Chapter 54: Application: Intelligent “Updated” Flags

In This Chapter

Temporary and persistent cookies

World time calculations

CGI-like intelligence

It happens to every active Web user all the time: You visit a site periodically and never know for sure what material is new since your last visit. Often, Web page authors may flag items with “New” or “Updated” `.gif` images after they update those items themselves. But if you fail to visit the site over a few modification sessions, the only items you find flagged are those that are new as of the most recent update by the page’s author. Several new items from a few weeks back may be of vital interest to you, but you won’t have the time to look through the whole site in search of material that is more recent than *your* last visit. Even if the items display their modification dates, do you remember for sure the date and time of your last visit to the page?

As much as I might expect a CGI program and database on a Web site to keep track of my last visit, that really is asking a great deal of the Web site. Besides, not every Web site has the wherewithal to build such a database system — if it can even put up its own CGIs. Plus, some users won’t visit sites if they need to identify themselves or register.

After surveying the way scriptable browsers store cookie information and how time calculations are performed under NN3+ and IE4+, I found that a feasible alternative is to build this functionality into HTML documents and let the scripting manage the feature for users. The goal is to save in the visitor’s cookie file the date and time of the last visit to a page and then use that point as a measure against items that have an authorship time stamp in the HTML document.

The Cookie Conundrum

Managing the cookie situation in this application is a bit more complicated than you may think. The reason is that you have to take into account the possible ways visitors may come and go from your site while surfing the Web. You cannot use just one cookie to store the last time a user visits the site, because you cannot predict when you should update that information with today’s date and time. For example, if you have a cookie with the previous visit in it, you eventually need to store today’s visit. But you cannot afford to overwrite the previous visit immediately (say in `onLoad`) because your scripts need that information to compare against items on the page not only right now, but even after the visitor vectors off from a link and comes back later. That also means you cannot update that last visit cookie solely via an `onUnload` event handler, because that, too, would overwrite information that you need when the visitor comes back a minute later.

To solve the problem, I devised a system of two cookies. One is written to the cookie that is given an expiration date of some time out in the future — the *hard cookie,* I call it. The other is a temporary — *soft cookie*, which stays in cookie memory but is never written to the file. Such temporary cookies are automatically erased as the browser quits.

The hard cookie stores the time stamp when a visitor first loads the page since the last launch of the browser. In other words, the hard cookie contains a time stamp of the current visit. Before the previous entry is overwritten, however, it is copied into the soft cookie. That soft cookie maintains the time stamp of the previous visit and becomes the measure against which author time stamps in the HTML document are compared. To guard against inadvertent overwriting of both cookies, a function triggered by the document's `onLoad` event handler looks to see if the soft cookie has any data in it. If so, then the function knows that the visitor has been to this page in the current session and leaves the current settings alone. Thus, the visitor can come to the site and see what's new, vector off to some other location, and come back to see the same new items flagged and pick up from there.

One potential downside to this system is that if a user never quits the browser (or if the browser quits only by crashing), the cookies will never be updated. If you discover that a great deal of your users keep their computers and browsers running all the time, you can build in a kind of timeout that invalidates the soft cookie if the hard cookie is more than, say, 12 hours old.

Time's Not on Your Side

Thanks to over fifteen years' experience programming applications that involve tracking time, I am overly sensitive to the way computers and programming languages treat time on a global basis. This issue is a thorny one, what with the vagaries of Daylight Savings Time and time zones in some parts of the world that differ from their neighbors by increments other than whole hours.

In the case of working with time in JavaScript, you're at the mercy of how the browser and JavaScript interpreter deal with times as reflected by often imperfect operating systems. Those scripters who tried to script time-sensitive data in NN2 must have certainly experienced the wide fluctuations in the way each platform tracked time internally (over and above the outright bugs, especially in the Mac version of NN2). Fortunately, the situation improved significantly with NN3 and has only gotten better in all scriptable browsers. That's not to say all the bugs are gone, but at least they're manageable.

