Cisco Reader Comment Card

General Information

1 Years of networking experience: ______ Years of experience with Cisco products: ______

2 I have these network types: ☐ LAN ☐ Backbone ☐ WAN
☐ Other: ______

3 I have these Cisco products: ☐ Switches ☐ Routers
☐ Other (specify models): ______

4 I perform these types of tasks: ☐ H/W installation and/or maintenance ☐ S/W configuration
☐ Network management ☐ Other: ______

5 I use these types of documentation: ☐ H/W installation ☐ H/W configuration ☐ S/W configuration
☐ Command reference ☐ Quick reference ☐ Release notes ☐ Online help
☐ Other: ______

6 I access this information through: ____% Cisco.com (CCO) ____% CD-ROM
____% Printed docs ____% Other: ______

7 I prefer this access method: ______

8 I use the following three product features the most: ______

______

______

Document Information

Document Title: Cisco Internet CDN Software Content Provider Guide

Part Number: 78-13749-01 S/W Release (if applicable):

On a scale of 1–5 (5 being the best), please let us know how we rate in the following areas:

______ The document is written at my technical level of understanding.	______ The information is accurate.
______ The document is complete.	______ The information I wanted was easy to find.
______ The information is well organized.	______ The information I found was useful to my job.

Please comment on our lowest scores: ______

______

______

______

Mailing Information

Company Name		Date
Contact Name	Job Title	
Mailing Address		
City	State/Province	ZIP/Postal Code
Country	Phone ( )	Extension
Fax ( )	E-mail	

Can we contact you further concerning our documentation? ☐ Yes ☐ No

You can also send us your comments by e-mail to **bug-doc@cisco.com**, or by fax to **408-527-8089**.

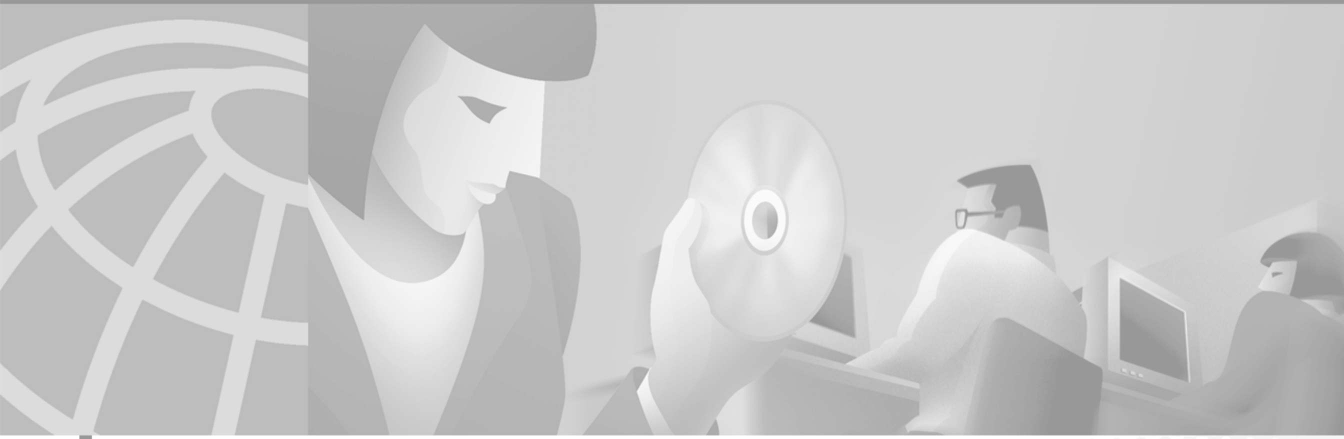
NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 4631 SAN JOSE CA

POSTAGE WILL BE PAID BY ADDRESSEE

ATTN DOCUMENT RESOURCE CONNECTION
CISCO SYSTEMS INC
170 WEST TASMAN DRIVE
SAN JOSE CA 95134-9883

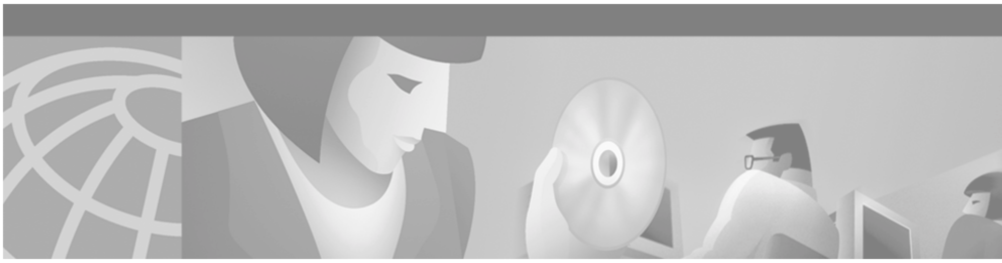


Cisco Internet CDN Software Content Provider Guide

Corporate Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 526-4100

Customer Order Number: DOC-7813749
Text Part Number: 78-13749-01

CONTENTS



Preface

This preface contains the following sections:

Document Objectives

This guide contains information that enables content providers to prepare their website content for deployment on a Cisco Internet Content Delivery Network (CDN). The Cisco Internet CDN is a collection of hardware devices and proprietary software that, together, significantly improves the delivery of web content to users of the Internet.

This document focuses on the role and responsibilities of the content provider organization, which has live and static content that it wants to deploy closer to users on the Internet using the CDN. The mechanics of content routing are explained, and clear steps for preparing your website for CDN deployment are provided. In addition, tips are given for troubleshooting content deployment issues.

Audience

This guide assumes you are a content provider who is coordinating with a service provider organization to deploy content on a Cisco Internet CDN. It is assumed that all CDN hardware and resources are owned and operated by your service provider, not by you. If this is not the case, refer to the *Cisco Internet CDN Software Configuration Guide and User Guide* for information on configuring CDN devices and using the CDN administrative user interface.

You should be familiar with basic TCP/IP and networking concepts and your enterprise's network topology.

Document Organization

This guide contains the following major sections:

Chapter	Title	Description
Chapter 1	Understanding CDNs	Contains a technical background of the Cisco Internet CDN product including a discussion of CDN hardware and software, content routing mechanics, and system architecture.
Chapter 2	Understanding the Content Provider Role	Contains information on the role and responsibilities of the content provider in a Cisco Internet CDN deployment.
Chapter 3	Deploying Web Site Content on an Internet CDN	Contains technical information and instructions on preparing and deploying website content on a Cisco Internet CDN.
Appendix A	Sample Manifest File Scripts	Contains detailed syntax information and code from the manifest generation scripts that are supplied with the CDN software.
Appendix B	CDN Supported Time Zones	Contains a list of timezone abbreviations for all time zones supported by the CDN software. Timezone designations are important when deploying content over a geographically dispersed CDN.

Document Conventions

This guide uses basic conventions to represent text and table information.

Convention	Description
boldface font	Commands, keywords, and button names are in **boldface**.
italic font	Variables for which you supply values are in *italics*. Directory names and filenames are also in italics.
`screen` font	Terminal sessions and information the system displays are printed in `screen` font.
`boldface screen` font	Information you must enter is in **`boldface screen`** font.
`italic screen` font	Variables you enter are printed in *`italic screen`* font.
Vertical bars (|)	Vertical bars separate alternative, mutually exclusive, elements.

Note Means *reader take note*. Notes contain helpful suggestions of references to materials not contained in this manual.

Obtaining Documentation

The following sections explain how to obtain documentation from Cisco Systems.

World Wide Web

You can access the most current Cisco documentation on the World Wide Web at the following URL:

http://www.cisco.com

Translated documentation is available at the following URL:

http://www.cisco.com/public/countries_languages.shtml

Documentation CD-ROM

Cisco documentation and additional literature are available in a Cisco Documentation CD-ROM package, which is shipped with your product. The Documentation CD-ROM is updated monthly and may be more current than printed documentation. The CD-ROM package is available as a single unit or through an annual subscription.

Ordering Documentation

Cisco documentation is available in the following ways:

- Registered Cisco Direct Customers can order Cisco product documentation from the Networking Products MarketPlace:

  http://www.cisco.com/cgi-bin/order/order_root.pl

- Registered Cisco.com users can order the Documentation CD-ROM through the online Subscription Store:

  http://www.cisco.com/go/subscription

- Nonregistered Cisco.com users can order documentation through a local account representative by calling Cisco corporate headquarters (California, USA) at 408 526-7208 or, elsewhere in North America, by calling 800 553-NETS (6387).

Documentation Feedback

If you are reading Cisco product documentation on Cisco.com, you can submit technical comments electronically. Click **Leave Feedback** at the bottom of the Cisco Documentation home page. After you complete the form, print it out and fax it to Cisco at 408 527-0730.

You can e-mail your comments to bug-doc@cisco.com.

To submit your comments by mail, use the response card behind the front cover of your document, or write to the following address:

Cisco Systems
Attn: Document Resource Connection
170 West Tasman Drive
San Jose, CA 95134-9883

We appreciate your comments.

Obtaining Technical Assistance

Cisco provides Cisco.com as a starting point for all technical assistance. Customers and partners can obtain documentation, troubleshooting tips, and sample configurations from online tools by using the Cisco Technical Assistance Center (TAC) Web Site. Cisco.com registered users have complete access to the technical support resources on the Cisco TAC Web Site.

Cisco.com

Cisco.com is the foundation of a suite of interactive, networked services that provides immediate, open access to Cisco information, networking solutions, services, programs, and resources at any time, from anywhere in the world.

Cisco.com is a highly integrated Internet application and a powerful, easy-to-use tool that provides a broad range of features and services to help you to

- Streamline business processes and improve productivity
- Resolve technical issues with online support
- Download and test software packages
- Order Cisco learning materials and merchandise
- Register for online skill assessment, training, and certification programs

You can self-register on Cisco.com to obtain customized information and service. To access Cisco.com, go to the following URL:

http://www.cisco.com

Technical Assistance Center

The Cisco TAC is available to all customers who need technical assistance with a Cisco product, technology, or solution. Two types of support are available through the Cisco TAC: the Cisco TAC Web Site and the Cisco TAC Escalation Center.

Inquiries to Cisco TAC are categorized according to the urgency of the issue:

- Priority level 4 (P4)—You need information or assistance concerning Cisco product capabilities, product installation, or basic product configuration.
- Priority level 3 (P3)—Your network performance is degraded. Network functionality is noticeably impaired, but most business operations continue.
- Priority level 2 (P2)—Your production network is severely degraded, affecting significant aspects of business operations. No workaround is available.
- Priority level 1 (P1)—Your production network is down, and a critical impact to business operations will occur if service is not restored quickly. No workaround is available.

Which Cisco TAC resource you choose is based on the priority of the problem and the conditions of service contracts, when applicable.

Cisco TAC Web Site

The Cisco TAC Web Site allows you to resolve P3 and P4 issues yourself, saving both cost and time. The site provides around-the-clock access to online tools, knowledge bases, and software. To access the Cisco TAC Web Site, go to the following URL:

http://www.cisco.com/tac

All customers, partners, and resellers who have a valid Cisco services contract have complete access to the technical support resources on the Cisco TAC Web Site. The Cisco TAC Web Site requires a Cisco.com login ID and password. If you have a valid service contract but do not have a login ID or password, go to the following URL to register:

http://www.cisco.com/register/

If you cannot resolve your technical issues by using the Cisco TAC Web Site, and you are a Cisco.com registered user, you can open a case online by using the TAC Case Open tool at the following URL:

http://www.cisco.com/tac/caseopen

If you have Internet access, it is recommended that you open P3 and P4 cases through the Cisco TAC Web Site.

Cisco TAC Escalation Center

The Cisco TAC Escalation Center addresses issues that are classified as priority level 1 or priority level 2; these classifications are assigned when severe network degradation significantly impacts business operations. When you contact the TAC Escalation Center with a P1 or P2 problem, a Cisco TAC engineer will automatically open a case.

To obtain a directory of toll-free Cisco TAC telephone numbers for your country, go to the following URL:

http://www.cisco.com/warp/public/687/Directory/DirTAC.shtml

Before calling, please check with your network operations center to determine the level of Cisco support services to which your company is entitled; for example, SMARTnet, SMARTnet Onsite, or Network Supported Accounts (NSA). In addition, please have available your service agreement number and your product serial number.

Understanding CDNs

This chapter provides a conceptual background to the Cisco Internet CDN product and contains the following sections:

What a CDN Does

When end users experience technical problems the first time they go to a website, they often lose patience, often moving on to another site that offers similar information and never returning to the original site.

The most basic technical problem that users encounter is slow content delivery. To help solve the problem of slow content delivery, Cisco Internet CDN Software enables your service provider to create and maintain Content Delivery Networks (CDNs) for your website content. By choosing to deploy your website content on a CDN, you offer your end users website content they can quickly and reliably access—even high-impact, high-bandwidth video and multimedia content that has historically been difficult to deliver reliably over the public Internet.

A Cisco Internet CDN is a collection of hardware devices and proprietary software that, together, significantly improves the delivery of web content to users of the Internet

The Cisco Internet CDN allows web content to be distributed to caches at various locations on the Internet and then accessed from those caches. Service providers can ensure better access to their content, because end users are able to obtain it from a cache that is both closer to them (in terms of network distance) and less heavily loaded than the web server where the content originates. In addition, these caches reduce the load on the web server that belongs to the content provider where content originates (the origin server).

See the "Mechanics of Content Routing" section on page 1-2 and the "Distributing Content" section on page 1-5 for more information on how the CDN software improves access to high bandwidth content on the Internet.

Security

The Cisco Internet CDN Software uses Secure Socket Layer (SSL) for Java to encrypt all inter-device communications.

Developed by Netscape, the SSL protocol is supported by both the Netscape and Microsoft browsers and is a widely accepted and deployed encryption technology on the Internet. SSL uses the sockets method of communication between client and server, coupled with RSA Security's public key encryption technology to secure data using digital certificates as it is transmitted between CDN devices over the Internet.

Mechanics of Content Routing

The primary job of the Cisco Internet CDN is to deliver content. With content distributed to up to 2000 geographically dispersed Content Engines, it is vital that client requests for cached content be handled by Content Engines that are suitable for the client. Suitability, for the CDN software, means that Content Engines are online, nearby, and cheap to communicate with. The selected Content Engines must also be authorized to store the requested content (the hosted domain). Thus, the job of the routing subsystem is to choose the most suitable Content Engines to handle a particular client request.

The following sections explain how content is routed on a Cisco Internet CDN and contain information on the following topics:

- End Users Requesting Content, page 1-3
- Routing End User Requests, page 1-3

End Users Requesting Content

When end users attempt to access content on your CDN hosted domain, their client machines communicate with a local domain name system (DNS) proxy.

If the proxy does not have information on how to properly route the hosted domain, it communicates through normal DNS mechanisms with one of the Content Routers that make up the CDN, which begins the routing process.

Routing End User Requests

This section describes how end user requests are processed by the CDN. See the numbered steps in Figure 1-1 as you read.

Content routing happens as part of the DNS lookup that occurs when a client machine requests content from a web page by clicking a URL (1). If the requesting client does not know the IP address associated with a particular DNS name (a hosted domain), it communicates with a DNS proxy (2), which either knows how to route the client request to a Content Engine (because of prior communication with one of the Content Routers) or does not know how to route the request. If it does not know how to route the request, it sends iterative name server (NS) requests to each authoritative server, eventually reaching (3) one of the Content Routers, which are the authoritative DNS servers for the hosted domain.

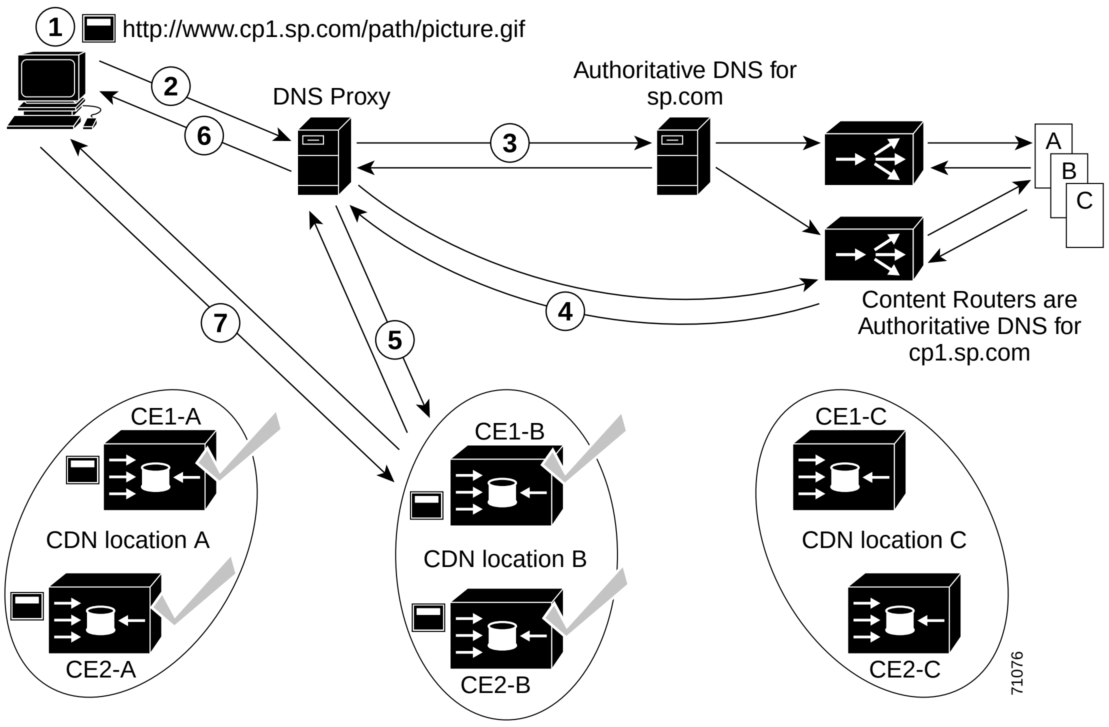
In response, the Content Router consults its proxy tables to find the preferred Content Engines that can serve the content from among those Content Engines that are authorized to serve content for the requested hosted domain. After the Content Router selects suitable Content Engines, it returns NS records (4) for the authorized devices.

Next, the DNS proxy sends an NS request to one of the Content Engines named by the Content Router (5), typically the first one. The Content Engine responds with its own A-record.

Next, the DNS proxy passes (6) the Content Engine A-record, which contain the Content Engine IP address, to the client.

Finally, the client makes its content request directly to the Content Engine (7) identified in the A-record, and the Content Engine serves the content.

Figure 1-1 Cisco Internet CDN Routing


1
http://www.cp1.sp.com/path/picture.gif
2
6
DNS Proxy
3
Authoritative DNS for sp.com
A
B
C
4
5
7
Content Routers are Authoritative DNS for cp1.sp.com
CE1-A
CDN location A
CE2-A
CE1-B
CDN location B
CE2-B
CE1-C
CDN location C
CE2-C
71076

Choosing Content Engines to Serve End User Requests

The Content Router chooses Content Engines that are suitable for the client request from its proxy tables—automatically maintained lists that contain information about the proximity of Content Engines to particular DNS proxies.

