRowHeaderRect	Returns the dimensions of the row header area.
GetRowHeaderStyle	Returns the position of text displayed in the row headers.
GetRowHeaderWidth	Returns the width of the row header area.
GetRowInfo	Returns an item's row header attributes.
GetRowText	Returns an item's row header text.
GetRowTextLen	Returns an item's row header text length.
GetSel	Returns the selection status of an item (multiple selection tree only).
GetSelCount	Returns the number of currently selected items (multiple selection tree only).
GetSelIndex	Returns the index of the selected item (single selection tree control).
GetSelIndexes	Fills an array with the index numbers of currently selected items (multiple selection tree only).
GetSelTextOnly	Returns a value indicating the display style of selected items.
GetShow3D	Returns a value indicating the current display method used for items.
GetShowBitmaps	Returns a value indicating the presence of item bitmaps.
GetShowButton0	Returns a value indicating the presence of level 0 expand/collapse buttons.
GetShowButtons	Returns a value indicating the presence of expand/collapse buttons (other than level 0).
GetShowGrid	Returns a value indicating the presence of grid lines.
GetShowHeader	Returns a value indicating the presence of column headers.
GetShowHeaderButtons	Returns a value indicating the presence of column header buttons.
GetShowLabels	Returns a value indicating the presence of label bitmaps.
GetShowPlusMinus	Returns a value indicating the presence of plus/minus bitmaps.
GetShowRowColHeaderButton	Returns the row/column header's display style.
GetShowRowHeader	Returns the row header display style.
GetShowTreeLines	Returns a value indicating the presence of tree lines.
GetShowTruncated	Returns a value that determines whether text is clipped or truncated using trailing "...".
GetSibling	Returns an item's sibling information.
GetString	Returns an item's text.

GetStringLen	Returns the length of an item's text.
GetTabKeyIntercept	Returns the status of TAB key handling while editing item data.
GetTopIndex	Returns the index number of the item currently shown at the top of the tree control.
GetTopParent	Returns the highest level parent index number for an item.
InsertString	Inserts a new item using a string.
MakeColumnOptimal	Sets the optimal column width so that the text and bitmaps of all items can be displayed.
MakeColumnVisible	Horizontally scrolls the specified column into view so that it is displayed in the tree control's client area.
MakeIntegralHeight	Vertically resizes the tree control so visible items are displayed in their entirety.
MakeRowHeaderOptimal	Sets the optimal row header width so that the text and bitmaps of all row headers can be displayed.
MakeRowVisible	Vertically scrolls the specified item into view so that it is displayed in the tree control's client area.
MoveItem	Moves an item to a new position in the tree control.
RecalcHorizontalExtent	Recalculates the optimal horizontal scrolling extent.
RegisterApp	Registers an application with SftTree/DLL.
SetAccessColumn	Sets the current column accessed by column specific messages/functions.
SetBitmaps	Registers the size and sets the default item bitmaps used for all items.
SetButtons	Registers the size and sets the bitmaps used for expand/collapse buttons.
SetCaretIndex	Sets the current item (caret location).
SetCellInfo	Sets a cell's attributes.
SetColumns	Sets the number of columns and their attributes.
SetCtlColors	Sets the tree control's color attributes.
SetDeleteCallback	Sets a deletion callback routine, called when items are deleted.
SetDisableNoScroll	Sets the scroll bar handling status.
SetDragBitmaps	Sets the drag & drop starting location attribute.
SetDragType	Sets the drag & drop detection attribute.
SetDrawInfoCallback	Sets a drawing callback routine. Applications can override certain drawing defaults using this callback.
SetDropHighlight	Sets the current drag & drop target location.
SetDropHighlightStyle	Sets the display attribute of the current drag & drop target item.
SetForwardChildMsgs	Sets the child window message handling status.
SetGridStyle	Sets the grid line display style.
SetHeader	Sets the current column's header text.
SetHeaderButton	Sets the column number of the currently pressed header button.
SetHeaderFont	Sets the font used for header text display.
SetHorizontalExtent	Sets the horizontal extent (in pixels) of the displayable area.
SetHorizontalOffset	Sets the horizontal offset (in pixels) of the displayed area.
SetItemBitmap	Sets an item's item bitmap.
SetItemData	Sets an item's application-specific 32-bit data value.
SetItemExpand	Sets an item's expand status.
SetItemLabel	Sets an item's label bitmap information.
SetItemLevel	Sets an item's level number.
SetItemLines	Sets the number of text lines used for item height calculation.
SetItemShown	Sets an item's visibility status.
SetItemStatus	Sets an item's enabled/disabled status.
SetNoFocusStyle	Sets the display style of selected items when the tree control does not have the input focus.
SetPlusMinus	Set the plus/minus bitmaps used for all items.
SetResizeHeader	Sets a value indicating if columns are resizable by the user.

SetRowColBitmap	Sets the bitmap displayed in the row/column header.
SetRowColBitmapStyle	Sets the position of the bitmap displayed in the row/column header.
SetRowColHeaderButton	Sets a value indicating if the row/column header button is currently down (pressed).
SetRowColHeaderStyle	Sets the position of the text displayed in the row/column header.
SetRowColText	Sets the row/column header text.
SetRowHeaderFont	Sets the font used for row header text display.
SetRowHeaderStyle	Sets the position of text displayed in the row headers.
SetRowHeaderWidth	Sets the width of the row header area.
SetRowInfo	Sets an item's row header attributes.
SetRowText	Sets an item's row header text.
SetSel	Selects or deselects an item (multiple selection tree only).
SetSelIndex	Sets the currently selected item (single selection tree only).
SetSelIndexes	Selects or deselects specified items (multiple selection tree only).
SetSelItemRange	Selects or deselects a range of items (multiple selection tree only).
SetSelTextOnly	Sets the display style of selected items.
SetShow3D	Sets the current display method used for items.
SetShowButton0	Displays or hides level 0 expand/collapse buttons.
SetShowButtons	Displays or hides expand/collapse buttons (other than level 0).
SetShowGrid	Displays or hides the grid lines.
SetShowHeader	Displays or hides the header.
SetShowHeaderButtons	Sets the header's display style.
SetShowRowColHeaderButton	Sets the row/column header's display style.
SetShowRowHeader	Sets the row header display style.
SetShowTreeLines	Displays or hides the tree lines.
SetShowTruncated	Sets a value that determines whether text is clipped or truncated using trailing "...".
SetString	Sets an item's text.
SetTabKeyIntercept	Sets the handling of the TAB key while editing item data.
SetTopIndex	Sets the index number of the item shown at the top of the tree control.
SortDependents	Sorts items.
TSftTree	Standard constructor.
UnregisterApp	Unregisters an application from SftTree/DLL.

OWL/C++ TSftTree Class

The TSftTree class defines all necessary functions to communicate with a tree control. TSftTree is derived from the class TControl.

Application

~TSftTree	Standard destructor.
RegisterApp	Registers an application with SftTree/DLL.
TSftTree	Standard constructor.
UnregisterApp	Unregisters an application from SftTree/DLL.

Callback Routines

SetDeleteCallback	Sets a deletion callback routine, called when items are deleted.
SetDrawInfoCallback	Sets a drawing callback routine. Applications can override certain drawing defaults using this callback.

Columns and Column Headers

GetAccessColumn	Returns the current column accessed by column specific messages/functions.
GetColumns	Returns column information and the number of columns defined.
GetHeader	Returns a column's header text.
GetHeaderButton	Returns the column number of the currently pressed header button.
GetHeaderFont	Returns the font used for header text display.
GetHeaderLen	Returns the length of a column's header text.
GetHeaderRect	Returns the dimensions of the header area.
GetOverheadWidth	Returns the width of the area added to the first column for hierarchical graphics components.
GetResizeHeader	Returns a value indicating if columns are resizable by the user.
GetShowHeader	Returns a value indicating the presence of column headers.
GetShowHeaderButtons	Returns a value indicating the presence of column header buttons.
SetAccessColumn	Sets the current column accessed by column specific messages/functions.
SetColumns	Sets the number of columns and their attributes.
SetHeader	Sets the current column's header text.
SetHeaderButton	Sets the column number of the currently pressed header button.
SetHeaderFont	Sets the font used for header text display.
SetResizeHeader	Sets a value indicating if columns are resizable by the user.
SetShowHeader	Displays or hides the header.
SetShowHeaderButtons	Sets the header's display style.

Current Item

GetCaretColumn	Returns the column where a mouse button was last clicked.
GetCaretIndex	Returns the current item (caret location).
SetCaretIndex	Sets the current item (caret location).

Drag & Drop

GetDragBitmaps	Returns the drag & drop starting location attribute.
GetDragInfo	Returns drag & drop operation information in a SFTTREE_DRAGINFO structure.
GetDragType	Returns the drag & drop detection attribute.
GetDropHighlight	Returns the current drag & drop target location.
GetDropHighlightStyle	Returns the display attribute of the current drag & drop target item.
SetDragBitmaps	Sets the drag & drop starting location attribute.
SetDragType	Sets the drag & drop detection attribute.
SetDropHighlight	Sets the current drag & drop target location.
SetDropHighlightStyle	Sets the display attribute of the current drag & drop target item.

Horizontal Scrolling

GetHorizontalExtent	Returns the last defined horizontal extent (in pixels) of the displayable area.
GetHorizontalOffset	Returns the current horizontal offset (in pixels) of the displayed area.
RecalcHorizontalExtent	Recalculates the optimal horizontal scrolling extent.
SetHorizontalExtent	Sets the horizontal extent (in pixels) of the displayable area.
SetHorizontalOffset	Sets the horizontal offset (in pixels) of the displayed area.

Item Attributes

GetCellInfo	Returns a cell's attributes.
GetEditColumnRect	Returns an item's column location in client area coordinates.
GetItemBitmap	Returns an item's item bitmap.
GetItemData	Returns an item's application-specific 32-bit data value.
GetItemExpand	Returns an item's expand status.
GetItemLabel	Returns an item's label bitmap information.
GetItemLevel	Returns an item's level number.
GetItemRect	Returns an item's location in client area coordinates.
GetItemShown	Returns an item's visibility status.
GetItemStatus	Returns an item's enabled/disabled status.
GetString	Returns an item's text.
GetStringLen	Returns the length of an item's text.
SetCellInfo	Sets a cell's attributes.
SetItemBitmap	Sets an item's item bitmap.
SetItemData	Sets an item's application-specific 32-bit data value.
SetItemExpand	Sets an item's expand status.
SetItemLabel	Sets an item's label bitmap information.
SetItemLevel	Sets an item's level number.
SetItemShown	Sets an item's visibility status.
SetItemStatus	Sets an item's enabled/disabled status.
SetString	Sets an item's text.

Item Hit Testing

CalcIndexFromPoint	Calculates the item index number given a point in tree control client area coordinates.
CalcIndexFromPointEx	Calculates the item index number given a point in tree control client area coordinates.

Items

AddString	Adds a new item at the end of the tree control at level 0 using a string.
ClearList	Removes all items from the tree control.

CopyItem	Copies an item to a new location.
DeleteDependents	Deletes an item's dependent items.
DeleteString	Deletes an item.
FindItem	Searches item data.
FindString	Searches a string.
FindStringExact	Searches a string.
GetCount	Returns the number of items in the tree control.
InsertString	Inserts a new item using a string.
MoveItem	Moves an item to a new position in the tree control.
Positioning & Resizing	
CalcOptimalColumnWidth	Calculates a column's optimal width so text and bitmaps are not clipped.
CalcOptimalRowHeaderWidth	Calculates the optimal width for row headers so text and bitmaps are not clipped.
GetTopIndex	Returns the index number of the item currently shown at the top of the tree control.
MakeColumnOptimal	Sets the optimal column width so that the text and bitmaps of all items can be displayed.
MakeColumnVisible	Horizontally scrolls the specified column into view so that it is displayed in the tree control's client area.
MakeIntegralHeight	Vertically resizes the tree control so visible items are displayed in their entirety.
MakeRowHeaderOptimal	Sets the optimal row header width so that the text and bitmaps of all row headers can be displayed.
MakeRowVisible	Vertically scrolls the specified item into view so that it is displayed in the tree control's client area.
SetTopIndex	Sets the index number of the item shown at the top of the tree control.
Relationships	
GetDependent	Returns dependent item information for a parent item.
GetDependentCount	Returns the number of dependents for a parent item.
GetParent	Returns an item's parent index.
GetSibling	Returns an item's sibling information.
GetTopParent	Returns the highest level parent index number for an item.
Row/Column Header	
GetRowColBitmap	Returns the bitmap displayed in the row/column header.
GetRowColBitmapStyle	Returns the position of the bitmap displayed in the row/column header.
GetRowColHeaderButton	Returns a value indicating if the row/column header button is currently down (pressed).
GetRowColHeaderRect	Returns the dimensions of the row/column header area.
GetRowColHeaderStyle	Returns the position of the text displayed in the row/column header.
GetRowColText	Returns the row/column header text.
GetRowColTextLen	Returns the row/column header text length.
GetShowRowColHeaderButton	Returns the row/column header's display style.
SetRowColBitmap	Sets the bitmap displayed in the row/column header.
SetRowColBitmapStyle	Sets the position of the bitmap displayed in the row/column header.
SetRowColHeaderButton	Sets a value indicating if the row/column header button is currently down (pressed).
SetRowColHeaderStyle	Sets the position of the text displayed in the row/column header.
SetRowColText	Sets the row/column header text.
SetShowRowColHeaderButton	Sets the row/column header's display style.
Row Header	
GetRowHeaderFont	Returns the font used for row header text display.

GetRowHeaderRect	Returns the dimensions of the row header area.
GetRowHeaderStyle	Returns the position of text displayed in the row headers.
GetRowHeaderWidth	Returns the width of the row header area.
GetRowInfo	Returns an item's row header attributes.
GetRowText	Returns an item's row header text.
GetRowTextLen	Returns an item's row header text length.
GetShowRowHeader	Returns the row header display style.
SetRowHeaderFont	Sets the font used for row header text display.
SetRowHeaderStyle	Sets the position of text displayed in the row headers
SetRowHeaderWidth	Sets the width of the row header area.
SetRowInfo	Sets an item's row header attributes.
SetRowText	Sets an item's row header text.
SetShowRowHeader	Sets the row header display style.
Scrolling	
GetDisableNoScroll	Returns the scroll bar handling status.
SetDisableNoScroll	Sets the scroll bar handling status.
Selection	
GetSel	Returns the selection status of an item (multiple selection tree only).
GetSelCount	Returns the number of currently selected items (multiple selection tree only).
GetSelIndex	Returns the index of the selected item (single selection tree control).
GetSelIndexes	Fills an array with the index numbers of currently selected items (multiple selection tree only).
GetSelTextOnly	Returns a value indicating the display style of selected items.
SetSel	Selects or deselects an item (multiple selection tree only).
SetSelIndex	Sets the currently selected item (single selection tree only).
SetSelIndexes	Selects or deselects specified items (multiple selection tree only).
SetSelItemRange	Selects or deselects a range of items (multiple selection tree only).
SetSelTextOnly	Sets the display style of selected items.
Sorting	
SortDependents	Sorts items.
Tree Attributes	
GetCtlColors	Returns the tree control's color attributes.
GetForwardChildMsgs	Returns the child window message handling status.
GetGridStyle	Returns the grid line display style.
GetItemHeight	Returns the current height (in pixels) of each item.
GetItemLines	Returns the number of text lines used for item height calculation.
GetItemsShown	Returns the number of items displayable in the tree control's client area, including partial items.
GetItemsShownComplete	Returns the number of items displayable in the tree control's client area, not including partial items.
GetNoFocusStyle	Returns the display style of selected items when the tree control does not have the input focus.
GetShow3D	Returns a value indicating the current display method used for items.
GetShowBitmaps	Returns a value indicating the presence of item bitmaps.
GetShowButton0	Returns a value indicating the presence of level 0 expand/collapse buttons.
GetShowButtons	Returns a value indicating the presence of expand/collapse buttons (other than level 0).
GetShowGrid	Returns a value indicating the presence of grid lines.
GetShowLabels	Returns a value indicating the presence of label bitmaps.
GetShowPlusMinus	Returns a value indicating the presence of plus/minus bitmaps.
GetShowTreeLines	Returns a value indicating the presence of tree lines.
GetShowTruncated	Returns a value that determines whether text is clipped or truncated

	using trailing "...".
GetTabKeyIntercept	Returns the status of TAB key handling while editing item data.
SetBitmaps	Registers the size and sets the default item bitmaps used for all items.
SetButtons	Registers the size and sets the bitmaps used for expand/collapse buttons.
SetCtlColors	Sets the tree control's color attributes.
SetForwardChildMsgs	Sets the child window message handling status.
SetGridStyle	Sets the grid line display style.
SetItemLines	Sets the number of text lines used for item height calculation.
SetNoFocusStyle	Sets the display style of selected items when the tree control does not have the input focus.
SetPlusMinus	Set the plus/minus bitmaps used for all items.
SetShow3D	Sets the current display method used for items.
SetShowButton0	Displays or hides level 0 expand/collapse buttons.
SetShowButtons	Displays or hides expand/collapse buttons (other than level 0).
SetShowGrid	Displays or hides the grid lines.
SetShowTreeLines	Displays or hides the tree lines.
SetShowTruncated	Sets a value that determines whether text is clipped or truncated using trailing "...".
SetTabKeyIntercept	Sets the handling of the TAB key while editing item data.

