Google Search Appliance

Feeds Protocol Developer's Guide

Google Search Appliance software version 7.2

Google, Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043
www.google.com

GSA-FEEDS_200.03
March 2015

Contents

Feeds Protocol Developer's Guide

This document is for developers who use the Google Search Appliance Feeds Protocol to develop custom feed clients that push content and metadata to the search appliance for processing, indexing, and serving as search results.

To push content to the search appliance, you need a feed and a feed client:

- The *feed* is an XML document that tells the search appliance about the contents that you want to push.
- The *feed client* is the application or web page that pushes the feed to a feeder process on the search appliance.

This document explains how feeds work and shows you how to write a basic feed client.

Overview

You can use feeds to push data into the index on the search appliance. There are two types of feeds:

- A *web feed* provides the search appliance with a list of URLs. A web feed:
  - Must be named "web", or have its feed type set to "metadata-and-url".
  - May include metadata, if the feed type is set to "metadata-and-url".
  - Does not provide content. Instead, the crawler queues the URLs and fetches the contents from each document listed in the feed.
  - Is incremental.
  - Is recrawled periodically, based on the crawl settings for your search appliance.
- A *content feed* provides the search appliance with both URLs and their content. A content feed:
  - Can have any name except "web".
  - Provides content for each URL.
  - May include metadata.
  - Can be either full or incremental.
  - Is only indexed when the feed is received; the content and metadata are analyzed and added to the index. The URLs submitted in a content feed are not crawled by the search appliance. Any URLs extracted from the content, that have not been submitted in a content feed, will be extracted and scheduled for crawling if they match the crawling rules.

The search appliance does not support indexing compressed files sent in content feeds.

The search appliance follows links from a content-fed document, as long as the links match URL patterns added under **Follow and Crawl Only URLs with the Following Patterns** on the **Content Sources > Web Crawl > Start and Block URLs** page in the Admin Console.

Web feeds and content feeds behave differently when deleting content. See "Removing Feed Content From the Index" on page 31 for a description of how content is deleted from each type of feed.

To see an example of a feed, follow the steps in the section "Quickstart" on page 7.

Why Use Feeds?

You should design a feed to ensure that your search appliance crawls any documents that require special handling. Consider whether your site includes content that cannot be found through links on crawled web pages, or content that is most useful when it is crawled at a specific time. For example, you might use a feed to add external metadata from an Enterprise Content Management (ECM) system.

Examples of documents that are best pushed using feeds include:

- Documents that cannot be fetched using the crawler. For example, records in a database or files on a system that is not web-enabled.
- Documents that can be crawled but are best recrawled at different times than those set by the automatic crawl scheduler that runs on the search appliance.
- Documents that can be crawled but there are no links on your web site that allow the crawler to discover them during a new crawl.
- Documents that can be crawled but are much more quickly uploaded using feeds, due to web server or network problems.

Impact of Feeds on Document Relevancy

For documents sent with content feed, a flat fixed page rank value is assigned by default, which might have a negative impact on the relevancy determination of the documents. However, you can specify PageRank in a feed for either a single URL or group of URLs by using the `pagerank` element. For more details, see "Defining the XML Record for a Document" on page 9.

Choosing a Feed Client

You push the XML to the search appliance using a feed client. You can use one of the feed clients described in this document or write your own. For details, see "Pushing a Feed to the Google Search Appliance" on page 27.

Quickstart

Here are steps for pushing a content feed to the search appliance.

1. Download sample_feed.xml to your local computer. This is a content feed for a document entitled "Fed Document".

2. In the Admin Console, go to **Content Sources > Web Crawl > Start and Block URLs** and add this pattern to "Follow and Crawl Only URLs with the Following Patterns":

   ```
   http://www.localhost.example.com/
   ```

   This is the URL for the document defined in sample_feed.xml.

3. Download pushfeed_client.py to your local computer. This is a feed client script implemented in Python 2.x. You must install Python 2.x to run this script. Google also provides a Python 3.x version, pushfeed_client3.py.

4. Configure the search appliance to accept feeds from your computer. In the **Admin Console**, go to **Content Sources > Feeds**, and scroll down to **List of Trusted IP Addresses**. Verify that the IP address of your local computer is trusted.

5. Run the feed client script with the following arguments (you must change "APPLIANCE-HOSTNAME" to the hostname or IP address of your search appliance):

   ```
   % pushfeed_client.py --datasource="sample" --feedtype="full"
     --url="http://<APPLIANCE-HOSTNAME>:19900/xmlfeed" --
   xmlfilename="sample_feed.xml"
   ```

6. In the Admin Console, go to **Content Sources > Feeds**. A data source named "sample" should appear within 5 minutes.

7. The URL `http://www.localhost.example.com/` should appear under Crawl Diagnostics within about 15 minutes.

8. Enter the following as your search query to see the URL in the results:

   ```
   info:http://www.localhost.example.com/
   ```

   If your system is not busy, the URL should appear in your search results within 30 minutes.

Designing an XML Feed

The feed is an XML file that contains the URLs. It may also contain their contents, metadata, and additional information such as the last-modified date. The XML must conform to the schema defined by gsafeed.dtd. This file is available on your search appliance at `http://<APPLIANCE-HOSTNAME>:7800/gsafeed.dtd`. Although the Document Type Definition (DTD) defines elements for the data source name and the feed type, these elements are populated when you push the feed to the search appliance. Any `datasource` or `feedtype` values that you specify within the XML document are ignored.

An XML feed must be less than 1 GB in size. If your feed is larger than 1 GB, consider breaking the feed into smaller feeds that can be pushed more efficiently.

Choosing a Name for the Feed Data Source

When you push a feed to the search appliance, the system associates the fed URLs with a data source name, specified by the `datasource` element in the feed DTD.

- If the data source name is "web", the system treats the feed as a web feed. A search appliance can only have one data source called "web".
- If the data source name is anything else, and the feed type is `metadata-and-url`, the system treats the feed as a web feed.
- If the data source name is anything else, and the feed type is not `metadata-and-url`, the system treats the feed as a content feed.

To view all of the feeds for your search appliance, log into the Admin Console and choose **Content Sources > Feeds**. The list shows the date of the most recent push for each data source name, along with whether the feed was successful and how many documents were pushed.

Note: Although you can specify the feed type and data source in the XML file, the values specified in the XML file are currently unused. Instead, the search appliance uses the data source and feed type that are specified during the feed upload step. However, we recommend that you include the data source name and feed type in the XML file for compatibility with future versions.

Choosing the Feed Type

The *feed type* determines how the search appliance handles URLs when a new content feed is pushed with an existing data source name.

Content feeds can be *full* or *incremental*; a web feed is always incremental. To support feeds that provide only URLs and metadata, you can also set the feed type to *metadata-and-url*. This is a special feed type that is treated as a web feed.

- When the `feedtype` element is set to `full` for a content feed, the system deletes all the prior URLs that were associated with the data source. The new feed contents completely replace the prior feed contents. If the feed contains metadata, you must also provide content for each record; a full feed cannot push metadata alone. You can delete all documents in a data source by pushing an empty full feed.
- When the `feedtype` element is set to `incremental`, the system modifies the URLs that exist in the new feed as specified by the `action` attribute for the record. URLs from previous feeds remain associated with the content data source. If the record contains metadata, you can incrementally update either the content or the metadata.
- When the `feedtype` element is set to `metadata-and-url`, the system modifies the URLs and metadata that exist in the new feed as specified by the `action` attribute for the record. URLs and metadata from previous feeds remain associated with the content data source. You can use this feed type even if you do not define any metadata in the feed. The system treats any data source with this feed type as a special kind of web feed and updates the feed incrementally. Unless the `metadata-and-url` feed has the `crawl-immediately=true` directive the search appliance will schedule the re-crawling of the URL instead of re-crawling it without delay.

It is not possible to modify a single field of a document's metadata by submitting a feed that contains only the modified field. To modify a single field, you must submit a feed that includes all the metadata fields along with the modified field.

Documents that have been fed by using content feeds are specially marked so that the crawler will not attempt to crawl them unless the URL is also one of the Start URLs defined on the **Content Sources > Web Crawl > Start and Block URLs** page. In this case, the URL is periodically accessed from the GSA as part of the regular connectivity tests.

To ensure that the search appliance does not crawl a previously fed document, use googleoff/googleon tags (see "Excluding Unwanted Text from the Index" in *Administering Crawl*) or robots.txt (see "Using robots.txt to Control Access to a Content Server" in *Administering Crawl*).

To update the document, you need to feed the updated document to the search appliance. Documents fed with web feeds, including metadata-and-urls, are recrawled periodically, based on the crawl settings for the search appliance.

Note: The `metadata-and-url` feed type is one way to provide metadata to the search appliance. A connector can also provide metadata to the search appliance. See "Content Feed and Metadata-and-URL Feed" in the *Connector Developer's Guide*. See also the *External Metadata Indexing Guide* for information about external metadata.

