Tutorial 14

Working with Document Nodes and Styles

HTML, CSS, and Dynamic HTML

5TH EDITION

Carey, P. (2013). New perspectives on HTML, CSS, and Dynamic HTML: Comprehensive (5th ed.). Australia ; United Kingdom: Course Technology Cengage Learning.

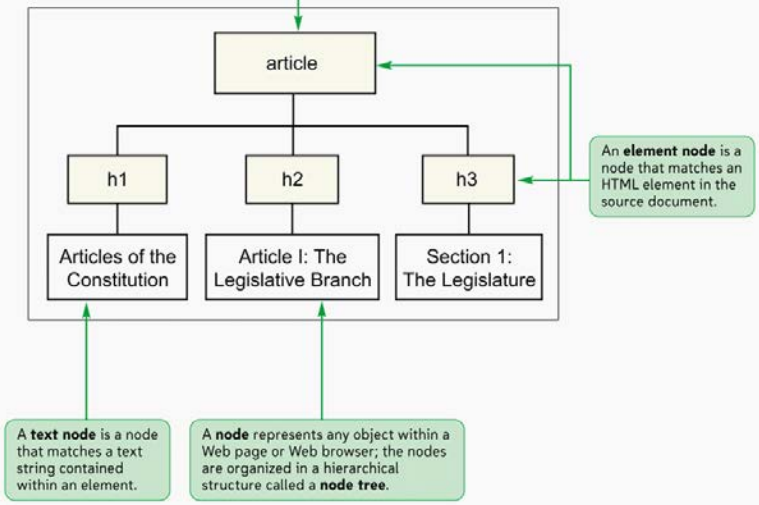
Objectives

- Understand the methods and properties of nodes and the node tree
- Learn to create element and text nodes
- Attach nodes to a Web page document
- Apply node properties and styles to create dynamic content
- Work with the properties and methods of attribute nodes

Objectives

- Work with element attributes
- Hide and redisplay Web page objects
- Understand how to create recursive functions to navigate a node tree
- Learn to work with the properties and methods of style sheet objects

Exploring the Node Tree



The `firstChild` property returns the first child of the specified node.

The `nextSibling` property returns the next sibling node of the specified node.

```
function createList(source, TOCList, headings) {

   /* Loop through all of the child nodes of the source document,
      sibling by sibling until no child nodes are left */

   for (var n = source.firstChild; n != null; n = n.nextSibling) {
      /* Determine the heading level (if any) of the current node */
      var nodeLevel = headings.indexOf(n.nodeName);

      if (nodeLevel != -1) {
         var listItem = document.createElement("li");
         listItem.innerHTML = n.innerHTML;
         TOCList.appendChild(listItem);
      }
   }
}
```

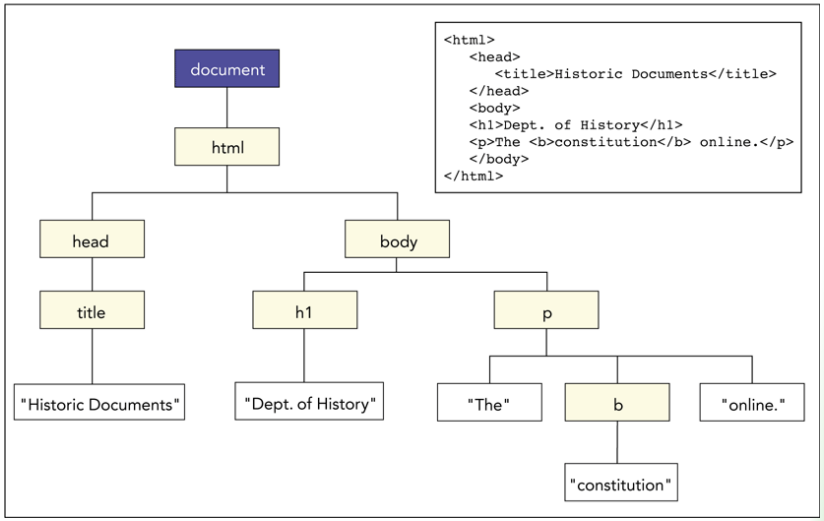
The `appendChild()` method appends the specified node to a parent node.

The `createElement()` method creates an element node with the specified tag name.

Working with Nodes

- The Node Tree
  - Nodes are arranged into a hierarchal structure called a **node tree**, which indicates the relationship between each of the nodes

Figure 14-7 A sample node tree



Working with Nodes

- The Node Tree
  - The parent of all nodes within a document is the document note, which is also known as the **root node**

Figure 14-8 Node relationships

Expression	Description
node.firstChild	The first child of *node*
node.lastChild	The last child of *node*
node.childNodes	A collection of all of the nodes that are direct children of *node*
node.previousSibling	The sibling prior to *node*
node.nextSibling	The sibling after *node*
node.ownerDocument	The root node of the document
node.parentNode	The parent of *node*

Specifying Node Relationships

- To access the parent of a node, use the reference

  node`.parentNode`

  where *node* is a node object in the node tree.

