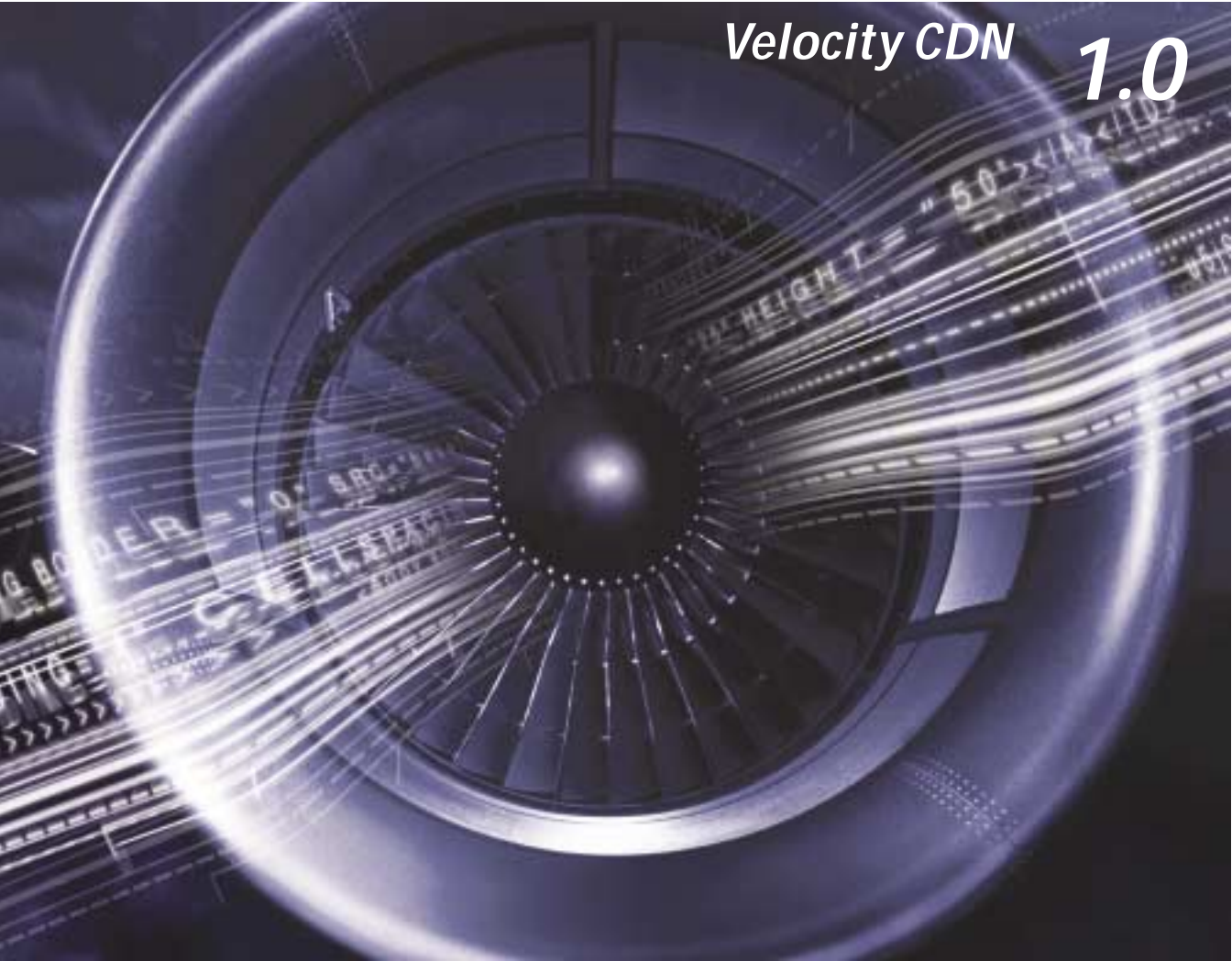




Velocity CDN **1.0**



*Managing Content
with Content Controller*

Legal Notices

Volera, Inc. makes no representations or warranties with respect to the contents or use of this documentation, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Volera, Inc. reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.