Content routing decisions are based on DNS proximity of the client's DNS server to Content Engines on the network that can serve the requested content. DNS proximity is measured by Round Trip Time (RTT) probes.

The Content Router returns information about the Content Engines it has selected from its routing tables to the DNS proxy in the form of NS records.

Each record identifies a Content Engine that can cache the desired content. The Content Router also returns Address (“glue”) records for the Content Engines so that the DNS proxy knows the IP addresses of the Content Engines.

Several NS records are sent in the reply. The number of records returned and the length of time that they can be used by the proxy (the Time To Live, or TTL value) are controlled by the Content Router. The Content Router adjusts these values depending on how confident it is that its choices are suitable. For example, if the Content Router is certain that particular Content Engines are appropriate for the client, then just two or three name server records are provided to the DNS proxy, with relatively long TTL values. If, however, the Content Router is uncertain about which Content Engines can best provide the requested content, then up to eight name server records are provided to the DNS proxy, with relatively short TTL values.

Distributing Content

In addition to understanding how content is routed between CDN devices, you must also understand what type of content is placed on the CDN to begin with, and how it is placed there. The following sections explain how content distribution from your origin server to the CDN typically occurs and contain information on the following topics:

- Proxy Cached and Pre-Positioned Content, page 1-6
- When to Pre-Position and When to Proxy Cache, page 1-6
- About Hosted Domains, page 1-7
- About the Manifest File, page 1-8
- Supported Content Types, page 1-10

Proxy Cached and Pre-Positioned Content

The Cisco Internet CDN distributes content in two ways:

- Proxy caching—Also referred to as “on-demand” caching. Using this method of content distribution, content is identified as belonging to the CDN by its domain name, which points to a CDN hosted domain. When content is requested by users, it is fetched directly from the origin server by each of the Content Engines in your hosted domain using reverse proxy caching. The content is stored on the Content Engines that belong to a hosted domain.

  After it is retrieved from the origin server, the cached content is periodically refreshed as indicated by an **expires** attribute associated with the content item.

  Subsequent requests for the content are retrieved from the Content Engine cache, assuming the TTL of the object has not expired.
- Pre-positioning (for video-on-demand streamed content, live streamed content, and content served using HTTP)—content is named in an XML-format manifest file, which is read by the CDN software at an interval set using the Content Distribution Manager graphical user interface. Content named in the manifest file is then “fetched” from the origin server, with the exception of live content, and placed on the Content Engines belonging to a hosted domain. Each pre-positioned content item can specify its own start and end time, during which it will be served to users.

When to Pre-Position and When to Proxy Cache

The CDN software allows content providers to pre-position and cache large amounts of content. In theory, an entire web site could be placed on a Internet CDN hosted domain. However, depending on the availability and cost of network bandwidth versus storage and CPU time on your service provider’s Content Engines, this may or may not be prudent. For example, while it is possible to cache a series of small JPG files from your web site on your CDN hosted domain, it may not be necessary to do so.

Although the needs of each content provider are different, consider certain guidelines when deciding whether to proxy-cache or pre-position a particular piece of content on your CDN:

- All streamed content must be pre-positioned using the manifest file.
- Large content items should be proxy-cached to provide good quality service to the first customer who requests the content item.
- Content that must always be available, even when it is not being accessed very often, should be pre-positioned. The content expiration tags in the manifest allow vital content to avoid being purged even when it has not been accessed in a while. See the "Manifest File Structure and Syntax" section on page 3-16 for instructions on setting the expiration time for content items.
- Content that should be generally available to users and may be frequently requested may be proxy-cached.
- Content for which you want to control the availability by specifying start- and stop serving times should be pre-positioned. See the "Manifest File Structure and Syntax" section on page 3-16 for instructions on setting start and stop times for CDN content items.
- Single files larger than 500 MB cannot be proxy-cached. This ceiling can be raised by your service provider. If you will be proxy-caching very large files, first coordinate with your service provider on the maximum allowable file size.

About Hosted Domains

As a content provider, your web site content will be stored on one or more subdomains or "hosted domains," of your web site.

Each hosted domain is created by the administrator for the authoritative DNS servers for your CDN. Subdomains are created as branches in the DNS tree for the web site you are hosting. For example, the following might be hosted domains for some popular web sites:

```
http://www.videos.cnn.com
```

```
http://www.sports.bbc.co.uk/
```

Each hosted domain contains a set of related content that you want to treat as a unit for the purposes of caching. That content is stored in the caches of Content Engines associated with the hosted domain.

For each hosted domain, you must define an origin server, which is the fully qualified domain name (FQDN) of the web server where the actual content for that hosted domain is stored.

See the "About Creating the CDN Subdomain" section on page 2-2 for more information on configuring DNS for CDN deployment.

If live, video-on-demand, or pre-positioned content will be distributed from the hosted domain, a manifest file is also required to identify which live and video-on-demand content on the origin server will be pre-positioned on a hosted domain.

Note The CDN software has a configureable limit on the number of content items deployed across all hosted domains. If you will be deploying a very large number of content items on your hosted domain, in excess of 400,000, coordinate with your service provider ahead of time to ensure that the ceiling is adjusted.

See the "Pre-Positioning Web Site Content" section on page 3-11 for more information on creating and validating manifest file content.

From the perspective of your end users, a hosted domain is identified by its DNS name. The end user accesses content either by entering a URL in a web browser or by clicking a link on a web page. When the user requests content, the DNS server returns the IP address of a cache that is storing the requested content and the user receives the content.

The Cisco Internet CDN Software routing system provides a way of translating a subdomain name into the IP address of a cache that stores content and is a good choice for the client making the request. The client can then send Hypertext Transfer Protocol (HTTP) requests directly to the selected cache. If the cache does not already contain the cached content, it obtains the content from the origin server for the requested hosted domain.

About the Manifest File

The manifest file is an XML-based reference file that you, the content provider, will use to list content that is to be served for a hosted domain. Each hosted domain has only one manifest file.

Manifest files are positioned on a web server at a location that you choose. The URL of this location is provided to the Internet CDN administrator when the hosted domain is created, and a link is created between the hosted domain and manifest file using the CDM graphical user interface. The CDN software fetches the manifest first from the origin server using this URL.

After it has been uploaded to the CDN, the manifest is copied out to each Content Engine assigned to the hosted domain according to a set distribution hierarchy.

Manifest files solve the following problems:

- They allow administrators to fetch content from multiple origin servers and fail-over to backup origin servers and/or the default origin server specified on the Hosted Domain Configuration page of the Content Distribution Manager user interface, thus providing for a degree of fault tolerance.
- They allow the system to import items via HTTP, while serving them using another streaming protocol based on a designated server-type to play the requested file.
- They allow content retrieval and distribution to be controlled by setting specific dates and times when it is to be available or not available.
- They allow the use of wildcards ("*") when specifying live content. This permits you to broadcast new live streams without having to update the manifest with a new item description.

Note Any number of origin servers can be defined in a manifest file. The number of content items, however, is limited to 10,000.

Administrators must be very careful to not make syntax mistakes when creating manifests. Like HTML, even a small mistake will cause errors when the file is imported and parsed. For this reason, Cisco provides a manifest file syntax validator. See the "Validating Manifest File Syntax" section on page 3-39 for directions on obtaining and using this utility to check your manifest file syntax.

For more information on the role of the content provider in deploying web site content on a CDN, see Chapter 2, "Understanding the Content Provider Role."

Supported Content Types

Cisco Internet CDN Software supports delivery of the following file formats:

- HTTP Content Types
- Apple QuickTime Content Types
- Microsoft Windows Media Content Types
- RealNetworks RealServer Content Types

HTTP Content Types

The Cisco CDN Software supports any standard content type served by an HTTP server, including:

- Graphics Interchange Format (GIF)
- Hypertext Markup Language (HTML, HTM)
- Joint Photographic Experts Group (JPG)
- Motion Picture Experts Group (MPEG, MPG)
- MPEG Audio Layer 3 (MP3)
- Portable Document Format (PDF)

Apple QuickTime Content Types

- Audio Visual Interleaved (AVI)
- QuickTime (QT)
- QuickTime Movie (MOV)

Microsoft Windows Media Content Types

- Microsoft Windows Media Player (ASF and ASX)
- Microsoft Windows Media Audio (WMA)
- Microsoft Windows Media Video (WMV)
- Microsoft PowerPoint (PPT)
- Microsoft Word (DOC)

RealNetworks RealServer Content Types

- RealAudio (RA)
- RealMedia (RM)
- RealPix (RP)
- RealText (RT)
- RealVideo (RV)

RealNetworks synchronized container format (SMIL)

Understanding the Content Provider Role

This chapter and the next, Chapter 3, "Deploying Web Site Content on an Internet CDN," describe the following processes, which you must complete to prepare your web site for a CDN deployment:

- About Creating the CDN Subdomain, page 2-2
- About Configuring Origin Servers for Live Streaming, page 2-4
- About Placing Content on Your CDN, page 2-5
- About Controlling Content Freshness after Deployment, page 2-7

As a content provider, your organization delivers high value, high-bandwidth content to your customers or staff over an intranet or the public Internet. You may be a corporation that wants to offer live, streamed broadcasts of its CEO's quarterly earnings announcements to employees at their desktops, or you may be a public university that needs to deliver classroom content to distance learners in remote locations.

Whatever your business, you need a reliable, fast, and efficient means to deliver high-bandwidth web content to end users. To do this, you have likely turned to an outside "service provider" organization to help you manage your content distribution. That service provider has chosen the Cisco Internet Content Delivery Network (CDN) Software to manage the distribution of content to your users.

This chapter is designed to help you understand more fully the delineation of your responsibilities in deploying your CDN solution, as well as those of your service provider.

With the necessary technology infrastructure and the expertise on-staff to maintain and fine-tune the Cisco CDN hardware and software, your service provider handles day to day maintenance of your CDN.

However, some of the work of deploying your web site content on a CDN rests with you, the content provider. This work involves:

1. Coordinating with your service provider to create the DNS subdomains of your domain on which CDN content will reside. This includes assigning authoritative DNS control of your CDN subdomains to the content routers that will direct user requests to Content Engines.
2. Configuring origin content servers for live streaming. (Optional.)
3. Identifying content on your web site that needs to be pre-positioned or on-demand cached on your CDN hosted domain.
4. Changing the URLs of content you want cached to point to your CDN hosted domain.
5. Generating a manifest file that identifies the web site content you wish to pre-position on your CDN.
6. Notifying your service provider of the name and location of a manifest file for deployment on your designated hosted domain.
7. Monitoring content freshness after deployment.

About Creating the CDN Subdomain

With the help of the Cisco Internet CDN solution, selected content from your web site is mirrored on one or more CDN subdomains referred to as *hosted domains*. Each of these hosted domains is a subdomain in the DNS tree where clients make DNS resolution requests to CDN-controlled Content Routers.

Hosted domains use a subdomain name that is related to your web site's domain name. For example, if your web site DNS address is:

http://www.cisco.com

Users might access a copy of your web site from the following accelerated CDN hosted domain:

http://www.cdn.cisco.com

Each hosted domain contains a collection of cached or pre-positioned video-on-demand content that can be served from Content Engines deployed throughout your CDN. Your organization may be deploying one or more CDN-enhanced hosted domains—each corresponding to a different DNS subdomain—at once.

The following section explains how to modify a DNS server to create a CDN subdomain and explains the various options available to you, as a content provider, in deploying a CDN subdomain for your web page.

Modifying DNS Configuration to Create a CDN Subdomain

CDN hosted domains are created when you modify your web site's DNS tree to create one or more new subdomains to host CDN content. Each subdomain must be delegated to the Content Routers responsible for managing user requests on the CDN.

Refer to the following guidelines when creating a subdomain for use with the Cisco Internet CDN Software:

1. The hosted domain name must be a valid subdomain name. For example:

   ```
   www.cdn.cisco.com
   ```

   a. The first part of the domain name (www in this example) is open, and can be defined by you when you create the hosted domain name.
   b. The remaining subdomain (the three segments after the first dot) must correspond to entries on your DNS server to provide a functional mapping for the CDN Content Routers that ultimately provide the appropriate DNS records.
2. The DNS server on which the subdomain is created must be given the right to act as the authoritative DNS server for the subdomain you specify.
3. The hosted domain name cannot contain underscore (_) characters.

Refer to the *Cisco Internet CDN Software User's Guide* or the documentation for your DNS server software for more information on creating subdomains or appending name server (NS) records.

Creating the Hosted Domain

After you have created the necessary DNS subdomains that will store your CDN content, your service provider can create the actual hosted domain using the CDN administrative software.

Using the CDN administrative interface, your service provider creates the hosted domain, assigns it a name, and associates it with the new DNS subdomain using the Content Distribution Manager graphical user interface.

Next, one or more Content Engines are assigned to the hosted domain using the Content Distribution Manager graphical user interface. Content Engines store the pre-positioned and cached content, then serve it to local users.

If you will be pre-positioning video on demand or live streamed content, your service provider must also link the hosted domain to a manifest file you have created.

See the "Pre-Positioning Web Site Content" section on page 3-11 for instructions on creating a manifest file that identifies the web site content you want to pre-position and the "On-Demand Caching Web Site Content" section on page 3-4 for instructions on modifying content URLs to cache web site content on your CDN hosted domain.

About Configuring Origin Servers for Live Streaming

If you will be hosting live content on your CDN, you may need to modify the configuration of your live content servers. Currently, the Cisco Internet CDN Software supports live broadcasts from the following platforms:

- Real Networks' RealServer Version 8.0 or later
- Microsoft's Windows Media Services

See the "Configuring Origin Server for Live Streaming" section on page 3-1 for detailed instructions on modifying your origin servers for live streaming over the CDN.

About Placing Content on Your CDN

Your web site contains a variety of content types, from static image files in GIF or JPG format, to external documents, maybe in Adobe PDF format, to live content streamed by RealServer or Windows Media Services.

As part of the process of preparing your website for deployment on a CDN, you must decide which of this content you want to have accessed from your CDN and which will continue to be accessed and served from your web servers.

There are two options for deploying content. These options are not mutually exclusive; both can be used to deploy content on a hosted domain:

- Caching—Modify your web site HTML for specific content items so that the HREF tags point to your CDN hosted domain instead of your web site origin servers. This results in the content item being cached on the Content Engines assigned to your hosted domain. This is appropriate for most web content served by HTTP, but cannot be used with live and on-demand content.
- Pre-positioning—Generate an XML-format manifest file that lists all the content you will be deploying on your CDN, then provide your service provider with the location of this XML file. Pre-positioning is appropriate for use with on-demand and live streams as well as content served using HTTP.

Each of these options is explained, in detail, in the sections that follow.

About Caching Content

After your hosted domain has been created and content has been pre-positioned on it, you are ready to change your web site's content URLs to point to the new hosted domain address supplied by your service provider.

See the "Caching Content on Your CDN" section on page 3-4 for instructions on modifying your web site's content URLs to point to your CDN.

About Pre-Positioning Content

To pre-position content, you must:

- Identify the content you want to place on your CDN.
- Create the manifest file.

About Identifying Content to Place on Your CDN

As the content provider, you are responsible for compiling a list of the content you want to deploy on the CDN and providing that list to your CDN administrator in the form of an XML-format manifest file.

The easiest way to generate a list of content on your web site is to create an automated script that "crawls" your entire web site (or as much of it as you want to be crawled) and identifies content that needs to be pre-positioned on the CDN based on rules you provide.

Cisco provides a free PERL script, Spider, you can use as a template for the creation of your spider script. When run, Spider outputs a database of web site content, based on rules you supply in a separate configuration file, that can be used to build a functioning manifest file in a fraction of the time it would take to author a manifest file by hand.

See the "Pre-Positioning Web Site Content" section on page 3-11 as well as "Listing Web Site Content Using the Spider Script" section on page A-3 for instructions on identifying content to place on your CDN.

About Creating the Manifest File

The easiest way to generate a manifest file that lists the content you want to pre-position is to create an automated script that builds the manifest, based on the data generated by the Spider script.

Cisco provides a free PERL script, Manifest, that you can use as a template for your own script. When run, Manifest outputs a valid, CDN manifest file based on the database of web site content generated by the Spider script and content rules that you supply in a separate configuration file.

See the "About the Manifest File" section on page 1-8 for more information on the manifest file. Then see the "Creating a Manifest File" section on page 3-12 to create your manifest file. Detailed information on the Manifest script can be found in the "Selecting Live and Pre-position Content Using the Manifest Script" section on page A-7.

After you have generated your manifest file, you will inform your service provider about it, and supply them with its location on your origin server. Your service provider will link the manifest file to the hosted domain created for your content, after which the process of pre-positioning content on the CDN can begin.

About Controlling Content Freshness after Deployment

One of the primary concerns you will have once you have deployed your content on a Cisco Internet CDN is controlling content freshness: making sure that users are accessing up-to-date content from CDN caches, and making sure that updated material is quickly disseminated throughout the CDN.

Refreshing Manifest File Content

Because you create and maintain the manifest file and host it on your web server, you can determine when any content item is refreshed.

The frequency with which individual content items on the CDN are refreshed is controlled using the both the **expires** and **ttl** (Time to Live) attributes in the manifest file.

- expires—designates a time (in yyyy-mm-dd hh:mm:ss format) after which the named content item will no longer be served from the CDN. The time specified is always as of the time zone specified by the **timezone** attribute of the <options> tag. If no time zone is specified, the default (GMT) is used.
- ttl—identifies the frequency, in minutes, with which the origin server is polled to see if a content item has changed, judging by the content item timestamp or file size. The default ttl value, which is controlled from the Content Distribution Manager administrative interface is 30 minutes.

Note The default ttl time can be modified by your service provider using the Content Distribution Manager administrative interface. Please consult with your service provider if you would like to change the default ttl value.

As content is changed and updated on your origin server, it will also be updated on the CDN at an interval represented by the **ttl** attribute. Keep in mind, however, that the **ttl** attribute represents the minimum time in which the content will be updated. Depending on the volume of content that must be refreshed at any given time and the file size of the piece of content that must be refreshed, it could take longer than the interval specified by the **ttl** attribute to refresh any one piece of content.

See the "Manifest File Structure and Syntax" section on page 3-16 for information on using the **expires** and **ttl** attributes.

Refreshing the Manifest File

While the **expires** and **ttl** attributes are the most common methods for controlling the freshness of individual content items that are named in the manifest file and that have been updated, if you will be making a large number of changes to your manifest file, you may want to update, or re-fetch, the entire manifest.

Fetching is initiated from the Content Distribution Manager administrative interface. Coordinate with the operations person at your service provider organization to re-fetch the manifest for your hosted domain(s).

Content that is only partially fetched from your origin server is not added to the CDN and will not be served from you hosted domain. See the "Verifying Content Freshness"section next for information on how content freshness and replication status is monitored by your service provider.

Verifying Content Freshness

The Cisco Internet CDN software provides a wealth of information to CDN administrators on the status of content replication and freshness through a variety of logging mechanisms.