- To reference the first child and last child of a node, use the following references:

  node`.firstChild`

  node`.lastChild`

Specifying Node Relationships

- To reference the collection of all child nodes, use the following reference:

  node`.childNodes`

- To reference the previous and next siblings, use the following references:

  node`.previousSibling`

  node`.nextSibling`

Creating Nodes

- To create an element node, use the method

  `document.createElement(`*`text`*`);`

  where *text* is the name of the element.
- To create an attribute node, use the method

  `document.createAttribute(`*`text`*`);`

  where *text* is the name of the attribute.
- To create a text node, use the method

  `document.createTextNode(`*`text`*`);`

  where *text* is the text string of the text node.

Creating Nodes

- To create a comment node, use the method

  `document.createComment(`*`text`*`);`

  where *text* is the text of the comment.

- To copy a preexisting node, use the method

  `node``.cloneNode(`*`deep`*`)`

  where *node* is the preexisting node, and *deep* is a Boolean value indicating whether to copy all descendants of the node (true) or only the node itself (false).

Creating and Attaching Nodes

Figure 14-12 Methods to add and remove nodes

Method	Description
node.appendChild(*new*)	Appends the node *new* as a new child of *node*
node.insertBefore(*new*, *child*)	Inserts the child node *new* as a new child of *node*, placing it before the specified *child* node; if *child* is omitted, the *new* node is inserted as the last child node
node.normalize()	Traverses all child nodes of *node*; any adjacent text nodes are merged into a single text node
node.removeChild(*old*)	Removes the child node *old* from *node*
node.replaceChild(*new*, *old*)	Replaces the child node *old* with the child node ***new***

Attaching and Removing Nodes

- To append a new node as a child of a preexisting node, use the method

  node`.appendChild(`*new*`)`

  where *node* is the preexisting node and *new* is the new child. The new child node is appended to the end of the child nodes collection. If *new* already exists as a node in the document fragment, it is moved from its current location to the new location

Attaching and Removing Nodes

- To insert a new node at a specific location in the child nodes collection, use

  node`.insertBefore(`*new*`, `*child*`)`

  where *child* is the child node in front of which the new node should be placed. If *new* already exists as a node in the document fragment, it is moved from its current location to the new location.

Attaching and Removing Nodes

- To remove a child node, use the method

  node`.removeChild(`*old*`)`

  where *old* is the child node to be removed.

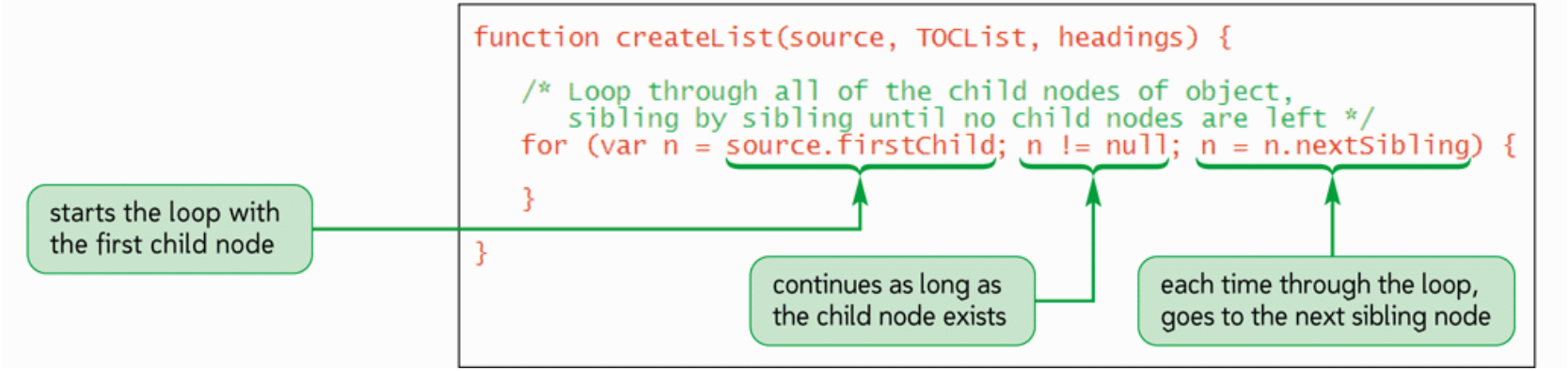
- To replace one child node with another, use the following method:

  node`.replaceChild(`*new*`, `*old*`)`

Working with Node Types, Names, and Values

- Looping Through a Child Node Collection

Figure 14-15 Looping through the child nodes collection in the source document



Working with Node Types, Names, and Values

Figure 14-17 Node types, names, and values

Node	.nodeType	.nodeName	.nodeValue
Element	1	*ELEMENT NAME*	null
Attribute	2	*attribute name*	*attribute value*
Text	3	#text	*text string*
Comment	8	#comment	*comment text*
Document	9	#document	null

Working with Node Types, Names, and Values

Figure 14-18 Node properties from the sample node tree

Node	.nodeType	.nodeName	.nodeValue
Document	9	#document	null
html	1	HTML	null
head	1	HEAD	null
body	1	BODY	null
title	1	TITLE	null
"Historic Documents"	3	#text	Historic Documents
h1	1	H1	null
"Dept. of History"	3	#text	Dept. of History
p	1	P	null
"The "	3	#text	The
b	1	B	null
"constitution"	3	#text	constitution
" online"	3	#text	online

Determining Node Properties

- To determine the type of object a node represents, use the property

  node.nodeType

  where *node* is a node object in the node tree. The nodeType property returns the value 1 for elements, 2 for attributes, and 3 for text nodes.