Further, Volera, Inc. makes no representations or warranties with respect to any software, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Volera, Inc. reserves the right to make changes to any and all parts of Volera software, at any time, without any obligation to notify any person or entity of such changes.

This product may require export authorization from the U.S. Department of Commerce prior to exporting from the U.S. or Canada.

Copyright © 1997-2001 Volera, Inc. All rights reserved. No part of this publication may be reproduced, photocopied, stored on a retrieval system, or transmitted without the express written consent of the publisher.

U.S. Patent Nos. 5,870,739; 5,873,079; 5,884,304. Patents pending.

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>).

Volera, Inc.
2211 North First Street
San Jose, CA 95131-2021
U.S.A.

www.volera.com

Managing CDN Content with Content Controller
August 2001

Online Documentation: To access the online documentation for this and other Volera products, and to get updates, see www.volera.com.

Volera Trademarks

Volera is a trademark of Volera, Inc. in the United States and other countries.

Third-Party Trademarks

All third-party trademarks are the property of their respective owners.

Contents

	About this Guide	7
1	Content Controller Overview	9
	How Content Controller Works	9
	Installing Content Controller	10
	Preparing to Use Content Controller	10
	Modifying CC Server's Configuration Settings	10
2	Creating Content Collections and Jobs	11
	Avoiding Job Configuration Conflicts.	11
	Creating Content Collections.	11
	Creating and Executing Content Management Jobs.	12
3	Prepopulating, Retaining, and Removing Cache Objects	13
	Defining Which End-User-Requested Content Will Be Cached	13
	Prepopulating Caches with Specific Content.	13
	Retaining Content in Cache (Pinning)	13
	How Exceleator Processes URL Masks	14
	About Wildcards in Pin Lists.	17
	Pin List Examples.	17
	Exceleator Records IP Addresses When Resolving URL Masks	20
	Removing Unwanted Content From Cache	21
4	Ensuring Cache Freshness	23
	Managing Cache Freshness	24
	How the Exceleator Appliance Checks for Object Freshness	24
	How an Exceleator Appliance Keeps the Oldest Cached Objects Fresh	24
	How Exceleator Handles the Freshest Objects in Cache	25
	Fine-Tuning Cache Freshness on Your Cache Device.	26
	Using Custom Cache Control Headers	26
	An Overview of How Headers Work	26
	Implementing Custom Cache Control Headers.	27
	An Implementation Example	28
5	Monitoring Content Delivery	31

About this Guide

This guide explains how Content Controller helps you manage CDN content and covers the topics outlined in the following table.

To	See
Learn about how Content Controller works and how to modify its configuration settings and performance	Chapter 1, “Content Controller Overview,” on page 9
Define content collections and create jobs for storing and managing the collections in your CDNs caches	Chapter 2, “Creating Content Collections and Jobs,” on page 11
Learn how to prepopulate cache, retain objects in cache, and remove objects from cache	Chapter 3, “Prepopulating, Retaining, and Removing Cache Objects,” on page 13
Manage the freshness of cached objects on your CDN	Chapter 4, “Ensuring Cache Freshness,” on page 23
Monitor CDN content delivery	Chapter 5, “Monitoring Content Delivery,” on page 31

1

Content Controller Overview

For a high-level overview of Content Controller functionality, see [Content Controller Overview](#) in *Planning Your Content Distribution Network (CDN)*

How Content Controller Works

Content Controller consists of two components: Content Controller Server and Content Controller Agent.

Content Controller Server runs on a VCDN management server with System Controller installed. Content Controller Server lets you create object collections and jobs to schedule and perform actions against the objects defined in the collections.

Content Controller Agent is an installed component on all Excelsior 2.1 cache devices, but it remains idle until it receives one or more jobs from Content Controller Server.

Content Controller Agent can handle multiple concurrent jobs, each containing multiple, and perhaps overlapping, collection definitions.

