ClearJump™

ScrollMark™

SDK Programmer's Guide

ScrollMark™ SDK Programmer's Guide

Printed in the United States of America

Published by ClearJump
1702-H Meridian Avenue #325
San Jose, CA 95125, USA
+1 (408) 404-6962
info@clearjump.com

For information about other ClearJump products, visit our web site at
www.clearjump.com

1.0-11.13.01

TABLE OF CONTENTS

1. INTRODUCTION

ClearJump's ScrollMark is a new kind of graphical user interface (GUI) control that brings convenient and intuitive bookmarking capability to scrolling controls such as text boxes, list boxes, and document view windows. Unlike traditional bookmarking features in today's applications that simply iterate through a series of saved points or provide a list of saved locations in a separate dialog box, ScrollMark bookmarks are readily visible to the user and immediately accessible with a simple click of the mouse. As shown in Figure 1, ScrollMark bookmarks also support user-definable labels that appear as tooltips when the mouse is over the bookmark. This makes it easy to identify what the bookmark refers to.

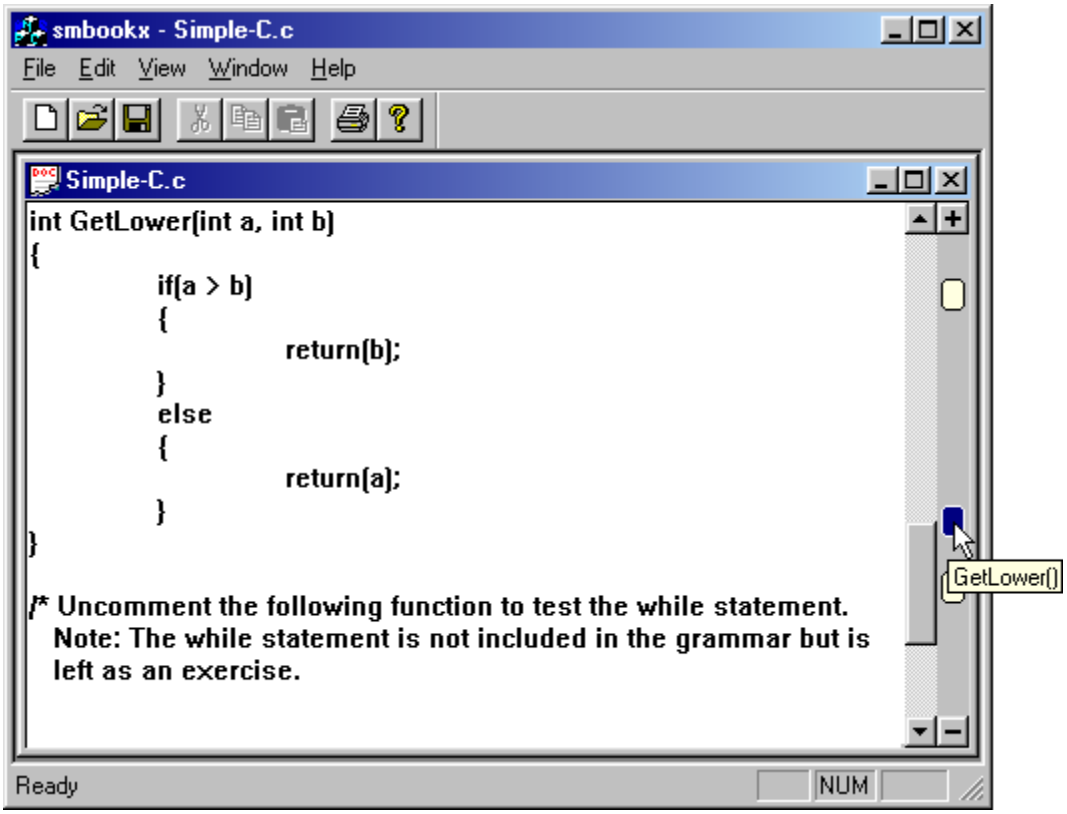

Figure 1: ScrollMark control in a text editor

Since ScrollMark bookmarks appear beside the control to which they apply, they can be used on more than one control without any confusion to the user. For example, a Web browser might have a list of places to visit in a list beside the main document view window. Both of these controls have scrolling capability. One ScrollMark control can be used with the list to save and jump to

positions within it and one can be used with the document view to save and jump to positions within the document.

To add bookmarks, simply click the Add Bookmark (+) button at the top of the ScrollMark control when the text box, list box, etc. is at the desired position. To jump to a bookmarked location, simply click on the bookmark. And to remove a bookmark, select it by clicking on it and then click the Delete Bookmark (-) button. Hold down the Ctrl or Shift key when clicking on a bookmark to select more than one, for example when removing more than one bookmark.

For some hands-on experience with ScrollMark bookmarks, try the ScrollMark Book text editor described in **Chapter 6, Example Programs**.

Typographical and Other Conventions

Source code listings, function declarations, program output and any text that appears on the screen appears in a `fixed pitch Courier` typeface. All other text is in the standard Times typeface. The electronic versions of this text contain hyperlinks that appear in **bold blue text** and can be clicked on with the mouse.

Throughout this document, the term "ScrollMark" refers to the ClearJump ScrollMark product. The terms "ScrollMark bookmark" and "bookmark" refer to ScrollMark bookmarks. The terms "ScrollMark control", "ScrollMark bar", and "bar" refer to the ScrollMark scrollbar-like control.

2. INSTALLATION & SUPPORT

System Requirements

- Microsoft Windows 95, 98, Me, NT 4, or 2000
- ANSI-compliant C/C++, or any other language that can call Windows-standard Dynamically Linked Library (DLL) functions
- To use the project files included with the sample programs requires Microsoft Visual C++ 5 or higher

Installing the ScrollMark SDK

To install the ScrollMark SDK

1. Insert the ScrollMark CD into your CD-ROM or DVD drive.
2. Wait a few seconds. If setup doesn't begin automatically, click the Windows Start menu and select Run. Then, click the Browse button and navigate to the drive into which you inserted the CD. Select the file, setup.exe, and click the Open button. Finally, click the OK button to begin the installation.
3. Follow the on-screen instructions to complete the installation.

The ScrollMark SDK installation creates the following folders and files on your system:

Folder	File	Description
bin		
	smctrl.dll	ScrollMark control DLL
	smbook.exe	Single-document text editor sample application
	smbookx.exe	Multiple-document text editor sample application
	smdlg.exe	Dialog controls sample application
docs		
	ScrollMarkSDK.pdf	Programming Manual
include		
	smapi.h	ScrollMark SDK header file
	scrollmarkctrl.h	ScrollMark C++ class header file
lib		

smctrl.lib	ScrollMark library file to be used with smctrl.dll
samples	ScrollMark samples
redist	Redistributable files.

