



Configuration Reference

This chapter describes the configuration options available through entries in the various configuration files that affect application appliance operation. The configuration files include these:

- [fgn.conf](#), page 5-1
- [httpd.conf](#), page 5-37
- [mimetypes.conf](#), page 5-38
- [useragent.conf](#), page 5-39
- [fgnsmnpd.conf](#), page 5-39
- [magentd.conf](#), page 5-39

The following sections cover the use of each of these files.

fgn.conf

This file is the main application appliance configuration file. It contains all of the application appliance-specific configuration elements. It includes the `mimetypes.conf` and `useragent.conf` files by reference.

The following topics are included in this section:

- [Application Class Specification](#), page 5-7
- [Destination Mapping Configuration](#), page 5-30
- [SSL Configuration](#), page 5-33
- [Dynamic Caching Configuration Guide](#), page 5-34

The Management Console lets you manage and edit the application appliance configuration file at both the cluster and node level within its graphical interface. You don't need to manually edit the `fgn.conf` file. For details on using the Management Console to manage application appliance node configuration, refer to [Chapter 7, "Management Console."](#)



Note

If the `fgn.conf` configuration file is edited on a Microsoft Windows system, it will contain Carriage Return (“\r”) characters in addition to the normal Unix LineFeed (“\n”) characters. When the `fgnctl` startup script is invoked, it detects this problem and warns you to correct this situation. If you receive this warning, you should edit the file to remove the Carriage Return characters. You can use the Linux `dos2unix` command to convert the file.

Table 5-1 describes each global configuration element that is important to understand or that you might want to change.

Table 5-1 fgn.conf Configuration Elements

Element	Description
AdminCookieExpire	Number of hours after which the AdminDefinedCookie expires. The default is 168 hours (7 days). Valid values range from 0 to INT_MAX. A floating point value is allowed.
AdminCookieReplaceExpire	Number of hours after which the AdminDefinedCookie expires, if it is a replacement cookie that was set as a result of receiving a request with another application appliance node's AdminDefinedCookie. (If such a request is received, the application appliance node replaces its AdminDefinedCookie with the new one.) The default is 48 hours. Valid values range from 0 to INT_MAX. A floating point value is allowed.
AdminDefinedCookie	Cookie specification used by a load balancer for application appliance clustering. This value takes the form <i>name=value</i> . Specify the same <i>name</i> for all the application appliances in a cluster. The <i>value</i> must be unique for each application appliance node in the cluster. The entire <i>name=value</i> string must be less than 40 characters.
ApplicationClass	This section of the configuration file defines individual Application Classes that apply to various URLs that might be requested by clients. For details, see the “Application Class Specification” section on page 5-7.
AppScope	Enables or disables AppScope performance monitoring. Valid values include On and Off (default). Set this element to On to allow AppScope performance monitoring for all classes. Note that this directive can also be used in an individual Application Class. If set there, it overrides the global default setting and controls performance monitoring for the class.
AppScopeCookieTTL	Once a client session has been randomly selected for performance measurement, this directive sets the length of time a particular client will be monitored, in seconds. The default is 0, meaning that only a single request will be measured for the client. Setting a larger value causes all requests from that client during the specified session time period to be measured for performance. Valid values range from 0 to 86400 (24 hours). This directive sets the lifespan of the FGNPERFMON cookie, which keeps track of the randomly assigned sampling group for a client (accelerated, pass-through, or not measured).
AppScopeOptimizeRatePercent	The percentage of all requests (or sessions) to be sampled for performance with acceleration (optimization) applied. All applicable optimizations for the class will be performed. Individual requests (or sessions, if AppScopeCookieTTL is greater than 0) are selected randomly. The default is 0. Valid values range from 0 to 100. This value plus AppScopePassThruRatePercent must not exceed 100. Note that this directive can also be used in an individual Application class. If set there, it overrides the global default setting and controls performance monitoring for the class.
AppScopePassThruRatePercent	The percentage of all requests (or sessions) to be sampled for performance without optimization. No optimizations for the class will be performed. The default is 100. Valid values range from 0 to 100. Individual requests (or sessions, if AppScopeCookieTTL is greater than 0) are selected randomly. Note that this directive can also be used in an individual Application class. If set there, it overrides the global default setting and controls performance monitoring for the class.

Table 5-1 *fgn.conf Configuration Elements (continued)*

Element	Description
AppScreen	Enables or disables the AppScreen feature at the global or AppScreen Class level. Valid values include On and Off (default). For more information on AppScreen, see Chapter 6, “AppScreen Configuration.”
AppScreenClass	This section of the configuration file defines individual AppScreen Classes that apply to various URLs that might be requested by clients. For more information on AppScreen Classes, see the “AppScreen Class” section on page 6-3.
BaseFileCompress	Configures the application appliance to compress base files for HTTP 1.1 requests. Valid values include On and Off (default).
BaseFilePrefix	Defines the prefix to be added to the start of base file URLs. This allows usage of simple regular expressions on urls to identify base files by means of the BaseFilePrefix. One such example is to enforce “stickiness” of basefile URLs to a given application appliance using a content switch. The default value is NULL.
BufferedLog	This directive allows users to turn on or off the buffering of the application appliance log. The default is OFF. This directive is deprecated and is useful only when LogLevel is set to debug (see the “Logging Level” section on page 5-38.)
BufferedLogSize	Specifies the maximum size (in KB) of shared memory segment used for buffering the application appliance log. The default is 100. Valid values range from 10 to 200. This directive is deprecated and is useful only when LogLevel is set to debug (see the “Logging Level” section on page 5-38.)
CacheDepth	The number of levels of subdirectories in the cache. The default is 8. Valid values range from 1 to 25.
CacheFanout	The number of subdirectories at each level of the cache directory tree. The default is 4. Valid values range from 1 to 100.
CachePruneKeepSize	Number of cache files selected for the next pruning. The default is 5. Valid values range from 2 to 5.
CachePruner	Allows users to turn cache pruners on or off when the application appliance starts.
CachePruneSampleSize	Number of cache files randomly selected for pruning. The default is 30. Valid values range from 8 to 30.
CacheRoot	Directory where the cache root is located. Any valid directory path is allowed. The default is \$AVS_HOME/perfnode/cache
CacheSize	Maximum size of the cache in KB. If not specified in fgn.conf, the default size is 100000 KB and the minimum size is 5000 KB. The default configured value in fgn.conf is 500000 KB.
CompressContent	Configures the application appliance to allow compression. Valid values include On (default) and Off.
CompressionMethod	Indicates the type of compression that the application appliance uses. Valid values include gzip and deflate (default). Deflate compression is essentially the same as gzip but without the gzip header and the trailing CRC value. It may work better for JavaScript files. Note that this directive can also be used in individual Application Classes.
ConnectionPooling	Configures the application appliance to allow pooled connections to the origin server. Valid values include On and Off (default).

Table 5-1 fgn.conf Configuration Elements (continued)

Element	Description
ConnectionReuse	Configures the application appliance to reuse TCP connections to the origin server. This improves performance. Valid values include: On The application appliance maintains the TCP connection to the origin server. Keepalive Keeps the connection to the origin server alive as long as the client's browser session exists. Off The connection is not reused.
ConnectionTimeout	Number of seconds of inactivity after which a connection is closed. The default is 600 seconds (10 minutes). Valid values range from 1 to INT_MAX.
ConnectionTimeToWait	Maximum number of seconds to wait for a connection to be made to the origin server. The default is 5 seconds. Valid values range from 1 to INT_MAX.
DefaultClientScript	Configures the application appliance to recognize the scripting language used on condensed content pages. Valid values include VBScript and JavaScript (default). Note that this directive can also be used in individual Application Classes. Note If the original content uses VBScript as its scripting language, you must set the DefaultClientScript VBScript directive globally or for the class.
DeltaOptimize	Configures the application appliance to allow delta optimization. Valid values include On (default) and Off.
DeltaOptimizeCacheableContent	Configures the application appliance to allow delta optimization of cacheable content. Normally, the application appliance detects cacheable content and prevents its delta optimization. Valid values include On and Off (default). Note that this directive can also be used in individual Application Classes.
DestinationMapping	This section of the configuration file defines virtual IP mappings. For details, see the “Destination Mapping Configuration” section on page 5-30 .
DNSTimeout	DNS caching timeout in seconds. A DNS lookup of an IP address is cached for this time. The default is 36000 (10 hours). Valid values range from 1 to INT_MAX.
ExcludeIFrames	Configures the application appliance to exclude IFrames from condensation. Valid values include On and Off (default).

Table 5-1 *fgn.conf Configuration Elements (continued)*

Element	Description
FgnStatLogArchivingPolicy	<p>Controls what happens to FgnStatLog files that are full and closed. This keyword can have these possible values:</p> <p>keep <i>num</i></p> <p>Keeps the <i>num</i> most recent log files and deletes all older files. The default value is 10; valid values range 1 to 10.</p> <p><i>num</i></p> <p>Same as keep <i>num</i>.</p> <p>leave</p> <p>delete</p> <p>move</p> <p>Each of these previously supported values now have no meaning. If they are specified, it is the same as specifying “keep 10” for the value of this directive. Additionally an entry is made in the error log.</p>
FgnStatLogFileSizeLimit	An integer that specifies the maximum size of each FgnStatLog file in megabytes. The default is 25; valid values range from 1 to 60. All log entries are placed in 1000 rotating files with names FgnStatLog.000 through FgnStatLog.999. This directive sets the maximum size of each file before the next file is created.
FgnWorkDir	Directory where the temporary files created by the shared memory data manager are located. Any valid directory path is allowed, except not on an NFS drive. The default is \$AVS_HOME/perfnode/workdir
FileCacheSize	Configures the size of the in-memory cache that the application appliance uses for base files and other cacheable objects. The default value is 64 MB. Valid values range from 8 to 1000 (1 GB).
FlashConnectLimit	Configures the application appliance to limit the number of artificial hosts used by the FlashConnect feature. Specify an integer limit; the default is 4. For more information on FlashConnect, see the “FlashConnect” section on page 2-12.
FlashConnectPrefix	Specifies a prefix to be inserted in embedded object URLs before the host name, when transformed by the FlashConnect feature. Specify a string; the default is “flashconnect”. For more information on FlashConnect, see the “FlashConnect” section on page 2-12.
HTTP10Compress	Configures the application appliance to compress files for HTTP 1.0 requests. Valid values include On and Off (default).
IgnoreOriginServerBody	<p>Specifies a comma separated list of response codes for which the response body must not be read (it’s ignored). For example: IgnoreOriginServerBody 302</p> <p>This example directs the application appliance to ignore the response body in the case of a 302 (redirect) response from the origin server.</p>
LogDir	Directory where the log files are located. Any valid directory path is allowed. The default is \$AVS_HOME/perfnode/logs
MaxAttemptsToOpenConnection	Maximum number of attempts to open a connection. The default is 12. Valid values range from 1 to 12.

