



Level: Beginner
Works with: Domino 6
Updated: 01-Oct-2002

Browser caching and response header rules

by
John
Chamberlain

If you have been following the earlier articles in this series, [Building Web applications in Domino 6: A tutorial on Web site addressing](#), [Building Web applications in Domino 6: Web site rules](#), and [Building Web applications in Domino 6: Accessing and protecting the file system](#), you know all about the Web Site Rule document in the new Server\Internet Sites view in the Domino Directory and know how to create redirection, substitution, and directory rules. In this article we will take a look at the final type of rule you can create: HTTP response header rules.

This article will be of interest to all system administrators and Webmasters who plan to host Web sites with Domino 6. It assumes a familiarity with the Domino Directory and administrative tasks. In addition, it would be helpful to read the three previous articles in this series mentioned above.

What is a response header rule?

Every HTTP browser request and server response begins with a set of headers that describe the data that is being transmitted. The HTTP response header rule allows an application designer to customize the headers that Domino sends with responses to specified requests.

Response rules differ from the other types of rules in one very important way. The Domino HTTP server task applies redirection, substitution, and directory rules to the *incoming request* before passing the request to an application for processing. Response rules, on the other hand, are applied to the *outgoing response*, just before the task transmits the response to the browser.

The most important use of response rules is to improve the performance of browser caching. An application designer can add headers that provide the browser with important information about the volatility of the material being cached. So let's start off this article by taking an in-depth look at browser and server caching.

Understanding caching

First, what is a "cache"? Broadly speaking, a cache is an area of memory or disk space in which a software program can store data for reuse. Caching is valuable in situations where creating or retrieving data from scratch is a relatively expensive operation because of processing time, resource usage, or transmission bandwidth. In networked environments such as the Internet, caches can be maintained at the source machine where data is generated (such as a Web server), at the machine that requested the data (such as the browser on a user's home computer), or on any node in the intervening network (such as a proxy server).

In the Web world, all modern browsers can cache responses to HTTP requests. The typical Web page that you see in your browser usually isn't the result of a single request, but is built up from a large number of requests. For example, the HTML for the page may contain many links to graphic elements, each one of which must be retrieved by the browser by sending a separate request. The browser will store the data returned for each request separately in its cache. To put it another way, the smallest "cache unit" for the browser is a single HTTP response

(although we should mention that browsers may also store other information in their caches such as cookies and saved user credentials).

Server-side caching

The situation on the server, called server-side caching, is different. Servers may also cache responses that were sent out to browsers, but servers also have the opportunity to cache the data and resources that were used to generate a response. This caching can take place in many different layers of the server code, and indeed, even in the operating system. Let's take a typical Domino request such as `/catalog.nsf/products?OpenView`. To generate the response to this request, Domino must read the database and convert the view index into HTML. Caching occurs everywhere: the operating system caches the file-system pages; Domino caches the database handle, design elements (such as the view template), and user credentials; and the view index itself may be available in memory.

Besides this low-level caching, Domino R5 also cached complete HTTP responses to NSF requests in a special cache called the "command cache." In Domino 6, we have *removed* the command cache. Why? Because tests showed that the command cache did not significantly improve the performance of typical Web applications over and above the lower-level caching that was already taking place. In Domino 6, we have done considerable work to enhance the performance of the NSF core code, further reducing the benefit of maintaining an extra high-level cache.

No matter where a cache is maintained, the effectiveness of the cache depends on two critical principles: timeliness and security. The program maintaining the cache must make every effort to insure that the data is kept up-to-date, and that cached data is displayed only to the appropriate user. For example, if you use the online Web services of your bank to check your account balances, you take two things for granted: first, that you always see up-to-the-minute numbers, not some numbers that were cached last week, and second, that you get *your* account balances, not somebody else's!

The requirement for a cache to conform to those two principles explains why the server-side command-cache in R5 wasn't very effective. Domino is designed to support applications that are very dynamic (the data changes constantly) and are very secure (the data is protected by many layers of access control). In such applications, only a very small percentage of complete HTTP responses are truly static and public and thus, worthy to be cached on the server. Instead, it is much more effective for Domino to cache the smaller units that are used to create responses, such as design elements, view indices, and so on.

Browser caching

That's the story for server-side caching. What about client-side (browser) caching? A browser determines the "cachability" of an HTTP response by looking for the headers Last-Modified, Expires, and Cache-Control, which we'll describe in a minute. This is where the HTTP response header rule comes into play. A Domino 6 application designer can *significantly* increase the performance of the application by creating rules that manipulate the caching headers to maximize the cachability of a response. Thus, the application designer helps the browser to maintain its cache much more effectively. What makes this method so powerful is that the application designer has much better knowledge of how dynamic an application really is. The Domino server and the user's browser must both be very conservative about caching to insure that the user never sees stale data, but the application designer can use her superior knowledge of the real behavior of the application to allow the browser to cache more responses.

