

Tracking CPS GUI and API Usage

- Track Usage, on page 1

Track Usage

Use the Audit History to track usage of the various GUIs and APIs.

If enabled, each request is submitted to the Audit History database for historical and security purposes. The user who made the request, the entire contents of the request and if it is subscriber-related (a network ID value), all network IDs are also stored in a searchable field.

Capped Collection

By default, the Audit History uses a 1 GB capped collection in MongoDB. The capped collection automatically removes documents when the size restriction threshold is hit. The oldest document is removed as each new document is added. For customers who want more than 1 GB of audit data, contact the assigned Cisco Advanced Services Engineer to get more information.

Configuration in Policy Builder is done in GB increments. It is possible to enter decimals, for example, 9.5 will set the capped collection to 9.5 GB.

PurgeAuditHistoryRequests

When using a capped collection, MongoDB places a restriction on the database and does not allow the deletion of data from the collection. Therefore, the entire collection must be dropped and re-created. This means that the PurgeAuditHistory queries have no impact on capped collections.

AuditRequests

As a consequence of the XSS defense changes to the API standard operation, any XML data sent in an AuditRequest must be properly escaped even if inside CDATA tags.

For example, <ExampleRequest>...</ExampleRequest>

For more information on AuditType, refer to Cisco Policy Suite Unified API 2.3.0 Guide.

Operation

By default, Audit History is ON but it can be turned OFF.

- `ua.client.submit.audit=true` — property used by Policy Builder and set in /etc/broadhop/pb/pb.conf
- `Submit Requests to Audit Log` — Unified API plug-in configuration in Policy Builder.

Initial Setup

There are three parts to the Audit History:

- Server — database and Unified API
- Policy Builder
- Audit Client — bundle that the Policy Builder uses to send Audit requests

Procedure

Step 1 Start the Policy Builder with the following property:

`-Dua.client.submit.audit=false` (set in `/etc/broadhop/pb/pb.conf`)

Step 2 Add and configure the appropriate plug-in configurations for Audit History and Unified API.

Step 3 Publish the Policy Builder configuration.

Step 4 Start the CPS servers.

Step 5 Restart the Policy Builder with the following property:

```
-Dua.client.submit.audit=true
```

```
-Dua.client.server.url=https://lbvip02:8443/ua/soap
```

or

```
-Dua.client.server.url=http://lbvip02:8080/ua/soap
```

Read Requests

The Audit History does not log read requests by default.

- GetRefDataBalance
- GetRefDataServices
- GetSubscriber
- GetSubscriberCount
- QueryAuditHistory

- QueryBalance
- QuerySession
- QueryVoucher
- SearchSubscribers

The Unified API also has a Policy Builder configuration option to log read requests which is set to false by default.

APIs

All APIs are automatically logged into the Audit Logging History database, except for QueryAuditHistory and KeepAlive. All Unified API requests have an added Audit element that should be populated to provide proper audit history.

Querying

The query is very flexible - it uses regex automatically for the id and dataid, and only one of the following are required: id, dataid, or request. The dataid element typically will be the networkId (Credential) value of a subscriber.



Note Disable Regex. The use of regular expressions for queries can be turned off in the Policy Builder configuration.

The id element is the person or application who made the API request. For example, if a CSR log into Control Center and queries a subscriber balance, the id will be that CSR's username.

The dataid element is typically the subscriber's username. For example, if a CSR log into Control Center and queries a subscriber, the id will be that of CSR's username, and the dataid will be the subscriber's credential (networkId value). For queries, the dataid value is checked for spaces and then tokenized and each word is used as a search parameter. For example, “networkId1 networkId2” is interpreted as two values to check.

The fromDate represents the date in the past from which to start the purge or query. If the date is null, the api starts at the oldest entry in the history.

The toDate represents the date in the past to which the purge or query of data includes. If the date is null, the api includes the most recent entry in the purge or query.

