NAME

`HTML_Node` − Abstract base class for nodes in an HTML tree

SYNOPSIS

```
class Parent_Node;

class HTML_Node {
public:
      class visitor {
      public:
            virtual ~visitor();
            virtual bool operator()(
                  HTML_Node*, int depth, bool is_end_tag
            ) const = 0;
      };

      virtual ~HTML_Node();

      Parent_Node*      parent() const;
      void              parent( Parent_Node *new_parent );
      virtual void      visit( visitor const&, int depth = 0 );
protected:
      HTML_Node( Parent_Node *parent = 0 );
};

Parent_Node*      parse_html_file( file_vector const& );
```

DESCRIPTION

`HTML_Node` is an abstract base class for nodes in an HTML tree that was built by parsing an HTML file into a tree structure like the HTML DOM (Document Object Model). Once built, the nodes of the tree (elements and text from the HTML file) can be traversed by a user-defined *visitor* class.

Public Interface

`Parent_Node* parent() const`

Returns a pointer to the current parent node for this node, or null if this node has no parent.

`void parent( Parent_Node *new_parent )`

If this node already has a parent that is not the current parent, this node is first removed from that parent's list of child nodes. Then, this node's parent node is set to `new_value`. If `new_parent` is not null, adds this node to the parent's list of child nodes.

`virtual void visit( visitor const&, int depth = 0 )`

Performs an in-order tree traversal starting at this node. For each node, the visitor's `operator()` is called once.

Protected Interface

`HTML_Node( Parent_Node *parent = 0 )`

Default constructor. If `parent` is not null, sets the parent and adds this node to that parent's list of child nodes.

Global Functions

`Parent_Node* parse_html_file( file_vector const& )`

Parses the given HTML file into an HTML tree and returns a pointer to the root node of an HTML tree.

The Visitor Class

`HTML_Node::visitor` is an abstract base class for object that ''visit'' nodes.

Public Interface

`virtual ~visitor()`

Destructor. It does nothing. It's defined only to ensure it's virtual as it should be for an abstract base class.

`virtual bool operator()( HTML_Node*, int depth, bool is_end_tag )`

The visit function. A derived class **must** override this since it's pure virtual. The `depth` indicates how "deep" the current node is in the tree. Depths start at zero. The `is_end_tag` argument is not used by `HTML_Node`, so it always passes `false`.

EXAMPLE

The following example "pretty prints" and HTML file:

```
class pretty_printer : public HTML_Node::visitor {
public:
      bool operator()( HTML_Node*, int depth, bool is_end_tag ) const;
};

bool pretty_printer::operator()(
      HTML_Node *node, int depth, bool is_end_tag
) const {
      while ( depth-- > 0 )
            cout << "    ";

      if ( Text_Node *const t = dynamic_cast< Text_Node* >( node ) ) {
            cout << t->text << endl;
            return true;
      }

      Element_Node *const e = dynamic_cast< Element_Node* >( node );
      if ( is_end_tag ) {
            cout << "</" << e->name() << ">\n";
            return false;
      }

      cout << '<' << e->name();

      for ( Element_Node::attribute_map::const_iterator
            att  = e->attributes.begin();
            att != e->attributes.end(); ++att
      )
            cout << ' ' << att->first << "=\"" << att->second << '"';
      cout << ">\n";
      return true;
}
```

SEE ALSO

Element_Node(3), **file_vector**(3), **Parent_Node**(3), **Text_Node**(3).

World Wide Web Consortium Document Object Model Working Group. *Document Object Model*, December 1998.

http://www.w3.org/DOM/

AUTHOR

Paul J. Lucas <*pjl@best.com*>

HISTORY

The HTML parser is derived from code in SWISH++, a really fast file indexing and searching engine (also by the author).

http://www.best.com/~pjl/software/swish/