Chapter 49: Application: Tables and Calendars

In This Chapter

Accommodating older browsers

Scripted tables

Date calculations

Working with HTML tables is a lot of fun, especially if, like me, you are not a born graphics designer. By adding a few tags to your page, you can make your data look more organized, professional, and appealing. Having this power under scripting control is even more exciting, because it means that in response to a user action or other variable information (such as the current date or time), a script can do things to the table as the table is being built. In IE4+ and W3C DOMs, scripts can modify the content and structure of a table even after the page has loaded, allowing the page to almost "dance."

You have three options when designing scripted tables for your pages, although only two are backward compatible with non-DHTML browsers:

* Static tables
* Dynamic tables
* Dynamic HTML tables

The design path you choose is determined by whether you need to dynamically update some or all fields of a table (data inside `<TD>...</TD>` tags) and which browser levels you need to support. To highlight the differences among the three styles, this chapter traces the implementation of a monthly calendar display in all three formats.

About the Calendars

Because the emphasis here is on the way tables are scripted and displayed, I quickly pass over structural issues of the calendar versions described in the following sections. The first two examples are backward compatible to the earliest browsers that didn't even know genuine `Array` objects. The final example, however, is a much more modern affair, utilizing table-related DOM objects and methods to simplify the code. It requires IE4+ for Windows (unfortunately, a bug in IE/Mac causes problems with the amount of TABLE object modification the script does) and in NN6.

All three calendars follow similar (if not over-simplified) rules for displaying calendar data. English names of the months are coded into the script, so that they can be plugged

into the calendar heading as needed. To make some of the other calendar calculations work (such as figuring out which day of the week is the first day of a given month in a given year), I define a method for my month objects. The method returns the JavaScript date object value for the day of the week of a month's first date. Virtually everything I do to implement the month objects is adapted from the custom objects discussion of Chapter 34.

Static Tables

The issue of updating the contents of a table's fields is tied to the nature of an HTML document being loaded and fixed in the browser's memory. Recall that for early browsers, you can modify precious few elements of a document and its objects after the document has loaded. That case certainly applies for typical data points inside a table's <TD> tag pair. After a document loads — even if JavaScript has written part of the page — none of its content (except for text and textarea field contents and a few limited form element properties) can be modified without a complete reload.

Listing 49-1 contains the static version of a monthly calendar. The scripted table assembly begins in the Body portion of the document. Figure 49-1 shows the results.

Listing 49-1
A Static Table Generated by JavaScript

```
<HTML>
<HEAD>
<TITLE>JavaScripted Static Table</TITLE>
<SCRIPT LANGUAGE="JavaScript">
// function becomes a method for each month object
function getFirstDay(theYear, theMonth){
    var firstDate = new Date(theYear,theMonth,1)
    return firstDate.getDay() + 1
}
// number of days in the month
function getMonthLen(theYear, theMonth) {
    var oneDay = 1000 * 60 * 60 * 24
    var thisMonth = new Date(theYear, theMonth, 1)
    var nextMonth = new Date(theYear, theMonth + 1, 1)
    var len = Math.ceil((nextMonth.getTime() -
        thisMonth.getTime())/oneDay)
    return len
}
// correct for Y2K anomalies
function getY2KYear(today) {
    var yr = today.getYear()
    return ((yr < 1900) ? yr+1900 : yr)
}
// create basic array
theMonths = new MakeArray(12)
// load array with English month names
function MakeArray(n) {
    this[0] = "January"
    this[1] = "February"
    this[2] = "March"
    this[3] = "April"
    this[4] = "May"
    this[5] = "June"
```

```
    this[6] = "July"
    this[7] = "August"
    this[8] = "September"
    this[9] = "October"
    this[10] = "November"
    this[11] = "December"
    this.length = n
    return this
}
// end -->
</SCRIPT>
</HEAD>

<BODY>
<H1>Month at a Glance (Static)</H1>
<HR>
<SCRIPT LANGUAGE="JavaScript">
<!-- start
// initialize some variables for later
var today = new Date()
var theYear = getY2KYear(today)
var theMonth = today.getMonth() // for index into our array

// which is the first day of this month?
var firstDay = getFirstDay(theYear, theMonth)
// total number of <TD>...</TD> tags needed in for loop below
var howMany = getMonthLen(theYear, theMonth) + firstDay

// start assembling HTML for table
var content = "<CENTER><TABLE BORDER>"
// month and year display at top of calendar
content += "<TR><TH COLSPAN=7>" + theMonths[theMonth] + " " + theYear +
"</TH></TR>"
// days of the week at head of each column
content += "<TR><TH>Sun</TH><TH>Mon</TH><TH>Tue</TH><TH>Wed</TH>"
content += "<TH>Thu</TH><TH>Fri</TH><TH>Sat</TH></TR>"
content += "<TR>"

// populate calendar
for (var i = 1; i < howMany; i++) {
    if (i < firstDay) {
        // 'empty' boxes prior to first day
        content += "<TD></TD>"
    } else {
        // enter date number
        content += "<TD ALIGN='center'>" + (i - firstDay + 1) + "</TD>"
    }
    // start new row after each week
    if (i % 7 == 0 &&  i != howMany) {
        content += "</TR><TR>"
    }
}
content += "</TABLE></CENTER>"

// blast entire table's HTML to the document
document.write(content)
// end -->
</SCRIPT>
</BODY>
</HTML>
```