Log files allow CDN operations personnel and administrators to monitor the status of all devices on the CDN as well as the status of content replication and content freshness on hosted domains.

From the Content Distribution Manager administrative interface, your service provider can verify whether content replication to a hosted domain was successful.

To verify if a piece of content has been refreshed, your service provider will look at manifest log files that reside on each Content Engine belonging to your hosted domain. Within that log file are entries pertaining to each content item on the hosted domain. Your operations person will want to look for log entries pertaining to the particular content file name and verify that it was successfully imported to or refreshed on the hosted domain.

In the event that errors were encountered during replication, either because the disk space allocation for the hosted domain content was exceeded, or because replication failed for a particular content item, your service provider has a number of tools available to them to correct the problem, including allocating more disk space for the hosted domain and re-fetching the manifest file. In the event that a corrupt content item is causing problems, your service provider may ask you to remove references to that content from your website or manifest file. See the "Obsoleting Bad Content" section next for more information on removing bad content from your hosted domain.

Because access to the CDN log files requires direct access to CDN devices, much of this logging is transparent to you as a content provider. Coordinate with your service provider regarding content updates.

Obsoleting Bad Content

If you have accidentally positioned a bad content item on your hosted domain with an expiration date in the distant future, you can remove that content. Do so by revising your content URL or your manifest file to omit the reference to the damaged content item. If you need to update your manifest file, remember to ask your service provider to re-fetch the manifest file. The bad content item will no longer be accessible from the CDN because it is not named in the manifest and the disk space allocated by the obsoleted content item will be reallocated for valid content.

See the "Creating a Manifest File" section on page 3-12 for instructions on creating your manifest file.

Deploying Web Site Content on an Internet CDN

This chapter contains information on the following topics:

Configuring Origin Server for Live Streaming

The Cisco Internet CDN Software supports media associated with a wide variety of content servers, including RealServer, Windows Media Server, and Apple QuickTime Server. In addition, the Internet CDN Software supports video-on-demand (VOD) via HTTP.

Both TCP and UDP Unicast streams are supported, however multicast streaming is not supported.

Live streaming is supported when using either the RealServer Version 8.0 or higher, as well as Windows Media Server platforms. Live streaming with QuickTime Server is not supported.

The following sections will help you configure your origin content servers for live streaming:

- Configuring RealServer for Live Streaming, page 3-2
- Configuring WMT Publisher for Live Streaming, page 3-3

Configuring RealServer for Live Streaming

Cisco Internet CDN Software supports the splitting of live streams, which enables live broadcasts to be forwarded from an origin RealServer (referred to as a transmitter) to one or more receiver RealServers.

Note Cisco Internet CDN Software only supports RealServer Version 8.0 and higher.

Splitting makes it possible to replicate streams to locations close to requesting clients, which improves the response time for client requests and the quality of the streamed broadcast and makes it possible to serve a larger number of clients.

Before taking advantage of the live splitting feature, make sure that the following conditions have been met:

- You must have RealServer Version 8.0 or higher running on your origin server.
- A pull-split source must be defined using the RealSystem Administrator utility.

  Refer to Chapter 12, “Splitting Live Presentations,” in the *RealServer Administration Guide* for instructions on setting live stream security as well as configuring your transmitting server for pull splitting using the RealSystem Administrator utility.

  The *RealServer Administration Guide* is available online at the following web address:

  http://service.real.com/help/library/guides/server8/realsrvr.htm
- If a pull-split listen port other than the default (2030) will be used, the manifest file for the hosted domain should identify the port to be used along with the host name under the server definition.

  For example, if port 2070 were to be used, the manifest file would read:

  ```
  <server name=”transmitting-server”>
  <host name= ”10.89.1.1:2070”/>
  </server>
  ```

  If no port is defined after the host name, the default port is used as the listen port.

See the “Manifest File Structure and Syntax” section on page 3-16 for instructions on modifying the <host /> attribute in your manifest file to include the nondefault listen port on the transmitting RealServer.

- The Security Type parameter is set to *None*.

Configuring WMT Publisher for Live Streaming

To serve live content using Windows Media Services over your CDN, make sure the following conditions have been met:

- Windows Media Services are installed and running on your origin server.

  If you are running Microsoft Windows® 2000 Server, Windows Media Services is included. Otherwise, download Windows Media Services from the Windows Media Web site at the following address and then install it:

  http://www.microsoft.com/windows/windowsmedia/default.asp

  Directions for installing Windows Media Services can be found at:

  http://www.microsoft.com/windows/windowsmedia/serve/wms_install.asp

  Note When installing Windows Media Services, make sure you are logged on as the administrator.

- Your firewall has been configured to allow traffic to and from the Windows Media servers and clients.

  Generally, content streamed using TCP Unicast uses TCP port 1755 for traffic in to and out from the Windows Media servers, while content streamed using UDP Unicast uses TCP Port 1755 for traffic in to the Windows Media servers and a UDP port between 1024 and 5000 for traffic out from the Windows Media servers.

  However, different configurations exist depending on your own network and firewall configuration. Refer to the Microsoft documentation on “General Protocol and Firewall Information” for Windows Media Services online at:

  http://www.microsoft.com/windows/windowsmedia/serve/firewall.asp

- A publishing point for live streamed content has been created. With live content, the Windows Media Encoder is used to encode directly from the source (that is, a live video feed) to a publishing point. Each requesting client receives a separate live stream.

  Refer to the Microsoft documentation on configuring Windows Media Services for live unicasting online at:

  http://www.microsoft.com/Windows/windowsmedia/serve/basics_wm4.asp

On-Demand Caching Web Site Content

This section explains the steps necessary to publish content on your CDN and to create functioning links to CDN content on your web page. This section contains information on the following topics:

- URLs for Content Served Using Web Server, page 3-7
- URLs for Content Served Using RealServer, page 3-7
- URLs for Content Served Using Quicktime Server, page 3-8
- URLs for Content Served Using Windows Media Services, page 3-9

For all example URLs, refer to the sample manifest file provided in Example 3-2 on page 3-38.

Caching Content on Your CDN

To place content on your CDN, you must alter your web site content URLs to point to your CDN hosted domain.

For example, if your web site contains the following content URL:

```
http://www.cisco.com/images/logo.gif
```

You can place the content named on your CDN hosted domain, http://www.elearning.cisco.com by modifying the original URL to read:

```
http://www.elearning.cisco.com/images/logo.gif
```

When a user visiting your web site requests the page on which the logo.gif is located, that image is retrieved from the nearest CDN cache. If the image is not already cached on a Content Engine that is assigned to the specific hosted domain, it is retrieved from the origin server and placed in the cache of Content Engines belonging to the hosted domain. Subsequent requests for the image are retrieved from the cache rather than the origin server.

Creating URLs that Link to CDN Content

All URLs pointing to CDN content use the following structure:

http://*hosted_domain_name*/*cdn-url*

Optionally, CDN URLs can contain a CDN media tag that force a media play server designation (for example, RealServer, Windows Media Server, QuickTime Server) for the content item using the following format:

http://*hosted_domain_name*/*cdn_media_tag*/*cdn-url*

Links rely on playserver mappings for each type of content that is hosted (that is, PDF, JPG, MPG, WMV, AVI, RM). Play server mappings can be found in the manifest file or the PlayServerTable and are consulted by the CDN software using the following hierarchy:

- A playserver designation embedded in the actual content URL is checked first.
- The playserver attribute for the <item /> tag corresponding to the piece of content is checked second.
- The playserver attribute for the <item-group /> tag corresponding to the piece of content is checked third.
- If defined, the playserver attribute for the hosted domain in <playServerTable> </playServerTable> in the manifest is checked fourth, according to the following order:
  - <content-type /> tags are checked first in <playServerTable>
  - <extension> tags are checked second in <playServerTable>

Though removing CDN media tags from URLs makes the job of modifying web site content for deployment on a CDN much easier, it also requires increased attention to playserver mappings in the manifest file to ensure that all file types being served have been associated with one of the supported playserver types.

The elements of these URLs are described in Table 3-1. Be aware that all URL information is case sensitive—ignoring case sensitivity in your published web pages will result in the CDN being unable to retrieve the requested content.

Table 3-1 Components of a CDN URL

URL Component	Description
hosted_domain_name	Fourth-level domain name assigned to your hosted domain or the alias assigned to that hosted domain. See the "About Creating the CDN Subdomain" section on page 2-2 for more information on creating CDN subdomains.
cdn_media_tag	Optional. The media server designated to handle the content item to which you are linking. If no cdn media tag is supplied, the playserver attributes in the manifest file are checked in order of preference as described on page 3-5. CDN-media is used as the default playserver designation and causes the request to be resolved using a playserver mapping in the manifest file. See Table 3-3 for a list of media formats, organized by extension. The following cdn media tags are supported in CDN URLs: • *cdn-media* • *cdn-real* • *cdn-qtss* • *cdn-http* • *cdn-wmt*
cdn-url	Relative location of the content item on the CDN device. This value is supplied by the *cdn-url* or *src* attribute in the <item> tag in the manifest file for each piece of content. Wildcard characters are accepted in the *cdn-url attribute* only when you link to live content. Actual content URLs must of course point to the actual content item.

URLs for Content Served Using Web Server

The web server can be used to serve content that cannot be served using other content servers (RealServer, WMT, or QuickTime). What follows are two examples of URLs for content that will be served using the web server:

```
http://www.cisco.meetings.com/agendas/q4results.pdf

http://www.cisco.meetings.com/cdn-http/agendas/q4results.pdf
```

- In the first example, no cdn media tag is applied in the URL, so the request will resolve to the web server based on a mapping of the PDF extension in the <playServerTable> area of the manifest file.
- In the second example, a cdn media tag, cdn-http, is supplied in the URL forcing the CDN software to stream the content file using the designated server.

URLs for Content Served Using RealServer

RealServer Version 8.0 or higher can be used to serve a variety of RealNetworks content types including RealAudio (RA), RealMedia (RM), and SMIL format files, as well as live presentations.

URLs for On-Demand Content Served Using RealServer

What follows are three examples of valid URLs for on-demand (not live) content served from the CDN using RealServer, based on the sample manifest file in Example 3-2 on page 3-38:

```
http://www.cisco.meetings.com/present/q4presentation.rm

http://www.cisco.meetings.com/cdn-real/present/q4presentation.rm

http://www.cisco.meetings.com/cdn-media/present/q4presentation.rm
```

- In the first example, no CDN media tag is applied in the URL, so the request will resolve to the content server based on a mapping of the RM extension in the <playServerTable> area of the manifest file.
- In the second example, a CDN media tag, cdn-real, is supplied in the URL forcing the CDN software to stream the content file using the designated server.

- In the third example, the default CDN media tag, cdn-media, is supplied, causing the file to be served according to the <extension> mapping for RM in the <playServerTable> area of the manifest file.

URLs for Live Content Served Using RealServer

Live presentations can be resolved just like any other content handled by RealServer, provided you know the name of the live content in advance. For example, your manifest file might name the following item:

```
<!--live content item-->
<item src="/encoder/q4live.rm" cdn-url="/live/q4presentation.rm"
type=”live”/>
```

In this case, your published URL linking to this file might look like this:

```
http://www.cisco.meetings.com/present/live/q4presentation.rm
```

However, because many live presentations are not named in advance of when they air, the Cisco CDN software allows you to wildcard references to live presentations in the manifest file in addition to specifying file names in advance.

Wildcarding allows the CDN software to properly map all live RealNetworks streams to RealServer, regardless of name, as long as they carry the proper file extension (for example, RM).

See the sample manifest file provided in Example 3-2 on page 3-38 to view the wildcarded mapping for RealServer live content.

If you will be wildcarding references to live presentations in your manifest file, you need to supply the cdn-live tag to any URLs for live content. For example:

```
http://www.cisco.meetings.com/cdn-live/present/live/q4presentation.rm
```

The cdn-live tag ensures that a connection is established to the RealServer encoder mountpoint based on the <extension> mapping in the <playServerTable>, even though no exact filename mapping is possible.

URLs for Content Served Using Quicktime Server

QuickTime server can be used to serve a variety of content types including MOV, QT, MP4, and AVI.

Note We recommend mapping AVI files to Windows Media as opposed to QuickTime, due to the easy availability of the Windows Media Player on end user desktops.

What follows are three examples of URLs for QuickTime content that will be served using QuickTime Server:

```
http://www.cisco.meetings.com/facilities/newHQ.mov

http://www.cisco.meetings.com/cdn-qtss/facilities/newHQ.mov

http://www.cisco.meetings.com/cdn-media/facilities/newHQ.mov
```

- In the first example, no CDN media tag is applied in the URL, so the request will resolve to the content server based on a mapping of the MOV extension in the <playServerTable> area of the manifest file.
- In the second example, a CDN media tag, cdn-qtss, is supplied in the URL forcing the CDN software to stream the content file using the designated server.
- In the third example, the default CDN media tag, cdn-media, is supplied, causing the file to be served according to the <extension> mapping for MOV in the <playServerTable> area of the manifest file.

URLs for Content Served Using Windows Media Services

Microsoft's Windows Media Services can be used to serve a variety of Windows media content types including Windows Media Audio (WMA), Windows Media Video (WMV), and Active Server (ASF) format files, as well as live presentations.

URLs for On-Demand Content Served Using Windows Media Services

What follows are three examples of valid URLs for on-demand (not live) content served from the CDN using Windows Media Services, based on the sample manifest file in Example 3-2 on page 3-38:

```
http://www.cisco.meetings.com/marketing/prodroadmap.asf

http://www.cisco.meetings.com/cdn-wmt/marketing/prodroadmap.asf

http://www.cisco.meetings.com/cdn-media/marketing/prodroadmap.asf
```

- In the first example, no CDN media tag is applied in the URL, so the request will resolve to Windows Media Services based on a mapping of the ASF extension in the <playServerTable> area of the manifest file.
- In the second example, a CDN media tag, cdn-wmt, is supplied in the URL forcing the CDN software to stream the content file using the designated server.
- In the third example, the default CDN media tag, cdn-media, is supplied, causing the file to be served according to the <extension> mapping for ASF in the <playServerTable> area of the manifest file.

URLs for Live Content Served Using Windows Media Services

Live presentations can be resolved just like any other content handled by Windows Media Services, provided you know the name of the live content in advance. For example, your manifest file might name the following item:

```
<!--live content item-->
<item src="/encoder/roadmap.asf" cdn-url="/live/prodroadmap.asf"
type=”live”/>
```

In this case, your published URL linking to this file might look like this:

```
http://www.cisco.meetings.com/marketing/live/prodroadmap.asf
```

However, because many live presentations are not named in advance of when they air, the Cisco CDN software allows you to wildcard references to live presentations in the manifest file in addition to specifying file names in advance.

Wildcarding allows the CDN software to properly map all live Windows Media streams to Windows Media Services, regardless of name, as long as they carry the proper file extension (for example, ASF).

See the sample manifest file provided in Example 3-2 on page 3-38 to view the wildcarded mapping (for RealServer live content in the example).

If you will be wildcarding references to live presentations in your manifest file, you need to supply the cdn-live tag to any URLs for live content. For example:

```
http://www.cisco.meetings.com/cdn-live/marketing/live/prodroadmap.asf
```

The cdn-live tag ensures that a connection is established to the Windows Media Services encoder mountpoint based on the <extension> mapping in the <playServerTable>, even though no exact filename mapping is possible.

Pre-Positioning Web Site Content

To pre-position content on your CDN, you must create a manifest file that points to that content. A manifest file might point to content in one location on your web server, or name content from a number of different web server locations. Before building a manifest file, however, you first have to know the relative location for all content items that you will be pre-positioning on the CDN.

After you have a list of all the content on your web site that you will be placing on your CDN and its relative location on your origin server, you can generate a manifest file that identifies the content—live or VoD—to the CDN software. See the "Pre-Positioning Web Site Content" section on page 3-11 for instructions on generating a manifest file.

Creating a List of Web Site Content

The simplest way to generate a list of web site content is manually. If you have a small amount of content that you will be pre-positioning, or content that is centralized in one or two locations on your web server, you may not need to separately list your web site content before generating a manifest file for it.

If you are pre-positioning only a small, or easily managed amount of content, proceed to the "Creating a Manifest File" section on page 3-12.

Spider Script

If, however, you will be placing all your website content on a CDN, the easiest and most efficient way to generate a list of all the content that must be pre-positioned is to use a spidering tool (called "Spider") to "crawl" your site. The Spider script follows any HREF links back to the content they point to, and makes a record of that content and its location.

For more information on the Spider script, see the "About Placing Content on Your CDN" section on page 2-5.

For instructions on using the Spider script to generate a list of your web site content, or modifying the Spider script to suit the needs of your own web servers, see the "Listing Web Site Content Using the Spider Script" section on page A-3.

Creating a Manifest File

After you have successfully crawled your web site, reviewed the output from your Spider script, and decided which content you will be pre-positioning, you are ready to generate a manifest file.

This section contains instructions for generating a manifest file and information on the following topics:

- Manifest File Limitations, page 3-13
- Manifest File Document Type Definitions, page 3-13
- Manifest File Structure and Syntax, page 3-16
- Validating Manifest File Syntax, page 3-39
- Repairing Manifest File Syntax, page 3-46

The manifest file is an XML-based file that provides powerful features for representing and manipulating CDN data. Manifest files need to be created for each of your hosted domains and exist in a one-to-one relationship with the hosted domain. After reading the manifest file, the CDN software uses its instructions to replicate content from your web server to Content Engines on the CDN that are assigned to your hosted domain.

Manifest Script

The easiest and most efficient way to generate your manifest file is to use an automated script that builds the manifest syntax based on content rules you specify, and use the list of content items generated by your Spider script.

As with the Spider script, Cisco provides a generic manifest generation script (“Manifest”) to its CDN customers. Written in PERL, this script can be used to output an XML-format manifest file that conforms to Cisco CDN standards.

For instructions on locating and using the Manifest script to generate a manifest file that points to your web site content, see the “Selecting Live and Pre-position Content Using the Manifest Script” section on page A-7.

Manifest File Limitations

When generating your manifest file, keep in mind the following limitations:

- For versions of the Internet CDN software prior to Version 2.1.1, the manifest cannot contain more than 10,000 content items.
- The total of size of all the objects named in any single manifest file for a hosted domain cannot exceed 2 gigabytes. (Live content does not count toward the hosted domain size.)

Manifest File Document Type Definitions

Example 3-1 provides type definitions for the various elements of an Internet CDN manifest file. Details on each manifest file element follow.