Full Feeds and Incremental Feeds

Incremental feeds generally require fewer system resources than full feeds. A large feed can often be crawled more efficiently if it is divided into smaller incremental feeds.

The following example illustrates the effect of a full feed:

1. Create a new data source by pushing a feed that contains documents D0, D1 and D2. The system serves D0, D1, and D2.
2. Use the same data source name, you push a full feed that contains documents D0, an updated D1, and a new D3. When the feed processing is complete, the system serves D0, the updated D1, and the new D3. Because document D2 was not defined in the full feed, it is removed from the index.

The following example mixes full and incremental feeds:

1. Create a new data source by pushing a feed that contains documents D0, D1 and D2. The system serves D0, D1 and D2.
2. Push an incremental feed that defines the following actions: "add" for D3, "add" for an updated D1, and "delete" for D2. The system serves D0, updated D1, and D3. D0 was pushed by the first feed; because it is not referenced in the incremental feed, the D0's contents remain in the search results.
3. Push a full feed that contains documents D0, D7, and D10. The system serves D0, D7, and D10 when the full feed processing is complete. D1 and D3 are not referenced in the full feed, so the system removes them from the index and does not add them back.

Defining the XML Record for a Document

You include documents in your feed by defining them inside a `record` element. All records must specify a URL which is used as the unique identifier for the document. If the original document doesn't have a URL, but has some other unique identifier, you must map the document to a unique URL in order to identify it in the feed.

Each record element can specify following attributes:

- `url` (required)—The URL is the unique identifier for the document. This is the URL used by the search appliance when crawling and indexing the document. All URLs must contain a FQDN (fully qualified domain name) in the host part of the URL. Because the URL is provided as part of an XML document, you must escape any special characters that are reserved in XML. For example, the URL `http://www.mydomain.com/bar?a=1&b2` contains an ampersand character and should be rewritten to `http://www.mydomain.com/bar?a=1&b2`.

- `displayurl`—The URL that should be provided in search results for a document. This attribute is useful for web-enabled content systems where a user expects to obtain a URL with full navigation context and other application-specific data, but where a page does not give the search appliance easy access to the indexable content.

- `action`—Set action to `add` when you want the feed to overwrite and update the contents of a URL. If you don't specify an action, the system performs an add. Set `action` to `delete` to remove a URL from the index. The `action="delete"` feature works for content, web, and metadata-and-URL feeds.

- `lock`—The `lock` attribute can be set to `true` or `false` (the default is false). When the search appliance reaches its license limit, unlocked documents are deleted to make room for more documents. After all other remedies are tried and if the license is still at its limit, then locked documents are deleted. For more information, see "License Limits" on page 32.

- `mimetype` (required)—This attribute tells the system what kind of content to expect from the `content` element. All MIME types that can be indexed by the search appliance are supported.

  Note: Even though the feeds DTD (see "Google Search Appliance Feed DTD" on page 41) marks `mimetype` as required, `mimetype` is required only for content feeds and is ignored for web and metadata-and-url feeds (even though you are required to specify a value). The search appliance ignores the MIME type in web and metadata-and-URL feeds because the search appliance determines the MIME type when it crawls and indexes a URL.

- `last-modified`—For content feeds only. Populate this attribute with the date time format specified in RFC822 (Mon, 15 Nov 2004 04:58:08 GMT). If you do not specify a last-modified date, then the implied value is blank. The system uses the rules specified in the Admin Console under **Index > Document Dates** to choose which date from a document to use in the search results. The document date extraction process runs periodically so there may be a delay between the time a document appears in the results and the time that its date appears.

- `authmethod`—This attribute tells the system how to crawl URLs that are protected by NTLM, HTTP Basic, or Single Sign-on. The `authmethod` attribute can be set to `none`, `httpbasic`, `ntlm`, or `httpsso`. If a value for `authmethod` is not specified and a protected URL is defined on the search appliance, the default value for `authmethod` is the previously specified value for that URL. If the URL has not been previously specified on the search appliance, then the default value for `authmethod` is set to `none`. If you want to enable crawling for protected documents, see "Including Protected Documents in Search Results" on page 13.

- `pagerank`—For content feeds only. This attribute specifies the PageRank of the URL or group of URLs. For metadata-and-url feeds the effective PageRank will be the sum of this value (converted to an internal representation) and the PageRank calculated from crawling. The default value is 96. To alter the PageRank of the URL or group of URLs, set the value to an integer value between 68 and 100. Note that this PageRank value does not determine absolute relevancy, and the scale is not linear. Setting PageRank values should be done with caution and with thorough testing. The PageRank set for a URL overrides the PageRank set for a group.

- `feedrank`—This is a linear scale version of `pagerank`. Valid values are 1 to 100. For content feeds this will be the effective PageRank. For metadata-and-url feeds the effective PageRank will be the sum of this value and the PageRank calculated from crawling. Note that this value is ignored if the record or group also has a `pagerank` attribute set, and this PageRank value does not determine absolute relevancy. Setting PageRank values should be done with caution and with thorough testing. The value set for a record element overrides the value set for a group.

- `crawl-immediately`—For web and metadata-and-url feeds only. If this attribute is set to `"true"`, then the search appliance crawls the URL immediately. If a large number of URLs with `crawl-immediately="true"` are fed, then other URLs to be crawled are deprioritized or halted until these URLs are crawled. This attribute has no effect on content feeds.

- crawl-once—For web feeds only. If this attribute is set to "true", then the search appliance crawls the URL once, but does not recrawl it after the initial crawl. `crawl-once` urls can get crawled again if explicitly instructed by a subsequent feed using crawl-immediately.

Grouping Records Together

Record elements must be contained inside the group element. The `group` element also allows you to apply an action to many records at once. For example, this:

```
<group action="delete">
  <record url="http://www.corp.enterprise.com/hello01" mimetype="text/plain"/>
  <record url="http://www.corp.enterprise.com/hello02" mimetype="text/plain"/>
  <record url="http://www.corp.enterprise.com/hello03" mimetype="text/plain"/>
</group>
```

Is equivalent to this:

```
<record url="http://www.corp.enterprise.com/hello01" mimetype="text/plain
"action="delete"/>
<record url="http://www.corp.enterprise.com/hello02" mimetype="text/plain"
action="delete"/>
<record url="http://www.corp.enterprise.com/hello03" mimetype="text/plain"
action="delete"/>
```

However, if you define any actions for records as a group, the record's definition always overrides the group's definition. For example:

```
<group action="delete">
  <record url="http://www.corp.enterprise.com/hello01" mimetype="text/plain"/>
  <record url="http://www.corp.enterprise.com/hello02" mimetype="text/plain"
action="add"/>
  <record url="http://www.corp.enterprise.com/hello03" mimetype="text/plain"/>
</group>
```

In this example, hello01 and hello03 would be deleted, and hello02 would be updated.

Providing Content in the Feed

You add document content by placing it inside the record definition for your content feed. You can compress content to improve performance, for more information, see "Content Compression" on page 12.

For example, using text content:

```
<record url="..." mimetype="text/plain">
  <content>Hello world. Here is some page content.</content>
</record>
```

You can also define content as HTML:

```
<record url="..." mimetype="text/html">
<content><![CDATA[<html> <title>hello world</title>
<body>
  <p>Here is some page content.</p>
</body> </html>]]></content>
</record>
```

To include non-text documents such as `.pdf` or `.doc` files, you must encode the content by using base64 encoding and specifying the appropriate mimetype. Using base64 encoding ensures that the feed can be parsed as valid XML.

Here is a record definition that includes base64 encoded content:

```
<record url="..." mimetype="...">
  <content encoding="base64binary">Zm9vIGJhcgo</content>
</record>
```

Because base64 encoding increases the document size by one third, it is often more efficient to include non-text documents as URLs in a web feed. Only contents that are embedded in the XML feed must be encoded; this restriction does not apply to contents that are crawled.

Content Compression

Starting in Google Search Appliance version 6.2, content can be zlib compressed (see http://en.wikipedia.org/wiki/Zlib), which improves performance, because less data is sent across the network.

To send compressed content:

1. Zlib compress the content.
2. Base64 encode the content.
3. Add the content text to the `content` element in the feed's `record` statement.
4. Specify the `encoding="base64compressed"` attribute to the `content` element, for example:

```
<record url='http://feed.example.com/myfeed.html' action='add'
mimetype='text/html'>
  <content encoding="base64compressed">eDQwDYoOyVEYo01NUUPAJqkCgg=</content>
</record>
```

Adding Metadata Information to a Record

Metadata can be included in record definitions for different types of feeds. You can encode metadata using base64, for more information, see "Metadata Base64 Encoding" on page 13.

The following table provides information about incremental web feeds and metadata-and-URL feeds.

Data Source Name	Feed Type	Push Behavior	Allows Metadata?	Allows Content?
`web`	`incremental`	incremental	no	no
any	`metadata-and-url`	incremental	yes	no

The following table provides information about incremental and full content feeds.