Purging

By default, the Audit History database is capped at 1 GB. Mongo provides a mechanism to do this and then the oldest data is purged as new data is added to the repository. There is also a PurgeAuditHistory request which can purge data from the repository. It uses the same search parameters as the QueryAuditHistory and therefore is very flexible in how much or how little data is matched for the purge.

Note Regex Queries! Be very careful when purging records from the Audit History database. If a value is given for dataid, the server uses regex to match on the dataid value and therefore will match many more records than expected. Use the QueryAuditHistory API to test the query.

Purge History

Each purge request is logged after the purge operation completes. This ensures that if the entire repo is destroyed, the purge action that destroyed the repo will be logged.

Control Center

The Control Center version 2.0 automatically logs all requests.

PurgeAuditHistoryRequest

This API purges the Audit History.

The query is very flexible - it uses regex automatically for the id and dataid, and only one of the following are required: id, dataid, or request. The dataid element typically will be the networkId (Credential) value of a subscriber.

The id element is the person or application who made the API request. For example, if a CSR logs into Control Center and queries a subscriber balance, the id will be that CSR's username.

The dataid element is typically the subscriber's username. For example, if a CSR logs into Control Center and queries a subscriber, the id will be that CSR's username, and the dataid will be the subscriber's credential (networkId value). For queries, the dataid value is checked for spaces and then tokenized and each word is used as a search parameter. For example, “networkId1 networkId2” is interpreted as two values to check.

The fromDate represents the date in the past from which to start the purge or query. If the date is null, the api starts at the oldest entry in the history.

The toDate represents the date in the past to which the purge or query of data includes. If the date is null, the api includes the most recent entry in the purge or query.

Note Size-Capped Database

If the database is capped by size, then the purge request ignores the request key values and drops the entire database due to restrictions of the database software.

Schema

```
<PurgeAuditHistoryRequest>
<key> AuditKeyType </key> [1]
</PurgeAuditHistoryRequest>
```

Example

```
<se:Envelope xmlns:se="http://schemas.xmlsoap.org/soap/envelope/">
  <se:Body>
    <PurgeAuditHistoryRequest xmlns="http://broadhop.com/unifiedapi/soap/types">
      <key>
        <id>username</id>
        <dataid>subscriber</dataid>
        <request>API Name</request>
        <fromDate>2011-01-01T00:00:00Z</fromDate>
        <toDate>2011-01-01T00:00:00Z</toDate>
      </key>
    </PurgeAuditHistoryRequest>
  </se:Body>
</se:Envelope>
```

To purge all CreateSubscriberRequest:

```
<se:Envelope xmlns:se="http://schemas.xmlsoap.org/soap/envelope/">
  <se:Body>
    <PurgeAuditHistoryRequest xmlns="http://broadhop.com/unifiedapi/soap/types">
      <key>
        <request>CreateSubscriberRequest</request>
      </key>
    </PurgeAuditHistoryRequest>
  </se:Body>
</se:Envelope>
```

To purge all CreateSubscriberRequest by CSR:

```
<se:Envelope xmlns:se="http://schemas.xmlsoap.org/soap/envelope/">
  <se:Body>
    <PurgeAuditHistoryRequest xmlns="http://broadhop.com/unifiedapi/soap/types">
      <key>
        <id>csrusername</id>
        <request>CreateSubscriberRequest</request>
      </key>
    </PurgeAuditHistoryRequest>
  </se:Body>
</se:Envelope>
```

To purge all actions by CSR for a given subscriber for a date range:

```
<se:Envelope xmlns:se="http://schemas.xmlsoap.org/soap/envelope/">
  <se:Body>
    <PurgeAuditHistoryRequest xmlns="http://broadhop.com/unifiedapi/soap/types">
      <key>
        <id>csrusername</id>
        <dataid>subscriber@gmail.com</dataid>
        <fromDate>2010-01-01T00:00:00Z</fromDate>
        <toDate>2012-11-01T00:00:00Z</toDate>
      </key>
    </PurgeAuditHistoryRequest>
  </se:Body>
</se:Envelope>
```

QueryAuditHistoryRequest

This API queries the Audit History.

