



# Configuring Conventional Caching Services for Standalone Content Engines

---

This chapter describes how to configure conventional caching services (HTTP, FTP [FTP-over-HTTP caching and native FTP caching], HTTPS, or DNS caching) for standalone Content Engines. It also describes how to configure the TFTP gateway, persistent connections, healing mode, and the Internet Cache Protocol (ICP) for standalone Content Engines. This chapter includes the following sections:

- [Overview of Configuring Conventional Caching Services, page 7-2](#)
- [Configuring HTTP Caching for Standalone Content Engines, page 7-7](#)
- [Configuring HTTPS Caching for Standalone Content Engines, page 7-24](#)
- [Configuring FTP Caching for Standalone Content Engines, page 7-36](#)
- [Configuring the TFTP Server and Gateway for Standalone Content Engines, page 7-58](#)
- [Configuring DNS Caching for Standalone Content Engines, page 7-62](#)
- [Configuring Standalone Content Engines to Send out TCP Keepalives, page 7-67](#)
- [Configuring Persistent Connections on Standalone Content Engine, page 7-68](#)
- [Configuring Healing Mode for Content Engine Clusters, page 7-70](#)
- [Configuring the Internet Cache Protocol for Content Engine Clusters, page 7-72](#)



**Note**

---

For complete syntax and usage information for the CLI commands used in this chapter, see the *Cisco ACNS Software Command Reference, Release 5.5* publication. For information about configuring caching for Content Engines that are registered with a Content Distribution Manager, see the *Cisco ACNS Software Configuration Guide for Centrally Managed Deployments, Release 5.5*.

---

# Overview of Configuring Conventional Caching Services

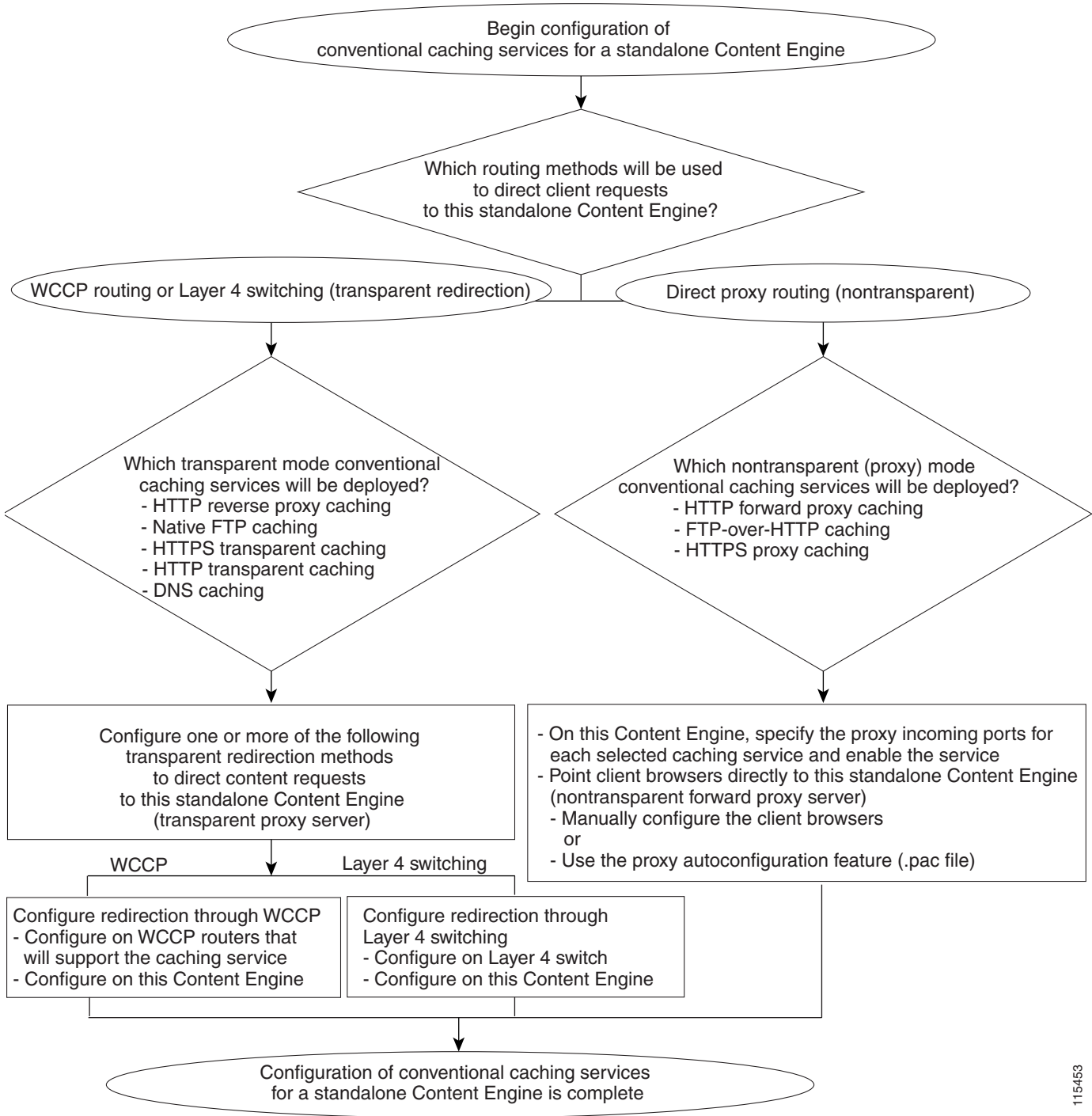
This section provides an overview of how to use the Content Engine CLI to configure conventional caching services (HTTP, HTTPS, FTP, and DNS caching) on standalone Content Engines. [Figure 7-1](#) provides a detailed view on how to configure conventional caching services for standalone Content Engines.

For information about how to use the Setup utility to configure the following three commonly used conventional caching services (HTTP reverse proxy caching, HTTP transparent caching using WCCP Version 2, and HTTP forward proxy caching), see the [“Configuring a Basic Configuration on Standalone Content Engines with the Setup Utility”](#) section on page 4-10.

For information about how to configure media caching services (WMT and RTSP caching and streaming services) in a locally managed deployment, see the following chapters:

- [Chapter 8, “Configuring RealMedia Services on Standalone Content Engines”](#)
- [Chapter 9, “Configuring WMT Streaming Media Services on Standalone Content Engines”](#)

Figure 7-1 Detailed View of Configuring Conventional Caching Services for Standalone Content Engines



115463

Table 7-1 is a checklist of tasks for configuring conventional caching services (HTTP, HTTPS, FTP, and DNS caching) for standalone Content Engines. This checklist also includes the steps involved in configuring these services on a standalone Content Engine.

**Table 7-1 Checklist for Configuring Conventional Caching Services for Standalone Content Engines**

Task	Additional Information and Instructions
<p><b>Start basic configuration of conventional caching services</b></p> <p>1. Configure one or more of the following routing methods to direct client requests to the standalone Content Engine:</p> <ul style="list-style-type: none"> <li>– Direct proxy routing (nontransparent)</li> <li>– Transparent redirection (WCCP routing or Layer 4 switching)</li> </ul>	<p>For direct proxy routing, see the “<a href="#">Configuring Client Browsers and Media Players for Direct Proxy Routing</a>” section on page 4-35.</p> <p>For WCCP routing, see the “<a href="#">Configuring Standalone Content Engines for WCCP Transparent Redirection</a>” section on page 6-9.</p> <p>For Layer 4 switching, see the “<a href="#">Configuring Layer 4 Switching as a Redirection Method</a>” section on page 6-50.</p>
<p>2. If direct proxy routing is to be used, is a *.pac file to be used?</p>	<ul style="list-style-type: none"> <li>• If no, then manually configure each client browser to point directly to the standalone Content Engine as a direct proxy server. See the “<a href="#">Manually Pointing Client Browsers to a Standalone Content Engine</a>” section on page 4-42.</li> <li>• If yes, then configure the standalone Content Engine and the client browsers to use a proxy autoconfiguration (PAC) file. See the “<a href="#">Using PAC Files to Point Client Browsers Directly to a Standalone Content Engine</a>” section on page 4-37.</li> </ul>
<p>3. Configure nontransparent (proxy) mode conventional caching services on this standalone Content Engine:</p> <ul style="list-style-type: none"> <li>– HTTP forward proxy caching</li> <li>– FTP-over-HTTP caching</li> <li>– HTTPS proxy caching</li> </ul>	<p>See the following sections in this chapter:</p> <ul style="list-style-type: none"> <li>• <a href="#">Configuring Nontransparent HTTP Forward Proxy Caching on Standalone Content Engines</a>, page 7-8</li> <li>• <a href="#">Configuring Nontransparent FTP-over-HTTP Caching on Standalone Content Engines</a>, page 7-38</li> <li>• <a href="#">Configuring HTTPS Proxy Caching for Standalone Content Engines</a>, page 7-25</li> </ul>
<p>4. Configure transparent mode conventional caching services for this standalone Content Engine:</p> <ul style="list-style-type: none"> <li>– HTTP reverse proxy caching</li> <li>– Native FTP caching</li> <li>– HTTPS transparent caching</li> <li>– Transparent HTTP forward proxy caching</li> <li>– DNS caching</li> </ul>	<p>See the following sections in this chapter:</p> <ul style="list-style-type: none"> <li>• <a href="#">Configuring HTTP Reverse Proxy Caching for Standalone Content Engines</a>, page 7-23</li> <li>• <a href="#">Configuring FTP Native Caching for Standalone Content Engines</a>, page 7-41</li> <li>• <a href="#">Configuring HTTPS Transparent Caching for Standalone Content Engines</a>, page 7-27</li> <li>• <a href="#">Configuring Transparent HTTP Forward Proxy Caching for Standalone Content Engines</a>, page 7-17</li> <li>• <a href="#">Configuring DNS Caching for Standalone Content Engines</a>, page 7-62</li> </ul>

**Table 7-1 Checklist for Configuring Conventional Caching Services for Standalone Content Engines (continued)**

Task	Additional Information and Instructions
<p>5. You can now do any of the following tasks:</p> <ul style="list-style-type: none"> <li>– Configure the TFTP server and gateway.</li> <li>– Configure the Content Engine to use persistent connections.</li> <li>– Configure healing mode or ICP for cache clusters.</li> <li>– Configure streaming services for the Content Engine.</li> <li>– Configure content services for the Content Engine.</li> <li>– Perform advanced configuration on the Content Engine.</li> <li>– Monitor and troubleshoot.</li> </ul>	See tasks 6 through 24 below in this table.
6. Configure the TFTP server and gateway on the Content Engine.	<a href="#">Configuring the TFTP Server and Gateway for Standalone Content Engines, page 7-58</a>
7. Configure persistent connections for the Content Engine.	<a href="#">Configuring Standalone Content Engines to Send out TCP Keepalives, page 7-67</a>
8. Configure healing mode for cache clusters.	<a href="#">Configuring Healing Mode for Content Engine Clusters, page 7-70</a>
9. Configure Internet Cache Protocol (ICP) for cache clusters.	<a href="#">Configuring the Internet Cache Protocol for Content Engine Clusters, page 7-72</a>
10. Configure WMT caching and streaming services on the Content Engine.	<a href="#">Chapter 9, “Configuring WMT Streaming Media Services on Standalone Content Engines”</a>
11. Configure RTSP caching and streaming services on the Content Engine.	<a href="#">Chapter 8, “Configuring RealMedia Services on Standalone Content Engines”</a>
<b>Configure content services (optional)</b>	After configuring caching and streaming services on the standalone Content Engine, you can configure such content services as access control, URL filtering, ICAP, and rules.
<b>Configure content services for conventional caching</b>	
12. Decide if end user access to the Internet is to be controlled (access control for HTTP, HTTPS, and FTP-over-HTTP requests).	<ul style="list-style-type: none"> <li>• If no, then go to task 13.</li> <li>• If yes, then configure content authentication and authorization, as described in <a href="#">Chapter 10, “Configuring Content Authentication and Authorization on Standalone Content Engines.”</a></li> </ul>
13. Decide if URL filtering is to be used.	<ul style="list-style-type: none"> <li>• If no, then go to task 14.</li> <li>• If yes, then configure URL filtering for HTTP, HTTPS, and FTP requests, as described in <a href="#">Chapter 11, “Configuring Content Preloading and URL Filtering on Standalone Content Engines.”</a></li> </ul>

**Table 7-1 Checklist for Configuring Conventional Caching Services for Standalone Content Engines (continued)**

Task	Additional Information and Instructions
14. Determine whether there is an external Internet Content Adaptation Protocol (ICAP) server.	<ul style="list-style-type: none"> <li>If no, then go to task 15.</li> <li>If yes, then configure ICAP for HTTP and FTP-over-HTTP requests, as described in <a href="#">Chapter 12, “Configuring ICAP on Standalone Content Engines.”</a></li> </ul>
15. Determine if there are any special requirements for processing content requests.	<ul style="list-style-type: none"> <li>If no, then go to task 16.</li> <li>If yes, configure rules for HTTP, HTTPS, FTP-over-HTTP, WMT, and RTSP requests, as described in <a href="#">Chapter 13, “Configuring the Rules Template on Standalone Content Engines.”</a></li> </ul>
<b>Perform advanced configuration tasks (optional)</b>	
16. Configure advanced transparent caching features (for example, traffic bypass, overload bypass, flow protection, and IP spoofing).	<a href="#">Chapter 15, “Configuring Advanced Transparent Caching Features on Standalone Content Engines”</a>
17. Set up additional network interfaces on the standalone Content Engine.	<a href="#">Chapter 16, “Configuring Additional Network Interfaces and Bandwidth on Standalone Content Engines”</a>
18. Configure bandwidth for interfaces and content services on this standalone Content Engine.	<a href="#">Chapter 16, “Configuring Additional Network Interfaces and Bandwidth on Standalone Content Engines”</a>
19. Set up login authentication and authorization for administrative users who will be accessing the Content Engine for configuration, monitoring, or troubleshooting purposes.	<a href="#">Chapter 17, “Configuring Administrative Login Authentication and Authorization on Standalone Content Engines”</a>
20. Configure this standalone Content Engine for system accounting with TACACS+.	<a href="#">Chapter 18, “Configuring AAA Accounting on Standalone Content Engines”</a>
21. Configure IP access control lists (ACLs) on this standalone Content Engine.	<a href="#">Chapter 19, “Creating and Managing IP Access Control Lists for Standalone Content Engines”</a>
22. View or modify TCP stack parameters for this standalone Content Engine.	<a href="#">Chapter 20, “Viewing and Modifying TCP Stack Parameters on Standalone Content Engines”</a>
23. View or modify the system logging settings for this standalone Content Engine.	<a href="#">Monitoring the Performance of Specific URLs, page 21-52</a>
<b>Monitor and troubleshoot</b>	
24. Monitor this standalone Content Engine with SNMP or the ACNS software alarms.	<a href="#">Chapter 21, “Monitoring Standalone Content Engines and Transactions”</a>
25. Troubleshoot problems by using tools such as traceroute or ping.	<a href="#">Chapter 21, “Monitoring Standalone Content Engines and Transactions”</a>

# Configuring HTTP Caching for Standalone Content Engines

HTTP is the main protocol used on the web for communication between web browsers and web servers. There are two commonly implemented HTTP versions today: HTTP 1.0 and HTTP 1.1. The ACNS 5.x software supports both HTTP 1.0 and HTTP 1.1.

HTTP runs over TCP port 80 (which is reserved for HTTP) and is a request-response protocol. The client (web browser) sends a request to a web server, and the web server responds with the content. Each request or response can carry a number of headers with it, specifying various properties of the client, the server, the object or communicating states between the client and the web server.

The HTTP 1.1 specification allows objects to be transmitted using Chunked Transfer Coding. In the ACNS 5.1 software and earlier releases, the proxying of chunked transfer encoded (CTE) objects was supported; however, the caching of these objects was not supported.

In the ACNS 5.2 software and later releases, caching of CTE HTTP objects is supported. A subsequent request for the same CTE object, which meets the standard HTTP caching freshness criteria, results in that object being fetched from the Content Engine's cache and sent to the client that is HTTP 1.1 compliant. HTTP 1.0 clients do not support CTE. Consequently, if an object is stored in a CTE format, then the Content Engine must refetch the object from the source HTTP server if the HTTP client is not HTTP 1.1 compliant.

To enable the caching of CTE HTTP objects on a standalone Content Engine, specify the **http cache-chunk-encoded enable** global configuration command. After enabling this feature, you can use the **show statistics http request EXEC** command to verify that this feature is working properly. Check the command output to verify whether the displayed value in the Chunked HTTP Responses: field increments after the Content Engine serves cached CTE HTTP objects to HTTP 1.1 compliant clients.



Note

For details on HTTP, see the IETF RFC 1945 (HTTP 1.0 specification) and RFC 2616 (HTTP 1.1 specification).

Table 7-2 lists the HTTP caching services that are supported by Content Engines that are running the ACNS 5.2.1 software and later releases. The type of services supported varies based on the method used to route the HTTP request to the Content Engine.

**Table 7-2 Supported HTTP Caching Services with Standalone Content Engines**

HTTP Caching Services	More Information
<b>Direct Proxy Mode</b>	
Nontransparent HTTP forward proxy caching	<a href="#">Configuring Nontransparent HTTP Forward Proxy Caching on Standalone Content Engines, page 7-8</a>
<b>Transparent Redirection Mode</b>	
Transparent HTTP forward proxy caching	<a href="#">Configuring Transparent HTTP Forward Proxy Caching for Standalone Content Engines, page 7-17</a>
Transparent HTTP reverse proxy caching	<a href="#">Configuring HTTP Reverse Proxy Caching for Standalone Content Engines, page 7-23</a>

In the ACNS 5.1 software and earlier releases, a maximum of eight active WCCP services were supported by a WCCP Version 2 router and a Content Engine. In the ACNS 5.2.1 software and later releases, up to 25 active WCCP Version 2 services are supported. In ACNS 5.2.1 software release, only 17 WCCP services were defined. In the ACNS 5.3.1 software release, 18 WCCP services are defined.

See [Table B-3](#) for a list of supported WCCP services.

## Configuring Nontransparent HTTP Forward Proxy Caching on Standalone Content Engines

You can use the Content Engine GUI or CLI to configure HTTP proxy caching on a standalone Content Engine (nontransparent forward proxy server).

From the Content Engine GUI, choose **Caching > HTTP Proxy**, and use the displayed HTTP Proxy window to configure the HTTP connection settings. For more information on how to use this window, click the **HELP** button in the window.

From the Content Engine CLI, follow these steps:

- 
- Step 1** Configure the client browsers to send their HTTP requests directly to the Content Engine (nontransparent forward proxy server).

Point the client browsers directly to the Content Engine so that HTTP requests from these browsers are sent directly to the Content Engine (direct proxy routing). You can use the proxy autoconfiguration feature (PAC file) or manually configure the client browsers by specifying the IP address and port number of the forward proxy server. For more information on this topic, see the “[Configuring Client Browsers and Media Players for Direct Proxy Routing](#)” section on page 4-35.

- Step 2** Configure the Content Engine to accept incoming HTTP requests on ports other than port 80.

```
ContentEngine(config)# http proxy incoming ports
```

*ports* is the port number used by the standalone Content Engine to receive HTTP requests directly from the client browsers. This number ranges from 1 to 65535. You can specify up to eight ports.

This example configures an incoming HTTP proxy on port 8080. Up to eight incoming proxy ports can be configured on the same command line.

```
ContentEngine(config)# http proxy incoming 8080
```

- Step 3** (Optional) Configure HTTP cache freshness for the Content Engine, as described in the “[Configuring HTTP Cache Freshness Settings](#)” section on page 7-9.
- Step 4** (Optional) Configure authenticated HTTP cache settings for the Content Engine, as described in the “[Configuring Authenticated HTTP Cache Settings](#)” section on page 7-12.
-

## Configuring HTTP Cache Freshness Settings

With HTTP 1.1, you can configure the Content Engine to check the freshness of its cached objects before serving the requested content to a client browser. A *fresh object* is a web object that is not stale. A cached object is considered fresh under any of the following conditions:

- The Content Engine has freshly retrieved the object from the origin server.
- The Content Engine contacts the origin server to check about the freshness of the cached object, and the origin server confirms that the cached object has not been modified since the Content Engine cached it.
- The age of the cached object has not exceeded its freshness lifetime. The age of a cached object is the time that the object has been stored in the Content Engine's cache without the Content Engine explicitly contacting the origin server to check if the object is still fresh.

You can use the If-Modified-Since (IMS) feature to configure a standalone Content Engine to revalidate the freshness of the content stored in its local cache before serving the content to a client browser. The Content Engine checks the freshness of its cached content under the following conditions:

- When the Content Engine receives an IMS message from the client browser. This occurs if the setting for the local cache on the client browser is configured to check for newer versions of the cached pages each time the page are accessed.
- When the Content Engine receives a request for expired content.



### Note

If clients click their **Reload** browser button to reload the requested content into their browser, this causes all Content Engines that are located between the client and the origin servers that contain the requested content to refresh their cached objects.

The Content Engine validates the freshness of requested content in its cache by sending an IMS request to the origin web server. The Content Engine also sends an IMS request to the origin web server when the maximum Time To Live (TTL) has expired.

Content freshness is based upon a conditional GET feature of the HTTP protocol. The Content Engine will retrieve the requested information from the origin server again if the content has changed since it was cached on the Content Engine. In the HTTP protocol, the conditional GET request uses the value of the Last-Modified response header that was received with the document when it was retrieved and stored in the Content Engine cache. This value (the last modification date and time of the cached document) is sent in the If-Modified-Since request header. The conditional GET request uses the time stamp from the Last-Modified: header and sends it along with the request in the If-Modified\_Since header.