Definitions and Structures

Definition	*Description*
SFTTREE_CELL	Cell information
SFTTREE_CELLINFOPARM	Cell information parameter
SFTTREE_CLASS	Window class name
SFTTREE_COLORS	Color information
SFTTREE_COLUMN	Column definition
SFTTREE_COLUMN_EX	Column definition
SFTTREE_DELETEPARM	Deletion callback definition parameter
SFTTREE_DELETEPROC	Deletion callback definition
SFTTREE_DRAGINFO	Drag & drop information
SFTTREE_DRAWINFOPARM	Drawing callback definition parameter
SFTTREE_DRAWINFOPROC	Drawing callback definition
SFTTREE_DRAWINGINFO	Drawing callback information
SFTTREE_NOCOLOR	No color
SFTTREE_ROW	Row header information
SFTTREE_ROWINFOPARM	Row header information parameter
SFTTREE_SORTPROC	Sorting callback definition
SFTTREE_SORTPROCEX	Sorting callback definition

Topic not available

This help topic is not available with the demo version of SftTree/DLL.
You will receive a 250 page manual and complete on-line help when purchasing SftTree/DLL.
SftTree/DLL supports C, C++ and other DLL-call capable languages. Special class implementations for MFC and OWL are included. The source code for the DLL is also available.

Simple Sample

The Simple sample displays one tree control as a child window of a frame window (non-dialog window) and demonstrates how to implement a tree control with item data editing and drag & drop handling.

Click here to run the Simple sample application. If the sample cannot be started, please reinstall the SftTree/DLL demo.

The example source code can be found in the following directories:

Language	***Directory***
C	C:\SftTree\Samples\C\Simple
C++ & MFC	C:\SftTree\Samples\VC\Simple
C++ & OWL	C:\SftTree\Samples\BC\Simple

Relevant portions of the source code are included here for easy access to the on-line help information.

C, Simple Sample

The complete source code can be found in the directory C:\SftTree\Samples\C\Simple.

```
/*********************************************************************/
/*                        Tree Edit Routines                         */
/*********************************************************************/

/* These routines handle item data editing. In this        */
/* example, an edit control is used. Any Windows control   */
/* can be used, even custom controls.                      */

/* Start editing in response to a                          */
/* SFTTREEN_LBUTTONDBLCLK_TEXT notification.               */

int m_editIndex;                        /* Index of item being edited */
int m_editCol;                          /* Column # being edited */
HWND m_hwndEdit;                        /* Control used for editing */

static void StartEdit(void)
{
    char szBuffer[80];
    LPRECT lpRect;

    /* Get the item index where double-click happened */
    m_editIndex = SftTree_GetCaretIndex(g_hwndTree); /* Save item # */
    m_editCol = -1;                      /* Let SftTree provide column # */
    lpRect = SftTree_GetEditColumnRect(g_hwndTree, m_editIndex, &m_editCol);
    if (!lpRect)                         /* No column active */
        return;
    /* Make the column completely visible */
    SftTree_MakeColumnVisible(g_hwndTree, m_editCol);
    /* Make the item completely visible */
    SftTree_MakeRowVisible(g_hwndTree, m_editIndex);
    /* Get the location again (in case it scrolled) */
    lpRect = SftTree_GetEditColumnRect(g_hwndTree, m_editIndex, &m_editCol);
    if (!lpRect)                         /* No column active */
        return;

    /* Change selection style to not shown anything */
    SftTree_SetNoFocusStyle(g_hwndTree, SFTTREE_NOFOCUS_NOTHING);

    /* Repaint now in case we scrolled to make item visible */
    UpdateWindow(g_hwndTree);

    /* Create the edit control.*/
    /* Based on the tree control attributes and your preference, you may */
    /* have to adjust the rectangle used for the edit control. */

    m_hwndEdit = CreateWindow(TEXT("EDIT"), TEXT(""),
                                        /* Class, Title */
            WS_CHILD|WS_BORDER|ES_LEFT|ES_AUTOHSCROLL, /* Styles */
            lpRect->left, lpRect->top, lpRect->right-lpRect->left, lpRect->bottom-
lpRect->top,
            g_hwndTree,                 /* Tree control */
            (HMENU) IDC_EDIT,           /* <-- provide unique ID */
            g_hInst,                    /* <-- provide instance handle */
            NULL);
    if (!m_hwndEdit)                    /* Failed */
        return;

    /* Set some edit control attributes */
```

```
    /* Set the font defined for item data editing */
    SendMessage(m_hwndEdit, WM_SETFONT, (WPARAM)(UINT)m_hEditFont, 0L);

    /* Copy the text found in the tree control */
    SftTree_GetTextCol(g_hwndTree, m_editIndex, m_editCol, szBuffer);
    SetWindowText(m_hwndEdit, szBuffer);
    /* Select all text in the edit control and display it */
#if defined(_WIN32) || defined(__WIN32__)
    SendMessage(m_hwndEdit, EM_SETSEL, 0, -1L);
#else
    SendMessage(m_hwndEdit, EM_SETSEL, TRUE, MAKELPARAM(0, -1));
#endif
    ShowWindow(m_hwndEdit, SW_SHOW);
    SetFocus(m_hwndEdit);               /* Set input focus to the control */
}

/* Quit editing in response to a SFTTREEN_QUITEDIT         */
/* notification.                                           */

static void QuitEdit(void)
{
    if (m_hwndEdit) {
        /* If the control has the focus, set the focus back to */
        /* tree control after destroying the control */

        BOOL fHadFocus = (GetFocus() == m_hwndEdit);

        DestroyWindow(m_hwndEdit);
        m_hwndEdit = NULL;
        /* Restore nofocus display method */
        SftTree_SetNoFocusStyle(g_hwndTree, SFTTREE_NOFOCUS_FRAME);
        if (fHadFocus)
            SetFocus(g_hwndTree);       /* Back to tree control */
    }
}

/* Validate edit data in response to a                     */
/* SFTTREEN_VALIDATEEDIT notification.                     */

static void ValidateEdit(void)
{
    if (m_hwndEdit) {
        char szBuffer[80];

        /* Get the text from the edit control */
        GetWindowText(m_hwndEdit, szBuffer, sizeof(szBuffer));

        /* Validate the data */
        if (lstrcmp(TEXT(""), szBuffer) == 0) {
            MessageBox(NULL, TEXT("Just to demonstrate data input validation, this
example rejects empty cells.  Please enter some data."),
                       TEXT("SftTree/DLL"), MB_OK|MB_TASKMODAL|MB_ICONSTOP);
            SetFocus(m_hwndEdit);
        } else {
            /* Save the data in the tree control */
            SftTree_SetTextCol(g_hwndTree, m_editIndex, m_editCol, szBuffer);
            DestroyWindow(m_hwndEdit);
            m_hwndEdit = NULL;
            /* Restore nofocus display method */
            SftTree_SetNoFocusStyle(g_hwndTree, SFTTREE_NOFOCUS_FRAME);
        }
    }
}
```

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

LRESULT CALLBACK
#if !defined(_WIN32) || defined(__BORLANDC__)
            _export
#endif
                SDI_WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
    switch (msg) {

    case WM_CREATE: {
        int height;                         /* Height in pixels */
        HDC hDC;                            /* Device context */
        int index;

        g_hwndTree = CreateWindowEx(
#if defined(WS_EX_CLIENTEDGE)
            WS_EX_CLIENTEDGE,
#else
            0x00000200L,
#endif
            TEXT(SFTTREE_CLASS),            /* Window class */
            TEXT(""),                       /* Caption (none) */
            SFTTREESTYLE_LEFTBUTTONONLY |   /* Only respond to left mouse button */
            SFTTREESTYLE_NOTIFY |           /* Notify parent window */
            SFTTREESTYLE_DRAGDROP |         /* Drag & drop enabled */
            SFTTREESTYLE_SCROLL |           /* Honor WS_H/VSCROLL */
            WS_BORDER |                     /* Border */
            WS_HSCROLL | WS_VSCROLL |       /* Vertical and horizontal scrollbars
*/
            WS_VISIBLE | WS_CHILD,          /* Visible, child window */
            0, 0, 0, 0,                     /* Location */
            hwnd,                           /* Parent window */
            (HMENU) IDC_TREE,               /* Tree control ID */
            g_hInst,                        /* Application instance */
            NULL);
        if (!g_hwndTree)
            return -1;

        /* Bitmaps should be loaded using LoadBitmap and must remain valid */
        /* until the tree control is destroyed or no longer uses the       */
        /* bitmaps.  Use DeleteObject to delete the bitmaps.               */
        /* All bitmaps used are of type HBITMAP.                           */

        SftTree_SetShowHeader(g_hwndTree, TRUE);
                                            /* Show column headers */

        /* Register the plus/minus bitmaps.  These bitmaps are     */
        /* used for all items in the tree control. All three       */
        /* bitmaps must be the same size. You cannot override      */
        /* the plus/minus bitmap for individual items.             */

        m_ahThreePlusMinBitmaps[0] = LoadBitmap(g_hInst, MAKEINTRESOURCE(IDB_PLUS));
                                            /* Expandable bitmap */
        m_ahThreePlusMinBitmaps[1] = LoadBitmap(g_hInst,
MAKEINTRESOURCE(IDB_MINUS));
                                            /* Expanded bitmap */
        m_ahThreePlusMinBitmaps[2] = LoadBitmap(g_hInst, MAKEINTRESOURCE(IDB_NONE));
                                            /* Leaf bitmap */
        SftTree_SetPlusMinus(g_hwndTree, m_ahThreePlusMinBitmaps);
```

```
                                        /* Use +/- bitmaps */

        /* Register the item bitmaps.  These bitmaps are used for  */
        /* all items in the tree control. All three bitmaps must   */
        /* be the same size.                                       */

        m_ahThreeItemBitmaps[0] = LoadBitmap(g_hInst,
MAKEINTRESOURCE(IDB_EXPANDABLE));
                                        /* Expandable bitmap */
        m_ahThreeItemBitmaps[1] = LoadBitmap(g_hInst,
MAKEINTRESOURCE(IDB_EXPANDED));
                                        /* Expanded bitmap */
        m_ahThreeItemBitmaps[2] = LoadBitmap(g_hInst, MAKEINTRESOURCE(IDB_LEAF));
                                        /* Leaf bitmap */
        SftTree_SetBitmaps(g_hwndTree, m_ahThreeItemBitmaps);
                                        /* Use item bitmaps */

        /* Register the cell bitmap size.  All cell bitmaps used   */
        /* must be the same size.  Only one bitmap needs to be     */
        /* registered, even if several are used.                   */

        {
            SFTTREE_CELLINFOPARM CellInfo;
            CellInfo.version = 2;           /* Mandatory, using version 2 format */
            CellInfo.index = -1;            /* Registering bitmap size */
            m_hCellBitmap = LoadBitmap(g_hInst, MAKEINTRESOURCE(IDB_SMILE1));
                                        /* Cell bitmap */
            CellInfo.Cell.hBmp = m_hCellBitmap;
            SftTree_SetCellInfo(g_hwndTree, &CellInfo);          /* Register use of
cell bitmaps */
        }

        SftTree_SetShowGrid(g_hwndTree, TRUE);
                                        /* Show grid */

        SftTree_SetShowTruncated(g_hwndTree, TRUE);
                                        /* Show ... if truncated */

        SftTree_SetNoFocusStyle(g_hwndTree, SFTTREE_NOFOCUS_FRAME);
                                        /* No focus, show frame */

        SftTree_SetSelTextOnly(g_hwndTree, TRUE);
                                        /* Select text only */

        SftTree_SetShowHeaderButtons(g_hwndTree, TRUE);
                                        /* Show column header as buttons */

        /* Define columns */

        {
            SFTTREE_COLUMN_EX aCol[2] = {
              { 0, 0,                   /* Reserved */
                100,                    /* Width (in pixels) */
                ES_LEFT,                /* Cell alignment */
                ES_LEFT,                /* Title style */
                TEXT("First Column"),   /* Column header title */
                NULL, NULL,             /* Reserved field and bitmap handle */
                SFTTREE_BMP_RIGHT },    /* Bitmap alignment */
              { 0, 0,                   /* Reserved */
                -1,                     /* Width (in pixels) */
                ES_LEFT,                /* Cell alignment */
                ES_LEFT,                /* Title style */
                TEXT("Second Column"),  /* Column header title */
```

```
            NULL, NULL,                   /* Reserved field and bitmap handle */
            SFTTREE_BMP_RIGHT }           /* Bitmap alignment */
        };
        SftTree_SetColumnsEx(g_hwndTree, 2, aCol);
                                          /* Set column attributes */
    }

    /* Set the row header attributes.                          */

    SftTree_SetShowRowHeader(g_hwndTree, SFTTREE_ROWSTYLE_BUTTONCOUNT1);
                                          /* Row style */

    /* Set the row/column header attributes                    */

    SftTree_SetRowColText(g_hwndTree, TEXT("R/C"));
                                          /* Row/column header text */

    SftTree_SetRowColHeaderStyle(g_hwndTree, ES_LEFT | SFTTREE_HEADER_UP);
                                          /* Row/column header style */

    /* Set the row/column header bitmap */

    m_hRowColBitmap = LoadBitmap(g_hInst, MAKEINTRESOURCE(IDB_SMILE1));
                                          /* Row/column header bitmap */
    SftTree_SetRowColBitmap(g_hwndTree, m_hRowColBitmap);
                                          /* Row/column bitmap */

    /* Set the row/column header bitmap alignment */

    SftTree_SetRowColBitmapStyle(g_hwndTree, SFTTREE_BMP_RIGHT);
                                          /* Row/column bitmap alignment */

    SftTree_SetDragBitmaps(g_hwndTree, TRUE);
                                          /* Allow drag & drop from bitmaps */

    SftTree_SetDragType(g_hwndTree, SFTTREE_DRAG_PIXELIMM);
                                          /* Select and move by a number of
pixels to start drag */

    SftTree_SetDropHighlightStyle(g_hwndTree, SFTTREE_DROPHIGHLIGHT_ONTOP);
                                          /* Highlight drop target */



/*--------------------------------------------------------------------------*/
    /* Add a few items.
*/

/*--------------------------------------------------------------------------*/

    index = SftTree_AddString(g_hwndTree, TEXT("An item"));
                                          /* Add an item */
    index = SftTree_AddString(g_hwndTree, TEXT("Another item"));
                                          /* Add another item */
    SftTree_SetItemLevel(g_hwndTree, index, 1);
                                          /* change level */
    SftTree_SetTextCol(g_hwndTree, index, 1, TEXT("2nd Column"));
                                          /* set text in next column */
    SftTree_SetRowText(g_hwndTree, index, TEXT("Second"));
                                         /* Row header text */
    index = SftTree_AddString(g_hwndTree, TEXT("A third item"));
                                          /* Add another item */
    SftTree_SetItemLevel(g_hwndTree, index, 2);
```

```
                                        /* change level */
        SftTree_SetTextCol(g_hwndTree, index, 1, TEXT("2nd Column"));
                                        /* set text in next column */
        index = SftTree_AddString(g_hwndTree, TEXT("A fourth item"));
                                        /* Add another item */
        SftTree_SetItemLevel(g_hwndTree, index, 1);
                                        /* change level */
        SftTree_SetTextCol(g_hwndTree, index, 1, TEXT("2nd Column"));
                                        /* set text in next column */
        /* Set a cell bitmap */
        {
            SFTTREE_CELLINFOPARM CellInfo;
            CellInfo.version = 2;       /* Mandatory, using version 2 format */
            CellInfo.index = index;
            CellInfo.iCol = 1;
            SftTree_GetCellInfo(g_hwndTree, &CellInfo);
            CellInfo.Cell.hBmp = m_hCellBitmap;
            SftTree_SetCellInfo(g_hwndTree, &CellInfo);
        }