Content Controller Server and Content Controller Agent work with each other to identify and remove duplicate object references within a single job to protect against the possibility of double billing through Content Accountant. However, this protection does not cover duplicate object references across multiple jobs. Therefore, collections for trending and billing need to be split into separate jobs to protect against double billing.

Installing Content Controller

Instructions for installing Content Controller are provided in *Getting Started with Content Controller*.

Preparing to Use Content Controller

Content Controller functionality is available in the VCDN browser-based management tool. For information on using this tool, see [Using the VCDN Management Tool](#) in *Creating and Managing Your CDN Using Velocity CDN*.

Modifying CC Server's Configuration Settings

All configuration settings are stored in the `vcdn.conf` file. If configuration changes are required due to an IP address or other change, you can modify the file as explained in [Understanding the vcdn.conf File](#) in *Advanced Topics*.

2

Creating Content Collections and Jobs

The power of Velocity CDN™ content management comes from creating content collections and grouping the collections into content jobs.

Avoiding Job Configuration Conflicts

As you develop a content distribution strategy for your CDN, it is vital that you approach collection and job creation methodically. Otherwise, your CDN implementation will not run as efficiently as it might, nor will it provide the service that your customers need and expect.

Additionally, you could duplicate content requests in multiple collections. This could, in turn, cause double-billing problems.

Creating Content Collections

To create a content collection, complete the following steps:

- 1 Name the collection.

A collection name must include from one to 16 alphanumeric characters. The name might give an indication as to the content of a collection or the target end users for collection content.

- 2 Specify the URL masks for the collection's objects.

For more information, see [“How Excelerator Processes URL Masks” on page 14](#).

You can import a list from an ASCII file. Once you save a collection, it is easier to work with the ASCII file than to edit the collection in VCDN.

Lists can be quite lengthy. The content in collections is not overwritten by importing a file. You can check URLs before importing. You can select the URLs, delete them, and reimport them.

3 Set monitoring and reporting frequency:

The statistics will become available after twice the reporting frequency. It waits the first period before starting to collect, then the next period to collect the data before reporting it.

3a Enable monitoring of the collection

3b Select the statistical reporting frequency

3c Set Maximum Concurrent Connections. The default value is 200.

4 Specify the monitoring host

5 Review the configuration and make any required changes.

6 Click Save and Exit.

Creating and Executing Content Management Jobs

To create and execute a content management job, complete the following steps:

- 1** Name the job, enter a description, and other notes.
- 2** Include collections in the job by selecting them.
- 3** Assign a Job Action.
- 4** Set the schedule.
- 5** Select the distribution groups to which the job will be sent.
- 6** View a summary.
- 7** Save the job.

3

Prepopulating, Retaining, and Removing Cache Objects

Defining Which End-User-Requested Content Will Be Cached

Velocity CDN™ uses URL masks to specify the objects it manages. These masks are explained in [“How Excelerator Processes URL Masks”](#) on page 14.

Prepopulating Caches with Specific Content

When you create content jobs, you can specify that the objects in its collections will be prepopulated in cache.

Retaining Content in Cache (Pinning)

Excelerator cache devices use a variety of metrics to remove or purge content and free up disk space. Sometimes this results in vital content being purged because it has been posted for longer than the maximum time, or it hasn't been requested a minimum number of times.

Content Controller lets you override the usual purge routines for content that you need to keep in cache. For example, some content demands high availability for quality of service reasons, even though it is accessed less frequently than other content in cache.

By using a combination of pinning and the principles explained in [Chapter 4, “Ensuring Cache Freshness,”](#) on page 23, you can guarantee that important content stays put and stays fresh.

How Excelerator Processes URL Masks

There are four basic types of URL masks you can enter in the pin list. The following table lists each type, provides a few examples of each, and provides information on how they are processed by cache devices.