Developer Support

For quick, accurate answers to your support questions, contact ClearJump Developer Support via e-mail at

smsupport@clearjump.com

When submitting a support request, be sure to include the following information:

1. Windows version.
2. ScrollMark version. From Windows Explorer, right click on smctrl.dll, select Properties, then click on the Version tab.
3. Compiler name and version.
4. Complete text of error messages, if any.
5. Detailed description of the question or problem. If applicable, include code snippets and/or steps to reproduce the problem.

3. USING THE SCROLLMARK SDK

SDK Components

The ScrollMark SDK consists of the following components:

- ScrollMark DLL
- ScrollMark API header files
- Sample programs
- Documentation

Project Configuration

In general, all that is required to use ScrollMark is to add the ScrollMark library to your project. The ScrollMark control is distributed as a standard Windows DLL named smctrl.dll. The DLL doesn't require any other libraries to work and is located in the Bin folder as listed in the chapter, **Installation & Support**. Connecting ScrollMark to your project is simple. Following are the steps required depending upon whether you'll be using the **C-language API** or the **C++ classes**.

Using the ScrollMark API

1. Include smapi.h in the source files that call the ScrollMark API. The smapi.h file is located in the Include folder.
2. Include smctrl.lib in the link settings. The file is located in the Lib folder.
3. Make sure that your executable can access smctrl.dll.

Using the ScrollMark C++ Classes

1. Include scrollmarkctrl.h in the source files that call the ScrollMark C++ class, CScrollmarkCtrl. The scrollmarkctrl.h file is located in the Include folder.
2. Include smctrl.lib in the link settings. The file is located in the Lib folder.
3. Make sure that your executable can access smctrl.dll.

Programming the ScrollMark Control

To use the ScrollMark control in your application, it must be connected to a scrollbar in a parent

window (e.g. list box control, rich edit control). The ScrollMark control will then dock itself to the edge of the window containing the scrollbar. If the scrollbar is vertical, the ScrollMark will appear to the right of it. If the scrollbar is horizontal, the ScrollMark will appear below it.

The ScrollMark control is not an HWND-based object. It is painted in the parent's non-client area. As a result, the ScrollMark control requires that the parent window pass all messages to the control first in order to be able to paint its content, adjust its layout and process mouse events.

The control's message handler returns values that indicate whether or not the message has been processed by the control. If the message has been processed by the control, the window procedure should not continue with any other processing of this message.

Connecting the Control

If you're using the C-language API, connecting the ScrollMark control to a scrollbar in your project consists of the following steps:

1. Make sure that the parent window has a scrollbar enabled.

2. Create the ScrollMark control by calling the SmCreate( ) function. The SmCreate( ) function returns a handle to the ScrollMark control that is used for all subsequent API calls to the control. Use the DOCK_RIGHT property for vertical scrollbars and DOCK_BOTTOM for horizontal scrollbars. If a window contains both a horizontal and a vertical scrollbar, you can create a separate ScrollMark control for each scrollbar.

   Tip: The parent window's window procedure can create the ScrollMark control in its WM_CREATE message handler.

3. The parent window's window procedure must call the SmWindowProc( ) function for every message that is sent to the window by the operating system. This enables the ScrollMark control to process messages related to its operation. Without this communication, the ScrollMark control will not operate.

4. When the parent window is destroyed, its window procedure must call the SmDestroy( ) function to release all resources allocated by the ScrollMark control. The SmDestroy( ) function can be called from the parent window's WM_DESTROY message handler.

If your application is written in C++, then the convenient CScrollmarkCtrl class can be used instead of the C-language API. This class is just a wrapper and the steps to use it are essentially the same as for the C-language API. Below is an example of a window procedure using the ScrollMark control.

```
//global ScrollMark bar handle
HSMBAR g_hScrollmarkBar = NULL;

//window procedure
LRESULT CALLBACK MyWndProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
      //let the ScrollBark bar process its messages
      if( g_hScrollmarkBar )
      {
            LRESULT lres;
            if(SmWindowProc( g_hScrollmarkBar, uMsg, wParam, lParam, &lres ))
                  //the message has been processed by the ScrollMark bar
                  return lres;
      }

      switch( uMsg )
      {
      case WM_CREATE:
            //create a vertical ScrollMark bar
            g_hScrollmarkBar = SmCreate( hwnd, 0,
                                         DOCK_RIGHT, SMSHAPE_RECT,
                                         NULL, NULL );
            //invalidate the ScrollMark control, it will adjust
            //the window area to include the ScrollMark bar
            if( g_hScrollmarkBar ) SmInvalidate( g_hScrollmarkBar );
            break;

      case WM_CLOSE:
            if( g_hScrollmarkBar )
            {
                  //destroy the ScrollMark bar
                  SmDestroy( g_hScrollmarkBar );
                  g_hScrollmarkBar = NULL;
            }
            PostQuitMessage( 0 );
            break;
      }
      return DefWindowProc( hwnd, uMsg, wParam, lParam );
}
```

Using the Control with MFC

If your application uses the Microsoft Foundation Classes, it's best to use ScrollMark's C++ wrapper class, CScrollmarkCtrl. This class can be used with any MFC class that is derived from the CWnd class.

To add a ScrollMark control to an MFC application, do the following:

1. In the parent window's class, define a CScrollmarkCtrl member. If the window contains two scrollbars (i.e. vertical and horizontal) then two CScrollmarkCtrl members can be used, one for each scrollbar.

2. Using the VC++ Class Wizard, insert a message handler for the WM_CREATE message or overload the OnInitialUpdate( ) method for a view-based window and in it create the CScrollmarkCtrl members defined in step 1. This can be done by calling the CScrollmarkCtrl::Create( ) method. Use the DOCK_RIGHT property for vertical scrollbars and DOCK_BOTTOM for horizontal scrollbars.

3. Using the Class Wizard, overload the WindowProc( ) method in the parent window's class. The overloaded WindowProc( ) method must call the CScrollmarkCtrl::WindowProc( ) method for each CScrollmarkCtrl member defined in step 1. This enables each ScrollMark control to process any messages that are intended for it and is key to its operation.

Below is an example of an edit view with a vertical ScrollMark bar.

```
//class declaration
class CSmbookView : public CEditView
{
public:
      CScrollmarkCtrl m_sm; //ScrollMark bar class
... //other declarations
}

//overloaded OnInitialUpdate()
void CSmbookView::OnInitialUpdate()
{
      // TODO: Add your specialized code here and/or call the base class
      m_sm.Create( m_hWnd );  //create ScrollMark bar
      CEditView::OnInitialUpdate();
}

//overloaded window procedure
LRESULT CSmbookView::WindowProc(UINT message, WPARAM wParam, LPARAM lParam)
{
      // TODO: Add your specialized code here and/or call the base class
      //let the ScrollMark bar process it messages
      LRESULT lres;
      if( m_sm.WindowProc(message, wParam, lParam, &lres) )
            //the message has been processed by the ScrollMark bar
            return lres;

      return CEditView::WindowProc(message, wParam, lParam);
}

//overloaded EN_CHANGE handle
void CSmbookView::OnChange()
{
      m_sm.Invalidate();  //update the ScrollMark bar
```

```
	CEditView::OnEditChange();
}
```

4. SCROLLMARK API

The ScrollMark API is a C-language interface to the ScrollMark engine. The smapi.h header file contains the API definitions.