Data Source Name	Feed Type	Push Behavior	Allows Metadata?	Allows Content?
any	`incremental`	incremental	yes	yes
any	`full`	full	yes	yes

If the metadata is part of a feed, it must have the following format:

```
<record url="..." ...>
  <metadata>
    <meta name="..." content="..." />
    <meta name="..." content="..." />
  </metadata>
  ...
</record>
```

Note: The `content=` attribute cannot be an empty string (""). For more information, see "Document Feeds Successfully But Then Fails" on page 35.

In version 6.2 and later, content feeds support the update of both content and metadata. Content feeds can be updated by just sending new metadata.

Generally, robots META tags with a value of noindex, nofollow, or noarchive can be embedded in the head of an HTML document to prevent the search appliance from indexing links or following them in the document. However, robots META tags in a feed file are not honored, just the META tags in the HTML documents themselves.

See the *External Metadata Indexing Guide* for more information about indexing external metadata and examples of metadata feeds.

Metadata Base64 Encoding

Starting in Google Search Appliance version 6.2, you can base64 encode metadata using the `encoding="base64binary"` attribute to the `meta` element. You can also base64 encode the metadata `name` attribute, however, both the `name` and `content` attributes must be base64 encoded if this option is used.

Note: Characters that be invalid XML characters feed correctly when encoded in base64.

For example:

```
<record url="http://example.com/myfeed.html" action="add" mimetype="text/html">
  <metadata>
    <meta encoding="base64binary" name="cHJvamVjdF9uYW1l"
content="Y2lyY2xlZ19yb2Nrcw==">
  </metadata>
</record>
```

Using the UTF-8 Encoding

Unless you have content in legacy systems that must use a national character set encoding, such as Shift_JIS, it is strongly recommended that all documents to be fed use the UTF-8 encoding. Do not escape `&` if using numeric character references, for example, the `&` character in `&#12521` should not be XML encoded as `&#12521`.

Including Protected Documents in Search Results

Feeds can push protected contents to the search appliance. If your feed contains URLs that are protected by NTLM, Basic Authentication, or Forms Authentication (Single Sign-on), the URL record in the feed must specify the correct type of authentication. You must also configure settings in the Admin Console to allow the search appliance to crawl the secured pages.

The `authmethod` attribute for the record defines the type of authentication. By default, `authmethod` is set to "none". To enable secure search from a feed, set the authentication attribute for the record to `ntlm`, `httpbasic`, or `httpsso`. For example, to enable authentication for protected files on localhost.example.com via Forms Authentication, you would define the record as:

```
<record url="http://www.localhost.example.com/" authmethod="httpsso">
```

To grant the search appliance access to the protected pages in your feed, log into the Admin Console.

For URLs that are protected by NTLM and Basic Authentication, follow these steps:

1. Open **Content Sources > Web Crawl > Secure Crawl > Crawler Access**
2. Define a pattern that matches the protected URLs in the feed.
3. Enter a username and password that will allow the crawler access to the protected contents. For contents on a Microsoft IIS server, you may also need to specify a domain.
4. The **Make Public** check box controls whether the search appliance checks for valid authentication credentials before including protected contents in the search results. If you select the **Make Public** check box, the record is displayed in search results. Otherwise, the record is shown when the user has valid authentication credentials; users who do not have access to the protected content will not see it in their search results. By default, search results are protected.

For URLs that are protected by Single Sign-on, follow these steps:

1. Open **Content Sources > Web Crawl > Secure Crawl > Forms Authentication**.
2. Under **Sample Forms Authentication protected URL**, enter the URL of a page in the protected site that will redirect the user to a login form. The login form must not contain JavaScript or frames. If you have more than one login page, create a Forms Authentication rule for each login.
3. Under **URL pattern for this rule**, enter a pattern that matches the protected URLs in the feed.
4. Click **Create**. In the browser page that opens, use the login form to enter a valid username and password. These credentials allow the crawler access to the protected contents. If the login information is accepted, you should see the protected page that you specified. If you can see the protected URL contents, click the **Save and Close** button. The Forms Authentication page now displays your rule.
5. Make any changes to the rule. For example, the **Make Public** check box controls whether the search appliance checks for valid authentication credentials before including protected contents in the search results. If you select the **Make Public** check box, the record is displayed in search results. Otherwise, the record is shown when the user has valid authentication credentials; users who do not have access to the protected content will not see it in their search results. By default, search results are protected.
6. When you have finished making changes to the rule, click **Save**.

Note: The **Make Public** check boxes will still apply to documents submitted through a content feed. If you submit a content feed with the `authmethod` attribute set, ensure that the fed URLs do not match any patterns on the **Content Sources > Web Crawl > Secure Crawl > Crawler Access** or **Content Sources > Web Crawl > Secure Crawl > Forms Authentication** pages that have the **Make Public** check box checked, unless you want those results to be public.

This is one way of providing access to protected documents. For more information on authentication, refer to the online help that is available in the search appliance's Admin Console, and in *Managing Search for Controlled-Access Content*.

Per-URL ACLs and ACL Inheritance

A per-URL ACL (access control list) has only a single URL associated with it. You can use feeds to add per-URL ACLs to the search appliance index. To specify a per-URL ACL, use the `acl` element, as described in "Specifying Per-URL ACLs" on page 15.

ACL information can be applied to groups of documents through inheritance. To specify ACL inheritance, use the attributes described in "Specifying ACL Inheritance" on page 17.

After you feed a per-URL ACL to the search appliance, the ACL and its inheritance chain appear on the **Index > Index Diagnostics** page.

For compatibility with feeds developed before software release 7.0, the search appliance supports the legacy format for specifying per-URL ACLs in feeds (deprecated). For more information, see "Legacy Metadata Format (Deprecated)" on page 19.

Take note that Google Search Appliance connectors, release 3.0, do not support feeding ACLs by using the legacy approach. They only support the approach documented in this section. If you update a search appliance to release 7.0 from an earlier release, re-crawling content is required.

The search appliance also supports other methods of adding per-URL ACLs to the index. For more information, see "Methods for Adding ACLs to the Index" in *Managing Search for Controlled-Access Content*.

You cannot use feeds to supply policy ACLs for prefix patterns or general URL patterns. To add policy ACLs for prefix patterns or general URL patterns, use either of the following methods:

- The **Search > Secure Search > Policy ACLs** page in the Admin Console

  For more information see Admin Console Help for this page.

- Policy ACL API

  For information about this API, see *Policy ACL API Developer's Guide*.

Specifying Per-URL ACLs

You can include a per-URL ACL in a feed by specifying a document, the principal (group or user), its access to the document, and ACL inheritance information.

`acl` Element

To specify all ACL information, including principals and inheritance, use the `acl` element. An `acl` element can have the following attributes:

- `url`
- `inheritance-type`
- `inherit-from`

The `acl` element can be the child of either a `group` or `record` element. For more information, see "Approaches to Using the acl Element" on page 18. The `acl` element is the parent of the `principal` element. For sample code, see "Example Feed with an acl Element" on page 18.

`url` Attribute

The `url` attribute directly associates the ACL with a URL. This attribute allows specifying ACLs for entities, such as folders and shares, without incrementing document count. For information about the `inheritance-type` and `inherit-from` attributes, see "Specifying ACL Inheritance" on page 17.

`principal` Element

To specify the principal, its name, and access to a document use the `principal` element. The `principal` element is a child of the `acl` element. The following code shows examples of the `principal` element:

```
<principal namespace="Default"
  case-sensitivity-type="everything-case-insensitive"
  scope="user" access="permit">yourdomain\username</principal>
<principal namespace="Default"
  case-sensitivity-type="everything-case-insensitive"
  scope="group" access="permit">yourdomain\groupname</principal>
```

A `principal` element can have the following attributes:

- `scope`
- `access`
- `namespace`
- `case-sensitivity-type`
- `principal-type`

`scope` Attribute

The `scope` attribute specifies the type of the principal. Valid values are:

- `user`
- `group`

The `scope` attribute is required.

`access` Attribute

The `access` attribute specifies the principal's permission to access the document. Valid values are:

- `permit`
- `deny`

The `access` attribute is required.

`namespace` Attribute

By keeping ACLs in separate namespaces, the search appliance is able to ensure that access to secure documents is maintained unambiguously. Namespaces are crucial to security when a search user has multiple identities and the permissions for documents are composed of ACLs from separate content sources. The `namespace` attribute specifies the global namespace for the user or group. The global namespace corresponds to the credential group for a content source. For detailed information about credential groups, see "Universal Login" in *Managing Search for Controlled-Access Content*.

`case-sensitivity-type` Attribute

The `case-sensitivity-type` attribute specifies whether or not the principal's name is case sensitive. At serve time, the search appliance compares the principal's name as entered on the Universal Login form with the one stored in the index. For example, suppose the name in the index is "ALEX" and the name on the form is "Alex." If the name is case sensitive, access to the document is denied. Valid values are:

- `everything-case-sensitive`
- `everything-case-insensitive`

`principal-type` Attribute

The `principal-type` attribute indicates that the domain string attached to the principal will not be transformed internally by the search appliance. The only valid value is "unqualified." This attribute is for support of SharePoint local groups.