The following example shows an IMS request that a Content Engine would send to an origin web server.

```
GET /index.html HTTP/1.1
Server: www.cisco.com
Connection: keep-alive
If-Modified-Since: Tue 12 Sep 2000 10:07:04 GMT    Accept: */*
```

If the content has not changed, the origin web server responds with a 304 Not Modified message, and does not send the content.

If the content has changed, the new version is transferred to the Content Engine again. Typically, the origin web server also sends the Content Engine a 200 OK response along with the new version of the content.

The following examples show these two possible responses from the origin web server to the Content Engine IMS request:

```
304 Not Modified
(end-of-request)
```

or

```
200 OK
(response headers)
(data)
(end-of-request)
```

The Expires response, which indicates the time that the response expires, also affects caching. This response header indicates the time that the response becomes stale and should not be sent to the client without an up-to-date check (using a conditional GET operation). If the HTTP header of a cached object does not specify an expiration time, the Content Engine can age out cached objects through the **http age-multiplier** and **http max-ttl** global configuration commands.

The Content Engine can also calculate an expiration time for each web object before it is written to disk. The Content Engine's algorithm to calculate an object's cache expiration date is as follows:

Expiration date = (Today's date – Object's last modified date) \* Freshness factor

The last modified date is provided by the end server's file system. The freshness factor is tunable and derived from the text and binary percentage parameters of the **http age-multiplier** global configuration command. Valid age-multiplier values are from 0 to 100 percent of the object's age. Default values are 30 percent for text (HTML) and 60 percent for binary objects (for example, gifs). After the expiration date has passed, the object is considered stale and subsequent requests causes a fresh retrieval of the content by the Content Engine.

When configuring the HTTP cache freshness settings on standalone Content Engines, keep the following important points in mind:

- You can specify the maximum size of an HTTP object that can be stored in the cache. The maximum size limit for an HTTP object is 2096128 kilobytes (2 GB). An object with a size above the configurable upper limit is not stored by the Content Engine.
- Use the minimum and maximum Time To Live (TTL) settings to limit the duration of HTTP objects in the cache. By default, HTTP cacheable objects are kept for 5 minutes minimum and 3 to 7 days maximum (3 days for text-type objects, 7 days for binary). If an object has an explicit expiration date, this takes precedence over the configurable TTL. The default values are 3 days for text files and 7 days for binary objects.
- For HTTP objects, use the **http min-ttl** and **ftp min-ttl** global configuration commands to set the minimum TTL.
- Use the **http cache-cookies** global configuration command to enable the Content Engine to cache binary objects that are served with HTTP set-cookies headers and no explicit expiration information, but which might be cacheable.

You can use the Content Engine CLI or GUI to configure the freshness settings for an HTTP cache on a standalone Content Engine.

From the Content Engine GUI, choose **Cache > HTTP Freshness**, and use the displayed HTTP Freshness window. To obtain more information about how to use the HTTP Freshness window to configure freshness settings, click the **HELP** button in the window.

From the Content Engine CLI, follow these steps:

**Step 1** Specify the freshness factor for HTTP cached objects:

```
ContentEngine(config)# http age-multiplier text 50% bin 70%
```

**Step 2** Set the minimum amount of time that the HTTP cached object is stored in the Content Engine cache. In the following example, this minimum time is set to 10 minutes:

```
ContentEngine(config)# http min-ttl 10
```

**Step 3** Set the upper limit on the estimated expiration dates for HTTP cached objects, as indicated in the following examples:

```
ContentEngine(config)# http max-ttl days text 2 binary 4
ContentEngine(config)# http max-ttl hours text 1 hours binary 4
```

The TTL sets a ceiling on estimated expiration dates. An explicit expiration date in the HTTP header takes precedence over the configured TTL. [Table 7-3](#) lists the valid range of values.

**Table 7-3 Time To Live Range of Values for HTTP Freshness**

Scale	Range
Days	1–1825
Hours	1–43800
Minutes	1–2628000
Seconds	1–157680000

**Step 4** Specify the method that the Content Engine is to use to handle requests to revalidate the content freshness of the HTTP objects in its cache. In the following example, the Content Engine is configured to revalidate all HTTP objects for every HTTP request:

```
ContentEngine(config)# http reval-each-request all
```

**Step 5** Set the upper limit of the HTTP object size in kilobytes (KB). In the following example, the maximum size for an HTTP object is set to 500 kilobytes:

```
ContentEngine(config)# http object max-size 500
```



**Note** The Content Engine does not store an object if the size of the object exceeds the specified limit. The maximum object size for cached HTTP objects is 2096128 KB (2 GB).

**Step 6** Configure the Content Engine to cache binary objects and associated cookies, which are served with HTTP set-cookies headers and no explicit expiration information, but which might be cacheable:

```
ContentEngine(config)# http cache-cookies
```

**Step 7** Display the TTL configuration changes made to HTTP objects in the Content Engine's cache:

```
ContentEngine# show http ttl
Maximum time to live in days: text 3 binary 7
Minimum time to live for all objects in minutes: 5
```

## Configuring Authenticated HTTP Cache Settings

The authenticated HTTP caching feature allows content that was authenticated through basic authentication and NTLM authentication to be cached and served to more than one user, while maintaining security. If an authenticated object is cached, then subsequent requests for that object (from new users) require authentication. The cached object is revalidated with the origin server through the authorization header for the new user. If the user is not authorized, the server sends a 401 (Unauthorized) response. If the user is authorized and the object is not modified, the cached object is served to the client.



### Note

If the authentication cache is not large enough to accommodate all authenticated users at the same time, the Content Engine purges older entries that have not yet timed out.

You can use the Content Engine GUI or CLI to configure the parameters for the HTTP authentication cache on standalone Content Engines:

- From the Content Engine GUI, choose **Caching > Auth.Cache**, and use the displayed Authenticated Cache window. To obtain more information about this window, click the **HELP** button in the window.
- From the Content Engine CLI, use the **http authentication** global configuration command, as described in [Table 7-4](#).

```
http authentication { cache { max-entries entries | max-group-entries number | timeout minutes | tll minutes } | header { 401 | 407 } | realm line }
```

**Table 7-4** Parameters for the http authentication CLI Command

Parameter	Description
<b>cache</b>	Configures parameters that are related to the authentication cache on the Content Engine.
<b>max-entries</b>	Sets the maximum number of entries retained in the authentication cache.
<i>entries</i>	Maximum number of entries retained in the authentication cache on the Content Engine. Valid values are from 500 to 32000 entries. This is limited by the physical resources available on the Content Engine.
<b>max-group-entries</b>	Sets the maximum number of entries retained in the authentication group cache. This is subject to physical resources on the Content Engine. This option is only available in the ACNS 5.2 software and later releases.
<b>Tip</b>	If you intend to use the group name pattern, make sure that you set the correct number of maximum group entries in the authentication group cache. This number should correspond to the maximum number of groups that could be returned during authorization queries (for example, the total number of groups defined on the AAA server.)

**Table 7-4 Parameters for the http authentication CLI Command (continued)**

Parameter	Description
<i>number</i>	Maximum number of entries retained in the authentication group cache on the Content Engine. Valid values are from 500 to 12000 entries. This is limited by the physical resources available on the Content Engine.
<b>timeout</b>	Sets the timeout value of records in the authentication cache. Specifies how long an inactive entry can remain in the authentication cache before it is purged. Once a record has been purged, any subsequent access attempt to restricted Internet content requires a server lookup for reauthentication. This is the least-recently-used (LRU) value. It is also referred to as the <i>idle time</i> .
<i>minutes</i>	Time in minutes (1–1440) between the user’s last Internet access and the removal of that user’s entry from the authorization cache, forcing reauthentication. The default is 240 minutes (4 hours); the minimum is 30 minutes; and the maximum is 1440 minutes (24 hours).
<b>ttl</b>	Sets an absolute Time To Live (TTL) for entries in the authentication cache. This option is only available in the ACNS 5.2.1 software and later releases. By default, this option is disabled, which means that there is no TTL timeout in effect. This means that there will be no check to time out an authentication cache entry based on its creation time relative to a TTL value.
<i>minutes</i>	Time in minutes (1–1440) that specifies the maximum amount of time that an entry is valid in the authentication cache entry after its creation. The minimum is 1 minute; the maximum is 1440 minutes (24 hours). For more information, see the <a href="#">“Specifying a Reauthentication Interval” section on page 7-14</a> .
<b>header</b>	Determines which HTTP header to use for authentication (user ID and password) when the style of the HTTP request indicates that no proxy server is present. Headers can be either HTTP 401 (Unauthorized) or HTTP 407 (Proxy Authentication Required). The default is HTTP 401.
<b>401</b>	Uses HTTP 401 to query users for credentials.
<b>407</b>	Uses HTTP 407 to query users for credentials.
<b>realm</b>	Configures the realm string for basic HTTP request authentication.
<i>line</i>	Name of the realm string to be authenticated. The default is Cisco Content Engine.

The maximum number of entries that is maintained in the authentication cache is 32,000. The minimum number is 500. The default value is 16,000 entries. Use the **http authentication max-entries** global configuration command to configure the maximum number of entries that is to be maintained in the authentication cache on this Content Engine, if necessary.

If the authentication cache is not large enough to accommodate all authenticated users at the same time, the Content Engine purges older entries that have not yet timed out. The default time interval between the user’s last Internet access and the removal of that user’s entry from the authorization cache is 240 minutes (4 hours). The minimum time interval is 1 minute, and the maximum is 1440 minutes (24 hours). The Content Engine forces reauthentication with the access control server once this time interval expires.

In this example, the length of time that entries are valid in the authentication cache is set at 1000 minutes:

```
ContentEngine(config)# http authentication cache timeout 1000
```

When LDAP, RADIUS, and TACACS+ are used in proxy redirection mode, the authentication record kept in the authentication cache is indexed by the username and the password entered. When LDAP, RADIUS, or TACACS+ is used in WCCP-enabled router redirection mode, the authentication record indexed is the IP address of the Content Engine that is sending the request in transparent mode. When an NTLM server is used in either proxy redirection mode or WCCP-enabled router redirection mode, all authentication records are indexed by using the IP address of the requesting client. By default, the Content Engine authenticates cache loads based on the URL syntax of the incoming request.

Use the **http authentication header** global configuration command to configure the Content Engine to send a message to the client when authorization has failed. You can choose **http authentication header 401** (Unauthorized) or **http authentication header 407** (Proxy Authorization Required).

In the following example, the Content Engine is configured to use the 407 header when asking end users for their authentication credentials (user ID and password):

```
ContentEngine(config)# http authentication header 407
```


**Note**

In the ACNS 5.2.1 software and later releases, the ability to send HTTP transaction log messages to a remote syslog server is supported. This allows you to monitor the remote syslog server for HTTP request authentication failures in real time. For more information, see the [“Monitoring HTTP Request Authentication Failures in Real Time”](#) section on page 21-48.

## Specifying a Reauthentication Interval

In the ACNS 5.1 software, an inactivity timer was used to determine when the client browser was prompted to reenter authentication credentials after being initially authenticated. This inactivity timer is configured with the **http authentication cache timeout** global configuration command. By default, the inactivity timeout period was 8 hours. This meant that as long as someone continued to use the client browser after the legitimate authenticated user was initially authenticated, the client was not forced to reenter that person’s authentication credentials.

In the ACNS 5.2.1 software and later releases, you can specify an absolute TTL for HTTP authentication cache entries for increased security in a shared workstation environment. This ability to specify an absolute TTL timeout adds security to the inactivity timeout mechanism for content that is served through the Content Engine. When the absolute TTL period has expired, the client browser is forced to reauthenticate itself, and the user must enter valid credentials.

A security vulnerability exists in a shared workstation environment that is using WCCP-enabled router redirection mode with any authentication method, or proxy redirection mode and the NTLM authentication method. In these cases, the Content Engine uses the client’s IP address as the index into the authentication record kept in the authentication cache, and the Content Engine can therefore authenticate users who have not presented valid credentials of their own if a different user using the same workstation has previously presented valid credentials that are cached in the authentication cache. To provide additional security in this situation, you can configure the absolute TTL for HTTP authentication cache entries; this will specify the absolute time during which an authentication cache entry is valid in the cache. If a cache lookup occurs on an entry and its configured TTL time is exceeded, then the entry is deleted and the Content Engine will query the user for credentials.

To support this feature, the **tll** option was added to the **http authentication cache** global configuration command.

```
ContentEngine(config)# http authentication cache ?
  max-entries           Maximum entries in authentication cache; this is subject to
                        physical resources on the platform
  max-group-entries     Maximum entries in authentication group cache; this is
                        subject to physical resources on the platform
  timeout               Amount of time between last access and cache removal
  ttl                   Maximum amount of time from creation an entry is valid in
                        the cache
```

In order to use the absolute TTL effectively in a shared workstation environment that uses the Internet Explorer browser, there are additional considerations, because by default the browser will automatically send the workstation logon credentials when the Content Engine queries the user for credentials. Therefore, for the absolute TTL to provide additional security, either the logon credentials must not be valid request authentication credentials for the Content Engine or the Internet Explorer browser must configure its security settings to not send the logon credentials automatically when the Content Engine queries the user for credentials. To configure this on the Internet Explorer browser, choose **Tools > Internet Options > Security > Internet (and/or the other zones) > Custom Level > User Authentication > Logon > Prompt for Username and Password**. For this browser configuration to be effective, it must receive a 401 HTTP reply code when it is queried for credentials, which is the default in transparent redirection using WCCP. To have the 401 HTTP reply code used when in proxy redirection mode, use the Content Engine **http authentication header 401** global configuration command.



**Note** In a shared workstation environment, the assumption is that each user will close the browser before leaving the shared workstation. If a user does not close the browser session, other users can continue to use that browser session without being asked to enter their own username and password credentials because the browser automatically sends the credentials for a session that has already been authenticated with the proxy.

For non-shared workstations that use the Internet Explorer browser, configure the Internet Explorer browsers on these workstations to automatically send their logon credentials by choosing **Tools > Internet Options > Security > Internet (and/or the other zones) > Custom Level > User Authentication > Logon > Automatic logon only in Intranet zone (or Automatic logon with current username and password)**. The workstation logon credentials of the nonshared workstation users must be the same credentials that will successfully authenticate with the Content Engine. This explicit configuration of the browser is only needed if WCCP mode is being used. In proxy mode, the client browsers will always send the credentials.

This absolute TTL timeout (the **tll** option) allows you to specify an absolute value for the maximum time that an authentication cache entry is considered valid. The TTL timeout is specified in minutes (1–1440). The minimum is 1 minute; the maximum is 1440 minutes (24 hours). By default, this absolute TTL timeout option is disabled on the Content Engine.

Although a short absolute configuration value increases security in a shared workstation environment, it also increases the number of requests that must be made to the authentication server.



**Note** The existing inactivity timeout (the **timeout** option) is not affected by the absolute TTL (the **tll** option); they are independent of each other. If both timeouts (the inactivity timeout and TTL timeout) are configured on the Content Engine, the authentication cache entry times out because either the inactivity timeout or the TTL timeout occurred depending on which timeout occurs first for that entry.

## Displaying the Configuration of the HTTP Authentication Cache

To display the current configuration of the HTTP authentication cache on a standalone Content Engine, enter the **show http authentication EXEC** command.

```
ContentEngine# show http authentication
HTTP Authentication:
    Header: Based on URL syntax
    Realm: "Cisco Content Engine"
    Cache Timeout: 480 (minutes)
    Cache TTL: 240 (minutes)
    Cache Maximum entries configured: 8000
    Cache Maximum entries platform limit: 8000
    Group Cache Maximum entries configured: 500
    Group Cache Maximum entries platform limit: 1000
```

## Enabling the Caching of Authenticated Content

By default, authenticated content is not cached in HTTP. To change this default policy, use the **http cache-authenticated** global configuration command.

**http cache-authenticated {all | basic | ntlm}**

Table 7-5 describes these command parameters.

**Table 7-5 Parameters for the http cache-authenticated CLI Command**

Parameter	Description
<b>cache-authenticated</b>	Caches and revalidates authenticated web objects.
<b>all</b>	Caches web objects that were authenticated with any authentication scheme (basic authentication or NTLM authentication).
<b>basic</b>	Caches web objects that were authenticated with the basic authentication method. Basic authentication is less secure than NTLM authentication because it transmits the user credential information in clear text format.
<b>ntlm</b>	Caches web objects that were authenticated with the NTLM authentication method.

Enable caching of both basic and NTLM authenticated content on the Content Engine:

```
ContentEngine(config)# http cache-authenticated all
```

Verify which types of object caching are currently enabled on the Content Engine:

```
ContentEngine(config)# show http cache-authenticated all
Basic authenticated objects are cached.
NTLM authenticated objects are cached.
```

Enable NTLM object caching on a standalone Content Engine:

```
ContentEngine(config)# http cache-authenticated ntlm
```

When NTLM object caching is enabled on a Content Engine, when the Content Engine receives an NTLM HTTP request from a client, it searches its cache for the requested content. If a cache hit occurs, the Content Engine sends the client the requested content. If a cache miss occurs, the Content Engine retrieves the requested content from the origin server, caches a local copy of the content, and sends the requested content to the client.

The cached objects are tagged as NTLM protected so that subsequent requests for these same objects are subjected to authentication before the Content Engine can serve the content to the client. If there is a cache hit and the requested object is an NTLM protected object, the Content Engine checks whether there is a secured connection between this client and the server.

- If there is, the Content Engine sends an if-modified-since (IMS) request to the server using the proxy server connection.
- If the Content Engine receives a 304 response from the server, the Content Engine serves the cached content to the client. If the Content Engine receives a 200 response from the server, the Content Engine caches the new object and serves it to the client. If there is no established secure connection between the client and the server, the Content Engine attempts to establish the secure connection using IMS messages.

## Displaying HTTP Authentication Cache Statistics

To display authentication cache statistics for a standalone Content Engine, use the **show statistics http-authcache** EXEC command:

- **Dels TTL** shows the number of entries that were deleted from the authentication cache because of the absolute TTL timeout (the **http authentication cache ttl** command option).
- **DelTimeout** shows the number of entries that were deleted from the authentication cache because of the inactivity timer (the **http authentication cache timeout** command option).
- **Dels Other** shows the number of entries deleted from the authentication cache for all other reasons (for example, entries that were deleted because the **clear users request-authenticated** EXEC command was entered).

## Configuring Transparent HTTP Forward Proxy Caching for Standalone Content Engines

When transparent redirection is being used to direct HTTP requests to a Content Engine, you can configure the following caching services on the Content Engine:

- The standard web-cache service (service 0), as described in the [“Configuring the Standard Web-Cache Service \(Service 0\) for Standalone Content Engines”](#) section on page 7-18
- The custom-web cache service (service 98), as described in the [“Configuring the Custom Web-Cache Service \(Service 98\) for Standalone Content Engines”](#) section on page 7-20

You can use the Content Engine GUI or CLI to configure these caching services on a standalone Content Engine. If you use the Content Engine GUI to enable and configure the standard web-cache service (service 0) or the custom-web-cache service (service 98) on a standalone Content Engine, then you must specify the designated router list for each of these WCCP services in the following Content Engine GUI windows: the Web Cache window (**WCCP > Web Cache**), and the Custom Web Cache window (**WCCP > Custom Web Cache**). For more information on how to use the Custom Web Cache window, click the **HELP** button in the window.

## Configuring the Standard Web-Cache Service (Service 0) for Standalone Content Engines

The standard web-cache service (service 0) permits a single WCCP Version 1-enabled router or one or more WCCP Version 2-enabled routers to redirect HTTP traffic to standalone Content Engines on port 80 only. In order for a standalone Content Engine to accept redirected HTTP requests on port 80, you must configure the standard web-cache service on the Content Engine (transparent HTTP forward proxy caching).

To configure a standalone Content Engine to run the web-cache service (service 0) with WCCP Version 2, use the **wccp web-cache** global configuration command. To disable this service, use the **no** form of this command.

```
wccp web-cache {mask {[dst-ip-mask hex_num] [dst-port-mask port_hex_num] [src-ip-mask hex_num] [src-port-mask port_hex_num]} | router-list-num num [assign-method-strict] [l2-redirect] [mask-assign] [password key] [weight percentage]}
```

Table 7-6 describes the command parameters.

**Table 7-6** Parameters of the **wccp web-cache** CLI Command

Parameter	Description
<b>mask</b>	Sets the mask used for Content Engine assignment. Configure at least 1; the maximum is 4.
<b>dst-ip-mask</b>	(Optional) Sets the mask used to match the packet destination IP address.
<i>hex_num</i>	IP address mask defined by a hexadecimal number (for example, 0xFC000000). The range is 0x00000000–FC000000.
<b>dst-port-mask</b>	(Optional) Sets the mask used to match the packet destination port number.
<i>port_hex_num</i>	Destination port mask defined by a hexadecimal number (for example, 0xFC00). The port range is 0–65535.
<b>src-ip-mask</b>	(Optional) Sets the mask used to match the packet source IP address.
<b>src-port-mask</b>	(Optional) Sets the mask used to match the packet source port number.
<b>router-list-num</b>	Sets the router list number.
<i>num</i>	Router list number (1–8).
<b>assign-method-strict</b>	Forces WCCP to strictly use only the configured assignment method.
<b>l2-redirect</b>	(Optional) Packet forwarding by Layer 2 redirect.
<b>mask-assign</b>	(Optional) Uses the mask method for Content Engine assignment.
<b>password</b>	(Optional) Sets the authentication password.
<i>key</i>	WCCP service password key.
<b>weight</b>	(Optional) Sets weight percentage for load balancing.
<i>percentage</i>	Percentage value (0–100).