The query is very flexible - it uses regex automatically for the id and dataid, and only one of the following are required: id, dataid, or request. The dataid element typically will be the networkId (Credential) value of a subscriber.

The id element is the person or application who made the API request. For example, if a CSR logs into Control Center and queries a subscriber balance, the id will be that CSR's username.

The dataid element is typically the subscriber's username. For example, if a CSR logs into Control Center and queries a subscriber, the id will be that CSR's username, and the dataid will be the subscriber's credential (networkId value). For queries, the dataid value is checked for spaces and then tokenized and each word is used as a search parameter. For example, "networkId1 networkId2" is interpreted as two values to check.

The fromDate represents the date in the past from which to start the purge or query. If the date is null, the api starts at the oldest entry in the history.

The toDate represents the date in the past to which the purge or query of data includes. If the date is null, the api includes the most recent entry in the purge or query.

Schema:

```
<QueryAuditHistoryRequest>
<key> AuditKeyType </key> [1]
</QueryAuditHistoryRequest>
```

Example:

```
<se:Envelope xmlns:se="http://schemas.xmlsoap.org/soap/envelope/">
  <se:Body>
    <QueryAuditHistoryRequest xmlns="http://broadhop.com/unifiedapi/soap/types">
      <key>
        <id>username</id>
        <dataid>subscriber</dataid>
        <request>API Name</request>
        <fromDate>2011-01-01T00:00:00Z</fromDate>
        <toDate>2011-01-01T00:00:00Z</toDate>
      </key>
    </QueryAuditHistoryRequest>
  </se:Body>
</se:Envelope>
```

To find all CreateSubscriberRequest:

```
<se:Envelope xmlns:se="http://schemas.xmlsoap.org/soap/envelope/">
  <se:Body>
    <QueryAuditHistoryRequest xmlns="http://broadhop.com/unifiedapi/soap/types">
      <key>
        <request>CreateSubscriberRequest</request>
      </key>
    </QueryAuditHistoryRequest>
  </se:Body>
</se:Envelope>
```

To find all CreateSubscriberRequest by CSR:

```
<se:Envelope xmlns:se="http://schemas.xmlsoap.org/soap/envelope/">
  <se:Body>
    <QueryAuditHistoryRequest xmlns="http://broadhop.com/unifiedapi/soap/types">
      <key>
        <id>csrusername</id>
        <request>CreateSubscriberRequest</request>
      </key>
    </QueryAuditHistoryRequest>
  </se:Body>
</se:Envelope>
```

To find all actions by CSR for a given subscriber for a date range:

```
<se:Envelope xmlns:se="http://schemas.xmlsoap.org/soap/envelope/">
  <se:Body>
    <QueryAuditHistoryRequest xmlns="http://broadhop.com/unifiedapi/soap/types">
      <key>
        <id>csrusername</id>
        <dataid>subscriber@gmail.com</dataid>
        <fromDate>2010-01-01T00:00:00Z</fromDate>
        <toDate>2012-11-01T00:00:00Z</toDate>
      </key>
    </QueryAuditHistoryRequest>
  </se:Body>
</se:Envelope>
```

Policy Builder

The Policy Builder automatically logs all save operations (Publish and Save to Client) to the Audit History database and also to a log file.

- Policy Builder Publish submits an entry to the Audit Logging Server (goes to database).
- Policy Builder Save to Client Repository submits an entry to the Audit Logging Server (goes to database).
- Whenever a screen is saved locally (Save button) XML is generated and logged for that user in `/var/log/broadhop/qns-pb.log`.