Example 3-1 Manifest Document Type Definitions (DTDs)

```
<!-- CdnManifest - DTD for Cisco iCDN 2.0 pre-positioned content manifest. Copyright (c)
2001 by Cisco Systems, Inc., Waltham, Massachusetts -->

<!ENTITY % playServerTable SYSTEM "PlayServerTable.dtd">
%playServerTable;

<!ELEMENT CdnManifest (playServerTable?, options?, server*, (item | item-group)*)>
  <!ELEMENT options EMPTY>
  <!ATTLIST options
            clearlog (true | false) "false"
            rd CDATA #IMPLIED
            prepos-tag CDATA #IMPLIED
            live-tag CDATA #IMPLIED
            notFoundUrl CDATA #IMPLIED
            noRedirectToOrigin (true | false) "false"
            timeZone CDATA #IMPLIED
            manifest-id CDATA #IMPLIED>

  <!ELEMENT host EMPTY>
  <!ATTLIST host
            name CDATA #REQUIRED
            proto (http) "http"
            port CDATA #IMPLIED
            user CDATA #IMPLIED
            password CDATA #IMPLIED>

  <!ELEMENT server (host+)>
  <!ATTLIST server
            name CDATA #REQUIRED>

  <!ELEMENT contains EMPTY>
  <!ATTLIST contains
            cdn-url CDATA #REQUIRED>

  <!ELEMENT item (contains*)>
  <!ATTLIST item
            cdn-url CDATA #IMPLIED
            src CDATA #REQUIRED
            server CDATA #IMPLIED
            playserver (real | wmt | http | qtss) #IMPLIED
            type (prepos | live) #IMPLIED
            ttl CDATA #IMPLIED
            serve CDATA #IMPLIED
            prefetch CDATA #IMPLIED
            expires CDATA #IMPLIED
            alternateUrl CDATA #IMPLIED
```

```
          streamProperty CDATA #IMPLIED
          noRedirectToOrigin (true | false) #IMPLIED>

<!ELEMENT item-group (item | item-group)*>
<!ATTLIST item-group
          server CDATA #IMPLIED
          playserver (real | wmt | http | qtss) #IMPLIED
          type (prepos | live) #IMPLIED
          ttl CDATA #IMPLIED
          alternateUrl CDATA #IMPLIED
          cdnPrefix CDATA #IMPLIED
          srcPrefix CDATA #IMPLIED
          streamProperty CDATA #IMPLIED
          noRedirectToOrigin (true | false) #IMPLIED>
```

Manifest File Structure and Syntax

Table 3-2 CDN Manifest File Syntax

Manifest Tag	Description	Options	Syntax Example

<CdnManifest> </CdnManifest>	Required.
	
	The <CdnManifest> tag marks the beginning and end of the manifest file content. At a minimum, each <CdnManifest> tag set must contain at least one item that will be fetched and stored on the hosted domain, and may optionally reference a list of host servers from which content will be fetched.
	
	Any number of servers, hosts, and items can be defined, up to a limit of 10,000 items in the manifest file.
			`<CdnManifest>`
			`<server name=“origin-server”>`
			`<host name=”www.name.com” proto="http" port="80" />`
			`</server>`
			`<item cdn-url= “logo.jpg” server=”originserver” src= “images/img.jpg” type=“prepos” playserver=“http” ttl=300/>`
			`</CdnManifest>`

Table 3-2 CDN Manifest File Syntax (continued)

Manifest Tag	Description	Options	Syntax Example

<playServerTable> </playServerTable>	Optional.
	
	Playserver tables provide a way for you to set default mappings for a variety of media types on your hosted domains. Mappings can be set for both MIME content types (the preferred mapping) and file extensions. Playserver tables allow you to override default mappings on the Content Engine for content types on a particular hosted domain.
	
	<playServerTable> tags are enclosed within the <CdnManifest> tags and name at least one playserver, for example, RealServer, to which certain MIME types and file extensions are mapped.
			`<CdnManifest>`
			`<playServerTable>`
			`<playServer name="real">`
			`<contentType name="application/x-pn-realaudio" />`
			`<contentType name="application/vnd.rn-rmadriver" />`
			`<extension name="rm" />`
			`<extension name="ra" />`
			`<extension name="rp" />`
			`<extension name="rt" />`
			`<extension name="smi" />`
			`</playServer>`
			`<playServer name="wmt">`
			`<extension name="asx" />`
			`<extension name="asf" />`
			`<extension name="avi" />`
			`</playServer>`
			`<playServer name="http">`
			`<contentType name="application/pdf" />`
			`<contentType name="application/postscript" />`
			`<extension name="pdf" />`
			`<extension name="ps" />`
			`</playServer>`
			`</playServerTable>`
			`<server name="test.origin.com/">`
			`<host name="http://tst.orgn.com" proto="http" />`
			`</server>`
			`<item cdn-url="pic1.mpg" src="pic1.mpg" server="seaotter.sightpath.com/" type="live" ttl="1" />`
			`</CdnManifest>`

Table 3-2 CDN Manifest File Syntax (continued)

Manifest Tag	Description	Options	Syntax Example

<playServer> </playServer>	Required for <playServerTable> tag.
	
	The <playServer> tag names a media server type on the Content Engine that will be responsible for playing the content types and files with extensions that are mapped to it using the <content-type> and/or <extension> tags.
	
	The <playServer> tag is enclosed within <playServerTable> tags.
		name (required)
		
		Each <playServer> tag names the type of server to which content will be mapped using the *name* attribute. Content Engines support four types of playservers:
		• real (RealMedia RealServer)
		• http (web server)
		• qtss (Apple QuickTime)
		• wmt (Microsoft Windows Media)
			`<CdnManifest>`
			`<playServerTable>`
			`<playServer name="real">`
			`<contentType name="application/x-pn-realaudio" />`
			`<contentType name="application/vnd.rn-rmadriver" />`
			`<extension name="rm" />`
			`<extension name="ra" />`
			`<extension name="rp" />`
			`<extension name="rt" />`
			`<extension name="smi" />`
			`</playServer>`
			`<playServer name="wmt">`
			`<extension name="asx" />`
			`<extension name="asf" />`
			`<extension name="avi" />`
			`</playServer>`
			`<playServer name="http">`
			`<contentType name="application/pdf" />`
			`<contentType name="application/postscript" />`
			`<extension name="pdf" />`
			`<extension name="ps" />`
			`</playServer>`
			`</playServerTable>`
			`<server name="test.origin.com/">`
			`<host name="http://tst.orgn.com" proto="http" />`
			`</server>`
			`<item cdn-url="pic1.mpg" src="pic1.mpg" server="tst.orgn.com/" type="live" ttl="1" />`
			`</CdnManifest>`

Table 3-2 CDN Manifest File Syntax (continued)

Manifest Tag	Description	Options	Syntax Example

<contentType>	Optional.
	
	The <contentType> tag names a MIME content type that is being mapped to a playserver.
	
	The <contentType> tag must be enclosed within a <playServer> tag set. When both <contentType> and <extension> tags are present in a PlayServerTable for a particular media type, the <contentType> mapping takes precedence.
		name (required)
		
		The MIME content type of media that will be mapped to the playserver.
			`<CdnManifest>`
			`<playServerTable>`
			`<playServer name="real">`
			`<contentType name="application/x-pn-realaudio" />`
			`<contentType name="application/vnd.rn-rmadriver" />`
			`<extension name="rm" />`
			`<extension name="ra" />`
			`<extension name="rp" />`
			`<extension name="rt" />`
			`<extension name="smi" />`
			`</playServer>`
			`</playServerTable>`
			`<server name="test.origin.com/">`
			`<host name="http://tst.orgn.com" proto="http" />`
			`</server>`
			`<item cdn-url="pic1.mpg" src="pic1.mpg" server="tst.orgn.com/" type="live" ttl="1" />`
			`</CdnManifest>`

Table 3-2 CDN Manifest File Syntax (continued)

Manifest Tag	Description	Options	Syntax Example

<extension>	Optional.
	
	The <extension> tag names a file extension that is being mapped to a playserver.
	
	The <extension> tag comes after the <playServer> tag. When both <contentType> and <extension> tags are present in a playServerTable for a particular media type, the <contentType> mapping takes precedence.
		name (required)
		
		Provides the file extension for a mapped content type. Valid extensions are:
		
		Playserver = *real*
		• rm
		• smi
		• ra
		• rp
		• rt
		
		Playserver = *http*
		• pdf
		• ps
		• asx
		• wax
		• wvx
		• wmx
		
		Playserver = *qtss*
		• qt
		• mov
		• mp4
		
		Playserver = *wmt*
		• asf
		• wma
		• wmv
		• wm
			`<CdnManifest>`
			`<playServerTable>`
			`<playServer name="real">`
			`<contentType name="application/x-pn-realaudio" />`
			`<contentType name="application/vnd.rn-rmadriver" />`
			`<extension name="rm" />`
			`<extension name="ra" />`
			`<extension name="rp" />`
			`<extension name="rt" />`
			`<extension name="smi" />`
			`</playServer>`
			`</playServerTable>`
			`<server name="test.origin.com/">`
			`<host name="http://tst.orgn.com" proto="http" />`
			`</server>`
			`<item cdn-url="pic1.mpg" src="pic1.mpg" server="tst.orgn.com/" type="live" ttl="1" />`
			`</CdnManifest>`

Table 3-2 CDN Manifest File Syntax (continued)

Manifest Tag	Description	Options	Syntax Example

<server> </server>	Required (minimum of one host).
	
	The <server> tags define a host or set of hosts (or “origin servers”) from which content is to be retrieved.
	
	The <server> tags are contained within <CdnManifest> tags and contain one or more <host> tags, which identify hosts (server locations) from which content will be retrieved.
	
	Within each <server> tag set, be sure to list hosts in order of importance.
		name (required)
		
		Primary hostname or IP address of the server from which content will be retrieved.
			`<CdnManifest>`
			`<server name=“origin-server”>`
			`<host name=”www.name.com” proto="http" port="80" />`
			`</server>`
			`<item cdn-url= “logo.jpg” server=”originserver” src= “images/img.jpg” type=“prepos” playserver=“http” ttl=300/>`
			`</CdnManifest>`

Table 3-2 CDN Manifest File Syntax (continued)

Manifest Tag	Description	Options	Syntax Example

<host />	Required.
	The <host/> tag (defines a web or live server from which content is to be retrieved for hosting on the hosted domain. Multiple host servers can be defined within a single <server> tag set.
	The <host> tag must be enclosed within <server> tags. Multiple <host> tags may appear within the same <server> tag set, and should be listed according to their importance, with the most important host listed first.
		name (required)
		Identifies the DNS name or IP address of the host. This attribute may also point to a directory on the host.
		port (optional)
		Identifies the TCP port through which traffic to and from the host will pass. The port used is dependent on the protocol used. The default port is 80.
		proto (optional)
		Identifies the communication protocol that is used to fetch content from the host. HTTP is the only supported protocol.
		user (optional)
		Identifies the secure login used to access the host.
		password (optional)
		Identifies the password for the user account required to access the host server.
			`<CdnManifest>`
			`<server name=“origin-ser ver”>`
			`<host name=”www.name.com” proto="http" port="80" />`
			`</server>`
			`<item cdn-url= “logo.jpg” server=”originserver” src= “images/img.jpg” type=“prepos” playserver=“http” ttl=300/>`
			`</CdnManifest>`

Table 3-2 CDN Manifest File Syntax (continued)

Manifest Tag	Description	Options	Syntax Example

<options/>	Optional.
	
	The <options/> tag is a manifest designation that allows you to specify global settings for the hosted domain using the predefined attributes described in the paragraphs that follow.
	
	The <options> tag is enclosed within the <CdnManifest> tags and specifies at least one global setting for the hosted domain. When omitted, default values or <item>-level equivalents are used.
	
	If parameters are defined in both the manifest file <options> tag and the <item> tag for a particular piece of content, the <item>-level designation takes precedence.
		manifest-id (optional)
		
		Specifies a unique, numeric identifier for the manifest file that is used to distinguish it from other manifest files on your CDN.
		
		noRedirectToOrigin (optional)
		
		When set to *false*, this attribute allows the CDN to redirect requests for a content item to the origin server if it has not been pre-positioned yet.
		
		When set to *true*, this attribute prevents the CDN from redirecting content to the origin server if it has not been pre-positioned on the hosted domain cache.
			`<CdnManifest>`
			`<server name=“origin-ser ver”>`
			`<options timeZone="EST" noRedirectToOrigin= "true" />`
			`<host name=”www.name.com” proto="http" port="80" />`
			`</server>`
			`<item cdn-url= “logo.jpg” server=”originserver” src= “images/img.jpg” type=“prepos” playserver=“http” ttl=300/>`
			`</CdnManifest>`

Table 3-2 CDN Manifest File Syntax (continued)

Manifest Tag	Description	Options	Syntax Example

<options />		**timeZone** (optional)
		
		Specifies the time zone that is used by all content items and item groups on the hosted domain.
		
		When not specified, the default timezone, Greenwich Mean Time (GMT), is used.
		
		See Appendix B, "CDN Supported Time Zones," for a list of supported time zone abbreviations.
		
		clearlog (optional)
		
		Determines whether or not the hosted domain log file is purged whenever a new manifest file is received. By default, this option is set to false.
		
		When set to false, the CDN software continues to append log file entries to the existing log file even after the manifest has changed.
		
		When set to true, hosted domain log file is purged when a new manifest is received.
			`<CdnManifest>`
			`<server name="origin-ser`
			`ver">`
			`<options` **`timeZone="EST"`**
			`clearlog= "true"` `/>`
			`<host`
			`name="www.name.com"`
			`proto="http"`
			`port="80" />`
			`</server>`
			`<item cdn-url=`
			`"logo.jpg"`
			`server="originserver"`
			`src= "images/img.jpg"`
			`type="prepos"`
			`playserver="http"`
			`ttl=300/>`
			`</CdnManifest>`

Table 3-2 CDN Manifest File Syntax (continued)

Manifest Tag	Description	Options	Syntax Example

<item />	Required.
	
	The <item> tag names a single piece of content on the hosted domain, for example, a graphic, MPEG video, or RealAudio sound file.
	
	Content items may be listed individually, or items with shared attributes may be grouped using the <item-group> tag.
	
	The <item> tag must be enclosed within <CdnManifest> tags and may also be enclosed within <item-group> tags.
	
	Each manifest can contain a maximum of 10,000 items.
		src (required)
		
		Relative path to the content item on the origin server, starting from the host URL.
		
		When no *cdn-url* value is specified, the *src* attribute is used in the published content URL.
		
		type (optional)
		
		Indicates how the content should be handled. Two options are supported:
		
		• **prepos—**Content should be pre-positioned on the hosted domain. This is the default value.
		
		• **live—**Content is a live broadcast and cannot be pre-positioned. When the type is *live*, the playserver must be either *real* or *wmt*.
		
		alternateUrl (optional)
		
		Names an absolute path to a content file (for example, an error message page) that is used if the src (source) location is invalid.
			`<CdnManifest>`
			`<server name=“origin-ser`
			`ver”>`
			`<host`
			`name=”www.name.com”`
			`proto="http"`
			`port="80" />`
			`</server>`
			`<item cdn-url=`
			`“logo.jpg”`
			`server=”originserver”`
			`src= “images/img.jpg”`
			`type=“prepos”`
			`playserver=“http”`
			`ttl=300/>`
			`</CdnManifest>`

Table 3-2 CDN Manifest File Syntax (continued)

Manifest Tag	Description	Options	Syntax Example

<item />		**cdn-url** (optional)
		
		Relative location of the content on the Content Engine.
		
		It is possible to use wildcard (*) values in the manifest file only when the content item is live content (type = "live"). The published content URL must point to the live content item on the live server.
		
		The value supplied for the *cdn-url* attribute becomes one part of the published request URL that end users see and link to. If no *cdn-url* value is supplied, the *cdn-url* is set to the *src* attribute.
			`<CdnManifest>`
			`<server name="origin-server">`
			`<host name="www.name.com" proto="http" port="80" />`
			`</server>`
			`<item` **`cdn-url= "images/logo.jpg"`** `server="originserver" src= "images/img.jpg" type="prepos" playserver="http" ttl=300/>`
			`</CdnManifest>`

Table 3-2 CDN Manifest File Syntax (continued)

Manifest Tag	Description	Options	Syntax Example

<item /> (continued)	(continued)	**noRedirectToOrigin** (optional)
		
		When set to *false*, allows the CDN to redirect requests for a content item to the origin server if the content item is not yet pre-positioned at the location specified by the *src* attribute tag.
		
		When set to *true*, prevents the CDN from redirecting content to the origin server if it cannot be found in the hosted domain cache
		
		expires (optional)
		
		Designates a time in yyyy-mm-dd hh:mm:ss format after which the content item will no longer be served from the CDN.
		
		All dates and times are interpreted as being local for the Content Engine.
		
		As long as the *expires* attribute designates a time in the future, the content item will continue to be served from the CDN.
			`<CdnManifest>`
			`<server name=“origin-ser ver”>`
			`<host name=”www.name.com” proto="http" port="80" />`
			`</server>`
			`<item cdn-url= “logo.jpg” server=”originserver” src= “images/img.jpg” type=“prepos”` **`playserver=“http”`** `ttl=300/>`
			`</CdnManifest>`

Table 3-2 CDN Manifest File Syntax (continued)

Manifest Tag	Description	Options	Syntax Example
<item /> (continued)	(continued)	**playserver** (optional) Names the server that will be used to play this media item. When specified, this value overrides any content mapping in the <playServerTable> area. Valid playservers are: • real (RealServer) • wmt (Windows Media Services) • qtss (QuickTime Server) • http The CDN software supports live streaming only on the RealServer and Windows Media Services platforms. **prefetch** (optional) Designates a time in yyyy-mm-dd hh:mm:ss format after which a content item should be retrieved from the origin server and repositioned on the Content Engine. Date and time are local to the Content Engine.	

Table 3-2 CDN Manifest File Syntax (continued)

Manifest Tag	Description	Options	Syntax Example

<item /> (continued)	(continued)	**serve** (optional)
		
		Specifies the date and time after which grouped content items can be requested from the Content Engine in yyyy-mm-dd hh:mm:ss format.
		
		The default time zone is GMT unless otherwise specified using the <options> tag.
		
		server (optional)
		
		Identifies the server from which the content item will be fetched. The name specified must match the <server name = ““> value.
		
		If omitted, the origin server of the hosted domain is assumed to be the server.
		
		This attribute can also be used within an <item-group>.
			`<CdnManifest>`
			`<server name=“origin-ser`
			`ver”>`
			`<host`
			`name=”www.name.com”`
			`proto="http"`
			`port="80" />`
			`</server>`
			`<item cdn-url=`
			`“logo.jpg”`
			`server=”originserver”`
			`src= “images/img.jpg”`
			`type=“prepos”`
			`playserver=“http”`
			`ttl=300 />`
			`</CdnManifest>`

Table 3-2 CDN Manifest File Syntax (continued)

Manifest Tag	Description	Options	Syntax Example

<item /> (continued)		**streamProperty** (optional)
		
		For use with Windows Media files (WMA, WMV, and ASF) only.
		
		Specifies one or more file attributes that are displayed in the Windows Media Player when the file is played.
		
