Barking Spider Software, Inc.
Tree Control
for
Windows

Barking Spider Software, Inc. Beaverton, Oregon
(503) 629-8236 VOICE/FAX/AUTO (503) 324-0187 VOICE

Introduction

What is the Barking Spider Tree Control ?

The Tree Control is part of a small, fast Windows DLL that removes the hassles of adding a tree display to your product. Using the DLL's powerful exported API, the programmer has complete control over bitmaps, text, spacing, line colors and more.

The Barking Spider Tree Control currently requires the following execution environment:
  Windows 3.1 or greater in standard or enhanced mode.

To program for the Tree Control your chosen language must be able to:
  Access DLL API's
  Process Windows messages
NOTE: We will be releasing Visual Basic and C++ interfaces

What is in this package?

Pertaining to the Tree Control, this package contains:
  The Tree/List Control DLL.
  The export library for the Tree/List Control DLL.

This manual.
BSCORE.H - The core C language header file. This file contains the constants, structures, notification messages, and error codes return from the exported API.
BSTREE.H - The Tree Control C language header file. This file contains the function declarations needed to interface with the DLL.
Treedem1 - Tree demo program 1. This program places the Tree Control window on it's main window. This program interfaces with the DLL using the DLL's API.
Treedem2 - Tree demo program 2. This program places the Tree Control window in a modal dialog. This program shows how to use the Tree Control with a dialog box editor using the Custom Control interface and the exported API.
Treedem3 - Tree demo program 3. This program places the Tree Control window on each of it's MDI children. The program interfaces with the Tree Control via the exported API.

What can you do with the Tree Control ?

The Tree Control allows you to create and modify visual displays of trees.

Tree modification APIs:
- Node addition
- Node insertion
- Node deletion
- Erase entire tree
- Add/change user data

Visual modification APIs:
- Set bitmaps
- Set icons
- Change bitmaps
- Change icons
- Change line colors
- Toggle line drawing
- Set bitmap/icon spacing
- Set font
- Change node text

In addition, the Tree Control simplifies user/tree interaction. All painting and scrolling tasks are handled by the control. Using a simple messaging scheme, the application is notified whenever a node is selected or double clicked. Keyboard interactions with the Tree Control produce the same messages as the equivalent mouse interactions.

Additional Tree Control features:

- Drag and Drop encapsulation
- System color change synchronization
- Programmer defined data store for each node
- User or control owned strings
- Multiple bitmaps per line
- Complete node hit test notification
- Node relation (parent, child, sibling) access
- Node deletion callback

Getting Started

What do you install?

To install the Barking Spider Tree Control run the install batch file provided.
Copy the Tree Control DLL to a directory in your path or into your project's executable directory.
Copy the Tree Control import library and the header files, **BSCORE.H** and **BSTREE.H**, into the appropriate directory(s) on your hard disk.

Programming Overview

This overview is provided as an aid to programmers who are new to using the Barking Spider Software Tree Control. It provides a brief overview of interaction with the Tree Control. For more complete information on APIs please refer to the API Reference in this manual. For a more indepth technical overview of the Tree Control's operations and APIs please refer to the **BSCORE.H** and **BSTREE.H** header files included in your package.

What will the control display?

The control will display a graphic representation of the tree described to it by the application. The display will start at the root and traverse in a depth first order displaying each node as it is encountered. One node is shown per line. The children of a node will be displayed indented, and on lines following the parent.

How do I create the tree display window?

The tree display creation step allows the programmer to specify display placement, size and window styles. This step does not create a tree do be displayed, it creates the frame and scroll bars, and readies the control's engine for processing.

The tree display can be created by:

Calling the DLL's **BST_CreateTree** exported API (NOTE: See **Treedem1** for sample code)

OR

Embedding a Tree Control in a dialog box with a dialog editor. (NOTE: This uses the Custom Control information embedded in the DLL. See **Treedem2** for sample code)

You must save the window handle of the newly created tree display window for later API calls. Because the Tree Control can have multiple active instances, the window handle is used to identify particular display windows.

How do I construct a tree?

The tree to be displayed must be created using the following steps:

Create the root node of the tree. This is accomplished by calling the **BST_AddChildrenToParent** API and passing NULL as the node's parent. The **BST_AddChildrenToParent** API requires a pointer to an array of structures that define each node being added. Since there is only one root per tree, the call to construct the root node will pass in a pointer to a single structure defining the root.

After calling the **BST_AddChildrenToParent** API to create the root, record the lpTreeNode member in the definition structure. This member is filled in by the Tree Control, and it contains the unique identifier that will be used to refer to this node in future operations.
NOTE: Many applications will not need to store the node identifiers beyond the creation stage.