Type	URL Mask Examples by Specificity	Notes
Hostname	<p>http://www.foo.gov/documents/picture.gif</p> <p>http://www.foo.gov/documents/</p> <p>http://www.foo.gov/</p> <p>foo.gov/documents/</p> <p>foo.gov/</p> <p>*.foo.gov/</p>	<p>Although these entries can include the protocol or scheme, the DNS name, the path, and the filename, only the DNS or hostname must be present in the mask. All DNS label portions must be indicated, if only by an asterisk wildcard.</p> <p>Exceclerator processes hostname entries before it processes other mask types. It also processes the most specific URL mask entries first.</p> <p>When an object match occurs, Exceclerator applies the pin type rule, and processing of the object is finished.</p> <p>For example, if the first URL mask in the examples column has a pin type rule of bypass, picture.gif will not be cached regardless of the pin type rules for the other URL masks.</p> <p>Hostname entries can have a dramatic impact on object pinning and cache bypassing.</p> <p>For example, if the first two URL masks in the examples column were not present, a pin type of Bypass on the third URL mask would prevent caching of all objects delivered through HTTP on the www.foo.gov Web site.</p> <p>If no scheme (HTTP, FTP, etc.) is indicated, the mask applies to all schemes. The last three masks would apply to objects delivered through any Web protocol.</p> <p>Finally, Exceclerator interprets hostnames literally. For example, the sixth entry would cover www.foo.gov, ww1.foo.gov, army.foo.gov, etc., but the fourth and fifth entries would not, because a scheme is assumed to immediately precede the hostname.</p>

Type	URL Mask Examples by Specificity	Notes
Path	<p>/documents/picture.gif</p> <p>/documents/picture.gif/</p> <p>/documents/</p>	<p>Exceleator processes path entries after all hostname entries have been considered. It assumes that the first forward slash immediately follows a hostname.</p> <p>For example, the first entry would apply only to a graphics file named PICTURE.GIF that is located in a DOCUMENTS directory at the root of the host.</p> <p>The forward slash in the second entry causes Exceleator to assume that PICTURE.GIF is a directory. The pin type rules associated with this entry would apply to any matched objects that have a URL directory path that starts with a documents directory followed by a subdirectory named PICTURE.GIF.</p> <p>The third entry would apply to any matched objects that contain a DOCUMENTS directory at the start of their URL paths.</p>
Filename	<p>/picture.gif</p> <p>/widget.js</p> <p>/default.htm</p>	<p>After the path entries have all been processed, Exceleator looks for specific filenames.</p> <p>For example, if requested objects named PICTURE.GIF, WIDGET.JS, and DEFAULT.HTM have not been covered by one of the hostname or path entries above, the files will have the pin type rule for their respective filename mask applied to them.</p> <p>If the first entry carries a pin type rule of Bypass, all PICTURE.GIF files that didn't match previously processed hostname or path masks would not be cached.</p>

Type	URL Mask Examples by Specificity	Notes
File Extension	/*_gif /*_js /*_htm	<p>File extension entries are processed last.</p> <p>These are simply filename entries with the root of the filename replaced by an asterisk, which makes them less specific than complete filenames.</p> <p>For example, if the examples shown all had pin types of Bypass, then only those .GIF, .JS, and .HTM files that had been cached and pinned because of hostname, path, or filename masks would be stored in cache. All other files with the named extensions would not be cached.</p>

About Wildcards in Pin Lists.

Only the asterisk (*) wildcard is allowed in pin list entries.

Excelerator interprets everything between an asterisk and the next delimiter to the right (a forward slash [/], a period[.], or a colon[:]) as a wildcard. This effectively allows only one asterisk between delimiters.

Pin List Examples

The following table provides brief examples of sample pin list entries and their effects on cache device caching.