List of Functions

Bar Manipulation Functions

SmCreate: Creates the ScrollMark bar.
SmDestroy Destroys the ScrollMark bar.
SmInvalidate: Updates the layout of and redraws the bar.
SmWindowProc: The ScrollMark bar window messages procedure.

Bar Property Functions

SmEnable: Enables/disables the ScrollMark bar.
SmEnableAutoDelete: Enables/disables auto-delete.
SmGetWidth: Gets the ScrollMark bar width.
SmIsAutoDelete: Checks the state of auto-delete.
SmIsEnabled: Checks the enable/disable status of the bar.
SmIsHorizontal: Checks the horizontal orientation.
SmIsTooltips: Checks the state of tooltips.
SmSetWidth: Sets the ScrollMark bar width.

Bookmark Manipulation Functions

SmActivateBookmark: Activates a bookmark at a particular index.
SmAddBookmark: Adds a ScrollMark bookmark.
SmDeleteBookmark: Deletes a ScrollMark bookmark.
SmGetActiveBookmark: Gets the index of the active bookmark (single-selection only).
SmSelectBookmark: Sets the ScrollMark bookmark selection.

Bookmark Property Functions

SmGetBookmark: Gets a ScrollMark bookmark at a given position.
SmGetBookmarkCount: Gets the number of ScrollMark bookmarks.
SmGetBookmarkInfo: Gets ScrollMark bookmark information.
SmGetBookmarkSize: Gets the ScrollMark bookmark size.

SmSetBookmarkLabel: Sets the ScrollMark bookmark label.
SmSetBookmarkSize: Sets the ScrollMark bookmark size.

Button Property Functions

SmGetButtonTooltip: Gets the tooltip for the button.
SmSetButtonTooltip: Sets the tooltip for the button.

Callback Functions

(*PSMNOTIFY): The developer-supplied callback function.

Bar Manipulation Functions

SmCreate

This function creates the ScrollMark bar.

Declaration

```
HSMBAR SmCreate( HWND hParent, DWORD style, BARDOCK dock, SMSHAPE
smshape, PSMNOTIFY pcb, PVOID pUser );
```

Parameters

`hParent`

Handle of the window containing the ScrollMark bar.

`style`

The following style flags govern the look and behavior of the ScrollMark bar and can be combined.

`SM_STYLE_AUTODELETE`: Automatically removes bookmarks that are out of range.
`SM_STYLE_NOTOOLTIPS`: Turns off tooltips for bookmarks and buttons.

`dock`

The following docking properties determine the location of the ScrollMark bar.

`DOCK_BOTTOM`: Place the bar below the host control.
`DOCK_RIGHT`: Place the bar to the right of the host control.

`smshape`

Defines the shape of the ScrollMark bookmarks. Valid shapes include

`SMSHAPE_ELLIPSE`: Elliptical bookmarks.
`SMSHAPE_RECT`: Rectangular bookmarks.
`SMSHAPE_ROUNDRECT`: Rounded rectangle bookmarks.

`pcb`

Points to a user callback function. See **PSMNOTIFY** for more information.

`pUser`

Pointer to user-defined data that will be sent to the user callback function.

Return Values

Returns a handle to the created ScrollMark bar or NULL if the function failed.

Remarks

The ScrollMark bar is destroyed by calling the SmDestroy function. The SmInvalidate function can be called immediately after SmCreate to redraw the parent window with the ScrollMark control.

When the `SM_STYLE_AUTODELETE` style is set, bookmarks that are beyond the range of the associated scrollbar will be automatically removed. The bookmarks are removed when the ScrollMark control is updated with SmInvalidate() or as a result of a window message. A bookmark might also fall out of range when the scrollbar range is changed.

See Also

BARDOCK, **PSMNOTIFY**, **SMSHAPE**, **SmEnableAutoDelete**, **SmSetBookmarkSize**, **SmSetWidth**, **SmDestroy**

SmDestroy

This function destroys the ScrollMark bar object.

Declaration

```
UINT SmDestroy( HSMBAR hsm );
```

Parameters

`hsm`

Handle to the ScrollMark bar to be destroyed.

Return Values

Returns `SMRET_SUCCESS` or a **ScrollMark error code**.

See Also

SmCreate

SmInvalidate

Invalidates the ScrollMark bar and causes it to be updated and redrawn.

Declaration

```
void SmInvalidate( HSMBAR hsm );
```

Parameters

`hsm`

Handle to the ScrollMark bar.

Return Values

None

Remarks

SmInvalidate must be called after the range of the scrollbar associated with the ScrollMark control is changed. SmInvalidate will reallocate the bookmarks in accordance with the new range.

SmWindowProc

This function is called by an application before any other message processing procedures whenever a Windows message is sent to the parent window.

Declaration

```
BOOL SmWindowProc( HSMBAR hsm, UINT message, WPARAM wParam, LPARAM
lParam, LRESULT* plResult );
```

Parameters

`hsm`

Handle to the ScrollMark bar.

`message`

Message code indicating the type of Windows message.

`wParam`

The w-parameter of the Windows message.

`lParam`

The l-parameter of the Windows message.

`plResult`

Pointer to a variable that receives the result of the message processing and depends on the type of message.

Return Values

Returns TRUE if the message has been processed and the calling application should return the value pointed to by plResult to the system without calling any other message handling procedures. Returns FALSE if the message is not handled by the ScrollMark bar and the application should continue with its own processing of the message.

Remarks

The ScrollMark control is not an HWND-based object. It is painted in the parent's non-client area. As a result, the ScrollMark control requires that the parent window pass all messages to the control first to be able to paint its content, adjust its layout and process mouse events.

The control's message handler returns values that indicate whether or not the message has been processed by the control. If the message has been processed by the control, the window procedure should not continue with any other processing of this message.

Example

```
HSMBAR g_hScrollmarkBar = NULL;
LRESULT CALLBACK MyWndProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
      //let the ScrollMark bar process its messages
      if( g_hScrollmarkBar )
      {
            LRESULT lres;
            if(SmWindowProc( g_hScrollmarkBar, uMsg, wParam, lParam, &lres ))
                  //the message has been processed by the ScrollMark bar
                  return lres;
      }
      switch( uMsg )
      {
      ... window message handlers
      }
      ...
}
```

Bar Property Functions

SmEnable

Enables (visible) or disables (invisible) the ScrollMark bar.

Declaration

```
UINT SmEnable( HSMBAR hsm, BOOL bEnable );
```

Parameters

`hsm`

Handle to the ScrollMark bar.