        /* Make all column widths optimal, so text and bitmaps are */
        /* not clipped horizontally.                               */

        SftTree_MakeColumnOptimal(g_hwndTree, -1);
                                        /* Make column widths optimal */

        /* Make row header width optimal, so text and bitmaps are  */
        /* not clipped horizontally.                               */

        SftTree_MakeRowHeaderOptimal(g_hwndTree);
                                        /* Make row header width optimal */

        SftTree_RecalcHorizontalExtent(g_hwndTree);
                                        /* Update horizontal scroll bar */

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

        hDC = GetDC(NULL);              /* Get a device context */

        /* Create the font to be used for item data editing.       */
        /* Delete the font (DeleteObject) once the tree control no */
        /* longer uses it.                                         */

        height = MulDiv(8, GetDeviceCaps(hDC, LOGPIXELSY), 72);
                                        /* Convert point-size to pixels */
        m_hEditFont = CreateFont(-height, 0, 0, 0, FW_NORMAL, 0, 0, 0, 0, 0, 0, 0,
0, TEXT("MS Sans Serif"));

        ReleaseDC(NULL, hDC);           /* Release device context */

        return 0;
     }
    case WM_DESTROY:
        QuitEdit();

        DeleteObject(m_ahThreePlusMinBitmaps[0]);/* Plus/minus bitmaps */
        DeleteObject(m_ahThreePlusMinBitmaps[1]);
        DeleteObject(m_ahThreePlusMinBitmaps[2]);

        DeleteObject(m_ahThreeItemBitmaps[0]);/* Default item bitmaps */
        DeleteObject(m_ahThreeItemBitmaps[1]);
```

```
        DeleteObject(m_ahThreeItemBitmaps[2]);
        DeleteObject(m_hOtherItemBitmap);    /* Another item bitmap */

        DeleteObject(m_hCellBitmap);         /* A cell bitmap */

        DeleteObject(m_hRowColBitmap);       /* Row/column bitmap */

        /* Delete the fonts used for the tree control.             */

        DeleteObject(m_hEditFont);           /* Item data editing font */

        if (g_hwndTree)
            DestroyWindow(g_hwndTree);
        PostQuitMessage(0);
        break;

    case WM_SIZE: {
        int cx = LOWORD(lParam);
        int cy = HIWORD(lParam);
        SetWindowPos(g_hwndTree, NULL, 0, 0, cx, cy, SWP_NOACTIVATE | SWP_NOZORDER);
        return 0;
     }

#if defined(WM_CONTEXTMENU)
    case WM_CONTEXTMENU:
        if (m_hwndEdit)
            break;                       // The message is for the edit control
        MessageBox(NULL, "Received WM_CONTEXTMENU message from the tree control.
You could bring up a context menu.",
            "WM_CONTEXTMENU", MB_OK| MB_TASKMODAL);
        break;
#endif

    case WM_SETFOCUS:
        if (g_hwndTree)
            SetFocus(g_hwndTree);
        break;

    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
        switch (id) {
        case IDC_TREE:
            switch(code) {
            case SFTTREEN_LBUTTONDOWN_BUTTON:
            case SFTTREEN_LBUTTONDBLCLK_BUTTON: {
                int index;
                BOOL fExpand, fControl;
                /* Get current position */
                index = SftTree_GetCaretIndex(hwndCtl);/* Get caret location */
                /* Check if item is expanded */
                fExpand = SftTree_GetItemExpand(hwndCtl, index);
                /* If the CONTROL key is pressed, expand all dependent levels */
                fControl = (BOOL)(GetKeyState(VK_CONTROL)&0x8000);
                /* Do the opposite */
                SftTree_SetItemExpand(hwndCtl, index, !fExpand, fControl);
```

```
                break;
              }
            case SFTTREEN_LBUTTONDBLCLK_TEXT:
                /* Edit the current cell */
                StartEdit();
                break;
            case SFTTREEN_QUITEDIT:
                /* Abandon editing */
                QuitEdit();
                break;
            case SFTTREEN_VALIDATEEDIT:
                /* Validate input data */
                ValidateEdit();
                break;
            case SFTTREEN_CANCELDRAG:
                SftTree_SetDropHighlight(g_hwndTree, -1, FALSE);
                                        /* Visually clear drop target */
                break;
            case SFTTREEN_ENDDRAG: {
                /* The user dropped something */
                int index;
                index = SftTree_GetDropHighlight(g_hwndTree);
                                        /* Drop target */
                SftTree_SetDropHighlight(g_hwndTree, -1, FALSE);
                                        /* Visually clear drop target */
                /* "index" now contains the drop target. */
                /* All selected items participate in the drag & drop operation */
                /* ... Application specific action goes here ... */
                break;
              }
            }
            break;

        case 1000:                      // Menu command, Exit
            DestroyWindow(hwnd);
            break;

        case IDCANCEL:                  // ESC has hit while editing
            QuitEdit();
            break;
        }
        break;
     }
    }
    return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

C++/MFC, Simple Sample

The complete source code can be found in the directory C:\SftTree\Samples\VC\Simple.

```
#define IDC_EDIT 101

/////////////////////////////////////////////////////////////////////////////
// CSampleView

IMPLEMENT_DYNCREATE(CSampleView, CView)

BEGIN_MESSAGE_MAP(CSampleView, CView)
    //{{AFX_MSG_MAP(CSampleView)
    ON_WM_CREATE()
    ON_WM_SIZE()
    //}}AFX_MSG_MAP
    ON_SFTTREEN_LBUTTONDBLCLK_TEXT(IDC_TREE, OnStartEdit)
    ON_SFTTREEN_QUITEDIT(IDC_TREE, OnQuitEdit)
    ON_SFTTREEN_VALIDATEEDIT(IDC_TREE, OnValidateEdit)
    ON_SFTTREEN_LBUTTONDOWN_BUTTON(IDC_TREE, OnLButtonExpandCollapse)
    ON_SFTTREEN_LBUTTONDBLCLK_BUTTON(IDC_TREE, OnLButtonExpandCollapse)
    ON_SFTTREEN_ENDDRAG(IDC_TREE, OnEndDrag)
    ON_SFTTREEN_BEGINDRAG(IDC_TREE, OnDragging)
    ON_SFTTREEN_DRAGGING(IDC_TREE, OnDragging)
    ON_SFTTREEN_CANCELDRAG(IDC_TREE, OnCancelDrag)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSampleView construction/destruction

CSampleView::CSampleView()
{
    m_pEdit = NULL;                     /* No item data editing */
}

CSampleView::~CSampleView()
{
}

/////////////////////////////////////////////////////////////////////////////
// CSampleView message handlers

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

    if (!m_Tree.CreateEx(
#if defined(WS_EX_CLIENTEDGE)
        WS_EX_CLIENTEDGE,
#else
        0x00000200L,
#endif
        SFTTREESTYLE_NOTIFY |           /* Notify parent window */
        SFTTREESTYLE_DRAGDROP |         /* Drag & drop enabled */
        SFTTREESTYLE_LEFTBUTTONONLY |   /* Only respond to left mouse button */
        SFTTREESTYLE_SCROLL |           /* Honor WS_H/VSCROLL */
        WS_BORDER |                     /* Border */
        WS_HSCROLL | WS_VSCROLL |       /* Vertical and horizontal scrollbars */
        WS_VISIBLE | WS_CHILD,          /* Visible, child window */
        CRect(0,0,0,0),                 /* Location */
        this,                           /* Parent window */
        IDC_TREE))                      /* Tree control ID */
```

```
        return -1;

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

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

    hDC = ::GetDC(NULL);                /* Get a device context */

    /* Create the font to be used for item data editing.       */
    /* Delete the font (DeleteObject) once the tree control no */
    /* longer uses it.                                         */

    height = MulDiv(8, ::GetDeviceCaps(hDC, LOGPIXELSY), 72);
                                        /* Convert point-size to pixels */
    m_EditFont.CreateFont(-height, 0, 0, 0, FW_BOLD, 0, 0, 0, 0, 0, 0, 0, _T("MS
Sans Serif"));

    ::ReleaseDC(NULL, hDC);             /* Release device context */

    /* Bitmaps should be loaded using LoadBitmap and must remain valid */
    /* until the tree control is destroyed or no longer uses the       */
    /* bitmaps. All bitmaps used are of type CBitmap.                  */

    m_Tree.SetShowHeader(TRUE);         /* Show column headers */

    /* Register the plus/minus bitmaps.  These bitmaps are     */
    /* used for all items in the tree control. All three       */
    /* bitmaps must be the same size. You cannot override      */
    /* the plus/minus bitmap for individual items.             */

    m_aThreePlusMinBitmaps[0].LoadBitmap(IDB_PLUS);
                                        /* Expandable bitmap */
    m_aThreePlusMinBitmaps[1].LoadBitmap(IDB_MINUS);
                                        /* Expanded bitmap */
    m_aThreePlusMinBitmaps[2].LoadBitmap(IDB_NONE);
                                        /* Leaf bitmap */
    m_Tree.SetPlusMinus(m_aThreePlusMinBitmaps);
                                        /* Use +/- bitmaps */

    /* Register the item bitmaps.  These bitmaps are used for  */
    /* all items in the tree control. All three bitmaps must   */
    /* be the same size.                                       */

    m_aThreeItemBitmaps[0].LoadBitmap(IDB_EXPANDABLE);
                                        /* Expandable bitmap */
    m_aThreeItemBitmaps[1].LoadBitmap(IDB_EXPANDED);
                                        /* Expanded bitmap */
    m_aThreeItemBitmaps[2].LoadBitmap(IDB_LEAF);
                                        /* Leaf bitmap */
    m_Tree.SetBitmaps(m_aThreeItemBitmaps);
                                        /* Use item bitmaps */

    /* Register the cell bitmap size.  All cell bitmaps used   */
    /* must be the same size.  Only one bitmap needs to be     */
    /* registered, even if several are used.                   */

    {
        SFTTREE_CELLINFOPARM CellInfo;
        CellInfo.version = 2;           /* Mandatory, using version 2 format */
        CellInfo.index = -1;            /* Registering bitmap size */
```

```
    m_CellBitmap.LoadBitmap(IDB_SMILE);
                                    /* Cell bitmap */
    CellInfo.Cell.hBmp = (HBITMAP) m_CellBitmap.m_hObject;
    m_Tree.SetCellInfo(&CellInfo);   /* Register use of cell bitmaps */
}

m_Tree.SetShowGrid(TRUE);           /* Show grid */

m_Tree.SetShowTruncated(TRUE);      /* Show ... if truncated */

m_Tree.SetNoFocusStyle(SFTTREE_NOFOCUS_FRAME);
                                    /* No focus, show frame */

m_Tree.SetSelTextOnly(TRUE);        /* Select text only */

m_Tree.SetShowHeaderButtons(TRUE);  /* Show column header as buttons */

/* Define columns */

{
    SFTTREE_COLUMN_EX aCol[2] = {
      { 0, 0,                       /* Reserved */
        100,                        /* Width (in pixels) */
        ES_LEFT,                    /* Cell alignment */
        ES_LEFT,                    /* Title style */
        _T("First Column"),         /* Column header title */
        NULL, NULL,                 /* Reserved field and bitmap handle */
        SFTTREE_BMP_RIGHT },        /* Bitmap alignment */
      { 0, 0,                       /* Reserved */
        -1,                         /* Width (in pixels) */
        ES_LEFT,                    /* Cell alignment */
        ES_LEFT,                    /* Title style */
        _T("Second Column"),        /* Column header title */
        NULL, NULL,                 /* Reserved field and bitmap handle */
        SFTTREE_BMP_RIGHT }         /* Bitmap alignment */
    };
    m_Tree.SetColumns(2, aCol);     /* Set column attributes */
}

/* Set the row header attributes.                          */

m_Tree.SetShowRowHeader(SFTTREE_ROWSTYLE_BUTTONCOUNT1);
                                    /* Row style */

/* Set the row/column header attributes                    */

m_Tree.SetRowColText(_T("?"));      /* Row/column header text */

m_Tree.SetRowColHeaderStyle(ES_LEFT | SFTTREE_HEADER_UP);
                                    /* Row/column header style */

/* Set the row/column header bitmap */

m_RowColBitmap.LoadBitmap(IDB_SMILE);
                                    /* Row/column header bitmap */
m_Tree.SetRowColBitmap(m_RowColBitmap);
                                    /* Row/column bitmap */

/* Set the row/column header bitmap alignment */

m_Tree.SetRowColBitmapStyle(SFTTREE_BMP_RIGHT);
                                    /* Row/column bitmap alignment */
```

```
    m_Tree.SetDragBitmaps(TRUE);         /* Allow drag & drop from bitmaps */

    m_Tree.SetDragType(SFTTREE_DRAG_PIXELIMM);
                                         /* Select and move by a number of pixels to
start drag */

    m_Tree.SetDropHighlightStyle(SFTTREE_DROPHIGHLIGHT_ONTOP);
                                         /* Highlight drop target */

    /*-------------------------------------------------------------------------
*/
    /* Add a few items.
*/
    /*-------------------------------------------------------------------------
*/

    int index;

    index = m_Tree.AddString(_T("An item"));
                                         /* Add an item */
    index = m_Tree.AddString(_T("Another item"));
                                         /* Add another item */
    m_Tree.SetItemLevel(index, 1);       /* change level */
    m_Tree.SetText(index, 1, _T("2nd Column"));
                                         /* set text in next column */
    /* Set an item's row header text using: */
    m_Tree.SetRowText(index, _T("Second"));/* Row header text */

    index = m_Tree.AddString(_T("A third item"));
                                         /* Add another item */
    m_Tree.SetItemLevel(index, 2);       /* change level */
    m_Tree.SetText(index, 1, _T("2nd Column"));
                                         /* set text in next column */
    index = m_Tree.AddString(_T("A fourth item"));
                                         /* Add another item */
    m_Tree.SetItemLevel(index, 1);       /* change level */
    m_Tree.SetText(index, 1, _T("2nd Column"));
                                         /* set text in next column */
    /* Set a cell bitmap */
    SFTTREE_CELLINFOPARM CellInfo;
    CellInfo.version = 2;               /* Mandatory, using version 2 format */
    CellInfo.index = index;
    CellInfo.iCol = 1;
    m_Tree.GetCellInfo(&CellInfo);
    CellInfo.Cell.hBmp = (HBITMAP) m_CellBitmap.m_hObject;
    m_Tree.SetCellInfo(&CellInfo);

    /*-------------------------------------------------------------------------
*/
    /* Once ALL TREE CONTROL ITEMS HAVE BEEN ADDED, you can set additional tree
*/
    /* control attributes.
*/
    /*-------------------------------------------------------------------------
*/

    /* Make all column widths optimal, so text and bitmaps are */
    /* not clipped horizontally.                               */

    m_Tree.MakeColumnOptimal(-1);        /* Make column widths optimal */

    /* Make row header width optimal, so text and bitmaps are  */
    /* not clipped horizontally.                               */
```

```
    m_Tree.MakeRowHeaderOptimal();       /* Make row header width optimal */

    m_Tree.RecalcHorizontalExtent();     /* Update horizontal scroll bar */

    return 0;
}

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

    m_Tree.MoveWindow(0, 0, cx, cy);
}


/* These routines handle item data editing. In this        */
/* example, an edit control is used. Any Windows control   */
/* can be used, even custom controls.                      */

/* Start editing in response to a                          */
/* SFTTREEN_LBUTTONDBLCLK_TEXT notification.               */

void CSampleView::OnStartEdit()
{
    CString str;
    LPCRECT lpRect;

    /* Get the item index where double-click happened */
    m_editIndex = m_Tree.GetCaretIndex(); /* Save item # */
    m_editCol = -1;                       /* Let SftTree/DLL provide column # */
    lpRect = m_Tree.GetEditColumnRect(m_editIndex, &m_editCol);
    if (!lpRect)                          /* No column active */
        return;
    /* Make the column completely visible */
    m_Tree.MakeColumnVisible(m_editCol);
    /* Make the item completely visible */
    m_Tree.MakeRowVisible(m_editIndex);
    /* Get the location again (in case it scrolled) */
    lpRect = m_Tree.GetEditColumnRect(m_editIndex, &m_editCol);
    if (!lpRect)                          /* No column active */
        return;

    /* Change selection style to not shown anything */
    m_Tree.SetNoFocusStyle(SFTTREE_NOFOCUS_NOTHING);

    /* Repaint now in case we scrolled to make item visible */
    m_Tree.UpdateWindow();

    /* Create the edit control.*/
    /* Based on the tree control attributes and your preference, you may */
    /* have to adjust the rectangle used for the edit control. */

    m_pEdit = new CEdit;
    m_pEdit->Create(
            WS_CHILD|WS_BORDER|ES_LEFT|ES_AUTOHSCROLL, /* Styles */
            (CRect) lpRect,
            &m_Tree,                    /* Tree control */
            IDC_EDIT);                  /* <-- provide unique ID */

    /* Set some edit control attributes */

    /* Set the font defined for item data editing */
```

```
    m_pEdit->SetFont(&m_EditFont, FALSE);

    /* Copy the text found in the tree control */
    m_Tree.GetText(m_editIndex, m_editCol, str);
    m_pEdit->SetWindowText(str);
    /* Select all text in the edit control and display it */
    m_pEdit->SetSel(0, -1, TRUE);
    m_pEdit->ShowWindow(SW_SHOW);
    m_pEdit->SetFocus();                /* Set input focus to the control */
}

/* Quit editing in response to a SFTTREEN_QUITEDIT         */
/* notification.                                           */

void CSampleView::OnQuitEdit()
{
    if (m_pEdit) {
        /* If the control has the focus, set the focus back to */
        /* tree control after destroying the control */

        BOOL fHadFocus = (GetFocus() == m_pEdit);

        m_pEdit->DestroyWindow();
        delete m_pEdit;
        m_pEdit = NULL;
        /* Restore nofocus display method */
        m_Tree.SetNoFocusStyle(SFTTREE_NOFOCUS_FRAME);
        if (fHadFocus)
            m_Tree.SetFocus();          /* Back to tree control */
    }
}

/* Validate edit data in response to a                     */
/* SFTTREEN_VALIDATEEDIT notification.                     */

void CSampleView::OnValidateEdit()
{
    if (m_pEdit) {
        CString str;

        /* Get the text from the edit control */
        m_pEdit->GetWindowText(str);

        /* Validate the data */
        if (str == _T("")) {
            AfxMessageBox(_T("Just to demonstrate data input validation, this
example rejects empty cells.  Please enter some data."));
            m_pEdit->SetFocus();
        } else {
            /* Save the data in the tree control */
            m_Tree.SetText(m_editIndex, m_editCol, str);
            m_pEdit->DestroyWindow();
            delete m_pEdit;
            m_pEdit = NULL;
            /* Restore nofocus display method */
            m_Tree.SetNoFocusStyle(SFTTREE_NOFOCUS_FRAME);
        }
    }
}

/* Respond to expand/collapse requests as the user clicks  */
/* on different tree components.  The events handled here  */
/* can be changed to suit your application.                */
```

```
void CSampleView::OnLButtonExpandCollapse()
{
    /* get index of item to expand/collapse */
    int index = m_Tree.GetCaretIndex();
    /* get current expand/collapsed status */
    BOOL fExpanded = m_Tree.GetItemExpand(index);
    /* if control key is used we'll expand all dependents */
    BOOL fDepth = (::GetKeyState(VK_CONTROL)&0x8000);

    m_Tree.SetItemExpand(index, !fExpanded, fDepth);
}

/* Drag and drop in ONE tree control is implemented very   */
/* easily by handling the SFTTREEN_ENDDRAG notification.   */
/* You can also respond to the SFTTREEN_BEGINDRAG and      */
/* SFTTREEN_DRAGGING notifications to override the drag    */
/* cursor or to cancel drag & drop if desired.             */

/* For multi-tree drag & drop see the DragDrop sample.     */

void CSampleView::OnEndDrag()
{
    /* The user dropped something */
    int index;
    index = m_Tree.GetDropHighlight();   /* Drop target */
    m_Tree.SetDropHighlight(-1, FALSE);  /* Visually clear drop target */
    /* "index" now contains the drop target. */
    /* All selected items participate in the drag & drop operation */
    /* ... Application specific action goes here ... */
}

void CSampleView::OnDragging()
{
    /* Drag & drop in progress */
    /* Here we handle this event to override the default */
    /* drag cursor.  This is completely optional. */
    LPSFTTREE_DRAGINFO lpInfo;
    lpInfo = m_Tree.GetDragInfo();
}

void CSampleView::OnCancelDrag()
{
    m_Tree.SetDropHighlight(-1, FALSE);  /* Visually clear drop target */
}
```