In the following example, the standard web-cache service (service 0) is configured on a standalone Content Engine and a WCCP-enabled router. By configuring this WCCP service on the router, the WCCP router redirects HTTP requests transparently to the Content Engine on port 80. By configuring this service on the Content Engine, the Content Engine listens on port 80 for redirected HTTP requests. If the Content Engine determines that it should accept and process the redirected HTTP request, it

retrieves the requested information from the origin server if it is not already stored in its cache, caches a copy of the content in its local storage if the content is cacheable, and then send the requested content to the client browser.

To configure the standard web-cache service (service 0) on a Content Engine and a WCCP router, follow these steps:

**Step 1** Enable WCCP Version 1 or Version 2 on the standalone Content Engine:

```
ContentEngine(config)# wccp version 1
```

or

```
ContentEngine(config)# wccp version 2
```

The Content Engine must be running WCCP Version 2 to support any of the WCCP services listed in [Table B-3](#) other than the web-cache service (service 0). If you enable WCCP Version 1 as opposed to Version 2 on this Content Engine, only a single WCCP router can be configured to support the only supported service (the standard web-cache service). If you select Version 2, up to eight WCCP routers can be specified to support a particular WCCP service, and all WCCP services are supported.



**Tip** You can also enable WCCP Version 1 or Version 2 on a standalone Content Engine by choosing **WCCP > Enable WCCP** from the Content Engine GUI and then clicking the **Version 1** or **Version 2** radio button.

**Step 2** Create a router list that specifies the routers that will support the web cache service. Enter the IP address or multicast address of every router that will support the web-cache service for this Content Engine. If different routers will be used for different WCCP services, you must create more than one router list. In this case, there is only one router on router list 1 (the router that you just configured for the standard web-cache service, which has an IP address of 10.0.1.1):

```
ContentEngine(config)# wccp router-list 1 10.0.1.1
```

If you use the Content Engine GUI to enable and configure WCCP on this Content Engine, then you must specify the designated router list for the web-cache service in the Web Cache window (**WCCP > Web Cache**) of the Content Engine GUI.

**Step 3** Inform the WCCP-enabled router in the specified router list that this standalone Content Engine is accepting redirected web cache requests on port 80:

```
ContentEngine(config)# wccp web-cache router-list-num 1
```

**Step 4** Exit global configuration mode.

```
ContentEngine(config)# exit
```

**Step 5** Write the running configurations to nonvolatile memory.

```
ContentEngine# write memory
```

**Step 6** Enable WCCP Version 2 on the router.

```
Router# configure terminal
Router(config)# ip wccp version 2
```

**Step 7** Enable the standard web-cache service on the router.

```
Router(config)# ip wccp web-cache
```

**Step 8** Specify the interface on which the standard web-cache service will run.

```
Router(config)# interface type number
```

In the following example, Ethernet interface 0/1 on the router is configured to run the standard web-cache service:

```
Router(config)# interface ethernet 0/1
```

**Step 9** Configure the router to check the HTTP traffic that arrives on the interface on which the standard web-cache service is configured (for example, Ethernet interface 0/1). The router will determine whether it should redirect these packets to the standalone Content Engine. This Content Engine functions as a transparent forward proxy server that will accept redirected HTTP requests on port 80 from this WCCP Version 2 router.

```
Router(config-if)# ip wccp web-cache redirect in
```

## Configuring the Custom Web-Cache Service (Service 98) for Standalone Content Engines

To enable a standalone Content Engine to accept redirected HTTP traffic on a port other than 80, configure the Content Engine to support the custom-web-cache service (service 98).

The **wccp custom-web-cache** global configuration command causes the Content Engine to establish WCCP Version 2 redirection services automatically with a Cisco router on a user-specified port number. The Content Engine then performs transparent web caching for all HTTP requests over that port while port 80 transparent web caching continues without interruption. For custom web caching, the custom-web-cache service (service 98) must be enabled on routers that are running WCCP Version 2. WCCP Version 1 does not support the custom-web-cache service.

To configure the custom-web-cache service on a standalone Content Engine, use the **wccp custom-web-cache** global configuration command. To disable the custom-web-cache service (service 98) on the Content Engine, use the **no** form of this command.

```
wccp custom-web-cache {mask {[dst-ip-mask hex_num] [dst-port-mask port_hex_num] [src-ip-mask hex_num] [src-port-mask port_hex_num]} | router-list-num num port port [assign-method-strict] [hash-destination-ip] [hash-destination-port] [hash-source-ip] [hash-source-port] [I2-redirect] [mask-assign] [password key] [weight percentage]}
```

Table 7-7 describes these command parameters.

**Table 7-7 Parameters of the wccp custom-web-cache CLI Command**

Parameter	Description
<b>mask</b>	Sets the mask used for Content Engine assignment. Configure at least 1 and up to 4.
<b>dst-ip-mask</b>	(Optional) Sets the mask used to match the packet destination IP address.
<i>hex_num</i>	IP address mask defined by a hexadecimal number (for example, 0xFC000000). The range is 0x00000000–FC000000.
<b>dst-port-mask</b>	(Optional) Sets the mask used to match the packet destination port number.
<i>port_hex_num</i>	Port mask defined by a hexadecimal number (for example, 0xFC00). The port range is 0–65535.

**Table 7-7 Parameters of the wccp custom-web-cache CLI Command (continued)**

Parameter	Description
<b>src-ip-mask</b>	(Optional) Sets the mask used to match the packet source IP address.
<b>src-port-mask</b>	(Optional) Sets the mask used to match the packet source port number.
<b>router-list-num</b>	Sets router list number.
<i>num</i>	Router list number (1–8).
<b>port</b>	Sets port number.
<i>port</i>	Port list number (1–65535).
<b>assign-method-strict</b>	Forces WCCP to strictly use only the configured assignment method.
<b>hash-destination-ip</b>	(Optional) Defines load-balancing hash of the destination IP address (the default).
<b>hash-destination-port</b>	(Optional) Defines load-balancing hash of the destination port.
<b>hash-source-ip</b>	(Optional) Defines load-balancing hash of the source IP address.
<b>hash-source-port</b>	(Optional) Defines load-balancing hash of the source port.
<b>l2-redirect</b>	(Optional) Packet forwarding by Layer 2 redirect.
<b>mask-assign</b>	(Optional) Uses the mask method for Content Engine assignment.
<b>password</b>	(Optional) Sets authentication password.
<i>key</i>	WCCP service password key.
<b>weight</b>	(Optional) Sets weight percentage for load balancing.
<i>percentage</i>	Percentage value (0–100).

The **l2-redirect** option permits the Content Engine to receive transparently redirected traffic from a WCCP Version 2-enabled router or switch if the Content Engine has a Layer 2 connection with the device and the device is configured for Layer 2 redirection.

The **weight** parameter represents a percentage of load redirected to the Content Engine cluster (for example, a Content Engine with a weight of 30 receives 30 percent of the total load). If the total of all weight parameters in the Content Engine cluster exceeds 100, the percentage load for each Content Engine is recalculated as the percentage that its weight parameter represents of the combined total.

In the following example, a standalone Content Engine and a router that are both running WCCP Version 2 are configured to support the custom-web-cache service (service 98). By configuring this WCCP caching service, the WCCP Version 2 routers can redirect HTTP traffic transparently to a Content Engine (transparent forward proxy server) on multiple ports (up to eight ports). By configuring the custom-web-cache service on the Content Engine and router, the Content Engine can intercept HTTP requests on multiple ports without having to configure a user-defined WCCP service (services 90 to 97). For more information about configuring user-defined WCCP services on a Content Engine and a router, respectively, see the [“Configuring WCCP Services on Standalone Content Engines”](#) section on page 6-14 and the [“Configuring WCCP Services on a Router”](#) section on page 6-27.

To configure the custom-web-cache service (service 98) on a standalone Content Engine and a router, follow these steps:

**Step 1** Enable WCCP Version 2 on the standalone Content Engine:

```
ContentEngine(config)# wccp version 2
```

The Content Engine must be running WCCP Version 2 to support the custom-web-cache service (service 98). WCCP Version 2 is required for any of the WCCP services listed in [Table B-3](#) other than the standard web-cache service (service 0).



**Tip** You can also enable WCCP Version 2 on a standalone Content Engine by choosing **WCCP > Enable WCCP** from the Content Engine GUI and then clicking the **Version 1** or **Version 2** radio buttons.

**Step 2** Create a router list that specifies the routers that will support the custom-web-cache service (service 98). Enter the IP address or multicast address of every router that will support the custom-web-cache service for this Content Engine. If different routers will be used for different WCCP services, you must create more than one router list.

In this case, there is only one router on router list 1 (the router that you just configured for the custom-web-cache service, which has an IP address of 10.0.1.1).

```
ContentEngine(config)# wccp router-list 1 10.0.1.1
```

If you use the Content Engine GUI to enable and configure WCCP on this Content Engine, then you must specify the designated router list for the custom-web-cache service in the Custom Web Cache window (**WCCP > Custom Web Cache**) of the Content Engine GUI.

**Step 3** Inform the WCCP-enabled router in the specified router list that this standalone Content Engine is accepting redirected custom web cache requests on port 31.

```
ContentEngine(config)# wccp custom-web-cache router-list-num 1 port 31
```

**Step 4** Exit global configuration mode.

```
ContentEngine(config)# exit
```

**Step 5** Write the running configurations to nonvolatile memory.

```
ContentEngine# write memory
```

**Step 6** Enable WCCP Version 2 on the router.

```
Router# configure terminal
Router(config)# ip wccp version 2
```

**Step 7** Enable the custom-web-cache service (service 98) on the router.

```
Router(config)# ip wccp 98
```

**Step 8** Specify the interface on which the custom-web-cache service will run.

```
Router(config)# interface type number
```

In the following example, the Ethernet 0 interface on the router is configured to run the custom-web-cache service:

```
Router(config)# interface ethernet 0
```

**Step 9** Configure the router to use the outbound interface for the custom-web-cache service.

```
Router(config-if)# ip wccp 98 redirect out
```

---

## Configuring HTTP Reverse Proxy Caching for Standalone Content Engines

The reverse-proxy service (service 99) is the predefined WCCP Version 2 service that permits WCCP Version 2-enabled routers to redirect reverse proxy packets to a standalone Content Engine that is functioning as a transparent reverse proxy server.

This section provides an example of how to configure this WCCP service with a WCCP Version 2-enabled router and a standalone Content Engine. In this example, WCCP Version 2 and the reverse-proxy service (service 99) are used to redirect HTTP reverse proxy requests transparently to the Content Engine on port 80. By configuring this service on the Content Engine, the Content Engine processes the redirected HTTP reverse proxy requests. As part of processing the request, the Content Engine retrieves the requested information from the origin server if it is not already stored in its cache, caches a local copy, and sends the requested content to client browser.

To configure the reverse-proxy service (service 99) on a standalone Content Engine and a WCCP router, follow these steps:

**Step 1** Enable WCCP Version 2 on the standalone Content Engine.

```
ContentEngine(config)# wccp version 2
```

The Content Engine must be running WCCP Version 2 to support the reverse-proxy service. WCCP Version 1 only supports the standard web-cache service (service 0); it does not support the reverse-proxy service (service 99). See [Table B-3](#) for a list of the WCCP services.



**Tip** You can also enable WCCP Version 2 on a standalone Content Engine by choosing **WCCP > Enable WCCP** from the Content Engine GUI and then clicking the Version 2 radio buttons.

---

**Step 2** Create a router list that specifies the routers that will support the reverse-proxy service. Enter the IP address or multicast address of every router that will support this WCCP service. If different routers will be used for different WCCP services, you must create more than one router list.

In this example, there is only one router on router list 1 (the router that you just configured for the reverse proxy cache service, which has an IP address of 10.0.1.1):

```
ContentEngine(config)# wccp router-list 1 10.0.1.1
```

If you use the Content Engine GUI to enable and configure WCCP on this Content Engine, then you must specify the designated router list for the reverse-proxy service in the Reverse Proxy window (**WCCP > Reverse Proxy**).

**Step 3** Inform the WCCP-enabled router in the specified router list that this standalone Content Engine is accepting redirected reverse proxy cache requests on port 80.

```
ContentEngine(config)# wccp reverse-proxy router-list-num 1
```

**Step 4** Exit global configuration mode.

```
ContentEngine(config)# exit
```

**Step 5** Write the running configurations to nonvolatile memory.

```
ContentEngine# write memory
```

**Step 6** Enable WCCP Version 2 on the router.

```
Router# configure terminal
Router(config)# ip wccp version 2
```

**Step 7** Enable the reverse-proxy service (service 99) on the router.

```
Router(config)# ip wccp 99
```

**Step 8** Specify the interface on which the reverse-proxy service will run.

```
Router(config)# interface type number
```

In the following example, the Ethernet 0 interface on the router is configured to run the reverse-proxy service:

```
Router(config)# interface ethernet 0
```

Configure the router to use the outbound interface for the reverse-proxy service. The router will check the reverse proxy packets on Ethernet interface 0 to determine if it should transparently redirect these packets to the Content Engine the (transparent reverse proxy server).

```
Router(config-if)# ip wccp 99 redirect out
```

---

## Configuring HTTPS Caching for Standalone Content Engines

The HTTPS protocol is essentially the HTTP protocol running over a Secure Socket Layer (SSL) transport. SSL is a protocol that provides a secure channel between two devices (for example, a client and a server). SSL uses public-key cryptography to ensure the security and privacy of the exchange between the client browser and the server. HTTPS uses a unique URL that begins with https:// (for example, https://abc.com). The default port number for HTTPS is port 443 instead of port 80, which is the default port for HTTP.

Content Engines can either SSL terminate or tunnel HTTPS client requests to the origin HTTPS server, as described in the following sections:

- [About SSL Termination of HTTPS Client Requests, page 7-25](#)
- [About Tunneling of HTTPS Client Requests, page 7-25](#)

For information about how to configure the different types of HTTPS caching on a standalone Content Engine, see:

- [Configuring HTTPS Proxy Caching for Standalone Content Engines, page 7-25](#)
- [Configuring HTTPS Transparent Caching for Standalone Content Engines, page 7-27](#)

## About SSL Termination of HTTPS Client Requests

If the Content Engine SSL terminates an HTTPS client request, this means that it decrypts the SSL-encrypted data. By decrypting the SSL-encrypted data, the Content Engine can see the HTTPS client request in plain text, which allows the Content Engine to perform numerous HTTP processing tasks (for example, caching, rule processing, and filtering) on such requests. Content Engines running the ACNS 5.1 software and later releases, support this SSL-termination feature. Content Engines running the ACNS 5.2 software and later releases, can also rewrite and redirect HTTPS requests as defined by the specified rules. If the Content Engine terminates an HTTPS request, it can apply most of the rules on the HTTPS request. (The **no-proxy**, **use-icap-service**, and **use-proxy** rule actions are not supported for HTTPS caching.) For more information, see [Chapter 13, “Configuring the Rules Template on Standalone Content Engines.”](#)

For the SSL-termination feature to work properly, you must install the SSL certificate and private key of the origin HTTPS servers on the Content Engine. The SSL-termination feature works in both transparent mode and proxy mode if the Content Engine has the correct certificates and private keys installed. For specific requested content to be cached, you must import the proper certificates and keys for these origin HTTPS servers into the Content Engine and configure the Content Engine to cache content from these origin HTTPS servers. For standalone Content Engines, this is performed through the Content Engine CLI, as described in the [“Configuring Certificates and Private Keys for HTTPS Caching” section on page 7-32.](#)

In the ACNS 5.2.1 software and later releases, the Content Engine SSL terminates HTTPS requests in WCCP mode and in manual proxy mode if the requested HTTPS servers are configured on the Content Engine, and tunnels the rest of the HTTPS traffic.

## About Tunneling of HTTPS Client Requests

Tunneling of HTTPS request is another mode that the ACNS software supports for HTTPS client requests. In this mode, the Content Engine can only support limited processing on the HTTPS client requests (for example, no caching support, and only limited filtering support).

The ACNS 3.0 software and later releases support HTTPS tunneling through the CONNECT method, which is the standard HTTPS tunneling method that is defined in the HTTP specification.

The ACNS 5.1.5 software and later releases support tunneling of transparently redirected native HTTPS traffic. This means that the Content Engine can accept native HTTPS traffic from the client and tunnel such requests to the origin HTTPS server. Even though the Content Engine can apply filtering on such traffic, it does not see the actual HTTPS content because the certificate and private key of the origin HTTPS server is not installed on the Content Engine.

## Configuring HTTPS Proxy Caching for Standalone Content Engines

With HTTPS proxy caching, direct proxy routing is used to direct HTTPS requests (HTTPS-over-HTTP from client browsers) directly to the Content Engine. The Content Engine functions as a nontransparent forward proxy server (nontransparent HTTPS proxy server) for these client browsers. For information about how to configure a client browser to point directly to a Content Engine, see the [“Pointing Client Browsers Directly to a Standalone Content Engine” section on page 4-36.](#)

Configure the Content Engine to operate in HTTPS proxy mode to allow the Content Engine to service HTTPS requests from client browsers that have been configured to use this Content Engine as their HTTPS proxy server.

**Note**

In order to support incoming HTTPS proxy requests, a DNS server must be configured (as described in the “[Configuring DNS Servers for the DNS Caching Service \(Service 53\)](#)” section on page 7-64).

From the Content Engine GUI, choose **Caching > HTTPS Proxy**, and use the displayed HTTPS Proxy window. To obtain more information about how to use the HTTPS Proxy window to configure HTTPS proxy caching, click the **HELP** button in the window.

From the Content Engine CLI, use the **https proxy** global configuration command, as described in [Table 7-8](#). The order in which the CLI commands are entered is not important.

**Table 7-8** *HTTPS Proxy Features and the Related CLI Command*

Content Engine CLI Commands (Abbreviated Syntax)	HTTPS Proxy Features
<b>https proxy incoming</b> <i>ports 1–65535, ports, . . .</i>	Supports up to eight incoming proxy ports.
<b>https proxy incoming</b> <i>ports 1–65535</i> <b>wccp custom-web-cache</b> . . .	Configures HTTPS incoming proxy ports and the custom-web-cache service to intercept HTTPS-over-HTTP requests.
<b>no https destination-port allow 443 563</b> <b>or</b> <b>https destination-port deny all</b>	Denies unwanted access to any destination HTTPS port.
<b>proxy-protocols outgoing-proxy exclude list</b> <i>word https proxy outgoing host {hostname   ip_address} port 1–65535</i>	Configures an outgoing HTTPS proxy server, using the global <b>exclude</b> option.
<b>proxy-protocols transparent default-server</b>	Uses the default outgoing HTTPS proxy server, if available.
<b>proxy-protocols transparent original-proxy</b>	Uses the outgoing HTTPS proxy server from the original request.
<b>proxy-protocols transparent reset</b>	Returns the incoming HTTPS request to the sending client during a cache miss.

The following example shows how to configure a standalone Content Engine (Content Engine A) to support HTTPS-over-HTTP proxy caching:

**Step 1** Configure the port numbers for the incoming proxy-mode requests.

```
ContentEngineA(config)# https proxy incoming ports
```

where *ports* are the ports (in addition to port 80) on which the standalone Content Engine will accept incoming HTTPS requests from client browsers. This number ranges from 1 to 65535. You can specify up to eight ports.

This example configures the Content Engine to accept HTTPS requests on ports 8080, 8081, and 9090: If multiple ports are entered, separate each port number with a blank space.

```
ContentEngineA(config)# https proxy incoming 8080 8081 9090
```

**Note**

If multiple ports are entered, separate each port number with a blank space.

Remember that if you use this command to configure the Content Engine to accept HTTPS-over-HTTP requests on a port other than port 80, you must also configure the client browsers to send their HTTPS requests to that port on the Content Engine. For more information, see the [“Pointing Client Browsers Directly to a Standalone Content Engine”](#) section on page 4-36.

**Step 2** Designate a proxy server as the primary outgoing HTTPS proxy server for the Content Engine:

```
ContentEngineA(config)# https proxy outgoing host host port primary
```

- *host* is the hostname or IP address of the parent cache (outgoing HTTPS proxy server) to which HTTPS missed traffic is directed.
- *port* is the port number used by the parent cache to accept cache miss HTTPS-over-HTTP requests from the Content Engine.

Use the **primary** keyword to set the specified host as the primary outgoing HTTPS proxy server. If several servers (hosts) are configured with the **primary** keyword, the last one configured becomes the primary outgoing HTTPS proxy server for the Content Engine.

In this example, Content Engine A is configured to send its cache miss HTTPS traffic (cache misses for browser HTTPS-over-HTTP requests) to the host 10.1.1.1 on port 8088. Host 10.1.1.1 is explicitly designated as the primary outgoing HTTPS proxy server for Content Engine A. Host 10.1.1.2 is configured as a backup outgoing HTTPS proxy server for Content Engine A.

```
ContentEngineA(config)# https proxy outgoing host 10.1.1.1 8088 primary  
ContentEngineA(config)# https proxy outgoing host 10.1.1.2 220
```

In the ACNS 5.1.x software and earlier releases, you could only configure the Content Engine to use one outgoing HTTPS proxy server. With the ACNS 5.2.1 software and later releases, you can specify up to eight outgoing proxy servers for HTTPS-over-HTTP proxy requests. The benefit of supporting up to eight outgoing HTTPS proxy servers is that if one outgoing proxy fails, the Content Engine will fail over to the next specified outgoing proxy server, thereby providing redundancy. All outgoing requests will be directed to the primary outgoing proxy server. If the primary proxy server fails, requests are directed to the next active proxy server on the list.