`bEnable`

Boolean value that indicates the state of the ScrollMark bar. TRUE makes the bar visible, FALSE makes it invisible.

Return Values

Returns `SMRET_SUCCESS` or a **ScrollMark error code**.

Remarks

This function can be used when the application enables/disables associated scrollbars dynamically.

See Also

SmIsEnabled

SmEnableAutoDelete

Enables or disables the auto-delete feature. The auto-delete feature is also controlled by the `SM_STYLE_AUTODELETE` style.

Declaration

```
UINT SmEnableAutoDelete( HSMBAR hsm, BOOL bDelete );
```

Parameters

`hsm`

Handle to the ScrollMark bar.

`bDelete`

Boolean value that indicates the auto-delete state of the ScrollMark bar. TRUE enables auto-delete and FALSE disables it.

Return Values

Returns `SMRET_SUCCESS` or a **ScrollMark error code**.

Remarks

When the `SM_STYLE_AUTODELETE` style is set, bookmarks that are beyond the range of the associated scrollbar will be automatically removed. The bookmarks are removed when the ScrollMark control is updated with SmInvalidate or as a result of a window message. A bookmark might also fall out of range when the scrollbar range is changed.

See Also

SmCreate, **SmInvalidate**, **SmIsAutoDelete**

SmGetWidth

Gets the current width of the ScrollMark bar in pixels. If the bar is drawn horizontally, this function gets the height of the bar.

Declaration

```
UINT SmGetWidth( HSMBAR hsm );
```

Parameters

`hsm`

Handle to the ScrollMark bar.

Return Values

Width (vertical bar orientation) or height (horizontal bar orientation) of ScrollMark bar in pixels.

Remarks

By default, the ScrollMark bar width is equal to the width of the scrollbar as defined by the operating system. An application can override it by using the SmSetWidth function.

See Also

SmSetWidth

SmIsAutoDelete

Indicates if auto-delete is enabled for the ScrollMark bar. The auto-delete feature is also controlled by the `SM_STYLE_AUTODELETE` style.

Declaration

```
BOOL SmIsAutoDelete( HSMBAR hsm );
```

Parameters

`hsm`

Handle to the ScrollMark bar.

Return Values

Returns TRUE if auto-delete is turned on (i.e. the `SM_STYLE_AUTODELETE` style is set), FALSE if not.

Remarks

When the `SM_STYLE_AUTODELETE` style is set, bookmarks that are beyond the range of the associated scrollbar will be automatically removed. The bookmarks are removed when the ScrollMark control is updated with SmInvalidate or as a result of a window message. A bookmark might also fall out of range when the scrollbar range is changed.

See Also

SmCreate

SmIsEnabled

Indicates whether or not the ScrollMark bar is enabled.

Declaration

```
BOOL SmIsEnabled( HSMBAR hsm );
```

Parameters

`hsm`

Handle to the ScrollMark bar.

Return Values

Returns TRUE if the bar is enabled (visible) or FALSE if it is disabled (invisible).

See Also

SmEnable

SmIsHorizontal

Indicates whether the ScrollMark bar is in a horizontal or vertical orientation.

Declaration

```
BOOL SmIsHorizontal( HSMBAR hsm );
```

Parameters

`hsm`

Handle to the ScrollMark bar.

Return Values

Returns TRUE if the bar is horizontal or FALSE if it's vertical.

Remarks

Generally, the bar will be horizontal when BARDOCK is set to `DOCK_BOTTOM` and will be vertical when BARDOCK is set to `DOCK_RIGHT`.

See Also

SmCreate, **BARDOCK**

SmIsTooltips

Indicates whether or not tooltips are enabled. By default, tooltips are enabled.

Declaration

```
BOOL SmIsTooltips( HSMBAR hsm );
```

Parameters

`hsm`

Handle to the ScrollMark bar.

Return Values

Returns TRUE if tooltips are turned on, FALSE if not (i.e. the `SM_STYLE_NOTOOLTIPS` style is set).

Remarks

Tooltips are enabled in SmCreate when the `SM_STYLE_NOTOOLTIPS` style is not set. There is no way to disable tooltips after the ScrollMark bar is created.

See Also

SmCreate, **SmGetButtonTooltip**, **SmSetButtonTooltip**, **SmSetBookmarkLabel**

SmSetWidth

Sets the width (or height if bar is in horizontal orientation) of the ScrollMark bar.

Declaration

```
UINT SmSetWidth( HSMBAR hsm, UINT nWidth );
```

Parameters

`hsm`

Handle to the ScrollMark bar.

`nWidth`

Width (or height if bar is in horizontal orientation) of bar in pixels. If nWidth is zero the ScrollMark bar width is reset to the default width. See remarks.

Return Values

Returns `SMRET_SUCCESS` or a **ScrollMark error code**.

Remarks

By default, the ScrollMark bar width is equal to the width of the scrollbar as defined by the operating system. An application can override it by using the SmSetWidth function.

See Also

SmGetWidth

Bookmark Manipulation Functions

SmActivateBookmark

Activates a ScrollMark bookmark at a particular index and re-positions the connected scrollbar accordingly.

Declaration

```
UINT SmActivateBookmark( HSMBAR hsm, UINT nIndex );
```

Parameters

`hsm`

Handle to the ScrollMark bar.

`nIndex`

Index of the bookmark to activate.

Return Values

Returns `SMRET_SUCCESS` or a **ScrollMark error code**.

See Also

SmGetActiveBookmark

SmAddBookmark

Adds a ScrollMark bookmark to the ScrollMark bar.

Declaration

```
UINT SmAddBookmark( HSMBAR hsm, const BOOKMARKINFO* pInfo, UINT* pnIndex
);
```

Parameters

`hsm`

Handle to the ScrollMark bar.

`pInfo`

Pointer to a BOOKMARKINFO data structure.

`pnIndex`

Pointer to a variable that receives the index of the new bookmark. The pnIndex parameter can be NULL if the index is not needed.

Return Values

Returns `SMRET_SUCCESS` or a **ScrollMark error code**.

See Also

BOOKMARKINFO, **SmDeleteBookmark**

Example

The following code fragment adds an unnamed bookmark at the current position.

```
...
BOOKMARKINFO bi;
bi.nPos = GetScrollPos ( hWnd, SB_VERT );
bi.pUser = NULL;
bi.pszLabel = NULL;
SmAddBookmark( g_hScrollmark, bi, NULL );
...
```

SmDeleteBookmark

Deletes a ScrollMark bookmark from the ScrollMark bar.

Declaration

```
UINT SmDeleteBookmark( HSMBAR hsm, UINT nIndex, BOOL bInvalidate );
```

Parameters

`hsm`

Handle to the ScrollMark bar.

`nIndex`

Index of the bookmark to be deleted.

`bInvalidate`

TRUE indicates that the bar's region should be invalidated after deleting the bookmark.

Return Values

Returns `SMRET_SUCCESS` or a **ScrollMark error code**.