Determining Node Properties

- To return the value of a node, use the following property:

  node.nodeValue

- For an element, the value of the nodeValue property is null. For an attribute, the value represents the attribute's value. For a text node, the value represents the text string contained in the node.

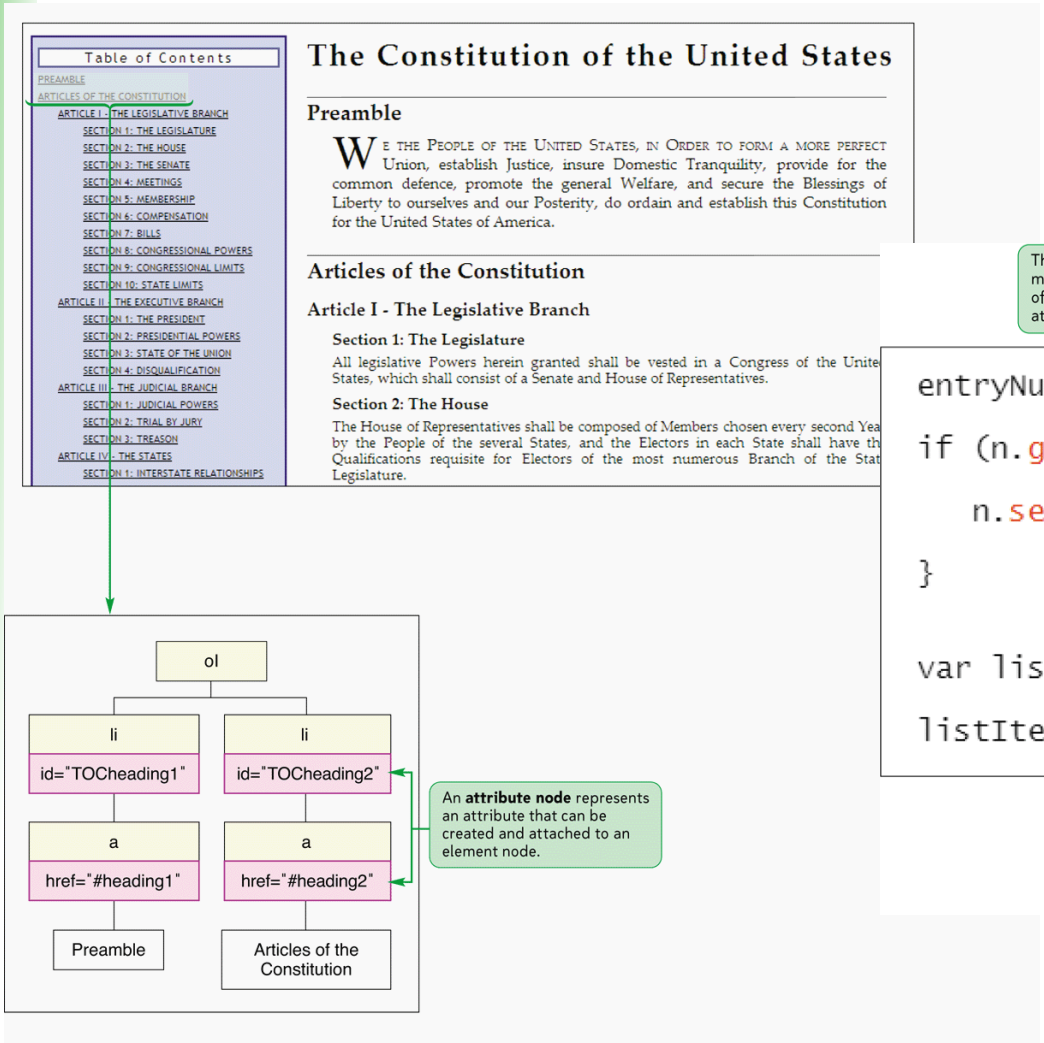
Determining Node Properties

- To return the name of a node, use the following property:

  node.nodeName

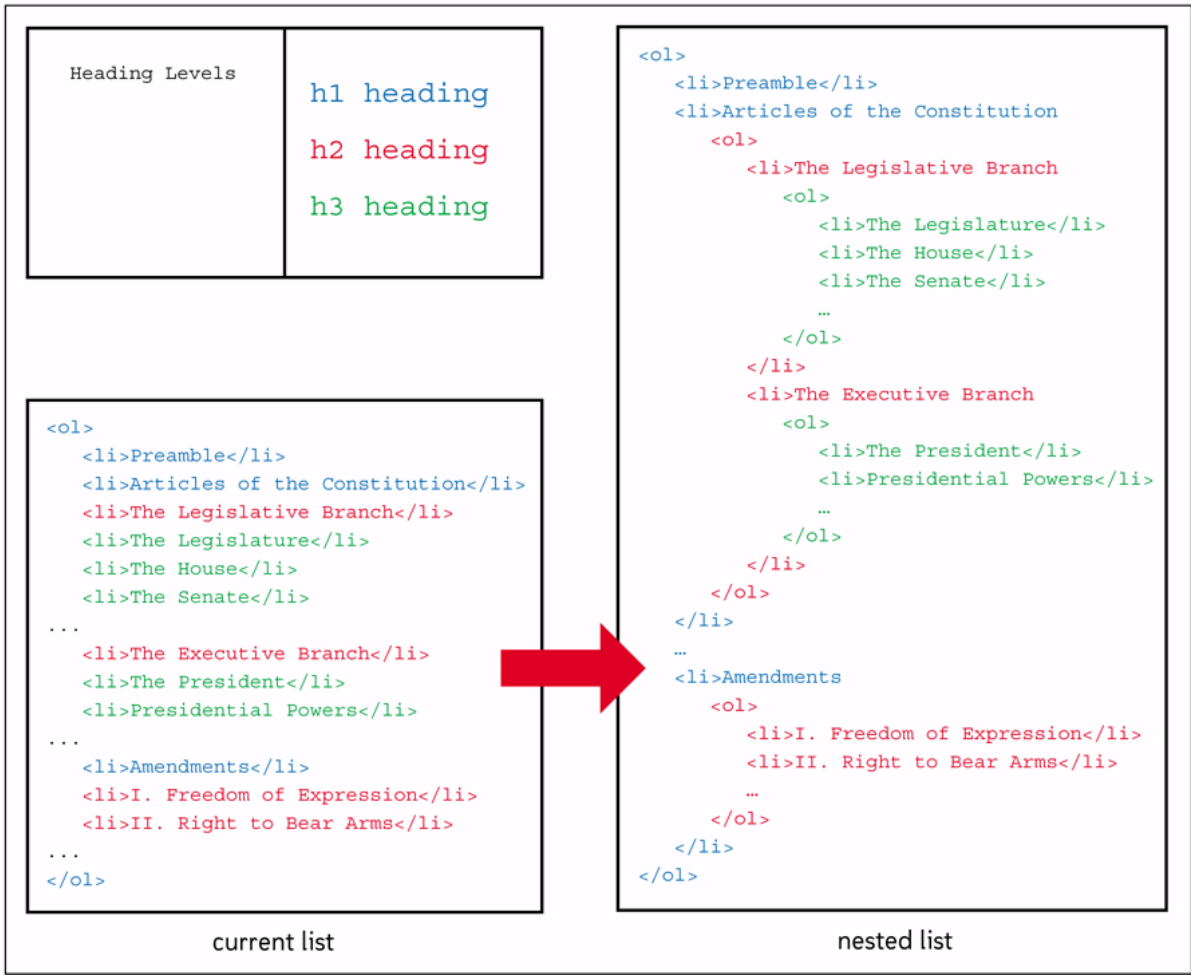
- For elements, the name of the node matches the name of the element in uppercase letters. For attributes, the node name matches the attribute name. For text nodes, the node name is #text.

Exploring Attribute Nodes



Creating a Nested List

Figure 14-22 Creating a nested list



Creating a Nested List

Figure 14-24 Inserting the if structure for creating nested lists

replaces the statement that appends the list item with the if structure

updates the value of the prevLevel variable

```
/* If the node comes from a heading element, create a list item
   and append it to the TOC */
if (nodeLevel != -1) {
   var listItem = document.createElement("li");
   listItem.innerHTML = n.innerHTML;

   if (nodeLevel == prevLevel) {
      /* Append the list item to the current list */
   }

   else if (nodeLevel > prevLevel) {
      /* Start a new nested list */
   }

   else {
      /* Append the list item to a higher nested list */
   }

   /* Update the previous node level value to the current level */
   prevLevel = nodeLevel;

}
```

Working with Attributes

Figure 14-33 Linking list items to page headings

heading elements with id values

```
<h1 id="head1">Preamble</h1>
…
<h1 id="head2">Articles of the Constitution</h1>
…
<h2 id="head3">Legislative Branch</h2>
…
<h3 id="head4">Section 1: The Legislature</h3>
…
<h3 id="head5">Section 2: The House</h3>
…
<h3 id="head6">Section 3: The Senate</h3>
…
```