Specifying ACL Inheritance

While ACLs can be found attached to documents, content systems allow for ACL information to be applied to groups of documents through inheritance. The search appliance is able to model a wide variety of security mechanisms by using the concept of ACL inheritance.

For example, in a Microsoft Windows File System, by default, a document inherits permissions from its folder. Permissions can be applied to documents without breaking inheritance. More specific permissions override less specific permissions.

In a Microsoft Windows Share, permissions can be applied to the share as a whole. All documents in the tree rooted at the shared folder implicitly inherit share permissions. Share permissions always override more specific permissions.

In Microsoft SharePoint, content is organized in hierarchies of sites, document collections, and documents. Each node in the hierarchy inherits permissions from its parent, but if a DENY occurs anywhere in the inheritance chain, the resulting decision is DENY.

ACL inheritance is specified by the following attributes of the `acl` element:

- `inheritance-type`
- `inherit-from`

`inheritance-type` Attribute

The `inheritance-type` attribute specifies how the permissions (PERMIT, DENY, INDETERMINATE) will be interpreted when the search appliance authorizes against parent and child ACLs and decides which takes precedence.

Valid values are:

- `parent-overrides`--The permission of the parent ACL dominates the child ACL, except when the parent permission is INDETERMINATE. In this case, the child permission dominates. If both parent and child are INDETERMINATE, then the permission is INDETERMINATE.
- `child-overrides`--The permission of the child ACL dominates the parent ACL, except when the child permission is INDETERMINATE. In this case, the parent permission dominates. If both parent and child are INDETERMINATE, then the permission is INDETERMINATE.
- `and-both-permit`--The permission is PERMIT only if both the parent ACL and child ACL permissions are PERMIT. Otherwise, the permission is DENY.
- `leaf-node`--ACL that terminates the chain.

`inherit-from` Attribute

The `inherit-from` attribute specifies the URL from which the ACL inherits permissions. If this attribute is absent, the ACL is a top-level node.

Note: If a per-URL ACL inherits from a non-existent URL, or inherits from a URL that does not have a per-URL ACL, the authorization decision is always INDETERMINATE because of the broken inheritance chain.

Approaches to Using the acl Element

There are two approaches to using the `acl` element:

- As the child of a `group` element
- As the child of a `record` element

If the `acl` element is the child of a `group` element, the `url` attribute is required. An `acl` element as the child of a `group` element can be used in the following scenarios:

- Updating the ACL of a record (independently of content) for a URL that was previously fed with attached ACLs.
- Modeling entities, such as folders, that would not otherwise be represented in the system.

If the `acl` element is the child of a `record` element, the `url` attribute is illegal. In this approach, the ACL is immediately associated with the document.

Google does not recommend mixing the two approaches to using an `acl` element for a given URL in the same feed.

Example Feed with an acl Element

The following code shows an example of a feed XML file with an `acl` element that inherits permissions.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE gsafeed PUBLIC "-//Google//DTD GSA Feeds//EN" "gsafeed.dtd">
<gsafeed>
 <header>
  <datasource>TestUrlAcl</datasource>
  <feedtype>incremental</feedtype>
 </header>
<group …>
 <acl url="http://dummyhost.corp.google.com/" inheritance-type=”child-overrides”
    inherit-from="http://corp.google.com/">
<!-- spaces are stripped from principal text content -->
  <principal scope="user" access="permit">edward</principal>
  <principal scope="user" access="deny">william</principal>
  <principal scope="user" access="deny">ben</principal>
  <principal scope="group" access="permit">nobles</principal>
  <!-- empty, will be skipped -->
  <principal scope="group" access="permit">   </principal>
  <principal scope="group" access="deny">playwrights</principal>
 </acl>
<!-- Other acl elements or group elements with records can appear in any order in
this file -->
<record url='http://dummyhost.corp.google.com/'…>
```

Legacy Metadata Format (Deprecated)

For compatibility with feeds developed before software release 7.0, the search appliance supports the legacy metadata format for specifying per-URL ACLs in feeds. The legacy approach is limited: it does not support namespaces or case sensitivity. However, the following meta names enable you to specify ACL inheritance in metadata format:

- `google:aclinheritfrom`
- `google:aclinheritancetype`

The valid value for `google:aclinheritfrom` is a URL string.

Google recommends against using the legacy format unless you have legacy feeds to maintain. Instead, Google recommends developing feeds using the approach described in Specifying Per-URL ACLs.

A per-URL ACL can be defined either in the metadata portion of the feed, or in the document itself, but not in both places.

Specifying Group and User Access in Metadata

You can include a per-URL ACL in a feed by specifying a document, and the names of the groups or users that have access. The list of groups and users appears inside the `record` element for the document that you are feeding. To specify groups or users that have access to the restricted URL, define `meta` elements with `name` and `content` attributes.

To specify a group, use the following attribute values:

- For the `name` attribute, the value must be `google:aclgroups`.
- For the `content` attribute, the value must be a single group name.

To specify more than one group, use more than one meta tag, one group for each tag.

A group name that you specify in a `content` attribute value must match the group name as it appears in the authentication mechanism (LDAP or GDATA database).

For example, to specify engineering (“eng”) as the group that has access to the URL, use the following code:

```
<meta name="google:aclgroups" content="eng"/>
```

To specify a user, use the following attribute values:

- For the `name` attribute, the value must be `google:aclusers`.
- For the `content` attribute, the value must be a single user name.

To specify more than one user, use more than one meta tag, one user for each tag.

A user name that you specify in a `content` attribute value must match the user name as it appears in the authentication mechanism (LDAP or GDATA database).

For example, to specify Joe, Maria, and Salim as the users that have access to the URL, use the following code:

```
<meta name="google:aclusers" content="joe"/>
<meta name="google:aclusers" content="maria"/>
<meta name="google:aclusers" content="salim"/>
```

If a content string ends in `=owner`, `=peeker`, `=reader`, or `=writer`, that suffix is stripped from the user name. Furthermore, if a content string ends in `=peeker`, that ACL entry is ignored.

Specifying Denial of Access to Users and Groups

The search appliance supports DENY ACLs. When a user or group is denied permission to view the URL, it does not appear in the search results. You can specify users and groups that are not permitted to view a document by using meta tags, as shown in the following examples.

To specify denial of access, the value of the `name` attribute must be `google:acldenyusers` or `google:acldenygroups`.

For example, to specify Joe as the user who is denied access to a document, use the following code:

```
<meta name="google:acldenyusers" content="joe"/>
```

To specify administration as the group that is denied access to a document, use the following code:

```
<meta name="google:acldenygroups" content="administration"/>
```

Generally, DENY takes precedence over PERMIT. The following logic determines authorization decisions for per-URL ACLs:

1. start with decision=INDETERMINATE
2. if the user is denied, return DENY
3. if the user is permitted, set decision to PERMIT
4. if any of the groups are denied, return DENY
5. if any of the groups are permitted, set decision to PERMIT
6. if decision is INDETERMINATE, set to DENY
7. return decision

Example of a Legacy Per-URL ACL for a Feed

The following example shows the legacy code for adding a per-URL ACL for `http://insidealpha.com/personnel.htm` in a feed:

```
<?xml version='1.0' encoding='UTF-8'?>
  <!DOCTYPE gsafeed PUBLIC "-//Google//DTD GSA Feeds//EN" "gsafeed.dtd">
  <gsafeed>
    <header>
      <datasource>IAIncrementalFeedContent</datasource>
      <feedtype>incremental</feedtype>
    </header>
    <group>
      <record url="http://insidealpha.com/personnel.htm">
        <metadata>
          <meta name="google:aclgroups" content="eng"/>
          <meta name="google:aclusers" content="joe"/>
          <meta name="google:aclusers" content="maria"/>
          <meta name="google:aclusers" content="salim"/>
          <meta name="google:acldenyusers" content="joe"/>
          ...
        </metadata>
      </record>
    </group>
  </gsafeed>
```

Feeding Groups to the Search Appliance

The search appliance can experience increased latency when establishing a user's identity and the groups that it belongs to. You can dramatically reduce the latency for group resolution by periodically feeding groups information to the search appliance. When the groups information is on the search appliance, it is available in the security manager for resolving groups at authentication time. Consequently, the information works for all authorization mechanisms.

Take note that the cumulative number of group members on the search appliance cannot exceed three million.

To feed groups to the search appliance, start by:

- Designing an XML Groups Feed
- Creating a Groups Feed Client

Designing an XML Groups Feed

The XML groups feed contains information about principals (groups) and its members (groups or users). The XML must conform to the schema defined in the Groups Feed Document Type Definition.

`xmlgroups` Element

To specify all groups information, including memberships, principals, and members, use the `xmlgroups` element.

`membership` Element

The `membership` element must contain one `principal` element. It contains zero to one `members` elements.

`members` Element

The `members` element contains zero to many `principal` elements.

`principal` Element

To specify the principal, its name, and access to a document, use the `principal` element. The `principal` element is a child of the `membership` or `members` element.

For any `principal` element that is a child of the `membership` element, the `scope` must be `GROUP` (users cannot have members) and should not include the `case-sensitivity-type` element. If you choose to include the `case-sensitivity-type`, it must be `EVERYTHING_CASE_SENSITIVE` because matching only happens against group members.

The following code shows examples of the `principal` element:

```
<principal namespace="Default"
  scope="GROUP">
  abc.com/group1
</principal>
```

A `principal` element can have the following attributes:

- `scope`
- `namespace`
- `case-sensitivity-type`
- `principal-type`

`scope` Attribute

The `scope` attribute specifies the type of the principal. Valid values are:

- `USER`
- `GROUP`

The `scope` attribute is required.

`namespace` Attribute

By keeping principals in separate namespaces, the search appliance is able to ensure that access to secure documents is maintained unambiguously. Namespaces are crucial to security when a search user has multiple identities. The `namespace` attribute specifies the global namespace for the user or group. The global namespace corresponds to the credential group for a content source. For detailed information about credential groups, see "Universal Login" in *Managing Search for Controlled-Access Content*.

`case-sensitivity-type` Attribute

The `case-sensitivity-type` attribute specifies whether or not the principal's name is case sensitive. At serve time, the search appliance compares the principal's name as entered on the Universal Login form with the one stored in the index. For example, suppose the name in the index is "ALEX" and the name on the form is "Alex." If the name is case sensitive, access to the document is denied. Valid values are:

- `EVERYTHING-CASE-SENSITIVE`
- `EVERYTHING_CASE_INSENSITIVE`

`principal-type` Attribute

The `principal-type` attribute indicates that the domain string attached to the principal will not be transformed internally by the search appliance. The only valid value is "unqualified." This attribute is for support of SharePoint local groups.

Example Feed with Groups

The following code shows an example of a feed XML file with groups.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE gsafeed PUBLIC "-//Google//DTD GSA Feeds//EN" "">
<xmlgroups>
  <membership>
    <principal namespace="Default"
      case-sensitivity-type="EVERYTHING_CASE_INSENSITIVE" scope="GROUP">
        abc.com/group1
    </principal>
    <members>
      <principal namespace="Default"
        case-sensitivity-type="EVERYTHING_CASE_INSENSITIVE" scope="GROUP">
          subgroup1
      </principal>
      <principal namespace="Default"
        case-sensitivity-type="EVERYTHING_CASE_INSENSITIVE" scope="USER">
          user1
      </principal>
    </members>
  </membership>
  <membership>
    <principal namespace="Default"
      case-sensitivity-type="EVERYTHING_CASE_INSENSITIVE" scope="GROUP">
        subgroup1
    </principal>
    <members>
      <principal namespace="Default"
        case-sensitivity-type="EVERYTHING_CASE_INSENSITIVE" scope="USER">
          example.com/user2
      </principal>
    </members>
  </membership>
</xmlgroups>
```

Example Feed with Empty Groups

The following code shows an example of a feed XML file with empty groups.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE gsafeed PUBLIC "-//Google//DTD GSA Feeds//EN" "">
<xmlgroups>
</xmlgroups>
```

Groups Feed Document Type Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT xmlgroups (membership+)>
<!ELEMENT membership (principal, members)>
<!ELEMENT members (principal+)>
<!ELEMENT principal (#PCDATA)>

<!ATTLIST principal
  scope (USER|GROUP) #REQUIRED
  namespace CDATA "Default"
  case-sensitivity-type (EVERYTHING_CASE_SENSITIVE|EVERYTHING_CASE_INSENSITIVE)
    "EVERYTHING_CASE_SENSITIVE"
  principal-type (unqualified) #IMPLIED>
```

Creating a Groups Feed Client

A feed client is required for pushing groups information.

You upload an XML feed using an HTTP POST to the feedergate server located on port 19900 of your search appliance. An XML feed must be less than 1 GB in size. If your feed is larger than 1GB, consider breaking the feed into smaller feeds that can be pushed more efficiently.

On the search appliance, a feedergate handler, feedergate_groupsfeed, accepts the POST request, parses the XML into serialized data and stores it in the recordio file. The recordio file cannot exceed 1GB. If the data size in the request plus the existing recordio file size is greater than 1GB, the request will be rejected with "error: group db at capacity."

The feedergate server requires two input parameters from the POST operation:

- groupsource: The name of the data source. Must be one of the following values: sharepoint, ggg, ldap, and others.
- groupsfilename: The groups XML file you want to push to the search appliance.
- feedtype: Specifies whether a feed is full or incremental. A full feed overwrites onboard groups. An incremental feed appends groups.

The URL that you should use for the groups feed is:

```
http://<APPLIANCE-HOSTNAME>:19900/xmlgroups
```

The following example shows the command to push a groups feed:

```
pushgroups_client.py --groupsource=others --feedtype=full
--url="http://<APPLIANCE_HOSTNAME>:19900/xmlgroups"
--groupsfilename="groups_feed.xml"
```

Note: To clear the groups database, use the empty content XML file as shown in "Example Feed with Empty Groups" on page 23 and perform a full push (`feedtype=full`).

The following example shows a groups feed client written in Python.

```
# Copyright 2013 Google Inc. All Rights Reserved.

"""A helper script that pushes a groups xml file to the feeder."""

import getopt
import mimetypes
import sys
import urllib2


def PrintMessage():
  """Print help message for command usage."""
  print """Usage: %s ARGS
  --groupsource:  sharepoint, ggg, ldap or others
  --url:  groupsfeed url of the feedergate, e.g. http://gsabox:19900/xmlgroups
  --groupsfilename:  The groups xml file you want to feed
  --help: output this message""" % sys.argv[0]


def main(argv):
  """Process command line arguments and send feed to the webserver."""
  try:
    opts, _ = getopt.getopt(argv[1:], None,
                            ["help", "groupsource=",
                             "url=", "groupsfilename="])
  except getopt.GetoptError:
  # print help information and exit:
    PrintMessage()
    sys.exit(2)

  groupsource = None
  url = None
  groupsfilename = None

  for opt, arg in opts:
    if opt == "--help":
      PrintMessage()
      sys.exit()
    if opt == "--groupsource":
      groupsource = arg
    if opt == "--url":
      url = arg
    if opt == "--groupsfilename":
      groupsfilename = arg

  params = []

  if (url and groupsfilename and
      groupsource in ("sharepoint", "ggg", "ldap", "others")):
    params.append(("groupsource", groupsource))
    data = ("data", groupsfilename, open(groupsfilename, "r").read())
    request_url = PostMultipart(url, params, (data,))
    print urllib2.urlopen(request_url).read()

  else:
    PrintMessage()
    sys.exit(1)
```

```
def PostMultipart(theurl, fields, files):
  """Create the POST request by encoding data and adding headers."""
  content_type, body = EncodeMultipartFormdata(fields, files)
  headers = {}
  headers["Content-type"] = content_type
  headers["Content-length"] = str(len(body))
  return urllib2.Request(theurl, body, headers)


def EncodeMultipartFormdata(fields, files):
  """Create data in multipart/form-data encoding."""
  boundary = "----------boundary_of_feed_data$"
  crlf = "\r\n"
  l = []
  for (key, value) in fields:
    l.append("--" + boundary)
    l.append('Content-Disposition: form-data; name="%s"' % key)
    l.append("")
    l.append(value)
  for (key, filename, value) in files:
    l.append("--" + boundary)
    l.append('Content-Disposition: form-data; name="%s"; filename="%s"'
             % (key, filename))
    l.append("Content-Type: %s" % GetContentType(filename))
    l.append("")
    l.append(value)
  l.append("--" + boundary + "--")
  l.append("")
  body = crlf.join(l)
  content_type = "multipart/form-data; boundary=%s" % boundary
  return content_type, body


def GetContentType(filename):
  """Determine the content-type of data file."""
  return mimetypes.guess_type(filename)[0] or "application/octet-stream"

if __name__ == "__main__":
  main(sys.argv)
```

For more information about developing a feed client, see “Pushing a Feed to the Google Search Appliance” on page 27.

Java Code

In Java, use the `DocIdPusher.pushGroupDefinitions()` method to send groups to the Google Search Appliance.

Using the Admin Console Feeds page

The **Content Sources > Groups** page in the Admin Console enables you to download the groups DB file to the search appliance, view information about groups on the search appliance, or delete all the contents in the groups DB file. For more information, click **Help > Content Sources > Groups** in the Admin Console.

Feeding Content from a Database

To push records from a database into the search appliance's index, you use a special content feed that is generated by the search appliance based on parameters that you set in the Admin Console. To set up a feed for database content, log into the Admin Console and choose **Content Sources > Databases**. You can find more information on how to define a database-driven data source in the online help that is available in the Admin Console, and in the section "Database Crawling and Serving" in *Administering Crawl*.

Records from a database cannot be served as secure content.

Saving your XML Feed

You should save a backup copy of your XML Feed in case you need to push it again. For example, if you perform a version update that requires you to rebuild the index, you must push all your feeds again to restore them to the search appliance. The search appliance does not archive copies of your feeds.