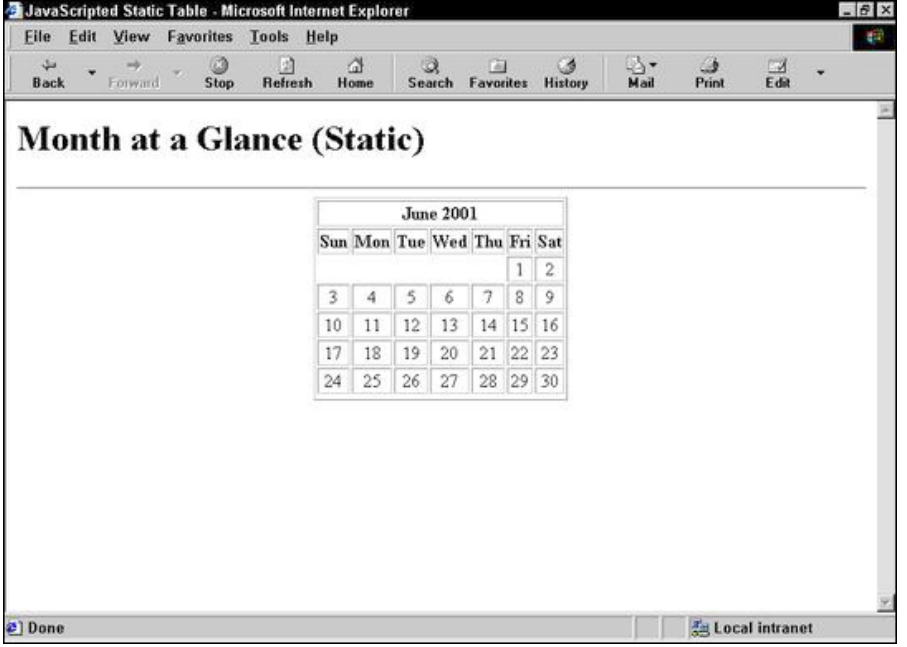

Figure 49-1
The static table calendar generated by Listing 49-1.

In this page, a little bit of the HTML — the `<H1>` heading and `<HR>` divider — is unscripted. The rest of the page consists entirely of the table definition, all of which is constructed in JavaScript. Though you may want to interlace straight HTML and scripted HTML within the table definition, bugs exist in NN2 and NN3 that make this tactic hazardous. The safest method is to define the entire table from the `<TABLE>` to `</TABLE>` tags in JavaScript and post it to the page in one or more `document.write()` methods.

Most of the work for assembling the calendar's data points occurs inside of the `for` loop. Because not every month starts on a Sunday, the script determines the day of the week on which the current month starts. For all fields prior to that day, the `for` loop writes empty `<TD></TD>` tags as placeholders. After the numbered days of the month begin, the `for` loop writes the date number inside the `<TD>...</TD>` tags. Whatever the script puts inside the tag pair is written to the page as flat HTML. Under script control like that in the example, however, the script can designate what goes into each data point — rather than writing fixed HTML for each month's calendar.

The important point to note in this example is that although the content of the page may change automatically over time (without having to redo any HTML for the next month), after the page is written, its contents cannot be changed. If you want to add controls or links that are to display another month or year, you have to rewrite the entire page. This can be accomplished by passing the desired month and year as a search string for the current page's URL and then assigning the combination to the `location.href` property. You also have to add script statements to the page that look for a URL search string, extract the passed values, and use those values to generate the calendar while the

page loads (see Chapter 17 for examples of how to accomplish this feat). But to bring a calendar such as this even more to life (while avoiding page reloading between views), you can implement it as a dynamic table.