**Step 3** View the current state of each HTTPS proxy server that is currently configured on the standalone Content Engine.

```
ContentEngineA# show https proxy
```

---

## Configuring HTTPS Transparent Caching for Standalone Content Engines

To configure a Content Engine to support HTTPS transparent caching, you must configure the Content Engine and a router support the WCCP Version 2 HTTPS caching service (the https-cache service [service 70]). The https-cache service is the WCCP HTTPS caching service that permits WCCP Version 2-enabled routers to intercept port 443 TCP traffic and redirect this HTTPS traffic to the Content Engine (that is acting as a transparent forward proxy server, which is configured for HTTPS transparent caching). The Content Engine retrieves the requested content, stores a copy locally (performs HTTPS transparent caching) if the content is cacheable, and serves the requested content to the client.

In the ACNS 5.1.x software, only one interception mode (the accept-only mode) was supported for the https-cache service. With the accept-only mode, the Content Engine can only accept requests that are directed to configured HTTPS servers.

```
ContentEngine(config)# wccp https-cache router-list-num 1
```

or

```
ContentEngine(config)# wccp service-number 95 router-list-num 1 port-list 1 https-cache
```

In both of the preceding examples, the Content Engine would only accept redirected HTTPS traffic if the HTTPS server was configured on the Content Engine (that is, you have used the **https server** global configuration command to specify the IP address or hostname and the private key and certificate of the origin HTTPS server on the Content Engine).

In the ACNS 5.2 software, another interception mode (the accept-all mode) was added for the https-cache service (service 70). When the accept-all mode is enabled, the Content Engine intercepts all HTTPS requests regardless of whether the origin HTTPS server is configured on the Content Engine. If the private key or certificate of the origin HTTPS server is not configured on the Content Engine, the Content Engine tunnels the request to the origin HTTPS server instead of SSL terminating the request. (In the ACNS 5.1 software, the Content Engine would bypass HTTPS requests that were directed to HTTPS servers that it had not been explicitly configured to support.)

The accept-all mode was added to support the filtering of HTTPS traffic.

```
ContentEngine(config)# wccp https-cache ?
  accept-all      Accept all https traffic by default
  mask            Specify mask used for CE assignment
  router-list-num Router list number
```

The accept-all mode works the same way as the traditional WCCP services do (for example, the standard web-cache service [service 0] that intercepts all web traffic by default). If the **wccp https-cache accept-all** option is used, the HTTPS cache (the Content Engine that has the **https-cache** service configured and enabled) operates in accept all mode (that is, all HTTPS traffic is intercepted by the Content Engine). If the **wccp https-cache accept-all** option is not used, the Content Engine operates in accept only mode.

The Content Engine listens for redirected HTTPS requests on the standard HTTPS port (default port 443). To intercept HTTPS traffic on ports other than the default port, configure a user-defined WCCP service (services 90 to 97). For more information on this topic, see the [“Configuring Standalone Content Engines to Support User-Defined WCCP Services”](#) section on page 6-15.

The following example shows how to configure HTTPS transparent caching on a standalone Content Engine and a single router running WCCP Version 2:

---

**Step 1** Configure the router to support the https-cache service (service 70).

- a. Enable WCCP Version 2 on the router (for example, Router A).

```
RouterA# configure terminal
RouterA(config)# ip wccp version 2
```

- b. Configure Router A to run the https-cache service (service 70).

```
RouterA(config)# ip wccp 70
```

- c. Configure Router A to intercept all outgoing HTTPS traffic.

```
Router(config)# ip wccp 70 redirect out
```

**Step 2** Configure the Content Engine to support the https-cache service.

- a. Configure the list of routers that will be used to support the https-cache service. In this case, router list number 1 is created and consists of only one router (Router A, which has an IP address of 10.2.202.1).

```
ContentEngine(config)# wccp router-list 1 10.1.202.1
```

- b. Configure the Content Engine to accept transparently redirected HTTPS requests from the routers listed in router list 1 (Router A).

```
ContentEngine(config)# wccp https-cache router-list-num 1
```

- c. Enable WCCP Version 2 on the Content Engine.

```
ContentEngine(config)# wccp version 2
```

- d. If you want the Content Engine to intercept all HTTPS traffic and tunnel the HTTPS traffic for which it does not have a private key or certificate, then enable the accept-all mode on the Content Engine. This feature is typically used for filtering purposes (for example, to enable the Content Engine to use SmartFilter or Websense software to filter tunneled HTTPS requests).

```
ContentEngine(config)# wccp https-cache accept-all
```

**Step 3** Load the private key or certificates of the HTTPS origin servers from which the Content Engine will cache content (that is, act as the HTTPS origin server) instead of tunneling the HTTPS requests to the origin HTTPS server.

To load the private key or certificate on to the Content Engine, use the **https EXEC** command.

```
ContentEngine# https ?
cert      Certificate management commands
certgroup Certificate chain management commands
key       Private key management commands
```

For more information, see the [“Configuring Certificates and Private Keys for HTTPS Caching”](#) section on page 7-32.

**Step 4** (Optional). Create a certificate group on the Content Engine. This feature is used for HTTPS requests that are SSL terminated by the Content Engine. For more information, see the [“Configuring Certificate Groups for HTTPS Caching”](#) section on page 7-33.

**Step 5** Configure the basic HTTPS server settings on the Content Engine. These basic settings must be set in order to enable HTTPS transparent caching on a Content Engine.

- a. Use the **https server** *server-name* global configuration command to specify the server name of an origin HTTPS server from which the Content Engine caches content. For example, in the following example, the origin HTTPS server named “abc1” is configured on the Content Engine. After you specify the server name, the submode for HTTPS configuration is invoked and the prompt changes to ContentEngine(config-https)#.

```
ContentEngine(config)# https server abc1
ContentEngine(config-https)#
```

- b. From HTTPS configuration submode, enter the certificate, the private key, and hostname of the HTTPS server (abc1), and then enable these settings (enter **enable** from the submode, as shown below) on the Content Engine. These are the minimal settings that you need to specify to enable caching of content of the specified HTTPS server.

```
ContentEngine(config-https)# cert ?
WORD Certificate name
ContentEngine(config-https)# key ?
WORD Private key name
ContentEngine(config-https)# host ?
WORD FQDN or ip address of the origin HTTPS server
ContentEngine(config-https)# enable
```

Use the **cert** and **key** command options to configure a Content Engine to use a set of SSL certificates and keys that will enable the Content Engine to act as the origin HTTPS server. The Content Engine will be able to decode HTTPS traffic from a client and perform normal HTTP operations on it, such as caching and request processing. The Content Engine will be able to initiate HTTPS connections to an origin server and fetch content from origin servers upon cache miss (or cache validation). For more information, see the “[Configuring HTTPS Server Settings on Standalone Content Engines](#)” section on page 7-31.

- Step 6** (Optional) Configure certain advanced HTTPS server settings on the Content Engine to provide more detailed control of various aspects of SSL communication.

- a. Configure certificate groups. For more information, see the “[Configuring Certificate Groups for HTTPS Caching](#)” section on page 7-33.
- b. Configure the Content Engine to use SSL Version 2 only. Enter the **protocol-version sslv2-only** command option or the **https server name protocol-version sslv2-only** global configuration command:

```
ContentEngine(config-https)# protocol-version ?
sslv2-only    Only use and understand SSL v2 protocol
sslv23-tlsv1  Use and understand SSL v2/v3 and TLS v1 protocols (default)
sslv3-only    Only use and understand SSL v3 protocol
tlsv1-only    Only use and understand TLS v1 protocol
```

or

```
ContentEngine(config)# https server name protocol-version ?
sslv2-only    Only use and understand SSL v2 protocol
sslv23-tlsv1  Use and understand SSL v2/v3 and TLS v1 protocols (default)
sslv3-only    Only use and understand SSL v3 protocol
tlsv1-only    Only use and understand TLS v1 protocol
```

- c. (Optional) Change the default HTTPS server authentication settings to meet the needs of your particular environment. For example, use the **serverauth ignore** command option (shown below), or the **https server name serverauth ignore** global configuration command to ignore particular errors in HTTPS server authentication.

```
ContentEngine(config-https)# serverauth ignore ?
cert-not-yet-valid  Ignore errors caused by using the certificate be
                    valid
domain-name         Ignore errors due to domain name mismatch
expired-date        Ignore certificate expiration errors
invalid-ca          Ignore errors caused by an unrecognized CA
```

- d. (Optional) Change the default session cache settings, and specify settings that meet your specific requirements by using the **session-cache** command option or the **https server name session-cache** global configuration command. For example, use the **size** option to specify the SSL session cache size setting. The default is 10,000 entries. Valid values are 0 to 20,000 entries.

```
ContentEngine(config-https)# session-cache ?
  size      SSL session cache size setting
  timeout   SSL session cache timeout setting
```

## Configuring HTTPS Server Settings on Standalone Content Engines

After you have specified the name of an origin HTTPS server on a Content Engine using the **https server server-name** global configuration command, you can specify a set of parameters for the configured HTTPS server (see the following example). You can specify these parameters from the HTTPS configuration submode or from global configuration mode.

```
ContentEngine(config)# https server abc1
ContentEngine(config-https)# ?
  cert          Select certificate to use for the HTTPS server
  certgroup     Select certificate chains for the HTTPS server
  enable        Enable caching of the HTTPS server
  host          Input hostname or ip address of the origin HTTPS server
  key           Select private key to use for the HTTPS server
  protocol-version SSL protocol versions for client/server communication
  serverauth    HTTPS server authentication commands
  session-cache SSL session caching parameters tuning
<cr>
```

Table 7-9 describes the global configuration commands for configuring the HTTPS server settings on a standalone Content Engine.

**Table 7-9 CLI Commands for Configuring HTTPS Transparent Caching**

Command	Purpose
<b>https server name cert</b> <b>https server name key</b>	Configures a Content Engine to use a set of SSL certificates and keys to let the Content Engine act as the origin HTTPS server. The Content Engine can decode HTTPS traffic from a client and perform normal HTTP operations on it, such as caching and request processing. The Content Engine can initiate HTTPS connections to an origin server and fetch content from origin servers upon cache miss (or cache validation).
<b>https server name host</b>	Specifies the IP address for the origin HTTPS server. Use the IP address of the Content Engine in a central office acting as the HTTPS server when using this command. The <b>https server name enable</b> global configuration command enables the use of this HTTPS server.
<b>https server name protocol-version</b>	Sets the SSL protocol version used to control communication between the client and the HTTPS server.

**Table 7-9 CLI Commands for Configuring HTTPS Transparent Caching (continued)**

Command	Purpose
<b>https server</b> <i>name</i> <b>serverauth</b>	Allows the use of authentication to reach the HTTPS server. You can also configure the authentication to ignore authentication errors such as invalid certification, domain name mismatches, certificate expiration errors, and unrecognized Certificate Authorities (CAs).  By default, HTTPS server authentication is enabled on the Content Engine. Use the <b>ignore</b> command option to ignore particular errors in HTTPS server authentication.
<b>https server</b> <i>name</i> <b>session-cache</b>	Specifies the size and timeout for the SSL session cache on the Content Engine.  Use the <b>size</b> option to specify the SSL session cache size setting. The default is 10,000 entries. Valid values are 0 to 20,000 entries.  Use the <b>timeout</b> option to specify the SSL session cache timeout setting. The default is 3,600 seconds. Valid values are 1 to 86,400 seconds.

## Configuring Certificates and Private Keys for HTTPS Caching

In the ACNS 5.1. software and later releases, the Content Engine CLI can be used to configure certificates and private keys on the Content Engine in order to support HTTPS caching. (The Content Engine GUI does not currently support this configuration capability.)

Use the **https EXEC** command to create, remove, and import certificates and private keys when using a standalone Content Engine as an HTTPS server. [Table 7-10](#) describes the parameters for the **https cert** *cert-name* and **https key** *key\_name* EXEC commands.

**Table 7-10 Parameters for the https cert and https key EXEC Commands**

Parameter	Description
<b>cert</b>	Enables creating, removing, and importing certificates.
<i>cert-name</i>	Name of the certificate.
<b>add-cert</b>	Adds a certificate from an external source.
<b>create</b>	Defines the name for a certificate.
<b>remove</b>	Removes a certificate with a given name.
<b>key</b>	Enables creating, removing, and importing a private key.
<i>key_name</i>	Name of the public and private key pair.
<b>create</b>	Defines the name for a private key.
<b>import</b>	Imports a private key from an external source.
<b>remove</b>	Removes a key with a given name.
<b>url</b>	Enables the use of a URL to point to the location of the private key or certificate.
<i>URL</i>	URL (HTTP, FTP-over-HTTP, or HTTPS) that points to the location of the private key or certificate.

You can assign a certificate and associate a key with the HTTPS server assuming that you have configured the Content Engine with the **https server** global configuration command. The Content Engine presents the certificate to HTTPS clients that make requests to the HTTPS server.

Use the **https cert** EXEC command to create certificates with a given name, or to import a certificate from external sources, or to remove existing certificates from the Content Engine.


**Note**

Private keys and certificates must be in the Privacy-Enhanced Mail (PEM) format. If the private keys and certificates are in Public Key Cryptography Standards 12 (PKCS12) format, the Content Engine will convert them internally to PEM format when importing the private keys or certificates.

## Configuring Certificate Groups for HTTPS Caching

In the ACNS 5.1 software and later releases, the Content Engine CLI can be used to configure certificate groups on the Content Engine in order to support HTTPS caching. (The Content Engine GUI does not currently support this configuration capability.)

Certificate groups are formed to represent a trust relationship chain from root Certificate Authority to end entity. Each one of the certificates in a certificate group except the end entity's certificate signs and trusts the next one in the chain. An end entity's certificate can be trusted only if all certificates in the certificate group leading to this certificate can be trusted. A certificate group can be used to represent an HTTPS server just like a single certificate, but with the added benefit that the client does not need to have all certificates locally. A certificate group can also be used to verify and authenticate an HTTPS server by comparing the server's certificates to the certificate group.

Use the **https certgroup** *cert-name* EXEC command to create, remove, or form certificate groups on a standalone Content Engine. [Table 7-11](#) describes the parameters for the **https certgroup** *cert-name* EXEC command.

**Table 7-11** Parameters for the **https certgroup** EXEC Command

Parameter	Description
<b>certgroup</b>	Enables adding, creating, or removing a certificate group.
<i>cert-name</i>	Name of the certificate group.
<b>import</b>	Adds a chain of certificates to the certificate group.
<b>create</b>	Creates a certificate group with the specified name.
<b>remove</b>	Removes a certificate group with the specified name.

## Configuring HTTPS Outgoing Proxy Servers for Standalone Content Engines

When you use the **https proxy outgoing host** global configuration command to configure the Content Engine to use an HTTPS outgoing proxy, all incoming HTTPS requests are directed to this outgoing proxy. The **proxy-protocols outgoing-proxy exclude** global configuration command specifies a global proxy exclude domain effective for all proxy server protocols, including HTTPS.

The Content Engine applies the following logic when an outgoing proxy is configured:

- If the destination server is specified by the global exclude option, then go directly to the destination server.
- If the destination server is not specified by the global exclude option and the request is not HTTPS, go directly to the destination server.
- If the destination server is not specified by the global exclude option, then go to the outgoing proxy server.

When a Content Engine intercepts a proxy request intended for another proxy server and there is no outgoing proxy configured for HTTPS, and the **proxy-protocols transparent default-server** global configuration command is invoked, the Content Engine addresses the request to the destination server directly and not to the client's intended proxy server. However, if the **proxy-protocols transparent reset** global configuration command is used on the Content Engine and a cache miss occurs, all transparently intercepted requests sent by clients are returned to the client and requested objects are not delivered. For more information about configuring HTTPS outgoing proxy exclusion settings on standalone Content Engines, see the [“Configuring HTTP and HTTPS Outgoing Proxy Exclusion Settings” section on page 14-5](#).

The following example shows how to configure the Content Engine (which is acting as a nontransparent forward proxy) as an HTTPS proxy server and have it listen on port 8081 for HTTPS requests from client browsers:

```
ContentEngine(config)# https proxy incoming 8081
```

In this example, Content Engine A is configured to send its missed HTTPS traffic (cache misses for browser requests for HTTPS content [HTTPS-over-HTTP requests]) to the host 10.1.1.1 on port 8088. Host 10.1.1.1 is explicitly designated as the primary outgoing HTTPS proxy server for Content Engine A. Host 10.1.1.2 is configured as a backup outgoing HTTPS proxy server for Content Engine A.

```
ContentEngineA(config)# https proxy outgoing host 10.1.1.1 8088 primary
ContentEngineA(config)# https proxy outgoing host 10.1.1.2 220
```

In the ACNS 5.1.x software and earlier releases, you could only configure the Content Engine to use one outgoing HTTPS proxy server. In the ACNS 5.2.1 software release, the ability to configure the Content Engine to use up to eight outgoing HTTPS proxy servers was added. For more information on this topic, see [Chapter 14, “Configuring Primary and Backup Proxy Servers for Standalone Content Engines.”](#)

In transparent mode, all HTTPS proxy-style requests intended for another HTTPS proxy server are accepted. The Content Engine acts on these transparently received requests as defined by the **proxy-protocols transparent** global configuration command. For more information on this topic, see the [“Configuring HTTP and HTTPS Outgoing Proxy Exclusion Settings” section on page 14-5](#).

## Preventing Unwanted Access to Any Destination Port

A Content Engine may not provide the desired level of security if rules controlling access (allow or deny) to destination ports are not configured on the Content Engine.



### Caution

The Content Engine may not provide the desired level of security if the policies controlling access to the destination ports have not been configured. Consequently, we strongly recommend that you set restrictions that allow or deny HTTPS traffic to destination ports. Default settings may not provide the desired level of security.

In order to prevent web users from creating HTTPS tunnels to non-HTTPS servers, access to destination ports can be restricted. The restrictions can be made for individual ports or for all ports. In the case of a conflict in the restriction rules, the individual port setting takes precedence over the “all” rule, and the “allow” rule takes precedence over the “deny” rule.

Preventing access to destination ports is supported with WCCP transparent redirection (that is, a WCCP Version 2-enabled router transparently intercepts and redirects HTTPS request to the Content Engine) as well as with direct proxy routing, in which client browsers send their HTTPS requests directly to the Content Engine.

You can use the Content Engine GUI or CLI to configure this feature. To use the Content Engine GUI, choose **Cache > HTTPS Proxy**, and use the displayed HTTPS Proxy window to restrict destination ports. To obtain more information about how to use the HTTPS Proxy window, click the **HELP** button in the window.

To use the Content Engine CLI, enter this command:

```
ContentEngine(config)# https destination-port ?  
    allow  Allow HTTPS traffic to port (443 and 563 allowed by default)  
    deny   Deny HTTPS traffic to port (port under 1024 denied by default)
```

To prevent unwanted access to any destination HTTPS port when a request is going through the Content Engine, use the following command sequence:

```
ContentEngine(config)# no https destination-port allow 443 563  
ContentEngine(config)# https destination-port deny all
```

This command sequence denies access to any port above and below 1024. Ports 443 and 563 (the standard HTTPS ports) must be explicitly denied access using the **no https destination-port allow 443 563** global configuration command.



### Note

TCP and UDP packets use port numbers defined by the application in use. Typically, the port range 0–255 is used for standard public applications such as FTP, and the port range 256–1023 is used by companies for nonstandard applications. For instance, FTP uses port 21, and Telnet uses port 23. Port numbers from 1024 through 65,536 are unregulated, so it is best to specifically deny access through any port number.

For example, when these commands are configured on the Content Engine and the request to access port xxx at <https://banking.bankabc.com> is redirected to this Content Engine through WCCP transparent redirection or direct proxy routing, the connection to port xxx is denied.

# Configuring FTP Caching for Standalone Content Engines

This section provides an overview of the FTP caching feature for standalone Content Engines and describes how to configure the different types of FTP caching:

- [Overview of FTP Caching with Standalone Content Engines, page 7-36](#)
- [Configuring Nontransparent FTP-over-HTTP Caching on Standalone Content Engines, page 7-38](#)
- [Configuring FTP Native Caching for Standalone Content Engines, page 7-41](#)

**Note**

You can use the Content Engine GUI or the CLI to configure FTP caching on a standalone Content Engine.

## Overview of FTP Caching with Standalone Content Engines

A standalone Content Engine can be configured for FTP caching in the following two usage modes:

- **FTP-over-HTTP mode**—The standalone Content Engine (acting as a nontransparent forward proxy server) caches the contents of the specified FTP URLs that are sent to it directly by clients who are using the HTTP protocol. This allows users to use their browsers running the HTTP protocol to send and receive files on remote FTP servers. For more information, see the [“Configuring Nontransparent FTP-over-HTTP Caching on Standalone Content Engines” section on page 7-38](#).
- **Native FTP mode**—The standalone Content Engine caches the contents of the FTP request that are sent from clients in the native FTP protocol. In the ACNS 5.3.1 software and later releases, native FTP caching is supported in transparent and nontransparent proxy mode. (Native FTP caching was only supported in transparent proxy mode in the ACNS 5.1 and 5.2 software releases.) For more information, see the [“Configuring FTP Native Caching for Standalone Content Engines” section on page 7-41](#).

In both of these usage modes, the Content Engine uses the FTP protocol to retrieve and locally cache the content of the FTP requests. These two usage modes differ in the protocol used by the client to issue the FTP request. In FTP-over-HTTP mode, clients use their browsers (the HTTP protocol) to issue FTP requests. In native FTP mode, clients use the native FTP protocol to issue FTP requests, as shown in the following example:

```
ContentEngine# ftp server.cisco.com
```

Table 7-12 summarizes the usage modes and the types of supported FTP caching.

