NAME

`Parent_Node` − Node in an HTML tree for HTML elements that contains child nodes

SYNOPSIS

```
class Parent_Node : public virtual HTML_Node {
public:
      typedef std::list< HTML_Node* > child_list;

      Parent_Node( Parent_Node *parent = 0 );
      virtual ~Parent_Node();

      void              add_child( HTML_Node *child );
      child_list &      children();
      child_list const& children() const;
      bool              empty() const;
      bool              remove_child( HTML_Node *child );

      // overridden
      virtual void      visit( visitor const&, int depth = 0 );

      // inherited
      Parent_Node*      parent() const;
      void              parent( Parent_Node *new_parent );
};
```

DESCRIPTION

`Parent_Node` is-an `HTML_Node` that contains child nodes, i.e., content. For example, the `SELECT` element below is a parent of the newline `Text_Node` after the `SELECT` and all of the `OPTION` elements:

```
<SELECT NAME="menu">          parent child
      <OPTION>Blueberry       child grandchild
      <OPTION>Chocolate       child grandchild
      <OPTION>Raspberry       child grandchild
</SELECT>
```

whereas an element such as `IMG` can have no child nodes.

Public Interface

Constructors

These are the same as those for `Element_Node`.

Destructor

In addition to destroying itself, the destructor also destroys all of its child nodes, if any.

`void add_child( HTML_Node *child )`

If `child` is null or it equals `this`, does nothing. Otherwise, if the child node already has a parent, it is removed from that parent's list of child nodes first; then it is added to this node's list of child nodes and sets the child's parent to `this`.

`child_list& children()`

`child_list const& children() const`

Returns a reference to this node's list of child nodes, or a reference to an empty list if there are none.

`bool empty() const`

Returns true only if this node has no child nodes.

`bool remove_child( HTML_Node *child )`

If `child` is null, does nothing and returns `false`. Otherwise, searches through this node's list of child nodes looking for the child. If found, removes the child, set the child's parent to null, and returns `true`; otherwise, returns `false`.

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

This member function overrides `HTML_Node`'s `visit()`. It simply calls `visit()` for each child node in order passing `depth`. In pseudo-code:

```
for ( every child )
      child->visit( v, depth );
```

It does *not* call `visit()` for itself. It is therefore effectively ''invisible'' in the HTML node tree.

SEE ALSO

Element_Node(3), **HTML_Node**(3).

AUTHOR

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