To accomplish a time tracking scheme for this application, I had to be aware of two times: the local time of the visitor and the local time of the page author. Making times match up in what can be widely disparate time zones, I use one time zone — GMT — as the reference point.

When a visitor arrives at the page, the browser needs to save that moment in time so that it can be the comparison measure for the *next* visit. Fortunately, whenever you create a new date object in JavaScript, it does so internally as the GMT date and time. Even though the way you attempt to read the date and time created by JavaScript shows you the results in your computer's local time, the display is actually filtered through the time zone offset as directed by your computer's time control panel. In other words, the millisecond value of every date object you create is maintained in memory in its GMT form. That's fine for the visitor's cookie value.

For the page author, however, I was presented with a different problem. Rather than force the author to convert the time stamps throughout the document to GMT, I wanted to let the author enter dates and times in local time. Aside from the fact that many people have trouble doing time zone conversions, looking at an existing item in the HTML with a local time stamp and instantly recognizing when that was last updated is much easier.

The problem, then, is how to let the visitor's browser know what time the author's time stamp is in GMT terms. To solve the issue, the author's time stamp needs to include a reference to the author's time zone relative to GMT. An Internet convention provides a couple of ways to do this: specifying the number of hours and minutes difference from GMT or, where supported by the browser, the abbreviation of the time zone. In JavaScript, you can create a new date object out of one of the specially formatted strings containing the date, time, and time zone. Three examples follow for the Christmas Eve dinner that starts at 6 p.m. in the Eastern Standard Time zone of North America:

```
var myDate = new Date("24 Dec 1997 23:00:00 GMT")
var myDate = new Date("24 Dec 1997 18:00:00 GMT-0500")
var myDate = new Date("24 Dec 1997 18:00:00 EST")
```

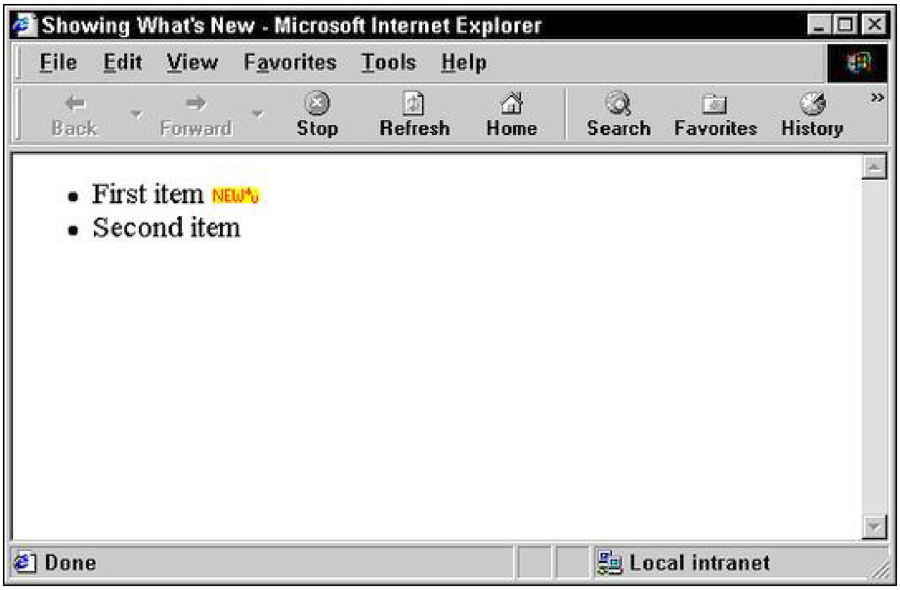
The first assumes you know the Greenwich Mean Time for the date and time that you want to specify. But if you don't, you can use the GMT designation and offset value. The syntax indicates the date and time is in a time zone exactly five hours west of GMT (values to the east would be positive numbers) in `hhmm` format. Browsers also know all of the time zone abbreviations for North America (EST, EDT, CST, CDT, MST, MDT, PST, and PDT, where "S" is for standard time and "D" is for daylight time).