**Table 7-12 Usage Modes and Types of Supported FTP Caching**

Usage Mode	Description	Transparent Redirection	Direct Proxy Routing
<b>FTP-over-HTTP</b>	Content Engine caches the contents of the specified FTP URLs that are sent to it directly by clients who are using the HTTP protocol.	Not supported	Client browsers send their nontransparent FTP-over-HTTP requests directly to the Content Engine (a nontransparent forward proxy server).  This type of caching is called nontransparent FTP-over-HTTP caching, and is supported in the ACNS 5.0 software and later releases. For more information, see the <a href="#">“Configuring Nontransparent FTP-over-HTTP Caching on Standalone Content Engines”</a> section on page 7-38.
<b>Native FTP</b>	Content Engine caches the contents of the FTP requests that are sent from FTP clients in the native FTP protocol	WCCP Version 2-enabled routers transparently intercept FTP native requests from FTP clients, and redirect these requests to Content Engines (transparent proxy servers).  This type of caching is called transparent FTP native caching, and is supported in the ACNS 5.1 software and later releases. For more information, see the <a href="#">“Configuring Transparent FTP Native Caching”</a> section on page 7-54.	FTP clients send their nontransparent FTP native requests directly to the Content Engine (a nontransparent proxy server).  This type of caching is called nontransparent FTP native caching, and is supported in the ACNS 5.3.1 software and later releases. For more information, see the <a href="#">“Configuring Nontransparent FTP Native Caching”</a> section on page 7-42.



**Note**

Transparent redirection of FTP requests is supported only by WCCP Version 2; transparent redirection through a Layer 4 switch is not supported.

In the ACNS 5.4.1 software release, the **authentication** option was added to the **ftp-native proxy** configuration command to support proxy authentication for nontransparent FTP native requests. For more information about this topic, see the “[Configuring Request Authentication for Nontransparent FTP Native Requests](#)” section on page 10-55.

In the ACNS 5.4.1 software release, the **access-list** option was added to the **ftp-native** configuration command to support IP ACLs to grant or deny access to the native FTP proxy service that is running on the Content Engine:

```
ContentEngine(config)# ftp-native ?
    access-list  Configure access-lists for ftp-native proxy-mode and
                  transparently redirected connections
    object       Configuration of FTP object
    proxy        Configuration for proxy-mode requests
```

For more information about this topic, see the “[Using IP ACLs to Control Native FTP Access](#)” section on page 19-19.

In the ACNS 5.3.1 software release, the **ftp** keyword was replaced with the **ftp-over-http** and **ftp-native** keywords to clearly differentiate between FTP native caching and FTP-over-HTTP caching:

```
ContentEngine(config)# ftp-over-http ?
age-multiplier  FTP-over-HTTP caching heuristic modifiers
  max-ttl        Maximum time to live for objects in the cache
  min-ttl        Minimum time to live for objects in the cache (in minutes,
                  default is 30)
  object         Configuration of FTP object
  proxy          Configuration for incoming proxy-mode requests
  reval-each-request Configuration of revalidation for every request
```

In the ACNS 5.3.1 software release, the **show ftp proxy EXEC** command was replaced with the **show ftp-over-http** and **show ftp-native EXEC** commands. In the ACNS 5.3.1 software release, the **show statistics ftp EXEC** command was replaced with the **show statistics ftp-over-http** and **show statistics ftp-native EXEC** commands. In the ACNS 5.3.1 software release, the **clear statistics ftp EXEC** command was replaced with the **clear statistics ftp-over-http** and **clear statistics ftp-native EXEC** commands.

## Configuring Nontransparent FTP-over-HTTP Caching on Standalone Content Engines

In the ACNS 5.0 software release, support for the proxying and caching of FTP-style requests over HTTP in proxy mode was added. When the Content Engine is configured in proxy mode, it can handle FTP-style requests over HTTP transport. When the Content Engine receives an FTP request from a client, it processes the request by searching its cache. If the object is not in its cache, it retrieves the object from an upstream FTP proxy server if this proxy server has been configured, or it retrieves the object directly from the origin FTP server.

With nontransparent FTP-over-HTTP caching, the standalone Content Engine is functioning as a nontransparent forward proxy server for FTP-over-HTTP requests from client browsers. The ACNS 5.1 software and later releases support proxying and caching of FTP URL client requests using proxy-mode HTTP requests when URLs specify the FTP protocol (for example, `ftp://ftp.mycompany.com/ftplib/ftp_file`).

The following example of an FTP-over-HTTP request shows how the end user can use a browser to access public files from an FTP server:

```
ftp://ftp.funet.fi/pub/cbm/crossplatform/converters/unix/
```

For these requests, the client uses HTTP as the transport protocol with the Content Engine, and the Content Engine uses FTP with the FTP server. When the Content Engine receives an FTP request from the web client, it first looks in its cache. If the object is not in its cache, it retrieves the object from an upstream FTP proxy server (if one is configured in the ACNS 5.2.1 software and later releases) or directly from the origin FTP server.

The FTP proxy supports anonymous as well as authenticated FTP requests. Only base64 encoding is supported for authentication. The FTP proxy accepts all FTP URL schemes defined in RFC 1738. In the case of a URL in the form `ftp://user@site/dir/file`, the proxy sends back an authentication failure reply and the browser supplies a popup window for the user to enter login information.

The FTP proxy supports commonly used MIME types, attaches the corresponding header to the client, chooses the appropriate transfer type (binary or ASCII), and enables the browser to open the FTP file with the configured application. For unknown file types, the proxy uses binary transfer as the default and instructs the browser to save the download file instead of opening it. The FTP proxy returns a formatted directory listing to the client if the FTP server replies with a known format directory listing. The formatted directory listing has full information about the file or directory and provides the ability for users to choose the download transfer type.

To use the Content Engine CLI to configure nontransparent FTP-over-HTTP caching for standalone Content Engines, follow these steps:

---

**Step 1** Configure the port numbers for the incoming proxy-mode FTP-over-HTTP requests.

```
ContentEngine(config)# ftp-over-http proxy incoming ports
```

*ports* are the port numbers on which the standalone Content Engine will accept incoming FTP-over-HTTP requests from client browsers in addition to port 80. Valid port numbers are 1 to 65535. You can specify up to eight ports.

The following example configures the Content Engine to accept FTP-over-HTTP requests from client browsers on ports 8080, 8081, and 9090. Up to eight incoming proxy ports can be configured on the same command line:

```
ContentEngine(config)# ftp-over-http proxy incoming 8080 8081 9090
```

If you enter an illegal port number, a message appears informing you of this situation. For example, if you attempt to specify port 9002 as the incoming port for FTP-over-HTTP requests, you are informed that this port is reserved for the RealProxy application, as follows:

```
ContentEngine(config)# ftp-over-http proxy incoming 9002  
Port 9002 is reserved for application the Real_Proxy.
```

Remember that if you use the **ftp-over-http proxy incoming** command to configure the Content Engine to accept FTP-over-HTTP requests on a port other than port 80, you must also configure the client browsers to send their FTP-over-HTTP requests to that port. For more information, see the [“Pointing Client Browsers Directly to a Standalone Content Engine”](#) section on page 4-36.

**Step 2** Configure transaction logging on the Content Engine.

The following example shows how to configure the Content Engine to use the Apache transaction log format then enable transaction logging on the Content Engine:

```
ContentEngine(config)# transaction-logs format apache  
ContentEngine(config)# transaction-logs enable
```

For more information on this topic, see the [“Enabling Transaction Logging”](#) section on page 21-33.

- Step 3** Configure FTP cache object freshness settings for FTP-over-HTTP caching. These parameters can be configured for either directory listings or particular objects in the cache.

**Tip**

The ACNS 5.x software can be tuned to balance HTTP and FTP object freshness with the cache hit rate. The ACNS software default parameters are weighted in favor of securing fresh content over maximizing the cache hit rate (to avoid the undesirable situation of increasing the cache hit rate by serving stale content). Text objects refer to HTML pages. Binary objects refer to all other web objects, such as GIFs and JPEGs.

- a. Specify the maximum size of an FTP object that should be stored in the Content Engine cache for FTP-over-HTTP caching. In following example, the maximum size for an FTP object size is set to 2 megabytes for FTP-over-HTTP caching.

```
ContentEngine(config)# ftp-over-http object max-size 2000
```

- b. For FTP-over-HTTP caching, you can also use the **ftp-over-http age-multiplier**, **ftp-over-http max-ttl**, **ftp-over-http reval-each-request**, and the **ftp-over-http min-ttl** global configuration commands.

The following example configures a maximum Time To Live (TTL) of 3 days in the cache for directory listing objects and file objects for FTP-over-HTTP caching:

```
ContentEngine(config)# ftp-over-http max-ttl days directory-listing 3 file 3
```

The following example configures the Content Engine to keep FTP objects in the cache for a minimum of 10 minutes and a maximum of 24 hours (1 day) for FTP-over-HTTP caching:

```
ContentEngine(config)# ftp-over-http min-ttl 10
ContentEngine(config)# ftp-over-http max-ttl hours directory-listing 24 file 24
```

- c. Force the Content Engine to revalidate all objects for every FTP-over-HTTP request.

```
ContentEngine(config)# ftp-over-http reval-each-request all
```

**Note**

In the ACNS 5.3.1 software release, the **ftp** keyword was replaced with the **ftp-over-http** and **ftp-native** keywords.

- Step 4** Configure one or more outgoing FTP proxy servers for the Content Engine with the **ftp-over-http proxy outgoing host** global configuration command. Enter the hostname or IP address for the outgoing FTP proxy servers. The primary outgoing FTP proxy server is the parent cache (upstream FTP proxy server) to which you want this Content Engine to direct all of its missed FTP traffic without using ICP or WCCP. For more information, see the [“Designating a Primary Outgoing FTP Proxy Server”](#) section on page 14-3.

- Step 5** Specify the password that has to be used during anonymous FTP-over-HTTP operation with the **ftp-over-http proxy anonymous-pswd** global configuration command.

- Step 6** Enable active mode on this Content Engine for FTP-over-HTTP mode.

```
ContentEngine(config)# ftp-over-http proxy active-mode enable
```

In FTP-over-HTTP caching mode, if the **ftp-over-http proxy active-mode** global configuration command is used the Content Engine first attempts to use active mode with the origin FTP server for the data connection. If the active mode fails, the Content Engine attempts to use passive mode for the data connection.

In FTP-over-HTTP mode, if the **ftp-over-http proxy active-mode** command is not used the Content Engine first attempts to use passive mode with the FTP server for the data connection, and automatically switches to active mode if passive mode is not supported by the FTP server.

**Step 7** View the current FTP-over-HTTP configuration on the standalone Content Engine.

```
ContentEngine(config)# exit
ContentEngine# show ftp-over-http
```

**Step 8** View the statistics for the FTP-over-HTTP requests that this standalone Content Engine has handled.

```
ContentEngine# show statistics ftp-over-http
```

For example, the command output shows the number of FTP-over-HTTP requests received by the Content Engine, the number of FTP-over-HTTP hits and misses, as well as the number of FTP-over-HTTP requests that the Content Engine has forwarded to the origin FTP server or to the specified outgoing proxy server. The command output also shows the number of FTP-over-HTTP errors.

**Step 9** (Optional). Clear the FTP-over-HTTP statistics on the Content Engine.

```
ContentEngine# clear statistics ftp-over-http
```

---

## Configuring FTP Native Caching for Standalone Content Engines

This section describes how to configure FTP native caching for standalone Content Engines:

- [Configuring Nontransparent FTP Native Caching, page 7-42](#)
- [Configuring Transparent FTP Native Caching, page 7-54](#)

For the ACNS 5.1 and 5.2 software releases, all of the following restrictions apply to FTP native caching support:

1. Maximum FTP object size of 200 megabytes
2. No support for bandwidth control for FTP client requests and FTP server pulls
3. No support for the Type of Service (ToS) bit for FTP client requests
4. No support for pre-positioned files in the cdnfs
5. No support for ICAP
6. No support for nontransparent proxy
7. No support for proxy authentication
8. No support for ICP
9. No support for healing mode
10. No support for Layer 4 switch FTP redirection
11. No support for FTP request proxy rules
12. No support for MIN-TTL and AGING-HEURISTIC-TTL cache control knob configurations
13. No support for any URL filtering schemes (good list, bad list, N2H2, Websense, and SmartFilter)
14. No support for caching files from a Macintosh FTP server
15. No support for offline operation of the FTP proxy server

For the ACNS 5.3.x software releases, all of the preceding 15 restrictions still apply except for restriction 6 (no support for nontransparent proxy). In the ACNS 5.3.1 software release, support of the nontransparent proxy was added for FTP native mode. For more information about this topic, see the [“Configuring Nontransparent FTP Native Caching” section on page 7-42](#).

For the ACNS 5.4.x software releases, all of the preceding 15 restrictions still apply except for the following two restrictions:

- Restriction 6 (no support for nontransparent proxy)
- Restriction 7 (no support for proxy authentication for nontransparent native FTP requests)

In the ACNS 5.4.1 software release, support for proxy authentication (request authentication at the Content Engine) was added for nontransparent FTP native requests. In the ACNS 5.4.1 software release, the **authentication** option was added to the **ftp-native proxy** configuration command to support proxy authentication for nontransparent FTP native requests. For authentication of nontransparent FTP native requests, the ACNS 5.4.1 software and later releases support RADIUS, TACACS+, NTLM, and LDAP.

By default, the ftp-native proxy authentication feature is disabled. Because the FTP protocol is inherently insecure, the authentication credentials can be sniffed off the network and expose user credentials that otherwise would have been provided over a secure channel (for example, in the case of HTTP). You can enable the ftp-native proxy authentication feature as follows:

```
ContentEngine(config)# ftp-native proxy authentication enable
```

## About IP ACL Support for FTP Native Requests

In the ACNS 5.4.1 software and later releases, you can use IP ACLs to grant or deny access to the native FTP proxy service that is running on a standalone Content Engine. For more information about this topic, see the [“Using IP ACLs to Control Native FTP Access” section on page 19-19](#).

## Configuring Nontransparent FTP Native Caching

In the ACNS 5.3.1 software and later releases, the ability to proxy and cache nontransparent FTP native requests is supported. With this capability, a Content Engine has the ability to handle client FTP native requests from FTP clients in proxy mode.

When the Content Engine receives an FTP native request from an FTP client (for example, an FTP native request from a Reflection X or WS-FTP client or a UNIX or DOS command line FTP program), the Content Engine processes the request. If the requested content is already stored in the local cache, the Content Engine serves the content to the FTP client. Otherwise, the Content Engine performs an FTP request to the origin FTP server to retrieve the requested content, and then stores the content in its local cache. This type of caching is called *nontransparent FTP native caching*. Native FTP requests are logged in the HTTP transaction log on the standalone Content Engine.

Both passive and active mode for retrieving files and directories is supported. In FTP native caching mode, if the **ftp-native proxy active-mode enable** global configuration command is used, the Content Engine uses the same mode with the FTP server for the data connection as the client used to access the Content Engine, which can be either active or passive. If the **ftp-native proxy active-mode enable** command is not specified, the Content Engine uses passive mode with the FTP server for the data connection.



### Note

Passive mode transfer between the FTP client and the FTP native proxy cannot be done if the Content Engine is behind a NAT because the Content Engine will not know its conduit IP address.

In the ACNS 5.4.1 software and later releases, proxy authentication (that is, authentication at the Content Engine) for nontransparent FTP native requests is also supported. The username that is provided by the FTP client (for example, the Reflection X client) during the proxy authentication is logged in the transaction log if either the extended-squid or custom logging formats have been configured on the Content Engine.

To configure nontransparent FTP native caching on standalone Content Engines, follow these steps:

**Step 1** Configure the port numbers for the incoming proxy-mode FTP requests (FTP native requests).

```
ContentEngine(config)# ftp-native proxy incoming ports
```

*ports* are the port numbers on which the standalone Content Engine will accept incoming requests (native FTP requests) from FTP clients. Valid port numbers are 1 to 65535. You can specify up to eight incoming ports.

The following example configures the Content Engine to accept FTP native requests from FTP clients on eight ports (port 8501, 8502, 8503, 8504, 8505, 8506, 8507, and 8508). Up to eight incoming proxy ports can be configured on the same command line, as shown in the following example:

```
ContentEngine(config)# ftp-native proxy incoming 8501 8502 8503 8504 8505 8506 8507 8508
```

If you reenter the **ftp-native proxy incoming** command, the Content Engine is reconfigured and only uses the ports specified in the most recently specified **ftp-native proxy incoming** command. For example, if you enter the following **ftp-native proxy incoming** command, the Content Engine uses the eight specified ports as incoming proxy ports:

```
ContentEngine(config)# ftp-native proxy incoming 8501 8502 8503 8504 8505 8506 8507 8508
```

However, if you reenter the following **ftp-native proxy incoming** command, the Content Engine is reconfigured to use only port 8501 as an incoming proxy port and drops the other seven previously configured ports as incoming proxy ports:

```
ContentEngine(config)# ftp-native proxy incoming 8501
```

If you enter an illegal port number, a message appears informing you about this situation. For example, if you attempt to specify port 554 as the incoming port for proxy-mode FTP native requests, you are informed that this port is reserved for the RTSP gateway that runs on the Content Engine, as follows:

```
ContentEngine(config)# ftp-native proxy incoming 554
Port 554 is reserved for application the RTSP_Gateway.
```

**Step 2** Specify the maximum size of an FTP object that should be stored in the Content Engine cache for FTP native caching. This parameter can be configured for either directory listings or particular objects in the cache.

```
ContentEngine(config)# ftp-native object max-size size
```

In the following example, the maximum size for an FTP object size is set to 2 MB for FTP native caching:

```
ContentEngine(config)# ftp-native object max-size 2000
```

**Step 3** (Optional). Configure the Content Engine to use IP ACLs to permit or deny access to the native FTP proxy service that is running on the Content Engine:

```
ContentEngine(config)# ftp-native access-list in {std-acl-num | std-acl-name}
ContentEngine(config)# ftp-native access-list out {ext-acl-num | ext-acl-name}
```

For example, the following commands define a standard access list that grants the FTP clients on the 192.168.1.0 subnetwork access to the native FTP proxy service that is running on the Content Engine:

```
ContentEngine(config)# ip access-list standard 3
ContentEngine(config-std-nacl)# permit 192.168.1.0 0.0.0.255
ContentEngine(config-std-nacl)# exit
ContentEngine(config)#
```

Associate this standard IP ACL (access list 3) with the native FTP proxy service, and activate this standard IP ACL on the Content Engine.

```
ContentEngine(config)# ftp-native access-list in 3
```

The Content Engine applies the specified IP ACL (for example, ACL\_3) to native FTP inbound traffic.



#### Note

In the ACNS 5.4.1 software and later releases, you can use IP ACLs to grant or deny access to the native FTP proxy service that is running on a standalone Content Engine. For more information, see the [“Using IP ACLs to Control Native FTP Access”](#) section on page 19-19.

#### Step 4

(Optional). Create and then download a custom acl-denied message to the Content Engine. The Content Engine will display this custom message to the FTP client (for example, Reflection X clients, WS-FTP clients, or clients of UNIX or DOS command line FTP programs) when it denies an incoming connection based on the IP ACLs that are defined for the native FTP proxy service.

- a. Create the acl-denied message and save it as a text file (for example, save it as a file named warning.txt to the errors directory on the server named myserver.com). The file size of the custom acl-denied message should not exceed 16 KB.

The following example uploads the warning.txt file to the errors directory on the server named myserver.com:

```
ContentEngine# ftp-native custom-message upload acl-denied
http://www.myserver.com/errors/ftp-native-acl-denied.txt
```

- b. Download the ftp-native-acl-denied.txt file to the Content Engine:

```
ContentEngine# ftp-native custom-message download acl-denied
http://www.myserver.com/errors/ftp-native-acl-denied.txt
```

- c. Display the contents of the custom acl-denied message that has been downloaded to the Content Engine:

```
ContentEngine# show ftp-native custom-message acl-denied ftp-native-acl-denied.txt
```

#### Step 5

(Optional). Configure the Content Engine to display a custom welcome message to welcome proxy mode connections from FTP clients.

- a. Use a text editor to create a custom welcome message and save it as a text file (for example, welcome.com).
- b. Use the HTTP, HTTPS, or FTP protocol to upload the welcome.txt file to a server. The file size of the custom welcome message should not exceed 16 KB.

The following example uploads the welcome.txt file to the mgmt directory on the server named myserver.com:

```
ContentEngine# ftp-native custom-message upload welcome
http://www.myserver.com/mgmt/welcome.txt
```

- c. Download the welcome.txt file to the Content Engine:

```
ContentEngine# ftp-native custom-message download welcome
http://www.myserver.com/errors/welcome.txt
```

- d. Verify that the downloaded file exists:

```
ContentEngine# show ftp-native custom-message
welcome
acl-denied
ContentEngine#
```

- e. Display the contents of the downloaded custom welcome message:

```
ContentEngine# show ftp-native custom-message welcome
Welcome to the Content Engine that is acting as your FTP proxy.
Login to the proxy using the proxy username and password.
```



**Note**

For more information, see the [“Creating Custom Messages for FTP Proxy Responses for FTP Native Requests”](#) section on page 5-19.

- Step 6** Use an FTP client (for example, a UNIX command line FTP program) to send a request to the Content Engine to verify that the custom welcome message is displayed when the Content Engine accepts an incoming FTP connection from an FTP client (for example, a Reflection X client, a WS-FTP client, or a UNIX or DOS command line FTP program).