Remarks

When deleting many bookmarks at once, the bInvalidate flag should be set to FALSE to avoid unnecessary redrawing of the control. After the last call to SmDeleteBookmark, the SmInvalidate functional can be called to redraw the control.

See Also

SmAddBookmark

SmGetActiveBookmark

Gets the index of the currently active ScrollMark bookmark. Invalid when multiple bookmarks are selected.

Declaration

```
UINT SmGetActiveBookmark( HSMBAR hsm );
```

Parameters

`hsm`

Handle to the ScrollMark bar.

Return Values

Index of the currently active bookmark or `SMRET_INVINDEX` if no bookmark is active or multiple bookmarks are selected.

Remarks

The active bookmark is the one that the user last clicked on with the mouse or that was last activated with the SmActivateBookmark function. There can only be one active bookmark in the ScrollMark bar. On the other hand, one or more bookmarks can be selected. For example, to delete multiple bookmarks at once, more than one is selected.

See Also

SmActivateBookmark

SmSelectBookmark

Changes the selection state of a ScrollMark bookmark.

Declaration

```
UINT SmSelectBookmark( HSMBAR hsm, UINT nIndex, BOOL bSelect );
```

Parameters

`hsm`

Handle to the ScrollMark bar.

`nIndex`

Index of the desired bookmark.

`bSelect`

TRUE if the bookmark's state is selected or FALSE if it is not selected.

Return Values

Returns `SMRET_SUCCESS` or a **ScrollMark error code**.

Remarks

To get the current selection state of a bookmark, use the SmGetBookmarkInfo function. Selecting a bookmark doesn't change the position of the associated scrollbar. Use SmActivateBookmark to scroll to the bookmark's position.

See Also

SmGetBookmarkInfo, **SmActivateBookmark**

Bookmark Property Functions

SmGetBookmark

Finds the index of the ScrollMark bookmark that is located at a particular position in the bar.

Declaration

```
UINT SmGetBookmark( HSMBAR hsm, int nPos );
```

Parameters

`hsm`

Handle to the ScrollMark bar.

`nPos`

Position of the bookmark in scrollbar units.

Return Values

Returns the index of the bookmark or `SMRET_INVINDEX` if no bookmark is found.

See Also

SmAddBookmark, **SmGetBookmarkInfo**

SmGetBookmarkCount

Gets the number of ScrollMark bookmarks in the ScrollMark bar.

Declaration

```
UINT SmGetBookmarkCount( HSMBAR hsm );
```

Parameters

`hsm`

Handle to the ScrollMark bar.

Return Values

Returns the number of bookmarks in the bar.

See Also

SmGetBookmarkInfo

SmGetBookmarkInfo

Gets information about a specific ScrollMark bookmark.

Declaration

```
INT SmGetBookmarkInfo( HSMBAR hsm, UINT nIndex, BOOKMARKINFO* pInfo );
```

Parameters

`hsm`

Handle to the ScrollMark bar.

`nIndex`

Index of the desired bookmark.

`pInfo`

Pointer to the BOOKMARKINFO structure that will contain the information.

Return Values

Returns `SMRET_SUCCESS` or a **ScrollMark error code**.

See Also

BOOKMARKINFO

SmGetBookmarkSize

Gets the current size of ScrollMark bookmarks in pixels.

Declaration

```
UINT SmGetBookmarkSize( HSMBAR hsm );
```

Parameters

`hsm`

Handle to the ScrollMark bar.

Return Values

Returns the current size of bookmarks in pixels.

See Also

SmSetBookmarkSize

SmSetBookmarkLabel

Assigns a label to the specified ScrollMark bookmark.

Declaration

```
UINT SmSetBookmarkLabel( HSMBAR hsm, UINT nIndex, LPCTSTR pszLabel );
```

Parameters

`hsm`

Handle to the ScrollMark bar.

`nIndex`

Index of the desired bookmark.

`pszLabel`

Pointer to a null-terminated string containing the text of the label or NULL for no label.

Return Values

Returns `SMRET_SUCCESS` or a **ScrollMark error code**.

Remarks

The label appears as a tooltip if tooltips are enabled in SmCreate. To retrieve the current label, use the SmGetBookmarkInfo function.

See Also

SmGetBookmarkInfo

SmSetBookmarkSize

Sets the size of ScrollMark bookmarks in the ScrollMark bar.

Declaration

```
UINT SmSetBookmarkSize( HSMBAR hsm, UINT nSmSize );
```

Parameters

`hsm`

Handle to the ScrollMark bar.

`nSmSize`

Size for the bookmarks in pixels. If a value of zero is specified, the default size is used. The default size is the current size of the system scrollbar buttons.

Return Values

Returns `SMRET_SUCCESS` or a **ScrollMark error code**.

See Also

SmGetBookmarkSize

Button Property Functions

SmGetButtonTooltip

Gets the tooltip for the button.

Declaration

```
LPCTSTR SmGetButtonTooltip( HSMBAR hsm, UINT smid );
```

Parameters

`hsm`

Handle to the ScrollMark bar.

`smid`

ID value of the button whose tooltip you want to get. The following buttons are available:

`SMID_BTN_ADDBOOKMARK`: Add Bookmark (+) button.
`SMID_BTN_DELBOOKMARK`: Delete Bookmark (-) button.

Return Values

Pointer to a string containing the tooltip text.

Remarks

The button tooltips are disabled if the `SM_STYLE_NOTOOTIPS` style is set in SmCreate.

See Also

SmSetButtonTooltip, **SmCreate**

SmSetButtonTooltip

Sets the tooltip for the button.

Declaration

```
BOOL SmSetButtonTooltip( HSMBAR hsm, UINT smid, LPCTSTR pszTooltip );
```

Parameters

`hsm`

Handle to the ScrollMark bar.

`smid`

ID value of the button whose tooltip you want to set. The following buttons are available:

`SMID_BTN_ADDBOOKMARK`: Add Bookmark (+) button.
`SMID_BTN_DELBOOKMARK`: Delete Bookmark (-) button.

`pszTooltip`

Pointer to the tooltip string.

Return Values

Returns TRUE if the tooltip is successfully set, otherwise FALSE.

Remarks

The button tooltips are disabled if the `SM_STYLE_NOTOOLTIPS` style is set in SmCreate.

See Also

SmGetButtonTooltip, **SmCreate**

Callback Functions

(*PSMNOTIFY)

This is the developer-supplied notification callback function that is called when a ScrollMark event occurs.

Declaration

```
typedef BOOL (*PSMNOTIFY)( UINT nMessage, LPARAM lParam, PVOID pUserData
);
```

Parameters

`nMessage`

A message code indicating the type of notification message. The message code can be one of the following:

`SMN_PAINT`: Paint an object. The lParam parameter is a pointer to an SMPAINT structure.
`SMN_COMMAND`: Command event. The lParam parameter is a pointer to an SMOBJ structure. The `SMN_COMMAND` notification is generated when the user clicks a button or bookmark.