Table 5-1 fgn.conf Configuration Elements (continued)

Element	Description
MaxBufferSizeForOriginServerObject	If the response from the origin server exceeds this size, the application appliance will start streaming (reading from the server and writing to the client simultaneously). It compresses the response, but no other optimizations are applied. The default value is 25000000 bytes (about 25 MB). Valid values range from 1000 to 25000000 bytes.
MaxCondensablePage	Maximum page size that can be condensed, in bytes. The default is 250000. Valid values range from the MinCondensablePage or 1024 bytes to 250000 bytes.
MaxConnectionPoolSize	Configures the maximum number of connections in the pool. When this limit is reached, the least recently used connection is closed and a new connection is made. The default is 20. Valid values range from 1 to 100.
MaxRequestsPerConnection	Maximum number of requests that a connection can serve. After serving this number of requests, the connection is closed. The default is INT_MAX. Valid values range from 1 to INT_MAX.
MaxSharedMemSegment	Defines the maximum size of shared memory segment to create. The default is 128. Valid values range from 4 to 128 MB.
MetaDataCacheSize	MetaDataCacheSize Size of the shared memory segment used by the shared memory data manager. The default is 16 MB. Valid values range from 4 KB to 16 MB.
MinCondensablePage	Minimum page size that can be condensed, in bytes. The default is 1024. Valid values range from 1 byte to the value of MaxCondensablePage.
MinSharedMemSegment	Defines the minimum size of shared memory segment to create. The default is 32. Valid values range from 4 to 128 MB.
Netscape4Compress	Configures the application appliance to compress base files for Netscape 4.x browsers. Valid values include On and Off (default). For this option to work, HTTP10Compress must be set to On.
ObjCachePercent	Defines the percentage of SharedMemory MetaDataCacheSize that would be used to store Metadata. The remaining is used to store base files. The default is 50%. Valid values range from 25% to 75%.
PruneInterval	Number of seconds that the pruning thread sleeps between cache pruning operations. After this interval, the thread wakes up to check if pruning is needed. The default is 900 (15 minutes). Valid values range from 1 to INT_MAX seconds.
RebaseDeltaPercent	The delta threshold at which rebasing is triggered. This number represents the size of a page delta relative to the page total size, expressed as a percentage. For more information about how this value is used, see the “ Smart Rebasing ” section on page 2-15. The default threshold is 50%. Valid values range from 0 to 10000%.
RebaseFlashForwardPercent	Rebase, based on the percent of FlashForwarded URLs in the response. Rebasing is triggered when the difference between the percentages of FlashForwarded URLs in the delta response and the base file exceed the threshold. The default is 50%. Valid values range from 0 to 10000.
RebaseHistorySize	This parameter controls how much history is kept before resetting. Once the sample collection reaches this size, we reset all rebase control parameters to zero and start over. This parameter is needed to prevent the base file from becoming too “rigid”. That is, if a base file has served well for, say over 1 million pages, then it would take another half million unfavorable responses before the base file can be rebased. The default value for this parameter is 1000. Valid values range from 10 to INT_MAX.

Table 5-1 *fgn.conf Configuration Elements (continued)*

Element	Description
RequestGroupingString	<p>Defines a string that is used to sort requests for AppScope reporting purposes. The string can contain a URL regular expression that defines a set of URLs in which URLs that differ only by their query parameters are to be treated as separate URLs in AppScope reports. The string can contain the parameter expander functions listed in Table 5-4 on page 5-16.</p> <p>Normally, in an AppScope report organized by URL, matching URLs that differ only in their query parameters are treated as the same URL and are not listed on separate lines. Use this configuration element to specify that for a URL, all variations based on query parameters are to be treated as separate URLs for reporting purposes. Each variation will appear on a separate line in the report.</p>
UrlMatchWithProtocol	<p>Use the true client URL for URL matching purposes in Application Classes. The true client URL includes the protocol (scheme). Valid values include On and Off (default if not specified). To use the true client URL for matching, set this to On. For example, when this keyword is set to On, a URL with the HTTP protocol is considered different from the same URL with the HTTPS protocol. When set to Off, those two URLs are considered the same for matching an Application Class.</p> <p>Beginning with software version 6.0, this keyword is set to On in the default configuration file.</p> <p>In previous versions, this keyword was called OldWayCondenserClassURLMatching.</p>
UTF8Detection	<p>Controls UTF-8 character set detection, determining whether pages with multibyte characters are Delta optimized. The UTF-8 character set is an international standard that allows web pages to display non-ASCII or non-English multibyte characters. If you Delta optimize such pages, it could break them. Valid values include On (detect UTF-8 pages and do not Delta optimize them; default if not specified) and Off (do not detect UTF-8 pages, and so allow them to be Delta optimized).</p>
UTF8Threshold	<p>Determines how many UTF-8 characters on a page constitute a UTF-8 character set page, for UTF-8 detection purposes. Use this threshold to fine tune the detection of multibyte UTF-8 character set pages. The default value for this parameter is 5. Valid values range from 1 to 1,000,000.</p>

Application Class Specification

This section describes how to configure an Application Class. The following topics are included:

- [OptimizationPolicy Element, page 5-12](#)
- [Base File Selection Policy, page 5-15](#)
- [Using Conditional Blocks, page 5-18](#)
- [Adaptive Caching Configuration, page 5-18](#)
- [Image Optimization Configuration, page 5-24](#)
- [URL Mapping Configuration, page 5-26](#)
- [Client View Logging Configuration, page 5-28](#)
- [XML/XSL Transformations, page 5-30](#)

Each named Application Class looks similar to the following:

```
<ApplicationClass Sample>
  Url "(.*)/catalog/(.*)$"
  CanonicalUrl ${1}/${param(category)}
  OptimizationPolicy DeltaOptimize, Compress,
    DeltaOptimizeAllUsers, FlashForward
  BaseFileAnonLevel 2
</ApplicationClass>
```

Each Application Class is defined by a series of configuration elements, described in [Table 5-2](#).

Table 5-2 ApplicationClass Configuration Elements

Element	Description
AppscopeLogNonInstrumented	Enables AppScope to log statistics for requests where client-side timings cannot be measured because the response page is not HTML or cannot accept the JavaScript instrumentation. Normally such requests are not logged. Valid values include On and Off (default). To allow AppScope to report on such transactions, define an Application Class with this keyword set to On. Then define a transaction type for this class by using the “ApplicationClass Matches” expression, and set the Substitute Client Timing, as described in the “ Substitute Client Timing ” section on page 8-38.
BaseFileAnonLevel	Specify an integer between 0 and 50 that defines the base file anonymity level for All-user condensation. This element is used only with the DeltaOptimizeAllUsers policy element. It sets the anonymity level for the base file, between 0 (no anonymity) and 50 (very high anonymity). For more details, see the “ Anonymous Base Files ” section on page 2-14.
CacheKeyModifier	Specify a regular expression if you want to modify the canonical URL portion of the cache key, for example to remove CDN information. For more details, see the “ Modifying the Cache Key ” section on page 5-20. The default is NULL.
CacheMaxTTL	The maximum time in seconds that an object without an explicit expiration time should be considered fresh. The default is 259200 (72 hours). Valid values range from 0 to INT_MAX.
CacheMinTTL	The minimum time in seconds that an object without an explicit expiration time should be considered fresh. For static caching (FlashForwardObject), this value should normally be 0. For dynamic caching (CacheDynamic) this value should be set to indicate how long the application appliance should cache the page. The default is 0. Valid values range from 0 to INT_MAX.
CacheParameter	Specify an expression including one or more parameter expander functions if you want to modify the parameter portion of the cache key. For more details, see the “ Modifying the Cache Key ” section on page 5-20. The default is NULL.
CacheTTLPercent	The percent of an object’s age at which an embedded object without an explicit expiration time is considered fresh. The default is 0. Valid values range from 0 to 100.
CanonicalUrl	Specify a string containing a canonical URL regular expression that defines a set of URLs to which the class applies. The CanonicalUrl element is described below in the “ Base File Selection Policy ” section on page 5-15. At least one URL must be specified using the Url or CanonicalUrl elements.
ExcludeNonASCII	Configures the application appliance to exclude non-ASCII data from condensation. Valid values include On and Off (default). Set this element to On if the content has UTF8 characters. This excludes such characters from Delta optimization but the rest of that page can still be Delta optimized.

Table 5-2 ApplicationClass Configuration Elements (continued)

Element	Description
ExcludeScripts	Configures the application appliance to exclude JavaScript data from condensation. Valid values include On and Off (default). Set this element to On if a page with JavaScript has rendering problems. This excludes the JavaScript from Delta optimization but the rest of that page can still be Delta optimized.
ExpiresSetting	Controls the period of time that the objects in the client's browser remain fresh. Configures the application appliance to FlashForward but not transform. Valid values include: CacheTTL Calculate the freshness similar to FlashForwarded objects and subject them to CacheMinTTL and CacheMaxTTL, if set. Unmodified Disables this feature (default) An integer between 0 and INT_MAX Number of seconds that the object should be considered fresh. (This sets the CacheTTL.)
ExtractMeta	Configures the application appliance to remove META elements from documents to prevent them from being condensed. Valid values include On and Off (default).
FFRefreshPolicy	Configures the application appliance to bypass FlashForward for stale embedded objects. Valid values include: Direct Bypass FlashForward for stale embedded objects so that they get refreshed directly. All Allow FlashForward to indirectly refresh embedded objects (default) Request headers that the application appliance sends to the origin server for stale embedded objects (indirect GET) may not be accepted by the origin server and cause errors. In this case, specify Direct to prevent this behavior.
FgnNamePrefix	Adds a prefix to the JavaScript function names (for example, b and o) used by the application appliance to avoid namespace collisions with existing functions on a Web site. Specify a prefix up to nine characters. The default prefix is fgn_.
IF/ELSE/ELSEIF/ENDIF	Defines a conditional block of configuration elements. For more information, see the “Using Conditional Blocks” section on page 5-18 .
ImageOptimization	Determines the degree of compression that is applied to JPEG and PNG images. For more information, see the “Image Optimization Configuration” section on page 5-24 .
LogClientView	Enables logging of true client IP addresses. For more information, see the “Client View Logging Configuration” section on page 5-28 .
MaxCacheableObjectSize	Maximum size of embedded cacheable objects that can be cached. The default is 10000000. Valid values range from MinCacheableObjectSize to 10000000 bytes.
MinCacheableObjectSize	Minimum size of embedded cacheable objects that can be cached. The default is 0. Valid values range from 0 to the lesser of MaxCacheableObjectSize or 1000000 bytes.
ModifyXForwardedFor	Modify the X-Forwarded-For header in requests sent to the origin server by adding the IP address of the client or requesting proxy. Valid values include On (default) and Off. To leave the X-Forwarded-For header unchanged, specify Off.