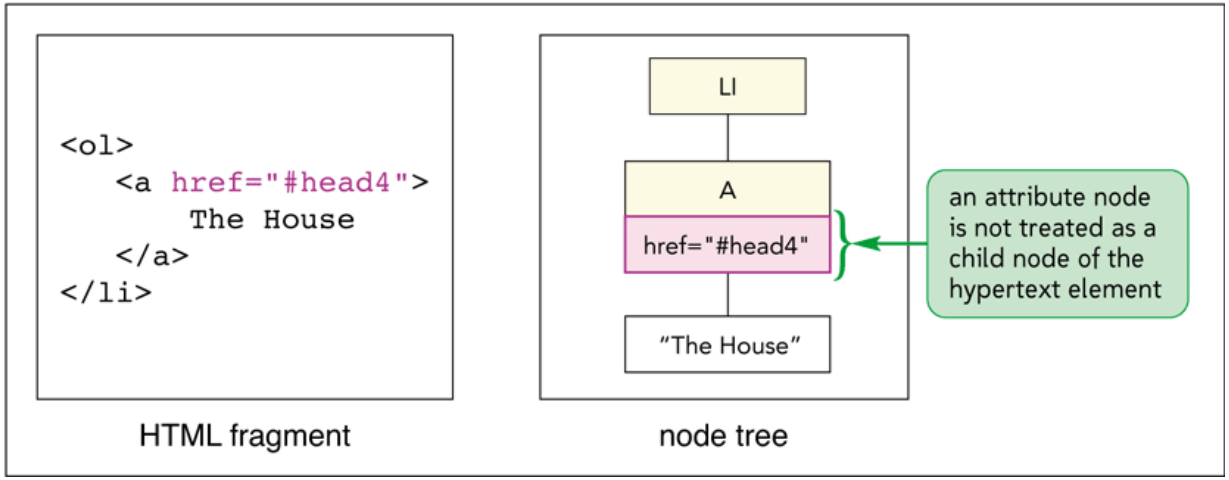
table of contents with links to each heading tag

```
<ol>
   <li><a href="#head1">Preamble</a></li>
   <li><a href="#head2">Articles of the Constitution</a></li>
   <ol>
      <li><a href="#head3">Legislative Branch</a></li>
      <ol>
         <li><a href="#head4">Section 1: The Legislature</a></li>
         <li><a href="#head5">Section 2: The House</a></li>
         <li><a href="#head6">Section 3: The Senate</a></li>
...
```

Working with Attributes

- Attribute Nodes

Figure 14-34 Attribute nodes



Working with Attributes

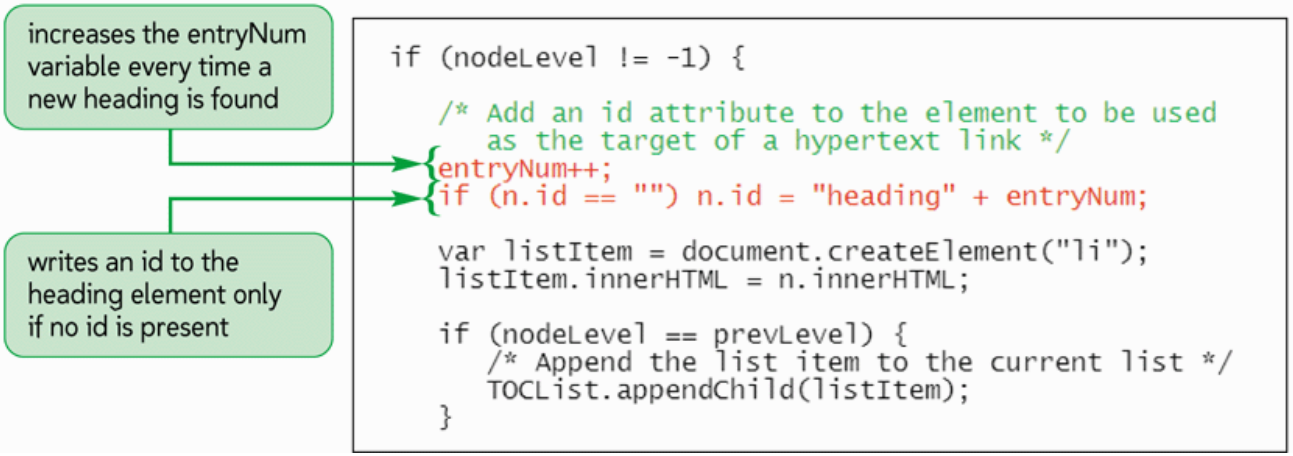
Figure 14-35 Methods for working with attribute nodes

Method	Description
node.attributes	Returns the collection of attributes associated with *node*
node.attributes[*i*].nodeName	Returns the attribute name from an item in the attributes collection where *i* is the index number
node.attributes[*i*].nodeValue	Returns the attribute value from an item in the attributes collection
document.createAttribute(*att*)	Creates an attribute node with the name *att*
node.getAttribute(*att*)	Returns the value of the attribute *att* from the *node* to which it has been attached
node.hasAttribute(*att*)	Returns a Boolean value indicating whether *node* has the attribute *att*
node.removeAttribute(*att*)	Removes the attribute *att* from the *node*
node.removeAttributeNode(*att*)	Removes attribute *att* from the *node*
node.setAttribute(*att*, *value*)	Creates or changes the value of the attribute *att* of the *node*