		The supported attributes are:
		• Abstract
		• Title
		• Author
		• Copyright
		
		ttl (optional)
		
		The *ttl* attribute identifies the period, in minutes, for which the content item should be controlled for changes before release. The default value is 30 minutes.
			`<CdnManifest>`
			`<server name=“origin-ser ver”>`
			`<host name=”www.name.com” proto="http" port="80" />`
			`</server>`
			`<item cdn-url= “q4results.asf” server=”originserver” src= “video/q4results.asf” type=“live” playserver=“wmt”` **`streamProperty = "author='paul roberts' title='Cisco Q4 Results' copyright='Cisco 02'`**`"` **`ttl=300`**`/>`
			`</CdnManifest>`

Table 3-2 CDN Manifest File Syntax (continued)

Manifest Tag	Description	Options	Syntax Example

<contains />	Optional.
	
	Identifies other pieces of content that are embedded within the content item currently being described. For example, the components of a SMIL-format file
	
	Requests for an item using <contains /> links are only accepted after the CDN determines that all dependent content items are present in the cache.
	
	The <contains /> tag must be enclosed within the <item> </item> tags.
		cdn-url (required)
		
		Identifies the public location for the content item on the hosted domain.
		
		The value supplied for the *cdn-url* attribute becomes one part of the published request URL that end users see and link to. If no *cdn-url* value is supplied, the *cdn-url* is set to the *src* attribute.
			`<CdnManifest>`
			`<server name=“origin-ser ver”>`
			`<host name=”www.name.com” proto="http" port="80" />`
			`</server>`
			
			`<item cdn-url="house.rp" src="house/house.rp">`
			`<contains cdn-url="img08.jpg"/>`
			`<contains cdn-url="img09.jpg"/>`
			`<contains cdn-url="img1.jpg"/>`
			`<contains cdn-url="img2.jpg"/>`
			`<contains cdn-url="img3.jpg"/>`
			`</item>`
			`</CdnManifest>`

Table 3-2 CDN Manifest File Syntax (continued)

Manifest Tag	Description	Options	Syntax Example

<item-group> </item-group>	Optional.
	
	The <item-group> tag names a collection of content items with shared attributes on the hosted domain, for example, a group of graphics on the same host with the same Time To Live (TTL) value. If an attribute is specified in both the <item-group> tag and separately for a grouped content item, the <item>-level attribute takes precedence over the group attribute.
	
	The <item-group> tag must be enclosed within <CdnManifest> tags and contain two or more <item> tags identifying content items that share the attributes named by the <item-group> tag.
		<item /> (required)
		
		Names a content item. Any <item-group> must have at least one content item and no more than 10,000 total items named in a single manifest file.
		
		alternateUrl (optional)
		
		An alternate absolute path to a single content file, for example an error page, that will be used in the place of the content item if the src (source) location is invalid.
		
		cdnPrefix (optional)
		
		Names a directory or partial path that is placed in the published request URL immediately before the value named by the item's *cdn-url* attribute.
			`<CdnManifest>`
			`<server name=“orgsv”>`
			`<host`
			`name=”www.name.com”`
			`proto="http"`
			`port="80" />`
			`</server>`
			`<item-group`
			`server="web-server"`
			`type="prepos" ttl="300">`
			`<item cdn-url="wild.ram"`
			`src="wildlife.ram"/>`
			`<item cdn-url="gg.mpeg"`
			`src="GoldenGate.mpeg”/>`
			`<item cdn-url="jbg.mp3"`
			`src="JohnnyBeGood.mp3"`
			`/>`
			`<item cdn-url="paul.asx"`
			`src="fin371k.asx" />`
			`</item-group>`
			`</CdnManifest>`

Table 3-2 CDN Manifest File Syntax (continued)

Manifest Tag	Description	Options	Syntax Example
<item-group> </item-group> (continued)	(continued)	**noRedirectToOrigin** (optional) When set to *false*, allows hosted domain Content Engines to redirect requests to the origin server if they cannot be found at the location specified by the *src* attribute. When set to *true*, prevents hosted domain Content Engines from redirecting requests to the origin server if they cannot be found in the hosted domain cache. **playserver** (optional) Names the server that will be used to play the grouped content items. This value overrides any content mapping in the <PlayServerTable>. Valid playservers are: • real (RealServer) • wmt (Windows) • qtss (QuickTime) • http Live streaming is only supported for real and wmt. See Table 3-3 for a list of files supported by each playserver.	

Table 3-2 CDN Manifest File Syntax (continued)

Manifest Tag	Description	Options	Syntax Example

<item-group>
</item-group>
(continued)
	(continued)	**server** (optional)
		
		The origin server from which the grouped content items will be fetched. The name specified must match the server name in the <server> tag.
		
		srcPrefix (optional)
		
		Names a directory or partial path that is used to build a directory structure for the items in the content item group. The value specified by the *srcPrefix* attribute is placed in the published request URL before the item’s *src* attribute.
		
		streamProperty (optional)
		
		For use with Windows Media files (WMA, WMV, and ASF) only. Specifies one or more file attributes that are displayed in the Windows Media Player when the file is played. The supported attributes are:
		
		• Abstract
		• Title
		• Author
		• Copyright
			`<CdnManifest>`
			`<server name=“orgsv”>`
			`<host`
			`name=”www.name.com”`
			`proto="http"`
			`port="80" />`
			`</server>`
			`<item-group`
			`server="web-server"`
			`type="prepos"` **`ttl="300"`**`>`
			`<item cdn-url="wild.ram"`
			`src="wildlife.ram"/>`
			`<item cdn-url="gg.mpeg"`
			`src="GoldenGate.mpeg”/>`
			`<item cdn-url="jbg.mp3"`
			`src="JohnnyBeGood.mp3"`
			`/>`
			`<item cdn-url="paul.asx"`
			`src="fin371k.asx" />`
			`</item-group>`
			`</CdnManifest>`

Table 3-2 CDN Manifest File Syntax (continued)

Manifest Tag	Description	Options	Syntax Example

<item-group> </item-group> (continued)	(continued)	**ttl** (optional)
		Identifies the period, in minutes, for which the grouped content item should be controlled for changes before release. The default value is 30 minutes.
		type (optional)
		Indicates how each grouped content item should be handled. Two options are supported:
		• **prepos**—Content should be pre-positioned on the hosted domain. This is the default value, which is applied in the event that no type is specified.
		• **live**—Content is a live broadcast and cannot be pre-positioned.
			`<CdnManifest>`
			`<server name=“orgsv”>`
			`<host`
			`name=”www.name.com”`
			`proto="http"`
			`port="80" />`
			`</server>`
			`<item-group`
			`server="web-server"`
			`type="prepos"` `ttl="300">`
			`<item cdn-url="wild.ram"`
			`src="wildlife.ram"/>`
			`<item cdn-url="gg.mpeg"`
			`src="GoldenGate.mpeg”/>`
			`<item cdn-url="jbg.mp3"`
			`src="JohnnyBeGood.mp3"`
			`/>`
			`<item cdn-url="paul.asx"`
			`src="fin371k.asx" />`
			`</item-group>`
			`</CdnManifest>`

Table 3-3 Supported Media File Formats Grouped by Manifest File Content Type

Extension	Supported Formats	Notes
http	• Audio Visual Interleaved (AVI) • Graphics Interchange Format (GIF) • Hypertext Markup Language (HTML, HTM) • Joint Photographic Experts Group (JPG) • Microsoft PowerPoint (PPT) • Microsoft Word (DOC) • Motion Picture Experts Group (MPEG, MPG) • MPEG Audio Layer 3 (MP3) • Portable Document Format (PDF) • QuickTime Movie (MOV) • ASX	The content item will be handled by an HTTP server; this tag is used for content that cannot be streamed by any of the servers listed in the previous section, for example, Adobe PDF, PostScript (PS), and MPG files.
media	• AVI • GIF • HTML, HTM • JPG • PPT • DOC • MPEG, MPG) • MP3 • PDF	This is the default value used by the Cisco Internet CDN Software. Use the media tag when no playserver is specified to handle a content item; the linked item may be a pre-positioned or a live content item.
qtss	• QuickTime (QT) • MOV	The content item will be handled by the Apple QuickTime Darwin Streaming Server.

Table 3-3 Supported Media File Formats Grouped by Manifest File Content Type (continued)

Extension	Supported Formats	Notes
real	• RealAudio (RA) • RealMedia (RM) • RealPix (RP) • RealText (RT) • Synchronized Container Format (SMIL)	The content item will be handled by RealServer.
wmt	• ASF (includes WMA and WMV) • ASX	The content item will be handled by Windows Media Services.

Sample Manifest File

Example 3-2 shows a functional manifest file. Use this sample as a model when creating or troubleshooting your own manifest files.

Example 3-2 Manifest File Containing Pre-positioned and Live Content

```
<?xml version="1.0"?>
<!DOCTYPE CdnManifest SYSTEM "CdnManifest.dtd">
<CdnManifest>

<!--playserver mappings for content type-->
<!--these are consulted after URL, item, then item-group mappings-->
<playServerTable>
<playServer name="real">
    <contentType name="application/x-pn-realaudio" />
    <contentType name="application/vnd.rn-rmadriver" />
    <extension name="rm" />
    <extension name="ra" />
    <extension name="rp" />
    <extension name="rt" />
    <extension name="smi" />
  </playServer>
<playServer name="qtss">
    <contentType name="application/qt-foo" />
    <extension name=".mov" />
  </playServer>
<playServer name="wmt">
    <extension name="asx" />
    <extension name="asf" />
    <extension name="avi" />
  </playServer>
<playServer name="http">
    <contentType name="application/pdf" />
    <contentType name="application/postscript" />
    <extension name="pdf" />
    <extension name="ps" />
  </playServer>
</playServerTable>

<!--manifest file options-->
<options timeZone="EST" />

<!--origin servers -->
<server name="origin-web-server">
  <host name="http://www.cisco.com/media"/>
 <host name= “192.168.3.15” />
</server>

<!--secondary origin server that also runs RealServer -->
<server name= “live-streamer”>
 <host name=”192.168.3.15” />
<server/>
```

```
<!--grouped content items-->
<item-group
      server="origin-web-server"
      type="prepos"
      ttl="300">
 <item cdn-url="newHQpresentation.rm" src="newHQpresentation.rm" />
  <item cdn-url="animatedlogo.mpg" src="animlogo.mpg" />
  <item cdn-url="companytheme.mp3" src="cotheme.mp3" />
  <item cdn-url="newHQlayout.avi" src="newHQ.mov" />
</item-group>]

<item-group
      server="origin-web-server"
      type="prepos"
      ttl="300">
 <item cdn-url="roadmap.asf" src="prodroadmap.asf" />
  <item cdn-url="roadmaptalk.wma" src="rmtalk.wma" />
  <item cdn-url="newHQlayout.avi" src="newHQ.avi" />
</item-group>]

<!--non-grouped content items-->
<item cdn-url="q3presentation.rm"
  src="present/q3presentation.rm"/>
<item cdn-url="q1.smi"
  src="house/house.smi">
    <contains cdn-url="q1presentation.rm"/>
    <contains cdn-url="q1.rt"/>
  </item>

<!--live content-->
<item-group server="live-streamer" type="live">
    <item src="/encoder/live.rm" cdn-url="/present/live/*" />
</item-group>
</CdnManifest>
```

Validating Manifest File Syntax

Because correct and accurate manifest file syntax is vital to the proper deployment of your web site content on the CDN, Cisco makes a manifest file syntax checker available, at no cost, to its customers. This command-line based utility can be used to proof the manifest files you have created for your hosted domain.

When run, the manifest validator reviews each line of your manifest file, identifying syntax errors where they exist and determining whether or not the manifest is valid and ready for use in importing content to your hosted domain. The results of the manifest validator's review of the manifest file are output to a text file in a location that you name.

Obtaining the Manifest File Validator

The manifest validator is designed to run in both the Windows (95/98, NT, 2000 and XP) environment and the Linux (RedHat 6.2) environments.

Cisco provides the manifest validator utility free of cost through its website, Cisco.com.

To obtain a copy of the manifest validator:

Step 1 Launch your preferred web browser and point it to:

```
http://www.cisco.com/cgi-bin/tablebuild.pl/cdn-sp
```

Step 2 When prompted, log in to Cisco.com using your designated Cisco.com username and password.

The Cisco Internet CDN Software download page appears, which lists the available software updates for the Cisco Internet CDN Software product.

Step 3 Locate the file named *manifest-validator.zip*. This is a ZIP archive containing the files for the manifest validator utility.

Step 4 Click the link for the *manifest-validator.zip* file. The download page appears.

Step 5 Click the Software License Agreement link. A new browser window will open displaying the license agreement.

Step 6 After you have read the license agreement, close the browser window displaying the agreement and return to the Software Download page.

Step 7 Click the filename link labeled Download.

Step 8 Click **Save to file** and then choose a location on your workstation to temporarily store the zip file.

Step 9 See the "Installing the Manifest File Validator" section on page 3-41 for instructions on installing the validator utility.

Installing the Manifest File Validator

Before installing the manifest validator, you must first install the Java (TM) 2 Runtime Environment, version 1.2 or 1.3 on your workstation. The Java 2 Runtime Environment (JRE) contains the Java virtual machine, runtime class libraries, and Java application launcher. These components are necessary to run the Cisco manifest validator utility.

If you are using an earlier version of the JRE on your workstation, please install either version 1.2 or 1.3. You can download the latest version of the JRE along with instructions for installing it from the following web site:

http://java.sun.com

After you install the JRE, use the following instructions to install, then run the Cisco manifest validator utility:

Step 1 Create a directory for the manifest validator on your local drive. For example:

c:\manifest

Step 2 Locate the manifest validator archive. This file is named *manifest-validator.zip* and was provided to you by your service provider.

Step 3 Unzip the manifest validator files into the directory you created.

Step 4 Verify that all manifest validator files are present in the manifest directory you created. Table 3-4 contains a list of the sources files that constitute the manifest validator utility and their purpose.

Table 3-4 Manifest Validator Source Files

Validator File Name	Purpose
xerces.jar	syntax parser
manval.zip	standalone manifest validator
CdnManifest.dtd	document type definitions for the CDN manifest file

Table 3-4 Manifest Validator Source Files (continued)

Validator File Name	Purpose
PlayServerTable.dtd	document type definitions for CDN PlayServerTables, used to define media servers (Real, Windows Media) for the CDN
validate	shell script used to run the validator
validate.bat	batch file to run the validator

Running the Manifest File Validator

After you have installed the Java 2 Runtime Environment and the Cisco manifest validator program files, you are ready to run the validator on a manifest file that you have created.

If you have not created a manifest file yet, see the "Pre-Positioning Web Site Content" section on page 3-11 and Appendix A, "Sample Manifest File Scripts."

The manifest file validator can be run in one of two modes:

- Default mode—the manifest validator checks the syntax of your manifest file to make sure that source files are named for each content item in the manifest.
- Check size mode—the manifest validator checks the syntax of your manifest file to make sure that source files are named for each content item in the manifest, then follows the URL for each content item to verify that the content is placed correctly and, if possible, to determine the size of the item. A special switch (-s) is used to run the validator in "check size" mode.

When running the manifest file validator, you are required to input the following information:

- The name and location of your manifest file, expressed either as a network file location <file> or a valid Internet URL <url>
- The name and location of the manifest validator's output file <output>.

- (Optional.) When running the manifest file validator in *check size* mode, the length of time <seconds> that the manifest file validator should attempt to confirm the existence of content named in the manifest.

To run the manifest validator utility:

Step 1 If you are running Windows, open a command prompt. Otherwise, proceed to Step 2.

Step 2 Change directories to the program directory for the manifest validator. For example, if you are running Windows, you would enter the following at the command prompt:

```
C:\>cd manifest
C:\manifest>
```

Step 3 Run the manifest validator as follows:

- If you are running Windows, enter the validate command and provide either a path and file name or URL pointing to your manifest file in the following format:

```
validate -f [<file> | -u <url>] -o <output> [-s <seconds>]
```

- If you are running Linux, enter the following commands, providing either a path and file name or URL that points to your manifest file in the following format:

```
chmod u+x validate
validate -f [<file> | -u <url>] -o <output> [-s <seconds>]
```

After you execute the validator, text output is displayed, which indicates that the validator is running.

Step 4 Wait until the following message is displayed, which indicates that the validator has completed processing the manifest file you pointed to:

```
Finish parsing /<manifest_file_name>.xml
```

Step 5 Locate the output file in the location you specified and review it for errors.

The final lines of the manifest file validator’s output will indicate whether or not the manifest is valid or not. For example, a valid manifest file output might read:

```
number of manifest warnings: 1
number of manifest errors: 0
manifest syntax is CORRECT
finish parsing
```

In this instance, one non-fatal syntax irregularity was located, but the manifest file as found to be syntactically correct. This file could be transferred to your service provider and used to deploy web site content to your CDN.

The output file for an invalid manifest file will list the number of errors and warnings issued. For example:

```
number of manifest warnings: 1
number of manifest errors: 1
manifest syntax is INCORRECT
finish parsing
```

See the "Understanding Manifest File Validator Output" section next for detailed information that will help you understand the manifest file validator results.

Understanding Manifest File Validator Output

Your manifest file validator output file will appear in the location you specified (using the -o option) when the validator was run.

Each output file will have a similar structure and syntax and will clearly identify any errors or warning messages stemming from your manifest file syntax. Manifest files are judged by the validator either to be:

- CORRECT—possibly containing syntax irregularities, but syntactically valid and ready for deployment on a CDN
- INCORRECT—containing syntax errors and unsuitable for deployment on a CDN

Syntax Errors

The manifest file validator issues syntax errors only when the manifest file validator cannot identify a source file for a listed content item—either because it is not listed, or it is listed using improper syntax. All files containing syntax errors are marked INCORRECT.

Syntax errors are identified in the output with the ERROR label. The line number containing the error is provided, as well as the manifest attribute for which the error was issued, valid options, and the default value for that attribute. For example, the following error appears in Example 3-3:

```
ERROR: japan.xml:13:Skip item because src is not defined.
```

In this example:

- *japan.xml* is the manifest file name.
- *13* is the manifest file line number where the error occurs.
- *src* is the manifest file attribute generating the warning.

Example 3-3 Manifest File Validator Output Containing Errors and Warnings

```
start parsing file japan.xml
start options
 option clearlog: false
 option rd: null
 option prepos-tag: null
 option live-tag: null
 option notFoundUrl: null
 option noRedirectToOrigin: false
 option timezone: JST
 option manifest-id: null
end options
start server
 server name: WMTServer
start host
  host name: origin.cdn-japan.com
  host proto: http
  host port: 80
  host user: ceadmin
  host password: 3kDC
  creating new hash entry for WMTServer and origin.cdn-japan.com
 end host
end server
WARNING: japan.xml:13:Attribute "src" is required and must be
specified for element type "item".
start item
 item src: null
 ERROR: japan.xml:13:Skip item because src is not defined.
end item
end CdnManifest
number of items processed: 1
number of manifest warnings: 1
number of manifest errors: 1
manifest syntax is INCORRECT
finish parsing
```

A total number of errors encountered in the manifest file is provided at the end of the validator output file.

Syntax Warnings

The manifest file validator issues syntax warnings for a wide variety of irregularities in the manifest syntax. Files containing syntax warnings may be marked CORRECT or INCORRECT, depending on whether or not syntax errors have also been issued.