Dynamic Tables

The only way to make data points of a table dynamically updatable in backward-compatible browsers is to turn those data points into text (or TEXTAREA) objects. The approach to this implementation is different from the static table because it involves the combination of *immediate* and *deferred* scripting. Immediate scripting facilitates the building of the table framework, complete with fields for every modifiable location in the table. Deferred scripting enables users to make choices from other interface elements, causing a new set of variable data to appear in the table's fields.

Listing 49-2 turns the preceding static calendar into a dynamic one by including controls that enable the user to select a month and year to display in the table. As testament to the support for absolute backward compatibility, a button triggers the redrawing of the calendar contents, rather than `onChange` event handlers in the SELECT elements. A bug in NN2 for Windows caused that event not to work for the SELECT object.

Form controls aside, the look of this version is quite different from the static calendar. Compare the appearance of the dynamic version shown in Figure 49-2 against the static version in Figure 49-1.

Listing 49-2
A Dynamic Calendar Table

```
<HTML>
<HEAD>
<TITLE>JavaScripted Dynamic Table</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!-- start
// function becomes a method for each month object
function getFirstDay(theYear, theMonth){
    var firstDate = new Date(theYear,theMonth,1)
    return firstDate.getDay()
}
// number of days in the month
function getMonthLen(theYear, theMonth) {
    var oneDay = 1000 * 60 * 60 * 24
    var thisMonth = new Date(theYear, theMonth, 1)
    var nextMonth = new Date(theYear, theMonth + 1, 1)
    var len = Math.ceil((nextMonth.getTime() -
        thisMonth.getTime())/oneDay)
    return len
}
// correct for Y2K anomalies
function getY2KYear(today) {
    var yr = today.getYear()
    return ((yr < 1900) ? yr+1900 : yr)
}
// create basic array
theMonths = new MakeArray(12)
// load array with English month names
function MakeArray(n) {
```

```
    this[0] = "January"
    this[1] = "February"
    this[2] = "March"
    this[3] = "April"
    this[4] = "May"
    this[5] = "June"
    this[6] = "July"
    this[7] = "August"
    this[8] = "September"
    this[9] = "October"
    this[10] = "November"
    this[11] = "December"
    this.length = n
    return this
}
// deferred function to fill fields of table
function populateFields(form) {
    // initialize variables for later from user selections
    var theMonth = form.chooseMonth.selectedIndex
    var theYear = form.chooseYear.options[form.chooseYear.selectedIndex].text
    // initialize date-dependent variables

    // which is the first day of this month?
    var firstDay = getFirstDay(theYear, theMonth)
    // total number of <TD>...</TD> tags needed in for loop below
    var howMany = getMonthLen(theYear, theMonth)

    // set month and year in top field
    form.oneMonth.value = theMonths[theMonth] + " " + theYear
    // fill fields of table
    for (var i = 0; i < 42; i++) {
        if (i < firstDay || i >= (howMany + firstDay)) {
            // before and after actual dates, empty fields
            // address fields by name and [index] number
            form.oneDay[i].value = ""
        } else {
            // enter date values
            form.oneDay[i].value = i - firstDay + 1
       }
    }
}

// end -->
</SCRIPT>
</HEAD>

<BODY>
<H1>Month at a Glance (Dynamic)</H1>
<HR>
<SCRIPT LANGUAGE="JavaScript">
<!-- start
// initialize variable with HTML for each day's field
// all will have same name, so we can access via index value
// empty event handler prevents
// reverse-loading bug in some platforms
var oneField = "<INPUT TYPE='text' NAME='oneDay' SIZE=2 onFocus=''>"
// start assembling HTML for raw table
var content = "<FORM><CENTER><TABLE BORDER>"
// field for month and year display at top of calendar
content += "<TR><TH COLSPAN=7><INPUT TYPE='text' NAME='oneMonth'></TH></TR>"
// days of the week at head of each column
content += "<TR><TH>Sun</TH><TH>Mon</TH><TH>Tue</TH><TH>Wed</TH>"
content += "<TH>Thu</TH><TH>Fri</TH><TH>Sat</TH></TR>"
content += "<TR>"

// layout 6 rows of fields for worst-case month
for (var i = 1; i < 43; i++) {
```

```
    content += "<TD ALIGN='middle'>" + oneField + "</TD>"
    if (i % 7 == 0) {
        content += "</TR><TR>"
    }
}

content += "</TABLE>"
// blast empty table to the document
document.write(content)

// end -->
</SCRIPT>
<SELECT NAME="chooseMonth">
<OPTION SELECTED>January<OPTION>February
<OPTION>March<OPTION>April<OPTION>May
<OPTION>June<OPTION>July<OPTION>August
<OPTION>September<OPTION>October<OPTION>November<OPTION>December
</SELECT>
<SELECT NAME="chooseYear">
<OPTION SELECTED>2000<OPTION>2001
<OPTION>2002<OPTION>2003
<OPTION>2004<OPTION>2005
<OPTION>2006<OPTION>2007
</SELECT>
<INPUT TYPE="button" NAME="updater" VALUE="Update Calendar"
onClick="populateFields(this.form)">
</FORM>
</BODY>
</HTML>
```