Working with Attributes

- Setting the Heading Element ids

Figure 14-37 Writing the heading id value



Working with Attributes

- Inserting Links

Figure 14-39 Creating list items as hypertext links

adds an id to every list item in the TOC

creates a hypertext link to the heading element in the source document

appends the hypertext link to the list item

```
if (nodeLevel != -1) {

   /* Add an id attribute to the element to be used
      as the target of a hypertext link */
   entryNum++;
   if (n.id == "") n.id = "heading" + entryNum;

   var listItem = document.createElement("li");
   listItem.id = "TOC" + n.id;

   /* Create a link to the heading node in the source document */
   var linkItem = document.createElement("a");
   linkItem.innerHTML = n.innerHTML;
   linkItem.href = "#" + n.id;

   /* Append the hypertext link to the TOC list item */
   listItem.appendChild(linkItem);

   if (nodeLevel == prevLevel) {
      /* Append the list item to the current list */
      TOCList.appendChild(listItem);
   }
```

Switching Style Sheets

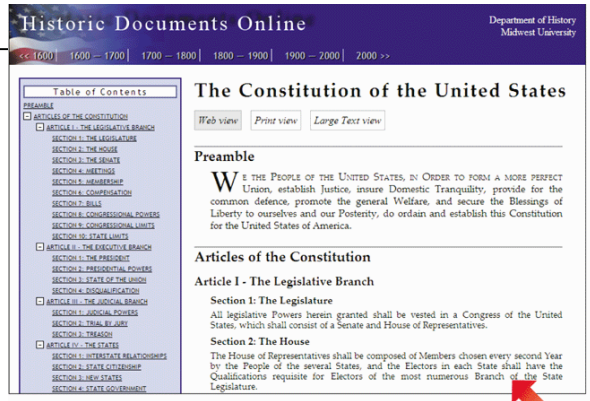
A **persistent style sheet** is always active and has no `title` attribute.

A **preferred style sheet** has a `rel` attribute value of `"stylesheet"` and a title; it is active by default.

An **alternate style sheet** has a `rel` attribute value of `"alternate stylesheet"` and a title; but is not active by default.

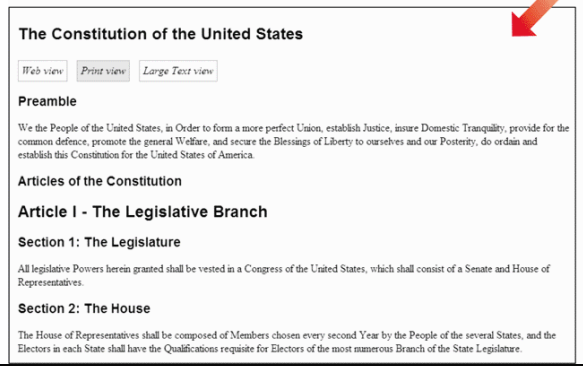
```
<link type="text/css" href="base.css"
      rel="stylesheet" />

<link type="text/css" href="hdo.css"
      rel="stylesheet" title="Web"  />

<link type="text/css" href="print.css"
      rel="alternate stylesheet" title="Print" disabled />
```

The `disabled` attribute disables a style sheet in the Web browser.



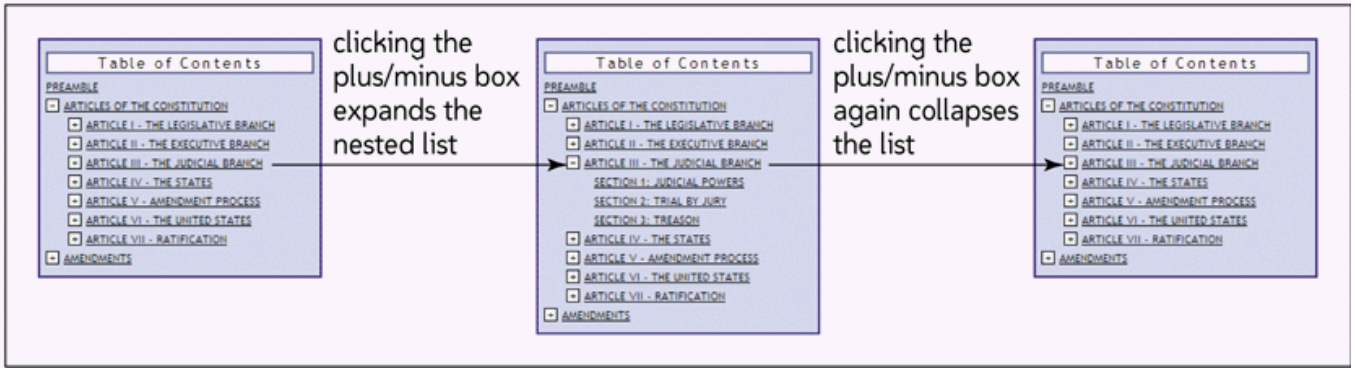

```
var links = document.getElementsByTagName("link") ;
for (var i = 0; i < links.length; i++) {

   if ( (links[i].rel == "stylesheet" ||
         links[i].rel == "alternate stylesheet") &&
         links[i].hasAttribute("title") ) {

      if (links[i].title != sheet) links[i].disabled = true
      else links[i].disabled = false;

   }

}
```

To switch between style sheets, you loop through all of the preferred and alternate style sheets, disabling the sheets you don't want to use, and enabling those sheets you do want to use.