`lParam`

Pointer to a data structure. The type of data structure depends on the type of message code specified by nMessage.

`pUserData`

Pointer to user data. Available for passing any other data.

Return Values

Returns TRUE if the message has been processed by the callback function and no default processing is necessary thereafter.

Remarks

This callback capability enables the functionality of the ScrollMark control to be customized. Using the SMN_PAINT notification, an application can owner-draw any element in the ScrollMark control.

The callback function is specified in the SmCreate function.

See Also

SmCreate, **SMN_PAINT**, **SMN_COMMAND**, **SMPAINT**, **SMOBJ**

Data Types

BARDOCK

This enumerated type defines the ScrollMark control's docking property.

Declaration

```
enum BARDOCK;
```

Values

`DOCK_BOTTOM`

Docks the control below the scrollbar (horizontal orientation).

`DOCK_RIGHT`

Docks the control to the right of the scrollbar (vertical orientation).

See Also

SmCreate

SMSHAPE

This enumerated type defines the shape of the ScrollMark bookmarks.

Declaration

```
enum SMSHAPE;
```

Values

`SMSHAPE_ELLIPSE`
: Creates elliptical bookmarks.

`SMSHAPE_RECT`
: Creates rectangular bookmarks.

`SMSHAPE_ROUNDRECT`
: Creates rounded-rectangular bookmarks.

See Also

SmCreate

BOOKMARKINFO

This structure defines the properties of a bookmark.

Declaration

```
typedef struct BOOKMARKINFO;
```

Fields

`int nPos`
Position of the bookmark in the associated scrollbar.

`LPCSTR pszLabel`
Bookmark's label. See **SmSetBookmarkLabel**.

`PVOID pUser`
User data associated with the bookmark.

`SMSHAPE nShape`
Bookmark shape. See **SMSHAPE**. Valid only when retrieving bookmark information.

`RECT rect`
Rectangular region of the bookmark in the container window coordinates. Valid only when retrieving bookmark information.

`UINT nState`
State of the bookmark. The state can be any combination of the following flags:

`EMS_DISABLED`: Object is disabled.
`EMS_HIDDEN`: Object is hidden.
`EMS_MOUSEOVER`: Object has mouse cursor over it.
`EMS_SELECTED`: Object is selected.

Valid only when retrieving bookmark information.

`BOOL bOutOfRange`
Out-of-range flag. TRUE if the bookmark is no longer within range of the control. Valid only when retrieving bookmark information.

See Also

SmGetBookmarkInfo, **SmSetBookmarkLabel**, **SMSHAPE**

SMOBJ

This data structure defines the ScrollMark object that generated an event.

Declaration

```
typedef struct SMOBJ;
```

Fields

`UINT nObjectID`

Identifier of the object that generated this event. This value can be one of following:

`SMID_BTN_ADDBOOKMARK`: Add Bookmark (+) button.
`SMID_BTN_DELBOOKMARK`: Delete Bookmark (-) button.
`SMID_BOOKMARK`: Bookmark object.

`UINT nSmIndex`

Index of the bookmark that generated the event. Valid only when nObjectID is `SMID_BOOKMARK`.

See Also

PSMNOTIFY, **SMID_BTN_ADDBOOKMARK**, **SMID_BTN_DELBOOKMARK**, **SMID_BOOKMARK**

SMPAINT

This data structure defines the object or region that is to be painted.

Declaration

```
typedef struct SMPAINT;
```

Fields

`UINT nObjectID`

Identifier of the object to be painted. This value can be one of the following:

`SMID_BTN_ADDBOOKMARK`: Add Bookmark (+) button.
`SMID_BTN_DELBOOKMARK`: Delete Bookmark (-) button.
`SMID_BOOKMARK`: Bookmark object.
`SMID_SCROLLMARKBAR`: ScrollMark bar region.

`HDC hDC`

Device context.

`UINT nSmIndex`

Index of the bookmark to be painted. Valid only when nObjectID is `SMID_BOOKMARK`.

`RECT rect`

Rectangle area of the object.

`UINT nState`

State of the object. The state can be any combination of the following `EMS_xxx` state flags:

`EMS_DISABLED`: Object is disabled.
`EMS_HIDDEN`: Object is hidden.
`EMS_MOUSEOVER`: Object has mouse cursor over it.
`EMS_SELECTED`: Object is selected.

See Also

PSMNOTIFY, **SMN_PAINT**, **SMID_BTN_ADDBOOKMARK**, **SMID_BTN_DELBOOKMARK**, **SMID_BOOKMARK**, **SMID_SCROLLMARKBAR**

Macros

`EMS_DISABLED`
: Object state is disabled.

`EMS_HIDDEN`
: Object state is hidden.

`EMS_MOUSEOVER`
: Object has mouse cursor over it.

`EMS_SELECTED`
: Object is selected.

`SMID_BOOKMARK`
: Identifier for bookmark object.

`SMID_BTN_ADDBOOKMARK`
: Identifier for Add Bookmark (+) button.

`SMID_BTN_DELBOOKMARK`
: Identifier for Delete Bookmark (-) button.

`SMID_SCROLLMARKBAR`
: Identifier for ScrollMark bar.

`SMN_COMMAND`
: Command notification event.

`SMN_PAINT`
: Paint notification event.

`SM_STYLE_AUTODELETE`
: Enables bookmark auto-delete style.

`SM_STYLE_BORDER`
: Enables flat border style.

`SM_STYLE_NOTOOLTIPS`
: Disables tooltips.

Return/Error Codes

`SMRET_SUCCESS`
: Function completed successfully, no errors.

`SMRET_NOMEMORY`
: Not enough memory.

`SMRET_SMEXISTS`
: Bookmark exists.

`SMRET_NOTFOUND`
: Object was not found.

`SMRET_BADPARAMS`
: Invalid parameters.

`SMRET_BADHANDLE`
: Invalid handle.

`SMRET_INVINDEX`
: Invalid index.

5. C++ Interface

CScrollmarkCtrl

The CScrollmarkCtrl class is actually a simple wrapper class for the ScrollMark API. All the methods are implemented as inline functions in the scrollmarkctrl.h file. As a result, there is no C++ file for this class, just the header file. So, when using this class, simply #include the scrollmarkctrl.h file in your source.

The names of the methods in the class are equivalent to the names of the corresponding functions in the C-language API less the "Sm" prefix. For example, the SmCreate function is called Create in the CScrollmarkCtrl class.

The wrapper is defined as follows:

Public Fields

`HSMBAR m_hsm;`
ScrollMark bar handle.

Public Methods

`inline CScrollmarkCtrl( );`
Constructor.