Table 5-2 ApplicationClass Configuration Elements (continued)

Element	Description
OptimizationPolicy	Specify a string of policy keywords separated by commas. The “ OptimizationPolicy Element ” section on page 5-12 describes the keywords that can be used in an OptimizationPolicy element. They control how the content corresponding to the specified URLs is treated by the application appliance.
ParamSummary	Enables the logging of query parameters for transactions in the FgnStatLog file (in the <PSM> element). Valid values include On (default) and Off.
ParamSummaryParamValueLimit	Sets the maximum number of bytes that are logged for each parameter value in the parameter summary of a transaction log entry in the FgnStatLog. Valid values range from 0 to 10,000 bytes; the default is 100 bytes. If a parameter value is longer than this limit, it is truncated at the limit.
PostContentBufferLimit	Sets the maximum number of kilobytes of POST data to scan for parameters for the purpose of logging transaction parameters in the FgnStatLog. Parameters beyond this limit will not be logged. Valid values range from 0 to 1000 KB; the default is 40 KB.
RequestCachePolicy	Provides a mechanism to override client request headers (primarily for embedded objects). Values include the following: OverrideAll All cache headers are ignored None No headers are overridden (default) Note that using this option violates the HTTP standard.
RequestHeader	Provides a mechanism to set client request headers to specific values. Note that case is ignored. Values include the following: Set <i>name value</i> Sets the value of the header identified by <i>name</i> to <i>value</i> . If the named header exists, its value is changed to <i>value</i> , and if it doesn't exist, it is created. Unset <i>name</i> Removes the header specified by <i>name</i> . It's not recommended to first unset and then set the same header in a single Application Class specification; the results may be unexpected.
ResponseCachePolicy	Provides a mechanism to override origin server response headers (primarily for embedded objects). Values include the following: OverrideAll All cache headers are ignored None No headers are overridden (default) Note that using this option violates the HTTP standard.

Table 5-2 ApplicationClass Configuration Elements (continued)

Element	Description
ResponseHeader	<p>Provides a mechanism to set response headers to specific values. Note that case is ignored. Values include the following:</p> <p><i>Set name value</i></p> <p>Sets the value of the header identified by <i>name</i> to <i>value</i>. If the named header exists, its value is changed to <i>value</i>, and if it doesn't exist, it is created.</p> <p><i>Unset name</i></p> <p>Removes the header specified by <i>name</i>.</p> <p>It's not recommended to first unset and then set the same header in a single Application Class specification; the results may be unexpected.</p>
Url	<p>Specify a string containing a URL to which the class applies. You can include a number of URL elements in the class definition to include multiple URLs in the class. URLs can be specified as regular expressions using the GNU POSIX syntax (see Appendix F, "Regular Expressions.") At least one URL must be specified using the <code>Url</code> or <code>CanonicalUrl</code> elements.</p>
Urlmap	<p>Allows the application appliance to alter URLs in the data stream between the origin server and the client browser. Note that this alteration applies only to HTML files, unless you also set the <code>UrlmapNonHtml</code> keyword to <code>On</code>.</p> <p>Specify a string that begins with the word <code>scope</code>, contains a scope keyword, and contains a replacement directive. For details on how to configure URL mapping, see the "URL Mapping Configuration" section on page 5-26.</p>
UrlmapNonHtml	<p>Configures the application appliance to allow URL mapping on files other than HTML files. Valid values include <code>On</code> and <code>Off</code> (default). Set this element to <code>On</code> if you want to map URLs in non-HTML files. This will enable URL mapping on all files, so make sure that you use the correct ApplicationClass to target selected files, otherwise performance can suffer.</p>
XsltMerge	<p>Enables the XSLT merge feature. Valid values include <code>On</code> and <code>Off</code> (default). This feature is used for XML source files. It causes the application appliance to transform the XML source by applying an XSLT stylesheet, and the resulting document is returned to the requester. The XSLT stylesheet can be specified in the XML document or specified by the <code>XsltUseStylesheet</code> directive. After the document is transformed, other optimizations are applied. For an example, see the "XML/XSL Transformations" section on page 5-30.</p>
XsltUseStylesheet	<p>Specify the URL of an XSLT stylesheet. Forces the use of this particular stylesheet, regardless of any XSL specified in the XML source file. This keyword applies only if the <code>XsltMerge</code> keyword is set to <code>On</code>.</p>

Application Classes are applied to a request in cascading order. That is, for a given request, the application appliance checks the classes in the order in which they are listed in the configuration file, until it finds a class that applies to the requested URL. The policy in that class is then applied to the request. If no class that applies to the request is found, the application appliance sends an uncondensed response.

OptimizationPolicy Element

Table 5-3 describes the keywords that can be used in an OptimizationPolicy element. The keywords are generally arranged in pairs. An OptimizationPolicy element can specify one keyword from each pair. If no keyword from a particular pair is specified, the default setting is used. An example OptimizationPolicy element is:

```
OptimizationPolicy DeltaOptimize,Compress,CondenseAllUser
```

If you specified this OptimizationPolicy element:

```
OptimizationPolicy DeltaOptimize,CondenseAllUser
```

the application appliance would also use Nocompress and NoMetaRefreshTo302 because compression and meta refresh keywords were not specified and these are the default settings.

Table 5-3 OptimizationPolicy Keywords

Keyword	Description
DeltaOptimize	The corresponding URLs are to be condensed. This is the default, if neither DeltaOptimize nor NoDeltaOptimize are specified in a particular policy.
NoDeltaOptimize	The corresponding URLs are not to be condensed.
Compress	The corresponding URLs are to be compressed.
Nocompress	The corresponding URLs are not to be compressed. This is the default, if neither Compress nor Nocompress are specified.
DeltaOptimizePerUser	The corresponding URLs are to be condensed using the Per-user condensation mode. For more details, see the “Configurable Condensation Modes” section on page 2-5 .
DeltaOptimizeAllUsers	The corresponding URLs are to be condensed using the All-user condensation mode. For more details, see the “Configurable Condensation Modes” section on page 2-5 . This is the default, if neither DeltaOptimizePerUser nor DeltaOptimizeAllUsers are specified.
ContentInspection	The content inspection feature is enabled for the corresponding URLs. This is the default, if neither ContentInspection nor NoContentInspection are specified.
NoContentInspection	The content inspection feature is disabled for the corresponding URLs.
FlashForward	The FlashForward feature is enabled for the corresponding URLs. Embedded objects will be transformed. For more details on this feature, see the “FlashForward Object Acceleration” section on page 2-6 .
NoFlashForward	The FlashForward feature is disabled for the corresponding URLs. This is the default, if neither FlashForward nor NoFlashForward are specified.
FlashForwardObject	The FlashForward static caching feature is enabled for the corresponding URLs. This is the default, if neither FlashForwardObject nor NoFlashForwardObject are specified.
NoFlashForwardObject	The FlashForward static caching feature is disabled for the corresponding URLs.
CacheDynamic	Adaptive Dynamic Caching is enabled for the corresponding URLs, even if the expiration settings in the response indicate that the content is dynamic. The expiration of cache objects is controlled by the cache expiration settings based on time or server load (performance assurance).
NoCacheDynamic	The Adaptive Dynamic Caching feature is disabled for the corresponding URLs. This is the default, if neither CacheDynamic nor NoCacheDynamic are specified.
DynamicETag	Just-In-Time Object Acceleration is enabled for the corresponding URLs. For more details on this feature, see the “Just-In-Time Object Acceleration” section on page 2-7 .

Table 5-3 OptimizationPolicy Keywords (continued)

Keyword	Description
NoDynamicETag	The Just-In-Time Object Acceleration feature is disabled for the corresponding URLs. This is the default, if neither DynamicETag nor NoDynamicETag are specified.
MetaRefreshTo302	Smart Redirect is enabled for the corresponding URLs. For more details on this feature, see the “Smart Redirect” section on page 2-11 .
NoMetaRefreshTo302	The Smart Redirect feature is disabled for the corresponding URLs. This is the default, if neither MetaRefreshTo302 nor NoMetaRefreshTo302 are specified.
MetaTagToResponseHeader	Configures the application appliance to parse for META tags and convert them to HTTP response headers. There is no counterpart to this policy keyword; if it is not specified then no such conversion is done.
FastRedirect	Intercepts 302 responses from the origin server and makes a second request on behalf of the client for the redirect URL, fetches it, and sends it to the client. This applies only to redirects within the same domain.
NoFastRedirect	The Fast Redirect feature is disabled for the corresponding URLs. This is the default, if neither FastRedirect nor NoFastRedirect are specified.
FlashConnect	The FlashConnect feature is enabled for the corresponding URLs. Embedded objects that match an Application Class that has the FlashConnectObject policy will be transformed. This feature is not enabled by default. For more details on this feature, see the “FlashConnect” section on page 2-12 . For details on configuring DNS in the origin server environment, see the “FlashConnect and Configuring DNS for the Origin Server” section on page 5-14 . For details on using FlashConnect and FlashForward together, see the “Using FlashConnect Together with FlashForward” section on page 5-15 .
FlashConnectObject	The FlashConnect performance acceleration feature is enabled for the corresponding embedded object URLs. For more details on this feature, see the “FlashConnect” section on page 2-12 . This feature is not enabled by default.
CacheForward	The CacheForward feature is enabled for the corresponding URLs. This allows the application appliance to serve the object from its cache (static or dynamic) even when the object has expired, if the CacheMaxTTL time period has not yet expired. At the same time, the application appliance sends an asynchronous request to the origin server to refresh its cache of the object. You may not specify this keyword together with CacheForwardWithWait; only one is valid.
NoCacheForward	The CacheForward feature is disabled for the corresponding URLs. This is the default, if neither CacheForward nor NoCacheForward are specified.

Table 5-3 OptimizationPolicy Keywords (continued)

Keyword	Description
CacheForwardWithWait	<p>The CacheForwardWithWait feature is enabled for the corresponding URLs. If the object has expired but the CacheMaxTTL time period has not yet expired, the application appliance sends a request to the origin server for the object. The rest of the users requesting this page will still continue to receive content from the cache during this time. When the fresh object is returned, it is sent to the requesting user and the cache is also updated. This is similar to CacheForward, except that a single user must wait for the object to be updated before their request is satisfied.</p> <p>This feature is useful in situations where you can't use CacheForward because the application requires a context for processing and an asynchronous update process isn't appropriate.</p> <p>You may not specify this keyword and CacheForward; only one is valid.</p>
NoCacheForwardWithWait	<p>The CacheForwardWithWait feature is disabled for the corresponding URLs. This is the default, if neither CacheForwardWithWait nor NoCacheForwardWithWait are specified.</p>

FlashConnect and Configuring DNS for the Origin Server

When using the FlashConnect feature it is essential to configure DNS in the origin server environment to send all requests for the rewritten object URLs to the appropriate origin server.

If a container page matches the FlashConnect policy, the FlashConnect feature causes embedded object URLs to be rewritten as in the following example.

Here is an example of a container page before optimization:

```
<html>
  
  
  
  
</html>
```

The embedded object URLs are rewritten by FlashConnect as follows:

```
<html>
  
  
  
  
</html>
```

URLs for embedded objects that match the FlashConnectObject policy are rewritten to be absolute with a host prefix that creates an artificial number of host servers (fgn00, fgn01, fgn02, fgn03, etc.). Additionally, the FlashConnect prefix ("flashconnect" in this example) is inserted before the actual host name to form a new absolute URL.

The origin server environment must have DNS configured with wildcards as follows. All URLs that match this expression:

```
fgn*.FlashConnectPrefix.myhost
```

must be mapped to the host myhost. The *FlashConnectPrefix* part is the string set by the FlashConnectPrefix global keyword.

In this way, all requests to the various artificial host servers are directed to the single actual host server.

Using FlashConnect Together with FlashForward

Unlike FlashForward, which relies on the application appliance having first cached the object, FlashConnect is not bound by this requirement; however, there is an issue that needs to be addressed in the fgn.conf configuration file if you use both of these policies.