URL Mask	Pin Type	Pin Links	Pin Images	Effect on Cache
http://www.foo.gov/documents/	cache	1	Yes	<p>As a general rule, you should always include fully qualified DNS or hostnames in the pin list. Exceleator resolves these more quickly than other masks, and you will be able to track the effects on pinning more easily.</p> <p>For this URL mask, Exceleator downloads, caches, and pins all objects whose URL starts with the mask. In other words, all objects below the documents directory will be downloaded, cached, and pinned. Also, all objects that are linked from one of the pinned objects will be downloaded, cached, and pinned. And finally, images that reside on other hosts will be downloaded, cached, and pinned as well.</p> <p>Objects will be refreshed according to the refresh settings (default or specific) as specified in the pin list entry.</p>
www.foo.gov/groups.html	cache	1	No	<p>Exceleator downloads, caches, and pins objects (including images) in the GROUPS.HTML page and in pages linked from that page. Any images referenced from other hosts, however, are not included.</p>

URL Mask	Pin Type	Pin Links	Pin Images	Effect on Cache
www.foo.gov/groups.html/	normal	1	Yes	<p>Excelerator downloads and caches objects in the subdirectory named groups.html and in pages linked from any of those objects.</p> <p>The forward slash tells Excelerator that this is a directory rather than a file.</p> <p>Objects are cached but not pinned in cache, meaning they might be bumped by more frequently accessed objects or objects that are pinned.</p> <p>Images linked from other hosts are downloaded and cached.</p>
www.foo.*	bypass	n/a	n/a	<p>Excelerator doesn't cache objects from any URLs whose DNS names begin with www.foo.</p> <p>All domain extensions (.com, .net, .org, etc.) are covered by the asterisk wildcard.</p> <p>Link and image pinning is not available for bypass pin types.</p> <p>If this entry appeared in a pin list with either of the previous two entries, it would not prevent caching of objects covered by them since it is less specific than they are.</p>
w*.f*.com	bypass	n/a	n/a	<p>Excelerator doesn't cache objects for any URLs whose first domain label begins with w and second domain label begins with f, providing the domain extension is .com.</p> <p>This mask doesn't prevent caching of objects on other domains such as .net, .gov, etc.</p>
w*.f*.*	bypass	n/a	n/a	<p>This mask functions like the previous entry, but the domain is not limited to .com.</p>

URL Mask	Pin Type	Pin Links	Pin Images	Effect on Cache
.foo.	cache	n/a	n/a	<p>This causes all objects on any Web server whose second domain label is foo to be pinned in cache.</p> <p>Link and image pinning are not available because the mask contains asterisks.</p> <p>This mask would not cover DNS names that don't have a domain label before foo. For example, foo.gov would not normally be covered. However, if foo.gov happened to resolve in DNS to the same IP address as www.foo.gov, Excelerator would apply the pinning rules specified for www.foo.gov to foo.gov. To understand more about IP addresses and URL masks, see "Excelerator Records IP Addresses When Resolving URL Masks" on page 20.</p>

Excelerator Records IP Addresses When Resolving URL Masks

As stated earlier, you should include fully qualified DNS or hostnames in URL masks whenever possible.

Excelerator resolves DNS names to their respective IP addresses and uses those addresses when pinning objects.

You can use this fact when constructing your pin list entries.

For example, if you use the DNS name www.foo.gov as the URL mask and you know that the DNS name foo.gov resolves to the same IP address, you don't need to include foo.gov in the pin list.

Since both URLs resolve to the same IP address, Excelerator will treat objects for both DNS names the same.

On the other hand, if www.foo.gov and foo.gov resolve to different IP addresses, separate pin list entries would be required to cover both sites.

Removing Unwanted Content From Cache

Just as it is vital to be able to pin some content in cache, it is essential to be able to purge some or all content from a caching server or a group of caching servers. The purge feature provides this service.

4

Ensuring Cache Freshness

When first introduced to Web content caching, many network administrators assume that the object cache on an Excelerator appliance is basically the same as a browser's cache, which all users access when they click the Back button. The logical extension from this assumption is the fear that Excelerator will serve stale content that doesn't accurately reflect the fresh content on the origin Web server.

Actually, most time-sensitive Web content is flagged by Webmasters in such a way that it cannot become stale unless a caching system ignores the Webmaster's settings. And in fact, the Excelerator appliance honors all flags that affect cache freshness, including Time to Expire, Don't Cache, and Must Revalidate directives.