Repeat this process for all nodes in the tree, substituting the correct parent pointer, and node definition structure(s) in the **BST_AddChildrenToParent** call.

Communications from the control

Communication with the Tree Control is simple. All of the user input is filtered by the control and notification messages are sent to the parent window of the control.

When an event occurs that the application needs to know about the Control issues a notification message. The lParam of the message contains a pointer to a **SELECT_NOTIF** structure. This structure conatins a pointer to the affected node and flags that describe the state of the node.

The three Tree Notification messages and their causes:

WM_BST_SELECT_NOTIF. This message is triggered by:
- A single left mouse button click while over a tree node.
- Highlight movement with Up arrows, Down arrows, Home, End, Page Up, Page Down, Ctrl Up arrows, or Ctrl Down arrows.

WM_BST_SELCET_NOTIF_DBLCLK. This message is triggered by:
- A left mouse button double click while over a tree node. NOTE: A **WM_BST_SELECT_NOTIF** is sent on the first click.
- A Carriage Return while a node is selected.
- The '+' key if the currently selected node has no children.
- The '-' key if the currently selected node has children.

WM_BST_DO_DRAG. This message is triggered by:
- Depressing and holding the left mouse button over a tree node, while moving the mouse a predetermined distance.

Demo API Reference

NOTE: This is not a complete listing of the Tree Control APIs. The APIs listed immediately belw were chosen to be documented because they are critical to the Tree Control's operation. A complete API reference is supplied with each purchase of the Tree Control.

BST_AddChildrenToParent ( )
BST_CreateTree ( )
BST_DeleteChildrenOfParent ( )

BST_EraseTree ( )
BST_SetBitmap ( )
BST_SetBitmapAndActiveBitmap ( )
BST_SetBitmapSpace ( )

Complete API Listing:

BST_AddChildrenToParent ( )
BST_ChangeNodeText ( )
BST_ChangeUserData ( )
BST_ConvertPointToSelectNotif ( )
BST_CreateTree ( )
BST_DeleteChildrenOfParent ( )
BST_DeleteNode ( )
BST_DragAcceptFiles ( )
BST_EraseTree ( )
BST_GetActiveNode ( )
BST_GetFirstChildOfParent ( )
BST_GetNextSibling ( )
BST_GetParent ( )
BST_GetPreviousSibling ( )
BST_GetVersion ( )
BST_InsertSiblingNodes ( )
BST_SetActiveNode ( )
BST_SetBitmap ( )
BST_SetBitmapAndActiveBitmap ( )
BST_SetBitmapSpace ( )
BST_SetDeleteNodeCallBack ( )
BST_SetFont ( )
BST_SetIcon ( )
BST_SetLineColor ( )
BST_SetXSpaceBeforeText ( )
BST_ShowActiveNode ( )
BST_ShowLines ( )

****

BST_AddChildrenToParent()

short _export FAR PASCAL **BST_AddChildrenToParent**(
HWND **hwndTree**,
LP_TREE_NODE **lpParentTreeNode**,
WORD **wNodeDefCount**,
LP_TREE_NODE_DEF **lpTreeNodeDef**)

Description:

This API allows the application to add one or more nodes to the tree as children of the given parent.

Parameters:

HWND **hwndTree -** This argument is the handle of the Tree Control display window that will contain the new tree node(s). This handle may be retrieved from a Tree Control display creation (see **BST_CreateTree())** or by querying the HWND of a dialog embedded Tree Control (see Treedem2.c, TreeDlgProc(), WM_INITDIALOG handler).

LP_TREE_NODE **lpParentTreeNode** - This argument points to the parent tree node which will receive the children specified by **lpTreeNodeDef**. If **lpParentTreeNode** is NULL then this tells the Tree Control that **lpTreeNodeDef** describes the root of the tree. (NOTE: There can be only ONE root node)

WORD **wNodeDefCount** - **wNodeDefCount** contains the number of nodes to be added.

LP_TREE_NODE_DEF **lpTreeNodeDef**: **lpTreeNodeDef** is a pointer to an array of TREE_NODE_DEFs that describes nodes to be added. There must be one complete TREE_NODE_DEF structure for each node specified by **wNodeDefCount**.(NOTE: Make sure that the structures are properly filled in, uninitialized fields may cause errors)

Comments:

If the given parent already has children, then the new children are added after the last pre-existing child.

This function will return an error if there are memory allocation problems or the Tree Control's limits are exceeded. This proceedure steps through the TREE_NODE_DEF array processing each structure one at a time. When an error occurs the Tree Control will set the **lpTreeNode** member of the TREE_NODE_DEF structure that caused the error to NULL.. The offending TREE_NODE_DEF and all subsequent TREE_NODE_DEFs in the array will not be added to the tree.