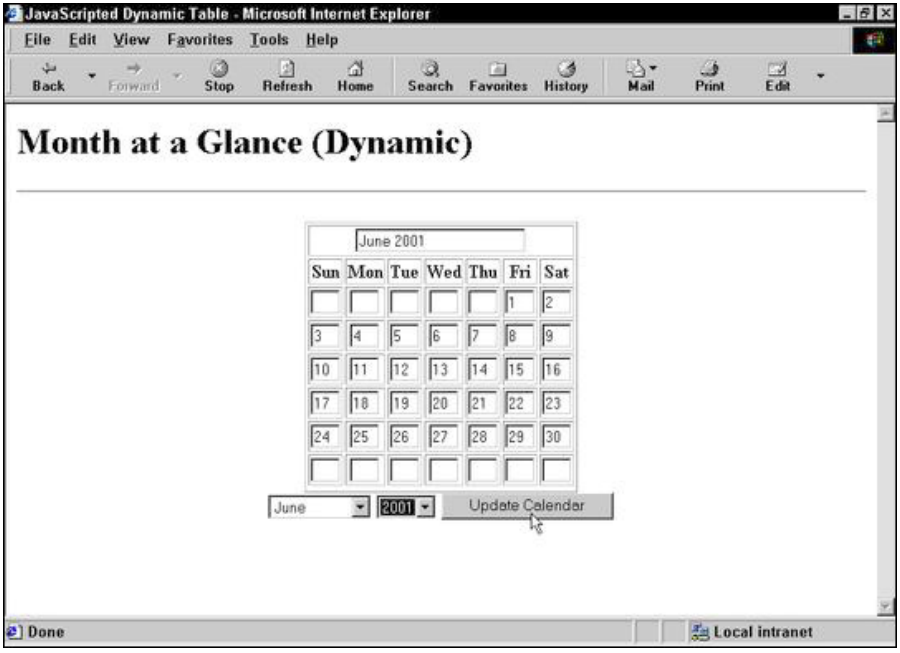

Figure 49-2
Dynamic calendar generated by Listing 49-2.

When you first load Listing 49-2, it creates an empty table. Even so, it may take a while to load, depending on the platform of your browser and the speed of your computer's processor. This page creates numerous text objects. An `onLoad` event handler in the Body definition also could easily set the necessary items to load the current month.

From a cosmetic point of view, the dynamic calendar may not be as pleasing as the static one in Figure 49-1. Several factors contribute to this appearance.

From a structural point of view, creating a table that can accommodate any possible layout of days and dates that a calendar may require is essential. That means a basic calendar consisting of six rows of fields. For many months, the last row remains completely empty. But because the table definition must be fixed when the page loads, this layout cannot change on the fly.

The more obvious cosmetic comparison comes from the font and alignment of data in text objects. Except for capabilities of browsers capable of using style sheets, you're stuck with what the browser presents in both categories. In the static version, you can define different font sizes and colors for various fields, if you want (such as coloring the entry for today's date). Not so in text objects in a backward-compatible program.

This cosmetic disadvantage, however, is a boon to functionality and interactivity on the page. Instead of the user being stuck with an unchanging calendar month, this version includes pop-up menus from which the user can select a month and year of choice. Clicking the Update Calendar button refills the calendar fields with data from the selected month.

One more disadvantage to this dynamic table surfaces, however: All text objects can be edited by the user. For many applications, this capability may not be a big deal. But if you're creating a table-based application that encourages users to enter values in some fields, be prepared (in other words, have event handlers in place) to either handle calculations based on changes to any field or to alert users that the fields cannot be changed (and restore the correct value).