Say there's an Application Class like this:

```
<ApplicationClass BadClass>
  Url ^.*\.jpg$
  OptimizationPolicy FlashForwardObject, FlashConnectObject
</ApplicationClass>
```

With this policy, any JPEG image will be FlashConnected, however, for JPEG images that do not return to the application appliance (for example, external site links) the FlashConnect URL will not resolve back to correct site, causing a broken link. The way to avoid this is to declare the FlashConnect policy in prior Application Classes that apply only to the internal site. Later Application Classes can then be used for FlashForwarding all remaining images.

As an example, imagine a site, mysite.com, with content like this:

```
<html>



</html>
```

The Application Classes should look like this:

```
<ApplicationClass HandleInternalImages>
  Url ^.*foo\.com.*\.jpg$
  OptimizationPolicy FlashForwardObject, FlashConnectObject
</ApplicationClass>

<ApplicationClass jpegs>
  Url ^.*\.jpg$
  OptimizationPolicy FlashForwardObject
</ApplicationClass>
```

Only those images that are hosted by mysite.com (me1.gif and me2.gif) will be both FlashConnected and FlashForwarded. External images not matching the domain will only be FlashForwarded.

Base File Selection Policy

The CanonicalUrl element in an Application Class is used to specify a base file selection policy. This is a regular expression that is used to match a variety of actual URLs. All matched URLs share a single base file.



Note

The application appliance requires GNU POSIX regular expression syntax. For more information, see [Appendix F, “Regular Expressions.”](#)

The CanonicalUrl element can contain parameter expander functions that evaluate to strings. [Table 5-4](#) lists the parameter expander functions that you can use.

Table 5-4 Parameter Expander Functions

Variable	Description
<code>\$(number)</code>	<p>This variable is expanded to the corresponding matching sub-expression (by <i>number</i>) in the URL pattern. Sub-expressions are marked in a URL pattern using parentheses (). The numbering of the sub-expressions begins with 1 and is the number of the left-parenthesis (“ counting from the left. You can specify any positive integer for the number. <code>\$(0)</code> matches the entire URL. For example, if the URL pattern is <code>((http://server/.*)/(.*)/a.jsp)</code>, and the URL that matched it is:</p> <pre>http://server/main/sub/a.jsp?category=shoes&session=99999</pre> <p>then:</p> <pre>\$(0) = "http://server/main/sub/a.jsp" \$(1) = "http://server/main/sub/" \$(2) = "http://server/main" \$(3) = "sub"</pre> <p>If the specified sub-expression does not exist in the URL pattern, then the variable expands to the empty string.</p>
<code>\$http_query_string()</code>	<p>This variable is expanded to the value of the whole query string in the URL. For example, if the URL is</p> <pre>http://myhost/dothis?param1=value1&param2=value2</pre> <p>then:</p> <pre>\$http_query_string() = "param1=value1&param2=value2"</pre> <p>This function applies to both GET and POST requests.</p>
<p><code>\$http_query_param(query-param-name)</code></p> <p>this obsolete syntax also is supported:</p> <p><code>\$param(query-param-name)</code></p>	<p>This variable is expanded to the value of the named query parameter (case sensitive). For example, if the URL is</p> <pre>http://server/main/sub/a.jsp?category=shoes&session=99999</pre> <p>then:</p> <pre>\$http_query_param(category) = "shoes" \$http_query_param(session) = "99999"</pre> <p>If the specified parameter does not exist in the query, then the variable expands to the empty string. This function applies to both GET and POST requests.</p>
<code>\$http_cookie(cookie-name)</code>	<p>Evaluates to the value of the named cookie. For example, <code>\$http_cookie(cookiexyz)</code>. The cookie name is case sensitive.</p>
<code>\$http_header(request-header-name)</code>	<p>Evaluates to the value of the specified HTTP request header. In the case of multivalued headers, it is the single representation as specified in the HTTP specification. For example, <code>\$http_header(user-agent)</code>. The HTTP header name is not case sensitive.</p>
<code>\$http_method()</code>	<p>Evaluates to the HTTP method used for the request, such as “GET” or “POST”.</p>

Table 5-4 Parameter Expander Functions

Variable	Description
Boolean Functions: <code>\$http_query_param_present (query-param-name)</code> <code>\$http_query_param_notpresent (query-param-name)</code> <code>\$http_cookie_present (cookie-name)</code> <code>\$http_cookie_notpresent (cookie-name)</code> <code>\$http_header_present (request-header-name)</code> <code>\$http_header_notpresent (request-header-name)</code> <code>\$http_method_present (method-name)</code> <code>\$http_method_notpresent (method-name)</code>	Evaluates to a Boolean value: “True” or “False,” depending on the presence or absence of the element in the request. The elements are: a specific query parameter (query-param-name), a specific cookie (cookie-name), a specific request header (request-header-name), or a specific HTTP method (method-name). All identifiers are case sensitive except for the HTTP request header name.
<code>\$regex_match (param1, param2)</code>	Evaluates to a Boolean value: “True” if the two parameters match and “False” if they do not match. The two parameters can be any two expressions, including regular expressions, that evaluate to two strings. For example, this function: <code>\$regex_match (\$http_query_param (URL), “^.*Store\.asp.*\$”)</code> compares the query URL with the regular expression string: <code>^.*Store\.asp.*\$</code> If the URL matches this regular expression, this function evaluates to True.

Here is an example of how this feature can be used. Suppose a catalog web site has the following pages:

```
http://server/catalog/business?category=pencils
http://server /catalog/business?category=erasers
http://server /catalog/consumer?category=pencils
http://server /catalog/consumer?category=erasers
```

Suppose the “pencils” pages for both the business and consumer sections have a lot of common content and similarly for the “erasers” pages. It would be efficient if the base-pages for these categories can be shared. That is, the base-page for the following two pages would be the same:

```
http://server/catalog/business?category=pencils
http://server/catalog/consumer?category=pencils
```

Using the CanonicalUrl element, you can configure the following Application Class:

```
<ApplicationClass PencilsAndErasers>
  Url " (./catalog) / (.*)$"
  CanonicalUrl $ (1) / $http_query_param (category)
  OptimizationPolicy DeltaOptimize, Compress,
    DeltaOptimizeAllUsers
</ApplicationClass>
```

This class will match all four URLs above. If the request URL is:

```
http://server/catalog/business?category=pencils
```

then `$(1)` is the string “http://server/catalog” and `$http_query_param (category)` expands to the string “pencils”. So the resulting canonical URL is: `http://server/catalog/pencils`

The same will hold for the URL `http://server/catalog/consumers?category=pencils`. Thus these two URLs will share the same base file.

Similarly the canonical URL for the eraser URLs will be: `http://server/catalog/erasers`, and they will share the same base file.

Using Conditional Blocks

You can use IF/THEN/ELSEIF/ENDIF conditional blocks to override global settings for an Application Class. The conditional blocks must come after all global settings in an Application Class specification; any global settings that appear after a conditional block are ignored.

The form of a conditional block is as follows:

```
IF Boolean-expression THEN
    configuration settings
ELSEIF Boolean-expression THEN
    configuration settings
ELSE
    configuration settings
ENDIF
```

In a conditional block, only the IF and ENDIF statements are required; ELSEIF and ELSE are optional. The ELSEIF statement combines an ELSE with another IF. Multiple ELSEIF statements are allowed, but only one ELSE is permitted.

Only one conditional block is allowed within an Application Class at the same level. However, a nested conditional block is allowed inside an IF, ELSEIF, or ELSE clause. This restriction applies to every conditional block recursively.

Like a standard conditional statement, if the Boolean expression evaluates to true, the statements following it (until an ELSE, ELSEIF, or ENDIF) are executed and the ELSE or ELSEIF block is skipped; and if it evaluates to false, the statements between it and the first ELSE or ELSEIF are not executed and the ELSE or ELSEIF block is executed.

For the Boolean expressions, you can use the parameter expander functions listed in [Table 5-4](#).

Adaptive Caching Configuration

This section includes reference information for configuring both static and dynamic adaptive caching in an Application Class.

For guidelines on when to use dynamic caching and the steps required to implement it, see the [“Dynamic Caching Configuration Guide”](#) section on page 5-34.

If an Application Class is found that matches a request, its cache policy is inspected and applied to that object.

There are two aspects of all cached objects:

- The key under which the object is cached. This is controlled by the keywords URL or CanonicalURL (described in the previous section), CacheKeyModifier, and CacheParameter. For details, see the [“Modifying the Cache Key”](#) section on page 5-20.
- The expiration behavior of the cached content. This can be time-based or performance assurance with load-based expiration. For details on configuring expiration, see the [“Configuring Cache Object Expiration”](#) section on page 5-22.

The keywords that are used to configure dynamic caching in an Application Class are as follows:

```
<ApplicationClass className>
```

```

# URL matching
Url OR CanonicalURL ...

# Class cache key modifier
CacheKeyModifier ...

# Class parameterization
CacheParameter ...

# Class policy
  OptimizationPolicy CacheDynamic, ...

# Class expiration
CacheMinTTL ...
CacheMaxTTL ...
ServerLoadShortWindow ...
ServerLoadLongWindow ...
ServerLoadTriggerPercent ...
CacheTTLChangePercent ...

# Conditional blocks to override class settings
IF ... THEN ...
ELSEIF ... THEN ...
ELSE ...
ENDIF
</ApplicationClass>

```

The following topics are included in this section:

- [Cache Object Key, page 5-19](#)
- [Modifying the Cache Key, page 5-20](#)
- [Configuring Cache Object Expiration, page 5-22](#)
- [Example of Cache Configuration, page 5-23](#)

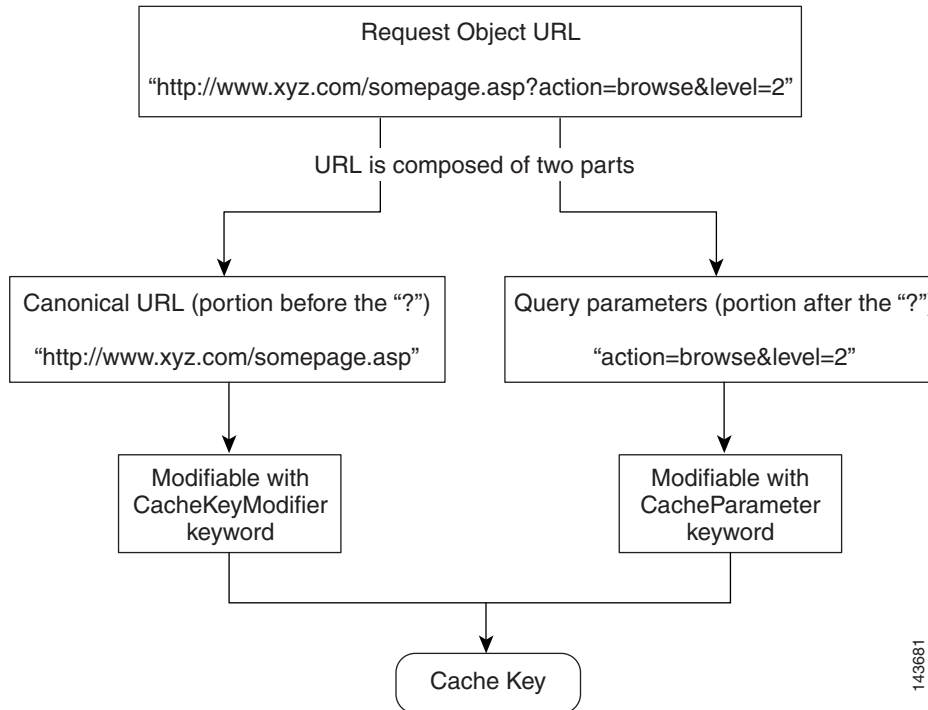
Cache Object Key

The cache object key is the unique identifier that is used to identify a cached object to be served to the client, thereby replacing a trip to the origin server.