The caching headers are defined in the HTTP/1.1 specification, [RFC 2616](#), which also outlines the general procedures for the server and browser to follow in order to insure that the caches are kept up-to-date.

The Last-Modified header

The most important header is Last-Modified, which indicates when the resource or resources used to generate a response were last changed on the server. Let's take a look at how this works with the simple example of a Domino database page element:

1. The user requests the page through the browser with an `?OpenPage` command.
2. Domino returns the page and includes a Last-Modified header that specifies the time that the page was last changed.
3. The browser adds the page to its local cache and saves the Last-Modified time with it.
4. The next time the user requests the page, the browser adds an If-Modified-Since header to its request that specifies the Last-Modified time of the cached page.
5. If the page hasn't been modified since the If-Modified-Since time, Domino doesn't send the page, but instead just returns the HTTP status code 304 (Not Modified). This signals the browser that its cached page is still valid, and the browser will display the cached page to the user.
6. If the page *has* been modified, Domino sends the whole page again with an updated Last-Modified header.

The browser replaces its cached page with the fresh version and records the new Last-Modified time.

This is a simple example; if the user requested a more complex element such as a document that contains subforms and shared fields, then Domino would have to determine the modification time for each of the individual components and return the most recent value as the Last-Modified time.

The Expires header

The Expires header is the opposite of Last-Modified: the server uses the Expires header to tell the browser when the resources are *expected* to change, that is, for how long the browser can expect its cached response to remain valid. An accurate Expires header can greatly help application performance because if a cached response is requested again within its Expires time, the browser immediately redisplay the cached response without bothering to send an If-Modified-Since request. Therefore, reusing a cached response with an Expires header uses zero bandwidth and zero server processing time.

Of course, generating Expires headers automatically is tricky because the server can't predict the future (perhaps someday quantum computers *will* be able to see into the future, but today we must live with this limitation). There's no way Domino can know when a piece of data might change or when an application designer might modify a design element. For that reason, Domino never sends Expires headers with a future date. Domino *does* send Expires headers with a *past* date on responses that should *not* be cached, because some browsers expect that behavior, but generating future dates automatically is out of the question.

However, the application designer has a huge advantage over the server; she *can* predict the future because of her knowledge of the application. That's where the response rule comes in handy; the designer can define a rule to add Expires headers to responses based on when she expects the resources to change. Her knowledge may be very exact, as in the case of regularly-scheduled updates, or more general, which may be acceptable if it isn't important for the user to always see up-to-the-second data. We'll discuss examples of the Expires header in a minute.

The Cache-Control header

The Cache-Control header provides explicit instructions to browser and proxy server caches. This header has a number of options defined in RFC 2616, but the only options generated by Domino are "no-cache" and "private":

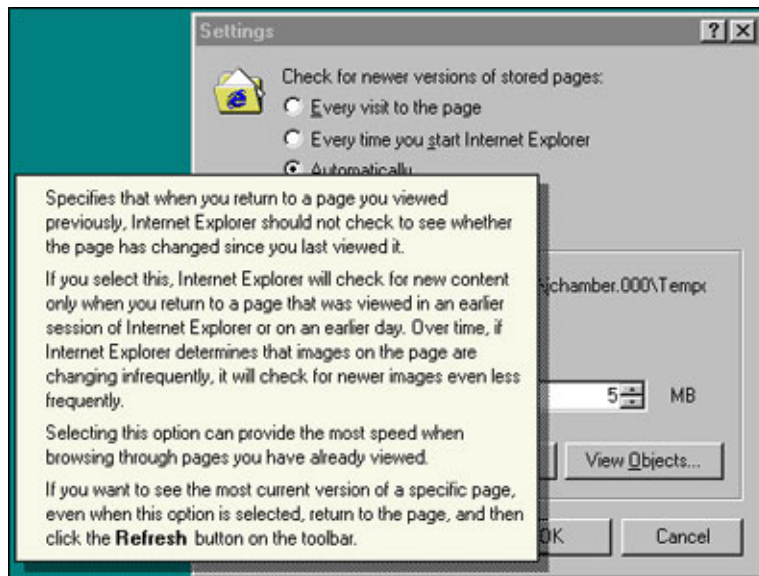
- "Cache-Control: no-cache" is added to responses that should definitely *not* be cached, such as responses that are dependent on the current time or are unique to the authenticated user. The Cache-Control header is new to HTTP 1.1. The HTTP 1.0 specification defined the header "Pragma: no-cache" for noncacheable responses, and Domino will use "Pragma: no-cache" instead of "Cache-Control: no-cache" in response to HTTP 1.0 requests.
- "Cache-Control: private" is added to responses that are cacheable but are specific to a particular browser configuration, such as the browser's language settings. It is up to the browser to decide if it will cache "private" responses. There isn't an HTTP 1.0 equivalent to "Cache-Control: private," so Domino downgrades private HTTP 1.0 responses to "Pragma: no-cache."