Syntax warnings are identified in the output with the WARNING label. In addition to the label, the line number containing the warning is provided, as well as the manifest attribute for which the warning was issued, valid options and the default value for that attribute. For example, the following warning might appear in the output for the japan.xml manifest file:

```
WARNING: /~content/manifest/japan.xml:12:Attribute "type" with value
"vod" must have a value from the list "(prepos|live)"
```

In this example:

- *japan.xml* is the manifest file name.
- *12* is the manifest file line number where the warning was issued.
- *type* is the manifest file attribute generating the warning.
- *vod* is the offending value.
- *(prepos* | *live)* are the valid options for that attribute.

A total number of warnings encountered in the manifest file is provided at the end of the validator output file.

Repairing Manifest File Syntax

After you have identified syntax errors and warnings using the output from the manifest file validator tool, you can correct your manifest file syntax, then re-run the manifest file script on the corrected file.

To repair your manifest file:

Step 1 Open your manifest file using your preferred XML- or text-editing tool.

Step 2 Referring to your manifest file validator output, use the line numbers provided by the manifest file validator to locate the syntax violations in your manifest file.

In general, it is a good idea to review each WARNING and ERROR tag in your manifest. Some warnings, while still allowing the manifest file validator to find your manifest file syntax correct, may still be the source of problems when you deploy your web site content.

Step 3 After you have made all necessary corrections for syntax warnings and errors, save your manifest file.

Step 4 Run the manifest file through the manifest file validator again and review the validator output for errors and warnings.

Step 5 Repeat Step 1 through Step 4 until all errors and warnings have been adequately resolved and until the manifest validator labels your manifest file CORRECT.

Sample Manifest File Scripts

This appendix contains information that you can use to automate the creation of manifest files for your web site.

This appendix contains the following sections:

Overview

Two sample scripts are provided in this appendix:

- Spider—crawls over the content of an origin server and outputs a database file containing a list of URLs for all *potentially* pre-positioned or live content objects on that origin server, regardless of whether that content will eventually be pre-positioned or streamed live from the hosted domain.

- Manifest—reads the database file output by the Spider script and, using rules set out by the content provider, produces an XML-format manifest file containing the URLs of just those items or types of content that a content provider wants to make available to users through a hosted domain.

These scripts shipped with your CDN software and can serve as the basis for your own automation scripts.

Installing PERL on Your Workstation

You need to have PERL installed on your workstation prior to working with or running the Spider or Manifest scripts. It may also be useful to have a PERL compiler available. PERL is open source software and can be downloaded for free from a variety of locations on the Internet. Refer to the Comprehensive PERL Archive Network (CPAN) at http://www.cpan.org, or http://www.perl.com.

Obtaining the Scripts

The Spider and Manifest scripts can be obtained from Cisco.com using the same procedure that is used to obtain updated versions of the Cisco Internet CDN Software.

To obtain the Manifest and Spider scripts from Cisco.com:

Step 1 Launch your preferred web browser and point it to:

`http://www.cisco.com/cgi-bin/tablebuild.pl/cdn-sp`

Step 2 When prompted, log in to Cisco.com using your designated Cisco.com username and password.

The Cisco Internet CDN Software download page appears, listing the available software updates for the Cisco Internet CDN Software product.

Step 3 Locate the file named *manifest-tools.zip*. This is a ZIP archive containing both the Manifest and Spider PERL scripts.

Step 4 Click the link for the *manifest-tools.zip* file. The download page appears.

Step 5 Click the Software License Agreement link. A new browser window will open displaying the license agreement.

Step 6 After you have read the license agreement, close the browser window displaying the agreement and return to the Software Download page.

Step 7 Click the filename link labeled Download.

Step 8 Click **Save to file** and then choose a location on your workstation to temporarily store the zip file containing the scripts.

Step 9 Use your preferred unzip program to unpack the scripts to a location on your workstation or your network.

After you have unzipped the scripts, you are ready to begin using them to build manifest files for your website. See the "Listing Web Site Content Using the Spider Script" section on page A-3 and the "Selecting Live and Pre-position Content Using the Manifest Script" section on page A-7 for instructions on running the scripts.

Listing Web Site Content Using the Spider Script

This section contains information on the following topics:

- Spider Script Syntax Guidelines, page A-5
- Combining Spider Data, page A-7
- Customizing the Spider Script, page A-7

In the simplest scenario, the spider is pointed to the address of an origin server and given the name of a database (.db) file into which it will place any valid URLs it discovers on that site. For example, if you wanted to analyze the contents of www.cisco.com for content that might be pre-positioned, you would issue the following command:

```
spider --start=www.cisco.com --db=ciscocontent.db
```

Limiting Scope

But spidering the entirety of www.cisco.com might take hours and produce much more information than you are interested in. What if you want to limit your review of an origin server to just a particular part of that server? The Spider script contains a variety of tools that enable you to limit as well as broaden the scope of a spider's action.

For example, to limit the spider's search of www.cisco.com to just that part of the server containing product-related support information, you could enter the following command:

```
spider --start=www.cisco.com/public/support/ --db=ciscocontent.db
```

Broadening Scope

Or, to ask the spider to follow links from www.cisco.com to the Cisco networking professionals forum, you could enter the following spider command:

```
spider --start=www.cisco.com --allow=forums.cisco.com
--db=ciscocontent.db
```

Re-spidering Servers

In addition to covering new origin servers, the Spider script can also be run on sites that have already been analyzed and that contain links into the CDN. When spidering a server that has already been analyzed, you use the **--hd** keyword to specify the name of hosted domain on which content from the origin server will be hosted, and the **--map** keyword to provide mapping information between URLs on the origin server and on the Internet CDN.

For example, the following commands will trace the content mapped to the /support area on the hosted domain www.hosted.cisco.com back to its origins in in the support area of www.cisco.com:

```
--start=http://www.cisco.com/public/support/tac/home.html
--hd=www.hosted.cisco.com
--map=http://www.cisco.com/public/support/tac/=/support
--db=ciscocontent.db
```

In each of these examples listed, the Spider analyzes the URL of each piece of content on the origin server or in that area of the origin server that has been targeted and applies filters to them that incorporate the parameters supplied when the Spider was run and identify potential pre-positioning or live streaming candidates. If the URL matches the pattern provided by the Spider, it is accepted and its URL is recorded in the database being created by the Spider. If the pattern does not match, the content is rejected and the Spider moves on.

Spider Script Syntax Guidelines

The Spider script accepts the following syntax:

spider {**--start=***origin_server_url*
[**--allow=***allowed_url* | **--depth=***number* | **--file=***filename* |
{**--hd=***hosted_domain_name* **--map=**{*origin_server_url_prefix*=*cdn_prefix*} } |
--limit=*number* | **--prefix**=*url_prefix* | **--reject=***disallowed_url* |]
--db=*database_name***.db**}

Table A-1 Spider Script Keywords

Keyword	Description	Syntax
--start	Names the location (URL) of the origin server that will be analyzed.	`--start=www.cisco.com`
--db	Names the database file in which content URLs from the origin server and any allowed locations will be placed.	`--db=ciscocontent.db`
--allow (optional)	Names a location other than that specified using the start keyword that will be accepted when it is found in URLs.	`--allow=forums.cisco.com`
--depth (optional)	Causes the Spider script to stop after following links a specified number of levels deep on the origin server.	`--depth=6`

Table A-1 Spider Script Keywords (continued)

Keyword	Description	Syntax
--file (optional)	Causes the Spider script to read its commands from a specified rules file, one line at a time.	`--file=cisco-rules.cfg`
--hd (optional)	Identifies a hosted domain on your CDN as the hosted domain for the content being spidered. Used with the **--map** keyword for mapping content from the CDN back to the origin server.	`--hd=www.hosted.cisco.com`
--map (optional)	Causes the Spider script to substitute the second URL prefix (appearing after the second =) for the first in any URLs from the origin server, or substitute the first prefix for the second when re-spidering content an origin server.	`--map=http://www.cisco.com/public/support/tac/=/support`
--limit (optional)	Causes the Spider script to stop after retrieving a specified number of pages from the origin server. The default is 100. Specifying 0 sets no limit for the number of pages retrieved.	`--limit 1000`
--prefix (optional)	Specifies a URL prefix which, when it is encountered, will be accepted by the Spider.	`--prefix=http://www.cisco.com/partners/CDN/`
--reject (optional)	Names a location that will be rejected when it is found in URLs.	`--reject=cgi-bin`

Combining Spider Data

What if you ran the Spider script on two separate locations on an origin server, but would like to combine the content into one database from which a manifest file will be generated?

The data output by the Spider can easily be combined—just open the *.db file containing the data you want to move, select that data, and copy it. Next, open the *.db file you want to serve as the merged file, locate the end of the file, and paste the data you copied into it.

The Manifest script can now be run on the merged data.

Customizing the Spider Script

Because the Spider script anticipates certain platforms and scenarios that might not correspond to your own web site configuration, Cisco provides you with the PERL source code for the Spider script, which you can modify to suit your own needs.

See the "Spider Script Source" section on page A-12 to review the source code for the Spider script.

Selecting Live and Pre-position Content Using the Manifest Script

Whereas the Spider script is used to gather a list of potential hosted content from an origin server, the Manifest file is where you will cull through all the information gathered by the Spider and decide which content you will actually import to the CDN for placement on a hosted domain.

This section contains information on the following topics:

- Pre-Positioned Versus Live Content, page A-8
- Manifest Script Syntax Guidelines, page A-8
- Customizing the Manifest Script, page A-11

Pre-Positioned Versus Live Content

The Manifest script distinguishes between content that needs to be pre-positioned and live, streamed content that, by definition, cannot be pre-positioned.

Using the **prepos** command, you identify and pre-position all content that meets criteria that you specify. For example, to pre-position all image files from cisco.com larger than one megabyte, you would enter the following command:

```
manifest --prepos=’type(image/*) and size > 1000k’
--db=ciscocontent.db --xml=cisco.xml
```

Using the **live** command, you identify the URLs of live content. Unlike pre-positioned content, live content cannot be identified by information stored in the header, so you will need to devise a method of locating live content based solely on information contained in the URL of that content. For example, you might identify streamed content with the following command:

```
manifest --live=‘match(rtsp://*)’
```

Manifest Script Syntax Guidelines

manifest {[**--file=***filename* | **--live=***’keyword_comparison’* | **--prepos=**‘*keyword_comparison*’ | **--set=**‘*attribute=value* : *keyword_comparison*’ | **--playservertable=***filename* | **--map=**{*origin_server_url_prefix*=*cdn_prefix*}] **--db=***database_name***.db** **--xml=***manifest_file_name***.xml**}

Table A-2 Manifest Script Keywords

Keyword	Description	Syntax
--file	Causes the Manifest script to read its commands from a specified rules file, one line at a time.	`--file=ciscocontent.cfg`
--live	Marks content URLs in the database file that match the terms of the keyword comparison as *live* (type="live") content in the manifest file.	`--live='match(rtsp://*)'`
--prepos	Marks content URLs in the database file that match the terms of the keyword comparison as pre-positioned content (type='prepos') in the manifest file.	`--prepos='type(image/jpg) and size > 1000k'`
--set	Sets the specified attribute to the value provided for all content items with URLs in the database file that match the keyword comparison.	`--set='ttl=10000 : match(*/urgent/*)'`
--playservertable	Adds the playserver table in the specified file to the manifest file. Playserver tables map MIME content types and filename extensions to specific server types to use (for example, "real" or "wmt") for the content in a specific hosted domain. See the "Manifest File Structure and Syntax" section on page 3-16 for more information on the <playServerTable> attributes.	`--playservertable=info.txt`

Table A-2 Manifest Script Keywords (continued)

Keyword	Description	Syntax
--map	Causes the Manifest script to substitute the second URL prefix (appearing after the second =) for the first in any URLs from the origin server.	`--map=http://www.cisco.com/public/support/tac/=/support`
--db	Names the database file in which content URLs from the origin server and any allowed locations are located. This file provides the data that the Manifest script analyzes.	`--db=ciscocontent.db`
--xml	Names the manifest file that is generated by the Manifest script.	`--xml=ciscomanifest.xml`
match	A comparison keyword that locates text in content URLs that are identical to a value that is provided.	`--prepos='match (http://forums.cisco.com/*)'`
size	A comparison keyword that identifies content named in the database file according to the specified filesize parameter (in kilobytes).	`--prepos='size >= 1000k'`
time	A comparison keyword that identifies content named in the database file according to the time since the content was last modified (in hours).	`--prepos= 'time < 72 hours'`
type	A comparison keyword that identifies content named in the database file according to its MIME type (text, application, image, and so on).	**--prepos='type(image/gif)'**

Customizing the Manifest Script

Because the Manifest script anticipates certain platforms and scenarios that might not correspond to your own web site configuration, Cisco provides you with the PERL source code for the Manifest script, which you can modify to suit your own needs.

See the "Manifest Script Source" section on page A-26 to review the source code for the Manifest script.

Creating a Rules File for the Spider and Manifest Scripts

When using the Spider and Manifest scripts on a large web server, the parameters and rules you set for your scripts may be numerous and complex. When this is the case, it may make more sense to create a file containing all your instructions to the scripts that you can then simply point to than having to type a long series of commands time and again.

Using a rules file makes it easy to re-run the Spider and Manifest scripts, and be confident that the scripts are receiving identical commands each time. In addition, the same commands file can be read by both the Manifest and Spider scripts without generating incorrect output; the Spider script simply ignores commands for the Manifest script, and vice versa.

To create a rules file for the Spider and Manifest scripts to use:

Step 1 Open your preferred text editor.

Step 2 Enter your commands one at a time and each on its own line. Each line of your rule file is sent to the scripts as a single argument.

For example, a rules file for the Cisco web site might read:

```
--start=www.cisco.com
--allow=forums.cisco.com
--reject=cgi-bin
--limit=0
--db=ciscocontent.db
--prepos='match(image/gif) and size > 1000k'
--xml=ciscomanifest.xml
```

Step 3 Save your file in a location relative to the Spider and Manifest scripts.

Step 4 Use the file command to run each script using your rules file. For example:

```
spider --file=cisco-rules.cfg
manifest --file=cisco-rules.cfg
```