Hybrids

It will probably be the rare scripted table that is entirely dynamic. In fact, the one in Figure 49-2 is a hybrid of static and dynamic table definitions. The days of the week at the top of each column are hard-wired into the table as static elements. If your table design can accommodate both styles, implement your tables that way. The fewer the number of text objects defined for a page, the better the performance for rendering the page, and the less confusion for the page's users.

Dynamic HTML Tables

If you have the luxury of developing for IE4+ and/or NN6, you have all the resources of the TABLE and related element objects, as described in Chapter 27. The resulting application will appear to be much more polished, because not only does your content flow inside a table (which you can style to your heart's delight), but the content is dynamic within the table.

Listing 49-3 blends the calendar calculations from the earlier two calendar versions with the powers of IE4+ and W3C DOMs. A change to a requested calendar month or year instantly redraws the body of the table, without disturbing the rest of the page (see Figure 49-3).

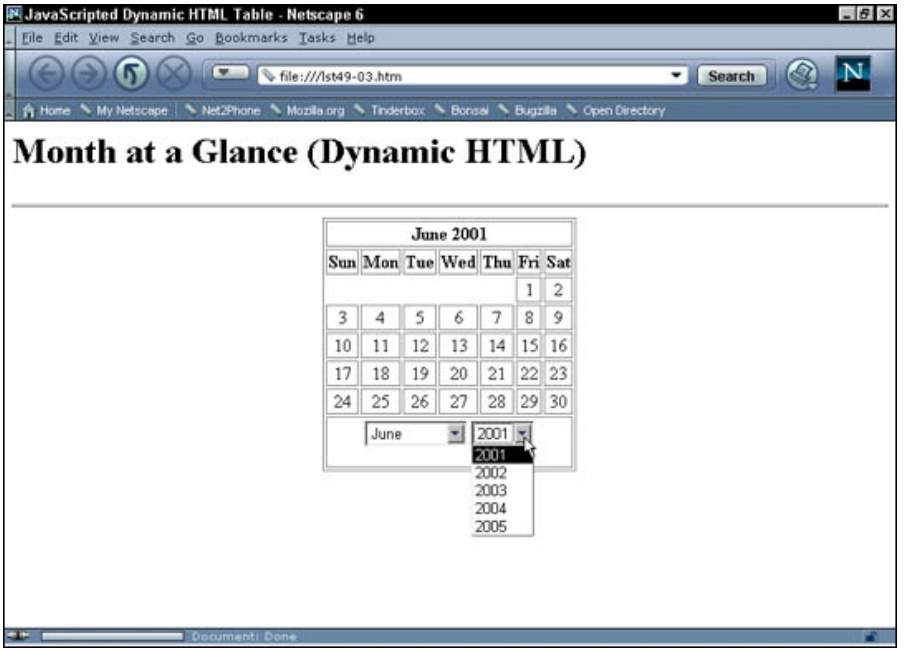

Figure 49-3
DHTML table

Basic date calculations are identical to the other two versions. Because this page has to be used with more modern browsers, it can use a genuine `Array` object for the month names. Also, the way the table must be constructed each time is very different from two previous versions. In this version, the script creates new table rows, creates new cells for those rows, and then populates those cells with the date numbers. Repeat loop logic is quite different, relying on a combination of `while` and `for` loops to get the job done.

Other features made possible by more modern browsers include automatic population of the list of available years. This page will never go out of style (unless browsers in 2050 no longer use JavaScript). There is also more automation in the triggers of the function that populates the table.