The HTTP protocol is not session based (each request for every page and dependent object is entirely autonomous, with no state preserved between requests), leading web application developers to use session tracking techniques like cookies. Thus, it is sometimes necessary to include more in the cache key than simply the URL.

The key that the application appliance uses for any given requesting URL comprises one or more of the following two components, which are illustrated in [Figure 5-1](#):

- Canonical URL (the URL portion up to a “?”). This can be modified by the CacheKeyModifier keyword.
- Query parameters (the URL portion after a “?”). This can be modified by the CacheParameter keyword, which can be used to include just selected query parameters, a cookie value, an HTTP header value, and other values.

Figure 5-1 How Cache Key is Formed from URL

143681

For details on how to use the CacheKeyModifier and CacheParameter keywords, see the next section, [“Modifying the Cache Key”](#).

Modifying the Cache Key

You can modify the cache key by using the following syntax elements:

- [CacheKeyModifier](#) keyword
- [CacheParameter](#) keyword

CacheKeyModifier

The CacheKeyModifier element in an Application Class is used to modify the canonical form of a URL—that is, the portion before the “?”—for use in forming the cache key.

The CacheKeyModifier element specifies a regular expression containing embedded variables that are expanded by the application appliance. You can include zero or more instances of the $\$(number)$ variable, as described in [Table 5-4 on page 5-16](#). Note that only regular expressions are supported here, not any of the other parameter expander functions.

The expanded string resulting from the CacheKeyModifier element replaces the default canonical URL portion of the cache key. If the CacheKeyModifier element is not specified, the canonical URL is used as the default value for the URL portion of the cache key (there may also be a query parameter portion).

Here is an example of using a CacheKeyModifier element to strip out the portion of a URL added by a content delivery network (CDN):

```

<ApplicationClass Example_1>
  Url "^.*mycdn\.net.*www(.*\.gif)$"
  CacheKeyModifier http://www$(1)
  OptimizationPolicy ...

```

```
</ApplicationClass>
```

The `Url` line specifies a regular expression that identifies URLs for which this Application Class is to be used. It reads: Start with any number of any characters, up to the literal “mycdn.net”, followed by any sequence of characters up to the literal “www”, followed by a sub-expression group (in parentheses) that ends the URL. The sub-expression group is any sequence of characters followed by the literal “.gif”, which must end the string. The sub-expression group can be expanded by using the $\$(number)$ syntax. In this case it’s the first and only sub-expression, so it can be referenced by $\$(1)$.

The `CacheKeyModifier` line replaces the original URL with a new string that begins with “http://www” and ends with the value of sub-expression group 1 from the previous line. The effect is to strip out the portion of the URL that was used for redirection to a CDN. It transforms a matched url such as the following:

```
http://a188.g.mycdn.net/f/188/920/1d/www.mysite.com/images/logo.gif
```

to this string:

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

CacheParameter

The `CacheParameter` element in an Application Class is used to modify the query parameter part of a URL—that is, the portion after the “?”—for use in forming the cache key.

The `CacheParameter` element specifies one or more parameter expander functions that evaluate to strings. These strings are appended to the canonical URL to form the last portion of the cache key. The parameter expander functions are listed in [Table 5-4 on page 5-16](#).

The string specified in the `CacheParameter` element replaces the default query parameter that is used in the cache key. If the `CacheParameter` element is not specified, the query parameter portion of the URL is used as the default value for this portion of the cache key (the canonical URL, possibly modified by the `CacheKeyModifier`, composes the first part of the cache key).

Here is an example of using a `CacheParameter` element to create a different instance of the dynamically cached page for each value of the “version” query parameter:

```
<ApplicationClass Example_2>
  Url "^.*dyncache/page3\.asp.*$"
  CacheParameter $http_query_param (version)
  OptimizationPolicy ...
</ApplicationClass>
```

The `Url` line specifies a regular expression that identifies URLs for which this Application Class is to be used. It reads: Start with any number of any characters, up to the literal “dyncache/page3.asp”, followed by any sequence of characters up to the end of the URL.

The `CacheParameter` line sets the value of the query parameter portion of the cache key to just the value of the “version” query parameter. (The default value of the query parameter portion of the cache key is the entire query parameter portion of the URL.)

It extracts from a matched url such as the following:

```
http:www.mysite.com/dyncache/page3.asp?session=nqyfxe46&m=int&version=12
```

this string, which is the value of the “version” parameter:

```
12
```

This is appended to the URL portion of the cache key to form the complete cache key.

Configuring Cache Object Expiration

There are two ways in which an object in the cache can be expired (excluding the natural process of cache pruning):

- Time based, where the object lives for some minimum and maximum time interval.
To specify time-based expiration for the cache, use the CacheMinTTL (default=0) and CacheMaxTTL (default=259200, 72 hours) keywords.
- Performance assurance with load-based expiration, where the origin server's load determines when the object expires.

This type of expiration allows you to dynamically increase the time to live (TTL) of cached responses, if the current response time (average computed over a short time window) from the origin servers is larger than the average response time (average computed over a longer time window) by a threshold amount. Similarly, the TTL is dynamically decreased if the reverse holds true. The starting value for the cache TTL is the CacheMinTTL value, or 0 if not specified. The purpose of a moving average-based calculation is to allow the cache to respond to trends in usage patterns, smoothing out uncharacteristic spikes.

Performance assurance with load-based expiration is controlled by the keywords listed in [Table 5-5](#).

Table 5-5 *Keywords Configuring Load-Based Expiration*

Keyword	Description
ServerLoadShortWindow	Defines the time window over which the short-term response time is computed, expressed in seconds. For example, if the value is set to 300, then the short-term response time is computed as the average of all responses that occurred in the last 5 minutes. The default is 300 seconds (5 minutes).
ServerLoadLongWindow	Defines the time window over which the long-term response time is computed, expressed in seconds. For example, if the value is set to 604800, then the long-term response time is computed as the average of all responses that occurred in the last 7 days. The default is 604800 seconds (7 days).
ServerLoadTriggerPercent	Defines the threshold that triggers a change in the cache TTL. This feature enables the application appliance to monitor server load in real-time and make intelligent “closed loop” content expiration decisions so that site performance is maximized and existing hardware resources are used most efficiently, even during periods of peak traffic load. If the ServerLoadShortWindow value exceeds the ServerLoadLongWindow value by the ServerLoadTriggerPercent (20% by default), then the object TTL is increased by the CacheTTLChangePercent (20% by default). Similarly, if the ServerLoadShortWindow value is less than the ServerLoadLongWindow value by the ServerLoadTriggerPercent, then the object TTL is decreased by the CacheTTLChangePercent. The default ServerLoadTriggerPercent is 20%.

Table 5-5 Keywords Configuring Load-Based Expiration (continued)

Keyword	Description
CacheTTLChangePercent	Percentage by which the cache TTL is increased or decreased in response to a change in the server load. For example, if this is set to 20 and the current TTL for a particular response is 300 seconds, and if the current server response time exceeds the trigger threshold, then the cache TTL for the response is raised to 360 seconds (20% increase). The default is 20%.
MovingAverageCacheSize	Size of the shared memory, in KB, used to store the moving average objects. The default is 500 KB. Each moving average object occupies approximately 1000 bytes, so 500 KB should be able to accommodate 500 of these objects. One object contains up to two hours of moving average data.

Example of Cache Configuration

This section includes an example of an Application Class specification for dynamic caching.

```

1 <ApplicationClass PetStoreClass>
2   Url "(.*)/estore/(.*)$"
3   OptimizationPolicy DeltaOptimize,DeltaOptimizeAllUsers, FlashForward
4   CacheMinTTL 20
5   CacheMaxTTL 60
6   IF ($http_cookie_present(SESSID)) THEN
7     OptimizationPolicy CacheDynamic
8     CacheParameter $http_cookie(SESSID)
9     CanonicalUrl $(0)/$http_cookie(SESSID)
10  ELSEIF ($http_query_param_present(SESSID)) THEN
11    OptimizationPolicy CacheDynamic
12    CacheParameter $param(SESSID)
13    CanonicalUrl $(0)/$param(SESSID)
14  ENDIF
15 </ApplicationClass>

```

Here is a detailed description of each line:

[1] <ApplicationClass PetStoreClass>

Identifies that this section begins an Application Class definition and is tagged with the name `PetStoreClass`. The name has no significance and is used only for the benefit of the application appliance administrator's reference. The name must be unique with respect to all other Application Classes in the same `fgn.conf` file.

[2] Url "(.*)/estore/(.*)\$"

Defines the URL regular expression that is used to match this class. Each URL that matches this regular expression (and none encountered above this class) is handled only by this class. This regular expression has two groupings and matches any pattern that has any number of characters including white space before the literal `"/estore/"` followed by any characters until the end of the line.

[3] OptimizationPolicy DeltaOptimize, DeltaOptimizeAllUsers,FlashForward

The class global optimization policy, setting DeltaOptimize mode, Delta Optimize for all users (as opposed to per user), and FlashForward. For more information on policies, see the [“OptimizationPolicy Element” section on page 5-12](#).

[4] CacheMinTTL 20

Set up the minimum time-to-live for cached objects to 20 seconds.

```
[5] CacheMaxTTL 60
```

Set up the maximum time-to-live for cached objects to 60 seconds.

```
[6] IF ($http_cookie_present(SESSID)) THEN
```

Begin a set of conditional rules that override class global policies. This first IF statement checks for the presence of a cookie identified by the name “SESSID”. Note that the cookie name is case sensitive.

```
[7] OptimizationPolicy CacheDynamic
```

Enables dynamic caching for objects that match the conditional test in line 6.

```
[8] CacheParameter $http_cookie(SESSID)
```

Specifies that the cache key is to include the value of the “SESSID” cookie, to uniquely identify the object.

```
[9] CanonicalUrl $(0)/$http_cookie(SESSID)
```

Specifies that the base file name is to consist of the canonical URL, a forward slash (/) and the value of the “SESSID” cookie, to uniquely identify the object. Remember that \$(0) refers to the entire URL string.

```
[10] ELSEIF ($http_query_param_present(SESSID)) THEN
```

If the condition in line 6 fails, execution continues here. This statement checks for the presence of a query parameter named “SESSID”. Note that the parameter name is case sensitive.

```
[11] OptimizationPolicy CacheDynamic
```

```
[12] CacheParameter $param(SESSID)
```

```
[13] CanonicalUrl $(0)/$param(SESSID)
```

These statements perform the same functions as lines 7, 8, and 9. Except in this case, the additional token that is used to generate the cache key and identify the base file is the value of the “SESSID” query parameter (not the cookie, as before).

```
[14] ENDIF
```

A syntactical element that terminates the conditional block.

```
[15] </ApplicationClass>
```

Signifies the end of this Application Class definition.