Spider Script Source

```
#!/usr/bin/perl -w
use strict;
my @todo = ();                  # Array of urls we still have to fetch
my %seen = ();                  # Hash of urls we've fetched

use Getopt::Long;
my $limit = 100;                # Maximum number of URLs we might 
fetch.
my $depth = 0;                  # Spidering depth (0 == infinite)
my @prefix = ();
my @filters = ();               # A filter is a regexp and a bool.
# (["mit\.edu", 1], [".", 0]) means accept mit.edu urls, reject all 
others
my @start =();                  # URLs to start spidering
my $db = "";                    # The filename to write the database 
to.
my $proxy = "";                 # The proxy to use when making HTTP 
requests

# These allow us be intelligent about spidering sites that have 
already
# been rewritten to contain links to the hosted domain.
my @map = ();                   # origin to cdn-url mappings
my $hd = "";                    # The hosted domain

my $debug = 0;                  # Print extra debugging info?

# Return an array containing each line from a file.
# Used by the --file option to allow stuffing @ARGV with args from a 
file.
#  '#' until end of line is a comment character (ie it is not 
returned)
#  whitespace is stripped from the beginning and end of lines
#  empty lines (or just comments and/or whitespace) are ignored
sub lines ( $ ) {
  my ($filename) = @_;
  open (F, "< $filename") or die "$filename: $!\n";
  my @lines = map { s/\#.*//g; s/\s*(\S*)\s*/$1/; $_ || (); } <F>;
  close F or die $!;
  return @lines;
}

# We want spider and manifest to be runnable from a single config
file, so
# each take all of the arguments of the other.  Naturally, these
arguments
# are ignored if they are irrelevent.  When running this way, it's
# important to use the "--start" option to name urls in spider, and
the
# "--db" option to name databases in manifest.
my $junk;
GetOptions("limit=n" => \$limit,
           "depth=n" => \$depth,
           "prefix=s" => \@prefix,
           "accept=s" => sub {my ($opt, $val) = @_; push @filters,
[$val,1]},
           "reject=s" => sub {my ($opt, $val) = @_; push @filters,
[$val,0]},
           "hd|rd=s" => \$hd,
           "map=s" => \@map,
           "db=s" => \$db,
           "proxy=s" => \$proxy,
           "start=s" => \@start,
           "<>" => sub { push @start, $_[0]; },
           # Arguments that all scripts take
           "file=s" => sub {my ($opt, $val) = @_; unshift @ARGV,
lines($val)},
           "debug!" => \$debug,
           # Arguments that are really only for 'manifest' or
'rewrite'
           "prepos=s" => \$junk,
           "live=s" => \$junk,
           "set=s" => \$junk,
           "recursive!" => \$junk,
           "playservertable=s" => \$junk,
           "xml=s" => \$junk,
           "file-map=s" => \$junk,
           "index=s" => \$junk,
           "od|origin=s" => \$junk,
           "always-rewrite=s" => \$junk,
  ) or die "Bad argument syntax\n";

my %rmap;                        # Reverse map
```

```
for my $map (@map) {
  my ($origin, $cdn) = split('=', $map);
  $rmap{$cdn} = $origin;
}

# Allow crawling to any --prefix specified paths.  They can be comma
separated.
@prefix = split(/,/,join(',',@prefix));
my %prefix;                      # Use a hash to avoid dupicates
for (@prefix) {
  $prefix{$_} = 1;
}

# Given a url, extract the "prefix".  That is, everything up to and
# including the last '/'.
sub prefix ( $ ) {
  my ($prefix) = @_;
  $prefix =~ s|(.*/).*|$1|;
  return $prefix;
}

use URI;

# The reason to do this at all is so rtsp and mms urls have methods
like
# host().
my $http_impl= URI::implementor('http');
URI::implementor('rtsp', $http_impl);
URI::implementor('mms', $http_impl);

push @todo, map { s|^|http://| unless /:/; URI->new($_)->canonical }
@start;

for my $uri (@todo) {
  next if $seen{$uri}++;
  $prefix{prefix($uri)} = 1;
}
unshift @todo, $depth if $depth; # Integers in the todo list limit
depth
my $depth_left = 1;             # Used to stop getting links if in
last round

my $prefix_re = "^(".join('|', map {quotemeta($_)} keys %prefix).")";
#warn "$prefix_re\n";
push @filters, [$prefix_re, 1]; # Accept appropriate prefixes
push @filters, [".",0];         # Reject anything that gets to the end

# Filter debugging
```

```
#for my $f (@filters) {
#  warn "$f->[0] $f->[1]\n";
#}

my %extractors = ("text/html" => \&html_extract,
                  # Real Networks formats
                  "application/smil" => \&smil_extract,
                  "image/vnd.rn-realpix" => \&rp_extract,
                  "text/vnd.rn-realtext" => \&rt_extract,
                  "audio/x-pn-realaudio" => \&list_extract,
                  "audio/x-pn-realaudio-plugin" => \&list_extract,
                  # Microsoft formats
                  "video/x-ms-asf" => \&asx_extract,
                  "audio/x-ms-wax" => \&asx_extract,
                  "video/x-ms-wvx" => \&asx_extract,
                  # Flash
                  "application/x-shockwave-flash" => \&swf_extract,
                  # JavaScript
                  "application/x-javascript" => \&js_extract,
                  # .m3u files aren't really standardized...
                  "audio/x-m3u" => \&list_extract,
                  "audio/m3u" => \&list_extract,
                  "audio/x-mpegurl" => \&list_extract,
                 );

# Web servers are often stupid.  Try to guess an extractor based on
these
# extensions if mime type doesn't work.
my %ext_extractors = (# Real networks
                      "smi" => \&smil_extract,
                      "rp" => \&rp_extract,
                      "rt" => \&rt_extract,
                      "ram" => \&list_extract,
                      "rpm" => \&list_extract,
                      # Microsoft
                      "asf" => \&asx_extract,
                      "wax" => \&asx_extract,
                      "wvx" => \&asx_extract,
                      # Flash
                      "swf" => \&swf_extract,
                      # JavaScript
                      "js" => \&js_extract,
                      # And for good measure
                      "m3u" => \&list_extract);

# Given a URI and a mime type, return the appropriate extractor if it
is a
# container type, else 0;
```

```
sub extractor ( $$ ) {
  my ($uri, $type) = @_;
  my $ext = lc($uri);
  $ext =~ s/(.*\.)//;           # Remove everything up to the last .

  # Sleezy hack, but blame Real.  They have code to differentiate .ram
  # files from .rm and .ra files instead of separate mime types.  I
really
  # don't want to suck down a multimegabyte binary file thinking it's
a ram
  # file, so bail now.
  return 0 if $ext =~ /^r[ma]$/;

  return $extractors{lc($type)} if exists $extractors{lc($type)};
  # Might want to use extention only for text/plain...
  return $ext_extractors{$ext} if exists $ext_extractors{$ext};
  return 0;
}

# HTML extractor

# The following hash is taken from HTML::LinkEtor.  I've commented out
all
# the places where links appear but they don't seem to be necessary to
view
# the page, leaving things that should be considered embedded.
# http://www.w3.org/TR/html4/ was used to determine what things meant.
# Applet and object are supported poorly -- that is the 'base'
attributes
# don't work yet, nor does 'archive'.

my %emb =
(
# a       => 'href',
 applet  => [qw(code)], #archive codebase)], unsupported for now
# area    => 'href',
# base    => 'href',
 bgsound => 'src',
# blockquote => 'cite',
 body    => [qw(background)],
# del     => 'cite',
# embed is not in w3c spec - described at
# http://home.netscape.com/assist/net_sites/embed_tag.html
 embed   => [qw(src pluginspage)],
# form    => 'action',
 frame   => [qw(src longdesc)],
 iframe  => [qw(src longdesc)],
 ilayer  => [qw(background)],
 img     => [qw(src lowsrc longdesc)], # usemap)], usemap is a local
anchor
 input   => [qw(src)], #usemap)], usemap is a local anchor
# ins     => 'cite',
# isindex => 'action',
# head    => 'profile',
 layer   => [qw(background src)],
#'link'   => 'href',
 object  => [qw(classid data)], #codebase archive usemap)],
unsupported for now
'q'      => [qw(cite)],
 script  => [qw(src)], #for)],  "for" is not in w3c spec. Unsure what
it means
 table   => [qw(background)],
 td      => [qw(background)],
 th      => [qw(background)],
# xmp     => 'href',  Deprecated, and I doubt an href is "embedded"
anyway
);

use HTML::LinkExtor;
my $ex = HTML::LinkExtor->new();
sub html_extract ( $ ) {
  my ($content) = @_;
  my (@refs,@embs);
  $ex->parse($content);
  for my $link ($ex->links) {
    my ($tag, %attr) = @$link;
  KEY:
    while (my ($key, $val) = each(%attr)) {
      if (exists $emb{$tag}) {
        for my $attr (@{$emb{$tag}}) {
          if ($attr eq $key) {
            push @embs, $val;
            next KEY;
          }
        }
      }
      push @refs, $val        # If it's not embedded, it must be a ref
    }
  }

  # Hackish.  Since js_extract is lame anyway, we're not even
bothering to
  # extract the JavaScript, just let js_extract look at the whole
thing.

  my ($js_refs, $js_embs) = js_extract($content);
  push @refs, @$js_refs;
  push @embs, @$js_embs;

  (\@refs,\@embs);
}

# Trivial list format extractor.  Assumes that URLs must be absolute
# because these formats are usually used in a way that precludes their
# interpretters from knowing the context, thus they must be absolute.
This
# has the advantage of being able to ignore "noise lines" like
--stop-- in
# Real files.
sub list_extract ( $ ) {
  my ($content) = @_;
  my @embs = grep { m|^\s*[a-zA-Z]+://| } split("\n", $content);
  ([],\@embs);
}

# JavaScript extractor can't be perfect, but we can at least check out
the
# first argument to any window.open calls.  If it's a constant
(enclosed by
# quotes), assume it's a url.

#  Furthermore, this won't extract corrcetly if there is a comma
*inside*
#  the first argument.

sub js_extract ( $ ) {
  my ($content) = @_;
  my @refs;
  while ($content =~ m/window\.open\s*\(\s*([^\)]+)\)/g) {
    my @args = split(/,/,$1);
    my $first = $args[0];
    push @refs, $1 if $first =~ /^\'([^\']*)\'$/;
    push @refs, $1 if $first =~ /^\"([^\"]*)\"$/;
  }
  (\@refs,[]);
}

# XML base extractors (smil, rp, rt, asx)
use XML::Parser;
my $xp = XML::Parser->new();
sub smil_extract ( $ ) {
  my ($content) = @_;
  my (@refs,@embs);
  my @links = ();
  $xp->setHandlers('Start' => sub {
                       shift; my $elt = shift;
                       my %attrs = @_;
                       push @refs, $attrs{href} if exists $attrs{href};
                       push @embs, $attrs{src} if exists $attrs{src};
                     });
  $xp->parse($content);
  (\@refs,\@embs);
}

# Real has a command syntax that can be used in RealText files as
something
# to do when a link is clicked.  One common use is to open a link in a
new
# window.  command() either returns its argument, or if it's argument
is
# such a command, returns the URL that it mentions.  If it's a command
that
# does not mention a URL, return an empty list.

sub command ( $ ) {
  my ($command) = @_;
  return $command unless $command =~ /command:/;
  return () unless $command =~ /command:openwindow/;
  return () unless $command =~ /,([^,\)]*)[,\)]/; # Put second
argument into $1
  my $url = $1;
  $url =~ s/\s*(\S*)\s*/$1/;    # Trim whitespace
  return $url;
}

sub rp_extract ( $ ) {
  my ($content) = @_;
  my (@refs,@embs);
  $xp->setHandlers('Start' => sub {
                       shift;
                       my $elt = shift;
                       my %attrs = @_;
                       push @refs, command($attrs{url}) if exists
$attrs{url};
                       push @embs, $attrs{name} if $elt eq "image";
                     });
  $xp->parse($content);
  (\@refs,\@embs);
}

sub rt_extract ( $ ) {
  my ($content) = @_;
  my @refs;
  $xp->setHandlers('Start' => sub {
                     shift; my $elt = shift;
                     my %attrs = @_;
                     push @refs, command($attrs{href}) if exists
$attrs{href};
                   });
  $xp->parse($content);
  (\@refs,[]);
}

sub asx_extract ( $ ) {
  my ($content) = @_;
  my (@refs,@embs);
  $xp->setHandlers('Start' => sub {
                     shift; my $elt = shift;
                     my %attrs = @_;
                     push @embs, $attrs{href} if exists $attrs{href};
                   });
  $xp->parse($content);
  (\@refs,\@embs);
}

sub bin ( $ ) {
  my ($str) = @_;
  my $num = 0;
  while ($str ne "") {
    $num *= 2;
    $num += substr($str,0,1);
    substr($str,0,1) = "";
  }
  return $num;
}

sub swf_extract ( $ ) {
  # For format info, see http://www.openswf.org/SWFfilereference.html
  my ($content) = @_;
  my (@refs,@embs);

  my $ndx = 8;                  # Start after  sig, ver and length

  # Skip a RECT.  See
http://www.openswf.org/SWFfilereference.html#RECT
  my $bits = substr($content, $ndx, 1); $ndx += 1;
  $bits = bin(unpack("B5", $bits));
  my $bytes = int(((5 + (4*$bits))+1)/8);
  $ndx += $bytes;
  $ndx += 4;                    # skip frame rate and count

  while ($ndx < length($content)) {
    my $buf = substr($content, $ndx, 2); $ndx += 2;
    $buf = unpack("S", $buf);
    my $tag = $buf >> 6;
    my $len = $buf & 0x3F;
    if ($len == 0x3f) {
      $len = substr($content, $ndx, 4); $ndx += 4;
      $len = unpack("L", $len);
    }
    if ($tag == 12) {             # DoAction
      my $action;
      while ($len) {
        my $action = substr($content, $ndx, 1); $ndx += 1; $len--;
        $action = unpack("C", $action);
        if ($action & 0x80) {
          my $sublen = substr($content, $ndx, 2); $ndx += 2; $len -=
2;
          $sublen = unpack("S", $sublen);
          $buf = substr($content, $ndx, $sublen);
          $ndx += $sublen; $len -= $sublen;
          if ($action == 0x83) {  # Get URL
            $buf =~ m/^([^\000]+)/;
            push @embs, $1;
          }
        }
      }
    }
    $ndx += $len;
  }
  (\@refs,\@embs);
}

use LWP::UserAgent;
#use LWP::Debug qw(+);

my $ua = new LWP::UserAgent;
$ua->proxy('http', $proxy);

sub fetch ( $ ) {
  my ($uri) = @_;
  warn "Retreiving $uri\n" if $debug;
  my $req = HTTP::Request->new(HEAD => $uri);
  $uri->scheme =~ /http|ftp|file/ or # Someday it would be nice to
DESCRIBE
    return HTTP::Response->new(200); # rtsp urls.  For now, act like
we got it.
```

```
  my $res = $ua->request($req);
  # Check the outcome of the response
  if (!$res->is_success) {
    warn "Unable to HEAD $uri: ".$res->status_line."\n" if $debug;
    # This is bit cheesy, but since some servers barf on HEAD
requests,
    # we do a GET on a hard failure.
    if ($res->code == 500) {
      $req = HTTP::Request->new(GET => $uri);
      $res = $ua->request($req);
      warn "Unable to GET $uri: ".$res->status_line."\n" unless
        $res->is_success;
    }
    return $res unless $res->is_success;
  }

  # If we can parse it then we should actually GET it, so we can
spider off
  # links.  Also, no need to retreive if we are going no deeper.
  return $res unless extractor($uri, $res->content_type) &&
$depth_left;

  unless ($req->method eq 'GET') { # Don't bother if we already had to
GET
    $req = HTTP::Request->new(GET => $uri);
    $res = $ua->request($req);
    warn "Unable to GET $uri: ".$res->status_line."\n" unless
$res->is_success;
  }
  # Insert Content-Length if it's not there
  $res->headers->header("Content-Length", length($res->content))
unless
    $res->headers->header("Content-Length");
  return $res;
}

# Make sure a URI points to its origin, not the cdn.
sub originify ( $ ) {
  my ($uri) = @_;
  # Look for URLs the content providers have already rewritten.  We
  # will revert them to their original form (to find their origin
  # location) and spider that instead.  (We don't want to spider the
cdn.)
  return $uri unless defined $uri->host && $uri->host eq $hd;

  # It's a link to the hd.  We have to reverse it or throw it away.
  my @path = $uri->path_segments;
  shift @path;                  # First segment is always nothing;
  if (@path and $path[0] =~ '^cdn-') {
    my @tail;
    shift @path;                # Remove cdn-* tag
    unshift @path, "";          # Put that first segment back.
    while (@path) {
      my $path = join('/', @path);
      if (exists $rmap{$path}) {
        my $new = URI->new(join('/', $rmap{$path}, @tail))->canonical;
        return $new;
      }
      unshift @tail, pop @path;
    }
  }
  warn "Unreversable cdn link: $uri\n";
  return 0;
}

# Convert an ARL back to it's original URL.  Works only on one type of
ARL.
# Maybe it should return 0 if it sees an ARL it doesn't understand?
sub deakamize ( $ ) {
  my ($arl) = @_;
  # 7 is hard coded because that is the typecode for this kind of ARL
  if ($arl =~
m@http://[^/]*akamai(?:tech)?.net/7/\d+/\d+/[\dabcdef]+/(.*)@) {
    return URI->new("http://$1");
  } else {
    return $arl;
  }
}

# Filter out schemes we don't understand and queries, then convert the
rest
# to a standard form - pointing into the origin instead of cdn.
sub canonicalize {
  my ($base, @urls) = @_;
  @urls = map { s/\#.*//; $_; } @urls; # Get rid of fragments
  @urls = map { URI->new_abs($_, $base)->canonical } @urls; # Standard
form
  @urls = grep {
    $_->scheme =~ /http|ftp|file|rtsp|mms/ &&  # Filter for "normal"
schemes
    ! $_->query;                # and non-queries
  } @urls;
  return map { deakamize(originify($_)); } @urls # To origin
}

# return true if $url is ACCEPTed by filters.
```

```
sub filter ( $$ ) {
  my ($filters, $url) = @_;
  for my $filter (@$filters) {
    return $filter->[1] if "$url" =~ $filter->[0];
  }
}

my %catalog = ();               # Maps URIs to response headers

# Main spidering loop
my $fetched = 0;
while (my $uri = shift @todo) {
  # Stop if we are at max --depth
  if (!ref($uri)) {             # A non-ref must be an integer, used
for DEPTH
    if ($uri) {                 # Non-zero means we keep going
      push @todo, $uri-1;
      $depth_left = $uri-1;
      next;
    }
    my $left = @todo;
    warn "Stopping with $left urls left because --depth=$depth\n";
    last;                       # Hit a zero, meaning stop
  }

  my $res = fetch($uri);
  next unless $res->is_success;
  $catalog{$uri} = $res->headers;
  $fetched++;

  if (my $extract = extractor($uri, $res->content_type)) {
    my ($refs, $embs) = &$extract($res->content);

    # Get urls into standard form
    @$refs = canonicalize($res->base, @$refs);
    @$embs = canonicalize($res->base, @$embs);

    # Get rid of urls we don't care about
    @$embs = grep { filter(\@filters, $_) } @$embs;
    @$refs = grep { filter(\@filters, $_) } @$refs;

    # Remove duplicate embs before saving.
    # Dup removal is unnecessary for correctness, but it avoids big
CONTAINS
    my %dup = ();
    @$embs = grep { !$dup{$_}++ } @$embs;
    $catalog{$uri}->header(CONTAINS=>"@$embs");
    # Add unseen urls to the todo list
    push @todo, grep { !$seen{$_}++ } (@$refs, @$embs);
  }

  # Stop if we have hit our --limit
  if ($fetched == $limit) {
    my $left = @todo;
    warn "Stopping with $left urls left because --limit=$limit\n";
    last;
  }

  if ($debug and $fetched % 100 == 0) {
    warn sprintf "%d fetched, %d todo\n", $fetched, scalar @todo;
  }
}

# Output loop. The only reason this loop isn't built into the input
loop is
# so the output can be sorted.  Maybe we don't really care about.  If
not,
# it should be moved into the input loop so that output will continue
as
# progress is made and memory for the catalog will not be required.

# datapoint: 20k URLs from hgtv cause 70Mb process and take > 11hrs

use IO::File;
my $DB = IO::Handle->new_from_fd(fileno(STDOUT),"w");
$DB = IO::File->new("> $db") or die "$db: $!\n" if $db;
my @headers = qw/Content-Type Content-Length Last-Modified CONTAINS/;
for my $key (sort keys %catalog) {
  print $DB "URL: $key\n";
  for my $h (@headers) {
    print $DB "$h: ".$catalog{$key}->header($h)."\n"
      if $catalog{$key}->header($h);
  }
  print $DB "\n";
}
close $DB or die $! if $db;
```