Listing 49-3
Dynamic HTML Calendar

```
<HTML>
<HEAD>
<TITLE>JavaScripted Dynamic HTML Table</TITLE>
<STYLE TYPE="text/css">
TD, TH {text-align:center}
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
/*******************
```

```
  UTILITY FUNCTIONS
********************/
// day of week of month's first day
function getFirstDay(theYear, theMonth){
    var firstDate = new Date(theYear,theMonth,1)
    return firstDate.getDay()
}
// number of days in the month
function getMonthLen(theYear, theMonth) {
    var oneDay = 1000 * 60 * 60 * 24
    var thisMonth = new Date(theYear, theMonth, 1)
    var nextMonth = new Date(theYear, theMonth + 1, 1)
    var len = Math.ceil((nextMonth.getTime() -
        thisMonth.getTime())/oneDay)
    return len
}
// create array of English month names
var theMonths =
["January","February","March","April","May","June","July","August",
"September","October","November","December"]
// return IE4+ or W3C DOM reference for an ID
function getObject(obj) {
    var theObj
    if (document.all) {
        if (typeof obj == "string") {
            return document.all(obj)
        } else {
            return obj.style
        }
    }
    if (document.getElementById) {
        if (typeof obj == "string") {
            return document.getElementById(obj)
        } else {
            return obj.style
        }
    }
    return null
}

/************************
  DRAW CALENDAR CONTENTS
************************/
// clear and re-populate table based on form's selections
function populateTable(form) {
    var theMonth = form.chooseMonth.selectedIndex
    var theYear =
parseInt(form.chooseYear.options[form.chooseYear.selectedIndex].text)
    // initialize date-dependent variables
    var firstDay = getFirstDay(theYear, theMonth)
    var howMany = getMonthLen(theYear, theMonth)

    // fill in month/year in table header
    getObject("tableHeader").innerHTML = theMonths[theMonth] +
    " " + theYear

    // initialize vars for table creation
    var dayCounter = 1
    var TBody = getObject("tableBody")
    // clear any existing rows
    while (TBody.rows.length > 0) {
        TBody.deleteRow(0)
    }
    var newR, newC
    var done=false
    while (!done) {
        // create new row at end
```

```
        newR = TBody.insertRow(TBody.rows.length)
        for (var i = 0; i < 7; i++) {
            // create new cell at end of row
            newC = newR.insertCell(newR.cells.length)
            if (TBody.rows.length == 1 && i < firstDay) {
                // no content for boxes before first day
                newC.innerHTML = ""
                continue
            }
            if (dayCounter == howMany) {
                // no more rows after this one
                done = true
            }
            // plug in date (or empty for boxes after last day)
            newC.innerHTML = (dayCounter <= howMany) ?
                dayCounter++ : ""
        }

    }
}

/*******************
  INITIALIZATIONS
********************/
// create dynamic list of year choices
function fillYears() {
    var today = new Date()
    var thisYear = today.getFullYear()
    var yearChooser = document.dateChooser.chooseYear
    for (i = thisYear; i < thisYear + 5; i++) {
        yearChooser.options[yearChooser.options.length] = new Option(i, i)
    }
    setCurrMonth(today)
}
// set month choice to current month
function setCurrMonth(today) {
    document.dateChooser.chooseMonth.selectedIndex = today.getMonth()
}
</SCRIPT>
</HEAD>

<BODY onLoad="fillYears(); populateTable(document.dateChooser)">
<H1>Month at a Glance (Dynamic HTML)</H1>
<HR>
<TABLE ID="calendarTable" BORDER=1 ALIGN="center">
<TR>
    <TH ID="tableHeader" COLSPAN=7></TH>
</TR>
<TR><TH>Sun</TH><TH>Mon</TH><TH>Tue</TH><TH>Wed</TH>
<TH>Thu</TH><TH>Fri</TH><TH>Sat</TH></TR>
<TBODY ID="tableBody"></TBODY>
<TR>
    <TD COLSPAN=7>
    <P>
    <FORM NAME="dateChooser">
        <SELECT NAME="chooseMonth"
        onChange="populateTable(this.form)">
            <OPTION SELECTED>January<OPTION>February
            <OPTION>March<OPTION>April<OPTION>May
            <OPTION>June<OPTION>July<OPTION>August
            <OPTION>September<OPTION>October
            <OPTION>November<OPTION>December
    </SELECT>
    <SELECT NAME="chooseYear" onChange="populateTable(this.form)">
    </SELECT>
    </FORM>
    </P></TD>
```

```
</TR>
</TABLE>
</BODY>
</HTML>
```

Further Thoughts

The best deployment of an interactive calendar requires the kind of Dynamic HTML currently available in IE4+ and W3C DOMs. Moreover, the cells in those DOMs can receive mouse events so that a user can click a cell and it will highlight perhaps in a different color or display some related, but otherwise hidden, information.

A logical application for such a dynamic calendar would be in a pop-up window or frame that lets a user select a date for entry into a form date field. It eliminates typing in a specific date format, thereby assuring a valid date entry every time. Without DHTML, you can create a static version of the calendar that renders the numbers in the calendar cells as HTML links. Those links can use a `javascript:` URL to invoke a function call that sets a date field in the main form.

> **Note**
>
> The dynamic calendar in Listing 49-2 assumes that the browser treats like-named text boxes in a form as an array of fields. While this is true in all versions of NN, IE3 does not follow this behavior. To accommodate this anomaly, you must modify the script to assign unique names to each field (with an index number as part of the name) and use the `eval()` function to assist looping through the fields to populate them. On the CD-ROM is Listing 49-2b, which is a cross-compatible version of the dynamic calendar.