`virtual ~CScrollmarkCtrl( );`
Destructor.

```
inline BOOL Create( HWND hParent, DWORD style = SM_STYLE_AUTODELETE, BARDOCK
dock = DOCK_RIGHT, SMSHAPE smshape = SMSHAPE_ROUNDRECT, PSMNOTIFY pcb = NULL,
PVOID pUser = NULL );
```
Creates the ScrollMark control. See **SmCreate**.

`inline UINT Destroy( );`
Destroys the ScrollMark bar object. See **SmDestroy**.

`inline void Invalidate( );`
Updates the layout and redraws the bar. See **SmInvalidate**.

```
inline BOOL WindowProc( UINT message, WPARAM wParam, LPARAM lParam, LRESULT*
plres );
```
Window messages procedure. See **SmWindowProc**.

`inline UINT Enable( BOOL bEnable );`
Enables or disables the ScrollMark bar. See **SmEnable**.

```
inline BOOL IsEnabled( );
```
Checks the enable/disable status of the bar. See **SmIsEnabled**.

```
inline UINT GetWidth( );
```
Returns the ScrollMark bar width in pixels. See **SmGetWidth**.

```
inline UINT SetWidth( UINT nWidth );
```
Sets the bar width in pixels. See **SmSetWidth**.

```
inline UINT EnableAutoDelete( BOOL bDelete );
```
Sets the auto-delete flag. See **SmEnableAutoDelete**.

```
inline BOOL IsAutoDelete( );
```
Checks the SM_STYLE_AUTODELETE style. See **SmIsAutoDelete**.

```
inline BOOL IsTooltips( );
```
Checks the SM_STYLE_NOTOOLTIPS style. See **SmIsTooltips**.

```
inline BOOL IsHorizontal( );
```
Checks the horizontal orientation. See **SmIsHorizontal**.

```
inline LPCTSTR GetButtonTooltip( UINT smid );
```
Gets the tooltip for the button. See **SmGetButtonTooltip**.

```
inline BOOL SetButtonTooltip( UINT smid, LPCTSTR pszTooltip );
```
Sets the tooltip for the button. See **SmSetButtonTooltip**.

```
inline UINT ActivateBookmark ( UINT nIndex );
```
Activates a bookmark at a particular index. See **SmActivateBookmark**.

```
inline UINT AddBookmark( const BOOKMARKINFO& info, UINT* pnIndex = NULL );
```
Adds a new bookmark. See **SmAddBookmark**.

```
inline UINT DeleteBookmark( UINT nIndex, BOOL bInvalidate = TRUE );
```
Removes the specified bookmark. See **SmDeleteBookmark**.

```
inline UINT GetActiveBookmark( );
```
Gets the index of the active bookmark (single-selection only). See **SmGetActiveBookmark**.

```
inline UINT GetBookmark( int nPos );
```
Gets the index of the bookmark at a given position. See **SmGetBookmark**.

```
inline UINT SelectBookmark( UINT nIndex, BOOL bSelect );
```
Changes the selection state of the specified bookmark. See **SmSelectBookmark**.

```
inline UINT GetBookmarkCount( );
```
Returns the number of bookmarks. See **SmGetBookmarkCount**.

```
inline UINT GetBookmarkInfo( UINT nIndex, BOOKMARKINFO* pInfo );
```
Retrieves information about the specified bookmark. See **SmGetBookmarkInfo**.

```
inline UINT GetBookmarkSize( );
```
Returns the current bookmark size in pixels. See **SmGetBookmarkSize**.

```
inline UINT SetBookmarkLabel( UINT nIndex, LPCTSTR pszLabel );
```
Sets the bookmark's label. See **SmSetBookmarkLabel**.

```
inline UINT SetBookmarkSize( UINT nSmSize );
```
Sets the bookmark's size in pixels. See **SmSetBookmarkSize**.

6. EXAMPLE PROGRAMS

The ScrollMark SDK includes several example programs. These programs illustrate how the ScrollMark control works with various user interface components such as edit boxes, list boxes, and scrollbars. In addition, the project files and source code for the example programs are included as samples of the use of the ScrollMark API. The following sections describe the general operation of each program.

ScrollMark Dialog Example

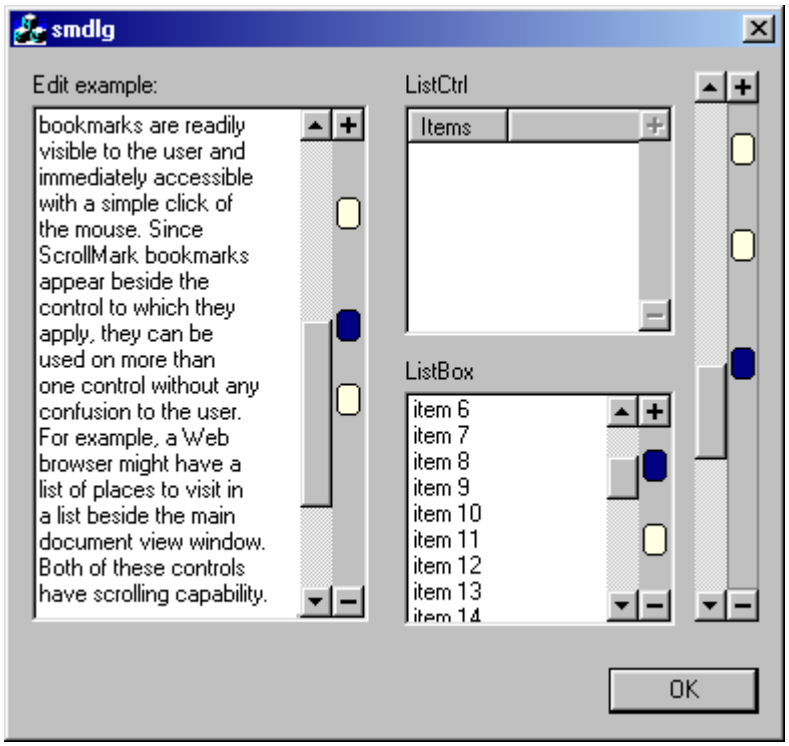

Figure 2: ScrollMark dialog controls example

The example program shown in Figure 2, smdlg.exe, illustrates how a ScrollMark control can be used with a variety of user interface components that support scrolling such as edit boxes, list controls, and list boxes. Even a simple scrollbar can be enhanced with a ScrollMark control to provide bookmarking capability.

To try out the various controls, move the scrollbar to a desired position and press the Add

Bookmark button (+) to add a bookmark at that location. Move the scrollbar to a couple different positions and add more bookmarks. Then click on the bookmarks to jump back to the saved positions. Selecting a bookmark and then clicking on the Delete Bookmark button (-) will remove it.

Note that for the text box control, you'll have to add some text before the scrollbar, and hence the ScrollMark control, becomes active.