Image Optimization Configuration

The ImageOptimization element in an Application Class controls how JPEG and PNG images are compressed by the application appliance. For an overview of Smart Image Optimization, see the [“Smart Image Optimization” section on page 2-8](#).

By default, image optimization is disabled in the application appliance. You must explicitly use the ImageOptimization element to enable it.

Image optimization is applied intelligently, and is not used for small images such as thumbnails, or when optimization reduces the file size by less than 10%. Nor is it used for images with many high frequency components that do not compress well.

There are two basic modes of image optimization: standard and advanced. In standard mode, the application appliance optimizes the image, smoothing it, if needed, to help reduce noise. To specify standard mode, use this configuration:

```
ImageOptimization standard
```

Use advanced mode to override the standard settings and control individual optimization options. In advanced mode, all options are disabled unless explicitly enabled. To specify advanced mode, use this configuration:

```
ImageOptimization advanced, [options]
```

For example:

```
ImageOptimization advanced,high,progressive
```

This example specifies that images be optimized less (high quality) and be converted to progressive mode.

The options available in advanced mode are listed in [Table 5-6](#).

Table 5-6 *Advanced Image Optimization Options*

Keyword	Description
grayscale	Transforms the image to a grayscale image.
high	Applies a higher quality (less compression) transformation to the image. This transformation yields images that are larger in size than those compressed without this option, but they have less visual deterioration. Image size is still smaller with this option than for uncompressed images.
IgnoreThumbnails	Causes small thumbnail images to be ignored (not transformed in any way). This option is enabled by default in standard mode.
progressive	Transforms the image to render progressively. This option is enabled by default in standard mode. This transformation yields a slightly larger image size, but it is progressively rendered by the browser. This is not useful in fast network environments such as LANs.
smooth	Applies a smoothing transformation to the image, if needed. This option is enabled by default in standard mode.

Images are processed in conjunction with the static cache; the processing cost of optimizing images makes this feature prohibitive for uncacheable images. The URL matching for the Application Class can include images other than JPEG and PNG safely; the optimization process is performed only if the JFIF or PNG header is detected in the image.

If an image or class of images is not well suited to optimization, you can use the standard Application Class mechanism to exclude images from optimization.

For example, on a Web site there may be a folder (“letterscans”) with a number of JPEGs that are created from scanning text documents. These don’t optimize well, nor do images that are essentially vector graphics. The built-in image check will in many cases not optimize these images (overriding any configuration settings), however, if this is not sufficient, a prior Application Class can be used to exclude a group of images.

The first Application Class that does not request image optimization for a group of objects, “catches” those objects and prevents them from being optimized by a subsequent class.

Here’s an example of two Application Classes, where the first prevents a group of images from being optimized by subsequent classes:

```
<ApplicationClass OptimizedImages_Override>
  Url "^.*/letterscans/.*\.jpg$"
  OptimizationPolicy NoDeltaOptimize,NoCompress, FlashForwardObject
</ApplicationClass>

# Images in the letterscans folder will not be
```

```
# optimized by the following class
<ApplicationClass OptimizedImages>
  Url "^.*\.(jpg)$"
  OptimizationPolicy NoDeltaOptimize,NoCompress, FlashForwardObject
  ImageOptimization standard
</ApplicationClass>
```

URL Mapping Configuration

URL mapping refers to the capability of the application appliance to alter URLs and other content in the data stream between the origin server and the client browser. URL mapping is configured in an Application Class by using the `Urlmap` element.

By default, URL mapping is disabled in the application appliance. You must explicitly use the `Urlmap` element to enable it.

A URL has the following format:

```
protocol://host[:port]/path
```

The `Urlmap` element can be used to alter any of these URL parts.

The format of the `Urlmap` element is as follows:

```
Urlmap scope scopeKeyword replacementDirective
```

Here is an example:

```
Urlmap scope html pattern (^.*)(https:)(//.*) to $urlmap_pattern(1)http:$urlmap_pattern(3)
```

The scope keyword defines where in the data stream URLs or other content will be altered; possible values are shown in [Table 5-7](#).

Table 5-7 *scopeKeyword Values*

scopeKeyword	Description
all	Alters URLs anywhere they are found (default)
content	Alters any content (not just URLs), based on a pattern
html	Alters URLs only within the URL attribute of META HTTP-EQUIV tags and within the SRC attribute of the following HTML tags: BASE, HREF, IMG, LINK, SCRIPT, and STYLE
header	Alters URLs only within the Location response-header field (such as in a response with a 302 status code)
cookie	Alters the domain section of cookies

The replacement directive specifies how a URL is to be altered; possible values are shown in [Table 5-8](#).

Table 5-8 *replacementDirective Values*

replacementDirective	Description
HOST <i>src</i> TO <i>dst</i>	Alters the host portion of URLs whose host portion is <i>src</i> so that the host portion becomes the string <i>dst</i>
PORT <i>src</i> TO <i>dst</i>	Alters the port portion of URLs whose port is <i>src</i> so that the port becomes the string <i>dst</i>

Table 5-8 replacementDirective Values (continued)

replacementDirective	Description
PROTOCOL HTTP(S) TO HTTP(S)	Alters the URL protocol from HTTP to HTTPS or from HTTPS to HTTP
PATTERN <i>src</i> TO <i>dst</i>	Alters any portion of the input stream. The pattern <i>src</i> is a regular expression that defines sub-expressions within the input stream that are to be altered. The <i>dst</i> string specifies the replacement pattern. For details, see “Using the PATTERN Directive” .

Using the PATTERN Directive

The PATTERN directive has the syntax: PATTERN *src* TO *dst*

The *src* part uses the regular expression syntax to define sub-expressions within the input stream that are to be altered. Sub-expressions are delimited using parentheses (). The numbering of the sub-expressions begins with 1 and is the number of the left-parenthesis (“(“ counting from the left. For example, the following pattern defines three sub-expressions in the input stream:

```
(^.*)(fast)(.*$)
```

The first sub-expression is everything before the word “fast.” The second sub-expression is the word “fast”. The third sub-expression is everything after the word “fast”.

The *dst* string specifies the replacement pattern; the complete string defined by *src* is replaced by the string defined by *dst*. In the *dst* string, you can use any of the parameter expander functions listed in [Table 5-4 on page 5-16](#), or one or more `$urlmap_pattern(number)` variables, which refer back to specific sub-expressions in the original input stream.

`$urlmap_pattern(0)` matches the entire input stream, `$urlmap_pattern(1)` matches the first sub-expression, `$urlmap_pattern(2)` matches the second sub-expression, and so on.

If the specified sub-expression does not exist in the input stream, then the variable evaluates to the empty string.



Note

The parameter expander functions listed in [Table 5-4](#) apply only to the context of the HTTP request from the client, not to the data stream being sent as a response to the client. For example, the function `$http_query_string()` evaluates to the query string of the request URL, as expected.

For example, given this Urlmap element:

```
urlmap scope content pattern (^.*)(fast)(.*$) to
$urlmap_pattern(1)faster$urlmap_pattern(3)
```

and this input stream:

```
Cisco makes accessing the web fast now.
```

The result will be this stream:

```
Cisco makes accessing the web faster now.
```

Examples

The following example changes the string “logos/dna.gif” to “http://www.google.com/logos/dna.gif” in the content.

```
<ApplicationClass google>
  Url "^.*google.com/index.html$"
  Policy NoCondense,NoCompress,FlashForward
  urlmap scope content pattern (^.*)(logos/dna.gif)(.*) to
  $urlmap_pattern(1)http://www.google.com/logos/dna.gif$urlmap_pattern(3)
</ApplicationClass>
```

The following example changes the protocol of all URLs in the input stream from HTTPS to HTTP.

```
<ApplicationClass t1a>
  Url "^.*t1a\.asp$"
  Policy NoCondense,NoCompress,CondenseAllUsers, NoFlashForward
  urlmap scope all protocol https to http
</ApplicationClass>
```

The following example changes the port of all URLs in the input stream from 911 to 80.

```
<ApplicationClass>
  Url "^.*t1b\.asp$"
  Policy NoCondense,NoCompress,CondenseAllUsers, NoFlashForward
  urlmap scope all port 911 to 80
</ApplicationClass>
```

The following example shows how this feature is used to change arbitrary HTML tags in the content. This example changes each H1 tag in the content to a B tag.

```
<ApplicationClass H1toB>
  Url "^.*t4\.asp$"
  Policy NoCondense,NoCompress,CondenseAllUsers, NoFlashForward
  urlmap scope content pattern (^.*)(\<H1\>)(\>.*) to
  $urlmap_pattern(1)\<B\>$urlmap_pattern(3)
</ApplicationClass>
```

Client View Logging Configuration

Client view logging refers to the capability of the application appliance to log the true IP address of clients.

Often, the application appliance is deployed behind various network devices such as SSL terminators, load balancers, proxies, etc. When those devices terminate the client TCP session, the source IP from those devices to the application appliance is typically altered to that of those devices. AppScope reports then do not reflect the true geographic distribution of incoming requests.

Client view logging is configured in an Application Class by using the LogClientView element. By default, Client view logging is disabled in the application appliance and you must explicitly use the LogClientView element to enable it.

Client view logging works by identifying a source from which the client IP address can be obtained. This can be an HTTP request header, an HTTP cookie, or a URL query parameter. The corresponding parameter expander functions (in [Table 5-4 on page 5-16](#)) are used to specify the client IP address source.

The syntax of the LogClientView element is as follows:

```
LogClientView parameter_expander_expression
```

The expression must evaluate to a well-formed IP address, or a list of them separated by commas. The first IP address in the list is recorded as the client IP address in the FgnStatLog file. If the expression does not evaluate to an IP address, the source socket IP is logged as the IP address, as usual.

The LogClientView element can be defined in a hierarchy along with the Application Class. If the child class doesn't have LogClientView specified, it inherits the LogClientView specified in the parent class. If the child class has LogClientView specified, it overrides the LogClientView specified in the parent class.

The client IP address of a request, as determined by the LogClientView element, is reported in the FgnStatLog by the <CLO><CIP> entry (see [Table A-1 on page A-3](#)).

Examples

Here are some examples of LogClientView elements that derive the client IP address from different sources.

This first example looks for the client IP address in the HTTP request header true-clinet-ip:

```
LogClientView $http_header(true-clinet-ip)
```

The following example looks for the client IP address in the URL query parameter true-clinet-ip:

```
LogClientView $http_query_param(true-client-ip)
```

The following example looks for the client IP address in the cookie true-clinet-ip:

```
LogClientView $http_cookie(true-client-ip)
```

It is possible that the client view can be derived from multiple places in a request. For example, the true client IP can sometimes be from a cookie, sometimes from a header, and sometimes from a parameter. You can use multiple parameter expander functions in one LogClientView expression, or you can use multiple LogClientView elements in one Application Class.

The evaluation order is the same as the order of the parameter expander functions. During the evaluation process, the first expression encountered with a value is used. If there are multiple values associated with a parameter expander function, the leftmost value is used. If the value is not valid, other values or parameter expander functions are not scanned for consideration.

