tCNode Reference

This section is a quick reference to methods in tCNode template class. It is recommended you read the entire article before referring to this section.

Construction

Recall figure 2 - tCNode type definition and you clearly see you do not have to use the original template declaration to create a data-type. In fact, the macro TNODE_SET_TYPE is by far the best way.
The macro creates some types you can use in your code. Thus, if you want a type DataTree where int type as key and char * type as data you should use this way:

➢ TNODE_SET_TYPE(DataTree, char *, int)

Automatically the following types are created:
- TDataTree (tCNode<char *,int>)
- TDataTreeRef (tCNode<char *,int>::tcnode_ref)
- TDataTreePtr (tCNode<char *,int>::tcnode_ptr) and
- TDataTreeNodes (tCNode<char *,int>::tcnode_subnodes)

➢ void addDataSorter(_IN std::string _name, _IN _SORTER _receiver, _IN BOOL _recursive = FALSE)

Create a new data sorter in current node. A data sorter can include all tree nodes below it (_recursive = TRUE) or just the child nodes. Data sorter are executed in current node when refreshDataSorters is called.
Parameters
- _name [in, required]: A string that identifies a single data sorter
- _receiver [in, required]: A sort function pointer with prototype bool function(tcnode_ptr _p1, tcnode_ptr _p2)
- _recursive [in, optional]: Set TRUE to include all nodes and subnodes below current node.

Return Value
- None

➢ tcnode_ref addShortcut(_IN std::string _label, _IN std::vector<tcnode_key> &_parm)

Create a string shortcut to a node in the tree.
Parameters
- _label [in, required]: A string that identifies the shortcut
- _parm [in, required]: A vector that contains the full address of the node described by an array of keys

Return Value
- A reference to the current node.

➢ tcnode_ref createNode(_IN tcnode_data _data, _IN tcnode_key _key)

Create a new child node indexed by _key. If node already exists, the node data is replaced by new _data.
Parameters
- _data [in, required]: The data itself defined at template declaration
- _key [in, required]: The key itself defined at template declaration

Return Value
A reference to new child node or the existing one if the key already exists.

- ➢ **std::vector<tcnode_ptr>& getAllSubNodesByKey(_OUT std::vector<tcnode_ptr> &_parm, _IN tcnode_key _key)**

Select all subnodes from current node where _key matches. It is recursive.
Parameters
- _parm [out, required]: Pointers to nodes that matches _key
- _key [in, required]: The search key

Return Value
- _parm is returned filled with pointers to nodes that matches _key. If _key not found, _parm is returned as was passed.

- ➢ **long getCount(void)**

Count all subnodes from current node recursively.
Parameters
- None

Return Value
- Total subnodes

- ➢ **tcnode_data & getData(void)**

Get a reference to DATA in current node.
Parameters
- None

Return Value
- Reference to DATA

- ➢ **std::vector<tcnode_ptr> & getDataSorterByName(_IN std::string _name, _OUT BOOL &_is_valid)**

Return a reference to full data sorter data, the complete array of pointers. _is_valid must be tested to know if data sorter is valid.
Parameters
- _name [in, required]: Data sorter name
- _is_valid [out, required]: TRUE if data sorter returned is valid otherwise FALSE

Return Value
A reference to data sorter. An array of pointers to nodes. TNODE_PTR_TO_REF converts PTR to REF.

- ➢ **long getDeep(void)**

Get current node deep or level. root has deep = 0.
Parameters
- None

Return Value
- Long integer representing current deep

- ➢ **tcnode_ptr getFirstSubNodeByKey(_IN tcnode_key _key)**

Return a node given a KEY. The search starts in current node and it is recursive.
Parameters
- _key [in, required]: key to execute the search

Return Value
- A pointer to subnode if found. NULL if not found. Application can use TNODE_PTR_TO_REF macro to convert pointer into reference.

- **Long getId(void)**

Get a number that is unique to identify a node.
Parameters
- None

Return Value
- Long integer representing node identifier

- **tcnode_key & getKey(void)**

Get a reference to KEY in current node.
Parameters
- None

Return Value
- Reference to KEY

- **std::vector<tcnode_ptr> & getNextDataSorterInfo(_IN BOOL _begin, _OUT std::string &_name, _OUT BOOL &_recursive, _OUT _SORTER &_sortfunc, _OUT BOOL &_is_valid)**

List one by one data sorters in a node.
Parameters
- _begin[in, required]: TRUE indicates first data sort. FALSE list next one
- _name[out, required]: the data sorter name
- _recursive[out, required]: TRUE indicates data sorter was set as recursive
- _sortFunc[out, required]: Sort function run by data sort. A variable to be passed must be declared asT<type name>::SortPredCall
- _is_valid[out, required]: TRUE indicates data returned is valid. Application should test _is_validto know when data sort list finished

Return Value
- A reference to data sorter data itself. An array of pointers to nodes. TNODE_PTR_TO_REF converts PTR toREF.

- **tcnode_ptr getNodeByFullAddress(_IN std::vector<tcnode_key> &_parm)**

Return a node given an address. See getNodeFullAddress to know how to get a node address.
Parameters
- _parm [in, required]: array of nodes representing an address

Return Value
- A pointer to subnode if found. NULL if not found. If you want, you can use TNODE_PTR_TO_REF macro to convert pointer into reference.

- **tcnode_ptr getNodeByKey(_IN tcnode_key _key)**

Return the child node given a KEY. It is not recursive, only child nodes level is searched. The recursive version isgetFirstSubNodeByKey.
Parameters
- _key [in, required]: key to execute the search

Return Value
- A pointer to subnode if found. NULL if not found. If you want, you can use TNODE_PTR_TO_REF macro to convert pointer into reference.

- **tcnode_ptr getNodeByShortcut(_IN std::string _parm)**

- Return a node given a shortcut name.

Parameters

- _parm [in, required]: the shortcut name

Return Value

A pointer to subnode if found. NULL if not found. If you want, you can use TNODE_PTR_TO_REF macro to convert pointer into reference.

➢ std::vector<tcnode_key> & getNodeFullAddress(_OUT std::vector<tcnode_key> &_parm)

Get full address of current node representing by an array of keys. You can use returned array to create shortcuts.

Parameters

- _parm [out, required]: receives array of keys.

Return Value

- A reference to _parm.

➢ tcnode_ref getParent(void)

Get a reference to parent node.

Parameters

- None

Return Value

- Reference to parent node. root returns a reference to itself.

➢ tcnode_ref getRoot(void)

Get a reference to root node.

Parameters

- None

Return Value

- Reference to root node

➢ tcnode_shortcuts & getShortcuts(void)

Get a map containing all list of defined shortcuts. The type tcnode_shorcuts is a typedef of:

Collapse | Copy Code

```
std::map<std::string, std::vector<tcnode_key> >
```

Shortcuts are not related to a single node but all tree. Thus, you can call this function from any part or level of tree.

Parameters

- None

Return Value

- A reference to tcnode_shortcuts

➢ tcnode_subnodes & getSubNodes(void)

Return all child nodes from current node. tcnode_subnodes is a map<key, tCNode>.

Parameters

- None

Return Value

- tcnode_subnodes that contain child nodes of current node

➢ BOOL hasSubNodes(void)

Return TRUE if node has subnodes (child nodes).
Parameters
- None

Return Value
- TRUE if node has subnodes or FALSE if it has not

- **bool isRoot(void)**

Return TRUE if node is root node.
Parameters
- None

Return Value
- TRUE if node is root node or FALSE if it is not

- **void refreshDataSorters(void)**

Run all data sorters defined in the current node.
Parameters
- None

Return Value
- None

- **bool removeSubNodeByKey(_IN tcnode_key _key)**

Find a child node that matches key and remove it. If application has data sorters defined, it must call refreshDataSorters to update internal references.
Parameters
- _key [in, required]: key to search specific node

Return Value
- TRUE if child node found and removed. FALSE otherwise

- **tcnode_ref removeSubNodes(void)**

Remove ALL child nodes of current node recursively. If application has data sorters defined, it must callrefreshDataSorters to update internal references.
Parameters
- None

Return Value
- None

- **std::vector<tcnode_ptr> & selectDataEqualsTo(_IN std::string _name, _OUT std::vector<tcnode_ptr> &_parm, _IN const tcnode_data _value)**

Select data sorter nodes that matches _value. You must pass an empty std::vector<tcnode_ptr> to be filled with result.
Parameters
- _name [in, required]: data sorter name to select nodes
- _parm [out, required]: array filled with results
- _value [in, required]: value to be searched

Return Value
A reference to _parm. An array of pointers to nodes. TNODE_PTR_TO_REF converts PTR to REF.

- **std::vector<tcnode_ptr> & selectDataEqualsTo(_IN std::string _name, _OUT**

std::vector<tcnode_ptr> &_parm, _IN const std::vector<tcnode_data> &_vals)

Select data sorter nodes that matches _vals array. You must pass an empty std::vector<tcnode_ptr> to be filled with result.
Parameters

- _name [in, required]: data sorter name to select nodes.
- _parm [out, required]: array filled with results.
- _vals [in, required]: array of values to be searched.

Return Value
A reference to _parm. An array of pointers to nodes. TNODE_PTR_TO_REF converts PTR to REF.

➢ tcnode_ref setData(_IN tcnode_data _data)

Change the DATA in current node.
Parameters

- _data [in, required]: data to replace current one

Return Value

- A reference to current node

➢ tcnode_ref setDataAndKey(_IN tcnode_data _data, _IN tcnode_key _key)

Change the DATA and KEY in current node.
Parameters

- _data [in, required]: data to replace in current one
- _key [in, required]: key to replace in current one

Return Value

- A reference to current

➢ tcnode_ref setKey(_IN tcnode_key _key)

Change the KEY in current node.
Parameters

- _key [in, required]: key to replace in current one

Return Value

- A reference to current

➢ tcnode_ref setShortcut(_IN std::string _label)

Set a string shortcut to current node.
Parameters

- _label [in, required]: a string to name the shortcut

Return Value

- A reference to current node

➢ BOOL subNodeExists(_IN tcnode_key _key)

Find a subnode starting search from current node. The search is recursive.
Parameters

- _key [in, required]: key to search

Return Value

- TRUE if found, otherwise FALSE

➢ template<class _RECV> void transverse(_IN _RECV _receiver)

Execute a callback function with prototype void function(DATA _data, KEY _key, long _deep). From current node, transverse will call the callback for every subnode passing DATA, KEY and DEEP (or level).

Parameters

- _receiver [in, required]: The callback function

Return Value

- None

➢ operators ==, != and =

== and != are used to compare single nodes. What makes a node equal or different from other is the internal id (getId). In the tCNode tree, each node has its own Id.

So, these two operators make sense when application keeps many references or pointers to a single node and needs to know if that pointer or reference means that node.

The copy assignment operator (=) copies everything: nodes, data sorters, shortcuts.