Return Codes:

BST_NO_ERROR
BST_ERR_MEMORY_ALLOC_FAILED
BST_ERR_LEVEL_LIMIT_EXCEEDED

BST_ERR_TOO_MANY_NODES
BST_ERR_ONLY_ONE_ROOT_ALLOWED
BST_ERR_INVALID_PARENT_FOR_INSERTION

****

BST_CreateTree()

```
HWND _export FAR PASCAL BST_CreateTree(
                HANDLE hInstance,
                HWND hwndApp,
                int x,
                int y,
                int nWidth,
                int nHeight,
                DWORD dwStyle,
                DWORD dwExStyle)
```

Description:

Creates an empty Tree Control display window. BST_CreateTree ( ) ORs the style bits specified in dwStyle to the Tree Control's required styles and then creates a display window with the appropriate styles.

Parameters:

HANDLE **hInstance** - Instance associated with the creation of the Tree Control window.

HWND **hwndApp** - Window handle of the parent window that contains the Tree Control display.

int **x** - X coordinate of the upper left corner of the Tree Control display, relative to the origin of the client area of the parent window.

int **y** - Y coordinate of the upper left corner of the Tree Control display, relative to the origin of the client area of the parent window.

int **nWidth** - Width of the Tree Control in device (pixel) units.

int **nHeight** - Height of the Tree Control in device (pixel) units.

DWORD **dwStyle** - Application requested window styles for the tree display window..

DWORD **dwExStyle** - Application requested CreateWindowEx ( ) extended styles.

Comments:

If this call is successful then it will return a window handle for the newly created Tree Control display window. This window handle is used to identify this particular Tree Control display window.

Return Codes:

If successful, BST_CreateTree ( ) will return the window handle of the newly created Tree Control. If failure, then a NULL will be returned.

****

BST_DeleteChildrenOfParent()

short _export FAR PASCAL **BST_DeleteChildrenOfParent**(
HWND **hwndTree**,
LP_TREE_NODE **lpTreeNode**)

Description:

Deletes all of the child tree nodes of the given parent tree node specified in **lpTreeNode**.

Parameters:

HWND **hwndTree** - This argument specifies the Tree Control that contains the nodes to be destroyed.

LP_TREE_NODE **lpTreeNode** - **lpTreeNode** points to a tree node who's children, if it has any, will be destroyed.

Comments:

When a tree node is deleted, all pointers to it are no longer valid. The Tree Control frees the deleted tree node's memory. If a notification of a tree node's deletion is desired, then the application can use a Tree Control API function, **BST_SetDeleteNodeCallBack ( )**, to register a callback function that the Tree Control will call just before deletion of the node.

NOTE: It is up to the application to manage any memory pointed to by the lpUserData member of the tree node.

Return Codes:

BST_NO_ERROR

****

BST_EraseTree()

short _export FAR PASCAL **BST_EraseTree(** HWND **hwndTree**)

Description:

This proceedure removes all of the tree nodes in the specified Tree Control but does not destroy the Tree Control display window.

Parameters:

HWND **hwndTree -** This argument specifies the Tree Control that will destroy all of it's children including the root node. It does not invalidate the Tree Control display window handle. This handle can still be used in any of the APIs, such as **BST_AddChildrenToParent ( )**.

Comments:

The Tree Control frees memory allocated by the control for the deleted tree node. The Control will NOT free any memory pointed to by the **lpUserData** of the node. If the application has assigned a pointer to dynamically allocated memory in the lpUserData member of the tree node, it is the responsibility of the application to free this memory.

If notification of a tree node's deletion is desired, then the application can use the **BST_SetDeleteNodeCallBack( )** API function to register a callback function that the Tree Control will call just before deletion of the node. If the application has assigned a pointer to the lpUserData member of the tree node, it is the responsibility of the application to free this memory if necessary.

Return Codes:

BST_NO_ERROR

****

BST_SetBitmap()

short _export FAR PASCAL **BST_SetBitmap**(

HWND **hwndTree**,
short **nBitmap**,
LP_TREE_NODE **lpTreeNode**,
HBITMAP **hBitmap**)

Description:

This function assigns a bitmap handle to a specified tree node for the specified bitmap space. The old bitmap or icon is erased but not deleted and then the new bitmap or icon is drawn.

Parameters:

HWND **hwndTree** - **hwndTree** is the Tree Control in which the tree node will receive the bitmap handle

short **nBitmap - nBitmap** is the number of the bitmap space in which this bitmap is to be displayed. The bitmap spaces are numbered from left to right and the number of the left most bitmap space is zero. The maximum bitmap space number is MAX_BITMAPS - 1.
Reference the Tree Control API documentation for **BST_SetBitmapSapce()** to learn the process of defining a bitmap space.