In the following example, the FTP client sends the Content Engine (IP address 172.31.255.255) a native FTP request on port 8501. After accepting the incoming FTP connection, the Content Engine displays the custom welcome message to the FTP client:

```
shell# ftp -d 172.31.255.255 8501
Connected to 172.31.255.255
220 Welcome to the Content Engine that is acting as your FTP proxy. Login to the proxy
using username and password.
```

- Step 7** Configure and enable transaction logging on the Content Engine.

In the ACNS 5.4.1 software and later releases, the username that is provided by the FTP client during the proxy authentication is logged in the transaction log if either the Extended Squid or Custom logging formats have been configured on the Content Engine:

The following example configures the Content Engine to use the Extended-Squid transaction log format, and then transaction logging is enabled on the Content Engine:

```
ContentEngine(config)# transaction-logs format extended-squid
ContentEngine(config)# transaction-logs enable
```

For the Custom transaction logging format, you must include the `%u` format-specifier when you configure the `transaction-logs format custom` command. For more information, see the [“Enabling Transaction Logging”](#) section on page 21-33.

- Step 8** Enable FTP native active mode on the Content Engine.

```
ContentEngine(config)# ftp-native proxy active-mode enable
```

**Step 9** (Optional). Enable the FTP proxy authentication feature on the Content Engine.

By default, this feature is disabled. Because the FTP protocol is inherently insecure, the authentication credentials can be sniffed off the network and expose user credentials that otherwise would have been provided over a secure channel (for example, in the case of HTTP).

```
ContentEngine(config)# ftp-native proxy authentication enable
```

If you enter the **ftp-native proxy authentication enable** command and you have not already configured an authentication service (for example, RADIUS, LDAP, NTLM, or TACACS) on the Content Engine, a message is displayed. The message indicates that you must configure an authentication service on the Content Engine before you can enable the FTP proxy authentication feature.

**Step 10** Configure the FTP clients (client side) to send their native FTP requests directly to the Content Engine. For more information, see the next section, [“Configuring the Client Side of Nontransparent FTP Native Caching.”](#)

**Step 11** View the current FTP native proxy configuration.

```
ContentEngine# show ftp-native
```

**Step 12** Display the statistics for the FTP native requests that this standalone Content Engine has handled.

```
ContentEngine# show statistics ftp-native
```

The command output shows the number of FTP native GET requests received by the Content Engine, the number of FTP native hits and misses for GET requests, as well as the number of FTP native PUT requests that have been received by this Content Engine. In the ACNS 5.4.1 software and later releases, the command output also shows the number of transparent FTP native requests, the number of nontransparent FTP native requests, the number of native FTP proxy authentication requests, and the number of failed native FTP proxy authentication requests.

**Step 13** Clear the FTP native statistics on the Content Engine.

```
ContentEngine# clear statistics ftp-native
```

## Configuring the Client Side of Nontransparent FTP Native Caching

Content Engines that are acting as nontransparent FTP proxy servers can accept FTP native requests from such FTP clients as Reflection X clients, WS-FTP clients, and UNIX or DOS command line FTP programs. This section provides some examples of how to configure these different types of FTP clients to send FTP native requests directly to the Content Engine, which is acting as a nontransparent FTP proxy server for these FTP clients.



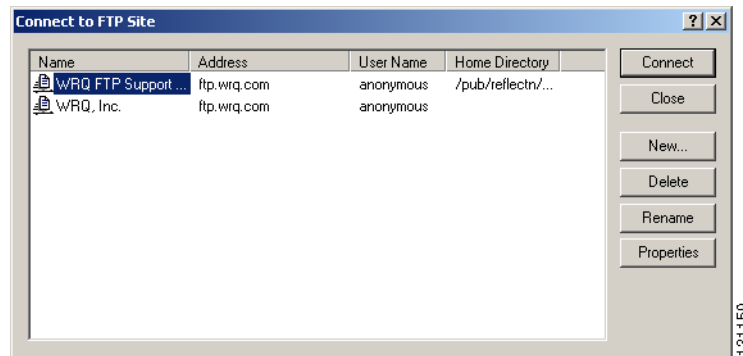
### Note

In the ACNS 5.4.1 software release, support for proxy authentication (that is, authentication at the Content Engine) was added for nontransparent FTP native requests. The username that is provided by the FTP client (for example, Reflection X clients, WS-FTP clients, or clients of UNIX or DOS command line FTP programs) during the proxy authentication process is logged in the transaction log if one of the following transaction logging formats have been configured on the Content Engine: Extended-Squid logging or Custom logging. For more information about the FTP proxy authentication feature, see the [“Configuring Request Authentication for Nontransparent FTP Native Requests”](#) section on page 10-55.

The following example shows how to configure the client side proxy FTP request to the Content Engine through the Windows-based Reflection X client software:

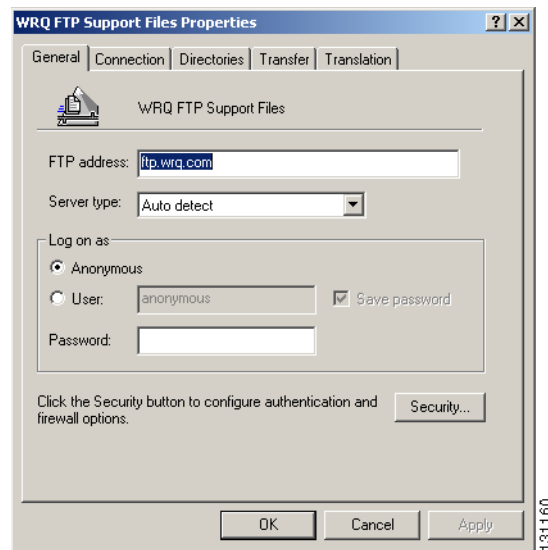
- Step 1** From the Windows Start menu, choose **Programs > Reflection > FTP Client**. The Connect to FTP Site window appears. (See [Figure 7-2](#).)

**Figure 7-2** Using Reflection X Client Software to Connect to an FTP Site

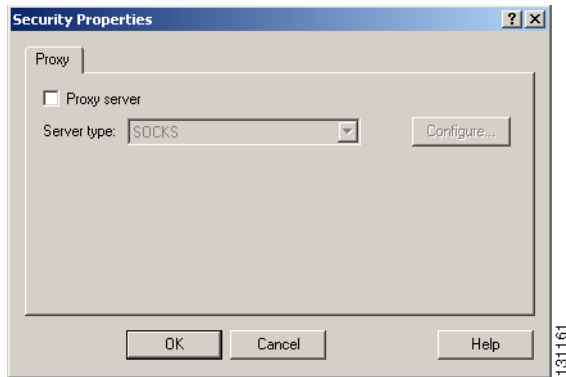


- Step 2** Select one of the FTP sites that are listed in the Connect to FTP Site window.
- Step 3** Click the **Properties** button. The Properties window for the selected FTP site appears. (See [Figure 7-3](#).)

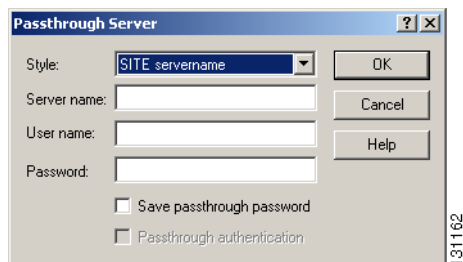
**Figure 7-3** Properties Window



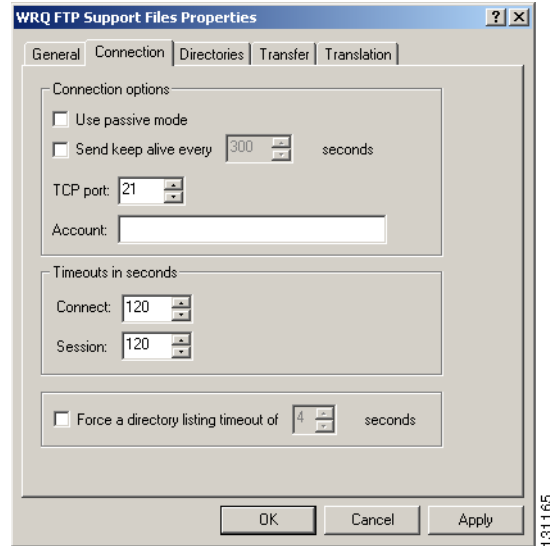
- Step 4** In the General tab, click the **Security** button. The Securities dialog box appears. (See [Figure 7-4](#).)

**Figure 7-4** *Securities Dialog Box*

- Step 5** In the Securities dialog box, complete the following steps:
- Select the Proxy-server check-box (in the Proxy tab).
  - Select the **Passthrough** entry from the Server type drop-down menu.
- Step 6** Click the **Configure** button. The Passthrough Server dialog box appears. (See [Figure 7-5](#).)

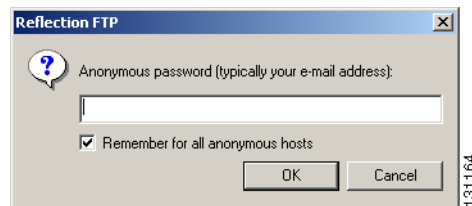
**Figure 7-5** *Passthrough Server Dialog Box*

- Step 7** In the Passthrough Server dialog box, complete the following steps:
- Select the **username@servername** entry in the Style drop-down menu.
  - Enter the Content Engine's hostname or IP address in the **Server name** edit-box.
  - Ensure that the **Passthrough authentication** check box is not checked.
  - Click **OK** to close the Passthrough Server dialog box, and to return to the Securities dialog box.
- Step 8** In the Securities dialog box, click **OK** to return to close the Securities dialog box.
- Step 9** In the Properties window ([Figure 7-3](#)), click the Connection tab.
- Step 10** In the Connection tab ([Figure 7-6](#)), set the TCP port to the Content Engine's FTP-native incoming proxy port.

**Figure 7-6** Connection Tab of the Properties Window

In the TCP port field, you must enter the same port number that you specified with the **ftp-native proxy incoming port** global configuration command (for example, port 7780) in [Step 1](#) of the “[Configuring Nontransparent FTP Native Caching](#)” section on page 7-42.

- Step 11** In the Connection tab, click **Apply** and then click **OK** to return to the Connect to FTP Site window. (See [Figure 7-2](#).)
- Step 12** Verify that the Reflection X client can connect to the FTP site.
- a. In the Connect to FTP Site window, click the **Connect** button. The Reflection FTP dialog box appears asking you to enter a password. (See [Figure 7-7](#).)

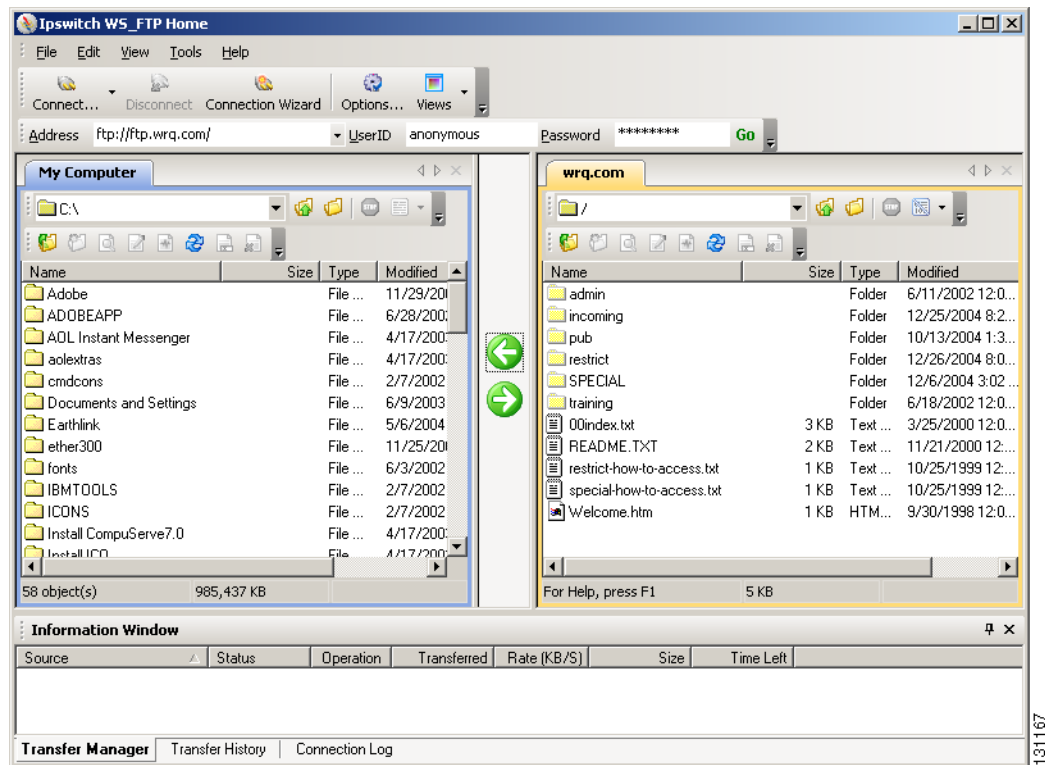
**Figure 7-7** Reflection FTP Password Dialog Box

- b. Enter the password and click **OK**.

WS-FTP is a Windows-based FTP client for transferring files from a Windows-based desktop to remote machines. This licensed software from Ipswitch, Inc. allows you to perform single file transfers from a Windows-based computer. The following example shows how to configure the client side proxy FTP request to the Content Engine through the Windows-based WS-FTP client software:

- Step 1** From the Windows Start menu, choose **Programs > Ipswitch WS\_FTP\_Home > WS\_FTP Home**. The WS-FTP Home window appears. (See [Figure 7-8](#).)

**Figure 7-8** Using WS-FTP Client Software to Connect to an FTP Site (an FTP Server)



- Step 2** From the File menu, select **Connect > Connection Wizard**.
- Step 3** Enter the site name (the name of the FTP server that you want to connect to).
- Step 4** Enter the IP address of the FTP server.
- Step 5** Enter the username and password of the FTP server.
- Step 6** Select FTP as the connection type.
- Step 7** Click the **Finish** button.
- Step 8** From the Tools menu, select **Options > Firewall**.
- Step 9** Click the **New** button.
- Step 10** Enter the name and IP address of the Content Engine (that is acting as this client's nontransparent proxy server).
- Step 11** Select **Script** (fwsc) as the type. This option specifies that you want to use a firewall script.
- Step 12** Enter the proxy port number.

You must enter the same port number that you specified with the **ftp-native proxy incoming port** global configuration command (for example, port 7780) in [Step 1](#) of the “[Configuring Nontransparent FTP Native Caching](#)” section on page 7-42.

- Step 13** Click **Okay**.
- Step 14** Click the Firescript editor to open the editor.
- Step 15** Create a firewall script.
- Step 16** In the Firescript editor, paste the content of the firewall script that you have created.

The following is an example of the content of a sample firewall script that you would paste into the Firescript editor:

```
[fwsc]
author=cisco
version=1
verdate=2004.11.23
required=HostUserId,HostPassword,HostAddress
connectto=firewall

[comment]
This script is similar to the "site hostname" script except that
the following line at the beginning of the script section is
removed:
    send ( " " ) {}

[script]

send ("SITE %HostAddress") {}

send ("USER %HostUserId")
{
    case (300..399) :
        continue;

    case any :
        return (false) ;
}

send ("PASS %HostPassword")
{
    case (200..299) and contains(lastreply, "ACCOUNT") and not
isempty(HostAccount) :
        continue;

    case (300..399) :
        continue;

    case (200..299) :
        jump success;

    case any :
        return (false) ;
}

send ("ACCT %HostAccount")
{
    case (200..299) :
        jump success;

    case any:
        return (false);
}
```

```
label success;
goss1;
return (true);
```

```
=====
```

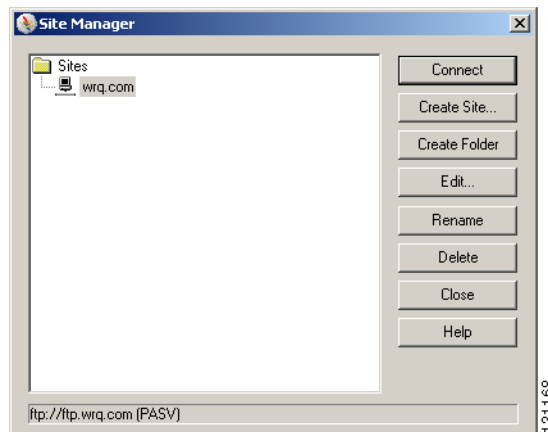
- Step 17** Save this firewall script as a file in the default location.
- Step 18** From the Tools menu, select **Options > Firewall**.
- Step 19** Select the Content Engine that you want to proxy FTP requests through. This is the Content Engine that you specified in [Step 10](#) of this procedure.
- Step 20** Click **Edit**.
- Step 21** Select the type field as script <fswc> (firescript name).



**Tip** The firewall script will actually proxy through the Content Engine with the site or username format internally.

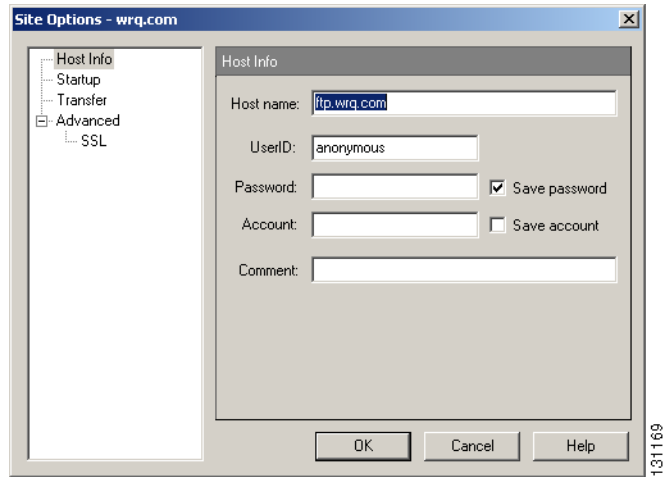
- Step 22** From the WS-FTP Home window, select **Tools > Site Manager**. The Site Manager window appears. (See [Figure 7-9](#).)

**Figure 7-9** Site Manager Window



- Step 23** In the Site Manager window, click on the site that is created already (for example, “wrq.com”), and click **Edit**. The Site Options window appears. (See [Figure 7-10](#).)

Figure 7-10 Site Options Window



**Step 24** In the Site Options window, click **Advanced**.

**Step 25** In the displayed Firewall drop-down list, select the Content Engine that has already been configured.

The following example shows how to use a UNIX command line FTP program to configure the client side proxy FTP request to the Content Engine that is acting as the nontransparent FTP proxy server:

```
shell# ftp -d 10.1.1.50 8501
Connected to 10.1.1.50
220 Welcome to FTP-proxy. Login to the proxy using username and password.
Name (10.1.1.50:admin): smartuser@abchost.company.com
---> USER smartuser
331 Password required for smartuser.
Password:
---> PASS XXXX

220-Welcome to FTP-proxy.
220-Login to origin server using the 'USER username@server-hostname' command, or
220 Login to origin server using the 'SITE server-hostname' followed by the 'USER
username' command.
ftp> site host.abchost.com
---> SITE host.abchost.com
220 via2.abchost.com FTP server (Version wu-2.6.0(1) Mon Feb 28 10:30:36 EST 2000) ready.
ftp> user anonymous
---> USER anonymous
331 Guest login ok, send your complete e-mail address as password.
Password:
---> PASS XXXX
230 Guest login ok, access restrictions apply.
ftp> quit
---> QUIT
shell#
```

## Configuring Transparent FTP Native Caching

The ftp-native service (service 60) is the WCCP caching service that permits WCCP Version 2 routers to redirect transparent FTP native requests transparently to a single port on the Content Engine. The Content Engine retrieves the requested FTP content, stores a copy locally (performs FTP native caching), and serves the requested content to the client.

A standalone Content Engine that is operating as an FTP proxy supports passive and active mode for retrieving files and directories. In FTP native caching mode, if the **ftp-native proxy active-mode enable** global configuration command is specified, the Content Engine uses the same mode with the FTP server for the data connection as the client used to access the Content Engine, which can be either active or passive:

```
ContentEngine(config)# ftp-native proxy active-mode ?
    enable Adhere to client's mode for native FTP
```

If the **ftp-native proxy active-mode enable** command is not specified, the Content Engine uses passive mode with the origin FTP server for the data connection.

If the Content Engine adheres to the client's mode (active or passive) for native FTP, the following occurs:

- The Content Engine (the FTP native proxy server) performs an active-mode data transfer to or from the origin FTP server if the FTP client issues an active-mode data transfer request.
- The Content Engine performs a passive-mode data transfer to or from the FTP server if the FTP client issues a passive-mode data transfer request.

