NAME

Blt_Tree − Tree data object.

SYNOPSIS

#include <bltTree.h>

struct Blt_Tree {
Tcl_Alloc(*size*)

Tcl_Free(*ptr*)

char *
Tcl_Realloc(*ptr, size*)

ARGUMENTS

int	*size*	(in)	Size in bytes of the memory block to allocate.
char	**ptr*	(in)	Pointer to memory block to free or realloc.

DESCRIPTION

These procedures provide a platform and compiler independent interface for memory allocation. Programs that need to transfer ownership of memory blocks between Tcl and other modules should use these routines rather than the native **malloc()** and **free()** routines provided by the C run-time library.

Tcl_Alloc returns a pointer to a block of at least *size* bytes suitably aligned for any use.

Tcl_Free makes the space referred to by *ptr* available for further allocation.

Tcl_Realloc changes the size of the block pointed to by *ptr* to *size* bytes and returns a pointer to the new block. The contents will be unchanged up to the lesser of the new and old sizes. The returned location may be different from *ptr*.

TREE OBJECT ROUTINES

The following library routines allow you to create and destroy tree objects. Each tree object has a name that uniquely identifies it. Tree objects can also be shared. For example, the **tree** and **hiertable** commands may access the same tree data object. Each client grabs a token associated with the tree. When all tokens are released the tree data object is automatically destroyed.

Blt_TreeCreate	Create a tree data object and optionally obtains a token associated with it.
Blt_TreeExists	Indicates if a tree by a given name exists.
Blt_TreeGetToken	Obtains a token for an existing tree data object.
Blt_TreeReleaseToken	Releases a token for a tree data object. The tree object is deleted when all outstanding tokens have been released.
Blt_TreeName	Returns the name of the tree object.
Blt_TreeChangeRoot	Specifies a node as the new root to a tree.

TREENODE ROUTINES

Tree objects initially contain only a root node. You can add or delete nodes with the following routines.

Blt_TreeCreateNode	Creates a new child node for a given parent in the tree.
Blt_TreeDeleteNode	Deletes a node and its children.

Blt_TreeNodeId	Returns the unique node identifier for a node.
Blt_TreeGetNode	Gets a node based upon its identifier.
Blt_TreeFindChild	Searches for a child node given by its label in a parent node.
Blt_TreeNodeLabel	Returns the current label for a node.
Blt_TreeRelabelNode	Resets a node's label.
Blt_TreeNodePath	Returns the fullpath to a node.
Blt_TreeNodeDepth	Returns the depth of the node.
Blt_TreeNodeDegree	Returns the number of children for a node.
Blt_TreeIsLeaf	Indicates if a node has no children.
Blt_TreeIsBefore	Indicates if a node is before another node in depth-first search order.
Blt_TreeIsAncestor	Indicates if a node is an ancestor or another.
Blt_TreeSortNode	Sorts the children of a node.
Blt_TreeSize	Returns the number of nodes in a node and its descendants.
Blt_TreeMoveNode	

NODE NAVIGATION

Each node can have zero or more children nodes. These routines let you navigate the tree hierarchy.

Blt_TreeNodeParent	Returns the parent node.
Blt_TreeFirstChild	Returns the first child of a parent node.
Blt_TreeLastChild	Returns the last child of a parent node.
Blt_TreeNextSibling	Returns the next sibling node in the parent's list of children.
Blt_TreePrevSibling	Returns the previous sibling node in the parent's list of children.
Blt_TreeRootNode	Returns the root node of the tree.
Blt_TreeNextNode	Returns the next node in depth-first order.
Blt_TreePrevNode	Returns the previous node in depth-first order.
Blt_TreeEndNode	Returns the last node in the tree as determined by depth-first order.
Blt_TreeApply	Walks through a node and all it descendants, applying a given callback procedure.
Blt_TreeApplyDFS	Walks through a node and all it descendants in depth-first search order, applying a given callback procedure.
Blt_TreeApplyBFS	Walks through a node and all it descendants in breadth-first search order, applying a given callback procedure.

NODE DATA VALUES

Data values can be stored at any node. Values have by both a string key and a Tcl_Obj value. Data value keys do not have to be homogenous across all nodes (i.e. nodes do not have to contain the same keys). There is also a special node array data type.

Blt_TreeGetValue	Gets the node data value given by a key.
Blt_TreeValueExists	Indicates if a node data value given by a key exists.
Blt_TreeSetValue	Sets a node's value of a key.

Blt_TreeUnsetValue	Remove the node data value and key.
Blt_TreeGetArrayValue	Gets the node data array value given by a key and an array index.
Blt_TreeSetArrayValue	Sets the node data array value given by a key and an array index.
Blt_TreeUnsetArrayValue	Remove the node data array value.
Blt_TreeArrayValueExists	Determines if an array element by a given index exists.
Blt_TreeFirstKey	Returns the key of the first value in the node.
Blt_TreeNextKey	Returns the key of the next value in the node.
Blt_TreePrivateValue	Lock the value to current client, making it private.
Blt_TreePublicValue	Unlock the value so that all clients can access it.
Blt_TreeGetKey	

NODE TRACES

Blt_TreeCreateTrace	Sets up a trace callback to be invoked when the node value is read, set, or unset.
Blt_TreeDeleteTrace	Deletes an existing trace.

NODE EVENTS

Blt_TreeCreateEventHandler	Sets up a callback to be invoked when events (create, delete, relabel, etc) take place on a node.
Blt_TreeDeleteEventHandler	Deletes an existing node callback.

KEYWORDS

alloc, allocation, free, malloc, memory, realloc