

Object Referencing

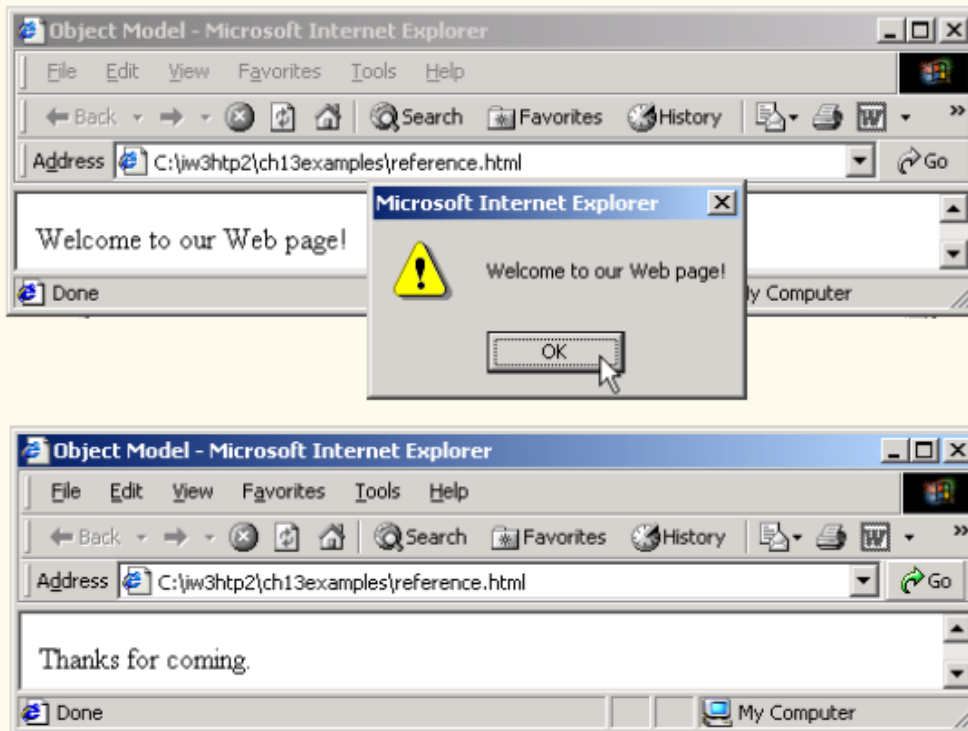
The simplest way to reference an element is by using the element's **id** attribute. The element is represented as an object, and its various XHTML attributes become properties that can be manipulated by scripting. Figure 1 uses this method to read the **innerHTML** *property* of a **p** element.

uses the **onload** *event* to call the JavaScript **start** function when document

loading completes. (Events are covered in depth in the next chapter.)

Function **start** dis-

```
1 <?xml version = "1.0"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Strict//EN"
3 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
45
<!-- Fig. 13.1: reference.html -->
6 <!-- Object Model Introduction -->
78
<html xmlns = "http://www.w3.org/1999/xhtml">
9 <head>
10 <title>Object Model</title>
11
12 <script type = "text/javascript">
13 <!--
14 function start()
15 {
16 alert( pText.innerHTML );
17 pText.innerHTML = "Thanks for coming.";
18 }
19 // -->
20 </script>
21
22 </head>
23
24 <body onload = "start()">
25 <p id = "pText">Welcome to our Web page!</p>
26 </body>
27 </html>
```



Line 24 uses the *onload event* to call the JavaScript **start** function when document loading completes. (Events are covered in depth in the next chapter.) Function **start** displays an **alert** box containing the value of **pText.innerHTML**. The object **pText** refers to the **p** element whose **id** is set to **pText** (line 25). The **innerHTML** property of the object refers to the text contained in that element (**Welcome to our Web page!**). Line 17 of function **start** sets the **innerHTML** property of **pText** to a different value. Changing the text displayed on screen in this manner is an example of a Dynamic HTML capability called *dynamic content*.

Collections all and children

Included in the Dynamic HTML Object Model is the notion of *collections*, which basically are arrays of related objects on a page. There are several special collections in the object model (several collections are listed in Fig. 13.10 and Fig. 13.11 at the end of this chapter). The Dynamic HTML Object Model includes a special collection, **all**. The *all collection* is a collection of all the XHTML elements in a document, in the order in which they appear. This provides an easy way of referring to any specific element, especially if it does not have an **id**. The script in Fig. 13.2 iterates through the **all** collection and displays the list of XHTML elements on the page by writing to the **innerHTML** property of a **p** element.

```
<?xml version = "1.0"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Strict//EN"
3 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
45
<!-- Fig 13.2: all.html -->
6 <!-- Using the all collection -->
78
<html xmlns = "http://www.w3.org/1999/xhtml">
9 <head>
10 <title>Object Model</title>
11
12 <script type = "text/javascript">
13 <!--
14 var elements = "";
15
16 function start()
17 {
18 for ( var loop = 0; loop < document.all.length; ++loop )
19 elements += "<br />" + document.all[ loop ].tagName;
20
21 pText.innerHTML += elements;
22 alert( elements );
23 }
24 // -->
25 </script>
26 </head>
27
28 <body onload = "start()">
29 <p id = "pText">Elements on this Web page:</p>
30 </body>
31 </html>
```


children of that element. If it encounters an element that has its own children (line 24), it recursively calls the **child** function, passing the object of the new element through which the function should loop. As that loop finishes, the loop which called it proceeds to the next element in its own array of **children**. We use the **tagName** property to gather [Fig.2 Looping through the all collection \(part 2 of 2\)](#).

the names of the tags we encounter while looping through the document, and we place them in the string **elements**. The script adds **ul** and **li** tags to display the element in a hierarchical manner on the page. When the original call to function **child** completes, line 39 changes the **outerHTML** property of the **p** element **myDisplay** to string **elements**.

Property **outerHTML** is similar to property **innerHTML** we introduced in the previous example, but it includes the enclosing XHTML tags (tags `<p id = "myDisplay">` and `</p>` in this case) as well as the content inside them.

```
1 <?xml version = "1.0"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Strict//EN"
3 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
45
<!-- Fig 13.3: children.html -->
6 <!-- The children collection -->
78
<html xmlns = "http://www.w3.org/1999/xhtml">
9 <head>
10 <title>Object Model</title>
11
12 <script type = "text/javascript">
13 <!--
14 var elements = "<ul>";
15
16 function child( object )
17 {
18 var loop = 0;
19
20 elements += "<li>" + object.tagName + "<ul>";
21
22 for ( loop = 0; loop < object.children.length; loop++ )
23 {
24 if ( object.children[ loop ].children.length )
25 child( object.children[ loop ] );
26 else
27 elements += "<li>" +
```

```

28 object.children[ loop ].tagName +
29 "</li>";
30 }
31
32 elements += " </ul> ";
33 }
34 // -->
35 </script>
36 </head>
37
38 <body onload = "child( document.all[ 4 ] );
39 myDisplay.outerHTML += elements;">
40
41 <p>Welcome to our <strong>Web</strong> page!</p>
42
43 <p id = "myDisplay">
44 Elements on this Web page:
45 </p>
46
47 </body>
48 </html>

```