The format of the URL that the Content Engine creates for a native FTP request depends on the FTP login name and the transfer mode (binary or ACSII file transfer mode):

- If the FTP login name is an actual username instead of “anonymous,” then the string “\*user\*:password\*@" is included in the URL before the host.
- If the mode used to transfer the file is binary mode, then the string “;type=i” is included at the end of the URL. The following is an example of the URL format that the Content Engine creates for a specific user when binary mode is being used:

```
ftp://*user*:password*@10.100.200.5/home/myhome/mybinfile.obj;type=i
```

The URL for an “anonymous” user login and an ACSII file transfer mode will not have any fields embedded in the URL, as shown in the following example:

```
ftp://10.100.200.5/home/myhome/mytextfile.txt
```

The following two examples demonstrate the use of native FTP with a Content Engine. In the first example, the user logs in with an actual username name (“huff”) and is able to retrieve the requested file (test.c) from the FTP server. In this case, the home directory for the user named “huff” is “/home/huff.”

```
ContentEngine# ftp server.cisco.com
Connected to server.cisco.com.
220 server.cisco.com FTP server (Version wu-2.6.0(1) Mon Feb 28 10:30:36 EST 2000) ready.
Name (server:huff): huff
331 Password required for myserver.
Password:
230 User huff logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> pwd
257 "/home/huff" is current directory.
ftp> get /tmp/test.c
200 PORT command successful.
150 Opening BINARY mode data connection for /tmp/test.c (645 bytes).
```

```

226 Transfer complete.
645 bytes received in 0.00077 seconds (8.2e+02 Kbytes/s)
ftp> quit
ContentEngine#

```

In the following example, the user logs in as an anonymous user and cannot retrieve the requested file (test.c) because the file is not located in the document root directory of the FTP server (“/”), which is the home directory for any anonymous user:

```

ContentEngine# ftp server.cisco.com
Connected to server.cisco.com.
220 server.cisco.com FTP server (Version wu-2.6.0(1) Mon Feb 28 10:30:36 EST 2000) ready.
Name (server:huff): anonymous
331 Guest login ok, send your complete e-mail address as password.
Password: test@cisco.com
230 Guest login ok, access restrictions apply.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> pwd
257 "/" is current directory.
ftp>
ftp> passive
Passive mode on
ftp> get
(remote-file) /tmp/test.c
(local-file) test.c
local: test.c remote: /tmp/test.c
227 Entering Passive Mode (172.31.255.255)
550 /tmp/test.c: No such file or directory.
ftp>
ContentEngine#

```

To configure transparent FTP native caching with a standalone Content Engine (transparent proxy server) and a WCCP Version 2-enabled router, follow these steps:

---

**Step 1** Enable FTP native active mode on the Content Engine.

```
ContentEngine(config)# ftp-native proxy active-mode enable
```

In FTP native caching mode, if this command is specified, then the Content Engine uses the same mode (active or passive) with the origin FTP server for the data connection as the client used to access the Content Engine. If this command is not specified, the Content Engine uses passive mode with the origin FTP server for the data connection.

**Step 2** Configure transaction logging on the Content Engine.

The following example shows how to configure the Content Engine to use the Apache transaction log format and then enable transaction logging on the Content Engine:

```
ContentEngine(config)# transaction-logs format apache
ContentEngine(config)# transaction-logs enable
```

For more information, see the [“Enabling Transaction Logging” section on page 21-33](#).

**Step 3** Specify the maximum size of an FTP object for FTP native caching. This parameter can be configured for either directory listings or particular objects in the cache:

```
ContentEngine(config)# ftp-native object max-size ?
<1 - 204800> Maximum size of a cacheable object in Kbytes (default is 204800)
```

**Step 4** On the Content Engine, configure transparent FTP native caching.

Transparent redirection of FTP requests is supported only by WCCP Version 2; transparent redirection through a Layer 4 switch is not supported.

- a. Define a list of routers that will be used to redirect FTP native requests to this Content Engine. The following example shows how to configure router list 1 to include a single WCCP Version 2 router, the router with the IP address of 10.77.157.41:

```
ContentEngine(config)# wccp router-list 1 10.77.157.41
```

- b. Specify the router list that the Content Engine should accept redirected FTP native requests from. The following example shows how to configure the Content Engine to accept redirected FTP native requests for routers that are part of router list 1:

```
ContentEngine(config)# wccp ftp-native router-list-num 1
```

- Step 5** (Optional). Configure the Content Engine to use IP ACLs to grant or deny access to the native FTP proxy service:

```
ContentEngine(config)# ftp-native access-list in {std-acl-num | std-acl-name}
ContentEngine(config)# ftp-native access-list out {ext-acl-num | ext-acl-name}
```

For more information, see the [“Using IP ACLs to Control Native FTP Access”](#) section on page 19-19.

- Step 6** On the WCCP router, configure the router to transparently intercept FTP native requests and redirect them to the Content Engine that is acting as the FTP proxy server.

- a. Enable service 60 on the router.

```
Router(config)# ip wccp 60
```

Service 60 is a predefined WCCP Version 2 caching service that permits WCCP Version 2 enabled routers to redirect FTP native requests transparently to a single port on the Content Engine. The Content Engine retrieves the requested FTP content, stores a copy locally, and serves the requested content to the requester.

- b. Specify the interface on which service 60 will run.

```
Router(config)# interface type number
```

- c. Configure the router to use the outbound interface for service 60.

```
Router(config-if)# ip wccp 60 redirect out
Router(config-if)# end
```

- Step 7** Verify that the FTP proxy is enabled on the Content Engine:

```
ContentEngine# show wccp modules
```

```
Modules registered with WCCP on this Content Engine
```

Module	Socket	Expire(sec)	Name	Supported Services
2	13	4	WMT Proxy	WMT
3	14	4	MMSU WMT Proxy	MMSU
6	15	4	WMT-RTSPU	RTSPU
1	16	4	RTSP Proxy	RTSP
0	17	3	HTTP Proxy	Web Cache Reverse Proxy Custom Web Cache HTTPS Cache WCCPv2 Service 90 WCCPv2 Service 91

```

WCCPv2 Service 92
WCCPv2 Service 93
WCCPv2 Service 94
WCCPv2 Service 95
WCCPv2 Service 96
WCCPv2 Service 97

5      22      3      FTP Proxy      FTP
ContentEngine#

```

- Step 8** Verify that transparent FTP native caching (shown as “FTP” in the following command output) is configured on the Content Engine.

```

ContentEngine# show wccp services
Services configured on this Content Engine
  Web Cache
  Reverse Proxy
  RTSP
  WMT
  MMSU
  DNS
  FTP
  RTSPU
  HTTPS Cache
ContentEngine#

```

- Step 9** (Optional). Specify a mask used in Content Engine assignments for incoming transparent FTP native requests with the **wccp ftp-native mask** global configuration command. For example, use the **dst-ip-mask** command option to set the mask to match the destination IP address of the redirected packet. The destination IP address mask is defined as a hexadecimal number (for example, 0xFC000000). The range is 0x00000000 to FC000000.

```

ContentEngine(config)# wccp ftp-native mask ?
  dst-ip-mask  Specify sub-mask used in packet destination-IP address
  src-ip-mask  Specify sub-mask used in packet source-IP address
ContentEngine(config)# wccp ftp-native mask

```

For more information about specifying a mask for a WCCP service, see the [“Configuring Layer 2 Forwarding with the Mask Load-Balancing Method”](#) section on page 6-36.

- Step 10** Enter the **l2-redirect** option to specify Layer 2 redirection as the packet-forwarding method (as opposed to GRE). Enter the **mask-assign** option to specify mask assignment as the load-balancing method (as opposed to the default hash assignment method) for the FTP native caching service.

```

ContentEngine(config)# wccp ftp-native router-list-num 1 l2-redirect mask-assign
WCCP configuration for FTP succeeded.
Please remember to config WCCP service 60 on the corresponding router.
ContentEngine(config)#

```

- Step 11** View the configuration for the masks for transparent FTP native requests.

```

ContentEngine# show wccp masks ftp-native

```

- Step 12** View the current FTP native proxy configuration.

```

ContentEngine# show ftp-native

```

**Step 13** Display the statistics for the FTP native requests that this standalone Content Engine has handled.

```
ContentEngine# show statistics ftp-native
```

The command output shows the number of FTP native GET requests received by the Content Engine, the number of FTP native hits and misses for GET requests, as well as the number of FTP native PUT requests that have been received by this Content Engine. In the ACNS 5.4.1 software and later releases, the command output also shows the number of transparent FTP native requests, the number of nontransparent FTP native requests, the number of FTP-native proxy authentication requests, and the number of failed FTP-native proxy authentication requests.

**Step 14** Clear the FTP native statistics on the Content Engine.

```
ContentEngine# clear statistics ftp-native
```

---

## Configuring the TFTP Server and Gateway for Standalone Content Engines

In the ACNS 5.1 software and later releases, the Trivial File Transfer Protocol (TFTP) gateway feature enables Content Engines to serve content files requested by networking devices that use native TFTP. Content Engines now perform TFTP-to-HTTP or TFTP-to-FTP translation, eliminating the need for the system administrator to configure and manage a dedicated TFTP server to serve TFTP requests. This feature allows the Content Engine to accept native TFTP requests from the client at the front end and serve the request using the HTTP or FTP protocol at the back end.

Content files include router software images, router configurations, set top box images, IP phone configuration files, and so forth. If the requested file is not available on the Content Engine, the Content Engine caches the file from the origin server. The Content Engine that is functioning as a caching engine, retrieves the file from the Internet on behalf of the device and forwards it to the device. Future requests by any devices for the same file are satisfied by forwarding the file from the Content Engine's local cache.



### Note

The Content Engine does not support transparently intercepted TFTP requests. Every TFTP server request addressed to the Content Engine must have the Content Engine IP address as its destination address.

---

After the TFTP server has been enabled on the Content Engine, and a client sends a TFTP request for a file, the following events occur:

1. The TFTP server on the Content Engine checks the access control list that is assigned to the TFTP application for authorization.
2. If the request is authorized, the TFTP server checks the directory specified in the TFTP request for the content file. If the request does not contain any directory path, the server searches the default local directory for the file.
3. If the requested file does not exist in a local directory, the TFTP server on the Content Engine creates an HTTP or FTP URL and sends it to the caching application.
4. The caching application searches for the requested file in the cache file system (cfs) and then in the pre-positioned content (cdnfs). If the file is found, it is sent to the TFTP server on the Content Engine.

5. If the requested file is not found, the caching application requests the file from the origin server specified by the URL, and then caches the content.
6. The cached file is then sent to the TFTP server on the Content Engine, which replies to the TFTP client. If the file is not found, a 404 “File not found” message is sent.

## Using the TFTP Service and Gateway on Standalone Content Engines

You can use the **tftp-server** global configuration command to configure the TFTP service and gateway on a standalone Content Engine to serve content in response to TFTP requests in two ways:

- Serve local content—Configure local directories and enable the TFTP server on the Content Engine, as described in the next section, “[Enabling the TFTP Server and Gateway](#).”
- Serve content from remote servers—Configure the TFTP gateway feature on the Content Engine, as described in the next section, “[Enabling the TFTP Server and Gateway](#).”

When both the TFTP server and gateway are enabled, the Content Engine responds to TFTP requests by searching for files in its default local directory if the full pathname is not specified in the request. Otherwise, it looks in the local directory that matches the directory in the pathname. If the file is not found, it uses the HTTP or FTP protocol to forward the request to the Content Engine that is functioning as a caching engine. If the file is found on a remote server, the Content Engine caches the file and sends the file to the client that issued the request. The Content Engine replies to subsequent requests for the file from its local cache. If the file is not found, the Content Engine replies to the request with a 404 “File not found” message.

By default, the TFTP service is disabled and access to the TFTP server is denied.

The default local directory assigned to the TFTP server is `/tftpboot`. However, this directory must be created using the **mkdir** command.

The TFTP timeout value is fixed at 5 seconds, and the number of retries is fixed at five retries. These values are nonconfigurable.

## Enabling the TFTP Server and Gateway

To serve requests for local content, follow these steps:

- 
- Step 1** Enable the TFTP service on the Content Engine.

```
ContentEngine(config)# inetd enable tftp
```

- Step 2** Configure the local TFTP directories on a standalone Content Engine, as follows:

- a. Create the local directories using the **mkdir** command.
- b. Identify the local directories to the TFTP server using the **tftp-server dir** command.

When you use the **tftp-server dir** command to identify one or more local directories, the first directory identified becomes the default directory. Enter the **tftp-server dir** command once for each directory that you want to identify.

The TFTP server searches for files without a fully qualified pathname in its default directory. The TFTP server only looks for files in the other local directories if the TFTP request explicitly identifies the directory.

If you do not configure any local directories, /tftpboot is automatically assigned as the default directory. However, you would still need to create the /tftpboot directory using the **mkdir** command before the TFTP server can serve requests.

To use the TFTP gateway to serve content from remote servers, you must enable the TFTP server and also identify the remote servers to which the TFTP gateway (the standalone Content Engine) will direct requests. To identify the servers to which the Content Engine will direct requests when it cannot find the requested files in its local directories, use the **tftp-server gw proto** command. For more information on this topic, see the next section, “[Configuring the TFTP Server and Gateway on Standalone Content Engines](#).”

To enable the TFTP server or gateway on a standalone Content Engine, follow these steps:

- 
- Step 1** Enable the TFTP service on the Content Engine.  
ContentEngine(config)# **inetd enable tftp**
  - Step 2** Define an access list that will permit access to the TFTP service by using the **ip access-list** global configuration command.
  - Step 3** Apply the access list to the TFTP service by using the **tftp-server access-list** command.
  - Step 4** Use the different options of the **tftp-server** command to configure the TFTP server, as described in the next section.
- 

**Note**

The Content Engine does not support transparent TFTP requests. It only accepts TFTP requests that explicitly contain the Content Engine hostname or IP address.

## Configuring the TFTP Server and Gateway on Standalone Content Engines

The TFTP server searches for files in the default local directory when it receives a request that does not identify the full directory path to the file. If you do not configure any local directories, the default directory is /tftpboot. Although this directory is automatically assigned as the default directory, you still need to create it (or any other directories you assign to the TFTP server) using the **mkdir** EXEC command.

When you use the **tftp-server dir** global configuration command to identify one or more local directories, the first directory identified becomes the default directory. Enter the **tftp-server dir** global configuration command once for each directory that you want to identify.

The TFTP server only looks for files in the other directories if the TFTP request explicitly identifies the directory.

To configure the TFTP server and gateway on a standalone Content Engine, use the **tftp-server** global configuration command and follow these steps:

- 
- Step 1** Identify one or more local directories that the Content Engine should search for requested files when the full pathname is not included in the TFTP request with the **tftp-server dir** global configuration command.

For example, the following commands configure two local directories from which the Content Engine will try to fulfill TFTP requests:

```
ContentEngine(config)# mkdir /local/mydir
ContentEngine(config)# mkdir /local/clients
ContentEngine(config)# tftp-server dir /local/mydir
ContentEngine(config)# tftp-server dir /local/clients
```

The first directory specified, /local/mydir, is considered the default directory.

**Step 2** Identify the IP access control list (ACL) that allows access to the TFTP server and gateway.

**tftp-server access-list** {acl-num | acl-name}

For example, configure the Content Engine to check access list 1 to determine if TFTP access should be allowed or denied:

```
ContentEngine(config)# tftp-server access-list 1
```

**Step 3** Enable the TFTP gateway feature and identify specific servers to which TFTP requests will be directed when the Content Engine cannot find the requested files in its local directories.

**tftp-server gw proto** {ftp | http} **server** {hostname | ip\_address}  
[name name passwd password] [path directory] **pri** priority



**Note** When you enter this command, you identify the protocol (FTP or HTTP), the hostname or IP address of the server, and the authentication information required to access each server. You can enter the **tftp-server gw proto** global configuration command twice to configure a primary and backup HTTP or FTP servers. Use the **priority** option to specify whether the server is primary (priority = 1) or backup (priority = 2).



**Note** Authenticated objects are never cached by the Content Engine for HTTP or FTP. If you want to cache these objects, leave the username and password fields in the **tftp-server gw** command blank. When used with FTP, this configuration is equivalent to allowing anonymous access.

Table 7-13 describes the parameters for the **tftp-server gw proto** global configuration command.

**Table 7-13 Parameters for the tftp-server gw proto Command**

Parameter	Description
<b>gw</b>	Configures TFTP gateway functionality for the Content Engine.
<b>proto</b>	Configures the protocol used to access the origin server to which the TFTP gateway will forward requests when the file cannot be found in a local directory.
<b>ftp</b>	Uses the FTP protocol to access the origin server.
<b>http</b>	Uses the HTTP protocol to access the origin server.
<b>server</b>	Configures the origin server.
<i>hostname</i>	Hostname of the origin server.
<i>ip-address</i>	IP address of the origin server.
<b>name</b>	(Optional) Sets the username for authentication to the origin server.
<i>name</i>	(Optional) Username for authentication to the origin server.
<b>passwd</b>	(Optional) Sets the password for authentication to the origin server.

**Table 7-13 Parameters for the `tftp-server gw proto` Command (continued)**

<i>password</i>	(Optional) Password for authentication to the origin server.
<b>path</b>	(Optional) Sets the pathname to search for files on the origin server.
<i>directory</i>	(Optional) Pathname to search for files on the origin server.
<b>pri</b>	Sets the priority of the origin server.
<i>priority</i>	Priority (1 or 2) of the origin server.

**Step 4** Enable the TFTP server on a standalone Content Engine:

```
ContentEngine(config)# inetd enable tftp
```

- a. Define a standard access list that permits access to the TFTP service for FTP clients on the 192.168.1.0 subnetwork:

```
ContentEngine(config)# ip access-list standard 1
ContentEngine(config-std-nacl)# ip access-list permit 192.168.1.0 0.0.0.255
ContentEngine(config-std-nacl)# exit
```

- b. Configure two local directories from which the Content Engine will try to fulfill TFTP requests:

```
ContentEngine(config)# tftp-server dir /local1/mydir
ContentEngine(config)# tftp-server dir /local1/clients
```

The first directory specified, /local1/mydir, is considered the default directory.

- c. Specify the IP access list that should be used to permit access to the TFTP service:

```
ContentEngine(config)# tftp-server access-list 1
```

- d. Enable the TFTP gateway feature on a standalone Content Engine, and identify the FTP server to which the Content Engine should forward requests when it cannot find the file in its local directories. Set the username and password for authentication to the origin server.

```
ContentEngine(config)# tftp-server gw proto ftp 192.168.100.1 pri 1 path
/myremotedir name myuser passwd mypassword
```

The directory name /myremotedir is used in the URL sent by the ACNS caching service on the Content Engine to retrieve the file from the remote server. The URL created by using this sample configuration would be as follows:

```
ftp://myuser:mypasswd@192.168.100.1/myremotedir/requested-file-name
```

## Configuring DNS Caching for Standalone Content Engines

This section describes how to deploy Domain Name System (DNS) caching on standalone Content Engines, and covers the following topics:

- [About DNS Caching for Standalone Content Engines, page 7-63](#)
- [Configuring the DNS Caching Service \(Service 53\) for Standalone Content Engines, page 7-65](#)

## About DNS Caching for Standalone Content Engines

DNS is a system used in the Internet for translating names of network nodes into IP addresses. DNS allows the network to translate domain names entered in requests into their associated IP addresses. For example, when end users (web clients) enter `http://www.cisco.com` into their browsers, DNS translates the domain name `cisco.com` into its associated IP address so that these requests can be processed (that is, the requested content can be served to the web clients).

DNS caching allows the Content Engine to cache DNS entries to avoid multiple WAN accesses for DNS server resolution. When you enable DNS caching on a standalone Content Engine, the Content Engine caches the results of recent DNS queries for faster resolution of identical queries in the future. This cached information is then made available to clients making future requests. The ability to store DNS information that can then be distributed to requesting clients turns the Content Engine into a DNS caching name server.

**Caution**

It is assumed that you are enabling the DNS caching with WCCP interception on a standalone Content Engine.

In centrally managed ACNS networks, configuring the DNS caching service with WCCP interception on a centrally managed Content Engine causes a conflict with the Content Router, because they will both be listening for DNS requests on the same port (port 53). Consequently, they are mutually exclusive and you should not configure DNS cache support with WCCP interception in such environments. You can, however, enable the standard DNS caching service (without WCCP interception support) in centrally managed ACNS networks.

To configure DNS caching on a standalone Content Engine, you can use the Content Engine GUI or CLI. You must specify the IP address of the DNS server that the Content Engine should use for domain name resolution, and then enable DNS caching on the Content Engine. By default, DNS caching is disabled on a Content Engine.

To enable DNS caching on a standalone Content Engine, you must complete the following tasks:

- Specify the list of DNS servers, which are used by the network to translate requested domain names into IP addresses that the Content Engine should use for domain name resolution.
- Specify the name of the local domain.
- Specify the DNS cache size; that is, the maximum number of records that the DNS cache on the Content Engine should store.
- Enable the WCCP Version 2 DNS caching service (the `dns service [service 53]`) on the Content Engine.

Transparent interception of DNS requests using WCCP was added in the ACNS 5.1 software release. To enable this feature, you must configure the WCCP Version 2 DNS caching service (service 53) on the Content Engine and the WCCP Version 2-enabled router. For more information on this topic, see the [“DNS WCCP Transparent Interception Overview”](#) section on page 7-64.

## Domain Name Resolution Requirements

Domain name resolution requires that at least one DNS name server be configured on the Content Engine. You can configure one or more DNS name servers for the Content Engine by defining a list of DNS servers for the Content Engine through the Content Engine GUI (the **System > DNS** option) or the CLI (the **ip name-server** global configuration command). For more information about defining this list of DNS servers, see the “[Configuring DNS Servers for the DNS Caching Service \(Service 53\)](#)” section on page 7-64.

## DNS WCCP Transparent Interception Overview

For transparent interception of DNS requests using WCCP, you must configure the DNS caching service (service 53) on the Content Engine and on a router that supports WCCP Version 2.

The DNS process interacts with the WCCP process in these ways:

- Maintains the bypass lists.
- Monitors the aliveness of the DNS process to make sure that it can accept requests. If the DNS cache has no servers that are responsive, it will deregister the service until it has acceptable servers.
- Configures and manages the WCCP DNS caching service (the dns service [service 53]).