C++/OWL, Simple Sample

The complete source code can be found in the directory C:\SftTree\Samples\BC\Simple.

```
#define IDC_EDIT 101

//
// Build a response table for all messages/commands handled
// by the application.
//
DEFINE_RESPONSE_TABLE1(TSampleWindow, TWindow)
//{{TSampleWindowRSP_TBL_BEGIN}}
    EV_WM_SIZE,
    EV_WM_CREATE,
    EV_WM_DESTROY,
//{{TSampleWindowRSP_TBL_END}}
    EV_SFTTREEN_LBUTTONDBLCLK_TEXT(IDC_TREE, EvStartEdit),
    EV_SFTTREEN_QUITEDIT(IDC_TREE, EvQuitEdit),
    EV_SFTTREEN_VALIDATEEDIT(IDC_TREE, EvValidateEdit),
    EV_SFTTREEN_LBUTTONDOWN_BUTTON(IDC_TREE, EvLButtonExpandCollapse),
    EV_SFTTREEN_LBUTTONDBLCLK_BUTTON(IDC_TREE, EvLButtonExpandCollapse),
    EV_SFTTREEN_ENDDRAG(IDC_TREE, EvEndDrag),
    EV_SFTTREEN_BEGINDRAG(IDC_TREE, EvDragging),
    EV_SFTTREEN_DRAGGING(IDC_TREE, EvDragging),
    EV_SFTTREEN_CANCELDRAG(IDC_TREE, EvCancelDrag),
END_RESPONSE_TABLE;


//{{TSampleWindow Implementation}}


//////////////////////////////////////////////////////
// TSampleWindow
// ==========
// Construction/Destruction handling.
TSampleWindow::TSampleWindow (TWindow* parent, const char far* title, TModule*
module)
    : TWindow(parent, title, module)
{
    m_pTree = NULL;                         /* Tree control */

    m_apThreePlusMinBitmaps[0] = NULL;   /* Three plus/minus bitmaps, see
SetPlusMinus in online help */
    m_apThreePlusMinBitmaps[1] = NULL;
    m_apThreePlusMinBitmaps[2] = NULL;

    m_apThreeItemBitmaps[0] = NULL;      /* Three default item bitmaps, see
SetBitmaps in online help */
    m_apThreeItemBitmaps[1] = NULL;
    m_apThreeItemBitmaps[2] = NULL;
    m_pOtherItemBitmap = NULL;           /* Another item bitmap, see SetBitmaps in
online help */

    m_pCellBitmap = NULL;                /* A cell bitmap */

    m_pRowColBitmap = NULL;              /* Row/column bitmap */

    m_pEditFont = NULL;                  /* Item data editing font */

    m_pEdit = NULL;                      /* No item data editing */
}
```

```
TSampleWindow::~TSampleWindow ()
{
    Destroy();
}

void TSampleWindow::EvSize (uint sizeType, TSize& size)
{
    TWindow::EvSize(sizeType, size);

    if (m_pTree)
        m_pTree->MoveWindow(0, 0, size.cx, size.cy, true);

}


int TSampleWindow::EvCreate (CREATESTRUCT far& createStruct)
{
    int result;

    result = TWindow::EvCreate(createStruct);

    m_pTree = new TSftTree(this,         /* 'this' is the parent window */
        IDC_TREE,                        /* tree control ID */
        0,0,0,0);                    /* Location */
    m_pTree->Attr.Style |=
        SFTTREESTYLE_NOTIFY |            /* Notify parent window */
        SFTTREESTYLE_DRAGDROP |          /* Drag & drop enabled */
        SFTTREESTYLE_LEFTBUTTONONLY |    /* Only respond to left mouse button */
        SFTTREESTYLE_SCROLL |            /* Honor WS_H/VSCROLL */
        SFTTREESTYLE_DISABLENOSCROLL |   /* Disable scrollbars instead of hiding */
        WS_HSCROLL | WS_VSCROLL |        /* Vertical and horizontal scrollbars */
        WS_VISIBLE | WS_CHILD;           /* Visible, child window */
    if (!m_pTree->Create())
            ; /* Error handling here */

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

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

    hDC = ::GetDC(NULL);                 /* Get a device context */

    /* Create the font to be used for item data editing.       */
    /* Delete the font (DeleteObject) once the tree control no */
    /* longer uses it.                                         */

    height = MulDiv(8, ::GetDeviceCaps(hDC, LOGPIXELSY), 72);
                                         /* Convert point-size to pixels */
    m_pEditFont = new TFont(-height, 0, 0, 0, FW_BOLD, 0, 0, 0, 0, 0, 0, 0, 0,
TEXT("MS Sans Serif"));

    ::ReleaseDC(NULL, hDC);              /* Release device context */


    /* Bitmaps must remain valid until the tree control is destroyed   */
    /* or no longer uses the bitmaps. All bitmaps used are of type     */
    /* TBitmap.                                                        */

    m_pTree->SetShowHeader(TRUE);        /* Show column headers */

    /* Register the plus/minus bitmaps.  These bitmaps are     */
```

```
/* used for all items in the tree control. All three       */
/* bitmaps must be the same size. You cannot override      */
/* the plus/minus bitmap for individual items.             */

m_apThreePlusMinBitmaps[0] = new TBitmap(*GetApplication(), IDB_PLUS);
                                    /* Expandable bitmap */
m_apThreePlusMinBitmaps[1] = new TBitmap(*GetApplication(), IDB_MINUS);
                                    /* Expanded bitmap */
m_apThreePlusMinBitmaps[2] = new TBitmap(*GetApplication(), IDB_NONE);
                                    /* Leaf bitmap */
m_pTree->SetPlusMinus(m_apThreePlusMinBitmaps);
                                    /* Use +/- bitmaps */

/* Register the item bitmaps.  These bitmaps are used for  */
/* all items in the tree control. All three bitmaps must   */
/* be the same size.                                       */

m_apThreeItemBitmaps[0] = new TBitmap(*GetApplication(), IDB_EXPANDABLE);
                                    /* Expandable bitmap */
m_apThreeItemBitmaps[1] = new TBitmap(*GetApplication(), IDB_EXPANDED);
                                    /* Expanded bitmap */
m_apThreeItemBitmaps[2] = new TBitmap(*GetApplication(), IDB_LEAF);
                                    /* Leaf bitmap */
m_pTree->SetBitmaps(m_apThreeItemBitmaps);
                                    /* Use item bitmaps */

/* Register the cell bitmap size.  All cell bitmaps used   */
/* must be the same size.  Only one bitmap needs to be     */
/* registered, even if several are used.                   */

{
    SFTTREE_CELLINFOPARM CellInfo;
    CellInfo.version = 2;           /* Mandatory, using version 2 format */
    CellInfo.index = -1;            /* Registering bitmap size */
    m_pCellBitmap = new TBitmap(*GetApplication(), IDB_SMILE);
                                    /* Cell bitmap */
    CellInfo.Cell.hBmp = *m_pCellBitmap;
    m_pTree->SetCellInfo(&CellInfo); /* Register use of cell bitmaps */

}

m_pTree->SetShowGrid(TRUE);         /* Show grid */

m_pTree->SetShowTruncated(TRUE);    /* Show ... if truncated */

m_pTree->SetNoFocusStyle(SFTTREE_NOFOCUS_FRAME);
                                    /* No focus, show frame */

m_pTree->SetSelTextOnly(TRUE);      /* Select text only */

m_pTree->SetShowHeaderButtons(TRUE); /* Show column header as buttons */

/* Define columns */

{
    SFTTREE_COLUMN_EX aCol[2] = {
      { 0, 0,                       /* Reserved */
        100,                        /* Width (in pixels) */
        ES_LEFT,                    /* Cell alignment */
        ES_LEFT,                    /* Title style */
        TEXT("First Column"),       /* Column header title */
        NULL, NULL,                 /* Reserved field and bitmap handle */
        SFTTREE_BMP_RIGHT },        /* Bitmap alignment */
```

```
          { 0, 0,                          /* Reserved */
            -1,                            /* Width (in pixels) */
            ES_LEFT,                       /* Cell alignment */
            ES_LEFT,                       /* Title style */
            TEXT("Second Column"),         /* Column header title */
            NULL, NULL,                    /* Reserved field and bitmap handle */
            SFTTREE_BMP_RIGHT }            /* Bitmap alignment */
        };
        m_pTree->SetColumns(2, aCol);      /* Set column attributes */
    }

    /* Set the row header attributes.                          */

    m_pTree->SetShowRowHeader(SFTTREE_ROWSTYLE_BUTTONCOUNT1);
                                           /* Row style */

    /* Set the row/column header attributes                    */

    m_pTree->SetRowColText(TEXT("?"));     /* Row/column header text */

    m_pTree->SetRowColHeaderStyle(ES_LEFT | SFTTREE_HEADER_UP);
                                           /* Row/column header style */

    /* Set the row/column header bitmap */

    m_pRowColBitmap = new TBitmap(*GetApplication(), IDB_SMILE);
                                           /* Row/column header bitmap */
    m_pTree->SetRowColBitmap(*m_pRowColBitmap);
                                           /* Row/column bitmap */

    /* Set the row/column header bitmap alignment */

    m_pTree->SetRowColBitmapStyle(SFTTREE_BMP_RIGHT);
                                           /* Row/column bitmap alignment */

    m_pTree->SetDragBitmaps(TRUE);         /* Allow drag & drop from bitmaps */

    m_pTree->SetDragType(SFTTREE_DRAG_PIXELIMM);
                                           /* Select and move by a number of pixels to
start drag */

    m_pTree->SetDropHighlightStyle(SFTTREE_DROPHIGHLIGHT_ONTOP);
                                           /* Highlight drop target */

    /* Use a delete callback routine */

    {
        SFTTREE_DELETEPARM Parm;           /* Parameter list */

        Parm.lpfnDelete = (SFTTREE_DELETEPROC) TSampleWindow::DeleteCallback;/* User
supplied delete routine */
        Parm.UserData = (DWORD)0;          /* User supplied data */
        m_pTree->SetDeleteCallback(&Parm);
    }

    /* Use a drawing callback routine */

    {
        SFTTREE_DRAWINFOPARM Parm;         /* Parameter list */

        Parm.lpfnDrawInfo = (SFTTREE_DRAWINFOPROC)
TSampleWindow::DrawInfoCallback;/* User supplied drawing info routine */
        Parm.UserData = (DWORD)0;          /* User supplied data */
```

```
        m_pTree->SetDrawInfoCallback(&Parm);
    }

/*---------------------------------------------------------------------------*/
/* Add a few items.                                                          */
/*---------------------------------------------------------------------------*/

    int index;

    index = m_pTree->AddString(TEXT("An item"));
                                        /* Add an item */
    index = m_pTree->AddString(TEXT("Another item"));
                                        /* Add another item */
    m_pTree->SetItemLevel(index, 1);    /* change level */
    m_pTree->SetString(TEXT("2nd Column"), index, 1);
                                        /* set text in next column */

    /* Set individual cell bitmaps using this sample code: */

    SFTTREE_CELLINFOPARM CellInfo;
    CellInfo.version = 2;            /* Mandatory, using version 2 format */
    CellInfo.index = index;
    CellInfo.iCol = 1;
    m_pTree->GetCellInfo(&CellInfo);
    CellInfo.Cell.hBmp = *m_pCellBitmap;
    m_pTree->SetCellInfo(&CellInfo);

    /* Set an item's row header text using: */
    m_pTree->SetRowText(TEXT("Second"), index);
                                        /* Row header text */

    index = m_pTree->AddString(TEXT("A third item"));
                                        /* Add another item */
    m_pTree->SetItemLevel(index, 2);    /* change level */
    m_pTree->SetString(TEXT("2nd Column"), index, 1);
                                        /* set text in next column */
    index = m_pTree->AddString(TEXT("A fourth item"));
                                        /* Add another item */
    m_pTree->SetItemLevel(index, 1);    /* change level */
    m_pTree->SetString(TEXT("2nd Column"), index, 1);
                                        /* set text in next column */

/*---------------------------------------------------------------------------*/
/* Once ALL TREE CONTROL ITEMS HAVE BEEN ADDED, you can set additional tree  */
/* control attributes.                                                       */
/*---------------------------------------------------------------------------*/

    /* Make all column widths optimal, so text and bitmaps are */
    /* not clipped horizontally.                               */

    m_pTree->MakeColumnOptimal(-1);     /* Make column widths optimal */

    /* Make row header width optimal, so text and bitmaps are  */
    /* not clipped horizontally.                               */

    m_pTree->MakeRowHeaderOptimal();    /* Make row header width optimal */

    m_pTree->RecalcHorizontalExtent();  /* Update horizontal scroll bar */

    return result;
}
```

```
void TSampleWindow::EvDestroy ()
{
    TWindow::EvDestroy();

    delete m_apThreePlusMinBitmaps[0];/* Plus/minus bitmaps */
    delete m_apThreePlusMinBitmaps[1];
    delete m_apThreePlusMinBitmaps[2];

    delete m_apThreeItemBitmaps[0];/* Default item bitmaps */
    delete m_apThreeItemBitmaps[1];
    delete m_apThreeItemBitmaps[2];
    delete m_pOtherItemBitmap;          /* Another item bitmap */

    delete m_pCellBitmap;               /* A cell bitmap */

    delete m_pRowColBitmap;             /* Row/column bitmap */

    /* Delete the fonts used for the tree control.             */

    delete m_pEditFont;                 /* Item data editing font */

}

/* Start editing in response to a                          */
/* SFTTREEN_LBUTTONDBLCLK_TEXT notification.               */

void TSampleWindow::EvStartEdit()
{
    char szBuffer[80];
    LPCRECT lpRect;

    /* Get the item index where double-click happened */
    m_editIndex = m_pTree->GetCaretIndex(); /* Save item # */
    m_editCol = -1;                     /* Let SftTree/DLL provide column # */
    lpRect = m_pTree->GetEditColumnRect(m_editIndex, &m_editCol);
    if (!lpRect)                        /* No column active */
        return;
    /* Make the column completely visible */
    m_pTree->MakeColumnVisible(m_editCol);
    /* Make the item completely visible */
    m_pTree->MakeRowVisible(m_editIndex);
    /* Get the location again (in case it scrolled) */
    lpRect = m_pTree->GetEditColumnRect(m_editIndex, &m_editCol);
    if (!lpRect)                        /* No column active */
        return;

    /* Change selection style to not shown anything */
    m_pTree->SetNoFocusStyle(SFTTREE_NOFOCUS_NOTHING);

    /* Repaint now in case we scrolled to make item visible */
    m_pTree->UpdateWindow();

    /* Create the edit control.*/
    /* Based on the tree control attributes and your preference, you may */
    /* have to adjust the rectangle used for the edit control. */

    m_pEdit = new TEdit(m_pTree, IDC_EDIT, TEXT(""),
         lpRect->left, lpRect->top, lpRect->right-lpRect->left, lpRect->bottom-
lpRect->top);
    m_pEdit->Attr.Style |= WS_CHILD|WS_BORDER|ES_AUTOHSCROLL|ES_LEFT; /* Styles */
    m_pEdit->Create();

    /* Set some edit control attributes */
```

```
    /* Set the font defined for item data editing */
    m_pEdit->SetWindowFont(*m_pEditFont, FALSE);

    /* Copy the text found in the tree control */
    m_pTree->GetString(szBuffer, m_editIndex, m_editCol);
    m_pEdit->SetWindowText(szBuffer);
    /* Select all text in the edit control and display it */
    m_pEdit->SetSelection(0, -1);
    m_pEdit->ShowWindow(SW_SHOW);
    m_pEdit->SetFocus();                /* Set input focus to the control */
}

/* Quit editing in response to a SFTTREEN_QUITEDIT         */
/* notification.                                           */

void TSampleWindow::EvQuitEdit()
{
    if (m_pEdit) {
        /* If the control has the focus, set the focus back to */
        /* tree control after destroying the control */

        BOOL fHadFocus = (GetFocus() == m_pEdit->HWindow);

        m_pEdit->Destroy();
        delete m_pEdit;
        m_pEdit = NULL;
        /* Restore nofocus display method */
        m_pTree->SetNoFocusStyle(SFTTREE_NOFOCUS_FRAME);
        if (fHadFocus)
            m_pTree->SetFocus();         /* Back to tree control */
    }
}

/* Validate edit data in response to a                     */
/* SFTTREEN_VALIDATEEDIT notification.                     */

void TSampleWindow::EvValidateEdit()
{
    if (m_pEdit) {
        char szBuffer[80];

        /* Get the text from the edit control */
        m_pEdit->GetWindowText(szBuffer, sizeof(szBuffer));

        /* Validate the data */
        if (lstrcmp(TEXT(""), szBuffer) == 0) {
            MessageBox(TEXT("Just to demonstrate data input validation, this example
rejects empty cells.  Please enter some data."), TEXT("SftTree/DLL"), MB_OK|
MB_TASKMODAL|MB_ICONSTOP);
            m_pEdit->SetFocus();
        } else {
            /* Save the data in the tree control */
            m_pTree->SetString(szBuffer, m_editIndex, m_editCol);
            m_pEdit->Destroy();
            delete m_pEdit;
            m_pEdit = NULL;
            /* Restore nofocus display method */
            m_pTree->SetNoFocusStyle(SFTTREE_NOFOCUS_FRAME);
        }
    }
}
```

```
/* Respond to expand/collapse requests as the user clicks  */
/* on different tree components.  The events handled here  */
/* can be changed to suit your application.                */

void TSampleWindow::EvLButtonExpandCollapse()
{
    /* get index of item to expand/collapse */
    int index = m_pTree->GetCaretIndex();
    /* get current expand/collapsed status */
    BOOL fExpanded = m_pTree->GetItemExpand(index);
    /* if control key is used we'll expand all dependents */
    BOOL fDepth = (::GetKeyState(VK_CONTROL)&0x8000);

    m_pTree->SetItemExpand(index, !fExpanded, fDepth);
}

/* Drag and drop in ONE tree control is implemented very   */
/* easily by handling the SFTTREEN_ENDDRAG notification.   */
/* You can also respond to the SFTTREEN_BEGINDRAG and      */
/* SFTTREEN_DRAGGING notifications to override the drag    */
/* cursor or to cancel drag & drop if desired.             */

/* For multi-tree drag & drop see the DragDrop sample.     */

void TSampleWindow::EvEndDrag()
{
    /* The user dropped something */
    int index;
    index = m_pTree->GetDropHighlight(); /* Drop target */
    m_pTree->SetDropHighlight(-1, FALSE);/* Visually clear drop target */
    /* "index" now contains the drop target. */
    /* All selected items participate in the drag & drop operation */
    /* ... Application specific action goes here ... */
}

void TSampleWindow::EvDragging()
{
    /* Drag & drop in progress */
    /* Here we handle this event to override the default */
    /* drag cursor.  This is completely optional. */
    LPSFTTREE_DRAGINFO lpInfo;
    lpInfo = m_pTree->GetDragInfo();
}

void TSampleWindow::EvCancelDrag()
{
    m_pTree->SetDropHighlight(-1, FALSE);/* Visually clear drop target */
}

void _export CALLBACK TSampleWindow::DrawInfoCallback(HWND hwnd,
LPSFTTREE_DRAWINGINFO lpInfo, DWORD UserData)
{
    /* You can change the SFTTREE_DRAWINGINFO structure here to override options */
}

int _export CALLBACK TSampleWindow::SortCallbackEx(HWND hwnd, LPCSTR lpszString1,
LPCSTR lpszString2, DWORD item1, DWORD item2)
{
    /* In this example, items are sorted in descending order. */

    int rc = lstrcmp(lpszString1, lpszString2);
    if (rc > 0)
```

```
        return -1;
    else if (rc < 0)
        return 1;
    return 0;
}

void _export CALLBACK TSampleWindow::DeleteCallback(HWND hwnd, int index, DWORD
itemData, DWORD UserData)
{
    /* Perform item specific processing here. */

}
```