Expanding and Collapsing a Document

Figure 14-41 Using a plus/minus box to expand and collapse the list



Expanding and Collapsing a Document

- Creating a plus/minus Box

Figure 14-43 Placement of the plus/minus box

```
<li id="TOChead34">
   <span>-</span>          plus/minus box
   <a href="#head34">
      Amendments
   </a>
   <ol>
      <li id="TOChead35">
         <a href="#head35">
            I. Freedom of Expression
         </a>
      </li>
      <li id="TOChead36">
         <a href="#head36">
            II. Right to Bear Arms
         </a>
      </li>
      …
   </ol>
…
</li>
```

HTML fragment

plus/minus box

- AMENDMENTS
  - I. FREEDOM OF EXPRESSION
  - II. RIGHT TO BEAR ARMS
  - III. QUARTERING OF SOLDIERS
  - IV. SEARCH AND SEIZURE
  - V. TRIAL AND PUNISHMENT
  - VI. SPEEDY TRIALS
  - VII. TRIAL BY JURY
  - VIII. PUNISHMENT
  - IX. ENUMERATION OF RIGHTS
  - X. POWERS OF THE STATE AND PEOPLE

Rendered elements

Expanding and Collapsing a Document

- Creating a plus/minus Box

Figure 14-47 Applying an event handler to the plus/minus box

```
/* Create a plus-minus box */
var plusMinusBox = document.createElement("span");
plusMinusBox.innerHTML = "-";

/* Insert the plus-minus box directly before
   the last item in the current list  */
TOCList.lastChild.insertBefore(plusMinusBox, TOCList.lastChild.lastChild);

/* Expand and collapse the TOC and the document when
   the plus-minus box is clicked */
plusMinusBox.onclick = expandCollapse;
```

runs the expandCollapse() function when the box is clicked

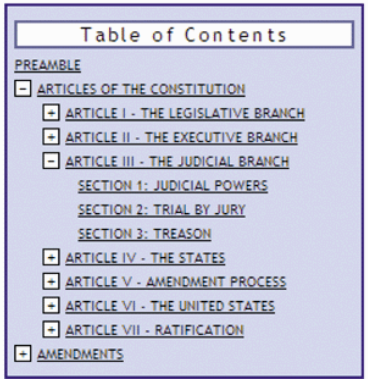
Expanding and Collapsing a Document

- Hiding and Display Objects

Figure 14-50 Hiding and displaying the nested list

```
function expandCollapse() {
   /* Reference of the nested list that should be hidden
      or displayed */
   var nestedList = this.nextSibling.nextSibling;

   if (this.innerHTML == "-") {
      /* Collapse the nested list, hiding the contents
         and changing to character to + */
      this.innerHTML = "+";
      nestedList.style.display = "none";
   } else {
      /* Expand the nested list, displaying the contents
         and changing the character to - */
      this.innerHTML = "-";
      nestedList.style.display = "";
   }

}
```

Figure 14-51 Collapsing the table of contents



Expanding and Collapsing a Document

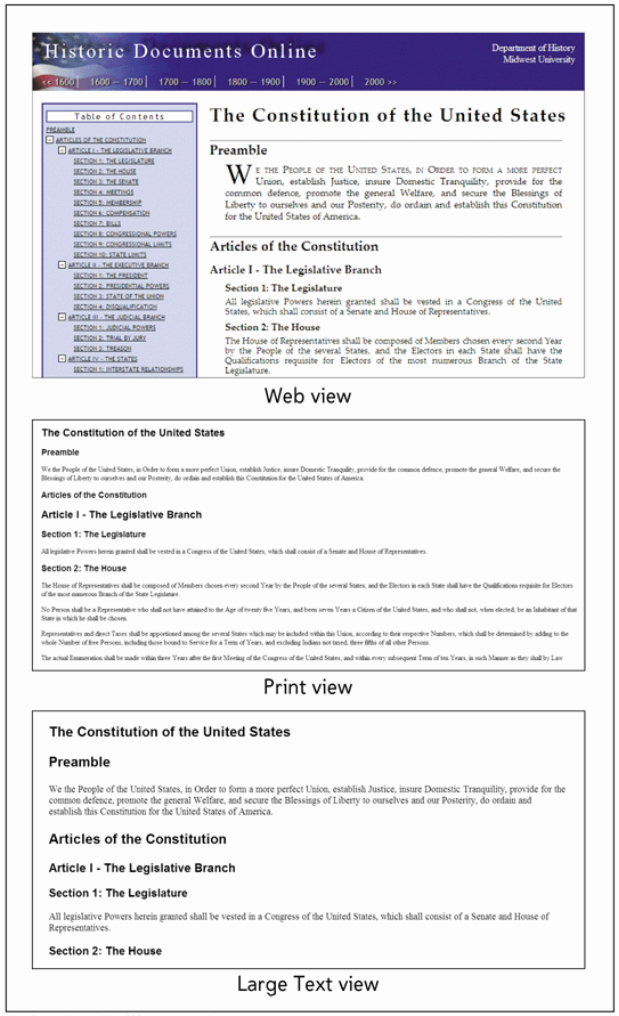
- Expanding and Collapsing the Source Document

