Oracle® Audit Vault and Database Firewall

Developer's Guide

Release 20
E93410-09
June 2022

Oracle Audit Vault and Database Firewall Developer's Guide, Release 20

E93410-09

Primary Author: Karthik Shetty

Contributors: Sumanth Vishwaraj, Rajesh Tammana, Mahesh Rao, Prabhu Sahoo, Sourav Basu, Vipin Samar

Contents

4 Java-Based Audit Trail Collection Plug-ins

List of Examples

List of Figures

List of Tables

Preface

Oracle Audit Vault and Database Firewall Developer's Guide explains how to develop Audit Collection Plug-ins for Oracle Audit Vault and Database Firewall.

Audience

Oracle Audit Vault and Database Firewall Developer's Guide is intended for developers who want to develop Audit Collection Plug-ins.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Documents

See Oracle Audit Vault and Database Firewall 20.1 Books.

Oracle Technology Network (OTN)

You can download free release notes, installation documentation, updated versions of this guide, technical reports, or other collateral from the Oracle Technology Network (OTN). Visit

http://www.oracle.com/technetwork/index.html

For security-specific information on OTN, visit

http://www.oracle.com/technetwork/topics/security/whatsnew/index.html

For the latest version of the Oracle documentation, including this guide, visit

http://www.oracle.com/technetwork/documentation/index.html

Oracle Audit Vault and Database Firewall Specific Sites

For OTN information specific to Oracle Audit Vault and Database Firewall, visit

http://www.oracle.com/technetwork/database/database-technologies/audit-vault-and-database-firewall/documentation/index.html

My Oracle Support

You can find information about security patches, certifications, and the support knowledge base by visiting My Oracle Support:

https://support.oracle.com/

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Translation

This topic contains translation (or localization) information for Oracle AVDF User Interface and Documentation.

The Web based User Interface or the Audit Vault Server console is translated and made available in the following languages. This includes the User Interface, error messages, and help text.

- French
- German
- Italian
- Japanese
- Korean
- Spanish
- Portuguese - Brazil

- Chinese - Traditional
- Chinese - Simplified

Oracle AVDF Documentation is available in the following languages:

- English
- Japanese

Changes in This Release for Oracle Audit Vault and Database Firewall

Review the changes made for development features in Oracle Audit Vault and Database Firewall.

Changes In Oracle Audit Vault and Database Firewall Release 20

New features in Oracle Audit Vault and Database Firewall release 20.

New Features in Oracle AVDF 20.4

CSV format support for audit collection. Refer to the following sections for complete information:

- CSV File Collection Plug-ins
- Schema For CSV File Collection Plug-in Mapper File
- CSV File Collection Plug-in Example

New Features in Oracle AVDF 20.1

- Introduced custom collector to collect audit data from JSON files. See the following sections:
  - JSON File Collection Plug-ins
  - Schema For JSON File Collection Plug-in Mapper File
  - JSON File Collection Plug-in Example
- Introduced custom collector to collect audit data from Basic Authentication based REST services with JSON data format. See the following sections:
  - JSON REST Collection Plug-ins
  - Schema For JSON REST Collection Plug-in Mapper File
  - JSON REST Collection Plug-in Example
- Introduced a new element `ComplexName`. See sections Database Table Collection Plug-in Mapper File and Creating a Database Table Mapper File for complete information.
- Introduced schema files for collection plug-in mapper files. See Schemas for complete information.

1

What is Oracle Audit Vault and Database Firewall

Learn about Oracle Audit Vault and Database Firewall software (Oracle AVDF), and about collection plug-ins.

1.1 Overview of Oracle Audit Vault and Database Firewall

Learn about Oracle Audit Vault and Database Firewall components, and what each component does.

Oracle Audit Vault and Database Firewall (Oracle AVDF) supports native database audit data collection and network-based SQL monitoring to deliver a comprehensive Database Activity Monitoring solution. It is comprised of these components:

- Audit Vault Server: A server that contains an embedded Oracle Database and other software components that manage the activities of Oracle Audit Vault and Database Firewall.
- Audit Vault Agent: A Java component that runs on a remote host and manages the collection of audit information based on commands from the Audit Vault server. The agent interfaces with the collection plug-ins under its control to gather audit records and sends it to the Audit Vault Server.
- Database Firewall: The Database Firewall is a dedicated server that runs the Database Firewall software. Each Database Firewall monitors SQL traffic on the network from database clients to target databases. The Database Firewall then sends the SQL data to the Audit Vault Server to be analyzed in reports.

Oracle Audit Vault and Database Firewall ships with several prepackaged collection plug-ins, which are software programs that know how to access and interpret audit data from target systems of various types. Collection plug-ins collect audit data from an audit trail generated by a target system and store it in an Audit Vault Server repository. Each collection plug-in is specific to a particular type of trail from a particular type of target. These collection plug-ins collect data from databases such as Oracle, SQL Server, Sybase ASE, and DB2.

> **See Also:**
>
> *Oracle Audit Vault and Database Firewall Administrator's Guide*

1.2 How Oracle Audit Vault Server and Agent Work

Audit Collection Plug-ins retrieve audit data in the form of audit trails, which are sequences of audit records.

Audit Collection Plug-ins retrieve audit data in the form of audit trails, which are sequences of audit records. Audit trails are generated by different target types and stored in database tables or XML audit records.

A target can write one or more audit trails; each audit trail is stored in a separate location, and can have its own format.

To elaborate a little on these terms:

- Target

  A target is a software component which generates an audit trail. A target is an instance of a target type and has specific properties such as connection credentials and trail types.

- Target Type

  A target type represents a collection of a particular type of target that generates the same type of audit data. Oracle Database, for example, is a target type which can have many instances. However, all Oracle Databases generate the same audit data and record the same fields.

- Audit Trail

  An Audit Trail identifies a location and format where audit data resides. Each audit trail is generated by one and only one target. Examples of audit trails are:

  – For targets that write data into files, the trail is the directory path plus the file mask.

  – For targets that write audit data into a database table, the name of the table is the trail for that target. `Unified_Audit_Trail` is an example of a database table audit trail in an Oracle database.

1.3 Types of Audit Collection Plug-Ins

Learn what audit collection plug-ins are, which audit collection plug-ins you should use for your audit trails, and what Java-based collection plug-ins you can use with Oracle Database Vault.

1.3.1 What Are Audit Collection Plug-ins?

Learn about audit collection plug-ins.

Audit collection is supported from many database types. See Product Compatibility Matrix for a list of supported database types and versions.

In case audit collection is not supported out of the box from a specific database type, then you can build custom audit collection plug-in to retrieve audit data stored in the audit trails.

A **collection plug-in** provides functionality similar to the prepackaged collection plug-ins shipped with Oracle Audit Vault and Database Firewall, by retrieving audit data stored in audit trails.

Oracle Audit Vault and Database Firewall allows developers and third-party vendors to build custom collection plug-ins. These custom plug-ins are capable of collecting audit data from a new **target type**.

You can write collection plug-ins that collect audit trails stored in database tables and XML files, or that are accessible in any other way.

You can support targets, such as relational databases, operating systems, mid-tier systems, or enterprise applications.

To obtain more individualized audit data, you can create custom collection plug-ins, and deploy them into existing Oracle Audit Vault and Database Firewall installations.

Related Topics

- Overview of Oracle Audit Vault and Database Firewall
  Learn about Oracle Audit Vault and Database Firewall components, and what each component does.

1.3.2 About Oracle AVDF Plug-In Types

You can create two types of collection plug-ins for Oracle AVDF. The actual type that you need to create depends on the properties of the audit trail that you want to collect.

To describe the audit data being collected, you create an XML file, called a **mapper** file, for the collection plug-in to use. Oracle Audit Vault Server uses this file to access and interpret the audit records being collected. You do not need to write code for this type of plug-in.

There is also a Java-based type of collection plug-in, which uses a Java API. You can design your own Java-based collection plug-in, or you can use one that is prepackaged with Oracle Audit Vault and Database Firewall.

1.3.3 Determining Which Audit Collection Plug-in Type to Create

The audit collection plug-in that you use depends on the type of audit trail that you are collecting for Oracle Audit Vault and Database Firewall.

You can easily define a mapper file (template) and a collection plug-in if the audit trails you wish to collect are stored in either of the following:

- Database Tables: Stored in database tables that conform to specific constraints
- XML/JSON/CSV Files: Stored in XML/JSON/CSV files based on the Oracle AVDF XML Audit File format
- REST: REST data source that generates data in JSON format.

Related Topics

- Database Table Collection Plug-in Example
  See examples of Oracle Audit Vault database table collection plug-in mapper files and database table plug-in manifest files.
- XML File Collection Plug-in Examples
  Learn about the plug-in mapper file and plug-in manifest file attributes and fields for Oracle Audit Vault and Database Firewall.

1.3.4 Java-Based Collection Plug-ins

When the audit trail you need to collect is not in a format that a Collection plug-in can easily read, you write Java-based collection plug-ins in Java code.

Using the Java API provided, you can write code to collect these more complex audit trails and send them to the Audit Vault Server repository.

1.4 Audit Vault Server Events and their Attributes

Oracle AVDF monitors the stream of events that occur in target systems.

1.4.1 About Audit Vault Server Events and Attributes

Learn about Audit Vault Server events, fields, and audit records.

Monitoring the activity, the **stream of events**, that occur in a target system is the essence of Oracle Audit Vault and Database Firewall. These events are described by **fields**. A collection of fields describing a single event that occurred on the target system is an **audit record**.

The following applies for Oracle Audit Vault and Database Firewall:

- Each target logs events as audit events that occur on that target. Audit records capture information about audit events.
- Audit records typically have a target type **event name** that describes what happened to what type of object. They also contain the **target** of the action that happened. In addition, they must contain a time when the action occurred, the subject, or actor, who caused the action to happen, and may also contain additional data.

  Audit Vault Server organizes the fields of an audit record into these groups: core fields, extension fields, large fields, and marker fields.

1.4.2 Understanding Core Fields

Learn what core fields are, and what their purpose is with Oracle Audit Vault and Database Firewall actions.

Core fields are the fundamental fields that describe an event, and most audit records contain some or all of these fields. However, not all core fields are required in every audit record.

Starting with Oracle Audit Vault and Database Firewall release 12.1.1, the core fields which describe the actions occurred are:

- `CommandClass` field: The action that caused the audit record to be generated.
- `UserName` and `OsUserName` fields: The subject or user who performed the action.
- `EventTime` field: When, what time, the action occurred.
- `ClientHostName`, `ClientIp`, and other related fields: Where, what location, of the action.
- `TargetType`, `TargetOwner`, and `TargetObject` fields: The object type, object owner, or target of the action.

> **See Also:**
>
> Core Fields for a complete list of core fields.

1.4.3 CommandClass and Target Types

Learn about the core fields `CommandClass` and `TargetType` in Oracle Audit Vault and Database Firewall.

The `CommandClass` and `TargetType` fields have well-known values, which cover a set of general-purpose events that occur in targets belonging to various domains, such as databases or operating systems.

Some examples of the `CommandClass` values are `Logon`, `Select`, `Update`, and `Shutdown`.

Related Topics

- Audit Vault Server Fields
  You can map Oracle Audit Vault and Database Firewall events and fields in your collection plug-ins.
- Actions and Target Types
  When you build collection plug-ins, you can use the target types and actions that Oracle Audit Vault can detect.

1.4.4 Other Oracle Audit Vault Fields

In addition to core fields, Oracle Audit Vault Server can interpret Large Fields, Marker fields, and Extension Fields

1.4.4.1 About Large Fields

In Oracle Audit Vault and Database Firewall, **large fields** are fields that contain arbitrarily large amount of data.

Large fields are fields that contain arbitrarily large amount of data.

Related Topics

- Large Fields
  In Oracle Audit Vault, **Large fields** are fields that can contain arbitrarily large amounts of data.

1.4.4.2 About Extension Field

In Oracle Audit Vault and Database Firewall, **Extension fields** provide a way to make target fields that do not have a semantically equivalent Oracle Audit Vault field, and do not map to Core or Large fields.

As a developer, you can determine the format used to store extension fields.

Related Topics

- Extension Field
  Extension fields store fields that cannot be accommodated in core or large fields, as name-value pairs, separated by delimiter, inside a single Audit Vault field.

1.4.4.3 About Marker Fields

In Oracle Audit Vault and Database Firewall, **Marker fields** provide unique identifiers of a record in an audit trail.

A marker field is constructed out of one or more fields in an audit record.

Related Topics

- Marker Field
  In Oracle Audit Vault, **marker fields** are fields that uniquely identify a record in a trail.

1.4.5 Storing Audit Records in Audit Vault

When you develop plug-ins for Oracle Audit Vault and Database Firewall, Oracle recommends that you follow Oracle guidelines for storing audit records.

As a plug-in developer, you must map the various events that occur within targets, and their fields, to the various fields allowed by Oracle Audit Vault. If a field in the audit record maps to one of the named fields (core, large, or marker fields) in Audit Vault, then you should map it as such.If a field in the audit record does not map to one of the named fields, then you can map it to an extension field of your choosing.

For the `Action` and `TargetType` Oracle Audit Vault Server fields, see the list of field values. If your audit record maps to one of these values semantically, then Oracle strongly encourages you to use that value. However, you are free to use other values than the Oracle Audit Vault Server fields.

Oracle strongly encourages you to follow these basic guidelines when you store values in Oracle Audit Vault:

- Do not store IDs that reference objects in the target database. Oracle Audit Vault does not have access to these objects. Consequently, values that refer to objects in the target database are meaningless. Instead, store literal names of objects, so that they can be understood by the auditors.
- Follow defined Audit Vault conventions. For example, all the `ACTION` fields and `TARGET TYPE` fields in Oracle Audit Vault have uppercase values. |Oracle recommends that you follow this convention, unless this convention is not applicable to your target type, and would cause the data stored in Oracle Audit Vault to be interpreted incorrectly.
- Map to the values if possible. For example, if `TABLE` exists in the list as a `TargetType`, do not add an audit record with the `TargetType` of `DATABASE TABLE`.

Finally, if you think a field in the audit record of a target merits becoming a core field, then Oracle recommends that you contact Oracle, so that this field can be reviewed and added to the model appropriately.

Related Topics

- Actions and Target Types
  When you build collection plug-ins, you can use the target types and actions that Oracle Audit Vault can detect.

1.5 The Collection Process

Learn how the Oracle Audit Vault collection process works.

1.5.1 Flow of Collection: User

When you develop a collection plug-in for Oracle Audit Vault and Database Firewall, your process of development and deployment has a consistent flow.

Collection plug-ins proceed through this lifecycle:

Flow of Development

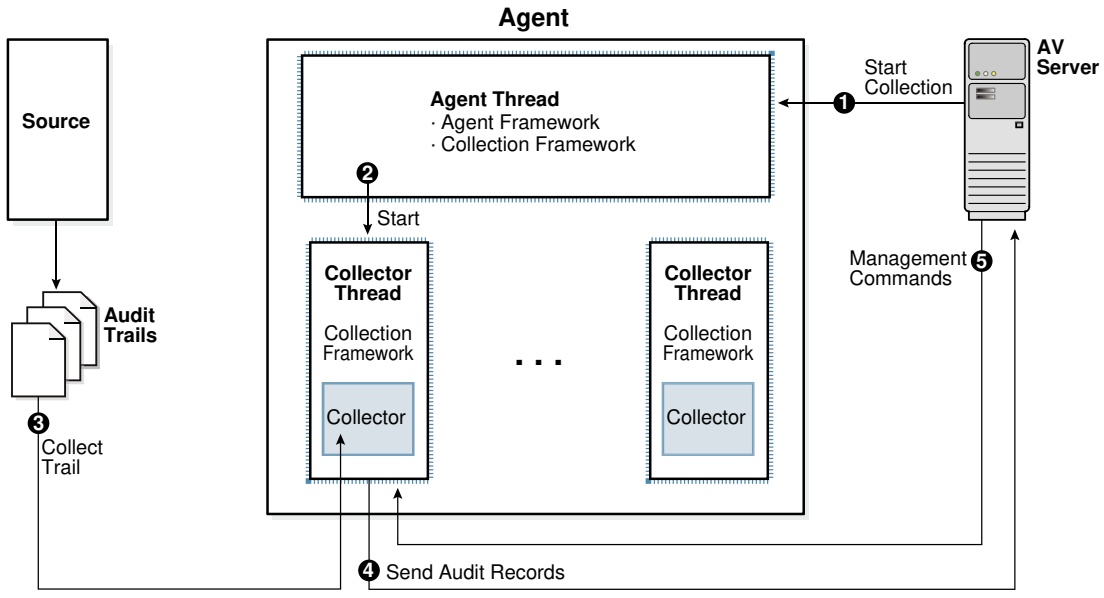
1. You, the developer, create a collection plug-in and provide it to the user.
2. Your user deploys the plug-in into the Oracle Audit Vault Server. The act of deploying a plug-in into the server creates a new version of the Oracle Audit Vault Agent. This new agent contains collector code from the collection plug-in.
3. Your user then deploys the new Agent onto the host where it needs to run.

   From then on, the user can start collecting audit trails supported by the collector code.
4. Your User starts collecting audit trails supported by the collector code.

Flow of Collection

After the user starts collection, the flow of collection proceeds as shown in this diagram.

Figure 1-1 Flow of Collection for Oracle Audit Vault Collection Agents



1.5.2 Flow of Control Inside the Audit Collection Plug-in

After you develop a collection plug-in for Oracle Audit Vault and Database Firewall, the collection plug-in follow this flow as it collects audit trails.

A collection plug-in accesses an audit trail, and extracts an audit record and its related fields from the audit trail. Next, it maps the audit record to an Oracle Audit Vault event, and all the fields to Oracle Audit Vault fields. The collection plug-in then passes the Oracle Audit Vault event and fields to the Audit Vault Agent, which sends the information to the Oracle Audit Vault Server.

Flow of Collection

1. The Oracle Audit Vault Server commands the Agent Framework to create a thread to collect a specific audit trail.
2. The new thread created by the agent collects a specific audit trail, and then turns over control of the thread to the Collection Framework.
3. Within the thread, the Collection Framework connects to the Oracle Audit Vault Server, and queries for configuration information for the audit trail being collected. Additionally, it requests information for the last checkpoint set for that trail. The checkpoint is stored in the Oracle Audit Vault Server in a checkpoint table.
4. With the information it now has, the Collection Framework refers to the plug-in manifest file for the correct Java class to start within the correct collection plug-in. It passes the configuration information to this class, and submits a request to initialize itself.
5. After the the collector has initialized itself, the Collection Framework loops repeatedly. Within each loop, the Collection Framework does the following:
   - Asks the collector for any additional audit records in the audit trail.

     The collector transforms (by mapping) any further audit records into the form of audit records as specified in the mapper file, and hands them to the Collection Framework through the Collection API.
   - The Agent sends the checkpoint information and other metric data received from the collection plug-in to the Audit Vault Server. The Audit Vault Server stores this information in a checkpoint table.
6. If the Audit Vault Server sends commands to the Collection Framework, such as a shutdown command, the Collection Framework passes them to the collector to act on. If the Collection Framework receives a `STOP` command from the Oracle Audit Vault Server, it notifies the collector to stop sending records. It then exits the collection thread, and shuts itself down.

1.5.3 Collection Concepts

To use Oracle Audit Vault, review basic Oracle Audit Valut basic concepts.

1.5.3.1 Collection Thread

Learn about how Oracle Audit Vault collection threads are run.

The Agent starts collection threads. Within each thread, the Audit Vault Collection Framework executes code provided by the collection plug-in. The collection

Framework is the run time infrastructure that exposes the collection API with which the collection plug-in interfaces. The collection plug-in also uses utility APIs if required.

1.5.3.2 Collection Phase

The **collection phase** is the phase in which Audit Vault collects audit trail records.

During the collection phase, the collection plug-in accesses the audit trail to extract new records. The exact mechanism of how audit trails are accessed depends on the audit trail. After a target audit record is retrieved from the trail, the collection plug-in transforms (maps) it into an audit record that can be sent to the Audit Vault Server.

The collection plug-in must also acquire information about the character set of the target records, the encoding used, and issues related to the time stamps. This is done to make these things coordinate with Audit Vault Server requirements.

Related Topics

- Mapping
  The mappings required from targets to Audit Vault Server depends on the fields in the target records.

1.5.3.3 Mapping

The mappings required from targets to Audit Vault Server depends on the fields in the target records.

These types of mapping are required for the Audit Vault Server:

- Event Mapping: Maps a target specific event to an Audit Vault Server event.
- Field Mapping: Maps the various field of the target records to Audit Vault Server fields.
- Value Mapping: Maps various field values collected into a set of normalized values for each field (for example, `0` and `1` may be mapped to `FALSE` and `TRUE` for a specific field).
- Complex Mapping: Complex mappings are used when there are no simple mappings from one target field to an Audit Vault Server field, or one target audit event to one Audit Vault Server event.

  Complex mappings can require additional data from the target, or require additional programming code in the collection plug-in.

> **See Also:**
>
> Configuring Quick JSON Target Type to Collect Audit Data from MongoDB

1.5.3.4 Checkpoint of a Trail

A **Checkpoint**, or a checkpoint of a trail, is the point up to which audit records were committed to the Oracle Audit Vault Server.

The collection plug-in sets a checkpoint periodically so that it can resume from the last checkpoint when restarted.

1.5.3.5 Recovery Phase Of Data Collection

Learn how Oracle AVDF manages recovery of audit records through checkpoints and recovery mechanisms.

Recovery happens when the Agent stops in the middle of collection and has to be restarted. At this point, the recovery process ensures there are no duplicate records in the Audit Vault Server.

Audit Vault Server ensures that every audit record is archived once and only once. For this purpose, Audit Vault Server implements a checkpoint and recovery mechanism.

In the recovery phase of data collection, a collection plug-in has stopped and restarted, resuming collection. The collection plug-in resumes collection from the checkpoint at which it previously stopped

If the collection plug-in has not collected any records from the audit trail, then the checkpoint occurs before the first record. If the collection plug-in has started collecting records and then stopped, then the checkpoint occurs immediately after the last record that it collected.

Resuming collection immediately after the checkpoint ensures that the collection plug-in does not miss any records. To avoid collecting duplicate records during recovery, the collection plug-in checks the **Marker field** of each record.

The collection plug-in should not collect and pass on to the agent any records that occurred before the last checkpoint. However, the Agent automatically filters out records committed after the last checkpoint, and recollected when the collection plug-in restarts. Collection plug-ins built using Oracle Audit Vault and Database Firewall SDK write the `EventTimeUTC` field into a file with the extension `.atc`. A script can subsequently read this file, and delete audit records as appropriate.

Related Topics

- Audit Collection Plug-ins
  Learn about Audit Collection Plug-ins, and how to create them for Oracle Audit Vault.

1.5.3.6 Audit Trail Clean Up

Audit Vault Server collection plug-ins can clean up archived audit trail data for targets.

Audit trail clean up is a feature that some targets provide to clean up audit records after they have been archived. If this type of feature exists in the target, an Audit Vault Server collection plug-in can integrate with it, to tell the target to what extent the audit trail has been archived. Identifying which portion of the audit trail is archived enables the target to clean up the audit trail (remove the original audit data) to that point, because cleaning up data that has been archived results in no loss of data. The collection plug-in gives the clean up utility information about the checkpoint, the point up to which data has been collected.

The collection plug-in can notify the clean-up feature of the target system by invoking the appropriate interface of the feature. For instance, the system may read a timestamp from a file in the file system and clean up the audit trail up to that timestamp. If that is the case, the plug-in can write that file periodically.

For example, Oracle Database targets provides this type of utility in the `DMBS_AUDIT_MGMT` package, and the Oracle Database prepackaged collection plug-ins integrate with it.

1.6 General Procedure for Writing Audit Collection Plug-ins

To ensure that you develop audit collection plug-ins correctly, follow this procedure.

The general procedure for writing collection plug-ins is:

1. Know what capability you want to add to Oracle Audit Vault and Database Firewall, a new target type or a new audit trail for an existing target type.
2. Check if Oracle provides a plug-in that does what you want. If so, then use it.

   Continue only if the plug-in you need does not exist.

   > **Note:**
   >
   > Do not create version-dependent target types. Version-dependent target types are different target types written to use for different versions of the same software. If you create version-dependent target types, then Oracle Audit Vault and Database Firewall is unable to collect from the target after it is upgraded to a different version.
   >
   > For example, suppose that you create the version-dependent target types SQL Server 2000 and SQL Server 2005 and a collection plug-in that collects the audit trail from a SQL Server 2000 target. If you upgrade that target to SQL Server 2005, then Oracle Audit Vault agent cannot collect its audit trail.

3. Understand the events that your target type writes and their fields.

   Use appropriate existing events and fields when you write your plug-in (for examples of existing events and fields. If the events or fields you need are not available, you can use extension fields. Oracle, from time to time, evaluates the set of fields that Audit Vault supports, and may add new fields if they apply to a broad set of target types. If you believe your fields satisfy this criterion, please contact Oracle Support.
4. Decide which type of collection plug-in to write.
5. Set up the development environment.
6. Learn more about the type of plug-in you are creating.

   Refer to "Java-Based Audit Trail Collection Plug-ins" for information about Java-based plug-ins
7. Determine the following for your collection plug-in:
   - How to connect to the target.
   - How to interrogate the target to learn what you must know.
   - Which platforms your plug-in will support.
8. Decide whether your plug-in will support audit trail cleanup.
9. Set up the collection plug-in parameters.

   For Java-based plug-ins, write the relevant code.
10. Create a plug-in manifest file to describe the collection plug-in.

11. Run the `avpack` utility to package the plug-in.
12. Test the plug-in the staging environment.
13. If the plug-in works, then make it available to the Oracle Audit Vault administrator to deploy in the development environment, using the command-line commands.

> **See Also:**
>
> - *Oracle Audit Vault and Database Firewall Administrator's Guide* to see the existing plug-ins offered.
> - About Extension Field
> - Types of Audit Collection Plug-Ins
> - Audit Vault Server Fields
> - Setting Up Your Development Environment
> - Audit Collection Plug-ins
> - Audit Trail Clean Up
> - Packaging Audit Collection Plug-ins
> - Packaging Audit Collection Plug-ins
> - Testing Audit Collection Plug-ins
> - *Oracle Audit Vault and Database Firewall Administrator's Guide* for information on command-line commands.

Related Topics

- Java-Based Audit Trail Collection Plug-ins
  Oracle Audit Vault and Database Firewall provides a set of Java-based audit trial collection plug-in, which enable you to create custom plug-ins.

2
Setting Up Your Development Environment

Learn about the process of setting up the Oracle Audit Vault Server development environment.

2.1 Before Setting Up the Development Environment

To develop audit collection plug-ins, you must first set up the development environment. This set up provides a consistent environment for developing and testing the collection plug-ins.

Before you set up a developer environment, you must complete the following tasks:

- Obtain and install Oracle AVDF 20: You must have this version so that you can test the collection plug-in execution and determine whether it captures the correct audit records from the target, and makes them available in the server. Also, doing early end-to-end integration tests helps to eliminate any connectivity problems, and other bugs in your code.
- Decide the type of collection plug-in to use.

Related Topics

- Determining Which Audit Collection Plug-in Type to Create
  The audit collection plug-in that you use depends on the type of audit trail that you are collecting for Oracle Audit Vault and Database Firewall.

> **See Also:**
>
> *Oracle Audit Vault and Database Firewall Installation Guide* for more information on installation of Oracle Audit Vault and Database Firewall.

2.2 Setting Up the Development Environment

To set up your development environment, you must first download the Oracle Audit Vault and Database Firewall SDK, and then set up the operating system environment.

To set up your environment for developing collection plug-ins you must follow these steps to download the SDK, and then configure the operating system folders, environment variables, and paths.

To download the SDK, do the following:

1. Log in to the Audit Vault Server console as an *administrator*.
2. Click the **Settings** tab.
3. Click the **System** tab in the left navigation menu.
4. Click the **Plug-ins** link under the **Monitoring** section on the main page.

5. In the **Plug-ins** dialog, click **Download SDK**.
6. Unzip the SDK into an empty directory.
1. Set the `AV_SDK_HOME` variable to the directory to which you extracted the SDK.

   For example:

   ```
   $ export AV_SDK_HOME=/home/username/avsdk
   ```

2. Set the `PATH` environment variable to `bin` directory of the Audit Vault Server.

   For example:

   ```
   $ export PATH=$AV_SDK_HOME/bin:$PATH
   ```

   This setting enables you to use existing scripts during the development cycle.

3. Set the `CLASSPATH` environment variable to include the `$AV_AGENT_HOME/av/jlib/agentre.jar` for your collection plug-in project.

   For example:

   ```
   $ export CLASSPATH=$AV_SDK_HOME/av/jlib/av.jar:$AV_SDK_HOME/av/
   jlib/av-common.jar:$AV_AGENT_HOME/av/jlib/agentre.jar
   ```

4. Create directories as necessary.
5. If you are using Java, ensure that the environment is set to point to the appropriate JDK (`PATH` and `JAVA_HOME` variables).

   Compile your classes with `JDK` by setting the `-target` option of the `javac` compiler to the same version. Refer to the `JDK` documentation for details.

Related Topics

- Audit Collection Plug-in Directory Structure
  Learn about the Oracle Audit Vault collection plug-in directory structure, the development environment, and how to stage plug-in `manifest.xml` files.

2.3 Audit Collection Plug-in Directory Structure

Learn about the Oracle Audit Vault collection plug-in directory structure, the development environment, and how to stage plug-in `manifest.xml` files.

2.3.1 General Directory Structure

To create your own collection plug-ins, review the general directory structure for Oracle Audit Vault collections.

The following figure shows a general directory structure.

Example 2-1 General Directory Structure

```
STAGE_DIR_ROOT
plugin-manifest.xml
      templates
          mapper.xml
       jars
          mycoll.jar
          myjdbc-lib.jar
```

```
config
   mycoll.properties
bin
   mycoll.exe
patches
   p3653288_GENERIC.zip
```

Explanation of General Directory Structure Components

In the example of a general directory structure, the `STAGE_DIR_ROOT` directory is the root directory where you stage your collection plug-in files. Place the `plugin-manifest.xml` directly in this directory. Under the `STAGE_DIR_ROOT` directory, create the following directories:

- `jars`: Holds all the binaries generated through the Java build process.

  Place your collector binaries for a Java-based plug-in in the `jars` directory. You should package the various collector Java classes into a jar file for easier access on the file system. For collection plug-ins, you do not need to package the `Collector.jar` in to this directory because it is part of the core agent and is automatically available for all collectors that are managed by an agent.

- `config`: Holds any configuration files that the collection plug-in requires to function. These configuration files can be resource bundles, property files, and so on.

- `bin`: Holds any native non-Java binary executables. For example, if your collector code invokes any native non-Java binaries, place them in the `bin` directory.

  Because the agent is supported on multiple platforms, you should build the non-Java binaries on all platforms that the agent supports. In addition, the collector process locates and loads the appropriate binary based on the execution platform, so use a similar naming convention.

- `patches`: Holds any OPatch patches for target-specified event attributes that the collector needs to function. If your collector adds new event attributes that are needed during run-time, then contact Oracle Support. Oracle Support will provide you with a patch that adds these events into the Audit Vault Server repository. This approval process is necessary to avoid collisions with other event attribute names across multiple plug-ins. After you have obtained these patches, place them in the `patches` directory. Then they will automatically be applied to the server during collection plug-in deployment.

- `templates`: This directory contains the mapper file which has the field mapping from source field to Oracle AVDF field.

Related Topics

- Description of Plug-in Manifest File
  The `plugin-manifest.xml` file is a core XML file that describes the collection plug-in.

2.3.2 Audit Collection Plug-In Directory Structure

Learn about the structure of a stage directory for a collection plug-in for Oracle Audit Vault and Database Firewall.

For a collection plug-in, place all mapper files in the `templates` directory, as shown in this example. This placement directs the collection plug-in to load the relevant template file based on the information that the file contains.

Example 2-2 Directory Structure For Collection Plug-In

```
STAGE_DIR_ROOT
plugin-manifest.xml
       templates
          mycoll-template.xml
       config
          mycoll.properties
       patches
          p3653288_GENERIC.zip
```

Related Topics

- Description of Plug-in Manifest File
  The `plugin-manifest.xml` file is a core XML file that describes the collection plug-in.

2.3.3 Java-Based Collection Plug-in Directory Structure

The structure of Java-based Collection plug-ins for Oracle Audit Vault and Database Firewall is very similar to the general directory structure.

The following example shows the structure of a stage directory for Java-based Collection plug-ins.

Example 2-3 Directory Structure for Java-Based Collection Plug-in

```
STAGE_DIR_ROOT
plugin-manifest.xml
       jars
          mycoll.jar
          myjdbc-lib.jar
       config
          mycoll.properties
       bin
          mycoll.exe
       patches
          p3653288_GENERIC.zip
```

2.3.4 Staging a plugin-manifest.xml File

You must stage the plugin-manifest.xml file directly under the `STAGE_DIR_ROOT` directory.

The `plugin-manifest.xml` file for Oracle Audit Vault and Database Firewall is a core XML file that describes the collection plug-in, and defines its attributes. The location where you place the plugin-manifest.xml file depends on which operating system your server is running.

Locations for Staging a plugin-manifest.xml File

- **On UNIX systems:** If your stage directory is `/opt/final-plugin-stage/`, then stage the `plugin-manifest.xml` file at `/opt/final-plugin-stage/plugin-manifest.xml`.

- **On Microsoft Windows systems:** If your stage directory is `c:\myplugin\final-stage-dir`, then stage the `plugin-manifest.xml` file at `c:\myplugin\final-stage-dir\plugin-manifest.xml`.

> **See Also:**
>
> - Description of Plug-in Manifest File for description and lists of attributes.
> - Example Code for a complete sample file.
> - Sample Schema for a plugin-manifest.xml file

2.3.5 About Mapper Files

Mapper files are XML files that mainly contain information about which target fields you must collect from the audit trail, and how these target fields map to Oracle Audit Vault Server fields.

Mapper files are specific to a target type, and contains target information, such as `securedTargetType`, `securedTargetVersion`, and so on.

Mapper files cover these details:

- The supported target name and target version.
- Mapping information from target fields to Audit Vault Server fields.
- Target fields for constructing markers, which uniquely identify each audit record.
- Audit table and datasource class names, where the audit trail type is database table.
- Event time timestamp format, where the audit trail type is XML file.

Package the mapper files as part of the collection plug-in. Place mapper files in the `templates` folder during the plug-in packaging process.

Related Topics

- Database Table Collection Plug-ins
  To use Oracle Audit Vault to collect audit data from the table type of trail, you can use database table collection plug-ins
- XML File Collection Plug-ins
  Learn how to use Oracle AVDF XML file collection plug-ins to collect audit data from an XML file type of trail.
- Audit Collection Plug-In Directory Structure
  Learn about the structure of a stage directory for a collection plug-in for Oracle Audit Vault and Database Firewall.

2.3.6 Description of Plug-in Manifest File

The `plugin-manifest.xml` file is a core XML file that describes the collection plug-in.

The `plugin-manifest.xml` defines the following elements and attributes:

- The **plugin** element represents the plug-in object with these attributes:
  - **Name**: A descriptive name for the collection plug-in.

  - **version**: The version should be updated along with each update to the collection plug-in, and should monotonically increase based on some ordering scheme. For instance, AVDF uses a versioning scheme comprising of five digits, delimited by periods: `majr.minr.minr.patch.hotfix`.
  - **provider:** The name of the provider. Typically, this name is the company or organization.
  - **copyright**: Any copyright notices for the collection plug-in.
- **TargetVersion**: Oracle Audit Vault and Database Firewall Version with which the collection plug-in is compatible. The **min** attribute represents the minimum version of the target.
- **extensionSet**: A set of `extensionPoint`s.
- **ExtensionPoint:** Each `extensionPoint` uniquely identifies the area of Oracle Audit Vault and Database Firewall (Oracle AVDF) that is being extended by the collection plug-in. Currently, Oracle AVDF supports one Extension Point, `securedTargetType`, as indicated by the `type` attribute.
  - **fileList:** A list of all the files that ship with the collection plug-in.
    * **jars:** A directory that contains Java files ending with the extension `.jar`, in the element `file`.
    * **templates:** A directory that contains the mapper files for a collection plug-in, in the element `file`.
    * **bin:** A directory that contains executable files, typically those that end with `.exe`, in the element `file`.
    * **config:** A directory that contains plug-in specific configuration files, in the element `file`.
    * **shell:** A directory that contains shell or batch command files, in the element `file`.
    * **patch:** A directory that contains event patches for the collection plug-in, in the element `file`.
    * **unresolved-external:** A directory that contains files that cannot be packaged with the collection plug-in for some reason, but are needed at run-time. Packaging succeeds but the plug-in deployment will fail until these files are made available in the `$OH/av/dropins` folder of Oracle Audit Vault Server. These files are in the element `file`.
  - **securedTargetTypeInfo:** This is a mandatory field that indicates the source type that this collection plug-in supports. Specify the source type by filling in the name attribute of this element.
  - **trailInfo:** A mandatory field that indicates the type of audit trails, on this source type, that the collection plug-in supports.
    * **trailType:** A mandatory field that indicates the type of trail described by this entry. Oracle Audit Vault and Database Firewall 12.1.1 supports these trail types: `TABLE`, `DIRECTORY`, `TRANSACTIONLOG`, `SYSLOG`, and `EVENTLOG`.

      `trailType` can also be any arbitrary string. In that case, it is treated as a custom trail type.

* **trailLocation:** Specifies the location of the trail; this is applicable only for `TABLE` and `CUSTOM` type trails only. This field *must not be set* for other trail types. If set for other types, then it is ignored.
* **className:** Specifies the Java class that handles the task of retrieving the audit data from this trail. Use the following:

  - `oracle.av.platform.agent.collfwk.ezcollector.table.DatabaseTableCollector` for database table collection plug-ins.

  - `-oracle.av.platform.agent.collfwk.ezcollector.xml.XMLFileCollector` for XML file collection plug-ins.

  - `oracle.av.platform.agent.collfwk.ezcollector.json.MultiJSONFileCollectorFactory` for JSON file collection plug-ins which reads from JSON files having one fully formed JSON per line.

  - `oracle.av.platform.agent.collfwk.ezcollector.json.JSONFileCollectorFactory` for JSON file collection plug-ins which reads from JSON files having only one fully formed JSON per file, and this single JSON contains an array of JSON records.

    To handle audit trails of different source versions of the same source type, you can optionally set the `srcVersion` attribute.

– **eventPatch:** This is an optional field containing any event patches that must be applied as part of the collection plug-in deployment. These patches are in the `eventPatch` element with the `name` attribute as the file name and an `order` attribute that indicates the order to apply the patches.

  Events attributes to be added are extended through patches generated by Oracle Audit Vault and Database Firewall Development. Partner developers can request specific events and attributes or both, to be added to the Oracle Audit Vault Event dictionary. If the core development team determines that a request is justified, it may issue a patch. You can bundle these patches with the collection plug-in for application during plug-in deployment.

See Also:

- Database Table Collection Plug-in Manifest File
- XML File Collection Plug-In Manifest File
- Java Based Collection Plug-in Manifest File
- External Dependencies

3
Audit Collection Plug-ins

Learn about Audit Collection Plug-ins, and how to create them for Oracle Audit Vault.

3.1 About Audit Collection Plug-ins

Find out about the different types of collection plug-ins: What they do, how they collect audit trails, and where to find out more about them.

Collection plug-ins can retrieve audit data stored in either database tables or XML file audit trails, without the need for writing code.

Collection plug-ins are template-based generalized collectors. Users must provide a mapper file to collect audit data from a trail.

These collection plug-ins are created by preparing an XML Mapper file that supplies the mapping information for target fields to Audit Vault Server fields and other details for target types and audit trails.

This process does not require any coding. Audit Vault contains all the code necessary to interpret Mapper files and use them to collect the audit data from the audit trail appropriately.

Collection plug-ins support two types of audit trails:

- Database Table: database table collection plug-ins can collect audit data from an audit table, using the information from the Mapper file.
- XML File: XML file collection plug-ins can collect audit data from XML audit files present in a single directory, using the information from the Mapper file.

To use the collection plug-in for Database tables or XML file trails, you perform the following steps:

1. Create an XML Mapper file for a target audit trail. This chapter discusses Mapper files in general and focuses on their creation.
2. Create a plugin-manifest file for this target type.
3. Create the collection plug-in by packaging the mapper file and plugin-manifest file.

You can now deploy this collection plug-in at the Audit Vault Server and use it to collect audit data after adding the target and any necessary collection attributes for this target.

See Also:

- Database Table Collection Plug-in Mapper File
- XML File Collection Plug-In Mapper File
- Target Collection Attributes
- Packaging Audit Collection Plug-ins
- Description of Plug-in Manifest File
- Creating a Database Table Mapper File
- Creating the XML File Audit Collection Mapper File

3.2 Database Table Collection Plug-ins

To use Oracle Audit Vault to collect audit data from the table type of trail, you can use database table collection plug-ins

Database table collection plug-ins support the collection of audit data from the table type of trail. They collect audit data from a single audit table. You can specify details of the audit table in the mapper file. These mapper files must conform to the schema.

Related Topics

- Database Table Collection Plug-in Mapper File
  See an example of an Oracle Audit Vault and Database Firewall database table collection plug-in mapper file.

3.2.1 Requirements for Database Table Collection Plug-ins

To use database table collection plug-ins for reading audit trails from target database tables, your data must meet Oracle Audit Vault and Database Firewall requirements.

You can use database table collection plug-ins for reading audit trails from target database tables if your data meets the requirements for collection.

Data Requirements for Table Collection Plug-Ins to Oracle Audit Vault and Database Firewall

- Audit data must be stored in a single database table.
- The target system has a user with privileges to read the audit data stored in this table.
- The columns in the audit tables can be mapped to various Audit Vault core fields and large fields.

  Also single or multiple fields can be mapped to extension and marker fields. Fields mapped to Audit Vault core fields, extension fields, and marker fields must be of `String` data type or convertible to `String`. They cannot be of large data type, such as a `CLOB`. Columns having `CLOB` data type should use large Audit Vault fields, such as `CommandText` or `CommandParam`.
- The audit trail must contain fields which map to the `CommandClass` Audit Vault core fields.

The value of the `CommandClass` core field must not be null. If it is null, then the record is treated as an invalid record, so you must provide the proper mapping.

- The audit file must have a field that can be mapped to the `UserName` core field. If a record has its `UserName` field as null, then the record is treated as invalid.
- The collection plug-in can collect the text of any command issued, as well as any parameters passed to the command, in large fields. No other fields can be mapped to large fields in Audit Vault and Database Firewall.
- The audit trail must contain a single column or group of columns that uniquely identify each audit record.
- The audit trail must contain a field of type `Timestamp` that is monotonically increasing, that is, the value of the field increases with every new audit record inserted into the trail. This field must mapped to the `EventTimeUTC` core field in the mapper file. If, for any audit record, this field value becomes null, the collector treats this as an abnormal condition and shuts down.

Related Topics

- Schemas
  Oracle AVDF uses these schemas for plug-in manifest files and collection plug-ins.

3.2.2 Example Audit Trail for a Database Table Collection Plug-in

This example audit trail shows the details of audit trail. This example file is used in other locations to demonstrate the creation and structure of a sample mapper file for Oracle Audit Vault and Database Firewall.

The following table lists the structure for the hypothetical target type, `DBSOURCE`, that generates and stores audit data in a table `AUD`:

Table 3-1 AUD Audit Table Data Fields and Mappings

Target Field	Data Type	Audit Vault Server Field	Map to Field Type
USER_ID	varchar	UserName	core field
OS_USER_ID	varchar	OSUserName	core field
ACTION	int	CommandClass	core field
STATUS	int	EventStatus	core field
EVENT_TIME	timestamp	EventTimeUTC	core field
OBJ_NAME	varchar	TargetObject	core field
OBJ_CREATOR	varchar	TargetOwner	core field
USER_HOST	varchar	ClientHostName	core field
SQL_TEXT	clob	CommandText	core field
SQL_BIND	clob	CommandParam	core field
TERMINAL	varchar	TerminalName	extension field
DB_ID	varchar	extension field	extension field
INSTANCE	varchar	extension field	extension field
PROCESS	int	extension field	extension field

Table 3-1 (Cont.) AUD Audit Table Data Fields and Mappings

Target Field	Data Type	Audit Vault Server Field	Map to Field Type
SESSION_ID	int	marker field	marker field
ENTRY_ID	int	marker field	marker field

Not all of the target fields map to core fields. The target fields that do not map to core fields map to extension fields, or to designated marker fields, which test the uniqueness of an audit record.

3.2.3 Creating a Database Table Mapper File

Learn how to create an Oracle Audit Vault XML mapper file for a database table collection plug-in, and learn about each XML element and attribute used in this type of mapper file.

See Also:

- Database Table Collection Plug-in Mapper File for the complete example.
- Example Audit Trail for a Database Table Collection Plug-in
- Oracle Audit Vault and Database Firewall Fields for descriptions of all fields.
- Target Collection Attributes to make sure that the mandatory collection attributes are set up for the target.

Example of Creating a Mapper File for Database Table Collection Plug-ins

- Top Level Element

```
<AVTableCollectorTemplate securedTargetType="DBSOURCE"
minSecuredTargetVersion="10.2.0"
           maxSecuredTargetVersion="11.0" version="1.0" >
```

The `AVTableCollectorTemplate` is the top level element, which marks the start of the mapper file. It has these mandatory attributes: `securedTargetType`, `maxSecuredTargetVersion`, and `version`. The `minSecuredTargetVersion` attribute is optional.

The accepted format for the `minSecuredTargetVersion`, `maxSecuredTargetVersion`, and `version` attributes uses numbers, separated by dots, such as 12.2, 10.3.2, 11.2.3.0.

- Table Name Information

```
<TableName>AUD</TableName>
```

You must provide the `TableName` of the audit table. This is a mandatory field.

The `TableName` field in this file must match the trail location in the **Add Audit Trail** screen.

> **Note:**
>
> The collector checks if the trail location matches the `TableName` specified in the mapper file and does not start if they do not match.
>
> The collector chooses the appropriate mapper file in the templates folder by validating the target collection attribute `av.collector.securedtargetversion` version and if it is within the range specified in the top level element `minSecuredTargetVersion` and `maxSecuredTargetVersion` attributes.
>
> For example, if the target collection attribute `av.collector.securedtargetversion` = 11.1.0.0, then the collector picks a mapper file that has a top level element in a range within the specified version:
>
> ```
> <AVTableCollectorTemplate securedTargetType="Oracle Database"
> minSecuredTargetVersion="10.2.0" maxSecuredTargetVersion="12.3"
> version="1.0">
> ```
>
> This enables multiple versions of the same plug in (different mapper files inside the templates folder) to address different target versions.

- Target Connection Information

```
<ConnectionInfo>

   <DataSource>platform.jdbc.dbsource.DBSourceDataSource</DataSource>
</ConnectionInfo>
```

You must provide the full name for the datasource class implementing `javax.sql.DataSource` interface. This is a mandatory field.

- Field Mapping Information

```
<FieldMappingInfo>
```

`FieldMappingInfo` must provide mapping information from target fields to various Audit Vault fields, along with the value transformations if any. This is a mandatory element.

Field mappings include `<Map>` elements, which contain `<Name>` elements that hold target field names, and `<MapTo>` elements that hold Audit Value field names to which targets are mapped.

There should be no many-to-one mappings from target fields to Audit Vault Server fields. For example, the following is invalid:

```
<!-- Invalid code
<Map>
    <Name>USER_ID</Name>
```

```
    <MapTo>UserName</MapTo>
</Map>
<Map>
    <Name>OS_USER_ID</Name>
    <MapTo>UserName</MapTo>
</Map>  -->
```

About Mappings for Core, Large, Extension, and Marker Fields

- The following sections explain mappings for core, large, extension, and marker fields:
  - Core Fields

    ```
    <CoreFields>
    ```

    `CoreFields` provides mapping from target fields to core fields of Audit Vault Server. The data type of target fields specified must belong to either a SQL `string` data type or a data type that can convert to a `String`.

    The following elements contain core fields.

    ```
    <Map>
      <Name>EVENT_TIME</Name>
      <MapTo>EventTimeUTC</MapTo>
    </Map>
    ```

    `EventTimeUTC` provides event time mapping information. It is a mandatory field.

    `EVENT_TIME` target fields must be of the SQL data type `Timestamp`.

    ```
    <Map>
      <Name>USER_ID</Name>
      <MapTo>UserName</MapTo>
    </Map>
    ```

    `UserName` represents the user who performs the action. If the mapping is not provided, Audit Data Collection still starts successfully, but every audit record will be treated as invalid.

    ```
    <Map>
      <Name>OS_USER_ID</Name>
      <MapTo>OSUserName</MapTo>
    </Map>
    ```

    ```
    <Map>
      <Name>ACTION</Name>
    ```

```
  <MapTo>CommandClass</MapTo>
</Map>
```

`CommandClass` represents the action of the event. If the mapping is not provided, Audit Data Collection still starts successfully, but all audit records are treated as invalid.

```
<Transformation>
          <ValueTransformation from="1" to="CREATE"/>
          <ValueTransformation from="2" to="INSERT"/>
          <ValueTransformation from="3" to="SELECT"/>
          <ValueTransformation from="4" to="CREATE"/>
          <ValueTransformation from="15" to="READ"/>
          <ValueTransformation from="30" to="LOGON"/>
          <ValueTransformation from="34" to="LOGOFF"/>
          <ValueTransformation from="35" to="ACQUIRE"/>
        </Transformation>
      </Map>
```

`CommandClass` contains a `Transformation` field with `ValueTransformation` values, from targets to the Audit Vault `CommandClass` field. These transformations are mandatory.

The *to* attributes are values for the `CommandClass` field. If you can meaningfully map an event to one of these values, then Oracle recommends that you do so. If this is not possible, then use a value that appropriately reflects the action that generated the audit event.

```
<Map>
  <Name> OBJ_NAME</Name>
  <MapTo>TargetObject</MapTo>
</Map>


<Map>
   <Name>TERMINAL</Name>
   <MapTo>TerminalName</MapTo>
</Map>


<Map>
  <Name>USER_HOST</Name>
  <MapTo>ClientHostName</MapTo>
</Map>


<Map>
  <Name>OBJ_CREATOR</Name>
```

```
  <MapTo>TargetOwner</MapTo>
 </Map>


<Map>
  <Name>STATUS</Name>
  <MapTo>EventStatus</MapTo>
        <Transformation>
          <ValueTransformation from="0" to="FAILURE"/>
          <ValueTransformation from="1" to="SUCCESS"/>
          <ValueTransformation from="2" to="UNKNOWN"/>
        </Transformation>
 </Map>
```

`EventStatus` contains a `Transformation` field with `ValueTransformation` values, from targets to Audit Vault `EventStatus` fields. These transformations are mandatory.

```
</CoreFields>
```

– Large Fields Information

```
<LargeFields>
  <Map>
    <Name>SQL_TEXT</Name>
    <MapTo>CommandText</MapTo>
 </Map>
<Map>
   <Name>COMMAND_PARAMETER</Name>
   <MapTo>CommandParam</MapTo>
 </Map>
</LargeFields>
```

`LargeFields` are target fields mapped to large fields in the Audit Vault Server, such as `CommandText` or `CommandParam`. The specified target fields must be of SQL data type `CLOB` or `String`, or be convertible to `String`.

– Extension Field

```
<ExtensionField>
   <Name>DB_ID</Name>
   <Name>INSTANCE</Name>
   <Name>PROCESS</Name>
</ExtensionField>
```

The `ExtensionField` is a target field name that must be stored as a name-value pair in the Extension field in Audit Vault Server. Target columns specified here should have a `String` value or a value that can be converted to `String` without loss of information.

ComplexName

```
<ExtensionField>
    <ComplexName>
        <Name>column_name</Name>
        <RegExp>exp</RegExp>
    </ComplexName>
</ExtensionField>
```

`ComplexName` is a tag in the `ExtensionField`.

The `column_name` is an audit table column name which is a string. For example: `comment$text`

`exp` is the regular expression from which we get a list of key value pairs from the text after processing. It should contain 2 groups, out of which one is for key and the other one for value. For example: `([^;]+):([^;]+)`

– Marker Field

```
<MarkerField>
  <Name>SESSION_ID</Name>
  <Name>ENTRY_ID</Name>
</MarkerField>
```

The `MarkerField` contains a list of target field names that uniquely identify each audit record. The target fields specified must be of SQL data type `String` or convertible to `String`. `MarkerField` is mandatory.

– End Tags

The field tags must be properly closed in order for the file to be valid. The following are examples of field end tags:

```
</FieldMappingInfo>

</AVTableCollectorTemplate>
```

See Also:

- Core Fields
- Actions and Target Types
- Large Fields
- Extension Field
- Marker Field

3.3 XML File Collection Plug-ins

Learn how to use Oracle AVDF XML file collection plug-ins to collect audit data from an XML file type of trail.

XML file collection plug-ins support collection of audit data from an XML file type of trail. All these XML audit files must be present in single directory. You can specify details of the XML audit data in the mapper file. This XML mapper file must conform to the schema.

Related Topics

- Schema For XML File Collection Plug-in Mapper File
  See how to set up a schema for an XML file collection plug-in mapper file for Oracle Audit Vault and Database Firewall.

3.3.1 Requirements for XML File Collection Plug-ins

To use XML collection plug-ins for reading audit trails from XML files, your data must meet Oracle Audit Vault and Database Firewall requirements.

You can use collection plug-ins for reading audit trails from XML audit record files if the XML files meet the requirements for collection.

XML File Audit Record File Requirements for Oracle Audit Vault and Database Firewall

- The audit trail must be stored in one or more XML files in a single directory path.
- The user must have read permission on the directory containing the XML audit files.
- XML files in this directory must be valid, well-formed XML documents, within the constraints of the XML 1.0 specification.
- The file and record start elements must be as specified in the mapper file.
- All the audit record elements should be at the same level in Audit XML files.
- All the audit record elements in Audit XML files must be the same.
- Under every audit record element, all the field elements must be at the same level and one level below the audit record start element.
- The XML audit file must have an element value that can be mapped to the `CommandClass` core field. If a record has its `CommandClass` field as null, then the record is treated as invalid.
- The XML audit file must have an element value that can be mapped to the `UserName` core field. If a record has its `UserName` field as null, then the record is treated as invalid.
- In the XML file, each audit record must have a timestamp as one of its element values.

  The value of the timestamp element must be monotonically increasing, that is, the value of the field increases with every new audit record inserted into the trail. The timestamp value should be strictly `Not Null`. Timestamp format must be according to `SimpleDateFormat` Java class.

This field must mapped to the `EventTimeUTC` core field in the mapper file. If mapping for event time is not specified in the mapper file, then the collection plug-in shuts down. If the field value for the event time in audit records is found null, then the collection plug-in takes the time of the record last sent from the same XML audit file.

- The audit trail must contain a single element value or group of element values in the audit record that uniquely identify each audit record in XML Audit files.
- Common information shared by all audit records in XML file should be present in the beginning of the XML file, under the file start element, at the same level as the audit record elements.
- If an audit data target produces audit files with multiple XML formats, then the user must provide a separate mapper file for each audit file format having a different start element.
- XML files in this directory should be of the same locale and encoding as the agent, as described in the examples below:
  - Valid: The user has an agent in a Chinese locale (`env`). XML files are also generated in a Chinese locale with same encoding (for example, `ZHS16GBK`). This setup is valid.
  - Invalid: The user has an agent in a German locale (env). XML files are generated/ moved from some other computer, which are Chinese encoded. The collectors fail to start because of an encoding mismatch, as well as a locale mismatch, in this case. This setup is invalid.

3.3.2 Example Audit Trail for an XML File Collection Plug-in

This example audit trail for an xml file collection plug-in shows the details of an XML file collection plug-in.

This example file is used in other locations to demonstrate the creation and structure of a sample mapper file for the creation and structure of a sample mapper file for an XML file collection plug-in in Oracle Audit Vault and Database Fireewall documentation.

The following table lists the audit record structure and mappings to Oracle Audit Vault Server fields for the hypothetical target type, `XMLSOURCE`, which generates and stores audit data in XML audit files.

Table 3-2 Audit Data Fields in XML Audit Records and Mappings

Target Field	Audit Vault Server Field	Map to Field Type
USER_ID	UserName	core field
OS_USER_ID	OSUserName	core field
ACTION	CommandClass	core field
STATUS	EventStatus	core field
EVENT_TIME	EventTimeUTC	core field
OBJ_NAME	TargetObject	core field
OBJ_CREATOR	TargetOwner	core field
USER_HOST	ClientHostName	core field
SQL_TEXT	CommandText	core field
SQL_BIND	CommandParam	core field
TERMINAL	TerminalName	extension field

Table 3-2 (Cont.) Audit Data Fields in XML Audit Records and Mappings

Target Field	Audit Vault Server Field	Map to Field Type
DB_ID	extension field	extension field
INSTANCE	extension field	extension field
PROCESS	extension field	extension field
SESSION_ID	marker field	marker field
ENTRY_ID	marker field	marker field

Example 3-1 Sample XML Audit Record

```
<?xml version="1.0" encoding="UTF-8"?>
<Audit>
    <AuditRecord>
        <Audit_type>1</Audit_type>
        <User_id>scott</User_id>
        <Os_user_id>usr1</Os_user_id>
        <Action>select</Action>
        <Status>0</Status>
        <Event_time>2010-11-11 12:23:59.166</Event_time>
        <Obj_name>emp</Obj_name>
        <Terminal>t1</Terminal>
        <Db_id>136</Db_id>
        <Session_id>170191</Session_id>
        <Entry_id>1</Entry_id>
    </AuditRecord>
    <AuditRecord>
        <Audit_type>3</Audit_type>
        <User_id>scott</User_id>
        <Os_user_id>usr1</Os_user_id>
        <Action>delete</Action>
        <Status>1</Status>
        <Event_time>2010-11-11 12:33:59.166</Event_time>
        <Obj_name>emp</Obj_name>
        <Terminal>t1</Terminal>
        <Db_id>136</Db_id>
        <Session_id>170191</Session_id>
        <Entry_id>2</Entry_id>
    </AuditRecord>
</Audit>
```

3.3.3 Creating the XML File Audit Collection Mapper File

To create an XML file collection plug-in mapper file, you must describe the collection plug-in mappings in this mapper file in accordance with Oracle Audit Vault and Database Firewall standards.

You must describe the collection plug-in mappings in this mapper file as follows:

Standards for Collection Plug-in Mappings in Mapper Files for Oracle Audit Vault and Database Firewall

- Top-Level Element

```
<AVXMLCollectorTemplate securedTargetType="XMLSOURCE"
    maxSecuredTargetVersion="11.0" version="1.0">
```

The `AVXMLCollectorTemplate` is the top level element and has these mandatory attributes: `securedTargetType`, `maxSecuredTargetVersion`, and `version`. The `minSecuredTargetVersion` attribute is optional.

The accepted format for the `minSecuredTargetVersion`, `maxSecuredTargetVersion`, and `version` attributes uses numbers, separated by dots, such as 12.2,10.3.2, 11.2.3.0.

- Header Information

```
<HeaderInfo>
  <StartTag>Audit</StartTag>
</HeaderInfo>
```

`HeaderInfo` is mandatory. It contains one child element, `StartTag`, which names the top-level element of the audit record file.

- Record Information

```
<RecordInfo>
  <StartTag>AuditRecord</StartTag>
</RecordInfo>
```

`RecordInfo` provides the starting element of audit records in XML audit files. `RecordInfo` is mandatory.

`StartTag` is the starting element of each audit record in XML audit files.

- Field Mapping Information

```
<FieldMappingInfo>
```

`FieldMappingInfo` provides mapping information from target fields to various Audit Vault fields, contained in these child elements, `CoreFields`, `LargeFields`, `ExtensionField`, and `MarkerField`.

Field mappings include `<Map>` elements, which contain `<Name>` elements that hold target field names, and `<MapTo>` elements that hold Audit Value field names that targets are mapped to.

There should be no many-to-one mappings from target fields to Audit Vault Server fields. For example, the following is invalid:

```
<!-- Invalid code
<Map>
    <Name>USER_ID</Name>
```

```
    <MapTo>UserName</MapTo>
</Map>
<Map>
    <Name>OS_USER_ID</Name>
    <MapTo>UserName</MapTo>
</Map>  -->
```

– Core Fields

```
<CoreFields>
```

`CoreFields` provides mapping from target fields to core fields of Audit Vault Server. Target fields specified in core field mappings must be of SQL data type, either a string or a data type that can convert to string.

The following elements contain core fields.

```
<Map>
 <Name>EVENT_TIME</Name>
  <MapTo>EventTimeUTC</MapTo>
  <TimestampPattern>yyyy-MM-dd HH:mm:ss.SSS</TimestampPattern>
</Map>
```

`EventTimeUTC` provides event time mapping information. The value in `TimestampPattern` specifies the timestamp format for event time. `EventTimeUTC` and `TimestampPattern` are mandatory.

When specifying the `TimestampPattern`, use the supported patterns and characters of the Java `SimpleDateFormat` class, *NOT* Oracle Database specific patterns.

For multibyte characters such as Chinese, specific words such as Month should be added into the pattern as characters in `SimpleDateFormat`. The AM and PM indicators are obtained based on locale, but should be explicitly mentioned in the TimestampPattern that you provide in the mapper file.

```
<Map>
  <Name>USER_ID</Name>
  <MapTo>UserName</MapTo>
</Map>
```

`UserName` represents the user who performed the action. If the mapping is not provided, Audit Data Collection still starts successfully, but every audit record is treated as invalid.

```
<Map>
  <Name>OS_USER_ID</Name>
  <MapTo>OSUserName</MapTo>
</Map>
 <Map>
  <Name>ACTION</Name>
```

```
  <MapTo>CommandClass</MapTo>
</Map>
```

`CommandClass` represents the action of the event. If the mapping is not provided, Audit Data Collection still starts successfully, but all audit records are treated as invalid.

```
        <Transformation>
          <ValueTransformation from="1" to="CREATE"/>
          <ValueTransformation from="2" to="INSERT"/>
          <ValueTransformation from="3" to="SELECT"/>
          <ValueTransformation from="4" to="CREATE"/>
          <ValueTransformation from="15" to="READ"/>
          <ValueTransformation from="30" to="LOGON"/>
          <ValueTransformation from="34" to="LOGOFF"/>
          <ValueTransformation from="35" to="ACQUIRE"/>
        </Transformation>
```

`CommandClass` contains a `Transformation` field with `ValueTransformation` values, from targets to the Audit Vault Server `CommandClass` field. These transformations are mandatory.

The *to* attributes are values for the `CommandClass` field. If you can meaningfully map an event to one of these values, Oracle recommends that you do so. If this is not possible, use a value that appropriately reflects the action that generated the audit event.

```
<Map>
  <Name>OBJ_NAME</Name>
  <MapTo>TargetObject</MapTo>
</Map>
<Map>
  <Name>USER_HOST</Name>
  <MapTo>ClientHostName</MapTo>
</Map>

<Map>
   <Name>TERMINAL</Name>
   <MapTo>TerminalName</MapTo>
</Map>
<Map>
  <Name>OBJ_CREATOR</Name>
  <MapTo>TargetOwner</MapTo>
</Map>
<Map>
  <Name>STATUS</Name>
  <MapTo>EventStatus</MapTo>
        <Transformation>
          <ValueTransformation from="0" to="FAILURE"/>
          <ValueTransformation from="1" to="SUCCESS"/>
          <ValueTransformation from="2" to="UNKNOWN"/>
```

```
        </Transformation>
</Map>
```

`EventStatus` contains a `Transformation` field with `ValueTransformation` values, from targets to Audit Vault `EventStatus` fields. These transformations are mandatory.

```
</CoreFields>
```

- Large Fields Information

```
<LargeFields>
  <Map>
    <Name>SQL_TEXT</Name>
    <MapTo>CommandText</MapTo>
  </Map>
<Map>
   <Name>COMMAND_PARAMETER</Name>
   <MapTo>CommandParam</MapTo>
  </Map>
</LargeFields>
```

`LargeFields` are target fields mapped to large fields in the Audit Vault Server. The specified target fields must be of SQL data type `CLOB` or `String`, or be convertible to `String`.

- Extension Fields

```
<ExtensionField>
   <Name>DB_ID</Name>
   <Name>INSTANCE</Name>
   <Name>PROCESS</Name>
</ExtensionField>
```

`ExtensionField`s are target field names that must be stored as a name-value pair in the `Extension` field in Audit Vault Server. Target fields specified must be of SQL data type `CLOB` or `String`, or be convertible to `String`.

- Marker Fields

```
<MarkerField>
  <Name>SESSION_ID</Name>
  <Name>ENTRY_ID</Name>
</MarkerField>
```

`MarkerField` contains a list of target fields that uniquely identify each audit record. The target fields specified must be of SQL data type `CLOB` or `String`, or be convertible to `String`. `MarkerField` is mandatory.

See Also:

- XML File Collection Plug-in Examples
- Core Fields
- Actions and Target Types
- Large Fields
- Extension Field
- Marker Field

3.3.4 XML Transformation for Non-Standard Audit Records

If you have audit records in a non-standard audit data format, you can apply XML transformation using XSL on the XML audit records.

To apply XML transformation on the audit records, you provide an XSL file that can transform the audit data from its original format to the format currently specified for the XML file collection plug-ins. Doing this means that you can enhance file collection plug-ins to support a variety of XML audit data formats.

Related Topics

- Example Audit Trail for an XML File Collection Plug-in
  This example audit trail for an xml file collection plug-in shows the details of an XML file collection plug-in.

3.3.4.1 Additional Requirement for XML Transformation Using XSL

To transform non-standard audit records into the current format, your transformer must follow Oracle Audit Vault and Database Firewall standards.

The transformer must write to audit files in an incremental order. That is, the transformer must write to one audit file until its maximum size is reached, and then move over to another file. Therefore, only one file can be active at a time. If the transformer finds more than one incomplete XML audit file, then the XML file collection plug-in stops.

3.3.4.2 Changes Required to Transform Non-Standard Audit Records

To transform non-standard audit records with Oracle Audit Vault and Database Firewall, you must complete this procedure.

You must perform these steps:

1. Add a section such as this example to the mapper file after `<RecordInfo>`, specifying the name of XSL file that you want to be used for transformation, and the `SourceFileStartTag` for the file to be transformed.

```
<XslTransformation>
    <XslFile>test_template.xsl</XslFile>
    <SourceFileStartTag>AUDIT</SourceFileStartTag>
</XslTransformation>
```

2. Provide the XSL file and place it in the `templates` folder of the plugin directory.
3. You can also make calls to Java functions from within the XSL file. To do this, place the `jar` file created in the `jars` folder of the plugin directory.

Related Topics

- Creating the XML File Audit Collection Mapper File
  To create an XML file collection plug-in mapper file, you must describe the collection plug-in mappings in this mapper file in accordance with Oracle Audit Vault and Database Firewall standards.
- Sample Non-Standard XML Audit Data Record
  See how to transform an XML data record to the proper XML format required for an XML file collection plug-in.
- Creating an XSL File for Transformation
  To create an XSL transformation file that defines transformation rules you must create a version that can transform the source audit records that your system creates, and place it in the `templates` folder of the plugin.

3.3.4.3 Sample Non-Standard XML Audit Data Record

See how to transform an XML data record to the proper XML format required for an XML file collection plug-in.

As you review this example, note that your source system can produce audit records with a different appearance.

Example 3-2 Audit.xml: Sample XML Audit Record

```
<?xml version="1.0" encoding="UTF-8"?>
<AUDIT>

   <AUDIT_RECORD TIMESTAMP="2013-06-07T08:27:53" NAME="Audit"
   SERVER_ID="0" VERSION="1" STARTUP_OPTIONS="C:/Program Files/MySQL/
MySQL
   Server 5.6/bin\mysqld --defaults-file=C:\ProgramData\MySQL\MySQL
Server
   5.6\my.ini" OS_VERSION="x86_64-Win64" MYSQL_VERSION=
   "5.6.11-enterprise-commercial-advanced"/>

   <AUDIT_RECORD TIMESTAMP="2013-06-07T08:30:46" NAME="Connect"
CONNECTION_ID="1"
   STATUS="0" USER="root" PRIV_USER="root" OS_LOGIN="" PROXY_USER=""
   HOST="localhost" IP="127.0.0.1" DB=""/>

   <AUDIT_RECORD TIMESTAMP="2013-06-07T08:31:21" NAME="Query"
CONNECTION_ID="1"
   STATUS="0" SQLTEXT="CREATE USER 'admin'@'localhost' IDENTIFIED BY
   'welcome_1'"/>

</AUDIT>
```

3.3.4.4 Creating an XSL File for Transformation

To create an XSL transformation file that defines transformation rules you must create a version that can transform the source audit records that your system creates, and place it in the `templates` folder of the plugin.

The `Audit.xml` transformed audit record file does not appear in your folder. It is just an example showing the result of transforming the `Audit.xml` file into the required XML format, using the XSL transformation file in the `test_template.xsl` example.

Example 3-3 test_template.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/
Transform">
  <xsl:output indent="yes" />
     <xsl:template match="/">
        <ROOT_DEST>
          <xsl:for-each select="AUDIT/AUDIT_RECORD">
            <Record_Dest>
                <USER><xsl:value-of select="@USER"/></USER>
                <PRIV_USER><xsl:value-of select="@PRIV_USER"/></PRIV_USER>
                <OS_LOGIN><xsl:value-of select="@OS_LOGIN"/></OS_LOGIN>
                <PROXY_USER><xsl:value-of select="@PROXY_USER"/></PROXY_USER>
                <HOST><xsl:value-of select="@HOST"/></HOST>
                <IP><xsl:value-of select="@IP"/></IP>
                <DB><xsl:value-of select="@DB"/></DB>
                <SQLTEXT><xsl:value-of select="@SQLTEXT"/></SQLTEXT>
                <CONNECTION_ID><xsl:value-of select=
                      "@CONNECTION_ID"/></CONNECTION_ID>
                <STATUS><xsl:value-of select="@STATUS"/></STATUS>
                <TIMESTAMP><xsl:value-of select="@TIMESTAMP"/></TIMESTAMP>
                <NAME><xsl:value-of select="@NAME"/></NAME>
                <SERVER_ID><xsl:value-of select="@SERVER_ID"/></SERVER_ID>
                <VERSION><xsl:value-of select="@VERSION" /></VERSION>
   <STARTUP_OPTIONS><xsl:value-of select="@STARTUP_OPTIONS"/> </
STARTUP_OPTIONS>
                <OS_VERSION><xsl:value-of select="@OS_VERSION"/></OS_VERSION>
                <MYSQL_VERSION><xsl:value-of select="@MYSQL_VERSION"/>
                    </MYSQL_VERSION>
            </Record_Dest>
          </xsl:for-each>
         </ROOT_DEST>
        </xsl:template>
  </xsl:stylesheet>
```

Example 3-4 Transformed Audit Record file

```
<ROOT_DEST>
     <Record_Dest>
         <USER></USER>
         <PRIV_USER></PRIV_USER>
```

```
            <OS_LOGIN></OS_LOGIN>
            <PROXY_USER></PROXY_USER>
            <HOST></HOST>
            <IP></IP>
            <DB></DB>
            <SQLTEXT></SQLTEXT>
            <CONNECTION_ID></CONNECTION_ID>
            <STATUS></STATUS>
            <TIMESTAMP>2013-06-07T08:27:53</TIMESTAMP>
            <NAME>Audit</NAME>
            <SERVER_ID>0</SERVER_ID>
            <VERSION>1</VERSION>
            <STARTUP_OPTIONS>C:/Program Files/MySQL/MySQL Server 5.6/
bin\mysqld
                --defaults-file=C:\ProgramData\MySQL\MySQL Server
                5.6\my.ini</STARTUP_OPTIONS>
            <OS_VERSION>x86_64-Win64</OS_VERSION>
            <MYSQL_VERSION>5.6.11-enterprise-commercial-advanced</
MYSQL_VERSION>
        </Record_Dest>
        <Record_Dest>
            <USER>root</USER>
            <PRIV_USER>root</PRIV_USER>
            <OS_LOGIN></OS_LOGIN>
            <PROXY_USER></PROXY_USER>
            <HOST>localhost</HOST>
            <IP>127.0.0.1</IP>
            <DB></DB>
            <SQLTEXT></SQLTEXT>
            <CONNECTION_ID>1</CONNECTION_ID>
            <STATUS>0</STATUS>
            <TIMESTAMP>2013-06-07T08:30:46</TIMESTAMP>
            <NAME>Connect</NAME>
            <SERVER_ID></SERVER_ID>
            <VERSION></VERSION>
            <STARTUP_OPTIONS></STARTUP_OPTIONS>
            <OS_VERSION></OS_VERSION>
            <MYSQL_VERSION></MYSQL_VERSION>
        </Record_Dest>
        <Record_Dest>
            <USER></USER>
            <PRIV_USER></PRIV_USER>
            <OS_LOGIN></OS_LOGIN>
            <PROXY_USER></PROXY_USER>
            <HOST></HOST>
            <IP></IP>
            <DB></DB>
            <SQLTEXT>CREATE USER 'admin'@'localhost' IDENTIFIED BY
                    'welcome_1'</SQLTEXT>
            <CONNECTION_ID>1</CONNECTION_ID>
            <STATUS>0</STATUS>
            <TIMESTAMP>2013-06-07T08:31:21</TIMESTAMP>
            <NAME>Query</NAME>
            <SERVER_ID></SERVER_ID>
            <VERSION></VERSION>
```

```
            <STARTUP_OPTIONS></STARTUP_OPTIONS>
            <OS_VERSION></OS_VERSION>
            <MYSQL_VERSION></MYSQL_VERSION>
        </Record_Dest>
</ROOT_DEST>
```

3.4 JSON File Collection Plug-ins

Learn how to use Oracle AVDF JSON file collection plug-ins to collect audit data from a JSON file type of trail.

JSON file collection plug-ins support collection of audit data from an JSON file type of trail. All these JSON audit files must be present in single directory. You can specify details of the JSON audit data in the mapper file.

Related Topics

- Schema For JSON File Collection Plug-in Mapper File
  See how to set up a schema for a JSON file collection plug-in mapper file for Oracle Audit Vault and Database Firewall.

3.4.1 Requirements for JSON File Collection Plug-ins

To use JSON collection plug-ins for reading audit trails from JSON files, your data must meet Oracle Audit Vault and Database Firewall requirements.

You can use collection plug-ins for reading audit trails from JSON audit record files if the JSON files meet the requirements for collection.

JSON File Audit Record File Requirements for Oracle Audit Vault and Database Firewall

- The audit trail must be stored in one or more JSON files in a single directory path.
- The user must have read permission on the directory containing the JSON audit files.
- JSON files in this directory must be valid, well-formed JSON documents, within the constraints of the JSON specification.
- The file and record start elements must be as specified in the mapper file.
- The JSON audit file must have a field whose `JSONPath` can be mapped to the `CommandClass` core field. If a record has its `CommandClass` field as null, then the record is treated as invalid.
- In the JSON file, each audit record must have a timestamp as one of its element values.

  The value of the timestamp element must be monotonically increasing, that is, the value of the field increases with every new audit record inserted into the trail. The timestamp value should be strictly `Not Null`. Timestamp format must be according to `SimpleDateFormat` Java class.

  This field must mapped to the `EventTimeUTC` core field in the mapper file. If mapping for event time is not specified in the mapper file, then the collection plug-in shuts down. If the field value for the event time in audit records is found null, then the collection plug-in takes the time of the record last sent from the same JSON audit file.
- The audit trail must contain a single element value or group of element values in the audit record that uniquely identify each audit record in JSON Audit files.

- If an audit data target produces audit files with multiple JSON formats, then the user must provide a separate mapper file for each audit file format having a different start element.
- JSON files in this directory should be of the same locale and encoding as the agent, as described in the examples below:
  - Valid: The user has an agent in a Chinese locale (`env`). JSON files are also generated in a Chinese locale with same encoding (for example, `ZHS16GBK`). This setup is valid.
  - Invalid: The user has an agent in a German locale (env). JSON files are generated/moved from some other computer, which are Chinese encoded. The collectors fail to start because of an encoding mismatch, as well as a locale mismatch, in this case. This setup is invalid.

3.4.2 Example Audit Trail for a JSON File Collection Plug-in

This example audit trail for a JSON file collection plug-in shows the details of a JSON file collection plug-in.

This example file is used in other locations to demonstrate the creation and structure of a sample mapper file for the creation and structure of a sample mapper file for a JSON file collection plug-in, in the Oracle Audit Vault and Database Firewall documentation.

The following table lists the audit record structure and mappings to Oracle Audit Vault Server fields for the hypothetical target type, `JSONSOURCE`, which generates and stores audit data in JSON audit files.

Table 3-3 Audit Data Fields in JSON Audit Records and Mappings

Target Field	Audit Vault Server Field	Map to Field Type
USER_ID	UserName	core field
OS_USER_ID	OSUserName	core field
ACTION	CommandClass	core field
STATUS	EventStatus	core field
EVENT_TIME	EventTimeUTC	core field
OBJ_NAME	TargetObject	core field
OBJ_CREATOR	TargetOwner	core field
USER_HOST	ClientHostName	core field
SQL_TEXT	CommandText	core field
SQL_BIND	CommandParam	core field
TERMINAL	TerminalName	extension field
DB_ID	extension field	extension field
INSTANCE	extension field	extension field
PROCESS	extension field	extension field
SESSION_ID	marker field	marker field
ENTRY_ID	marker field	marker field

Example 3-5 Sample JSON Audit Record

```
{
   "ITEMS":[
      {
         "SESSION_ID":123,
         "AUDIT_TYPE":1,
         "USER_ID":"scott",
         "OS_USER_ID":"usr1",
         "ACTION":"select",
         "STATUS":0,
         "EVENT_TIME":"2020-11-28 12:23:59.166",
         "OBJ_NAME":"emp",
         "OBJ_CREATOR":"scott",
         "TERMINAL":"t1",
         "DB_ID":136,
         "ENTRY_ID":1
      },
      {
         "SESSION_ID":123,
         "AUDIT_TYPE":1,
         "USER_ID":"scott",
         "OS_USER_ID":"usr1",
         "ACTION":"delete",
         "STATUS":0,
         "EVENT_TIME":"2020-11-28 12:24:22.177",
         "OBJ_NAME":"emp",
         "OBJ_CREATOR":"scott",
         "TERMINAL":"t1",
         "DB_ID":136,
         "ENTRY_ID":2
      }
   ]
}
```

3.4.3 Creating the JSON File Audit Collection Mapper File

To create a JSON file collection plug-in mapper file, you must describe the collection plug-in mappings in this mapper file in accordance with Oracle Audit Vault and Database Firewall standards.

You must describe the collection plug-in mappings in this mapper file as follows:

Standards for Collection Plug-in Mappings in Mapper Files for Oracle Audit Vault and Database Firewall

- Top-Level Element

```
<AVJSONCollectorTemplate securedTargetType="JSONSOURCE"
      maxSecuredTargetVersion="11.0"
      version="1.0">
```

The `AVJSONCollectorTemplate` is the top level element and has these mandatory attributes: `securedTargetType`, `maxSecuredTargetVersion`, and `version`. The `minSecuredTargetVersion` attribute is optional.

The accepted format for the `minSecuredTargetVersion`, `maxSecuredTargetVersion`, and `version` attributes uses numbers, separated by dots, such as 12.2,10.3.2, 11.2.3.0.

- Header Information

```
<HeaderInfo>
  <StartTag>ITEMS</StartTag>
</HeaderInfo>
```

`HeaderInfo` is mandatory. It contains one child element, `StartTag`, which names the top-level element of the audit record file.

- Record Information

```
<RecordInfo>
  <StartTag>SESSION_ID</StartTag>
</RecordInfo>
```

`RecordInfo` provides the starting element of audit records in JSON audit files. `RecordInfo` is mandatory.

`StartTag` is the starting element of each audit record in JSON audit files. If the JSON file has one fully formed JSON record per line, then the `HeaderInfo` and `RecordInfo` also have the same start tag, which is the first element of the JSON record.

- Field Mapping Information

```
<FieldMappingInfo>
```

`FieldMappingInfo` provides mapping information from target fields to various Audit Vault fields, contained in these child elements, `CoreFields`, `LargeFields`, `ExtensionField`, and `MarkerField`.

Field mappings include `<Map>` elements, which contain `<Name>` elements that hold target field names, and `<MapTo>` elements that hold Audit Value field names that targets are mapped to.

There should be no many-to-one mappings from target fields to Audit Vault Server fields. For example, the following is invalid:

```
<!-- Invalid code
<Map>
    <Name>$.USER_ID</Name>
    <MapTo>UserName</MapTo>
</Map>
<Map>
```

```
    <Name>$.OS_USER_ID</Name>
    <MapTo>UserName</MapTo>
</Map>  -->
```

– Core Fields

```
<CoreFields>
```

`CoreFields` provides mapping from target fields to core fields of Audit Vault Server. Target fields specified in core field mappings must be of SQL data type, either a string or a data type that can convert to string.

The following elements contain core fields.

```
<Map>
 <Name>$.EVENT_TIME</Name>
  <MapTo>EventTimeUTC</MapTo>
  <TimestampPattern>yyyy-MM-dd HH:mm:ss.SSS</TimestampPattern>
</Map>
```

`EventTimeUTC` provides event time mapping information. The value in `TimestampPattern` specifies the timestamp format for event time. `EventTimeUTC` and `TimestampPattern` are mandatory.

When specifying the `TimestampPattern`, use the supported patterns and characters of the Java `SimpleDateFormat` class, *NOT* Oracle Database specific patterns.

For multibyte characters such as Chinese, specific words such as Month should be added into the pattern as characters in `SimpleDateFormat`. The AM and PM indicators are obtained based on locale, but should be explicitly mentioned in the TimestampPattern that you provide in the mapper file.

```
<Map>
  <Name>$.USER_ID</Name>
  <MapTo>UserName</MapTo>
</Map>
```

`UserName` represents the user who performed the action. If the mapping is not provided, Audit Data Collection still starts successfully, but every audit record is treated as invalid.

```
<Map>
  <Name>$.OS_USER_ID</Name>
  <MapTo>OSUserName</MapTo>
</Map>
 <Map>
  <Name>$.ACTION</Name>
  <MapTo>CommandClass</MapTo>
</Map>
```

`CommandClass` represents the action of the event. If the mapping is not provided, Audit Data Collection still starts successfully, but all audit records are treated as invalid.

```
        <Transformation>
          <ValueTransformation from="1" to="CREATE"/>
          <ValueTransformation from="2" to="INSERT"/>
          <ValueTransformation from="3" to="SELECT"/>
          <ValueTransformation from="4" to="CREATE"/>
          <ValueTransformation from="15" to="READ"/>
          <ValueTransformation from="30" to="LOGON"/>
          <ValueTransformation from="34" to="LOGOFF"/>
          <ValueTransformation from="35" to="ACQUIRE"/>
        </Transformation>
```

`CommandClass` contains a `Transformation` field with `ValueTransformation` values, from targets to the Audit Vault Server `CommandClass` field. These transformations are mandatory.

The *to* attributes are values for the `CommandClass` field. If you can meaningfully map an event to one of these values, Oracle recommends that you do so. If this is not possible, use a value that appropriately reflects the action that generated the audit event.

```
<Map>
  <Name>$.OBJ_NAME</Name>
  <MapTo>TargetObject</MapTo>
</Map>
<Map>
  <Name>$.USER_HOST</Name>
  <MapTo>ClientHostName</MapTo>
</Map>

<Map>
   <Name>$.TERMINAL</Name>
   <MapTo>TerminalName</MapTo>
</Map>
<Map>
  <Name>$.OBJ_CREATOR</Name>
  <MapTo>TargetOwner</MapTo>
</Map>
<Map>
  <Name>$.STATUS</Name>
  <MapTo>EventStatus</MapTo>
        <Transformation>
          <ValueTransformation from="0" to="FAILURE"/>
          <ValueTransformation from="1" to="SUCCESS"/>
          <ValueTransformation from="2" to="UNKNOWN"/>
        </Transformation>
</Map>
```

`EventStatus` contains a `Transformation` field with `ValueTransformation` values, from targets to Audit Vault `EventStatus` fields. These transformations are mandatory.

```
</CoreFields>
```

- Large Fields Information

```
<LargeFields>
  <Map>
    <Name>$.SQL_TEXT</Name>
    <MapTo>CommandText</MapTo>
  </Map>
<Map>
   <Name>$.COMMAND_PARAMETER</Name>
   <MapTo>CommandParam</MapTo>
  </Map>
</LargeFields>
```

`LargeFields` are target fields mapped to large fields in the Audit Vault Server. The specified target fields must be of SQL data type `CLOB` or `String`, or be convertible to `String`.

- Extension Fields

```
<ExtensionField>
   <Name>$.DB_ID</Name>
   <Name>$.INSTANCE</Name>
   <Name>$.PROCESS</Name>
</ExtensionField>
```

`ExtensionField`s are target field names that must be stored as a name-value pair in the `Extension` field in Audit Vault Server. Target fields specified must be of SQL data type `CLOB` or `String`, or be convertible to `String`.

- Marker Fields

```
<MarkerField>
  <Name>$.SESSION_ID</Name>
  <Name>$.ENTRY_ID</Name>
</MarkerField>
```

`MarkerField` contains a list of target fields that uniquely identify each audit record. The target fields specified must be of SQL data type `CLOB` or `String`, or be convertible to `String`. `MarkerField` is mandatory.

See Also:

- JSON File Collection Plug-in Example
- Core Fields
- Actions and Target Types
- Large Fields
- Extension Field
- Marker Field

3.5 CSV File Collection Plug-ins

Learn how to use Oracle AVDF CSV file collection plug-ins to collect audit data from a CSV file type of trail.

CSV file collection plug-ins support collection of audit data from an CSV file type of trail. All these CSV audit files must be present in single directory. You can specify details of the CSV audit data in the mapper file.

3.5.1 Requirements for CSV File Collection Plug-ins

To use CSV collection plug-ins for reading audit trails from CSV files, your data must meet Oracle Audit Vault and Database Firewall requirements.

You can use collection plug-ins for reading audit trails from CSV audit record files if the CSV files meet the requirements for collection.

CSV File Audit Record File Requirements for Oracle Audit Vault and Database Firewall

- The audit trail must be stored in one or more CSV files in a single directory path.
- The user must have read permission on the directory containing the CSV audit files.
- CSV files in this directory must be valid, well formed CSV documents, with `COMMA` as the field delimiter.
- If a record has its `CommandClass` field as null, then the record is treated as invalid.
- In the CSV file, each audit record must have a timestamp as one of its element values.

  The value of the timestamp element must be monotonically increasing, that is, the value of the field increases with every new audit record inserted into the trail. The timestamp value should be strictly `Not Null`. Timestamp format must be according to `SimpleDateFormat` Java class.

  This field must mapped to the `EventTimeUTC` core field in the mapper file. If mapping for event time is not specified in the mapper file, then the collection plug-in shuts down.
- CSV files in this directory should be of the same locale and encoding as the agent, as described in the examples below:

- Valid: The user has an agent in a Chinese locale (`env`). CSV files are also generated in a Chinese locale with same encoding (for example, `ZHS16GBK`). This setup is valid.
- Invalid: The user has an agent in a German locale (env). CSV files are generated/ moved from some other computer, which are Chinese encoded. The collectors fail to start because of an encoding mismatch, as well as a locale mismatch, in this case. This setup is invalid.

3.5.2 Example Audit Trail for a CSV File Collection Plug-in

This example audit trail for a CSV file collection plug-in shows the details of a CSV file collection plug-in.

This example file is used in other locations to demonstrate the creation and structure of a sample mapper file for the creation and structure of a sample mapper file for a CSV file collection plug-in, in the Oracle Audit Vault and Database Firewall documentation.

The following table lists the audit record structure and mappings to Oracle Audit Vault Server fields for the hypothetical target type, `CSVSOURCE`, which generates and stores audit data in CSV audit files.

Table 3-4 Audit Data Fields in CSV Audit Records and Mappings

Target Field	Audit Vault Server Field	Map to Field Type
EVENT_NAME	CommandClass	core field
EVENT_TIME	EventTimeUTC	core field
CLIENT_IP	ClientIP	core field
USER_ID	UserName	core field
TARGET_OBJECT	TargetObject	core field
EVENT_STATUS	EventStatus	core field
SESSION_ID	marker field	marker field
ENTRY_ID	marker field	marker field
COMMAND_TEXT	CommandText	large field
COMMAND_PARAM	CommandParam	large field
SESSION_ID	extension field	extension field
ENTRY_ID	extension field	extension field

Example 3-6 Sample CSV Audit Record

```
5678,createUser,2020-10-01T16:11:23.661+0530,127.0.0.1,1234,admin,user1,0,0,n
ot applicable,1234,"insert into foo.bar","foobar",111
5679,dropUser,2020-10-02T16:11:23.661+0530,127.0.0.1,1234,admin,user2,0,0,not
 applicable,1234,"delete from foo.bar","foobar",222
5680,createCollection,2020-10-03T16:11:23.661+0530,127.0.0.1,1234,admin,colle
ction1,100,18,authentication failed,1234,"insert into foo.bar","foobar",333
5681,dropCollection,2020-10-04T16:11:23.661+0530,127.0.0.1,1234,admin,collect
```

```
ion2,200,13,not authorized to perform operation,1234,"delete from
foo.bar","foobar",444


Below is the index corresponding to each field:
EVENT_ID field has index 0
EVENT_NAME field has index 1
EVENT_TIME field has index 2
CLIENT_IP field has index 3
CLIENT_PORT field has index 4
USER_ID field has index 5
TARGET_OBJECT field has index 6
EVENT_STATUS field has index 7
ERROR_ID field has index 8
ERROR_MESSAGE field index 9
SESSION_ID field has index 10
COMMAND_TEXT field has index 11
COMMAND_PARAM field has index 12
ENTRY_ID field has index 13
```

3.5.3 Creating the CSV File Audit Collection Mapper File

To create a CSV file collection plug-in mapper file, you must describe the collection plug-in mappings in this mapper file in accordance with Oracle Audit Vault and Database Firewall standards.

You must describe the collection plug-in mappings in this mapper file as follows:

Standards for Collection Plug-in Mappings in Mapper Files for Oracle Audit Vault and Database Firewall

- Top-Level Element

```
<AVCSVCollectorTemplate securedTargetType="CSVSOURCE"
      maxSecuredTargetVersion="11.0"
      version="1.0">
```

The `AVCSVCollectorTemplate` is the top level element and has these mandatory attributes: `securedTargetType`, `maxSecuredTargetVersion`, and `version`. The `minSecuredTargetVersion` attribute is optional.

The accepted format for the `minSecuredTargetVersion`, `maxSecuredTargetVersion`, and `version` attributes uses numbers, separated by dots, such as 12.2,10.3.2, 11.2.3.0.

- Header Information

```
<HeaderInfo>
  <StartTag>CSV</StartTag>
</HeaderInfo>
```

`HeaderInfo` is mandatory. The `StartTag` must be set to CSV.

- Record Information

```
<RecordInfo>
  <StartTag>CSV</StartTag>
</RecordInfo>
```

`RecordInfo` is mandatory. `StartTag` must be set to CSV.

- Field Mapping Information

```
<FieldMappingInfo>
```

`FieldMappingInfo` provides mapping information from target fields to various Audit Vault fields, contained in these child elements, `CoreFields`, `LargeFields`, `ExtensionField`, and `MarkerField`.

Field mappings include `<Map>` elements, which contain `<Name>` elements that hold target field names, and `<MapTo>` elements that hold Audit Value field names that targets are mapped to.

In CSV Plugin Mapper file, the `Name` element must contain the index of the field in the CSV file. In our sample, below are the index corresponding to each field:

```
EVENT_ID field has index 0
EVENT_NAME field has index 1
EVENT_TIME field has index 2
CLIENT_IP field has index 3
CLIENT_PORT field has index 4
USER_ID field has index 5
TARGET_OBJECT field has index 6
EVENT_STATUS field has index 7
ERROR_ID field has index 8
ERROR_MESSAGE field index 9
SESSION_ID field has index 10
COMMAND_TEXT field has index 11
COMMAND_PARAM field has index 12
ENTRY_ID field has index 13
```

There should be no many-to-one mappings from target fields to Audit Vault Server fields. For example, the following is invalid:

```
<!-- Invalid code
<Map>
    <Name>0</Name>
    <MapTo>UserName</MapTo>
</Map>
<Map>
    <Name>1</Name>
    <MapTo>UserName</MapTo>
```

```
</Map> -->
```

- Core Fields

  ```
  <CoreFields>
  ```

  `CoreFields` provides mapping from target fields to core fields of Audit Vault Server. Target fields specified in core field mappings must either be a string or a data type that can be converted to string.

  The following elements contain core fields.