Example log in `qns-pb.log` from Local Save in Policy Builder:

```
2013-02-06 11:57:01,214 [UIThread [vt75cjqhk7v4noguyc9c7shp]] DEBUG
c.b.c.r.BroadhopResourceSetAudit -
Audit: Local file change made by: broadhop. Updated File:
file:/var/broadhop/pb/workspace/tmp-ITC2/checkout/ConfiguredExtensionPoint-43730cd7-b238-4b29-a828-d9b4
47e5a64f-33851.xmi
```

XML Representation of changed screen:

```
<?xml version="1.0" encoding="UTF-8"?>
<policy:ConfiguredExtensionPoint xmlns:policy="http://broadhop.com/policy"
id="43730cd7-b238-4b29-a828-d9b447e5a64f-33851">
  <extensionPoint
    href="virtual:URI#_vxG4swK1Ed-M48DL9vicxQ"/>
  <policies
    href="Policy-default-_sY__4L_REeGCdakzuzzlAg.xmi#_sY__4L_REeGCdakzuzzlAg"/>
</policy:ConfiguredExtensionPoint>
```

Controlling Local Save output:

In the logback.xml file that controls Policy Builder logging, add `com.broadhop.client.resourceset.BroadhopResourceSetAudit` as a category and set it to the desired level.

Reporting

For reporting purposes the following is the database structure in Mongo:

```
{
  "_id" :
  ObjectId("5097d75be4b0d5f7ab0d90fe"),
  "_id_key" :
```

```
  "username",
  "comment_key" :
  "comment",
"data_id_key" : [
  "networkId11921" ],
"timestamp_key" :
  ISODate("2012-11-05T15:12:27.673Z"),
  "request_key" :
  "DeleteQuotaRequest",
  "data_key" :
"<DeleteQuotaRequest><audit><id>username</id></audit><networkId><![CDATA
[networkId11921]]></networkId><balanceCode>DATA</balanceCode><code>Recurring</code>
<hardDelete>false</hardDelete></DeleteQuotaRequest>
"}
```

The following table describes the various Reporting Keys.

Table 1: Reporting Keys

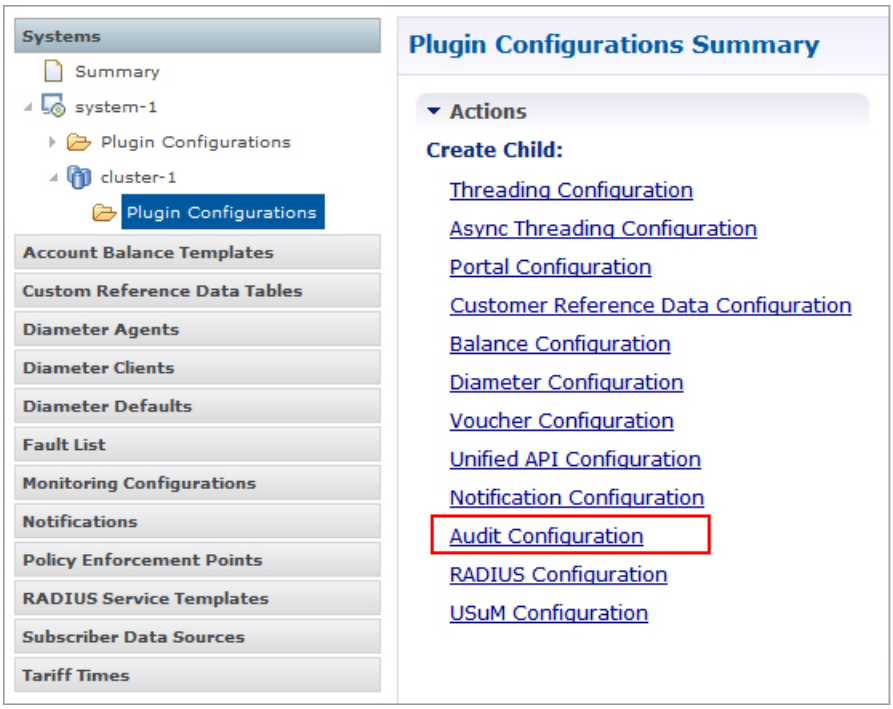
Field	Description
_id	The database unique identifier.
_id_key	the username of person who performed the action. In the above example the CSR who issued the debit request.
comment_key	Some description of the audit action.
data_id_key	The credential of the subscriber. It is a list and so, if the subscriber has multiple credentials, then they will all appear in this list. Please note that, it is derived from the request data and so, for a CreateSubscriber request, there may be multiple credentials sent in the request and each will be saved in the data_id_key list. In the DebitRequest case, only one credential is listed because the request only has the single networkId field.
timestamp_key	The time the request was logged. If the timestamp value is null in the request then the Audit module automatically populates this value.
request_key	The name of the request. This provides a way to search on type of API request.
data_key	The actual request XML.