If the application designer doesn't like the default Domino behavior for the Cache-Control header, she can use a response rule to override Domino. We'll see examples of this below.

Caching in the real world

But before we go any further, here's a big caveat: our discussion of caching seems pretty straightforward, but in the real world things usually aren't that simple. First of all, every browser implements caching a little differently, and in fact, different versions of the same browser may have slightly different behavior. Browsers also give the user some control over caching, and choosing the right options is very important.

For example, the following screen shows the caching settings window for Microsoft Internet Explorer 5.5, with the help description for the "Automatically" option. Note that if this option is selected, IE will only check the validity of a cached page (by sending a request with an If-Modified-Since header) if the page was last accessed in an earlier browser session or on a previous day. That is, IE will *not* check the page if the user navigates to it more than once in the current session (for example, by backing up to it or selecting it from the History list). Improper browser settings are a bane to site administrators and Lotus support personnel. We regularly receive reports of strange caching behavior that we track down to being caused by incorrect browser settings rather than a problem in Domino or in the customer's application.



And of course a user may be going through one or more proxy servers that do their own caching, again with individual idiosyncrasies. From personal experience, I can assure you that trying to devise a caching scheme that will work identically for all browsers under all configurations is impossible! You should always err on the side of conservatism; it's better to not cache something than to risk showing the user obsolete data. Therefore, you should only override Domino's default behaviors if you are really sure what you are doing.

But assuming that you *are* confident in what you are doing, let's look at how you can use the HTTP response header rule to improve browser caching.

Creating a response header rule

We thoroughly covered the basics of Web site rule documents in the [previous article](#). To review quickly, Web site rules are response documents to Web Sites defined in the Server\Internet Sites view of the Domino Directory. To create a Web site rule for a site, open the Site document, click the Web Site action button, and choose Create Rule. This opens an empty Web Site Rule document. Change the Type of rule field to "HTTP response headers" and the form will look like this:

Web Site Rule			
Basics Administration			
Basics			
Description:			
Type of rule:	HTTP response headers		
Incoming URL pattern:			
HTTP response codes:	200, 206		
Expires header:	<input checked="" type="radio"/> Don't add header <input type="radio"/> Add header only if application did not <input type="radio"/> Always add header (override application's header)		
Custom headers:	Name:	Value:	<input type="checkbox"/> Override
	Name:	Value:	<input type="checkbox"/> Override
	Name:	Value:	<input type="checkbox"/> Override

In the Description field, you can enter any descriptive text that identifies or explains the rule. In the Incoming URL pattern field, you specify a template to match against URLs. For response header rules, the pattern is matched against the final form of a URL, after substitution and redirection rules have been applied to it. For example, if you have a substitution rule that transforms /help/* to /support.nsf/helpview/* and you want to create a response rule to match the response, the pattern for the response rule should be /support.nsf/helpview/*.

The pattern can include one or more asterisks as wildcard characters. For example, the pattern /*/catalog/*.htm will match the URLs /petstore/catalog/food.htm, /clothing/catalog/thumbnails.htm, and so on. A wildcard is not required in a response rule. This allows you to create a rule that matches a specific resource, for example, /cgi-bin/account.pl. Also, as with all rules, the incoming pattern cannot contain a query string.

Response header rules are different from other rules in that not only do they have to match a URL pattern, they also have to match the HTTP response status code. You need to specify one or more status codes in the HTTP response codes field. For example, the status code on the following response is 200, which means that the request succeeded:

```
HTTP/1.1 200 OK
Server: Lotus-Domino
Date: Fri, 15 Mar 2002 14:25:30 GMT
Content-Type: text/html
etc.
```

The full list of possible status codes are defined in RFC 2616. The Domino Web server returns a subset of these codes, the most common of which are:

```
200: OK (the request succeeded)
206: Partial content (successful response to a byte-range request)
302: Found (external redirection)
304: Not modified (page hasn't changed)
400: Bad request (many possible errors)
401: Unauthorized
403: Forbidden
404: Not found
405: Method not allowed (WebDAV isn't enabled)
500: Internal server error (error doesn't match any other code)
```