By default, the ACNS DNS caching service (service 53) uses the DNS servers configured on the Content Engine rather than the original DNS server. For information about how to configure a list of DNS servers on the Content Engine, see the next section, “[Configuring DNS Servers for the DNS Caching Service \(Service 53\)](#).”

## Configuring DNS Servers for the DNS Caching Service (Service 53)

By default, the Content Engine uses a DNS server from its list of configured DNS servers for domain name resolution.

- List of configured DNS servers—DNS servers that are used in the network and have been added to the list of DNS servers that the Content Engine should use for domain name resolution. (This list of configured DNS servers is created through the **ip name-server** command or the **System > DNS** Content Engine GUI.).
- Original DNS server—DNS server from the original request (hereafter referred to as the original DNS server)

If the DNS WCCP interception feature is enabled (that is, service 53 is configured on the Content Engine and a WCCP Version 2-enabled router), you can use the **dns use-original-server** global configuration command to define which DNS server a standalone Content Engine should use to resolve a domain name, as described in [Table 7-14](#).

**Table 7-14** Specifying DNS Servers for the DNS Caching Service with WCCP Version 2 Interception

CLI Command (Abbreviated Syntax)	Purpose
<code>dns use-original-server only</code>	Configures the DNS cache service (service 53) on a Content Engine to use only the original DNS server and not a DNS server from its list of configured DNS servers.
<code>dns use-original-server after-configured</code>	Configures the DNS cache service on a Content Engine to try the configured DNS servers first and if they fail, then to try the original DNS server.
<code>dns use-original-server before-configured</code>	Configures the DNS cache service on a Content Engine to try the original DNS server first, then the configured DNS servers.
<code>no dns use-original-server</code>	Configures the DNS cache service on a Content Engine to use only the list of configured DNS servers. This is the default.

You can use the Content Engine GUI or the CLI to configure one or more DNS servers for the Content Engine



**Note** From the Content Engine GUI, choose **System > DNS**, and use the displayed DNS window. For more information about the DNS window, click the **HELP** button in the window.

## Configuring the DNS Caching Service (Service 53) for Standalone Content Engines

The DNS caching service (the dns service [service 53]) is the WCCP service that permits WCCP Version 2-enabled routers to redirect client requests transparently to a Content Engine for the Content Engine to resolve the DNS name. After the Content Engine resolves the DNS name, it stores the resolved DNS name locally so that it can use these resolved names for future DNS requests.

To configure DNS caching for a standalone Content Engine, follow these steps:

**Step 1** Enable WCCP Version 2 on the router. (WCCP Version 1 does not support the dns caching service.)

```
Router# configure terminal
Router(config)# ip wccp version 2
```

**Step 2** Enable the dns caching service (service 53) on the router.

```
Router(config)# ip wccp 53
```

**Step 3** Specify the interface on which service 53 will run.

```
Router(config)# interface type number
```

**Step 4** Configure the router to use the outbound interface for service 53.

```
Router(config-if)# ip wccp 53 redirect out
```

- Step 5** Configure the Content Engine to run WCCP Version 2.

```
ContentEngine(config)# wccp version 2
```

- Step 6** Configure the Content Engine to run service 53.

```
ContentEngine(config)# wccp dns
```

- Step 7** Configure the DNS server port to listen for new client queries and invoke the query resolution routines. Once the hostname has been resolved to an IP address, it is stored in the memory-based DNS cache.

In the following example, the listener IP address, port number, and hostname are configured first. Then DNS caching is enabled on the Content Engine.

```
ContentEngine(config)# dns listen 10.1.1.0 port 53 hostname acme
```

If the DNS listen name does not match a DNS name, use the **dns pin** global configuration commands to pin an IP address to name mapping. The **dns pin** global configuration commands (**both**, **cname**, **forward**, and **reverse**) allow you to lock an IP address against a name within the cache. The **forward** option maps the hostname to the IP address. The **reverse** option maps the IP address to the hostname. The **both** option maps in both the forward and reverse directions. The **cname** option inserts the canonical name (CNAME) mapping.

- Step 8** Set the length of time that must elapse before an unanswered request is discarded with the **dns retry-period** global configuration command.

- Step 9** Set the interval between retransmission of User Datagram Protocol (UDP) DNS requests sent to an upstream DNS server with the **dns retry-timeout** global configuration command.

Because the DNS protocol is using UDP packets that can be lost or dropped, the burden of retransmitting DNS requests is on the requester. Typically, a retransmit is initiated every 3 seconds until a response is received, or if a response is not received, the request times out after 60 seconds. If a DNS server times out, then a new upstream server is selected to query. If there are no more servers to query upstream, then the initial DNS server contacted returns a DNS failed response to the requesting client.

- Step 10** Configure this standalone Content Engine to query the configured name servers repeatedly if the initial DNS server contacted fails to respond with the **dns serial-lookup** global configuration command.

- Step 11** Start the DNS server on this standalone Content Engine.

```
ContentEngine(config)# dns enable
```




---

**Note** Enabling the DNS server creates an entry of 127.0.0.1 as the name server for the system and starts the memory-based DNS cache.

---

- Step 12** Specify the maximum number of resource records that can be stored in the DNS cache on this standalone Content Engine with the **dns max-cache-memory** global configuration command. (This is an optional step if you want to use the default setting of 10,000 records.)

To avoid unduly straining overall system resources, it is important that you impose a strict maximum memory limit for Content Engine. In the following example, the size of the DNS cache is set to 20,000 records:

```
ContentEngine(config)# dns-cache size 20000
```

**Note**

To use the Content Engine GUI to configure the maximum DNS cache size, choose **System > DNS** from the Content Engine GUI. Use the displayed DNS window to specify the size of the DNS cache and click **Update**.

## Disabling DNS Caching on Standalone Content Engines

To disable DNS caching on a standalone Content Engine, enter the **no dns-cache size** global configuration command.

```
ContentEngine(config)# no dns-cache size
```

## Configuring Standalone Content Engines to Send out TCP Keepalives

By default, the Content Engine does not automatically send out keepalives. To configure a standalone Content Engine to send out TCP keepalives on HTTP connections, you must enter the **http tcp-keepalive enable** global configuration command. After entering the **http tcp-keepalive enable** command, the Content Engine will send out a keepalive every 75 seconds on an HTTP connection. If a response is received, the Content Engine continues to send a keepalive every 75 seconds. If a response is not received (the device does not respond), the Content Engine waits 90 seconds and logs a miss. After four misses, the Content Engine considers the HTTP connection to be down and closes the connection.

Specify how many times the Content Engine should attempt to connect to the device before closing the connection with the **tcp keepalive-probe-cnt** global configuration command. The count can be from 1 to 10. The default is 4 attempts.

Specify how often the Content Engine is to send out a TCP keepalive with the **tcp keepalive-probe-interval** global configuration command. The interval can be from 1 to 120 seconds. The default is 75 seconds.

Configure the Content Engine to wait for a response (the device does not respond) before it logs a miss with the **tcp keepalive-timeout** global configuration command. The timeout can be from 1 to 120 seconds. The default is 90 seconds.

# Configuring Persistent Connections on Standalone Content Engine

Content Engines by default use persistent connections to the server for improving performance. In the ACNS 5.0.7 software and later releases, the **rule action no-persistent-connection** global configuration command allows you to disable or enable persistent connections for specific domains, source and destination IP addresses, or ports. This is useful when a server does not support persistent connections.

## Enabling Persistent Connections

The Content Engine keeps a connection persistent if persistence is allowed for the persistence idle timeout period (which is 600 seconds by default). If HTTP persistent connections are enabled, then no keepalive is needed and the Content Engine will keep the connection open until the idle timeout period is exceeded.



**Note** The Content Engine does not automatically send out keepalives. To configure the Content Engine to send out TCP keepalives over an HTTP connection, you must enter the **http tcp-keepalives enable** global configuration command.

Once a response or data is sent over the persistent connection, the idle period restarts. HTTP persistent connections can be configured for either the client or server or both.

The Content Engine GUI or the CLI can be used to enable and disable persistent connections on standalone Content Engines.

To use the Content Engine GUI to enable persistent connections on a standalone Content Engine, choose **Caching > Persist. Connect**, and use the displayed Persistent Connections window. For more information about how to use the Persistent Connections window to enable persistent connections, click the **HELP** button in the window.

To use the Content Engine CLI to enable persistent connections on standalone Content Engines, use the **http persistent-connections** global configuration command:

```
ContentEngine(config)# http persistent-connections [all|client-only|server-only]
timeout seconds
```

Table 7-15 describes the HTTP persistent connection parameters.

**Table 7-15 HTTP Persistent Connection CLI Parameters**

Parameter	Description
<b>persistent-connections</b>	Sets persistent connections configuration options.
<b>all</b>	(Optional) Makes client and server connections persistent.
<b>client-only</b>	(Optional) Makes only a client connection persistent.
<b>server-only</b>	(Optional) Makes only a server connection persistent.
<b>timeout</b>	(Optional) Sets persistent connections timeout value (idle timeout period).
<i>seconds</i>	Persistent connections timeout in seconds (1–86400).

The persistence does not start until an initial request is made (for example, a GET request) or until data starts to flow over the persistent connection. If there is no initial request or data sent over a persistent connection, the read-write (rw)-timeout setting takes effect. The rw-timeout setting also is used if the connection goes idle for some reason before it has finished sending or receiving the data. In this case, the connection is timed out for the period specified by the rw-timeout setting. The rw-timeout setting can be set for reading and writing data to either the server or the client through the **tcp server-rw-timeout** and **tcp client-rw-timeout** global configuration commands. By default, the rw-timeout for both the server and the client is set to 120 seconds. For more information on this topic, see the “[Viewing or Modifying TCP Parameters on Standalone Content Engines](#)” section on page 20-2.

## Disabling Persistent Connections

To disable all persistent connections, client-only persistent connections, or server-only persistent connections on a standalone Content Engine use the **no** form of the **http persistent-connections** [**all** | **client-only** | **server-only** | **timeout seconds**] global configuration command:

```
ContentEngine(config)# no http persistent-connections [all|client-only|
server-only|timeout seconds]
```

To disable specific persistent connections to specific domains, IP addresses, or ports use the **rule action no-persistent-connection** global configuration command:

```
ContentEngine(config)# rule action no-persistent-connection
pattern-list list_num [protocol {http|https|ftp}]
```

The **rule action no-persistent-connection** global configuration command has the following options:

- all—Do not use persistent connection for all connections.
- client-only—Do not use persistent connection for client connections only.
- server-only—Do not use persistent connection for server connections only.

You can specify the criteria for disabling persistent connections by creating a pattern list using one or more of the following supported patterns:

- src-ip
- dst-ip
- dst-port
- url-regex
- domain
- header-field user-agent
- header-field referer
- header-field request-line

[Table 7-16](#) describes the syntax for the **rule action no-persistent-connection** command.

**Table 7-16** Parameters of the rule action no-persistent-connection Command

Parameter	Description
<b>action</b>	Describes the action that the rule is to take.
<b>no-persistent-connection</b>	Sets persistent connection configuration options.
<b>pattern-list</b>	Configures the pattern list.

**Table 7-16 Parameters of the rule action no-persistent-connection Command (continued)**

<i>list_num</i>	Pattern list number (1 to 512).
<b>protocol</b>	Protocol for which this rule is to be matched.
<b>http</b>	Matches this rule with the HTTP protocol.
<b>https</b>	Matches this rule with the HTTPS protocol.
<b>ftp</b>	Matches this rule with the FTP protocol.

The following example disables a persistent connection for the domain mywebsite.com, based on a pattern in pattern list 10:

```
ContentEngine(config)# rule action no-persistent-connection server-only pattern-list 10
WARNING: rule action no-persistent-connection will affect end-to-end NTLM authentication
to these servers
ContentEngine(config)#

ContentEngine# show rule all
Rules Template Configuration
-----
Rule Processing Enabled

Actions :
rule action no-persistent-connection server-only pattern-list 100 protocol all

Pattern-Lists :
rule pattern-list 100 domain mywebsite.com
ContentEngine#
```

For more information about using the Rules Template feature to configure rules for standalone Content Engines, see [Chapter 13, “Configuring the Rules Template on Standalone Content Engines.”](#)

## Configuring Healing Mode for Content Engine Clusters

Healing mode allows a newly added Content Engine to query and obtain cache objects from all other Content Engines in the Content Engine cluster on a cache miss. If the object is not found in the cluster, the Content Engine processes the request through the outgoing proxy or origin server. The Content Engine in healing mode is called a *healing client*. The Content Engines in the Content Engine cluster that respond to healing client requests are called *healing servers*.

When a Content Engine is added to an existing Content Engine cluster running WCCP Version 2, it can receive requests for content that was formerly served by another Content Engine in the Content Engine cluster. This event is termed a *near-miss*, because if the request had been sent to the former Content Engine, it would have been a cache hit. A near-miss lowers the overall cache hit rate of the Content Engine cluster.



### Note

Healing mode is only invoked on a healing client when the request is transparently redirected to the Content Engine. Healing mode is not invoked when the client sends the request directly to the Content Engine (acting as a nontransparent forward proxy server).

To allow a Content Engine in a Content Engine farm to query and obtain cache objects from other Content Engines in the cluster, you must enable healing mode on the Content Engine, using the Content Engine GUI or CLI to enable healing mode on a standalone Content Engine.

To configure the clustering parameters (the parameters related to WCCP service clusters) from the Content Engine GUI, choose **WCCP > Clustering**. Use the Clustering window to specify these parameters for this Content Engine. For more information about how to use the Clustering window to specify these parameters, click the **HELP** button in the window.

To use the Content Engine CLI to enable healing mode on standalone Content Engines, use the **http cluster** global configuration command:

```
http cluster {heal-port number | http-port number | max-delay seconds | misses number}
```

Table 7-17 describes the **http cluster** command parameters.

**Table 7-17** Healing Mode CLI Parameters

Parameter	Description
<b>cluster</b>	Configures cache cluster options for the Content Engine.
<b>heal-port</b>	Listening port for the healing server for healing requests.
<i>number</i>	Healing server listener port number (1–65535). The default is 14333.
<b>http-port</b>	HTTP port number over which requests from the healing Content Engine are sent to other Content Engines in the cluster. The default port number for the HTTP healing port is port 80. Valid port numbers are from 1 to 65535.
<i>number</i>	HTTP request forwarding port number (1–65535). The default is 80.
<b>max-delay</b>	Maximum wait for a response from the Content Engine cluster.
<i>seconds</i>	Maximum delay in seconds (0–10).
<b>misses</b>	Duration of healing mode for the Content Engine.
<i>number</i>	Total number of misses (0–999) before healing mode is disabled.

Specify the port number over which requests from the healing Content Engine (healing client) are sent to the healing servers (other Content Engines in the cluster) with the **http cluster http-port** global configuration command. The default port number is port 80. If you choose to configure a port other than port 80, make sure that it matches the port that was specified on the healing servers in the cluster, using the **http proxy incoming** global configuration command. Otherwise, the healing client is not able to retrieve objects from the healing servers.

Specify the port number over which the healing client sends healing queries and the healing server sends healing responses with the **http cluster heal-port** global configuration command. The default port number is 14333. If a port other than the default is configured, make sure that all Content Engines in the cluster use the same port.

Specify the maximum number of misses that the healing Content Engine can receive from the cluster from the last healing mode hit response until the healing process is disabled with the **http cluster misses** global configuration command. The default is 0 misses.

After a WCCP bucket redistribution, the Content Engine will try to populate its cache from other Content Engines on every cache miss. You can configure the maximum number of seconds a Content Engine should wait for a response from its neighbors before retrieving the object itself. The default is 0 seconds. To specify the maximum time in seconds that a healing Content Engine waits for a healing response from the cluster before considering the healing request a miss, use the **http cluster max-delay** global configuration command.

To enable the healing client, you should, at the least, configure the **max-delay** and **misses** options. The default port number for **http-port** is 80. If you use the default port, you do not have to configure **http-port**. The default port number for **heal-port** is 14333.

The following example enables the healing mode feature by setting the HTTP port for forwarding HTTP requests to a healing server, setting the maximum delay to wait for a response from the cluster in seconds before considering the healing request a miss, and setting the maximum number of misses that the healing Content Engine (healing client) can receive from the cluster before healing mode is disabled at the healing client.

```
ContentEngine(config)# http cluster http-port 8080
ContentEngine(config)# http cluster max-delay 5
ContentEngine(config)# http cluster misses 5
```

To disable the healing client, you should, at the least, configure either the **misses** or **max-delay** option to 0, or you can use the **no** form of the **http cluster misses** and **http cluster max-delay** global configuration commands:

```
ContentEngine(config)# no http cluster misses
ContentEngine(config)# no http cluster max-delay
```

## Configuring the Internet Cache Protocol for Content Engine Clusters

Internet Cache Protocol (ICP) is a lightweight message format used for communicating among Content Engines and for supporting interoperability with older proxy protocols. ICP is used to exchange hints about the existence of URLs in neighboring Content Engines in a Content Engine cluster (farm). Content Engines exchange ICP queries and replies to gather information for use in selecting the most appropriate location from which to retrieve an object.

Although ICP has been used traditionally to scale the overall size of a cluster of Content Engines beyond a single unit, history has shown ICP to be a poor way of scaling a Content Engine clustering solution. In fact, because of the way that traffic is currently directed toward a transparent network Content Engine cluster, the requirement for ICP is all but negated for the majority of Content Engine deployments.

The ICPv2 protocol is documented in two standards documents:

- *RFC 2186: Internet Cache Protocol (ICP), Version 2*
- *RFC 2187: Application of Internet Cache Protocol (ICP), Version 2*



### Note

The ability to act as both an ICP server (servicing requests from neighboring Content Engines) and an ICP client (sending requests to neighboring Content Engines) is supported.

The following example shows how to use the Content Engine CLI to restrict the ICP parent and sibling to specific domain sets:

```
ContentEngine(config)# icp client add-remote-server 10.1.1.1 parent icp-port 3130
http-port 3128 domain_x.com domain_y.com domain_z.com
ContentEngine(config)# icp client add-remote-server 10.1.1.1 sibling icp-port 3130
http-port 3128 domain_a.com domain_b.com domain_c.com
ContentEngine(config)# icp client enable
ContentEngine(config)#
```

You can use the Content Engine CLI or GUI to configure ICP on a standalone Content Engine that is part of a cache cluster, as described in the following sections:

- [Configuring Standalone Content Engines as ICP Clients, page 7-73](#)
- [Configuring Standalone Content Engines as ICP Servers, page 7-74](#)

## Configuring Standalone Content Engines as ICP Clients

You can configure your Content Engine cluster to generate ICP queries before retrieving requested objects from the Internet using ICP client functionality. With ICP, you can configure parent and sibling Content Engines in a caching hierarchy. ICP parents are essentially one step higher than ICP siblings in a hierarchy of Content Engines.

You can configure a standalone Content Engine to be either a parent or a sibling:

- Parent Content Engines are able to retrieve data during a cache miss.
- Sibling Content Engines cannot retrieve data and instead forward the request to the parent Content Engines.

You can use the Content Engine CLI or GUI to configure a standalone Content Engine as an ICP client, as follows:

- From the Content Engine GUI, choose **Caching > ICP Client**, and use the ICP Client window. To obtain more information about this window, click the **HELP** button in the window.
- From the Content Engine CLI, use the **icp client** global configuration commands to configure a standalone Content Engine as an ICP client. Configurations made without enabling ICP functionality are stored within the configuration until removed.

Table 7-18 describes the parameters of the **icp client** global configuration command.

**Table 7-18** *icp client Command Summary*

Command	Purpose
<b>icp client enable</b>	Enables the ICP client on a Content Engine.
<b>icp client add-remote-server</b>	Adds a remote ICP client server.
<b>icp client exclude</b>	Excludes ICP client local domains.
<b>icp client max-fail</b>	Sets the maximum number of retries allowed. Valid values are 0 to 100. The default is 20.
<b>icp client max-wait</b>	Configures how long the Content Engine waits before retrieving the requested data directly from the Internet.
<b>icp client modify-remote-server</b>	Modifies the ICP client remote server parameters.

The following example shows how to use the Content Engine CLI to restrict ICP parent and sibling to specific domain sets:

```
ContentEngine(config)# icp client add-remote-server 172.16.0.0
parent icp-port 3130 http-port 3128 domain_x.com domain_y.com domain_z.com

ContentEngine(config)# icp client add-remote-server 172.16.0.0
sibling icp-port 3130 http-port 3128 domain_a.com domain_b.com domain_c.com

ContentEngine(config)# icp client enable
Icp Client started
```

## Configuring Standalone Content Engines as ICP Servers

You can also configure a standalone Content Engine to act as an ICP server. This allows the Content Engine to probe the hierarchy of Content Engines by multicasting an ICP message to ICP parent and sibling clients in the hierarchy.

You can use the Content Engine GUI or the CLI to configure a standalone Content Engine as an ICP server, as follows:

- From the Content Engine GUI, choose **Caching > ICP Server** and use the displayed ICP Server window. To obtain more information about this window, click the **HELP** button in the window.
- From the Content Engine CLI, use the **icp server** global configuration commands to establish and configure the Content Engine as an ICP server. Configurations made without enabling ICP functionality are stored within the configuration until removed.

Table 7-19 describes the parameters for the **icp server** global configuration command.

**Table 7-19** *icp server Command Summary*

Command	Purpose
<b>icp server enable</b>	Enables the ICP server on a Content Engine.
<b>icp server http-port</b>	Configures the HTTP proxy port on a Content Engine to listen for ICP-generated requests. The range is from 0 to 65535. The default port number is 3128.
<b>icp server port</b>	Configures the ICP server port on a Content Engine to listen for ICP requests. The range is from 0 to 65535. The default port number is 3130.