In addition, the Excelerator appliance can be fine-tuned for cache freshness in the following ways:

- ◆ Accelerated checking of objects that have longer than desirable Time to Expire headers.
- ◆ Delayed checking of objects that have shorter than desirable Time to Expire headers.
- ◆ Checking objects for freshness that do not include Time to Expire headers.

For more information on configuring Excelerator for cache freshness, see [“Managing Cache Freshness” on page 24](#).

To	See
Understand VCDN cache freshness features and how to use them	“Managing Cache Freshness” on page 24

Transmit special caching instructions from your Web servers to your CDN caches through customized HTTP header information

[“Using Custom Cache Control Headers” on page 26](#)

Managing Cache Freshness

Cache freshness is a primary concern of most VCDN administrators. The following sections briefly explain how your cache device ensures fresh content for network users and the options you have for adjusting this feature.

How the Excelerator Appliance Checks for Object Freshness

Although the following explanation is an over-simplification, it lays the foundation for the specific examples that follow this section.

An Excelerator appliance has timers that it applies to every cached object.

Each time an object is cached or revalidated, the cache device starts a timer for that object. As long as the timer is running, the cache device will vend the object from cache. After the time has expired and when the cache device receives a request for the object, it will issue an IF-MODIFIED-SINCE request to the origin Web server.

If the object has changed, Excelerator retrieves the updated object into cache and serves it to the requesting browser before restarting the timer.

If the object has not changed, the cache device vends the object from cache and resets the timer, and the countdown for vending the object from cache begins again.

If a browser forces a refresh of the object, Excelerator honors the browser request, retrieves and caches the object regardless of whether it has changed, and restarts the timer.

How an Excelerator Appliance Keeps the Oldest Cached Objects Fresh

More than 80% of all Web objects have either no Time to Expire directives or they are set to stay cached for as long as weeks or even months.

Since many of these objects actually change fairly frequently, the cache device has two timers for ensuring their freshness. You can configure these timers in System Controller.

HTTP Maximum: This timer overrides an object's Time to Expire settings if it is longer than the timer's value.

The default timer value is six hours. This means that Excelerator will not vend an object that has been in cache longer than six hours without first checking whether it should be refreshed.

HTTP Default: Excelerator applies this timer to objects that don't have Time to Expire settings.

The default timer value is two hours. This means that Excelerator will not vend an object that has no Time to Expire setting that has been in cache longer than two hours without first checking whether it should be refreshed.

How Excelerator Handles the Freshest Objects in Cache

Most Webmasters ensure that their time-sensitive objects have appropriate Time to Expire directives. Late-breaking news stories and photographs, for example, might stay in cache for only a few minutes before expiring.

By default, the Excelerator appliance simply honors the Webmasters' instructions and revalidates the objects in cache as directed.

However, some cache device installations, such as those connected through a modem might need to limit how often these objects are refreshed. The cache device has a third timer for this purpose, also accessible in System Controller.

HTTP Minimum: This timer sets the minimum number of hours or minutes Excelerator will serve HTTP data from cache before revalidating it against content on the origin Web server. No requested object will be revalidated sooner than specified by this value.

The default value for this timer is 0, meaning that Excelerator honors the Time to Expire directive for each object (assuming, of course, it is not longer than the HTTP Maximum timer).

If the timer is set to a value other than 0, it then overrides any object's Time to Expire directive that is shorter than the value set.

Fine-Tuning Cache Freshness on Your Cache Device

The default timer settings explained in the previous sections are tuned for most cache device installations. However, as you read about them you might think of special requirements you have that could be met better if the default settings were adjusted.

Perhaps you are accelerating content that doesn't contain Time to Expire directives but changes frequently and needs to be refreshed more often than every two hours. You can adjust the HTTP Default timer in System Controller so that Accelerator refreshes the objects more frequently.

Perhaps you have severe Internet bandwidth restrictions and an environment with users who don't require object freshness checks every six hours. You can adjust the HTTP Maximum timer in System Controller to a different setting that meets your requirements and conserves bandwidth.

If you choose to adjust the timer values, avoid settings that result in object refreshing more often than is necessary. Otherwise you could easily negate the bandwidth and response-time benefits of having the cache device on your network.