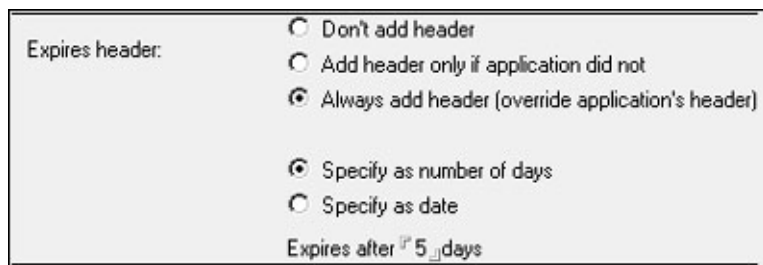
The next set of fields allows you to control the Expires header. You have three options:

- Don't add header
Choose this option if you created this rule only for the purpose of adding custom headers (explained below). Note, however, that the response might still include an Expires header generated by Domino as explained above, or by an application, if the request invoked a CGI program or servlet.
- Add header only if application did not
If this option is selected, the HTTP task will check to see if the response already includes an Expires header; if it doesn't, the task will add one based on your definition in this rule.
- Always add header
The HTTP task will always add an Expires header to the response. If the response already has a header, it will be overwritten.

If you select either of the last two options, another set of options appears that allows you to specify the future date for the header value. The time part of the header is always set to one minute before midnight (23:59:59 GMT). You have two options to specify the date: as a number of days in the future or as a specific date.

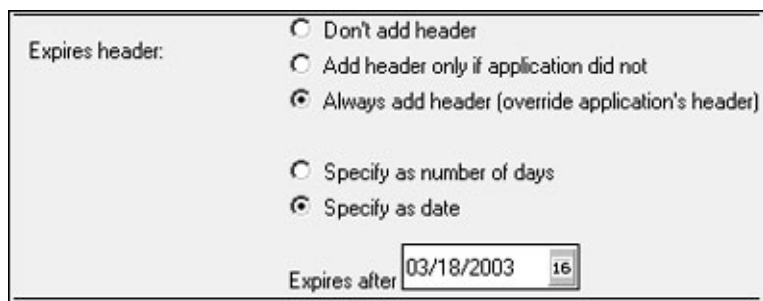
The first option is "Specify as number of days." The Expires date value will be set to the number of days beyond the current system date. For example, if you specify 5 days (as shown in the screen below), then a response that is generated on Monday, June 3, 2002 will get the following Expires header:

```
Expires: Sat, 8 Jun 2002 23:59:59 GMT
```



The second option is "Specify as date." If you choose this option, a calendar control appears to allow you to pick a specific date (as shown in the screen below). In this example, all matching responses will get the following Expires header:

Expires: Tue, 18 Mar 2003 23:59:59 GMT



An Expires header example

Let's look at a typical scenario that would benefit from using response header rules to set the Expires header. In this example, the application database shopping.nsf includes image resources that rarely change. In addition, the images are used for aesthetic reasons and aren't critical to the functionality of the application, so it's not a disaster if a user sees a stale image. So for this case, you can feel comfortable creating a rule that always adds an Expires header set to ten days in the future:

Description:	Product catalog images
Type of rule:	HTTP response headers
Incoming URL pattern:	/shopping.nsf/*.gif*
HTTP response codes:	200, 206
Expires header:	Always add header Specify as number of days Expires after 10 days

Setting custom headers

The bottom part of the Web Site Rule document for response header rules, the Custom header section, allows you to define up to three additional HTTP headers of your choice. For each header, you specify its name (without the trailing colon), its value, and whether or not it should override any existing header of the same name. One common use for these fields is to specify the other cache headers we discussed earlier: Last-Modified, Cache-Control, and Pragma. Let's look at two examples.

In the first example, we will revisit the graphic images scenario. As we mentioned above, the only options Domino uses for the Cache-Control header are "no-cache" and "private." RFC 2616 defines another option, "public," which indicates that the response can be placed in a cache shared by all users (the opposite of "private"). Most browsers and caching proxies should assume "public" to be the default. However, if you want to make very sure that the response is placed in a shared cache, you can add the Cache-Control header yourself. Here is how the complete rule looks in the Web Site Rule document.

Web Site Rule	
<div> <div>Basics</div> <div>Administration</div> </div>	
Basics	
Description:	<input type="text" value="Product catalog images"/>
Type of rule:	<input type="text" value="HTTP response headers"/>
Incoming URL pattern:	<input type="text" value="/shopping.nsf/*.gif"/>
HTTP response codes:	<input type="text" value="200, 206"/>
Expires header:	<input type="radio"/> Don't add header <input type="radio"/> Add header only if application did not <input checked="" type="radio"/> Always add header (override application's header)
	<input checked="" type="radio"/> Specify as number of days <input type="radio"/> Specify as date
	Expires after <input type="text" value="10"/> days
Custom headers:	<div> Name: <input type="text" value="Cache-Control"/> Value: <input type="text" value="public"/> <input type="checkbox"/> Override </div> <div> Name: <input type="text"/> Value: <input type="text"/> <input type="checkbox"/> Override </div> <div> Name: <input type="text"/> Value: <input type="text"/> <input type="checkbox"/> Override </div>

You can see that the Custom headers section allows you to add the new Cache-Control header name and value to the options you've already set for the Expires header.

For the second example, let's consider a scenario where you *don't* want the responses to be cached. Suppose you are running a servlet application for your sales force that allows them to access customer account information. You obtained this servlet from an outside vendor, and you are concerned that it may not always set the HTTP headers correctly. To insure that your sales people always get the latest data, you decide to create a rule that will override the servlet's headers with values that you know will prevent caching:

Description:	Customer account servlet
Type of rule:	HTTP response headers
Incoming URL pattern:	/servlet/salesaction*
HTTP response codes:	200, 206
Expires header:	Always add header Specify as date Expires after 01/01/90
Custom headers:	Name: Cache-Control Value: no-cache Override: Yes Name: Pragma Value: no-cache Override: Yes Name: Last-Modified Value: Fri, 01 Jan 1990 00:00:01 GMT Override: Yes

The rule includes both a Cache-Control: no-cache *and* a Pragma: no-cache header to cover both HTTP/1.1 and HTTP/1.0 caching mechanisms that the response might pass through. Setting the Last-Modified and Expires headers to dates far in the past helps insure that even if a browser or proxy caches a response, the cached version will be recognized as out-of-date when it is requested again.

Response rules aren't just for caching

So far, we've only talked about response rules in the context of browser caching. However, response rules can be useful in other situations, too. For example, RFC 2616 allows HTTP application writers to define their own headers. Suppose you have written Java applets that run in a user's browser and act as the interface to your Web applications. For these applets, you have defined the new response header "Error-Message," which tells the applet the text to display when an error is encountered. You could then define the following rules so that the

applets display specific messages when a user is not authorized to access an application (in these cases the server returns the HTTP response code 401 Unauthorized):

Description:	Unauthorized message for budget application
Type of rule:	HTTP response headers
Incoming URL pattern:	/budget.nsf*
HTTP response codes:	401
Custom headers:	Name: Error-Message Value: You are not authorized to access the budget application. Please contact the accounting department help desk.

Description:	Unauthorized message for personnel application
Type of rule:	HTTP response headers
Incoming URL pattern:	/personnel.nsf*
HTTP response codes:	401
Custom headers:	Name: Error-Message Value: You are not authorized to access the personnel application. Please send an email to Roger Jones in Human Resources.

Using @SetHTTPHeader

For Domino database requests, there is another way for an application designer to set HTTP response headers: the formula function `@SetHTTPHeader`, which is also new in Domino 6. For example, if you create a computed field on a form with this formula:

```
@SetHTTPHeader("Cache-Control";"no-store")
```

the HTTP task will add the Cache-Control header to responses to requests that use the form, such as `?OpenForm` and `?OpenDocument` URLs. The advantage of `@SetHTTPHeader` over the HTTP response header rule is that it does not require a configuration change to the Domino Directory; application designers often do not have access rights to modify the Directory and therefore, have to enlist the help of a site administrator to get rules created. However, `@SetHTTPHeader` can only be used in cases where Domino formulas are evaluated, whereas rules can be used for all parts of an application.

The [Domino Designer 6 Help](#) contains information about `@SetHTTPHeader` and its read-only counterpart `@GetHTTPHeader`.

Conclusion

With this article about the HTTP response header rule, we've wrapped up our discussion of Domino 6 Web site rules. Be sure to review the previous articles in this series to gain a complete understanding of the other rule types. A firm knowledge of all the rule types will allow you to provide better administration, user navigation, and performance of your Web applications and to get the most out of the Domino 6 Web site interface.

ABOUT THE AUTHOR

John Chamberlain is a Senior Software Engineer on the Domino 6 Web Server team. He has been on the team since its early days, and recently celebrated his fifteenth anniversary as a Lotus employee.