Manifest Script Source

```
#!/usr/bin/perl -w

use strict;

use Getopt::Long;
my @db = ();                    # URL databases to read in
my $xml = "";                   # XML filename to write manifest to
my @setters = ();               # General attribute setters
my $playservertable = "";       # File that contains the
PlayServerTable
my @map = ();                   # origin to cdn-url mappings
my $debug = 0;                  # Print extra debugging info?
my $recursive = 1;              # Should prepos containers prepos
their kids?

# Return an array containing each line from a file.
# Used by the --file option to allow stuffing @ARGV with args from a
file.
#  '#' until end of line is a comment character (ie it is not
returned)
#  whitespace is stripped from the beginning and end of lines
#  empty lines (or just comments and/or whitespace) are ignored
sub lines ( $ ) {
  my ($filename) = @_;
  open (F, "< $filename") or die "$filename: $!\n";
  my @lines = map { s/\#.*//g; s/\s*(\S*)\s*/$1/; $_ || (); } <F>;
  close F or die $!;
  return @lines;
}

# Convert a glob pattern to a regular expression.
# Assumes that the glob matches only if it matches the entire string.
sub glob2regex ( $ ) {
  # Note: This does not allow the writer of glob patterns to escape
them.
  # * and ? are always special, [,],and- are always passed through.
  my ($glob) = @_;
  $glob = quotemeta($glob);  # First, escape everything
  $glob =~ s/\\\*/.*/g;         # Convert * to .*
  $glob =~ s/\\\?/./g;          # Convert ? to .
  $glob =~ s/\\(\[|\-|\])/$1/g; # Reconstruct things like [a-z].
  return "^$glob\$";
}

sub process_type ( $$ ) {
  my ($opt, $val) = @_;
  push @setters, parse_setter("type=$opt:$val");
}

# We want all scripts to be runnable from a single config file, so
each
# take all of the arguments of the others.  Naturally, these arguments
are
# ignored if they are irrelevent.  If you want to use identical
command
# lines for all three scripts, be sure to use the --start, --db, and
--xml
# options.

my $junk;
GetOptions("prepos=s" => \&process_type,
           "live=s" => \&process_type,
           "set=s" => sub {my $val = $_[1]; push @setters,
parse_setter($val)},
           "recursive!" => \$recursive,
           "playservertable=s" => $playservertable,
           "xml=s" => \$xml,
           "map=s" => \@map,
           "db=s" => \@db,
           "<>" => sub { push @db, $_[0]; },
           # Arguments that all scripts take
           "file=s" => sub {my ($opt, $val) = @_; unshift @ARGV,
lines($val)},
           "debug!" => \$debug,
           # Arguments that are really only for 'spider' or 'rewrite'
           "limit=n" => \$junk,
           "depth=n" => \$junk,
           "prefix=s" => \$junk,
           "accept=s" => \$junk,
           "reject=s" => \$junk,
           "hd|rd=s" => \$junk,
           "start=s" => \$junk,
           "file-map=s" => \$junk,
           "index=s" => \$junk,
           "od|origin=s" => \$junk,
           "always-rewrite=s" => \$junk,
  ) or die "Bad argument syntax\n";

sub parse_setter ( $ ) {
  my ($setter) = @_;
  my ($settings, $sub) = split ':', $setter, 2;
  my @settings = split ' ', $settings;
  my %settings = ();
  for my $setting (@settings) {
    my ($key, $val) = split '=', $setting;
    $settings{$key} = $val;
  }
  return [parse_sub($sub), \%settings];
}

sub parse_sub ( $ ) {
  local ($_) = @_;
  warn "Changing `$_'\n" if $debug;

  # Allow comparisons to size, including shortcuts like 10k
  s/\bsize\b/\$H{'Content-Length'}/gio;
  s/\b(\d+)GB?\b/($1*1024M)/gio;
  s/\b(\d+)MB?\b/($1*1024K)/gio;
  s/\b(\d+)KB?\b/($1*1024)/gio;

  # Allow matching on URL
  s+\bmatch\(([^\)]*)\)+"m\@".glob2regex($1)."\@i"+gioe;

  # Allow matching on type
  s+\btype\(([^\)]*)\)+"(exists \$H{'Content-Type'} &&
\$H{'Content-Type'} =~ m@".glob2regex($1)."@)"+gioe;

  warn " into `$_'\n" if $debug;
  my $sub = eval "sub { my (\$i) = \@_; $_ }";
  $sub or die "Unable to understand `$_'\n";
}

my %map;
for my $map (@map) {
  my ($origin, $cdn) = split('=', $map);
  $map{$origin} = $cdn;
}

# Convert one URI to another according to map.  Always return a new
URI,
# even if contents are unchanged.
sub translate ( $$ ) {
  my ($uri, $map) = @_;
  # Try each prefix, longer ones first
  for my $prefix (sort { length($b) <=> length($a) } keys %$map) {
    if (index($uri, $prefix) == 0) {
      my $t = "$uri";
      substr($t, 0, length($prefix)) = $map->{$prefix};
      return URI->new($t)
    }
  }
  return $uri->clone;
}

my @backups = grep { /=/ } @ARGV; # args with = in them are backup
specifiers
@ARGV = grep { ! /=/ } @ARGV;

use URI;

my %servers = ();
my %items = ();                  # Catalog of all URLs
my @items = ();                  # Same as %items, but sorted.

my $depth = 0;
my %default = ();               # Current attributes in effect because
of group
my @chain = ();                  # Undo chain as groups are closed

sub push_params ( % ) {
  my %hash = @_;
  my $str = params(%hash);
  my $changes = {};
  while (my ($key,$val) = each %hash) {
    $changes->{$key} = $default{$key};
    $default{$key} = $val;
  }
  push @chain, $changes;
  return $str;
}

sub pop_params () {
  my $changes = pop @chain;
  while (my ($key,$val) = each %{$changes}) {
    $default{$key} = $val;
  }
}

my %xml_ent = ('&' => 'amp', '<' => 'lt', '>' => 'gt', '"' => 'quot');
sub xml_attr ( $ ) {
  my ($val) = @_;
  $val =~ s/([\&\<\>\"])/&$xml_ent{$1};/;
  return "\"$val\"";
}

sub params ( % ) {
  my %hash = @_;
  my $str = "";
  while (my ($key,$val) = each %hash) {
    $str .= " $key=" . xml_attr($val)
      unless defined $default{$key} && $default{$key} eq $val;
  }
  return $str;
}

sub open_server ( @ ) {
  return ("  " x $depth++) . "<server".params(@_).">\n";
}
sub close_server ( ) {
  die "More close_server()s than open_server()s!" unless  $depth;
  return ("  " x --$depth) . "</server>\n";
}
sub host ( % ) {
  return ("  " x $depth) . "<host".params(@_)."/>";
}

sub open_group ( @ ) {
  return ("  " x $depth++) . "<item-group".push_params(@_).">\n";
}
sub close_group ( ) {
  die "More close_group()s than open_group()s!" unless  $depth;
  pop_params();
  return ("  " x --$depth) . "</item-group>\n";
}

sub item ( $ ) {
  my ($item) = @_;
  my $str = ("  " x $depth) . "<item".params(%{$item->{attrs}});
  if (! @{$item->{contains}}) {
    return "$str/>";
  }
  $str .= ">\n";
  $depth++;
  for my $contained (@{$item->{contains}}) {
    if ($contained->{type} eq 'prepos') {
      my $path = translate($contained->{uri}, \%map)->path;
      $str .= ("  " x $depth) .
"<contains".params('cdn-url'=>$path)."/>\n";
    }
  }
  $depth--;
  $str .=  ("  " x $depth) . "</item>";
  return $str;
}

sub header () {
  return <<HEADER
```

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE CdnManifest SYSTEM "CdnManifest.dtd">

<CdnManifest>
HEADER
}

sub footer () {
  return "</CdnManifest>\n";
}

sub set ( $$ ) {
  my ($setter, $i) = @_;
  my ($pred, $attrs) = @$setter;
  use vars "%H";
  local ($_, %H) = ($i->{uri}, %{$i->{hdrs}});
  if (&$pred($i)) {
    for my $key (keys %$attrs) {
      $i->{attrs}->{$key} = $attrs->{$key};
    }
    return 1;
  }
  return 0;
}

sub print_items {
  my $server = "";
  for my $i (@_) {
    my $uri = $i->{uri};
    if ($uri->host ne $server) {
      print close_group() if $server;
      print open_group(server=>$uri->host);
      $server = $uri->host;
    }
    $i->{attrs}->{src} = $uri->path;
    my $t = translate($uri, \%map);
    $i->{attrs}->{'cdn-url'} = $t if $t ne $uri; # Only include if
needed
    print item($i) . "\n";
  }
  print close_group() if $server;
}

sub preposition_contents {
  my ($item) = @_;

  for my $contained (@{$item->{contains}}) {
    next if $contained->{type}; # Already set, or live
    warn "Forcing prepos of ".$contained->{uri}."\n" if $debug;
    $contained->{type} = $contained->{attrs}->{type} = 'prepos';
    preposition_contents($contained);
  }
}

# Input loop.  Read in all the URLs from the databases
@ARGV = @db;
while (<>) {
  my $item = {  uri => undef,   # Original url
                hdrs => {},     # Headers from database
                attrs => {},
                contains => [], # items that this one contains
                type => ""      # Convenience.  It's just
attrs->{type}
             };
  { do { # do {} while IS NOT A LOOP, the extra braces allow "last" to
work
    chomp;
    last unless $_;
    my ($header, $val) = split(": ", $_, 2);
    $item->{hdrs}->{$header} = $val;
  } while (<>); }
  die "Headers without a URL!\n" unless exists $item->{hdrs}->{URL};
  my $uri = $item->{uri} = URI->new($item->{hdrs}->{URL});
  push @items, $items{$uri} = $item;
  $servers{$uri->host} = 1;
}

# Figure out what contains what
my %missing = ();               # Tracks URI that have been reported
missing
for my $item (@items) {
  if (exists $item->{hdrs}->{CONTAINS}) {
    my @contains = ();
    my @missing = ();
    for my $c (split ' ', $item->{hdrs}->{CONTAINS}) {
      my $contained = $items{$c};
      if ($contained) {
        push @contains, $contained;
      } else {
        # Only consider $c missing if it has not yet been reported
        push @missing, $c unless $missing{$c}++;
      }
    }
    warn $item->{uri}." contains missing urls:\n ".join("\n ",
@missing)."\n"
      if @missing;
    $item->{contains} = \@contains;
  }
}

# Report URLs that were missing multiple times
my $intro = 0;
while (my ($uri, $times) = each(%missing)) {
  if ($times > 1) {
    warn "Some URLs are missing multiple times:\n" unless $intro++;
    warn sprintf " %3d %s\n", $times, $uri;
  }
}
warn "\n" if $intro;

# Run all the command line setters
for my $item (@items) {
  for my $setter (@setters) {
    set($setter, $item);
  }
  $item->{type} = $item->{attrs}->{type} if exists
$item->{attrs}->{type};
}

# Recursively preposition anything that is contained in a
prepositioned item
if ($recursive) {
  for my $item (@items) {
    next unless $item->{type} eq 'prepos';
    preposition_contents($item);
  }
}

# Spit out the manifest file
if ($xml) {
  open XML, "> $xml" or die "$xml: $!\n";
  select XML;
}

print header();
for my $s (keys %servers) {
  print open_server(name=>$s);
  print host(name=>$s, proto=>'http')."\n";
  print close_server();
}
print "\n<!-- Prepositioned Items -->\n";
print open_group(type=>"prepos");
print_items(grep {$_->{type} eq 'prepos'} @items);
print close_group();
print "\n<!-- Live Items -->\n";
print open_group(type=>"live");
print_items(grep {$_->{type} eq 'live'} @items);
print close_group();

if ($playservertable) {
  open TABLE, "< $playservertable" or die "$playservertable: $!\n";
  print <TABLE>;
  close TABLE or die $!;
}

print footer();
close XML or die $! if $xml;
select STDOUT;

# Collect and print some simple statistics
my %space = ();                # Amount of space used by various
types
my $space;
my %num = ();                  # Number of pieces of each type pf
content
my $num = 0;
for my $item (grep {$_->{type} eq 'prepos'} @items) {
  if (! exists $item->{hdrs}->{'Content-Type'}) {
    warn $item->{uri} . " has no content type.\n";
    next;
  }
  my $type = $item->{hdrs}->{'Content-Type'};
  $type =~ s/[\s,;].*$//;
  $num{$type} ||= 0;
  $num{$type}++;
  $num++;

  $space{$type} ||= 0;
  if (exists $item->{hdrs}->{'Content-Length'}) {
    $space{$type} += $item->{hdrs}->{'Content-Length'};
    $space += $item->{hdrs}->{'Content-Length'};
  }
}

my $k = 1024; my $m = 1024*$k; my $g = 1024*$m;
sub abbrev ( $ ) {
  my ($num) = @_;
  return 0 unless defined $num;
  return ($num/$g,"G") if $num > $g;
  return ($num/$m,"M") if $num > $m;
  return ($num/$k,"K") if $num > $k;
  return ($num,"b");
}

for my $type (sort { $space{$a} <=> $space{$b} } keys %num) {
  warn sprintf "%22s %5d %4d%s\n", $type, $num{$type},
abbrev($space{$type});
}
warn sprintf "%22s %5d %4d%s\n", "Total", $num, abbrev($space);
```

CDN Supported Time Zones

This appendix lists time zone designations and abbreviations that are supported by the Cisco Internet CDN Software. Time zones can be specified for any content item or content item group using the manifest file.

Specifying a time zone for a content item affects the expiration or activation of that item in a CDN where remote nodes are dispersed across different time zones.

This appendix contains the following sections:

Supported Time Zone Abbreviations

ACT—Australian Central Time

AET—Australian Eastern Time

ART—Argentina Time

AST—Arabia Standard Time

BET—Bering Standard Time

BST—Bering Summer Time

CAT—Central Africa Time

EAT—East Africa Time

ECT—Ecuador Time

EET—East European Time

EST—Eastern Standard Time (U.S.)

GMT—Greenwich Mean Time

HST—Hawaiian Standard Time

IST—Israeli Standard Time

JST—Japan Standard Time

MET—Middle European Time

MST—Mountain Standard Time

NST—Newfoundland Standard Time

PNT—Pitcairn Time

PST—Pacific Standard Time

SST—Samoa Standard Time

UTC—Coordinated Universal Time

WET—Western European Time

Supported Time Zone Designations by Continent

Africa

Africa/Abidjan

Africa/Accra

Africa/Addis_Ababa

Africa/Algiers

Africa/Asmera

Africa/Bangui

Africa/Banjul

Africa/Bissau

Africa/Blantyre

Africa/Bujumbura

Africa/Cairo

Africa/Casablanca

Africa/Conakry

Africa/Dakar

Africa/Dar_es_Salaam

Africa/Djibouti

Africa/Douala

Africa/Freetown

Africa/Gaborone

Africa/Harare

Africa/Johannesburg

Africa/Kampala

Africa/Khartoum

Africa/Kigali

Africa/Kinshasa
Africa/Lagos
Africa/Libreville
Africa/Lome
Africa/Luanda
Africa/Lubumbashi
Africa/Lusaka
Africa/Mbabane
Africa/Malabo
Africa/Maseru
Africa/Mogadishu
Africa/Monrovia
Africa/Nairobi
Africa/Ndjamena
Africa/Niamey
Africa/Nouakchott
Africa/Ouagadougou
Africa/Porto-Novo
Africa/Sao_Tome
Africa/Timbuktu
Africa/Tripoli
Africa/Tunis
Africa/Windhoek

Americas

America/Adak
America/Anchorage
America/Anguilla

America/Antigua

America/Aruba

America/Asuncion

America/Barbados

America/Belize

Atlantic/Bermuda

America/Bogota

America/Buenos_Aires

America/Caracas

America/Cayenne

America/Cayman

America/Chicago

America/Costa_Rica

America/Curacao

America/Dawson_Creek

America/Denver

America/Dominica

America/Edmonton

America/El_Salvador

America/Fortaleza

America/Godthab

America/Grand_Turk

America/Grenada

America/Guadeloupe

America/Guatemala

America/Guayaquil

America/Guyana

America/Halifax

America/Havana

America/Indianapolis

America/Jamaica

America/La_Paz

America/Lima

America/Los_Angeles

America/Managua

America/Manaus

America/Martinique

America/Mazatlan

America/Mexico_City

America/Montevideo

America/Montreal

America/Montserrat

America/Nassau

America/New_York

America/Noronha

Antarctica/Palmer

America/Panama

America/Paramaribo

America/Phoenix

America/Porto_Acre

America/Port-au-Prince

America/Port_of_Spain

America/Puerto_Rico

America/Regina

America/Santiago

America/Santo_Domingo

America/Sao_Paulo

America/Scoresbysund

America/St_Johns

America/St_Kitts

America/St_Lucia

America/St_Thomas

America/St_Vincent

Atlantic/Stanley

America/Tegucigalpa

America/Thule

America/Tijuana

America/Tortola

America/Vancouver

America/Winnipeg

Antarctica

Antarctica/Casey

Antarctica/DumontDUrville

Antarctica/Mawson

Antarctica/McMurdo

Asia and India

Asia/Aden

Asia/Almaty

Asia/Amman

Asia/Anadyr

Asia/Aqtau

Asia/Aqtobe

Asia/Ashkhabad

Asia/Baghdad

Asia/Bahrain

Asia/Baku

Asia/Bangkok

Asia/Beirut

Asia/Bishkek

Asia/Brunei

Asia/Calcutta

Asia/Colombo

Asia/Dacca

Asia/Damascus

Asia/Dubai

Asia/Dushanbe

Asia/Hong_Kong

Asia/Irkutsk

Asia/Jakarta

Asia/Jayapura

Asia/Jerusalem

Asia/Kabul

Asia/Kamchatka

Asia/Karachi

Asia/Katmandu

Asia/Krasnoyarsk

Asia/Kuala_Lumpur

Asia/Kuwait

Asia/Macao

Asia/Magadan

Asia/Manila

Asia/Muscat

Asia/Nicosia

Asia/Novosibirsk

Asia/Phnom_Penh

Asia/Pyongyang

Asia/Qatar

Asia/Rangoon

Asia/Riyadh

Asia/Saigon

Asia/Seoul

Asia/Shanghai

Asia/Singapore

Asia/Taipei

Asia/Tashkent

Asia/Tbilisi

Asia/Tehran

Asia/Thimbu

Asia/Tokyo

Asia/Ujung_Pandang

Asia/Ulan_Bator

Asia/Vientiane

Asia/Vladivostok

Asia/Yakutsk

Asia/Yekaterinburg

Asia/Yerevan

Indian/Antananarivo

Indian/Chagos

Indian/Christmas

Indian/Cocos

Indian/Comoro

Indian/Kerguelen

Indian/Mahe

Indian/Maldives

Indian/Mauritius

Indian/Mayotte

Indian/Reunion

Atlantic Nations

Atlantic/Azores

Atlantic/Canary

Atlantic/Cape_Verde

Atlantic/Faeroe

Atlantic/Jan_Mayen

Atlantic/Reykjavik

Atlantic/South_Georgia

Atlantic/St_Helena

Australia

Australia/Adelaide

Australia/Brisbane

Australia/Broken_Hill

Australia/Darwin

Australia/Hobart

Australia/Lord_Howe

Australia/Perth

Australia/Sydney

Europe

Europe/Andorra

Europe/Amsterdam

Europe/Athens

Europe/Belgrade

Europe/Berlin

Europe/Brussels

Europe/Bucharest

Europe/Budapest

Europe/Chisinau

Europe/Copenhagen

Europe/Dublin

Europe/Gibraltar

Europe/Helsinki

Europe/Istanbul

Europe/Kaliningrad

Europe/Kiev

Europe/Lisbon

Europe/London

Europe/Luxembourg

Europe/Madrid

Europe/Malta

Europe/Minsk

Europe/Monaco

Europe/Moscow

Europe/Oslo

Europe/Paris

Europe/Prague

Europe/Riga

Europe/Rome

Europe/Samara

Europe/Simferopol

Europe/Sofia

Europe/Stockholm

Europe/Tallinn

Europe/Tirane

Europe/Vaduz

Europe/Vienna

Europe/Vilnius

Europe/Warsaw

Europe/Zurich

Pacific Nations

Pacific/Apia

Pacific/Auckland

Pacific/Chatham

Pacific/Easter

Pacific/Efate

Pacific/Enderbury

Pacific/Fakaofo

Pacific/Fiji

Pacific/Funafuti

Pacific/Galapagos

Pacific/Gambier

Pacific/Guadalcanal

Pacific/Guam

Pacific/Kiritimati

Pacific/Kosrae

Pacific/Majuro

Pacific/Marquesas

Pacific/Nauru

Pacific/Niue

Pacific/Norfolk

Pacific/Noumea

Pacific/Pago_Pago

Pacific/Palau

Pacific/Pitcairn

Pacific/Ponape

Pacific/Port_Moresby

Pacific/Rarotonga

Pacific/Saipan

Pacific/Tahiti

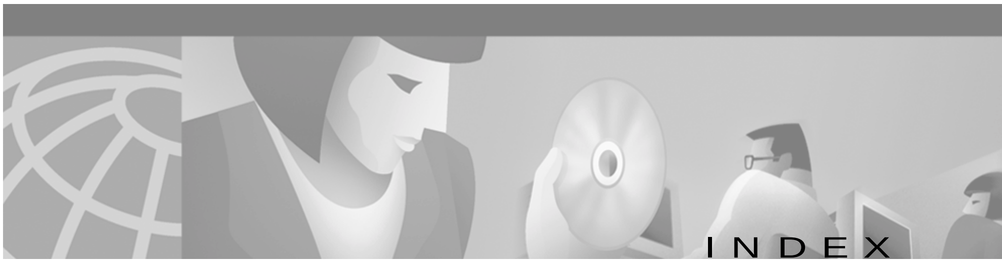
Pacific/Tarawa

Pacific/Tongatapu

Pacific/Truk

Pacific/Wake

Pacific/Wallis


INDEX