Inhaltsverzeichnis

Searching a Tree of Objects with Linq, Revisited

A while back, I wrote about searching through a tree using linq to objects. That post was mostly snippets of code about delegates, lambda's, yield and how it applies to linq — more a technical exploration than an example. So I thought I'd follow it up with concrete extension methods to make virtually any tree searchable by Linq.

Linq, IEnumerable<T>, yield

All that is required to search a tree with Linq is creating a list of all nodes in the tree. Linq to Objects can operate on IEnumerable<T>. Really, **Linq to objects is a way of expressing operations we've been doing forever in loops with if/else blocks**. That means there isn't any search magic going on, it is a linear traversal of all elements in a set and examining each to determine whether it matches our search criteria.

To turn a tree into a list of node we need to walk and collect all children of every node. A simple task for a recursive list that carries along a list object to stuff every found node into. But there is a better way, using yield to return each item as it is encountered. Now we don't have to carry along a collection. **Iterators using yield implement a pattern in which a method can return more than once**. For this reason, a method using yield in C# must return an IEnumerable, so that the caller gets a handle to an object it can traverse the result of the multiple return values.

IEnumerable is basically an unbounded set. This is also the reason why unlike collections, it does not have a Count Property. It is entirely possible for an enumerator to return an infinite series of items.

Together IEnumerable<T> and yield are a perfect match for our problem, i.e. recursively walking a tree of nodes and return an unknown number of nodes.

Two types of Tree Traversal

Depth First

In depth-first traversal, the algorithm will dig continue to dig down a nodes children until it reaches a leaf node (a node without children), before considering the next child of the current parent node.

Breadth First

In breadth-first traversal, the algorithm will return all nodes at a particular depth first before considering the children at the next level. I.e. First return all the nodes from level 1, then all nodes from level 2, etc.

Tree to IEnumerable<T> Extension methods

```
    public static class TreeToEnumerableEx
    {
     public static IEnumerable<T> AsDepthFirstEnumerable<T>(this T head,
Func<T, IEnumerable<T>> childrenFunc)
     {
      yield return head;
      foreach (var node in childrenFunc(head))
      {
       foreach (var child in AsDepthFirstEnumerable(node, childrenFunc))
       {
        yield return child;
       }
      }
     }

     public static IEnumerable<T> AsBreadthFirstEnumerable<T>(this T head,
Func<T, IEnumerable<T>> childrenFunc)
     {
      yield return head;
      var last = head;
      foreach(var node in AsBreadthFirstEnumerable(head,childrenFunc))
      {
       foreach(var child in childrenFunc(node))
       {
        yield return child;
        last = child;
       }
       if(last.Equals(node)) yield break;
      }
     }
    }
```