dhorn::tree

# Introduction

The dhorn::tree class provides an easy way for parsing, creating, and iterating trees in C++. The dhorn::tree class stands on its own with no other classes it relies on. This is in contrast to most other apis in other languages where separate classes are introduced to represent individual nodes, etc. Replacing such classes is the familiar C++ iterator (dhorn::tree::iterator). Iterators are required for all operations on a tree, and their use is detailed extensively below.

# Classes/Structures/Types/Enumerations/etc.

## dhorn::tree::iterator Class

Iterators are extremely important for all data structure in the STL, but they are even more so for the dhorn::tree class. This is because, like std::vector and related data structures, the order of elements follows no pre-defined rule and is important to the client (unlike std::set, etc.). One can think of an iterator as a pointer to an individual node or cell in the tree. Each operation on a dhorn::tree that requires an iterator (which is just about all of them) generally operate either on that node, or the child of that node. For example, calling the dhorn::tree::insert function will add a child to the node represented by the iterator. Similarly, calling dhorn::tree::remove will remove that node and all of its children from the tree. Specifics on what these functions do, and the error handling associated with them, is described later in this document. The “begin” iterator returned by dhorn::tree::begin represents the first logical node pointed to by the root of the tree. Like all data structures, if the dhorn::tree is empty, the begin iterator is “equal to” the “end” iterator return by dhorn::tree::end. The “end” iterator should logically be thought of as the root or “sentinel” node of the tree. You cannot assign a value to the sentinel node, nor can you remove the sentinel node. The available functions for the dhorn::tree::iterator class are described below: