OPENTRUST

Managed File Transfer 3.3.0 Audit Logs Guide

Managed File Transfer 3.3.0 Audit Logs Guide

Release Date: 2015-04-28
Revision: r149906

OpenTrust
175 rue Jean-Jacques Rousseau
CS 70056
92138 Issy-les-Moulineaux Cedex
France
www.opentrust.com

Contents

This page intentionally left blank.

Preface

The following sections contain preface information:

1. Related Documentation

2. Resources

Please use the information provided to contact the appropriate OpenTrust department or representative.

2.1. Contact Support

Support Web Site, including the Support Download Site	`https://support.opentrust.com/` (Login requires a username and password)
Email	`support@opentrust.com`

2.2. Contact Professional Services

Email	`support@opentrust.com`

2.3. Provide Documentation Feedback

As part of an ongoing process to create documentation that is easy to understand and use, as well as relevant to audience roles as administrator users, we welcome feedback about this guide. Please email any comments or suggestions to: `documentation_feedback@opentrust.com`

3. Document Conventions

OpenTrust documentation uses typographical conventions with specific meanings. These conventions are described in the following table.

Convention	How It Is Used
bold	Indicates the most important part of a step in step-based instructions. Example: Click the **OK** button.
italic	Indicates a reference to another document or guide. Example: See the *Release Notes*. Indicates the name of an access right. Example: The *unlock* right allows an administrator to help an end user unlock a smart card.
`monospaced font`	Indicates a file name, directory name or path, code examples and elements, application output, and user-entered text. Example: Save the file in the `/webserver` directory.
`italicized monospaced font`	Indicates an environment-specific or implementation-specific variable. Example: Save the file in the *`root_directory`*`/webserver` directory.
Important:	Contains important information that must be paid attention to. Failure to do so may have a negative impact on the application.
Note:	Contains valuable supplementary information.
Tip:	Contains helpful information that may be useful, for example, a shortcut or another way of performing a task.

This page intentionally left blank.

1 Audit Logs Concepts and Architecture

1.1. Introduction

This guide is for administrators, software consultants, or developers who need to understand the Managed File Transfer audit log system. This guide includes information on the following topics:

- "OTC Audit Logs Architecture Overview" on page 7: General overview of the Managed File Transfer audit logs concepts and architecture.
- "Managed File Transfer Audit Logs SQL Database" on page 11: Structure of the audit log SQL database for developers or software architects planning to use the log module's SQL database for reporting purposes.
- "Sample Audit Events" on page 17: Sample log event message XML.

Descriptions and instructions for how to use the audit log UI are located in the *Server Configuration Guide*.

1.2. OTC Audit Logs Architecture Overview

1.2.1. Audit and Alert Log Streams

In adherence to the design principles set forth in standards such as the Common Criteria[1]Managed File Transfer establishes a clear distinction between *audit events* and *alert events*. Both types of events are considered audit logs in the OpenTrust log system. Audit events record attempted actions and their success status, following a strict data type: each audit event entry must, at a minimum, tell *who* attempted *which action* on *what object*, and *whether it was successful*. In contrast, alert events are essentially free-form. The only constraint placed on alert events is that their name be an uppercase symbol. See "Symbols" on page 11 for more information on symbols. Alert events are used to signal fatal errors requiring an administrator's attention or intervention, such as errors that occur while communicating with an application or an external service such as an LDAP directory, unauthorized access to the application, certain kinds of detected blocking programming errors, and inconsistency between the data source configuration and the actual data found in the data source for a given user, such as a missing field, etc.

Each log entry contains one or more of the following properties:

source information
: Date, title, unique identifier, host name, software module, etc.

actors
: The users involved in the log event. Actors have a role - such as message sender, message recipient, administrator, etc. - to distinguish them in a given event.

details
: A list of named strings conveying information about the event.

attachments
: A software object which is relevant to the given event. The audit logs system supports three types of attachments: software exceptions, certificates, and CRLs.

tracking identifiers
: Alphanumeric tags, unique in a given name space, that allow one to link together related events. For example, log entries arising from the processing of the same message sent will have the same tracking number. A given event may have several tracking identifiers, with different name spaces.

"Sample Audit Events" on page 17 contains a detailed list of the properties associated with each type of audit log entry and explains what kind of information the audit log entries may contain.

[1]The Common Criteria for Information Technology Security Evaluation (abbreviated as Common Criteria or CC) is an international standard (ISO/IEC 15408) for computer security certification. See `http://www.commoncriteriaportal.org` for more information.

1.2.2. Log Workflow

Log messages are created by the application as structured XML messages and stored locally in a queue implemented as a database table on the MFT database server. If the database server is not available at the time of the event[2], the log messages are stored temporarily in a disk queue. The disc queue is the `/opt/opentrust/mft/var/log/xml` directory on all machines for events generated by the OTC framework (startup scripts, audit web interface, etc.), the `/opt/opentrust/mft/var/log/xml/admin` directory for events generated by the administrator application, and the `/opt/opentrust/mft/var/log/xml/user` directory for events generated by the end user application.

Every five minutes, a cron job named `logs-manager send-logs` processes the XML logs stored in the database table queue and stores them in the definitive table using the relational database model described in “Managed File Transfer Audit Logs SQL Database” on page 11.

1.2.3. Signature and Chaining on Logs

To preserve the integrity of the logs, two mechanisms are available: *signature* ensures that logs cannot be modified without being noticed and *chaining* ensures that logs cannot be deleted without being noticed. Chaining consists of adding information about the previous log entry (hash and identifier) when a new log entry is created. The source identifier of this new log is also added to the log entry. Finally, the new log is signed and persisted. To enable chaining, it is necessary to enable signature.

1.2.3.1. Signature

During initialization, the log events signature mode is `asynchronous`; log events are signed when they are sent to the log module by the `logs-manager send-logs` script.

1.2.4. Audit Logs Archiving and Purge

Since secure log storage is a key part of the security of any system, audit logs should be securely stored and then archived on a medium designed for long-term archival. When logs have been securely archived, it may be useful to purge the logs from the active system, to regain storage space.

The audit logs archiving and purge process is handled by the program. The following examples do not specify the complete path of the command for clarity.

1.2.4.1. Log Sending Queue Archiving and Purge

Audit logs stored in XML format in the sending queue can be extracted into a single log file, for archiving. Individual log events are then marked as archived in the database. Only log events successfully processed by **`logs-manager send-logs`** for signature and sending to the log module will be processed by this command.

Run, as the **`root`** or user, the **`logs-manager`** program, with the `archive` command:

```
logs-manager archive --file archive.xml --confirm
```

This command will store the log events **not archived before** into the `archive.xml` xml file, and mark those log events as archived. The system user must have write access to the file. If the **`--confirm`** option is not given, the command will just output the approximate file size of the archive file, and the number of events which would be archived, without marking them as archived.

If audit log signature is active, the XML archive file will be signed to ensure integrity. The signature format is XMLDSig, in enveloped mode.

An optional **`--until "YYYY-MM-DD HH:mm:SS"`** argument may be specified, to archive logs until a given date.

Note:

Since the signing process holds the entire XML archive file in memory, the file size of the archive file should not exceed the size of free memory on the system. Likewise, check the remaining disk space before executing the archive command, to avoid filling up the disk with the archive file. Use the **`--until`** argument to control the size of the archive file.

[2]When the application is starting, the database server may not be started when first audit logs are created.

To actually purge the logs after archival, and delete them from the log sending queue database table, use the **`logs-manager`** program, with the **`purge`** command:

```
logs-manager purge-queue --confirm
```

An optional **`--until "YYYY-MM-DD HH:mm:SS"`** argument may be specified, to purge logs until a given date.

After the audit events have been purged from the queue , it will not be possible anymore to download those audit events in XML format, nor verifying their signature in the log module UI.

Note:

The execution of the **`logs-manager archive`** and **`logs-manager purge-queue`** are logged, and the resulting audit event is not archived or purged until another execution of the command.

1.2.4.2. Log Module Database Purge

The log module database can be purged to reclaim disk space. To purge the log module database, use the **`logs-manager purge-log-database`** command:

```
logs-manager purge-log-database --until "YYYY-MM-DD HH:mm:SS" --confirm
```

The **`--until`** argument is required.

Log events recorded up until the given date are deleted from the log. module database. They will not be available anymore in the log module user interface for searching and viewing.

1.2.4.3. Reclaiming Physical Disk Space

To immediately reclaim the physical disk space after deleting the log events, the following command has to be run, as `root`:

```
                /etc/init.d/mft exec mft-db vacuum
```

This command is run automatically every day at midnight by **`cron`**.

This page intentionally left blank.

2 Managed File Transfer Audit Logs SQL Database

2.1. Understand the Layout of the Audit Logs Database

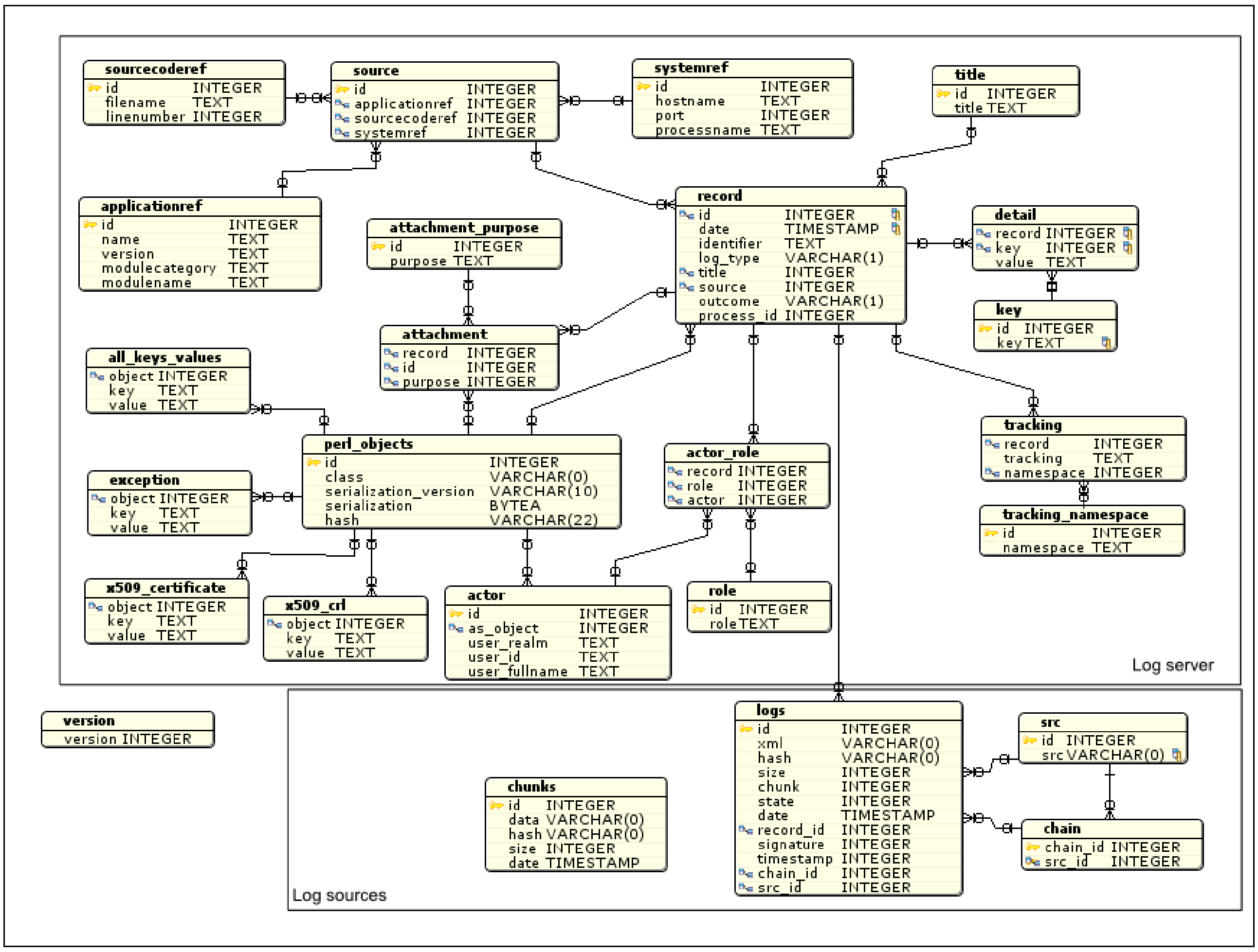
The purpose of this section is to explain what these tables contain and what their relationships mean. This document does *not* explain how to use your reporting tool to perform SQL queries of interest. Refer to the documentation supplied by the reporting tool provider.

2.1.1. Symbols

A number of columns in the tables of the log database are of type TEXT, and documented here as being (uppercase or lowercase) *symbols*. This term is used in the Lisp sense of a tightly syntax-constrained string to chose among a restricted, although extensible set of options to serve as a key into various conceptual-level tables specific to the application. Depending on the context, there may or may not exist a complete documentation of all symbols allowed in a particular symbol column.

2.1.2. Table Overview

Figure 2.1. Logs database schema relational diagram



The relational model of the logs database is shown in Figure 2.1, “Logs database schema relational diagram” on page 11, with related tables grouped together using a color code. The tables in the schema can almost be classified in two categories: the *event tables* are pictured in warm colors on the left-hand side of the diagram, while the *object tables* are on the right-hand side in tones of blue and green. The `all_key_values` table is *sui generis* and is pictured in grey.

2.1.2.1. Event Tables

The most important table in the logs database is the `record` table, featuring one row per log record, for both audit and alert events (see “Audit and Alert Log Streams” on page 7). This table is related to nine ancillary tables by means of foreign-key relationships:

Table names: `title`
Cardinality w.r.t. `record` qtable: one: each record has exactly one title
Information conveyed: This table simply holds the titles of the log entries, as uppercase symbols (see “Symbols” on page 11).
Comments: The choice of storing the titles into a table of their own, as opposed to putting them in a column of the `record` as the normal form rules would have it, is an optimization to save disk space: the log title, which is a string of variable length (for example `CA_CERTIFICATE_SIGN`) is replaced with a fixed-length integer in the `record` table, while its cost in the `title` table is amortized because all log records with the same title will re-use the same row. This denormalization essentially amounts to constructing an extensible record type on top of ordinary SQL by means; such a pattern is re-used throughout the database schema.
Table names: `source`, `sourcecoderef`, `applicationref` and `systemref`
Cardinality w.r.t. `record` qtable: zero or one: all these fields are optional in any log record
Information conveyed: These tables hold the information about the spatial origin of the log entry, in various senses of the word, from network coordinates of the originating program to line number information.
Comments: The `source` table works in the same fashion as the `title` table, making use of a denormalization to save disk space when multiple log entries share the same origin characteristics. In turn, the ancillary tables `sourcecoderef`, `applicationref` and `systemref` serve to fold together common fields in the XML log entries using the same mechanism. The columns are split among the latter three tables in a way that maximises the likelihood of collisions: hopefully several log entries will be created by the same process (thus sharing all information in table `systemref`), the same line of code, and the same application module.
Table names: `detail` and `key`
Cardinality w.r.t. `record` qtable: zero, one or many
Information conveyed: These tables hold free-form key/value information about the recorded event. The Common Criteria mandatory fields are *not* represented as details.
Comments: Keys are lowercase symbols (see “Symbols” on page 11), and they are denormalized in a table of their own for the very same reason as the `titles` table. Conversely, the `detail` table is *not* a denormalization of a field that should go into the `record` table, as the reversed direction of the foreign-key relationship demonstrates: there can indeed be several details for a single log entry, including several details with the same key.
Table names: `tracking` and `tracking_identifier`
Cardinality w.r.t. `record` qtable: zero, one or many
Information conveyed: These tables hold alphanumeric tags that allow one to “connect the dots” between related events. Log entries arising from the processing of the same request will have the same tracking number. Again, using a denormalizing table allows the disk space occupied by such tracking numbers to remain low, at the expense of some write performance.
Comments:
The following non-key columns are defined:
Column: `log_type`
Found in table: `record`
Meaning: A one-character code indicating the type of the log entry: A for audit and L for alert.
Column: outcome
Found in table: `record`
Meaning: A one-character code indicating the result of the log entry: S for success, F for failure and D for damage (case of a failure where all actions could not be properly rollbacked). Only set in audit events, pursuant to Common Criteria.
Column: `process_id`
Found in table: `record`
Meaning: The process ID of the program that created the log entry. Not in `source` because of a lower collision efficiency.
Column: `date`
Found in table: `record`
Meaning: The absolute time at which the logged event occurred.
Column: `identifier`
Found in table: `record`
Meaning: An XML-level unique identifier used to chain log events together. Currently set to a unique string that is otherwise unused.
Column: `filename`

Found in table: `sourcecoderef`
Meaning: The name of the source code file from which the event was logged.
Column: `linenumber`
Found in table: `sourcecoderef`
Meaning: The line number in the source code file from which the event was logged.
Column: `hostname`
Found in table: `systemref`
Meaning: The DNS hostname of the host creating the log entry.
Column: `port`
Found in table: `systemref`
Meaning: The port number the process creating the log entry is listening to. Omitted in case of e.g. a cron job.
Column: `processname`
Found in table: `systemref`
Meaning: The UNIX process name (as seen in **ps**) of the process creating the log entry.
Column: `name`
Found in table: `applicationref`
Meaning: The name of the application creating the log entry, e.g. `OpenTrust PKI`.
Column: `version`
Found in table: `applicationref`
Meaning: The version number of the application creating the log entry, e.g. `4.0.1`.
Column: `modulecategory`
Found in table: `applicationref`
Meaning: The category name of the functional component creating the log
Column: `modulename`
Found in table: `applicationref`
Meaning: The unique identifier of the functional component creating the log among all components of the same `modulecategory`. The only `modulecategory` using this field is `ca`: it will contain the identifier of the given CA.
Column: `key`
Found in table: `key`
Meaning: The key of a free-form detail, as a lowercase symbol (see “Symbols” on page 11) for its format.
Column: `value`
Found in table: `detail`
Meaning: The value of a free-form detail, as a free-form string.
Column: `tracking`
Found in table: `tracking`
Meaning: An identifier encoded as a string that allows one to “connect the dots” between related events. A tracking identifier is expected to be unique among its `namespace`.
Column: `namespace`
Found in table: `tracking_namespace`
Meaning: The name of a domain of related tracking identifiers.

2.1.2.2. Object Tables

The *object tables* are the tables at the right of the diagram. Their content is managed implementation-wise by an extensible object-relational mapping framework (ORM), in such a way that they represent software objects manipulated by the application. Therefore this part of the schema may grow as more log-worthy concepts are modeled in software objects and added into the application.

The `perl_objects` table is the main table. Every loggable object owns exactly one row in this table, and the `id` primary key is also the unique identifier of the object when manipulated by the ORM. `perl_objects` contains only private data and its columns are almost undocumented here; the user of a reporting tool should never feel the need to include this table into a query. However, objects also export public data into the logs database by means of a dedicated table according to their type: Exception objects own exactly one row in table `exception`; likewise, application users, X509 certificates, X509 CRLs, all correspond respectively to one row in tables `actor`, `x509_certificate` and `x509_crl`. The ID of the owned row is the same as the `id` field of their row in `perl_objects`, and this incidentally also applies to record objects modeled primarily in the `record` table; hence the foreign-key relationships going from each of the aforementioned type-specific tables into `perl_objects`. This layout, typical of object-oriented ORMs, is of no importance to the user of a reporting system who can ignore the `perl_objects` table and its relationships.

From a conceptual vantage point, the audit log system attaches objects to log entries in one of two fashions, again according to their types:

- user objects are treated as *actors*, in accordance with the Common Criteria specification and glossary: each auditable action must log what users were involved in the operation, and for which *role* (e.g. beneficiary, validating administrator).

- all other object types are treated as *attachments* that provide optional (albeit useful) information about the event: for example, the text of a newly-created certificate. The joining information in this case is the *purpose* of the attachment: e.g. issued certificate, raised exception, etc.