DragDrop Sample

The DragDrop sample displays two tree controls in a dialog and demonstrates how to implement drag & drop between tree controls.

Click here to run the DragDrop sample application. If the sample cannot be started, please reinstall the SftTree/DLL demo.

The example source code can be found in the following directories:

Language	***Directory***
C	C:\SftTree\Samples\C\Dragdrop
C++ & MFC	C:\SftTree\Samples\VC\Dragdrop
C++ & OWL	C:\SftTree\Samples\BC\Dragdrop

Relevant portions of the source code are included here for easy access to the on-line help information.

C, DragDrop Sample

The complete source code can be found in the directory C:\SftTree\Samples\C\Dragdrop.

```
/*********************************************************************/
/*                      Left Tree Control Handling                   */
/*********************************************************************/

static void DraggingLeft()
{
    /* Drag & drop in progress, which was started on the left side tree */
    LPSFTTREE_DRAGINFO lpInfo;
    lpInfo = SftTree_GetDragInfo(m_hwndLeftTree);

    if (lpInfo->hwnd != m_hwndLeftTree) { // The target is another window

        // clear old drop target
        if (m_hwndLastTarget)
            SftTree_SetDropHighlight(m_hwndLastTarget, -1, FALSE);
        m_hwndLastTarget = NULL;

        // the target is another control, we have to set the cursor and
        // also update the drop target
        // In this example the "other" control could be the right tree control

        if (lpInfo->hwnd == m_hwndRightTree) { // target is right tree
            POINT pt;
            int index;

            // set the drop OK cursor
            lpInfo->fDropOK = TRUE;
            lpInfo->hCursor = LoadCursor(g_hInst, MAKEINTRESOURCE(IDC_DRAG));

            // update the drop target in the right tree
            pt = lpInfo->ptDrag;
            MapWindowPoints(HWND_DESKTOP, m_hwndRightTree, &pt, 1);
            index = SftTree_CalcIndexFromPointEx(m_hwndRightTree, &pt);
            SftTree_SetDropHighlight(m_hwndRightTree, index, TRUE);

            // remember who is the drop target
            m_hwndLastTarget = m_hwndRightTree;
        }

    } else { // left side tree is the drop target
        // clear old drop target if different
        if (m_hwndLastTarget && m_hwndLastTarget != m_hwndLeftTree)
            SftTree_SetDropHighlight(m_hwndLastTarget, -1, FALSE);
        m_hwndLastTarget = m_hwndLeftTree;
    }
}

static void EndDragLeft()
{
    /* The user dropped something */

    LPSFTTREE_DRAGINFO lpInfo;
    int index;
    char szBuffer[80];
    LPCSTR lpszText;

    lpInfo = SftTree_GetDragInfo(m_hwndLeftTree);

    if (lpInfo->hwnd != m_hwndLeftTree) { // The target is another window
```

```
        // In this example the "other" control could be the right tree control

        if (lpInfo->hwnd == m_hwndRightTree) { // target is left tree
            index = SftTree_GetDropHighlight(m_hwndRightTree);
            lpszText = TEXT("The drop target is item %d in the right tree control");
        } else {
            index = -1;
            lpszText = TEXT("There is no valid drop target");
        }
    } else { // target is the right tree control
        index = SftTree_GetDropHighlight(m_hwndLeftTree);
        lpszText = TEXT("The drop target is item %d in the left tree control");
    }

    // always clear drop targets.  If you forget, or do it too late,
    // the vertical scrolling may still be active
    if (m_hwndLastTarget)
        SftTree_SetDropHighlight(m_hwndLastTarget, -1, FALSE);
    m_hwndLastTarget = NULL;

    wsprintf(szBuffer, lpszText, index);
    MessageBox(NULL, szBuffer, TEXT("SftTree/DLL"), MB_OK|MB_TASKMODAL);
}

/*********************************************************************/
/*                     Right Tree Control Handling                   */
/*********************************************************************/

static void DraggingRight()
{
    /* Drag & drop in progress, which was started on the right side tree */
    LPSFTTREE_DRAGINFO lpInfo;
    lpInfo = SftTree_GetDragInfo(m_hwndRightTree);

    if (lpInfo->hwnd != m_hwndRightTree) { // The target is another window

        // clear old drop target
        if (m_hwndLastTarget)
            SftTree_SetDropHighlight(m_hwndLastTarget, -1, FALSE);
        m_hwndLastTarget = NULL;

        // the target is another control, we have to set the cursor and
        // also update the drop target
        // In this example the "other" control could be the left tree control

        if (lpInfo->hwnd == m_hwndLeftTree) { // target is left tree
            POINT pt;
            int index;

            // set the drop OK cursor
            lpInfo->fDropOK = TRUE;
            lpInfo->hCursor = LoadCursor(g_hInst, MAKEINTRESOURCE(IDC_DRAG));

            // update the drop target in the left tree
            pt = lpInfo->ptDrag;
            MapWindowPoints(HWND_DESKTOP, m_hwndLeftTree, &pt, 1);
            index = SftTree_CalcIndexFromPointEx(m_hwndLeftTree, &pt);
            SftTree_SetDropHighlight(m_hwndLeftTree, index, TRUE);

            // remember who is the drop target
            m_hwndLastTarget = m_hwndLeftTree;
        }
```

```
    } else { // right side tree is the drop target
        // clear old drop target if different
        if (m_hwndLastTarget && m_hwndLastTarget != m_hwndRightTree)
            SftTree_SetDropHighlight(m_hwndLastTarget, -1, FALSE);
        m_hwndLastTarget = m_hwndRightTree;
    }
}

static void EndDragRight()
{
    /* The user dropped something */

    LPSFTTREE_DRAGINFO lpInfo;
    int index;
    char szBuffer[80];
    LPCSTR lpszText;

    lpInfo = SftTree_GetDragInfo(m_hwndRightTree);

    if (lpInfo->hwnd != m_hwndRightTree) { // The target is another window

        // In this example the "other" control could be the left tree control

        if (lpInfo->hwnd == m_hwndLeftTree) { // target is left tree
            index = SftTree_GetDropHighlight(m_hwndLeftTree);
            lpszText = TEXT("The drop target is item %d in the left tree control");
        } else {
            index = -1;
            lpszText = TEXT("There is no valid drop target");
        }
    } else { // target is the right tree control
        index = SftTree_GetDropHighlight(m_hwndRightTree);
        lpszText = TEXT("The drop target is item %d in the right tree control");
    }

    // always clear drop targets.  If you forget, or do it too late,
    // the vertical scrolling may still be active
    if (m_hwndLastTarget)
        SftTree_SetDropHighlight(m_hwndLastTarget, -1, FALSE);
    m_hwndLastTarget = NULL;

    wsprintf(szBuffer, lpszText, index);
    MessageBox(NULL, szBuffer, TEXT("SftTree/DLL"), MB_OK|MB_TASKMODAL);
}

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

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

    case WM_INITDIALOG: {
        int index;

        // Center this dialog
        CenterWindow(hwndDlg);

        // LEFT SIDE TREE CONTROL
```

```
        m_hwndLeftTree = GetDlgItem(hwndDlg, IDC_LEFTTREE);

        /* Register the item bitmaps.  These bitmaps are used for  */
        /* all items in the tree control. All three bitmaps must   */
        /* be the same size.                                       */

        m_ahThreeItemBitmaps[0] = LoadBitmap(g_hInst,
MAKEINTRESOURCE(IDB_EXPANDABLE));
                                        /* Expandable bitmap */
        m_ahThreeItemBitmaps[1] = LoadBitmap(g_hInst,
MAKEINTRESOURCE(IDB_EXPANDED));
                                        /* Expanded bitmap */
        m_ahThreeItemBitmaps[2] = LoadBitmap(g_hInst, MAKEINTRESOURCE(IDB_LEAF));
                                        /* Leaf bitmap */
        SftTree_SetBitmaps(m_hwndLeftTree, m_ahThreeItemBitmaps);
                                        /* Use item bitmaps */

        SftTree_SetShowTruncated(m_hwndLeftTree, TRUE);
                                        /* Show ... if truncated */

        SftTree_SetNoFocusStyle(m_hwndLeftTree, SFTTREE_NOFOCUS_FRAME);
                                        /* No focus, show frame */

        SftTree_SetSelTextOnly(m_hwndLeftTree, TRUE);
                                        /* Select text only */

        /* Define columns */

        {
            SFTTREE_COLUMN_EX aCol[1] = {
              { 0, 0,                   /* Reserved */
                -1,                     /* Width (in pixels) */
                ES_LEFT,                /* Cell alignment */
                ES_LEFT,                /* Title style */
                TEXT("Title"),          /* Column header title */
                NULL, NULL,             /* Reserved field and bitmap handle */
                SFTTREE_BMP_RIGHT }     /* Bitmap alignment */
            };
            SftTree_SetColumnsEx(m_hwndLeftTree, 1, aCol);
                                        /* Set column attributes */
        }

        /* Set the row header attributes.                          */

        SftTree_SetShowRowHeader(m_hwndLeftTree, SFTTREE_ROWSTYLE_BUTTONCOUNT0);
                                        /* Row style */

        SftTree_SetRowHeaderStyle(m_hwndLeftTree, ES_LEFT | SFTTREE_HEADER_UP);
                                        /* Row header style */

        SftTree_SetDragBitmaps(m_hwndLeftTree, TRUE);
                                        /* Allow drag & drop from bitmaps */

        SftTree_SetDragType(m_hwndLeftTree, SFTTREE_DRAG_PIXELIMM);
                                        /* Select and move by a number of
pixels to start drag */

        SftTree_SetDropHighlightStyle(m_hwndLeftTree,
SFTTREE_DROPHIGHLIGHT_BETWEEN);
                                        /* Draw line to represent drop target
*/

        // Add a few items
```

```
        for (index = 3 ; index <= 30 ; ++index) {
            char szBuffer[80];
            int i;
            wsprintf(szBuffer, TEXT("Item %d"), index-3);
            i = SftTree_AddString(m_hwndLeftTree, szBuffer);
            SftTree_SetItemLevel(m_hwndLeftTree, i, abs(3 - (index % 6)));
        }

        SftTree_MakeRowHeaderOptimal(m_hwndLeftTree);
                                            /* Make row header width optimal */

        SftTree_RecalcHorizontalExtent(m_hwndLeftTree);
                                            /* Update horizontal scroll bar */

        // RIGHT SIDE TREE CONTROL

        m_hwndRightTree = GetDlgItem(hwndDlg, IDC_RIGHTTREE);

        /* Register the item bitmaps.  These bitmaps are used for  */
        /* all items in the tree control. All three bitmaps must   */
        /* be the same size.                                       */

        SftTree_SetBitmaps(m_hwndRightTree, m_ahThreeItemBitmaps);
                                            /* Use item bitmaps */

        SftTree_SetShowTruncated(m_hwndRightTree, TRUE);
                                            /* Show ... if truncated */

        SftTree_SetNoFocusStyle(m_hwndRightTree, SFTTREE_NOFOCUS_FRAME);
                                            /* No focus, show frame */

        /* Define columns */

        {
            SFTTREE_COLUMN_EX aCol[1] = {
              { 0, 0,                       /* Reserved */
                -1,                         /* Width (in pixels) */
                ES_LEFT,                    /* Cell alignment */
                ES_LEFT,                    /* Title style */
                TEXT("Title"),              /* Column header title */
                NULL, NULL,                 /* Reserved field and bitmap handle */
                SFTTREE_BMP_RIGHT }         /* Bitmap alignment */
            };
            SftTree_SetColumnsEx(m_hwndRightTree, 1, aCol);
                                            /* Set column attributes */
        }

        SftTree_SetDragBitmaps(m_hwndRightTree, TRUE);
                                            /* Allow drag & drop from bitmaps */

        SftTree_SetDragType(m_hwndRightTree, SFTTREE_DRAG_PIXELIMM);
                                            /* Select and move by a number of
pixels to start drag */

        SftTree_SetDropHighlightStyle(m_hwndRightTree, SFTTREE_DROPHIGHLIGHT_ONTOP);
                                            /* Highlight drop target */

        // Add a few items

        for (index = 3 ; index <= 30 ; ++index) {
            char szBuffer[80];
            int i;
```

```
            wsprintf(szBuffer, TEXT("Item %d"), index-3);
            i = SftTree_AddString(m_hwndRightTree, szBuffer);
            SftTree_SetItemLevel(m_hwndRightTree, i, abs(3 - (index % 6)));
        }
        SftTree_RecalcHorizontalExtent(m_hwndRightTree);
                                            /* Update horizontal scroll bar */
#if defined(WANT3D)
        Ctl3dSubclassCtlEx(m_hwndLeftTree, CTL3D_LISTBOX_CTL);
        Ctl3dSubclassCtlEx(m_hwndRightTree, CTL3D_LISTBOX_CTL);
#endif
        return TRUE;
     }

    case WM_DESTROY: {
        DeleteObject(m_ahThreeItemBitmaps[0]);/* Default item bitmaps */
        DeleteObject(m_ahThreeItemBitmaps[1]);
        DeleteObject(m_ahThreeItemBitmaps[2]);
        break;
     }

    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_LEFTTREE:
                switch (code) {
                case SFTTREEN_LBUTTONDBLCLK_TEXT:
                case SFTTREEN_LBUTTONDOWN_BUTTON:
                case SFTTREEN_LBUTTONDBLCLK_BUTTON: {
                    /* Respond to expand/collapse requests as the user clicks  */
                    /* on different tree components.  The events handled here  */
                    /* can be changed to suit your application.                */
                    int index;
                    BOOL fExpand, fControl;
                    /* Get current position */
                    index = SftTree_GetCaretIndex(hwndCtl);/* Get caret location */
                    /* Check if item is expanded */
                    fExpand = SftTree_GetItemExpand(hwndCtl, index);
                    /* If the CONTROL key is pressed, expand all dependent levels */
                    fControl = (BOOL)(GetKeyState(VK_CONTROL)&0x8000);
                    /* Do the opposite */
                    SftTree_SetItemExpand(hwndCtl, index, !fExpand, fControl);
                    break;
                  }
                case SFTTREEN_ENDDRAG:
                    EndDragLeft();
                    break;
                case SFTTREEN_CANCELDRAG:
                    // always clear drop targets
                    if (m_hwndLastTarget) {
                        SftTree_SetDropHighlight(m_hwndLastTarget, -1, FALSE);
                        m_hwndLastTarget = NULL;
                    }
                    MessageBox(NULL, TEXT("Drag & drop cancelled."),
```

```
TEXT("SftTree/DLL"), MB_OK|MB_TASKMODAL);
                    break;
                case SFTTREEN_BEGINDRAG:
                case SFTTREEN_DRAGGING:
                    DraggingLeft();
                    break;
                }
                break;
            case IDC_RIGHTTREE:
                switch (code) {
                case SFTTREEN_LBUTTONDBLCLK_TEXT:
                case SFTTREEN_LBUTTONDOWN_BUTTON:
                case SFTTREEN_LBUTTONDBLCLK_BUTTON: {
                    /* Respond to expand/collapse requests as the user clicks  */
                    /* on different tree components.  The events handled here  */
                    /* can be changed to suit your application.                */
                    int index;
                    BOOL fExpand, fControl;
                    /* Get current position */
                    index = SftTree_GetCaretIndex(hwndCtl);/* Get caret location */
                    /* Check if item is expanded */
                    fExpand = SftTree_GetItemExpand(hwndCtl, index);
                    /* If the CONTROL key is pressed, expand all dependent levels */
                    fControl = (BOOL)(GetKeyState(VK_CONTROL)&0x8000);
                    /* Do the opposite */
                    SftTree_SetItemExpand(hwndCtl, index, !fExpand, fControl);
                    break;
                  }
                case SFTTREEN_ENDDRAG:
                    EndDragRight();
                    break;
                case SFTTREEN_CANCELDRAG:
                    // always clear drop targets
                    if (m_hwndLastTarget) {
                        SftTree_SetDropHighlight(m_hwndLastTarget, -1, FALSE);
                        m_hwndLastTarget = NULL;
                    }
                    MessageBox(NULL, TEXT("Drag & drop cancelled."),
TEXT("SftTree/DLL"), MB_OK|MB_TASKMODAL);
                    break;
                case SFTTREEN_BEGINDRAG:
                case SFTTREEN_DRAGGING:
                    DraggingRight();
                    break;
                }
                break;
            case IDOK:
            case IDCANCEL:
                if (code == BN_CLICKED)
                    SendMessage(hwndDlg, WM_COMMAND, id, 0);
                break;
            }

        } else {
            switch (id) {
            case IDOK:
                EndDialog(hwndDlg, TRUE);
                break;
            case IDCANCEL:
                EndDialog(hwndDlg, FALSE);
                break;
            }
        }
```

```
            break;
        }
    }
    return FALSE;
}
```