Figure 14-53 Inserting the expandCollapseDoc() function

```
function expandCollapseDoc() {

   var displayStatus = "";
   var source = document.getElementById("doc");
   headings = ["H1", "H2", "H3", "H4", "H5", "H6"];

   /* Loop through every page element in the source document */
   for (var n = source.firstChild; n != null; n = n.nextSibling) {

      if (headings.indexOf(n.nodeName) != -1) {
         /* Determine the display status of the matching
            TOC list item */
      }

      if (n.nodeType == 1) {
         /* Set the display status only if the node represents
            a page element */
      }

   }
}
```

Switching Between Style Sheets

Figure 14-58 Three views of the Constitution document



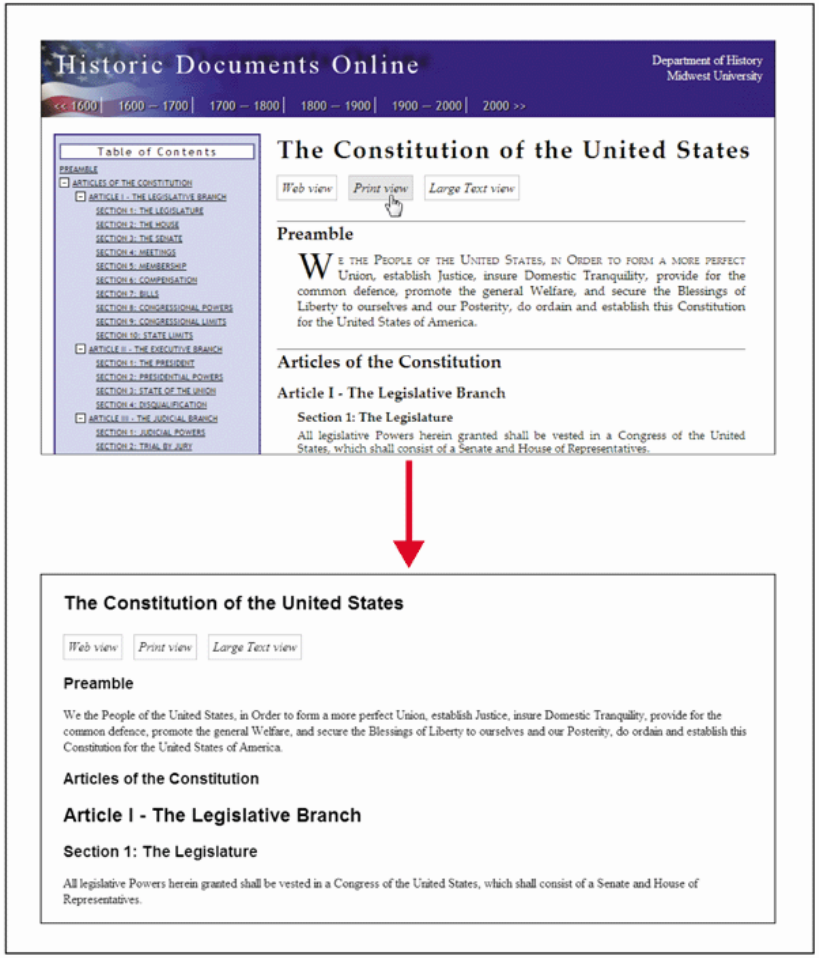
Hudyma Natallia/Shutterstock.com

Switching Between Style Sheets

- Style sheets can be classified as persistent, preferred, and alternate:
  - Persistent style sheets are always active
  - Preferred style sheets are turned on by default, but can be turned off by actions of the user
  - Alternate style sheets are not turned on by default, but can be turned on as an alternate to the preferred style sheet

Switching Between Style Sheets

Figure 14-59 Switching between style sheets



Hudyma Natallia/Shutterstock.com

Switching Between Style Sheets

Figure 14-61 HTML code for the style switcher buttons

```
<article id="doc">

   <div id="styleOptions">
      <button onclick="changeStyle('Web')">Web view</button>
      <button onclick="changeStyle('Print')">Print view</button>
      <button onclick="changeStyle('Large Text')">Large Text view</button>
   </div>
```

Switching Between Style Sheets

- Properties of the styleSheet object

Figure 14-64 Properties of the styleSheet object

Property	Description
styleSheet.cssText	The text of the style declarations in *styleSheet* (Internet Explorer only)
styleSheet.disabled	A Boolean value indicating whether *styleSheet* is disabled (true) or enabled (false)
styleSheet.href	The URL of *styleSheet*; for embedded style sheets, the value of href is an empty text string [read-only]
styleSheet.media	A text string containing the list of media types associated with *styleSheet* [read-only]
styleSheet.rules	Returns the collection of rules within *styleSheet* (Internet Explorer only)
styleSheet.cssRules	Returns the collection of rules within *styleSheet*
styleSheet.title	The title of *styleSheet* [read-only]
styleSheet.type	The MIME type of *styleSheet* [read-only]