The join tables `attachment`, and `actor_role` model these relationships that exist between log entries and loggable objects. They are the only two “real” join tables in the whole logs database design, all other foreign-key relationships being either artifacts of the ORM or disk space optimizations.
Table names: `attachment` and `attachment_purpose`
Cardinality: Zero, one or several attachments per log entry; several attachments for the same purpose are allowed.
Information conveyed: `attachment` is a join table that models the relationship “object *o* has purpose *p* with respect to log entry *e*”, where *o* is actually a row in exactly one of the type-specific loggable attachment classes, namely at this time `exception`, `x509_certificate` or `x509_crl`. The purpose *p* is a lowercase symbol (see “Symbols” on page 11) which is denormalized into its own table to save disk space.
Comments:
Table names: `actor`, `actor_role` and `role`
Cardinality: Zero, one or several actors per log entry; several *different* actors for the same role are allowed.
Information conveyed: `actor` is the table that contains nominative information about application users. `actor_role` is a join table that models the relationship “user *a* is acting with role *r* in the action of log entry *e*”, where the log entry is necessarily an audit event. The role is a lowercase symbol (see “Symbols” on page 11) which is denormalized into its own table to save disk space.
Comments:
Table names: `exception`
Cardinality: Zero, one or several exceptions per log entry - the join table being `attachment`
Information conveyed: Models the details of a fatal error that occurred during ans operation, as a free-form key-value table.
Comments:
Table names: `x509_certificate` and `x509_crl`
Cardinality: Zero, one or several certificates and/or CRLs per log entry - the join table being `attachment`
Information conveyed: Models the various security fields of an X509 object of the respective kind. Keys and values are standardized, see below.
Comments:
Several tables in this region of the database schema make use of a key-value system:
Table: `exception`
Acceptable keys: `class` (the class of the exception, e.g. `OpenTrust::Error::LDAP`), `file` and `line` (source-level coordinates of the exception throwing site), and `raw_text` (the whole text of the exception, as it should be reported to an assigned OpenTrust technical representative).
Table: `x509_certificate`
Acceptable keys: `issuer_dn` (the DN of the CA issuing this certificate, as an RFC2253 string), `subject_dn` (the DN of the certificate holder, as an RFC2253 string), `email` (the email address of the certificate holder, as an RFC822 string), and `serial` (the certificate serial number, as an hexadecimal string).
Table: `x509_crl`
Acceptable keys: `issuer_dn` (the DN of the CA issuing this CRL, as an RFC2253 string), `x509_version` (the X509 compatibility level of this CRL, either 1 or 2 as an ASCII decimal string), `last_update`, (respectively `next_update`): a timestamps, in “epoch” seconds, that is, ASCII decimal strings representing the number of non-leap seconds elapsed between midnight of January 1st, 1970 and the last update (resp. next update) of the CRL, represented as decimal strings.
In addition to the various key and value columns, the following non-key columns are defined:
Column: `role`
Found in table: `role`
Meaning: The role played by an actor during some loggable event, as a lowercase symbol.
Column: `user_id`
Found in table: `actor`
Meaning: the user identifier of the user, e.g. an email address, or a DN.
Column: `user_realm`
Found in table: `actor`
Meaning: the realm in which the `user_id` is considered unique, as a free-form string
Column: `user_fullname`
Found in table: `actor`
Meaning: the human-readable name of the actor, as a free-form string. Note that if the actor changes name for whatever reason (e.g. if a woman gets married), an other `actor` record would be created - in other words, the (`user_id`, `user_realm`) pair is *not* an adequate composite key for the `actor` table.
Column: `purpose`
Found in table: `attachment_purpose`
Meaning: The purpose of a given attached object (`exception`, `x509_certificate` or `x509_crl`) with respect to some loggable event, as a lowercase symbol (see “Symbols” on page 11)

2.1.2.3. The `all_keys_values` table

This "one of a kind" table simply contains a copy of all key-value information available in tables `detail`, `key`, `exception`, `x509_certificate` and `x509_crl` thus making it easier to perform type-independent searches.