C++/MFC, DragDrop Sample

The complete source code can be found in the directory C:\SftTree\Samples\VC\Dragdrop.

```
/////////////////////////////////////////////////////////////////////////////
// CSampleDlg dialog

CSampleDlg::CSampleDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CSampleDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CSampleDlg)
        // NOTE: the ClassWizard will add member initialization here
    //}}AFX_DATA_INIT
    // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);

    // the remembered drop target
    m_lastDropTarget = NULL;
}

void CSampleDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CSampleDlg)
        // NOTE: the ClassWizard will add DDX and DDV calls here
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CSampleDlg, CDialog)
    ON_SFTTREEN_LBUTTONDBLCLK_TEXT(IDC_LEFTTREE, OnLButtonExpandCollapseLeft)
    ON_SFTTREEN_LBUTTONDOWN_BUTTON(IDC_LEFTTREE, OnLButtonExpandCollapseLeft)
    ON_SFTTREEN_LBUTTONDBLCLK_BUTTON(IDC_LEFTTREE, OnLButtonExpandCollapseLeft)
    ON_SFTTREEN_ENDDRAG(IDC_LEFTTREE, OnEndDragLeft)
    ON_SFTTREEN_BEGINDRAG(IDC_LEFTTREE, OnDraggingLeft)
    ON_SFTTREEN_DRAGGING(IDC_LEFTTREE, OnDraggingLeft)
    ON_SFTTREEN_CANCELDRAG(IDC_LEFTTREE, OnCancelDrag)
    ON_SFTTREEN_LBUTTONDBLCLK_TEXT(IDC_RIGHTTREE, OnLButtonExpandCollapseRight)
    ON_SFTTREEN_LBUTTONDOWN_BUTTON(IDC_RIGHTTREE, OnLButtonExpandCollapseRight)
    ON_SFTTREEN_LBUTTONDBLCLK_BUTTON(IDC_RIGHTTREE, OnLButtonExpandCollapseRight)
    ON_SFTTREEN_ENDDRAG(IDC_RIGHTTREE, OnEndDragRight)
    ON_SFTTREEN_BEGINDRAG(IDC_RIGHTTREE, OnDraggingRight)
    ON_SFTTREEN_DRAGGING(IDC_RIGHTTREE, OnDraggingRight)
    ON_SFTTREEN_CANCELDRAG(IDC_RIGHTTREE, OnCancelDrag)
    //{{AFX_MSG_MAP(CSampleDlg)
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSampleDlg message handlers

BOOL CSampleDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Set the icon for this dialog.  The framework does this automatically
    //  when the application's main window is not a dialog
#if defined(_WIN32)
    SetIcon(m_hIcon, TRUE);         // Set big icon
    SetIcon(m_hIcon, FALSE);        // Set small icon
#endif
```

```
    // INITIALIZE the left side tree control

    /* Associate the tree control created from the dialog      */
    /* resource with the C++ object.                           */
    m_LeftTree.SubclassDlgItem(IDC_LEFTTREE, this /* parent window */);
    /* You could use DDX/DDV instead and add the following     */
    /* line to the DoDataExchange function of the tree         */
    /* control's parent window (remove the //).                */
    // DDX_Control(pDX, IDC_LEFTTREE, m_LeftTree);

    /* Register the item bitmaps.  These bitmaps are used for  */
    /* all items in the tree control. All three bitmaps must   */
    /* be the same size.                                       */

    m_aThreeItemBitmaps[0].LoadBitmap(IDB_EXPANDABLE);
                                        /* Expandable bitmap */
    m_aThreeItemBitmaps[1].LoadBitmap(IDB_EXPANDED);
                                        /* Expanded bitmap */
    m_aThreeItemBitmaps[2].LoadBitmap(IDB_LEAF);
                                        /* Leaf bitmap */
    m_LeftTree.SetBitmaps(m_aThreeItemBitmaps);
                                        /* Use item bitmaps */
    m_LeftTree.SetShowTruncated(TRUE);  /* Show ... if truncated */

    m_LeftTree.SetNoFocusStyle(SFTTREE_NOFOCUS_FRAME);
                                        /* No focus, show frame */

    m_LeftTree.SetSelTextOnly(TRUE);    /* Select text only */

    /* Define columns */

    {
        SFTTREE_COLUMN_EX aCol[1] = {
          { 0, 0,                       /* Reserved */
            -1,                         /* Width (in pixels) */
            ES_LEFT,                    /* Cell alignment */
            ES_LEFT,                    /* Title style */
            _T("Title"),                /* Column header title */
            NULL, NULL,                 /* Reserved field and bitmap handle */
            SFTTREE_BMP_RIGHT }         /* Bitmap alignment */
        };
        m_LeftTree.SetColumns(1, aCol);  /* Set column attributes */
    }

    /* Set the row header attributes.                          */

    m_LeftTree.SetShowRowHeader(SFTTREE_ROWSTYLE_BUTTONCOUNT0);
                                        /* Row style */

    m_LeftTree.SetRowHeaderStyle(ES_LEFT | SFTTREE_HEADER_UP);
                                        /* Row header style */

    m_LeftTree.SetDragBitmaps(TRUE);    /* Allow drag & drop from bitmaps */

    m_LeftTree.SetDragType(SFTTREE_DRAG_PIXELIMM);
                                        /* Select and move by a number of pixels to
start drag */

    m_LeftTree.SetDropHighlightStyle(SFTTREE_DROPHIGHLIGHT_BETWEEN);
                                        /* Draw line to represent drop target */

    // Add a few items
```

```
    for (int index = 3 ; index <= 30 ; ++index) {
        int i;
        CString str;
        str.Format(_T("Item %d"), index-3);
        i = m_LeftTree.AddString(str);
        m_LeftTree.SetItemLevel(i, abs(3 - (index % 6)));
    }

    m_LeftTree.MakeRowHeaderOptimal();   /* Make row header width optimal */

    m_LeftTree.RecalcHorizontalExtent(); /* Update horizontal scroll bar */

    // INITIALIZE the right side tree control

    /* Associate the tree control created from the dialog      */
    /* resource with the C++ object.                           */
    m_RightTree.SubclassDlgItem(IDC_RIGHTTREE, this /* parent window */);
    /* You could use DDX/DDV instead and add the following     */
    /* line to the DoDataExchange function of the tree         */
    /* control's parent window (remove the //).                */
    // DDX_Control(pDX, IDC_RIGHTTREE, m_RightTree);

    m_RightTree.SetBitmaps(m_aThreeItemBitmaps);
                                        /* Use item bitmaps */
    m_RightTree.SetShowTruncated(TRUE);  /* Show ... if truncated */

    m_RightTree.SetNoFocusStyle(SFTTREE_NOFOCUS_FRAME);
                                        /* No focus, show frame */

    /* Define columns */

    {
        SFTTREE_COLUMN_EX aCol[1] = {
          { 0, 0,                       /* Reserved */
            -1,                         /* Width (in pixels) */
            ES_LEFT,                    /* Cell alignment */
            ES_LEFT,                    /* Title style */
            _T("Title"),                /* Column header title */
            NULL, NULL,                 /* Reserved field and bitmap handle */
            SFTTREE_BMP_RIGHT }         /* Bitmap alignment */
        };
        m_RightTree.SetColumns(1, aCol); /* Set column attributes */
    }

    m_RightTree.SetDragBitmaps(TRUE);    /* Allow drag & drop from bitmaps */

    m_RightTree.SetDragType(SFTTREE_DRAG_PIXELIMM);
                                        /* Select and move by a number of pixels to
start drag */

    m_RightTree.SetDropHighlightStyle(SFTTREE_DROPHIGHLIGHT_ONTOP);
                                        /* Highlight drop target */
    // Add a few items

    for (index = 3 ; index <= 30 ; ++index) {
        int i;
        CString str;
        str.Format(_T("Item %d"), index-3);
        i = m_RightTree.AddString(str);
        m_RightTree.SetItemLevel(i, abs(3 - (index % 6)));
    }

    m_RightTree.RecalcHorizontalExtent();/* Update horizontal scroll bar */
```

```
#if defined(WANT3D)
    ::Ctl3dSubclassCtlEx(m_LeftTree.m_hWnd, CTL3D_LISTBOX_CTL);
    ::Ctl3dSubclassCtlEx(m_RightTree.m_hWnd, CTL3D_LISTBOX_CTL);
#endif

    CenterWindow();

    return TRUE;  // return TRUE  unless you set the focus to a control
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CSampleDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting

        SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        CDialog::OnPaint();
    }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CSampleDlg::OnQueryDragIcon()
{
    return (HCURSOR) m_hIcon;
}

// LEFT SIDE TREE CONTROL EVENTS

/* Respond to expand/collapse requests as the user clicks  */
/* on different tree components.  The events handled here  */
/* can be changed to suit your application.                */

void CSampleDlg::OnLButtonExpandCollapseLeft()
{
    /* get index of item to expand/collapse */
    int index = m_LeftTree.GetCaretIndex();
    /* get current expand/collapsed status */
    BOOL fExpanded = m_LeftTree.GetItemExpand(index);
    /* if control key is used we'll expand all dependents */
    BOOL fDepth = (::GetKeyState(VK_CONTROL)&0x8000);

    m_LeftTree.SetItemExpand(index, !fExpanded, fDepth);
```

```
}

void CSampleDlg::OnEndDragLeft()
{
    /* The user dropped something */

    LPSFTTREE_DRAGINFO lpInfo;
    int index;
    CString str, text;

    lpInfo = m_LeftTree.GetDragInfo();

    if (lpInfo->hwnd != m_LeftTree.m_hWnd) { // The target is another window

        // In this example the "other" control could be the right tree control

        if (lpInfo->hwnd == m_RightTree.m_hWnd) { // target is left tree
            index = m_RightTree.GetDropHighlight();
            text = _T("The drop target is item %d in the right tree control");
        } else {
            index = -1;
            text = _T("There is no valid drop target");
        }
    } else { // target is the right tree control
        index = m_LeftTree.GetDropHighlight();
        text = _T("The drop target is item %d in the left tree control");
    }

    // always clear drop targets.  If you forget, or do it too late,
    // the vertical scrolling may still be active
    if (m_lastDropTarget)
        m_lastDropTarget->SetDropHighlight(-1);
    m_lastDropTarget = NULL;

    str.Format(text, index);
    AfxMessageBox(str);
}

void CSampleDlg::OnDraggingLeft()
{
    /* Drag & drop in progress, which was started on the left side tree */
    LPSFTTREE_DRAGINFO lpInfo;
    lpInfo = m_LeftTree.GetDragInfo();

    if (lpInfo->hwnd != m_LeftTree.m_hWnd) { // The target is another window

        // clear old drop target
        if (m_lastDropTarget)
            m_lastDropTarget->SetDropHighlight(-1);
        m_lastDropTarget = NULL;

        // the target is another control, we have to set the cursor and
        // also update the drop target
        // In this example the "other" control could be the right tree control

        if (lpInfo->hwnd == m_RightTree.m_hWnd) { // target is right tree

            // set the drop OK cursor
            lpInfo->fDropOK = TRUE;
            lpInfo->hCursor = AfxGetApp()->LoadCursor(IDC_DRAG);

            // update the drop target in the right tree
            CPoint pt = lpInfo->ptDrag;
```

```
            ::MapWindowPoints(HWND_DESKTOP, m_RightTree.m_hWnd, &pt, 1);
            int index = m_RightTree.CalcIndexFromPointEx(&pt);
            m_RightTree.SetDropHighlight(index, TRUE);

            // remember who is the drop target
            m_lastDropTarget = &m_RightTree;
        }

    } else { // left side tree is the drop target
        // clear old drop target if different
        if (m_lastDropTarget && m_lastDropTarget != &m_LeftTree)
            m_lastDropTarget->SetDropHighlight(-1);
        m_lastDropTarget = &m_LeftTree;
    }
}

// RIGHT SIDE TREE CONTROL EVENTS

/* Respond to expand/collapse requests as the user clicks  */
/* on different tree components.  The events handled here  */
/* can be changed to suit your application.                */

void CSampleDlg::OnLButtonExpandCollapseRight()
{
    /* get index of item to expand/collapse */
    int index = m_RightTree.GetCaretIndex();
    /* get current expand/collapsed status */
    BOOL fExpanded = m_RightTree.GetItemExpand(index);
    /* if control key is used we'll expand all dependents */
    BOOL fDepth = (::GetKeyState(VK_CONTROL)&0x8000);

    m_RightTree.SetItemExpand(index, !fExpanded, fDepth);
}

void CSampleDlg::OnEndDragRight()
{
    /* The user dropped something */

    LPSFTTREE_DRAGINFO lpInfo;
    int index;
    CString str, text;

    lpInfo = m_RightTree.GetDragInfo();

    if (lpInfo->hwnd != m_RightTree.m_hWnd) { // The target is another window

        // In this example the "other" control could be the left tree control

        if (lpInfo->hwnd == m_LeftTree.m_hWnd) { // target is left tree
            index = m_LeftTree.GetDropHighlight();
            text = _T("The drop target is item %d in the left tree control");
        } else {
            index = -1;
            text = _T("There is no valid drop target");
        }
    } else { // target is the right tree control
        index = m_RightTree.GetDropHighlight();
        text = _T("The drop target is item %d in the right tree control");
    }

    // always clear drop targets.  If you forget, or do it too late,
    // the vertical scrolling may still be active
    if (m_lastDropTarget)
```

```
        m_lastDropTarget->SetDropHighlight(-1);
    m_lastDropTarget = NULL;

    str.Format(text, index);
    AfxMessageBox(str);
}

void CSampleDlg::OnDraggingRight()
{
    /* Drag & drop in progress, which was started on the right side tree */
    LPSFTTREE_DRAGINFO lpInfo;
    lpInfo = m_RightTree.GetDragInfo();

    if (lpInfo->hwnd != m_RightTree.m_hWnd) { // The target is another window

        // clear old drop target
        if (m_lastDropTarget)
            m_lastDropTarget->SetDropHighlight(-1);
        m_lastDropTarget = NULL;

        // the target is another control, we have to set the cursor and
        // also update the drop target
        // In this example the "other" control could be the left tree control

        if (lpInfo->hwnd == m_LeftTree.m_hWnd) { // target is left tree

            // set the drop OK cursor
            lpInfo->fDropOK = TRUE;
            lpInfo->hCursor = AfxGetApp()->LoadCursor(IDC_DRAG);

            // update the drop target in the left tree
            CPoint pt = lpInfo->ptDrag;
            ::MapWindowPoints(HWND_DESKTOP, m_LeftTree.m_hWnd, &pt, 1);
            int index = m_LeftTree.CalcIndexFromPointEx(&pt);
            m_LeftTree.SetDropHighlight(index, TRUE);

            // remember who is the drop target
            m_lastDropTarget = &m_LeftTree;
        }

    } else { // right side tree is the drop target
        // clear old drop target if different
        if (m_lastDropTarget && m_lastDropTarget != &m_RightTree)
            m_lastDropTarget->SetDropHighlight(-1);
        m_lastDropTarget = &m_RightTree;
    }
}

void CSampleDlg::OnCancelDrag()
{
    // always clear drop targets
    if (m_lastDropTarget) {
        m_lastDropTarget->SetDropHighlight(-1); /* Visually clear drop target */
        m_lastDropTarget = NULL;
    }
    AfxMessageBox(_T("Drag & drop cancelled."));
}
```