Using Custom Cache Control Headers

In addition to fine tuning cache freshness using the system's global HTTP timers as explained in [Fine-Tuning Cache Freshness on Your Cache Device](#) above, you can configure each proxy service to recognize custom headers in HTTP packets. Your Web server can then use these headers for transmitting caching instructions that only the configured cache devices will recognize and follow.

An Overview of How Headers Work

Only the accelerator service containing the custom header definition follows the cache policies specified in the custom headers.

All other caches, including the non-configured caches, requesting browsers, and external proxy caches (transparent caches, client accelerators, etc.) do not recognize the custom headers and therefore follow only the cache policies specified by the standard cache control headers.

This means that you have the following options for configuring your Web server:

- ◆ You can specify that browsers and/or external caches cannot cache the objects, but the accelerator can.

This lets you offload request-processing from the origin Web server while still requiring that users return to the site each time they request an object.

- ◆ You can also specify separate cache times for browsers, external caches, and the accelerator you are defining.

Implementing Custom Cache Control Headers

To implement custom cache control headers, you must do the following:

- ◆ Enter a header string in the Custom Cache Control Header dialog box, for example, MYCACHE.
- ◆ Configure the Web server to send an HTTP header containing the defined string and the time in seconds that the object should be retained in cache, for example, MYCACHE: 60.

If the number is non-zero, Excelerator treats the reply as if it had the following headers:

```
Cache-Control: public  
Cache-Control: max-age=number
```

If the number is zero (0), Excelerator treats the reply as if it had the following header:

```
Cache-Control: no-cache
```

- ◆ Ensure that the Web server continues to send standard HTTP cache-control headers so that browsers and external caches follow the caching policies you intend them to.

For example, you could do the following:

- ◆ Use an Expires or Cache-Control: Max-Age header to specify that browsers should cache an object for two minutes
- ◆ Use a Cache-Control: Private header to prevent external caches from caching the object at all
- ◆ Use a custom cache control header, such as MYCACHE: 1800, to indicate that the accelerator should cache the object for 30 minutes.

Custom Cache Control Headers override the following standard HTTP cache-control headers on the cache device but do not affect how browsers and external caches respond to them:

```
Cache-Control: no-store
Cache-Control: no-cache
Cache-Control: max-age=number
Cache-Control: private
Cache-Control: public
Pragma: no-cache
Expires: date
```

An Implementation Example

For example, you might do the following:

1. While configuring a Web server accelerator service, you insert a string in the Custom Cache Control Header list with the value of FOOTTL.

The cache device will now recognize FOOTTL as a custom cache control header on objects requested through the service you are configuring.

2. You then configure the accelerated Web server to send FOOTTL: 600 in the headers of objects you want to be cached at the cache device.

The cache device will recognize this header as overriding the standard HTTP cache-control headers listed above when objects are requested through the accelerator service you are configuring.

3. Finally, you ensure that the Web server continues to send the following standard HTTP cache-control headers:
 - ◆ Cache-Control: Max-Age headers that cause browsers to cache objects for no longer than two minutes
 - ◆ Cache-Control: Private headers that cause external caches to not cache the objects.

When your Web server sends an object with the FOOTTL header in response to a cache device request made through the accelerator service, your cache device recognizes the custom header and caches the object for 10 minutes. Requesting browsers cache the object for only two minutes. And external caches do not cache the object.

Thus, the cache device off loads a processing burden from the Web server by caching the frequently requested objects for 10 minutes (the value you specified in Step 2). Browsers, on the other hand, must always access the cache device to get the objects if their previous requests are older than two

minutes. And the objects in the cache device's cache are kept fresh due to their relatively brief time-to-live value.

5

Monitoring Content Delivery

In addition to monitoring the effectiveness and status of caching servers and groups, you will usually want to monitor the status of specific content as well.

The monitoring functionality in Content Controller lets you determine the content delivery performance for a content collection, how long certain content should remain in the cache, details of who is accessing the content, when they are accessing it and from what location, and other critical business decisions.