2.1.2.4. Chaining Tables

Chaining tables are three tables used to proceed logs chaining. The most important table for logs chaining is the `logs` table, featuring one row per XML log entry. To proceed chaining, this table is related to two other tables: `src` and `chain`.
Table names: `src`
Cardinality w.r.t. `logs` qtable: zero or one: this field is empty if chaining is disabled, but mandatory if chaining is enabled.
Information conveyed: This table simply holds chaining sources strings.
Comments: The chaining sources are stored into a table of their own to save disk space: they are just referenced by their shorter identifier in the `logs` table as opposed to putting their value in a column of their own in this table.
Table names: `chain`
Cardinality w.r.t. `logs` qtable: zero or one: this field is empty if chaining is disabled, but mandatory if chaining is enabled.
Information conveyed: This table indicates the last chained log identifier for each source.
Comments: The choice of storing the last chained log identifier into a table of its own, as opposed to putting them in a column of the `logs` as is standard, is an optimization to improve database queries when requesting the last chained log.

Appendix A. Sample Audit Events

A.1. Sample XML Messages

A.1.1. Audit Events

Sample audit trail of a successful certificate revocation. Five audit events are recorded:

Authentication: An authentication of a user on the end user application.

```
<?xml version="1.0" encoding="UTF-8"?>
<auditrecord date="2010-11-23T15:48:41Z" id="mftdemo.opentrust.com-11112-20101123-154841-65-0"

  status="success">
  <source>
    <systemlocation hostname="mftdemo.opentrust.com " processid="11112"/>
    <applicationlocation application="OpenTrust MFT" applicationversion="2.0.0"
      modulecategory="USER" modulename="user"/>
  </source>
  <causality>
    <tracking identifier="53185" namespace="user"/>
    <principal domain="MFTDomain:MFT" name="John Smith" role="user" uid="jsmith"/>
  </causality>
  <title>AUTHENTICATION</title>
  <details key="authentication_scheme">mft_internal_auth_scheme</details>
  <details key="request_url">https://mftdemo.opentrust.com.dev.opentrust.com/zephyr/connect
    </details>
  <details key="credential_ident">jsmith</details>
  <details key="remote_client">172.18.16.50</details>
</auditrecord>
```

Message Sent: A user sends a message with a file upload to another user.

```
<?xml version="1.0" encoding="UTF-8"?>
<auditrecord date="2010-11-23T15:49:14Z" id="mftdemo.opentrust.com-11112-20101123-154914-75-0"

  status="success">
  <source>
    <systemlocation hostname="mftdemo.opentrust.com" processid="11112"/>
    <applicationlocation application="OpenTrust MFT" applicationversion="2.0.0"
      modulecategory="USER" modulename="user"/>
  </source>
  <causality>
    <tracking identifier="614246" namespace="message"/>
    <principal domain="MFTDomain:MFT" name="John Smith" role="message_sender" uid="jsmith"/>
    <principal domain="MFTDomain:MFT" name="Robert Doe" role="message_recipient" uid="rdoe"/>
  </causality>
  <title>MESSAGE_SENT</title>
  <details key="file">recipies.odt (29377 bytes)</details>
  <details key="message_subject">Recipes</details>
  <details key="message_sending_policy">Default Sending Policy</details>
  <details key="message_lifetime">7</details>
  <details key="upload_mode">standard</details>
  <details key="message_upload_duration">0 (message was queued)</details>
</auditrecord>
```

Mail Sent: A mail notification is sent to a user.

```
<?xml version="1.0" encoding="UTF-8"?>
<auditrecord date="2010-11-23T15:51:36Z" id="mftdemo.opentrust.com-11112-20101123-155136-73-9"

  status="success">
  <source>
    <systemlocation hostname="mftdemo.opentrust.com" processid="11112"/>
    <applicationlocation application="OpenTrust MFT" applicationversion="2.0.0"
      modulecategory="USER" modulename="user"/>
  </source>
  <title>MAIL_SENT</title>
  <details key="mail_returnpath">john.smith@oexample.com</details>
  <details key="mail_subject">[OpenTrust Zephyr] New message: Recipes</details>
  <details key="mail_sender">john.smith@example.com</details>
  <details key="mail_recipient">robert.doe@example.com</details>
  <details key="mail_profile_name">Message sent to a registered user</details>
</auditrecord>
```