LP_TREE_NODE **lpTreeNode** - **lpTreeNode** is a pointer to the tree node that will be assigned the given bitmap handle.

HBITMAP **hBitmap** - **hBitmap** is the handle to the bitmap which will be drawn in the specified bitmap space for the given tree node.

Comments:

Bitmap/icon handles are NOT the property of the Tree Control. The tree control treats the bitmap/icon handle as read only. It will only use the handle to draw the bitmap/icon associated with the tree node. If the tree node already has a bitmap/icon handle stored in the specified bitmap position then the old handle is simply overwritten.

It is the responsibility of the application to manage creation and destruction of bitmaps/icons. If the application deletes/destroys bitmaps/icons before the tree nodes referencing them are destroyed, then the Tree Control may possibly reference invalid bitmap/icon handles.

For more information regarding bitmaps/icons and tree nodes, refer to the TREE_NODE structure documentation.

Return Codes:

BST_NO_ERROR

****

BST_SetBitmapAndActiveBitmap

```
short _export FAR PASCAL BST_SetBitmapAndActiveBitmap(
                HWND  hwnd,
                short nBitmap,
                LP_TREE_NODE lpTreeNode,
                HBITMAP hBitmap,
                HBITMAP hActiveBitmap)
```

Description:

This API allows the Tree Control to manage the changing of bitmaps when a node is selected. The **hBitmap** will be used to draw the bitmap if the tree node is not active or else **hActiveBitmap** will be used to draw the bitmap if the tree node is active.

Parameters:

HWND **hwndTree** - **hwndTree** specifies the Tree Control display to operate on.

short **nBitmap - nBitmap** is the index of the bitmap space to display the bitmaps. The bitmap spaces are numbered from left to right and the first space is numbered zero.

LP_TREE_NODE **lpTreeNode -** This is a pointer to the tree node that the bitmaps will be assigned to.

HBITMAP **hBitmap** - **hBitmap** is a handle to the bitmap which will be drawn in the bitmap space **nBitmap,** when the node is not selected.

HBITMAP **hActiveBitmap** - **hActiveBitmap** is the handle to the bitmap which will be drawn in the specified bitmap space when the node is active.

Comments:

This proceedure assigns two bitmap handles to a specified tree node for the specified bitmap space. It erases, but does not delete the old bitmap, and draws the new bitmap.

It is the application's responsibility to manage the creation and deletion of bitmaps. If the application destroys the bitmaps that the Tree Control is using, then there will be trouble in paradise.

For more information regarding bitmaps and tree nodes, refer to the

TREE_NODE structure documentation.

Return Codes:

BST_NO_ERROR

****

BST_SetBitmapSpace()

short _export FAR PASCAL **BST_SetBitmapSpace**(
HWND **hwndTree**,
short **nBitmap**,
short **nWidth**,
short **nHeight**,
BOOL **bCenterBitmap**)

Description:

This proceedure allows the application to globally specify the placement of the bitmaps displayed with each node.

Parameters:

HWND **hwndTree** - A handle to the Tree Control display which is being formatted.

short **nBitmap -** Identifies the bitmap space. Bitmap spaces are numbered from left to right starting at zero.

short **nWidth -** The width, in pixels, of the bitmap space. See the comments for a discussion of bitmap spaces.

short **nHeight** - **nHeight** is the height, in pixels, of the bitmap space.

BOOL **bCentered -** If this parameter is TRUE then the bitmaps will be centered in the bitmap space.

Comments:

This API allows the application to specify the space reserved for bitmaps to the left of the node text. This 'whitespace' is present on all lines displaying a node even if the corresponding bitmap handle is NULL.

If either **nHeight** or **nWidth** is zero, then the bitmap space does not reserve any screen real estate. This is used to deactivate a bitmap space.

The dimensions of a bitmap space are defined globally for all tree nodes. This is to align the columns of bitmaps, and text strings. In addition hit testing is performed based on the bitmap spaces not on the bitmap sizes.

The bitmap space is aligned so that the space is centered with the node's text line. If the **bCentered** paramter is set to TRUE then the bitmap or icon will be centered between the right and left boundaries of the space. If **bCentered** is set to FALSE then the bitmap or icon will be aligned with the left boundary of the space.

If a bitmap is larger than it's corresponding bitmap space then the bitmap will be clipped. An icon that is larger than the bitmap space will not be clipped, so care must be taken to ensure that the bitmap spaces are large enough when icons are used.

Return Codes:

BST_NO_ERROR