Feed Limitations

For information about feed limitations, see Specifications and Usage Limits.

Pushing a Feed to the Google Search Appliance

This section describes how to design a feed client. To design your own feed client, you should be familiar with these technologies:

- HTTP—Hypertext Transfer Protocol (http://www.w3.org/Protocols/)
- XML—Extensible Markup Language (http://www.w3.org/XML/)
- A scripting language, such as Python

If you don't want to design your own feed client script, you can use one of the following methods to push your feed:

- Google provides an example of a Python 2.x feed client script, pushfeed_client.py, that you can use to push an XML feed. (Google also provides a Python 3 version, pushfeed_client3.py). You can also use this script in a `cron` job to automate feeds.
- "Using a Web Form Feed Client" on page 28 explains how to write a simple HTML form that allows a user to push an XML feed from a web page. Adapt the HTML for your use and add this page to any web server that has HTTP access to the search appliance.

Important: The IP address of the computer that hosts the feed client must be in the **List of Trusted IP Addresses**. In the **Admin Console**, go to **Content Sources > Feeds**, and scroll down to **List of Trusted IP Addresses**. Verify that the IP address for your feed client appears in this list.

Designing a Feed Client

You upload an XML feed using an HTTP POST to the feedergate server located on port 19900 of your search appliance. The search appliance also supports HTTPS access to the feedergate server through port 19902, enabling you to upload an XML feed file by using a secure connection. An XML feed must be less than 1 GB in size. If your feed is larger than 1 GB, consider breaking the feed into smaller feeds that can be pushed more efficiently.

The feedergate server requires three input parameters from the POST operation:

- `datasource` specifies the name of the data source. Your choice of data source name also implies the type of feed: for a web feed, the `datasource` name must be "web".
- `feedtype` specifies how the system pushes the feed. The `feedtype` value must be "full", "incremental", or "metadata-and-url".
- `data` specifies the XML feed to push with this data source and feed type. Note that although the `data` parameter may contain a data source and a feed type definition as part of the XML, these will be ignored by the search appliance. Only the data source and feed type provided as POST input parameters are used.

The URL that you should use is:

```
http://<APPLIANCE-HOSTNAME>:19900/xmlfeed
```

You should post the feed using `enctype="multipart/form-data"`. Although the search appliance supports uploads using `enctype="application/x-www-form-urlencoded"`, this encoding type is not recommended for large amounts of data.

The feed client should URL encode the XML data submitted to the search appliance.

Using a Web Form Feed Client

Here is an example of a simple HTML form for pushing a feed to the search appliance. Because the web form requires user input, this method cannot be automated.

To adapt this form for your search appliance, replace `APPLIANCE-HOSTNAME` with the fully qualified domain name of your search appliance.

```
<html>
  <head>
    <title>Simple form for pushing a feed</title>
  </head>
  <body>
    <h1>Simple form for pushing a feed</h1>
    <form enctype="multipart/form-data" method=POST
      action="http://<APPLIANCE-HOSTNAME>:19900/xmlfeed">
      <p>Name of datasource:
        <input type="text" name="datasource">
        <br>
        (No spaces or non alphanumeric characters)
      </p>
      <p>Type of feed:
        <input type="radio" name="feedtype" value="full" checked>
        Full
        <input type="radio" name="feedtype" value="incremental">
        Incremental
        <input type="radio" name="feedtype" value="metadata-and-url">
        Metadata and URL
      </p>
      <p>
        XML file to push:
        <input type="file" name="data">
      </p>
      <p>
        <input type="submit" value=">Submit<">
      </p>
    </form>
  </body>
</html>
```

How a Feed Client Pushes a Feed

When pushing a feed, the feed client sends the POST data to a search appliance. A typical POST from a scripted feed client appears as follows:

```
POST /xmlfeed HTTP/1.0
Content-type: multipart/form-data
Content-length: 855
Host: myserver.domain.com:19900
User-agent: Python-urllib/1.15
feedtype=full&datasource=sample&data=%3C%3Fxml+version%3D%221.0%22+encoding%3D%2
2UTF-8%22%3F%3E%0A%3C%21DOCTYPE+gsafeed+SYSTEM+..
```

The response from the search appliance is as follows:

```
HTTP/1.0 200 OK
Content-Type: text/plain
Date: Thu, 30 Apr 2009 23:16:10 GMT
Server: feedergate_1.0
Connection: Close
Content-Length: 7

Success
```

The success message indicates that the feedergate process has received the XML file successfully. It does not mean that the content will be added to the index, as this is handled asynchronously by a separate process known as the "feeder". The data source will appear in the Feeds page in the Admin Console after the feeder process runs.

The feeder does not provide automatic notification of a feed error. To check for errors, you must log into the Admin Console and check the status on the **Content Sources > Feeds** page. This page shows the last five feeds that have been uploaded for each data source. The timestamp shown is the time that the XML file has been successfully uploaded by the feedergate server.

You can automate the process of uploading a feed by running your feed client script with a `cron` job.

Turning Feed Contents Into Search Results

URL Patterns and Trusted IP lists defined in the Admin Console ensure that your index only lists content from desirable sources. When pushing URLs with a feed, you must verify that the Admin Console will accept the feed and allow your content through to the index. For a feed to succeed, it must be fed from a trusted IP address and at least one URL in the feed must pass the rules defined on the Admin Console.

URL Patterns

URLs specified in the feed will only be crawled if they pass through the patterns specified on the **Content Sources > Web Crawl > Start and Block URLs** page in the Admin Console.

Patterns affect URLs in your feed as follows:

- **Do Not Follow Patterns**—If a URL in the feed matches a pattern specified under **Do Not Crawl URLs with the Following Patterns**, the URL is removed from the index.
- **Follow Patterns**—When this pattern is used, all URLs in the feed must match a pattern in this list. Any other URLs are removed from the index.

Entries in duplicate hosts also affect your URL patterns. For example, suppose you have a canonical host of `foo.mycompany.com` with a duplicate host of `bar.mycompany.com`. If you exclude `bar.mycompany.com` from your crawl using patterns, then URLs on both `foo.mycompany.com` and `bar.mycompany.com` are removed from the index.

Trusted IP Lists

To prevent unauthorized additions to your index, feeds are only accepted from machines that are included in the **List of Trusted IP Addresses**. To view the list of trusted IP addresses, log into the Admin Console and open the **Content Sources > Feeds** page.

If your search appliance is on a trusted network, you can disable IP address verification by selecting **Trust all IP addresses**.

Adding Feed Content

For web feeds, the feeder passes the URLs to the crawl manager. The crawl manager adds the URLs to the crawl schedule. URLs are crawled on the schedule specified by the documentation on the continuous crawler.

For content feeds, the content is provided as part of the XML and does not need to be fetched by the crawler. URLs are passed to the server that maintains Crawl Diagnostics in the Admin Console. This will happen within 15 minutes if your system is not busy. The feeder also passes the URLs and their contents to the indexing process. The URLs will appear in your search results within 30 minutes if your system is not busy.

Removing Feed Content From the Index

There are several ways of removing content from your index using a feed. The method used to delete content depends on the kind of feed that has ownership.

For content feeds, remove content by performing one of these actions:

- Push the URL as part of an incremental feed, using the "delete" action to remove the content. This is the fastest way to remove content. URLs will be deleted within about 30 minutes.
- Remove the URL from the feed and perform a full feed. Because a full feed overwrites the earlier feed contents, any URLs that are omitted from the new full feed will be removed from the index. The content is deleted within about 30 minutes.
- Remove the data source and all of its contents. To remove a data source, log into the Admin Console and open the **Content Sources > Feeds** page. Choose the data source that you want to remove and click **Delete**. The contents will be deleted within about 30 minutes. The **Delete** option removes the fed documents from the search appliance index. The feed is then marked Delete in the Admin Console.
- After deleting a feed, you can remove the feed from the Admin Console Feed Status page by clicking **Destroy**.

For web and metadata-and-URL feeds, remove content by performing one of these actions:

- In the XML record for the document, set `action` to `delete`. The `action="delete"` feature works for content, web, and metadata-and-URL feeds.
- Remove the URL from the web server. The next time that the URL is crawled, the system will encounter a 404 status code and remove the content from the index.
- Specify a pattern that removes the URL from the index. For example, add the URL to the **Do Not Follow Patterns** list. The URL is removed the next time that the feeder delete process runs.

Note: If a URL is referenced by more than one feed, you will have to remove it from the feed that owns it. See the Troubleshooting entry "Fed Documents Aren't Updated or Removed as Specified in the Feed XML" on page 35 for more information.

Time Required to Process a Feed

The following factors can cause the feeder to be slow to add URLs to the index:

- The feed is large.
- The search appliance is currently using a lot of resources to crawl other documents and serve results.
- Other feeds are pending.

In general, the search appliance can process documents that are pushed as content feeds more quickly than it can crawl and index the same set of documents as a web feed.