Audit Configuration

Procedure

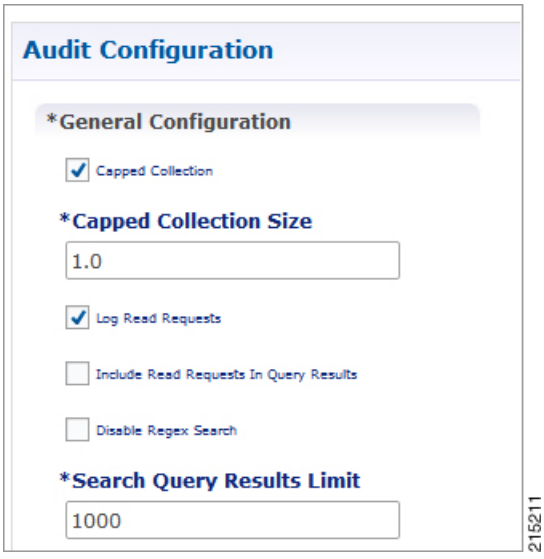
Step 1 Click the **Reference Data** tab, and then click **Systems** > ***system name*** > **Plugin Configurations**.

Figure 1: Plugin Configurations Summary



Step 2 Click **Audit Configuration** in the right pane to open the **Audit Configuration** dialog box.

Figure 2: Audit Configuration dialog box



Step 3 Under **Audit Configuration** there are different panes: **General Configuration**, **Queue Submission Configuration**, **Database Configuration**, and **Shard Configuration**. An example configuration is provided in the following figures:

Figure 3: Queue Submission Configuration pane

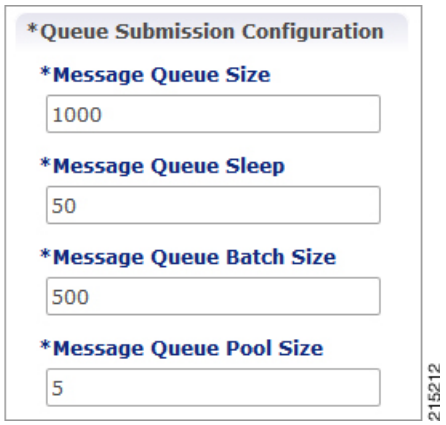


Figure 4: Database Configuration pane

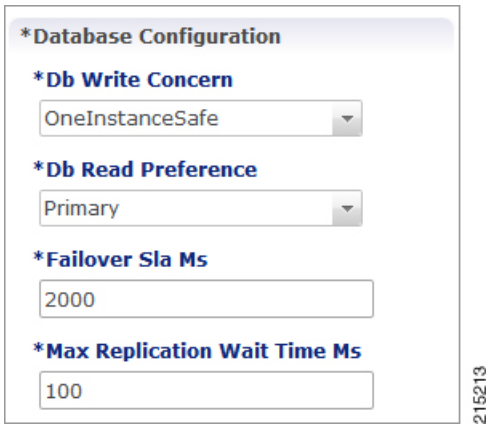
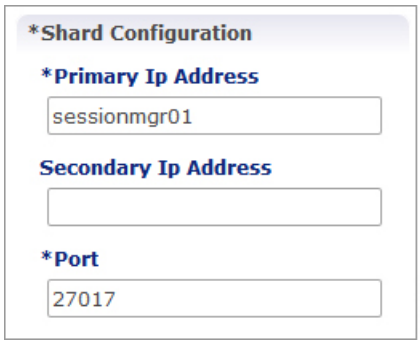


Figure 5: Shard Configuration pane



The following parameters are used to size and manage the internal queue that aids in the processing of Audit messages. The application offloads message processing to a queue to speed up the response time from the API.