When a user visits a page with this application embedded in it, the visitor's browser converts the author's time stamp to GMT (with the help of the author's zone offset parameter), so that both the author time stamp and last visit time stamp (in the soft cookie) are comparing apples to apples.

The Application

All of this discussion may make the application sound complicated. That may be true, internally. But the goal, as in most of the samples in this book, is to make the application easy to use in your site, even if you’re not sure how everything works inside.

The sample page described in this chapter and on the CD-ROM (`whatsnew.htm`) is pretty boring to look at, because the power all lies in the scripting that users don’t see (see Figure 54-1). Though this figure may be the most uninspired graphic presentation of the book, the functionality may be the most valuable addition that you make to your Web site.



- First item NEW4U
- Second item

Figure 54-1
An item flagged as being new since my last visit to the page

When you first open the document (do so from a copy on your hard disk so that you can modify the author time stamp in a moment), all you see are the two items on the page without any flags. Although both entries have author time stamps that pre-date the time you’re viewing the page, a soft cookie does not yet exist against which to compare those times. But the act of making the first visit to the page has created a hard cookie of the date and time that you first opened the page.

Quit the browser to get that hard cookie officially written to the cookie file. Then open the `whatsnew.htm` file in your script editor. Scroll to the bottom of the document, where you see the `<BODY>` tag and the interlaced scripts that time stamp anything that you want on the page. This application is designed to display a special `.gif` image that says “NEW 4U” whenever an item has been updated since your last visit.

Each interlaced script looks like this:

```
<SCRIPT LANGUAGE="JavaScript1.1">
document.write(newAsOf("12 Oct 2001 13:36:00 PDT"))
```

```
</SCRIPT>
```

By virtue of all scripts in this page being at the JavaScript 1.1 level, only those browsers so equipped will bother with the scripting (which also means that others lose out on this great visitor service). The `document.write()` method writes to the page whatever HTML comes back from the `newAsOf()` function. The parameter to the `newAsOf()` function is what holds the author time stamp and zone offset information. The time stamp value must be in the string format, as shown in the preceding example, with the date and time following the exact order ("`dd mmm yyyy hh:mm:ss`"). Month abbreviations are in English (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec).

As you see in the code that follows, the `newAsOf()` function returns an `<IMG>` tag with the “NEW 4U” image if the author time stamp (after appropriate conversion) is later than the soft cookie value. This image can be placed anywhere in a document. For example, at my Web site, I sometimes place the image before a contents listing rather than at the end. This means, too, that if part of your page is written by `document.write()` methods, you can just insert the `newAsOf()` function call as a parameter to your own `document.write()` calls.

If you want to see the author time stamping work, edit one of the time stamps in the `whatsnew.htm` file to reflect the current time. Save the document and relaunch the browser to view the page. The item whose author time stamp you modified should now show the bright “NEW 4U” image.

The Code

The sample page starts by initializing three global variables that are used in the statements that follow. One variable is a Boolean value indicating whether the visitor has been to the page before. Another variable, `lastVisit`, holds the value of the soft cookie whenever the visitor is at this page. One other variable, `dateAdjustment`, is (unfortunately) necessary to take into account a date bug that persists in Macintosh versions of Navigator (times of new date objects can be off by one hour). I use this variable to automatically handle any discrepancies.

```
<HTML>
<HEAD>
<TITLE>Showing What's New</TITLE>
<SCRIPT LANGUAGE="JavaScript1.1">
<!-- begin hiding

// globals
var repeatCustomer = false
var lastVisit = 0 // to hold date & time of previous access in GMT
milliseconds
var dateAdjustment = 0 // to accommodate date bugs on some platforms
```