ScrollMark Book Examples

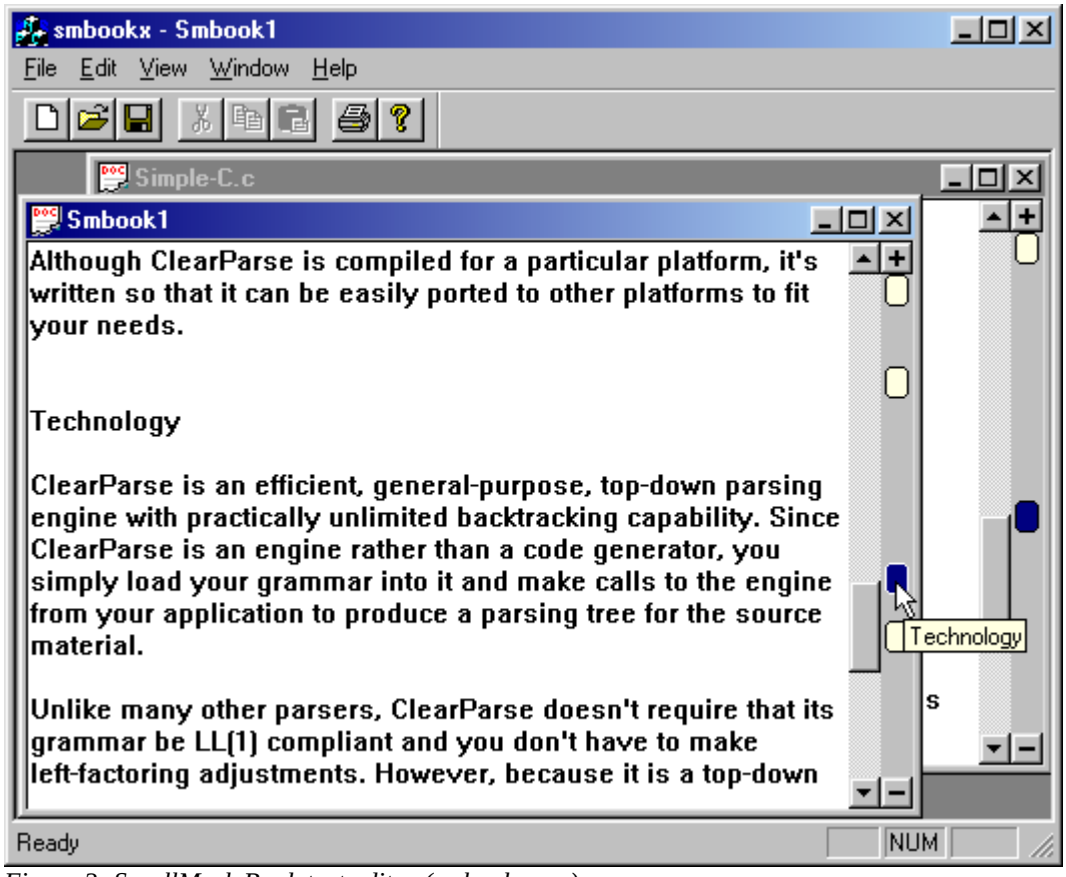

Figure 3: ScrollMark Book text editor (smbookx.exe)

The ScrollMark Book example programs, smbook.exe and smbookx.exe, are functional text editors that can load single or multiple text files, respectively. In addition to basic editing and clipboard functions, they also include ScrollMark bookmarking capability for each file that is loaded. As shown in Figure 3, the file Smbook1 and Simple-C.c have their own, independent bookmarks.

To use either ScrollMark Book editor, load a text file by selecting Open from the File menu. Then, add bookmarks as desired by clicking on the Add Bookmark (+) button (or pressing Ctrl+F2 in smbookx). To go to a specific bookmark, click on it (or press F2 in smbookx until you arrive at the desired bookmark). To remove bookmarks, click on the desired bookmark and click on the Delete Bookmark (-) button (or press Shift+F2 in smbookx). Multiple bookmarks can be selected by holding down the Shift or Ctrl key while clicking.

In the smbookx editor, you can also assign labels to the bookmarks. In Figure 3, you can see that the mouse is over a bookmark labeled "Technology". To put a label on a bookmark, from the View menu go to ScrollMark and then select Properties or simply press the F3 key. The label text you enter appears as a tooltip when the mouse cursor is over it.

INDEX

T

V

W

WARRANTY

CLEARJUMP AND ITS SUPPLIERS DISCLAIM ALL WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT.

NEITHER CLEARJUMP NOR ITS SUPPLIERS WARRANT THAT THE FUNCTIONS CONTAINED IN THE SOFTWARE WILL MEET YOUR REQUIREMENTS OR THE OPERATION OF THE SOFTWARE WILL BE UNINTERRUPTED OR ERROR-FREE. YOU AGREE TO ASSUME FULL RESPONSIBILITY FOR THE SELECTION OF THE SOFTWARE TO ACHIEVE YOUR INTENDED RESULTS, AND FOR THE INSTALLATION, USE AND RESULTS OBTAINED FROM THE SOFTWARE. YOU ALSO ASSUME THE ENTIRE RISK OF ANY USE OF THE SOFTWARE. NO DISTRIBUTOR, DEALER OR ANY OTHER ENTITY OR PERSON IS AUTHORIZED TO EXPAND OR ALTER THIS WARRANTY.

THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS, AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM COUNTRY/STATE TO COUNTRY/STATE. SOME COUNTRIES/STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED CONDITIONS AND WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. CLEARJUMP DISCLAIMS ALL CONDITIONS AND WARRANTIES OF ANY KIND IF THE SOFTWARE HAS BEEN CUSTOMIZED, REPACKAGED OR ALTERED IN ANY WAY BY YOU OR ANY OTHER PARTY.

EXCLUSIVE REMEDY AND LIMITATION OF LIABILITY

THE SOLE AND EXCLUSIVE REMEDY FOR ANY BREACH OF THE LIMITED WARRANTY SET FORTH ABOVE WILL BE, AT CLEARJUMP'S OPTION, (a) RETURN OF THE PURCHASE PRICE OR (b) REPLACEMENT OF THE DEFECTIVE SOFTWARE. IN NO EVENT WILL CLEARJUMP OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, INCIDENTAL, SPECIAL OR CONSEQUENTIAL DAMAGES OR FOR ANY LOST PROFITS, LOST SAVINGS, LOST REVENUES OR LOST DATA ARISING FROM OR RELATING TO THE SOFTWARE, ITS USE OR INABILITY OF USE. IN NO EVENT WILL CLEARJUMP'S OR ANY OF ITS SUPPLIERS' LIABILITY OR DAMAGES TO YOU OR ANY OTHER PARTY EVER EXCEED THE AMOUNT PAID BY YOU TO USE THE SOFTWARE, REGARDLESS OF THE FORM OF THE CLAIM (WHETHER IN CONTRACT, TORT OR OTHERWISE). SOME COUNTRIES/STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

ALL USE OF THE SOFTWARE IS SUBJECT TO THE TERMS OF THE SOFTWARE LICENSE AGREEMENT.