Table 2: Audit Configuration Parameters

Parameter	Description
General Configuration	
Capped Collection	Select this check-box to activate capped collection function.
Capped Collection Size	By default, the Audit History uses a 1 GB capped collection in MongoDB. The capped collection automatically removes documents when the size restriction threshold is hit. Configuration in Policy Builder is done in GB increments. It is possible to enter decimals, for example, 9.5 will set the capped collection to 9.5 GB.
Log Read Requests	Select this check-box if you want read requests to be logged.
Include Read Requests in Query Results	Select this check-box only if you want to include read requests to be displayed in query results.
Disable Regex Search	If you select this check-box, the use of regular expressions for queries is turned off in the Policy Builder configuration.
Search Query Results Limit	This parameter limits the search results.
Queue Submission Configuration	
Message Queue Size	Total number of messages the queue can hold at any given time.
Message Queue Sleep	The amount of time for the runnable to sleep between batch processing. The time is in milliseconds.
Message Queue Batch Size	The number of messages to process in a given wake cycle.
Message Queue Pool Size	The number of threads in the execution pool to handle message processing.
Database Configuration	
Db Write Concern	Controls the write behavior of sessionMgr and for what errors exceptions are raised. Default option is OneInstanceSafe.
Db Read Preference	Read preference describes how sessionMgr clients route read operations to members of a replica set. The recommended option is typically Secondary Preferred. http://docs.mongodb.org/manual/core/read-preference/
Failover Sla Ms	This parameter is used to enter the amount of time to wait before starting failover database handling. The time is in milliseconds.

Parameter	Description
Max Replication Wait time Ms	This option specifies a time limit, in milliseconds, for the write concern. This parameter is applicable only if you select TwoInstanceSafe in Db Write Concern. This parameter causes write operations to return with an error after the specified limit, even if the required write concern eventually succeeds. When these write operations return, MongoDB does not undo successful data modifications performed before the write concern exceeded the replication wait time limit. This time is in milliseconds.
Shard Configuration	
Primary Ip Address	The IP address of the sessionmgr node hosting the Audit database.
Secondary Ip Address	The IP address of the sessionmgr node that provides fail over support for the primary database. This is the mirror of the database specified in the Primary IP Address field. Use this only for replication or replica pairs architecture. This field is present but deprecated to maintain backward compatibility.
Port	Enter the Port number of the Audit database as defined in /etc/broadhop/mongoConfig.cfg. The default value in Policy Builder is 27017. For All-In-One deployments, the default Audit database port number is configured as 27017 (no update is needed to this field). For HA or GR deployments, the default Audit database port is 27725. You must update this field to match the Audit database port (27725) or as defined in /etc/broadhop/mongoConfig.cfg.

According to your network requirements, configure the parameters in Audit Configuration and save the configuration.

Pre-configured auditd

In the `/usr/share/doc/audit-version/` directory, the audit package provides a set of pre-configured rules files.

The Linux Audit system provides a way to track security-relevant information on your system. Based on pre-configured rules, Audit generates log entries to record as much information about the events that are happening on your system as possible.

In the `/usr/share/doc/audit-version/` directory, the audit package provides a set of pre-configured rules files.

To use these pre-configured rule files, create a backup of your original `/etc/audit/audit.rules` file and copy the configuration file of your choice over the `/etc/audit/audit.rules` file:

```
cp /etc/audit/audit.rules /etc/audit/audit.rules_backup
cp /usr/share/doc/audit-version/stig.rules /etc/audit/audit.rules
```

For more information on auditd process, refer to the link.