Feed Files Awaiting Processing

To view a count of how many feed files remain for the search appliance to process into its index, add `/getbacklogcount` to a search appliance URL at port 19900. The count that this feature provides can be used to regulate the feed submission rate. The count also includes connector feed files.

The syntax for `/getbacklogcount` is as follows:

```
http://SearchApplianceHostname:19900/getbacklogcount
```

Changing the Display URL in Search Results

You can change the display URL on search results by pushing a feed with the `displayurl` attribute set.

Use this feature when you want to use one URL in the index and another for display to the user. For example, you might change the display URL if URL content is not in a web enabled server (and you need to specify a proxy server that uses doc IDs in a back-end content management system) or if you split a large file into segments and each segment is indexed with a separate URL and the display URL for each result points to the original file.

The following example shows use of the `displayurl` attribute.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE gsafeed PUBLIC "-//Google//DTD GSA Feeds//EN" "">
<gsafeed>
  <header>
    <datasource>replace</datasource>
    <feedtype>incremental</feedtype>
  </header>
  <group>
    <record url="http://oldurl.example.com/docID=123"
      displayurl="http://newurl.example.com/myscript.cgi?docID=123"
      action="add" mimetype="text/html" lock="true">
      <content>Hello World - document data goes here!</content>
    </record>
  </group>
</gsafeed>
```

License Limits

If your index already contains the maximum number of URLs, or your license limit has been exceeded, then the index is full.

When the index is full, the system reduces the number of indexed documents as follows:

- Documents are removed to bring the total number of documents to the license limit.
- Documents with the `lock` attribute set to `true` are deleted last.

Increasing the Maximum Number of URLs to Crawl

To increase the maximum number of URLs in your index, log into the Admin Console and choose **Content Sources > Web Crawl > Host Load Schedule**. Check the **Maximum Number of URLs to Crawl**. This number must be smaller than the license limit for your search appliance. To increase the license limit, contact Sales.

Troubleshooting

Here are some things to check if a URL from your feed does not appear in the index. To see a list of known and fixed issues, see the latest release notes for each version.

Error Messages on the Feeds Status Page

If the feeds status page shows “Failed in error” you can click the link to view the log file.

ProcessFeed: parsing error

This message means that your XML file could not be understood. The following are some possible causes of this error:

- There is an error in the DOCTYPE line in your XML file. This line should be: `<!DOCTYPE gsafeed PUBLIC "-//Google//DTD GSA Feeds//EN""">`
- You have not escaped the ampersand or other reserved characters in a URL in your XML file.
- You have included content that requires base64 encoding. See “Providing Content in the Feed” on page 11.

If none of the above are the cause of the error, run `xmllint` against your XML file to check for errors in the XML. The `xmllint` program is included in Linux distributions as part of the `libxml2` package.

The following is an example that shows how you would use `xmllint` to test a feed named `full-feed.xml`.

```
$ xmllint -noout -valid full-feed.xml; echo $?

  0
```

The return code of zero indicates that the document is both valid and well-formed.

If the `xmllint` command fails and displays the `parsing error` message, ensure that you have the correct DTD file, or you can remove the `-valid` flag from the `xmllint` command line so that the `xmllint` command doesn’t try to validate the XML file’s elements. For more information on the DTD, see “Google Search Appliance Feed DTD” on page 41.

Feed Push is Not Successful

Before a search appliance can start processing a feed, you need to successfully push a feed to port 19900.

If the feed push is not successful, check the following:

- List the IP address of the computer that hosts the feed client in the **List of Trusted IP Addresses**. For more information, see “Pushing a Feed to the Google Search Appliance” on page 27.
- Verify that the feed client connects to port 19900 on the correct IP address.
- Verify that port 19900 is reachable on the search appliance by running `tracepath applianceIP/19900` from the Linux command line.
- Check with your network team for firewall or connectivity issues to ensure access to port 19900.

Fed Documents Aren't Appearing in Search Results

Some common reasons why the URLs in your feed might not be found in your search results include:

1. The crawler is still running. Wait a few hours and search again. For large document feeds containing multiple non-text documents, the search appliance can take several minutes to process all of the documents. You can check the status of a document feed by going to the **Content Sources > Feeds** page. You can also verify that the documents have been indexed by going to **Index > Diagnostics > Index Diagnostics** and browsing to the URL, or entering the URL in **"**URLs starting with.**"** Documents that are fed into the search appliance can show up in Crawl Diagnostics up to 15 minutes before they are searchable in the index.

2. The URLs were removed by an exclusion pattern specified under **Content Sources > Web Crawl > Start and Block URLs**. See "URL Patterns" on page 30.

3. The URLs were removed by a full feed that did not include them. See Removing Feed Content From the Index.

4. The URLs don't match the pattern for the collection that you were searching. Check the patterns for your collection under **Index > Collections**. Make sure that the collection specified in the upper right hand corner of the Crawl Diagnostics page contains the URL that you are looking for.

5. The URLs are listed in multiple feeds. Another feed that contains this URL requested a `delete` action.

6. A metadata-and-URL feed was submitted with the `feedtype` element set to `incremental` or `full`. Incremental can only be used on a content feed. If this is the case, the feed is treated as a content feed and not crawled. Once a URL is part of a content feed, the feed is not recrawled even if you later send a web or metadata feed. If you run into this issue, remove the URL from the URL pattern (or click the **Delete** link on the feeds page) and after the feed URLs have been deleted, put the URL patterns back, and send a proper metadata-and-url feed.

7. The documents were removed because your index is full. See "License Limits" on page 32.

8. The feed that you pushed was not pointing to a valid host. Verify that the feed has an FQDN (fully qualified domain name) in the host part of the URL.

9. More relevant documents are pushing the fed URL down in the list. You can search for a specific URL with the query `info:[url]` where `[url]` is the full URL to a document fed into the search appliance. Or use `inurl:[path]` where `[path]` is part of the URL to documents fed into the search appliance.

10. The fed document has failed. In this scenario, none of the external metadata fed by using a content feed or metadata-and-URL feed would get indexed. In the case of metadata-and-URL feeds, just the URL gets indexed without any other information. For additional details about the failure, click **Index > Diagnostics > Index Diagnostics**.

11. The URLs are on a protected server and cannot be indexed. See "Including Protected Documents in Search Results" on page 13.

12. The URLs are on a protected server and have been indexed, but you do not have the authorization to view them. Make sure that `&access=a` is somewhere in the query URL that you are sending to the search appliance. See "Including Protected Documents in Search Results" on page 13.

13. You did not complete the upgrade from a previous version and are still running in "Test mode" with the old Index. Review the *Update Instructions* for the current version of the software, and make sure that you have accepted the upgrade and completed the update process.

Document Feeds Successfully But Then Fails

A content feed reports success at the feedergate, but thereafter, reports the following document feed error:

```
Failed in error
documents included: 0
documents in error: 1
error details: Skipping the record, Line number: nn,
Error: Element record content does not follow the DTD, Misplaced metadata
```

This error occurs when a `metadata` element contains a `content` attribute with an empty string, for example:

```
<meta name="Tags" content=""/>
```

If the `content` attribute value is an empty string:

- Remove the `meta` tag from the `metadata` element, or:
- Set the value of the `content` attribute to show that no value is assigned. Choose a value that is not used in the `metadata` element, for example, `_noname_`:

  ```
  <meta name="Tags" content="_noname_"/>
  ```

  You can then use the `inmeta` search keyword to find the attribute value in the fed content, for example:

  ```
  inmeta:tags~_noname_
  ```

Fed Documents Aren't Updated or Removed as Specified in the Feed XML

All feeds, including database feeds, share the same name space and assume that URLs are unique. If a fed document doesn't seem to behave as directed in your feed XML, check to make sure that the URL isn't duplicated in your other feeds.

When the same URL is fed into the system by more than one data source, the system uses the following rules to determine how that content should be handled:

- If the URL is referenced by a web feed and a content feed, the URL's content is associated with the data source that crawled the URL last.
- If the URL is referenced by more than one content feed, the URL's content is associated with the data source that was responsible for the URL's last update.
- If the URL is referenced in the Admin Console's list of **Crawl URLs** and a content feed, the URL's content is associated with the content feed. The search appliance will not recrawl the URL until the content feed requests a change. To return the URL to its original status, delete the URL from the feed that originally pushed the document to the index.
- If the URL has already been crawled by the search appliance, and is then referenced in a web feed, the search appliance immediately injects the URL into the queue to be recrawled as if it were a new, uncrawled URL. The URL's Enterprise PageRank is not affected. However, the change interval is reset to the default until the crawl scheduler process next runs.

Document Status is Stuck “In Progress”

If a document feed gives a status of “In Progress” for more than one hour, this could mean that an internal error has occurred. Please contact Google to resolve this problem, or you can reset your index by going to **Administration > Reset Index**.

Insufficient Disk Space Rejects Feeds

If there is insufficient free disk space, the search appliance rejects feeds, and displays the following message in the feed response:

```
Feed not accepted due to insufficient disk space. Contact Google for Work Support.
```

The HTTP return code is 200 OK, so a program sending a feed should check the message text. For more information on response messages, see “How a Feed Client Pushes a Feed” on page 29.

Feed Client TCP Error

If you are using Java to develop your feed client, you may encounter the following exception when pushing a feed:

```
Java.net.ConnectException: Connection refused: connect
```

Although it looks like a TCP error, this error may reveal a problem in parsing the MIME boundary parameter syntax, for example, missing a ‘--’ before the argument. MIME syntax discussed in more detail here: http://www.w3.org/Protocols/rfc1341/7_2_Multipart.html

Example Feeds

Here are some examples that demonstrate how feeds are structured:

Web Feed

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE gsafeed PUBLIC "-//Google//DTD GSA Feeds//EN" "">
<gsafeed>
  <header>
    <datasource>web</datasource>
    <feedtype>incremental</feedtype>
  </header>
  <group>
    <record url="http://www.corp.enterprise.com/hello02" mimetype="text/plain"></
record>
  </group>
</gsafeed>
```

Web Feed with Metadata

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE gsafeed PUBLIC "-//Google//DTD GSA Feeds//EN" "">
<gsafeed>
  <header>
    <datasource>example3</datasource>
    <feedtype>metadata-and-url</feedtype>
  </header>
  <group>
    <record url="http://www.corp.enterprise.com/search/
employeesearch.php?q=jwong"
      action="add" mimetype="text/html" lock="true">
      <metadata>
        <meta name="Name" content="Jenny Wong"/>
        <meta name="Title" content="Metadata Developer"/>
        <meta name="Phone" content="x12345"/>
        <meta name="Floor" content="3"/>
        <meta name="PhotoURL"
          content="http://www/employeedir/engineering/jwong.jpg"/>
        <meta name="URL"
          content="http://www.corp.enterprise.com/search/
employeesearch.php?q=jwong"/>
      </metadata>
    </record>
  </group>
</gsafeed>
```

Web Feed with Base64 Encoded Metadata

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE gsafeed PUBLIC "-//Google//DTD GSA Feeds//EN" "">
<gsafeed>
  <header>
    <datasource>example3</datasource>
    <feedtype>metadata-and-url</feedtype>
  </header>
  <group>
    <record url="http://example.com/myfeed.html" action="add" mimetype="text/
html">
      <metadata>
        <meta encoding="base64binary" name="cHJvamVjdF9uYW1l"
content="Y2lyY2xlZ19yb2Nrcw==">/>
      </metadata>
    </record>
  </group>
</gsafeed>
```

Full Content Feed

```
<?xml version="1.0" encoding="UTF8"?>
<!DOCTYPE gsafeed PUBLIC "-//Google//DTD GSA Feeds//EN" "">
<gsafeed>
  <header>
    <datasource>sample</datasource>
    <feedtype>full</feedtype>
  </header>
  <group>
    <record url="http://www.corp.enterprise.com/hello01" mimetype="text/plain"
      last-modified="Tue, 6 Nov 2007 12:45:26 GMT">
      <content>This is hello01</content>
    </record>
    <record url="http://www.corp.enterprise.com/hello02" mimetype="text/plain"
      lock="true">
      <content>This is hello02</content>
    </record>
    <record url="http://www.corp.enterprise.com/hello03" mimetype="text/html">
    <content><![CDATA[
      <html>
        <head><title>namaste</title></head>
        <body>
          This is hello03
        </body>
      </html>
    ]]></content>
    </record>
    <record url="http://www.corp.enterprise.com/hello04" mimetype="text/html">
      <content encoding="base64binary">Zm9vIGJhcgo</content>
    </record>
  </group>
</gsafeed>
```

Incremental Content Feed

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE gsafeed PUBLIC "-//Google//DTD GSA Feeds//EN" "">
<gsafeed>
  <header>
    <datasource>hello</datasource>
    <feedtype>incremental</feedtype>
  </header>
  <group action="delete">
    <record url="http://www.corp.enterprise.com/hello 01"mimetype="text/plain"/>
  </group>
  <group>
    <record url="http://www.corp.enterprise.com/hello02" mimetype="text/plain">
      <content>UPDATED - This is hello02</content>
    </record>
    <record url="http://www.corp.enterprise.com/hello03" mimetype="text/plain"
action="delete"/>
    <record url="http://www.corp.enterprise.com/hello04" mimetype="text/plain">
      <content>UPDATED - This is hello04</content>
    </record>
  </group>
</gsafeed>
```

Python Implementation of Creating a base64 Encoded Content Feed

The following `create_base64_content_feeds.py` script goes through all PDF files under MY_DIR and creates a content feed for each of them that is added to the `base64_pdfs.xml` file. This file can then be used to add the documents that are under MY_DIR to the index.

```
import base64
import os

MY_DIR = '/var/www/files/'
MY_FILE = 'base64_pdfs.xml'


def main():
  files = os.listdir(MY_DIR)

  if os.path.exists(MY_FILE):
    os.unlink(MY_FILE)

  fh = open(MY_FILE, 'wb')
  fh.write('<?xml version="1.0" encoding="utf-8"?>\n')
  fh.write('<!DOCTYPE gsafeed PUBLIC "-//Google//DTD GSA Feeds//EN" "">\n')
  fh.write('<gsafeed>\n')
  fh.write('<header>\n')
  fh.write('\t<datasource>pdfs</datasource>\n')
  fh.write('\t<feedtype>incremental</feedtype>\n')
  fh.write('</header>\n')
  fh.write('<group>\n')

  for my_file in files:
    if '.pdf' in my_file:
      encoded_data = base64.b64encode(open(MY_DIR + my_file, 'rb').read())
      fh.write('<record url="googleconnector://localhost.localdomain/' +
               my_file + '" mimetype="application/pdf">\n')
      fh.write('<content encoding="base64binary">' + encoded_data +
               '</content>\n')
      fh.write('</record>')

  fh.write('</group>\n')
  fh.write('</gsafeed>\n')
  fh.close()
  print 'Writing to file: %s' % MY_FILE

if __name__ == '__main__':
  main()
```

Google Search Appliance Feed DTD

The `gsafeed.dtd` file follows. You can view the DTD on your search appliance by browsing to the `http://<APPLIANCE-HOSTNAME>:7800/gsafeed.dtd` URL.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT gsafeed (header, group+)>
<!ELEMENT header (datasource, feedtype)>
<!-- datasource name should match the regex [a-zA-Z_][a-zA-Z0-9_-]*,
    the first character must be a letter or underscore,
    the rest of the characters can be alphanumeric, dash, or underscore. -->
<!ELEMENT datasource (#PCDATA)>
<!-- feedtype must be either 'full', 'incremental', or 'metadata-and-url' -->
<!ELEMENT feedtype (#PCDATA)>

<!-- group element lets you group records together and
    specify a common action for them -->
<!ELEMENT group ((acl|record)*)>

<!-- record element can have attribute that overrides group's element-->
<!ELEMENT record (acl?,metadata*,content*)>
<!ELEMENT metadata (meta*)>
<!ELEMENT meta EMPTY>
<!ELEMENT content (#PCDATA)>

<!-- acl element allows directly associating acls with a url -->
<!ELEMENT acl (principal*)>
<!ELEMENT principal (#PCDATA)>
```

```
<!-- default is 'add' -->
<!-- last-modified date as per RFC822 -->
<!-- 'scoring' attribute is ignored for content feeds -->
<!ATTLIST group
  action (add|delete) "add"
  pagerank CDATA #IMPLIED>
<!ATTLIST record
  url CDATA #REQUIRED
  displayurl CDATA #IMPLIED
  action (add|delete) #IMPLIED
  mimetype CDATA #REQUIRED
  last-modified CDATA #IMPLIED
  lock (true|false) "false"
  authmethod (none|httpbasic|ntlm|httpsso|negotiate) #IMPLIED
  pagerank CDATA #IMPLIED
  crawl-immediately (true|false) "false"
  crawl-once (true|false) "false"
  scoring (content|web) #IMPLIED >

<!ATTLIST metadata
  overwrite-acls (true|false) "true">

<!ATTLIST acl
  url CDATA #IMPLIED
  inheritance-type (child-overrides|parent-overrides|and-both-permit|leaf-node)
    "leaf-node"
  inherit-from CDATA #IMPLIED>

<!ATTLIST principal
  scope (user|group) #REQUIRED
  access (permit|deny) #REQUIRED
  namespace CDATA "Default"
  case-sensitivity-type (everything-case-sensitive|everything-case-insensitive)
    "everything-case-sensitive"
  principal-type (unqualified) #IMPLIED>

<!ATTLIST meta
  encoding (base64binary) #IMPLIED
  name CDATA #REQUIRED
  content CDATA #REQUIRED>

<!-- for content, if encoding is specified, it should be either base64binary
    (base64 encoded) or base64compressed (zlib compressed and then base64
    encoded). -->
<!ATTLIST content encoding (base64binary|base64compressed) #IMPLIED>
```

Index