C++/OWL, DragDrop Sample

The complete source code can be found in the directory C:\SftTree\Samples\BC\Dragdrop.

```
//
// Build a response table for all messages/commands handled
// by the application.
//
DEFINE_RESPONSE_TABLE1(TSampleDlg, TDialog)
    EV_SFTTREEN_LBUTTONDBLCLK_TEXT(IDC_LEFTTREE, EvLButtonExpandCollapseLeft),
    EV_SFTTREEN_LBUTTONDOWN_BUTTON(IDC_LEFTTREE, EvLButtonExpandCollapseLeft),
    EV_SFTTREEN_LBUTTONDBLCLK_BUTTON(IDC_LEFTTREE, EvLButtonExpandCollapseLeft),
    EV_SFTTREEN_LBUTTONDBLCLK_TEXT(IDC_RIGHTTREE, EvLButtonExpandCollapseRight),
    EV_SFTTREEN_LBUTTONDOWN_BUTTON(IDC_RIGHTTREE, EvLButtonExpandCollapseRight),
    EV_SFTTREEN_LBUTTONDBLCLK_BUTTON(IDC_RIGHTTREE, EvLButtonExpandCollapseRight),
    EV_SFTTREEN_ENDDRAG(IDC_LEFTTREE, EvEndDragLeft),
    EV_SFTTREEN_BEGINDRAG(IDC_LEFTTREE, EvDraggingLeft),
    EV_SFTTREEN_DRAGGING(IDC_LEFTTREE, EvDraggingLeft),
    EV_SFTTREEN_CANCELDRAG(IDC_LEFTTREE, EvCancelDrag),
    EV_SFTTREEN_ENDDRAG(IDC_RIGHTTREE, EvEndDragRight),
    EV_SFTTREEN_BEGINDRAG(IDC_RIGHTTREE, EvDraggingRight),
    EV_SFTTREEN_DRAGGING(IDC_RIGHTTREE, EvDraggingRight),
    EV_SFTTREEN_CANCELDRAG(IDC_RIGHTTREE, EvCancelDrag),
//{{TSampleDlgRSP_TBL_BEGIN}}
    EV_WM_DESTROY,
//{{TSampleDlgRSP_TBL_END}}
END_RESPONSE_TABLE;


//{{TSampleDlg Implementation}}


//////////////////////////////////////////////////////
// TSampleDlg
// ==========
// Construction/Destruction handling.
TSampleDlg::TSampleDlg (TWindow *parent, TResId resId, TModule *module)
    : TDialog(parent, resId, module)
{
    /* Associate the tree control created from the dialog      */
    /* resource with a C++ object.                             */
    pLeftTree = new TSftTree(this, IDC_LEFTTREE);
    pRightTree = new TSftTree(this, IDC_RIGHTTREE);

    m_apThreeItemBitmaps[0] = NULL;      /* Three default item bitmaps, see
SetBitmaps in online help */
    m_apThreeItemBitmaps[1] = NULL;
    m_apThreeItemBitmaps[2] = NULL;

    m_lastDropTarget = NULL;
}


TSampleDlg::~TSampleDlg ()
{
    Destroy();

    // INSERT>> Your destructor code here.
}

bool TSampleDlg::EvInitDialog (HWND hWndFocus)
{
    bool result;
```

```
    result = TDialog::EvInitDialog(hWndFocus);

    // LEFT TREE CONTROL

    /* Bitmaps must remain valid until the tree control is destroyed   */
    /* or no longer uses the bitmaps. All bitmaps used are of type     */
    /* TBitmap.                                                        */

    /* Register the item bitmaps.  These bitmaps are used for  */
    /* all items in the tree control. All three bitmaps must   */
    /* be the same size.                                       */

    m_apThreeItemBitmaps[0] = new TBitmap(*GetApplication(), IDB_EXPANDABLE);
                                        /* Expandable bitmap */
    m_apThreeItemBitmaps[1] = new TBitmap(*GetApplication(), IDB_EXPANDED);
                                        /* Expanded bitmap */
    m_apThreeItemBitmaps[2] = new TBitmap(*GetApplication(), IDB_LEAF);
                                        /* Leaf bitmap */
    pLeftTree->SetBitmaps(m_apThreeItemBitmaps);
                                        /* Use item bitmaps */

    pLeftTree->SetShowTruncated(TRUE);   /* Show ... if truncated */

    pLeftTree->SetNoFocusStyle(SFTTREE_NOFOCUS_FRAME);
                                        /* No focus, show frame */

    pLeftTree->SetSelTextOnly(TRUE);     /* Select text only */

    /* Define columns */

    {
        SFTTREE_COLUMN_EX aCol[1] = {
          { 0, 0,                       /* Reserved */
            -1,                         /* Width (in pixels) */
            ES_LEFT,                    /* Cell alignment */
            ES_LEFT,                    /* Title style */
            TEXT("Title"),              /* Column header title */
            NULL, NULL,                 /* Reserved field and bitmap handle */
            SFTTREE_BMP_RIGHT }         /* Bitmap alignment */
        };
        pLeftTree->SetColumns(1, aCol);  /* Set column attributes */
    }

    /* Set the row header attributes.                          */

    pLeftTree->SetShowRowHeader(SFTTREE_ROWSTYLE_BUTTONCOUNT0);
                                        /* Row style */

    pLeftTree->SetRowHeaderStyle(ES_LEFT | SFTTREE_HEADER_UP);
                                        /* Row header style */

    pLeftTree->SetDragBitmaps(TRUE);     /* Allow drag & drop from bitmaps */

    pLeftTree->SetDragType(SFTTREE_DRAG_PIXELIMM);
                                        /* Select and move by a number of pixels to
start drag */

    pLeftTree->SetDropHighlightStyle(SFTTREE_DROPHIGHLIGHT_BETWEEN);
                                        /* Draw line to represent drop target */

    // Add a few items
```

```
    int index;
    for (index = 3 ; index <= 30 ; ++index) {
        int i;
        char szBuffer[80];
        wsprintf(szBuffer, "Item %d", index-3);
        i = pLeftTree->AddString(szBuffer);
        pLeftTree->SetItemLevel(i, abs(3 - (index % 6)));
    }

    /* Make row header width optimal, so text and bitmaps are  */
    /* not clipped horizontally.                               */

    pLeftTree->MakeRowHeaderOptimal();   /* Make row header width optimal */

    pLeftTree->RecalcHorizontalExtent(); /* Update horizontal scroll bar */


    // RIGHT TREE CONTROL

    /* Register the item bitmaps.  These bitmaps are used for  */
    /* all items in the tree control. All three bitmaps must   */
    /* be the same size.                                       */

    pRightTree->SetBitmaps(m_apThreeItemBitmaps);
                                        /* Use item bitmaps */

    pRightTree->SetShowTruncated(TRUE);  /* Show ... if truncated */

    pRightTree->SetNoFocusStyle(SFTTREE_NOFOCUS_FRAME);
                                        /* No focus, show frame */
    /* Define columns */

    {
        SFTTREE_COLUMN_EX aCol[1] = {
          { 0, 0,                       /* Reserved */
            -1,                         /* Width (in pixels) */
            ES_LEFT,                    /* Cell alignment */
            ES_LEFT,                    /* Title style */
            TEXT("Title"),              /* Column header title */
            NULL, NULL,                 /* Reserved field and bitmap handle */
            SFTTREE_BMP_RIGHT }         /* Bitmap alignment */
        };
        pRightTree->SetColumns(1, aCol); /* Set column attributes */
    }

    pRightTree->SetDragBitmaps(TRUE);    /* Allow drag & drop from bitmaps */

    pRightTree->SetDragType(SFTTREE_DRAG_PIXELIMM);
                                        /* Select and move by a number of pixels to
start drag */

    pRightTree->SetDropHighlightStyle(SFTTREE_DROPHIGHLIGHT_ONTOP);
                                        /* Highlight drop target */
    // Add a few items

    for (index = 3 ; index <= 30 ; ++index) {
        int i;
        char szBuffer[80];
        wsprintf(szBuffer, "Item %d", index-3);
        i = pRightTree->AddString(szBuffer);
        pRightTree->SetItemLevel(i, abs(3 - (index % 6)));
    }
```

```
    pRightTree->RecalcHorizontalExtent();/* Update horizontal scroll bar */

    return result;
}


void TSampleDlg::EvDestroy ()
{
    TDialog::EvDestroy();

    delete m_apThreeItemBitmaps[0];      /* Default item bitmaps */
    delete m_apThreeItemBitmaps[1];
    delete m_apThreeItemBitmaps[2];
}


void TSampleDlg::EvCancelDrag()
{
    // always clear drop targets
    if (m_lastDropTarget) {
        m_lastDropTarget->SetDropHighlight(-1); /* Visually clear drop target */
        m_lastDropTarget = NULL;
    }
    MessageBox(TEXT("Drag & drop cancelled."));
}

// LEFT TREE EVENT HANDLERS

void TSampleDlg::EvLButtonExpandCollapseLeft()
{
    /* get index of item to expand/collapse */
    int index = pLeftTree->GetCaretIndex();
    /* get current expand/collapsed status */
    BOOL fExpanded = pLeftTree->GetItemExpand(index);
    /* if control key is used we'll expand all dependents */
    BOOL fDepth = (::GetKeyState(VK_CONTROL)&0x8000);

    pLeftTree->SetItemExpand(index, !fExpanded, fDepth);
}

void TSampleDlg::EvEndDragLeft()
{
    /* The user dropped something */

    LPSFTTREE_DRAGINFO lpInfo;
    int index;
    char szBuffer[80];
    LPCSTR lpszText;

    lpInfo = pLeftTree->GetDragInfo();

    if (lpInfo->hwnd != pLeftTree->HWindow) { // The target is another window

        // In this example the "other" control could be the right tree control

        if (lpInfo->hwnd == pRightTree->HWindow) { // target is left tree
            index = pRightTree->GetDropHighlight();
            lpszText = TEXT("The drop target is item %d in the right tree control");
        } else {
            index = -1;
            lpszText = TEXT("There is no valid drop target");
        }
    } else { // target is the right tree control
```

```
        index = pLeftTree->GetDropHighlight();
        lpszText = TEXT("The drop target is item %d in the left tree control");
    }

    // always clear drop targets.  If you forget, or do it too late,
    // the vertical scrolling may still be active
    if (m_lastDropTarget)
        m_lastDropTarget->SetDropHighlight(-1);
    m_lastDropTarget = NULL;

    wsprintf(szBuffer, lpszText, index);
    MessageBox(szBuffer);
}

void TSampleDlg::EvDraggingLeft()
{
    /* Drag & drop in progress, which was started on the left side tree */
    LPSFTTREE_DRAGINFO lpInfo;
    lpInfo = pLeftTree->GetDragInfo();

    if (lpInfo->hwnd != pLeftTree->HWindow) { // The target is another window

        // clear old drop target
        if (m_lastDropTarget)
            m_lastDropTarget->SetDropHighlight(-1);
        m_lastDropTarget = NULL;

        // the target is another control, we have to set the cursor and
        // also update the drop target
        // In this example the "other" control could be the right tree control

        if (lpInfo->hwnd == pRightTree->HWindow) { // target is right tree

            // set the drop OK cursor
            lpInfo->fDropOK = TRUE;
            lpInfo->hCursor = GetApplication()->LoadCursor(IDC_DRAG);

            // update the drop target in the right tree
            POINT pt = lpInfo->ptDrag;
            ::MapWindowPoints(HWND_DESKTOP, pRightTree->HWindow, &pt, 1);
            int index = pRightTree->CalcIndexFromPointEx(&pt);
            pRightTree->SetDropHighlight(index, TRUE);

            // remember who is the drop target
            m_lastDropTarget = pRightTree;
        }

    } else { // left side tree is the drop target
        // clear old drop target if different
        if (m_lastDropTarget && m_lastDropTarget != pLeftTree)
            m_lastDropTarget->SetDropHighlight(-1);
        m_lastDropTarget = pLeftTree;
    }
}

// RIGHT TREE EVENT HANDLERS

void TSampleDlg::EvLButtonExpandCollapseRight()
{
    /* get index of item to expand/collapse */
    int index = pRightTree->GetCaretIndex();
    /* get current expand/collapsed status */
    BOOL fExpanded = pRightTree->GetItemExpand(index);
```

```
    /* if control key is used we'll expand all dependents */
    BOOL fDepth = (::GetKeyState(VK_CONTROL)&0x8000);

    pRightTree->SetItemExpand(index, !fExpanded, fDepth);
}

void TSampleDlg::EvEndDragRight()
{
    /* The user dropped something */

    LPSFTTREE_DRAGINFO lpInfo;
    int index;
    char szBuffer[80];
    LPCSTR lpszText;

    lpInfo = pRightTree->GetDragInfo();

    if (lpInfo->hwnd != pRightTree->HWindow) { // The target is another window

        // In this example the "other" control could be the left tree control

        if (lpInfo->hwnd == pLeftTree->HWindow) { // target is left tree
            index = pLeftTree->GetDropHighlight();
            lpszText = TEXT("The drop target is item %d in the left tree control");
        } else {
            index = -1;
            lpszText = TEXT("There is no valid drop target");
        }
    } else { // target is the right tree control
        index = pRightTree->GetDropHighlight();
        lpszText = TEXT("The drop target is item %d in the right tree control");
    }

    // always clear drop targets.  If you forget, or do it too late,
    // the vertical scrolling may still be active
    if (m_lastDropTarget)
        m_lastDropTarget->SetDropHighlight(-1);
    m_lastDropTarget = NULL;

    wsprintf(szBuffer, lpszText, index);
    MessageBox(szBuffer);
}

void TSampleDlg::EvDraggingRight()
{
    /* Drag & drop in progress, which was started on the right side tree */
    LPSFTTREE_DRAGINFO lpInfo;
    lpInfo = pRightTree->GetDragInfo();

    if (lpInfo->hwnd != pRightTree->HWindow) { // The target is another window

        // clear old drop target
        if (m_lastDropTarget)
            m_lastDropTarget->SetDropHighlight(-1);
        m_lastDropTarget = NULL;

        // the target is another control, we have to set the cursor and
        // also update the drop target
        // In this example the "other" control could be the left tree control

        if (lpInfo->hwnd == pLeftTree->HWindow) { // target is left tree

            // set the drop OK cursor
```

```
            lpInfo->fDropOK = TRUE;
            lpInfo->hCursor = GetApplication()->LoadCursor(IDC_DRAG);

            // update the drop target in the left tree
            POINT pt = lpInfo->ptDrag;
            ::MapWindowPoints(HWND_DESKTOP, pLeftTree->HWindow, &pt, 1);
            int index = pLeftTree->CalcIndexFromPointEx(&pt);
            pLeftTree->SetDropHighlight(index, TRUE);

            // remember who is the drop target
            m_lastDropTarget = pLeftTree;
        }

    } else { // right side tree is the drop target
        // clear old drop target if different
        if (m_lastDropTarget && m_lastDropTarget != pRightTree)
            m_lastDropTarget->SetDropHighlight(-1);
        m_lastDropTarget = pRightTree;
    }
}
```