For reading and writing cookie data, I use virtually the same cookie functions from the outline table of contents (see Chapter 52). The only difference is that the cookie writing function includes an expiration date, because I want this cookie to hang around in the cookie file for awhile — at least until the next visit, whenever that may be.

```
// shared cookie functions
var mycookie = document.cookie
// read cookie data
function getCookieData(name) {
    var label = name + "="
    var labelLen = label.length
    var cLen = mycookie.length
    var i = 0
    while (i < cLen) {
        var j = i + labelLen
        if (mycookie.substring(i,j) == label) {
            var cEnd = mycookie.indexOf(";",j)
            if (cEnd ==    -1) {
                cEnd = mycookie.length
            }
            return unescape(mycookie.substring(j,cEnd))
        }
        i++
    }
    return ""
}

// write cookie data
function setCookieData(name,dateData,expires) {
    mycookie = document.cookie = name + "=" +
        dateData + "; expires=" + expires
}
```

Notice that the `setCookieData()` function still maintains a level of reusability by requiring a name for the cookie to be passed as a parameter along with the data and expiration date. I could have hard-wired the name into this function, but that goes against my philosophy of designing for reusability.

Next comes a function that figures out if any problems with JavaScript date accuracy exist on any platform. Essentially, the function creates two date objects, one to serve as a baseline. Even the baseline date can be bad (as it is on Mac versions of NN3), so to test against it, you want to use the second object to create another date using the first date object's own values as a parameter. If any major discrepancies occur, they will show up loud and clear.

```
// set dateAdjustment to accommodate Mac bug in Navigator 3
function adjustDate() {
    var base = new Date()
    var testDate = base
    testDate = testDate.toLocaleString()
    testDate = new Date(testDate)
    dateAdjustment = testDate.getTime() - base.getTime()
}
```

In truth, this function always shows some adjustment error, because both the baseline date and test date cannot be created simultaneously. Even in an accurate system, the two will vary by some small number of milliseconds. For the purposes here, this amount of variance is insignificant.

Setting the stage

Functions in the next part of the script get your cookies all in a row. The first function (`saveCurrentVisit()`) deals with the visitor's local time, converting it to a form that will be useful on the next visit. Although one of the local variables is called `nowGMT`, all the variable does is take the new date object and convert it to the GMT milliseconds value (minus any `dateAdjustment` value) by invoking the `getTime()` method of the date object. I use this name in the variable to help me remember what the value represents:

```
// write date of current visit (in GMT time) to cookie
function saveCurrentVisit() {
    var visitDate = new Date()
    var nowGMT = visitDate.getTime() - dateAdjustment
    var expires = nowGMT + (180 * 24 * 60 * 60 *1000)
    expires = new Date(expires)
    expires = expires.toGMTString()
    setCookieData("lastVisit", nowGMT, expires)
}
```

From the current time, I create an expiration date for the cookie. The example shows a date roughly six months (180 days, to be exact) from the current time. I leave the precise expiration date up to your conscience and how long you want the value to linger in a user's cookie file.

The final act of the `saveCurrentVisit()` function is to pass the relevant values to the function that actually writes the cookie data. I assign the name `lastVisit` to the cookie. If you want to manage this information for several different pages, then assign a different cookie name for each page. This setup can be important in case a user gets to only part of your site during a visit. On the next visit, the code can point to page-specific newness of items.

The bulk of what happens in this application takes place in an initialization function. All the cookie swapping occurs there, as well as the setting of the `repeatCustomer` global variable value:

```
// set up global variables and establish whether user is a newbie
function initialize() {
    var lastStoredVisit = getCookieData("lastVisit")
    var nextPrevStoredVisit = getCookieData("nextPrevVisit")

    adjustDate()

    if (!lastStoredVisit) {
        // never been here before
        saveCurrentVisit()
        repeatCustomer = false
    } else {
        // been here before...
        if (!nextPrevStoredVisit) {
            // but first time this session
            // so set cookie only for current session
            setCookieData("nextPrevVisit",lastStoredVisit,"")
            lastVisit = parseFloat(lastStoredVisit)
            saveCurrentVisit()
            repeatCustomer = true
        } else {
            // back again during this session (perhaps reload or Back)
            lastVisit = parseFloat(nextPrevStoredVisit)
            repeatCustomer = true
       }
    }
}
```

```
initialize()
```

The first two statements retrieve both hard (`lastVisit`) and soft (`nextPrevVisit`) cookie values. After calling the function that sets any necessary date adjustment, the script starts examining the values of the cookies to find out where the visitor stands upon coming to the page.