Here's an example of a single LogClientView expression that includes all three types of parameter expander functions:

```
LogClientView
$http_header(true-clinet-ip)$http_query_param(true-client-ip)$http_cookie(true-client-ip)
```

Here's a similar configuration that uses three separate statements to do the same thing:

```
LogClientView $http_header(true-clinet-ip)
LogClientView $http_query_param(true-client-ip)
LogClientView $http_cookie(true-client-ip)
```

Given the configuration above, and this request:

```
GET /?true-client-ip=10.0.0.1 HTTP/1.1
true-client-ip: 192.168.0.1
Cookie: true-client-ip="209.140.0.1"
```

The application appliance will log the IP 192.168.0.1 as the true client IP address because the \$http_header expression is evaluated and satisfied first.

XML/XSL Transformations

The application appliance can apply an XSL stylesheet transformation to an XML source document and return the resulting HTML document to the requestor. The application appliance applies other optimizations after the XML is transformed, but before the result is returned to the requestor.

Specify the ApplicationClass keyword `XsltMerge On` to enable XML document transformation.

Here is an example of an XML source document and an associated XSL stylesheet, and the HTML document that is returned by the application appliance when the XSL stylesheet is applied to transform the XML document.

Here is the example XML document, `foo.xml`:

```
<?xml version="1.0"?>
<doc>Hello</doc>
```

Here is the corresponding XSL stylesheet, `foo.xsl`:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="doc">
<out><xsl:value-of select="."/></out>
</xsl:template>
</xsl:stylesheet>
```

The output would be this HTML file, `foo.html`:

```
<out>Hello</out>
```

Destination Mapping Configuration

This section of the configuration file defines where requests will be sent to. For an overview of this feature, see the [“Destination Mapping” section on page 2-16](#).

Here’s an example of how to specify destination mapping in the `fgn.conf` file:

```
<DestinationMapping>
  Dest 10.0.3.88:8080 -> 192.168.3.88:80 HostHeader=ws1.domainname.com
  Dest default:9999 -> 192.168.0.1:3128 Proxy

  Name www.domainname1.com:9080 -> ws1.domainname.com:80
  Name www.domainname2.com -> 192.168.3.88:80 HostHeader=domain2.com
  Name 10.0.3.88:9080 -> 192.168.3.88
  Name default -> 192.168.0.1:3128 PROXY
</DestinationMapping>
```

Each rule in the `DestinationMapping` block is specified on a single line and has the following format:

```
Type map-from -> map-to [PROXY] [HostHeader=str] [Protocol=prot]
```

where:

- *Type* is either `Dest` or `Name`. `Dest` means that the *map-from* field is based on the IP packet address. `Name` means that the *map-from* field is based on the URL. Note that a port in the *map-from* field for a `Name` rule is based on URL host name or host header, not on the IP packet.
- *map-from* is the IP address or URL host name that is mapped to a different location. It can be one of the following kinds of values:
 - *hostname* or *hostname:port*, if *Type* is `Name`. *hostname* is a URL host name and *port* is a port number.

- *IPaddress* or *IPaddress:port*, if *Type* is Dest. *IPaddress* is an IP address and *port* is a port number.
- default or default:*port*, valid for both Types of rules. *port* is a port number. The default keyword represents all *map-from* locations other than those previously specified in the rules block. If default without a specific port is needed, it must be the last rule; any subsequent rules are ignored. Multiple default rules specific to different ports are valid.

If port is not specified in the *map-from* value, it means map all ports.

- *map-to* is the symbolic or numeric address that *map-from* is mapped to. It can be one of the following kinds of values:
 - *hostname* or *hostname:port*, where *hostname* is a URL host name and *port* is a port number.
 - *IPaddress* or *IPaddress:port*, where *IPaddress* is an IP address and *port* is a port number.
 - If *Type* is Name and the *map-from* value is default, then you can specify the value default for *map-to* also. This directs the connection to the host whose name is the same as what the Host header specifies. This rule is useful if an internal DNS server alters the DNS resolution.

An example of this rule is:

```
Name default -> default Protocol=ClientProtocol
```

If port is not specified in the *map-to* value, it means to use the port specified in the *map-from* value.

- PROXY is an optional keyword that indicates the mapping is a proxy request. It ensures that the application appliance sends the absolute URI to the mapped proxy server.
- HostHeader=*str* optionally directs the application appliance to replace the host header value with the value of the string specified by *str*. It has no impact on which server the application appliance connects to.
- Protocol=*prot* optionally directs the application appliance to connect to the backend (origin) server by using the protocol specified by *prot*. Valid protocol values include:
 - HTTP: Use the HTTP protocol. This is the default.
 - HTTPS: Use the HTTPS protocol. Specify this for SSL proxy mode.
 - ClientProtocol: Use the same protocol as specified by the client request. This also uses SSL proxy mode, but only if the client request uses HTTPS.

If two rules overlap, the first one listed takes precedence.

The rules are not case sensitive.

Destination Mapping Examples

This section provides some typical examples that show how to specify destination mapping rules.

Example 1

In this example, the application appliance handles both an intranet application on domain name testIntranet.enterprise.com (the origin server runs at 10.0.0.5 on port 80) and an Internet application. The Internet application has to be accessed through a corporate proxy server 10.0.0.1 on port 3128.

```
<DestinationMapping>
  Name testIntranet.enterprise.com -> 10.0.0.5:80
  Name default -> 10.0.0.1:3128 PROXY
</DestinationMapping>
```

Because the application appliance IP address is not required in the rules, this mapping structure stays the same across all the application appliances in the cluster, making the configuration and maintenance easy.

Example 2

In this example, the application appliance handles multiple internal sites, `site1.enterprise.com`, `site2.enterprise.com`, `site3.enterprise.com`, and `site4.enterprise.com` running on 10.0.0.5, 10.0.0.6, 10.0.0.7, and 10.0.0.8, respectively. All are on port 80.

```
<DestinationMapping>
  Name site1.enterprise.com -> 10.0.0.5:80
  Name site2.enterprise.com -> 10.0.0.6:80
  Name site3.enterprise.com -> 10.0.0.7:80
  Name site4.enterprise.com -> 10.0.0.8:80
</DestinationMapping>
```

Because the application appliance IP address is not required in the rules, there is no need to create multiple listening ports in order to deal with multiple sites.

Example 3

In this example, the application appliance (10.0.3.88) is behind a load balancer with a VIP to handle a site (`www.enterprise.com`), but this site has to be accessed through a proxy (`proxy.enterprise.com` on port 3128). HTTP 1.0 clients have to be supported as well.

```
<DestinationMapping>
  Name 10.0.3.88 -> proxy.enterprise.com:3128 PROXY HostHeader=www.enterprise.com
  Name default -> proxy.enterprise.com:3128 PROXY
</DestinationMapping>
```

Example 4

In this example, an SSL Terminator forwards decrypted requests to the application appliance (10.0.0.5 on port 8080).

```
<DestinationMapping>
  Dest default:8443 -> 10.0.0.5:8080
</DestinationMapping>
```

This rule doesn't rely on the application appliance IP address. You can apply it to every SSL Terminator in the cluster.

Example 5

In this example, the application appliance handles a long list of content sites. The load balancer groups and redirects them into four different ports (8081, 8082, 8083, 8084). Accessing the four groups needs to go through four proxy servers (`proxy1.enterprise.com`, `proxy2.enterprise.com`, `proxy3.enterprise.com`, and `proxy4.enterprise.com`, all on port 3128).

```
<DestinationMapping>
  Dest default:8081 -> proxy1.enterprise.com:3128 PROXY
  Dest default:8082 -> proxy2.enterprise.com:3128 PROXY
  Dest default:8083 -> proxy3.enterprise.com:3128 PROXY
  Dest default:8084 -> proxy4.enterprise.com:3128 PROXY
</DestinationMapping>
```

Again, there is no application appliance IP address in the rules, making the configuration the same across all the nodes in the cluster.

Example 6

In this example, the application appliance transparently supports the web protocol (HTTP or HTTPS) that the client specifies. The external DNS server resolves the hostname to the application appliance, and the application appliance uses the internal DNS server to resolve the same hostname into the origin web server address.

```
<DestinationMapping>
  Name default->default Protocol=ClientProtocol
</DestinationMapping>
```

SSL Configuration

You can configure the application appliance purely as an SSL terminator, with no condensation. To do this, specify a DestinationMapping block that configures the application appliance to proxy for the SSL destination address(es) being terminated. Also set an OptimizationPolicy element to prevent condensation.

Here is an example of this kind of configuration:

```
<DestinationMapping>
  Dest default:8443 -> 10.0.0.2:8080 PROXY
</DestinationMapping>

<ApplicationClass DefaultByPass>
  Url ".*$"
  OptimizationPolicy NoDeltaOptimize, NoCompress
</ApplicationClass>
```

To configure the application appliance to provide condensation in addition to SSL termination, use an Application class that allows condensation.

You typically configure the application appliance for SSL proxying by using the following destination mapping rule:

```
<DestinationMapping>
  Name hostname -> originIP Protocol=https
</DestinationMapping>
```

If you want to support both HTTP and HTTPS at the back end, depending on the original client request protocol, you can do this:

```
<DestinationMapping>
  Name hostname:80 -> originIP Protocol=http
  Name hostname:443 -> originIP Protocol=https
</DestinationMapping>
```

For the above scenario, if you know the incoming request host header is always *hostname*, then a more convenient way to configure it is like this:

```
<DestinationMapping>
  Name default -> originIP Protocol=ClientProtocol
</DestinationMapping>
```

If you have an internal DNS server that can resolve the *hostname* into *originIP*, you can also do this:

```
<DestinationMapping>
  Name default -> default Protocol=ClientProtocol
</DestinationMapping>
```

This is particularly useful in a test environment where you can add an entry “*hostname applicationApplianceIP*” into your client hosts file. Then your client machine will resolve the *hostname* into the *applicationApplianceIP*, making the request go to the application appliance. But the *hostname* will be resolved at the application appliance into the actual origin server virtual IP address.

For details on defining the DestinationMapping block, see the “[Destination Mapping Configuration](#)” section on page 5-30. For details on defining the Application Class, see the “[Application Class Specification](#)” section on page 5-7.

Dynamic Caching Configuration Guide

Before deploying the application appliance, we recommend that you analyze traffic patterns to determine the suitability of the various features offered by the application appliance (FlashForward, Adaptive Dynamic Caching, Image Optimization, Compression, and so on). There are many ways to determine the optimal configuration settings in an fgn.conf configuration file, however, the following guidelines provide a framework to guide the process of configuring dynamic caching.

- [When is Dynamic Caching Appropriate?](#), page 5-34
- [When is Dynamic Caching Inappropriate?](#), page 5-35
- [Configuring Dynamic Caching](#), page 5-35

For reference information on configuring dynamic caching in the fgn.conf file, see the “[Adaptive Caching Configuration](#)” section on page 5-18.

When is Dynamic Caching Appropriate?

Dynamic caching is appropriate in the following cases:

- Dynamic caching is useful only if the response can be reused over some period of time, however short.
- Dynamic caching is useful only if a cached response can be used multiple times before the response expires. This can happen in one of the following ways:
 - The same response is usable by multiple users within a certain period of time: that is, the content is sharable. This is the ideal situation. It can be useful even if the cache expiration period is very small, as long as enough users access the response within that period. An example of such a situation would be a hierarchical directory browser application (for example, a catalog store or a document management application) where the directory is generated dynamically, but is the same view seen by all users. Dynamic caching can avoid the cost of regenerating the directory pages for each user.
 - The response is usable by only a single user (it is personalized), but the user accesses it multiple times. This can happen in one of the following ways:
 - The reuse of the response occurs across multiple browser sessions for that user. For example, the response generated in one session can be used unchanged in a second session.
 - The reuse of the response can occur only within a single browser session because the response is tied to that particular session.
- Dynamically cacheable responses are identifiable using one or more of the following:
 - The URL matches a regular expression.
 - The presence or absence of specific query parameters.
 - The presence or absence of specific cookies in the request.

- The presence or absence of specific HTTP headers in the request.

For example, a particular application uses the same URL for various actions, only some of which are cacheable. Consider the case where the URLs in the application are:

1. `http://xyz.com/doiit.jsp?action=login&username=xyz`
2. `http://xyz.com/doiit.jsp?action=browse&level=1`
3. `http://xyz.com/doiit.jsp?action=browser&level=2`

All the URLs are the same, however, the response to (1) cannot be cached, but assume the response to (2) and (3) can be cached. In this case, dynamic caching is achievable by testing for the presence or absence of the “username” query parameter by using an IF/ELSE/ENDIF conditional block.

When is Dynamic Caching Inappropriate?

Dynamic caching is not appropriate in the following cases:

- The response becomes stale immediately upon delivery. Examples of this include:
 - The response sets cookies specific to that session. For example, the response to a login page is specific to a particular session.
 - The response contains data specific to a previous action in the session. For example, a confirmation number for a transaction that was just executed is not cacheable.
- The life of a response is indeterminate; that is, the response contains data that becomes stale based on a future action. For example, the portfolio page of a brokerage account user changes when the user executes transactions.
- Different versions of the response cannot be distinguished using the URL, the URL query parameters, or the cookies in the request. For example, the response contains some personalized settings such as the name of the user, however, there is no URL query parameter or cookie to directly identify the user. Note that there is something in the request that indirectly identifies the user, in order for the origin server to generate the personalized response, for example, a session cookie. You could dynamically cache a version of the page that is different for each user session. However in this case, dynamic caching is useful only if the user accesses the page multiple times within a single browser session.

Configuring Dynamic Caching

Once the appropriateness of dynamic caching has been established for a page, you must configure it in the application appliance. Follow the general steps outlined in this section.

1. Identify the URL or URLs for an Application Class. This involves developing the regular expression to match the URL(s).
 - Run the performance test using Accelerometer or another performance testing program on the LAN. The LAN is used so that the focus is on identifying server latency problems, not network latency problems.
 - Examine the error log to look at entries where origin server response time is large. Large means that it is a significant part of the page download time as measured by the end user. For example, if a page download on the LAN takes 10 seconds, then any individual component download with an origin server response time of greater than one second is worth examining.
 - This should form the short list of URLs to target.
2. Identify the dynamically cacheable instances of the URL. This should include the following:

- The regular expression matching the URL.
 - In case the regular expression matches many responses, some of which can be dynamically cached, and some which cannot be, then can the cacheable responses be separated from the uncacheable ones by testing for the presence or absence of specific query parameters, cookies, or HTTP headers? If so, an IF/ELSE/ENDIF conditional block can be used to set up different policies for the different responses (see the [“Using Conditional Blocks”](#) section on page 5-18).
3. Identify the versions of the response. Once the dynamically cacheable instances are identified, then the next step is to figure out how many different versions of the response are to be cached, using Cache Parameterization. This involves looking at the request parameters and the response content to determine what defines the content of the response. Examine these request parameters:
- URL and its components
 - query parameters
 - cookies
 - HTTP header values

Next, develop the minimal set of parameters that uniquely determines the content of the response. Based on this, add [CacheParameter](#) and [CacheKeyModifier](#) settings to the Application Class for this URL. For example:

- The response is determined uniquely by the URL and all its parameters. In this case, nothing needs to be done since that is the default cache key.
- The response is determined by only some of the parameters. In this case, use [CacheParameter](#) to specify those parameters. For example, a site might have a URL like: `http://xyz.com/dosomething.asp?action=browse&dir=x&session=13345`. The contents of the response might be solely determined by the “action” and “dir” query parameters, and the “session” query parameter has no bearing on the response. In that case, use:

```
CacheParameter $http_query_param(action)
CacheParameter $http_query_param(dir)
```

- The response is dependent on the value of some cookie(s). Add them to the [CacheParameter](#) list, for example:

```
CacheParameter $http_cookie(foo)
```

- Some parts of the URL need to be ignored. For example, some sites embed the session id in the URL itself, but the response is not dependent on the session id. Then URL subexpressions can be used in the [CacheKeyModifier](#). For example, say a site has a URL of the form: `http://xyz.com/sess12345/dosomething.asp?action=browse&dir=x`. If “sess12345” has no bearing on the response, then specify the URL-matching regular expression and [CacheKeyModifier](#) as follows:

```
Url "^(.*)/(sess.*)/(dosomething.asp)$"
CacheKeyModifier ${1}/${3}
```

Doing this eliminates the (sessNNNN) string from the cache key.

4. Identify how long the response can be cached. This involves considering the nature of the content, for example, a new item may be viewed for three hours, after which it becomes stale.

There are two main timing related parameters, the [CacheMinTTL](#) and [CacheMaxTTL](#). The minimum time-to-live ([CacheMinTTL](#)) is the minimum time that the content can be cached for, which corresponds to the live-time of the content. In the case of a new item that is valid for three

hours, this value would be $3*60*60 = 10800$ seconds. The maximum time-to-live (CacheMaxTTL) is used to determine how the application appliance handles the case when the object has passed its CacheMinTTL value.

You can also configure object expiration using performance assurance with load-based expiration. This allows you to dynamically increase the time-to-live of cached responses, if the current response time (average computed over a short time window) from the origin servers is larger than the average response time (average computed over a longer time window) by a threshold amount. Similarly, the TTL is dynamically decreased if the reverse holds true.

For more details on configuring cache expiration, see the [“Configuring Cache Object Expiration” section on page 5-22](#).

httpd.conf

This file is the standard Apache web server configuration file, slightly modified for the application appliance. This file includes the fgn.conf configuration file by reference.

You may want to modify the following entries:

- Port - Set this to the port on which you want to allow access to the application appliance server for HTTP requests.
- AdminPort - Set this to the port on which you want to allow access to the application appliance management functions from the Management Console.
- Listen - Set this to each port on which you want the application appliance to listen for requests. There can be multiple Listen entries for HTTP, HTTPS, and management function requests.
- User - Change this entry to the user under which the server should be run. This is set to nobody by default.
- Group - Change this entry to the group under which the server should be run. This is set to nobody by default.
- ServerAdmin - Change this to the email address of the server administrator, for error reporting.
- MaxClients - Sets the maximum number of concurrent clients (active TCP/IP sessions) that the application appliance can support. This is set to 500 by default.
- SSL entries, including Listen and VirtualHost - Change these entries to enable the application appliance to handle SSL connections. For details, see the section [“SSL Configuration Entries”](#).
- LogLevel - Used this entry to enable a debugging level of logging. For details, see the section [“Logging Level”](#).
- FgnLogFormat and CustomLog - Use these entries to define and use an extended access_log format. For details, see [Appendix A, “Logs.”](#)
- CoreDumpDirectory - Use this entry to change where core files are located.

SSL Configuration Entries

To configure the application appliance to handle SSL connections, you must set Listen and VirtualHost configuration entries.

Listen is defined as follows:

```
<IfDefine SSL>
Listen 8443
```

```
</IfDefine>
```

Set the port to listen to for SSL connections.

The VirtualHost block is defined as follows:

```
<VirtualHost _default_:443>
SSLCertificateFile file-path
SSLCertificateKeyFile file-path
.....
</VirtualHost >
```

Set the SSLCertificateFile entry to a PEM encoded certificate. If the certificate is encrypted, you will be prompted for a pass phrase.

Set the SSLCertificateKeyFile entry to the key file, if it is not combined with the certificate.

The application appliance can support multiple certificates through multiple virtual hosts. Each virtual host supports a single certificate.


Note

The AVS software supports the Rainbow SSL hardware accelerator card (if you upgraded the older Performance Suite software and it is running on your own hardware). Use the manufacturer's install procedures to install the card and related drivers. To enable the AVS software to use the accelerator card, in the SSL Global Context section, add the following directive: SSLCryptoDevice cswift

Logging Level

Normally, you will not need to change the LogLevel configuration entry, which controls the logging level employed by the application appliance. The normal setting of this directive is as follows:

```
LogLevel error
```

To configure the application appliance so that statistical log entries are written both to the error_log file and the FgnStatLog file, set LogLevel to debug. Normally, statistical log entries are written only the FgnStatLog file, as described in the [“Log File Management” section on page A-1](#).

Do not use the LogLevel debug directive on a production system, as this will slow performance and generate much unnecessary log information.

mimetypes.conf

This file is used to configure MIME type support for the application appliance. It contains a list of MIME types that are excluded from condensation and/or compression.

Whenever the application appliance receives data from the origin server, it checks the MIME type of the data against the entries in this file. If the MIME type of the data matches one of the MIME types listed in this file, the data is not condensed if the directive is NoDeltaOptimizeMimeType, and it is not compressed if the directive is NoCompressMimeType.

Here is an example of specifying a MIME type that is not to be condensed:

```
NoDeltaOptimizeMimeType application/pdf
```

Here is an example of specifying a MIME type that is not to be compressed:

```
NoCompressMimeType application/x-javascript
```

You can use regular expressions in the `mimetypes.conf` file to specify MIME types to be excluded from condensation.

**Note**

The application appliance requires GNU POSIX regular expression syntax. For more information, see [Appendix F, “Regular Expressions.”](#)

For more details about MIME type exclusion, see the [“MIME-Type Exclusion” section on page 2-16](#).

useragent.conf

This file is used to configure user agent (browser) support for Delta Optimization. It contains a list of user agent identifiers for user agents that support Delta Optimization. The user agent identifiers are strings that correspond to the HTTP User-Agent header returned by browsers.

If the application appliance receives a request from a browser whose user-agent string is not listed in this file, then content will not receive Delta Optimization for that request, but will receive all other appropriate optimizations.

You can use regular expressions (using the GNU POSIX syntax) in the `useragent.conf` file to specify user agents that should explicitly receive Delta Optimization. For more information, see [Appendix F, “Regular Expressions.”](#)

**Tip**

You should only list user agents that you know are supported for Delta Optimization. For details on supported browsers, see the [“Web Browser Support” section on page 2-3](#).

fgnsnmpd.conf

This file is used to configure SNMP support. The most important configuration directive in this file is the `Port` directive:

```
Port 8080
```

If you change the default port that the application appliance listens to for HTTP requests, you must also change this `Port` directive to the same port.

For additional SNMP information, see [Appendix B, “SNMP MIB.”](#)

magentd.conf

This file is used to configure the community string for sending SNMP traps. The default directive is:

```
COMMUNITY      public
                ALLOW ALL OPERATIONS
                USE NO ENCRYPTION
```

To specify a different community string, change “public” to a different string.