CheckTab Sample

The CheckTab sample displays a tree control in a dialog and demonstrates how to implement a multi-column list box with check boxes. The check boxes are simulated by changing the label bitmap in response to a SFTTREEN_LBUTTONDOWN_LABEL notification.

This example also shows how a C++ class can be derived from CSftTree and TSftTree. Notifications are handled by the derived class, instead of the parent window (the dialog). For this reason, this example is not provided for C, only for C++.

Click here to run the CheckTab sample application. If the sample cannot be started, please reinstall the SftTree/DLL demo.

The example source code can be found in the following directories:

Language	***Directory***
C++ & MFC	C:\SftTree\Samples\VC\Checktab
C++ & OWL	C:\SftTree\Samples\BC\Checktab

Relevant portions of the source code are included here for easy access to the on-line help information.

C++/MFC, CheckTab Sample

The complete source code can be found in the directory C:\SftTree\Samples\VC\Checktab.

```
////////////////////////////////////////////////////////////////////////////
// CTable

CTable::CTable()
{
    m_LabelBitmapNo.LoadBitmap(IDB_NOCHECK);/* Load label bitmaps */
    m_LabelBitmapYes.LoadBitmap(IDB_CHECK);
}

CTable::~CTable()
{
}

BEGIN_MESSAGE_MAP(CTable, CSftTree)
#if defined(_WIN32) && (_MFC_VER >= 0x0400)
    // Message reflection is only supported starting with VC++ 4.0
    ON_SFTTREEN_LBUTTONDOWN_LABEL_REFLECT(OnLButtonDownLabel)
    ON_SFTTREEN_LBUTTONDBLCLK_LABEL_REFLECT(OnLButtonDownLabel)
#else
    // This achieves the same result as message reflection, but isn't
    // quite as elegant.
#endif
    //{{AFX_MSG_MAP(CTable)
    ON_WM_CREATE()
    ON_WM_KEYDOWN()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////////
// CTable message handlers

void CTable::Initialize()
{

    /* Bitmaps should be loaded using LoadBitmap and must remain valid */
    /* until the tree control is destroyed or no longer uses the       */
    /* bitmaps. All bitmaps used are of type CBitmap.                  */

    SetShowHeader(TRUE);            /* Show column headers */

    /* Register the label bitmap size.  All label bitmaps used */
    /* must be the same size.  Only one bitmap needs to be     */
    /* registered, even if several are used.                   */

    SetItemLabel(-1, m_LabelBitmapNo);/* Register the label bitmap size */
    SetShowTreeLines(FALSE);        /* No tree lines */
    SetShowButtons(FALSE);          /* No expand/collapse buttons */
    SetShowGrid(TRUE);              /* Show grid */
    SetShowTruncated(TRUE);         /* Show ... if truncated */
    SetNoFocusStyle(SFTTREE_NOFOCUS_FRAME);
                                        /* No focus, show frame */
    SetSelTextOnly(TRUE);           /* Select text only */
    SetShowHeaderButtons(TRUE);     /* Show column header as buttons */

    /* Define columns */
    {
        SFTTREE_COLUMN_EX aCol[2] = {
          { 0, 0,                       /* Reserved */
            100,                        /* Width (in pixels) */
```

```
            ES_LEFT,                        /* Cell alignment */
            ES_LEFT | SFTTREE_HEADER_UP,    /* Title style */
            _T("Option"),                   /* Column header title */
            NULL, NULL,                     /* Reserved field and bitmap handle */
            SFTTREE_BMP_RIGHT },            /* Bitmap alignment */
          { 0, 0,                           /* Reserved */
            -1,                             /* Width (in pixels) */
            ES_LEFT,                        /* Cell alignment */
            ES_LEFT | SFTTREE_HEADER_UP,    /* Title style */
            _T("Description"),              /* Column header title */
            NULL, NULL,                     /* Reserved field and bitmap handle */
            SFTTREE_BMP_RIGHT }             /* Bitmap alignment */
        };
        SetColumns(2, aCol);        /* Set column attributes */
    }

    /* Set the row header attributes.                          */
    SetShowRowHeader(SFTTREE_ROWSTYLE_TITLECOUNT1);
                                    /* Row style */
    SetRowHeaderStyle(ES_CENTER | SFTTREE_HEADER_UP);
                                    /* Row header style */

    /* Set the row/column header attributes                    */
    SetShowRowColHeaderButton(FALSE);
                                    /* Row/column header as title */
    //SetRowColText(_T("?"));         /* Row/column header text */
    SetRowColHeaderStyle(ES_LEFT | SFTTREE_HEADER_UP);
                                    /* Row/column header style */
}

void CTable::PreSubclassWindow()
{
    // This virtual function is called if a child control is created by a dialog
    // resource and SubclassDlgItem is used.  If a child window is created
dynamically
    // using Create, the OnCreate member function is called instead

    Initialize();
}

int CTable::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    // This message handler is called if a child control is created dynamically
    // using Create.  If it is created by a dialog resource and SubclassDlgItem is
    // used the PreSubclassWindow virtual function is called instead.

    if (CSftTree::OnCreate(lpCreateStruct) == -1)
        return -1;

    Initialize();
    return 0;
}

void CTable::OnLButtonDownLabel()
{
    // The item was clicked
    int index = GetCaretIndex();
    if (index >= 0)
        SetCheck(index, !GetCheck(index));
}

void CTable::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
{
```

```
    if (nChar == _T(' '))
        OnLButtonDownLabel();
    else
        CSftTree::OnKeyDown(nChar, nRepCnt, nFlags);
}

#if defined(_WIN32) && (_MFC_VER >= 0x0400)
    // We use message reflection under VC++ 4.0 and up
#else
    // 16-bit and 32-bit up to VC++ 2.2 needs this to handle messages
BOOL CTable::OnChildNotify(UINT message, WPARAM wParam, LPARAM lParam, LRESULT*
pLResult)
{
#if defined(_WIN32)
     WORD NotifyCode = HIWORD(wParam);
     WORD idItem = LOWORD(wParam);
     HWND hwndCtl = (HWND) lParam;
#else
    WORD NotifyCode = HIWORD(lParam);
    WORD idItem = wParam;
    HWND hwndCtl = (HWND) LOWORD(lParam);
#endif
    // Make sure it's a notification for this window
    if (hwndCtl != m_hWnd)
        return FALSE;
    switch (NotifyCode) {
    case SFTTREEN_LBUTTONDOWN_LABEL: OnLButtonDownLabel(); return TRUE;
    case SFTTREEN_LBUTTONDBLCLK_LABEL: OnLButtonDownLabel(); return TRUE;
    }
    return FALSE;
}

#endif
```

C++/OWL, CheckTab Sample

The complete source code can be found in the directory C:\SftTree\Samples\BC\Checktab.

```
//
// Build a response table for all messages/commands handled
// by the application.
//
DEFINE_RESPONSE_TABLE1(TSftTreeTable, TSftTree)
    EV_SFTTREEN_LBUTTONDOWN_LABEL_AT_CHILD(EvLButtonLabel),
    EV_SFTTREEN_LBUTTONDBLCLK_LABEL_AT_CHILD(EvLButtonLabel),
//{{TSftTreeTableRSP_TBL_BEGIN}}
    EV_WM_KEYDOWN,
//{{TSftTreeTableRSP_TBL_END}}
END_RESPONSE_TABLE;


//{{TSftTreeTable Implementation}}


TSftTreeTable::TSftTreeTable (TWindow* parent, int id, int x, int y, int w, int h,
TModule* module):
    TSftTree(parent, id, x, y, w, h, module)
{
}

TSftTreeTable::TSftTreeTable (TWindow* parent, int id, TModule* module):
    TSftTree(parent, id, module)
{
}


TSftTreeTable::~TSftTreeTable ()
{
    Destroy();

    delete m_pLabelBitmapYes;            /* label bitmap */
    delete m_pLabelBitmapNo;             /* label bitmap */

}

void TSftTreeTable::SetupWindow()
{
    m_pLabelBitmapYes = NULL;            /* label bitmap */
    m_pLabelBitmapNo = NULL;             /* label bitmap */

    SetShowHeader(TRUE);           /* Show column headers */

    /* Register the label bitmap size.  All label bitmaps used */
    /* must be the same size.  Only one bitmap needs to be     */
    /* registered, even if several are used.                   */

    m_pLabelBitmapYes = new TBitmap(*GetApplication(), IDB_CHECKYES);
    m_pLabelBitmapNo = new TBitmap(*GetApplication(), IDB_CHECKNO);
                                         /* Load a label bitmap */
    SetItemLabel(-1, *m_pLabelBitmapYes);
                                         /* Register the label bitmap size */

    SetShowTreeLines(FALSE);       /* No tree lines */

    SetShowButtons(FALSE);         /* No expand/collapse buttons */

    SetShowGrid(TRUE);             /* Show grid */
```

```
    SetShowTruncated(TRUE);        /* Show ... if truncated */

    SetNoFocusStyle(SFTTREE_NOFOCUS_FRAME);
                                   /* No focus, show frame */

    SetSelTextOnly(TRUE);          /* Select text only */

    SetShowHeaderButtons(TRUE);    /* Show column header as buttons */

    /* Define columns */

    {
        SFTTREE_COLUMN_EX aCol[2] = {
          { 0, 0,                          /* Reserved */
            100,                           /* Width (in pixels) */
            ES_LEFT,                       /* Cell alignment */
            ES_LEFT | SFTTREE_HEADER_UP,   /* Title style */
            TEXT("Option"),                /* Column header title */
            NULL, NULL,                    /* Reserved field and bitmap handle */
            SFTTREE_BMP_RIGHT },           /* Bitmap alignment */
          { 0, 0,                          /* Reserved */
            -1,                            /* Width (in pixels) */
            ES_LEFT,                       /* Cell alignment */
            ES_LEFT | SFTTREE_HEADER_UP,   /* Title style */
            TEXT("Description"),           /* Column header title */
            NULL, NULL,                    /* Reserved field and bitmap handle */
            SFTTREE_BMP_RIGHT }            /* Bitmap alignment */
        };
        SetColumns(2, aCol);       /* Set column attributes */
    }

    /* Set the row header attributes.                          */

    SetShowRowHeader(SFTTREE_ROWSTYLE_TITLECOUNT1);
                                   /* Row style */

    SetRowHeaderStyle(ES_CENTER | SFTTREE_HEADER_UP);
                                   /* Row header style */

    /* Set the row/column header attributes                    */

    SetShowRowColHeaderButton(FALSE);
                                   /* Row/column header as title */

    //SetRowColText(TEXT("?"));      /* Row/column header text */

    SetRowColHeaderStyle(ES_LEFT | SFTTREE_HEADER_UP);
                                   /* Row/column header style */
}

void TSftTreeTable::EvKeyDown (uint key, uint repeatCount, uint flags)
{
    TSftTree::EvKeyDown(key, repeatCount, flags);

    if (key == ' ')
        EvLButtonLabel();
    else
        TSftTree::EvKeyDown(key, repeatCount, flags);
}

void TSftTreeTable::EvLButtonLabel()
{
```

```
    // The item was clicked
    int index = GetCaretIndex();
    if (index >= 0)
        SetCheck(index, !GetCheck(index));
}
```