The first test is whether the person has ever been to the page before. You do this by checking whether a hard cookie value (which would have been set in a previous visit) exists. If no such cookie value exists, then the current visit time is written to the hard cookie and `repeatCustomer` is set to `false`. These actions prepare the visitor's cookie value for the *next* visit.

Should a user already be a repeat customer, you have to evaluate whether this visit is the user's first visit since launching the browser. You do that by checking for a value in the soft cookie. If that value doesn't exist, then it means the user is here for the first time "today." You then grab the hard cookie and drop it into the soft cookie before writing today's visit to the hard cookie.

For repeat customers who have been here earlier in this session, you update the `lastVisit` global variable from the cookie value. The variable value will have been destroyed when the user left the page just a little while ago, whereas the soft cookie remains intact, enabling you to update the variable value now.

Outside of the function definition, the script automatically executes the `initialize()` function by that single statement. This function runs every time the page loads.

The date comparison

Every interlaced script in the body of the document calls the `newAsOf()` function to find out if the author's time stamp is after the last visit of the user to the page. This function is where the time zone differences between visitor and author must be neutralized so that a valid comparison can be made:

```
function newAsOf(authorDate) {
    authorDate = new Date(authorDate)
    var itemUpdated = authorDate.getTime()
    return ((itemUpdated > lastVisit) && repeatCustomer) ?
        "<IMG SRC='updated.gif' HEIGHT=10 WIDTH=30>" : ""
}
// end hiding -->
</SCRIPT>
</HEAD>
```

As you saw earlier, calls to this function require one parameter: a specially formatted date string that includes time zone information. The first task in the function is to re-cast the author's date string to a date object. You reuse the variable name (`authorDate`) because its meaning is quite clear. The date object created here is stored internally in the browser in GMT time, relative to the time zone data supplied in the parameter. To assist in the comparison against the `lastVisit` time (stored in milliseconds), the `getTime()` method converts `authorDate` to GMT milliseconds.

The last statement of the function is a conditional expression that returns the `<IMG>` tag definition for the "NEW 4U" image only if the author's time stamp is later than the soft cookie value and the visitor has been here before. Otherwise, the function returns an empty string. Any `document.write()` method that calls this function and executes via this branch writes an empty string — nothing — to the page.

A live <BODY>

For the sample document, I have you create a simple bulleted list of two entries, imaginatively called "First item" and "Second item." Interlaced into the HTML are scripts that are ready to insert the "NEW 4U" image if the author time stamp is new enough:

```
<BODY>
<UL>
<LI>First item
<SCRIPT LANGUAGE="JavaScript1.1">
<!--
document.write(newAsOf("20 Oct 2000 09:36:00 PDT"))
//-->
</SCRIPT>
<LI>Second item
<SCRIPT LANGUAGE="JavaScript1.1">
<!--
document.write(newAsOf("18 Oct 2000 17:40:00 PDT"))
//-->
</SCRIPT>
</UL>
</BODY>
</HTML>
```

All these script tags make the HTML a bit hard to read, but I believe the functionality is worth the effort. Moreover, by specifying the JavaScript 1.1 language attribute, the scripts are completely ignored by older JavaScript-enabled browsers. Only the now very rare, exceedingly brain-dead browsers, which get tripped up on the SGML comment lines, would know that something out of the ordinary is taking place.

Further Thoughts

You can, perhaps, go overboard with the way that you use this technique at a Web site. Like most features in JavaScript, I recommend using it in moderation and confining the flags to high-traffic areas that repeat visitors frequent. One hazard is that you can run out of the 20 cookies if you have too many page-specific listings.

You can share the same cookie among documents in related frames. Locate all the functions from the script in this chapter's Head section into a Head section of a framesetting document. Then, modify the call to the `newAsOf()` function by pointing it to the parent:

```
document.write(parent.newAsOf("18 Oct 2000 17:40:00 PDT"))
```

That way, one cookie can take care of all documents that you display in that frameset.