  ```
  <Map>
              <Name>2</Name>
              <MapTo>EventTimeUTC</MapTo>
              <TimestampPattern>yyyy-MM-dd'T'HH:mm:ss.SSSZ</
  TimestampPattern>
           </Map>
  ```

  `EventTimeUTC` provides event time mapping information. The value in `TimestampPattern` specifies the timestamp format for event time. `EventTimeUTC` and `TimestampPattern` are mandatory.

  When specifying the `TimestampPattern`, use the supported patterns and characters of the Java `SimpleDateFormat` class, *NOT* Oracle Database specific patterns.

  For multibyte characters such as Chinese, specific words such as Month should be added into the pattern as characters in `SimpleDateFormat`. The AM and PM indicators are obtained based on locale, but should be explicitly mentioned in the TimestampPattern that you provide in the mapper file.

  ```
           <Map>
              <Name>5</Name>
              <MapTo>UserName</MapTo>
           </Map>
  ```

  `UserName` represents the user who performed the action. If the mapping is not provided, Audit Data Collection still starts successfully, but every audit record is treated as invalid.

  ```
   <Map>
    <Name>1</Name>
    <MapTo>CommandClass</MapTo>
  </Map>
  ```

`CommandClass` represents the action of the event. If the mapping is not provided, Audit Data Collection still starts successfully, but all audit records are treated as invalid.

```
            <Transformation>
               <ValueTransformation from="createUser" to="CREATE" />
               <ValueTransformation from="createCollection"
to="CREATE" />
               <ValueTransformation from="authenticate"
to="AUTHENTICATE" />
               <ValueTransformation from="dropCollection" to="DROP" />
               <ValueTransformation from="dropUser" to="DROP" />
            </Transformation>
```

`CommandClass` contains a `Transformation` field with `ValueTransformation` values, from targets to the Audit Vault Server `CommandClass` field. These transformations are mandatory.

The *to* attributes are values for the `CommandClass` field. If you can meaningfully map an event to one of these values, Oracle recommends that you do so. If this is not possible, use a value that appropriately reflects the action that generated the audit event.

```
         <Map>
            <Name>1</Name>
            <MapTo>TargetObject</MapTo>
            <Transformation>
               <FieldTransformation from="createUser" to="6" />
               <FieldTransformation from="createCollection" to="6" />
               <FieldTransformation from="authenticate" to="6" />
               <FieldTransformation from="dropCollection" to="6" />
               <FieldTransformation from="dropUser" to="6" />
            </Transformation>
         </Map>
             <Map>
            <Name>1</Name>
            <MapTo>TargetType</MapTo>
            <Transformation>
               <ValueTransformation from="createUser" to="USER" />
               <ValueTransformation from="createCollection"
to="COLLECTION" />
               <ValueTransformation from="authenticate" to="USER" />
               <ValueTransformation from="dropCollection"
to="COLLECTION" />
               <ValueTransformation from="dropUser" to="USER" />
            </Transformation>
         </Map>
         <Map>
            <Name>3</Name>
            <MapTo>ClientIP</MapTo>
         </Map>
```

```
            <Map>
               <Name>7</Name>
               <MapTo>EventStatus</MapTo>
               <!-- Specifying value transformation for Status
source field value.
               Mandatory: EventStatus value transformation.
               There are three possible values for EventStatus:
               SUCCESS, FAILURE, UNKNOWN -->
               <Transformation>
                  <ValueTransformation from="0" to="FAILURE" />
                  <ValueTransformation from="100" to="SUCCESS" />
                  <ValueTransformation from="200" to="UNKNOWN" />
               </Transformation>
            </Map>
```

`EventStatus` contains a `Transformation` field with `ValueTransformation` values, from targets to Audit Vault `EventStatus` fields. These transformations are mandatory.

– Large Fields Information

```
<LargeFields>
   <Map>
      <Name>11</Name>
      <MapTo>CommandText</MapTo>
   </Map>
   <Map>
      <Name>12</Name>
      <MapTo>CommandParam</MapTo>
   </Map>
</LargeFields>
```

`LargeFields` are target fields mapped to large fields in the Audit Vault Server. The specified target fields must be of type `String` or convertible to `String`.

– Extension Fields

```
<ExtensionField>
   <ComplexName>
      <Name>10</Name>
      <DisplayName>sessionid</DisplayName>
   </ComplexName>
   <ComplexName>
      <Name>13</Name>
      <DisplayName>entryid</DisplayName>
   </ComplexName>
</ExtensionField>
```

`ExtensionField`s are target field names that must be stored as a name-value pair in the `Extension` field in Audit Vault Server. Target fields specified must be of type `String` or convertible to `String`.

– Marker Fields

```
<MarkerField>
   <Name>10</Name>
   <Name>13</Name>
</MarkerField>
```

`MarkerField` contains a list of target fields that uniquely identify each audit record.

See Also:

- JSON File Collection Plug-in Example
- Core Fields
- Actions and Target Types
- Large Fields
- Extension Field
- Marker Field

3.6 JSON REST Collection Plug-ins

Learn how to use Oracle AVDF JSON collection plug-ins to collect audit data from a JSON type of trail.

JSON collection plug-ins support collection of audit data from an JSON type of trail. All these JSON audit files must be present in single directory. You can specify details of the JSON audit data in the mapper file.

Related Topics

- Schema For JSON REST Collection Plug-in Mapper File
  See how to set up a schema for a JSON REST collection plug-in mapper file for Oracle Audit Vault and Database Firewall.

3.6.1 Requirements for JSON REST Collection Plug-ins

To use JSON collection plug-ins for reading audit trails from JSON files, your data must meet Oracle Audit Vault and Database Firewall requirements.

You can use collection plug-ins for reading audit trails from JSON audit record files if the JSON files meet the requirements for collection.

JSON Audit Record File Requirements for Oracle Audit Vault and Database Firewall

- The audit trail must be stored in one or more JSON files in a single directory path.

- The user must have read permission on the directory containing the JSON audit files.
- JSON files in this directory must be valid, well-formed JSON documents, within the constraints of the JSON specification.
- The file and record start elements must be as specified in the mapper file.
- The JSON audit file must have a field whose `JSONPath` can be mapped to the `CommandClass` core field. If a record has its `CommandClass` field as null, then the record is treated as invalid.
- In the JSON file, each audit record must have a timestamp as one of its element values.

  The value of the timestamp element must be monotonically increasing, that is, the value of the field increases with every new audit record inserted into the trail. The timestamp value should be strictly `Not Null`. Timestamp format must be according to `SimpleDateFormat` Java class.

  This field must mapped to the `EventTimeUTC` core field in the mapper file. If mapping for event time is not specified in the mapper file, then the collection plug-in shuts down. If the field value for the event time in audit records is found null, then the collection plug-in takes the time of the record last sent from the same JSON audit file.
- The audit trail must contain a single element value or group of element values in the audit record that uniquely identify each audit record in JSON audit files.
- If an audit data target produces audit files with multiple JSON formats, then the user must provide a separate mapper file for each audit file format having a different start element.
- JSON files in this directory should be of the same locale and encoding as the agent, as described in the examples below:
  - Valid: The user has an agent in a Chinese locale (`env`). JSON REST files are also generated in a Chinese locale with same encoding (for example, `ZHS16GBK`). This setup is valid.
  - Invalid: The user has an agent in a German locale (env). JSON REST files are generated/moved from some other computer, which are Chinese encoded. The collectors fail to start because of an encoding mismatch, as well as a locale mismatch, in this case. This setup is invalid.

3.6.2 Example Audit Trail for a JSON REST Collections Plug-in

This example audit trail for a JSON collection plug-in shows the details of a JSON collection plug-in.

This example file is used in other locations to demonstrate the creation and structure of a sample mapper file for the creation and structure of a sample mapper file for a JSON collection plug-in, in the Oracle Audit Vault and Database Firewall documentation.

The following table lists the audit record structure and mappings to Oracle Audit Vault Server fields for the hypothetical target type, `JSONSOURCE`, which generates and stores audit data in JSON audit files.

Table 3-5 Audit Data Fields in JSON Audit Records and Mappings

Target Field	Audit Vault Server Field	Map to Field Type
USER_ID	UserName	core field
OS_USER_ID	OSUserName	core field
ACTION	CommandClass	core field
STATUS	EventStatus	core field
EVENT_TIME	EventTimeUTC	core field
OBJ_NAME	TargetObject	core field
OBJ_CREATOR	TargetOwner	core field
USER_HOST	ClientHostName	core field
SQL_TEXT	CommandText	core field
SQL_BIND	CommandParam	core field
TERMINAL	TerminalName	extension field
DB_ID	extension field	extension field
INSTANCE	extension field	extension field
PROCESS	extension field	extension field
SESSION_ID	marker field	marker field
ENTRY_ID	marker field	marker field

Example 3-7 Sample JSON Audit Record

```
{
   "ITEMS":[
      {
         "SESSION_ID":123,
         "AUDIT_TYPE":1,
         "USER_ID":"scott",
         "OS_USER_ID":"usr1",
         "ACTION":"select",
         "STATUS":0,
         "EVENT_TIME":"2020-11-28 12:23:59.166",
         "OBJ_NAME":"emp",
         "OBJ_CREATOR":"scott",
         "TERMINAL":"t1",
         "DB_ID":136,
         "ENTRY_ID":1
      },
      {
         "SESSION_ID":123,
         "AUDIT_TYPE":1,
         "USER_ID":"scott",
         "OS_USER_ID":"usr1",
         "ACTION":"delete",
         "STATUS":0,
         "EVENT_TIME":"2020-11-28 12:24:22.177",
```

```
            "OBJ_NAME":"emp",
            "OBJ_CREATOR":"scott",
            "TERMINAL":"t1",
            "DB_ID":136,
            "ENTRY_ID":2
         }
      ]
}
```

3.6.3 Creating the JSON REST Audit Collection Mapper File

To create a JSON file collection plug-in mapper file, you must describe the collection plug-in mappings in this mapper file in accordance with Oracle Audit Vault and Database Firewall standards.

You must describe the collection plug-in mappings in this mapper file as follows:

Standards for Collection Plug-in Mappings in Mapper Files for Oracle Audit Vault and Database Firewall

- Top-Level Element

```
<AVJSONCollectorTemplate securedTargetType="JSONSOURCE"
      maxSecuredTargetVersion="11.0"
      version="1.0">
```

The `AVJSONCollectorTemplate` is the top level element and has these mandatory attributes: `securedTargetType`, `maxSecuredTargetVersion`, and `version`. The `minSecuredTargetVersion` attribute is optional.

The accepted format for the `minSecuredTargetVersion`, `maxSecuredTargetVersion`, and `version` attributes uses numbers, separated by dots, such as 12.2,10.3.2, 11.2.3.0.

- Header Information

```
<HeaderInfo>
  <StartTag>ITEMS</StartTag>
</HeaderInfo>
```

`HeaderInfo` is mandatory. It contains one child element, `StartTag`, which names the top-level element of the audit record file.

- Record Information

```
<RecordInfo>
  <StartTag>SESSION_ID</StartTag>
</RecordInfo>
```

`RecordInfo` provides the starting element of audit records in JSON audit files. `RecordInfo` is mandatory.

`StartTag` is the starting element of each audit record in JSON audit files. If the JSON file has one fully formed JSON record per line, then the `HeaderInfo` and `RecordInfo` also have the same start tag, which is the first element of the JSON record.

- Service Details

```
<ServiceDetails>
```

Example code:

```
 <!-- Query format for providing the start time and end time query
parameters -->
      <QueryFormat>{startTime}/{endTime}</QueryFormat>
        <!-- Timestamp format for start time and end time -->
      <TimeFormat>yyyy-MM-dd hh:mm:ss.SSS</TimeFormat>
      <NextLink>
           <!-- Next link start tag -->
         <NextLinkStartTag>next</NextLinkStartTag>
             <!-- Next link pattern -->
         <NextLinkPattern>$.next.$ref</NextLinkPattern>
      </NextLink>
        <!-- Authentication mechanism for REST Service -->
      <RESTAuthentication>
           <!-- Username and password based Basic Authentication -->
         <BasicAuth/>
      </RESTAuthentication>
   </ServiceDetails>
```

Here is the explanation of the fields. All the fields are mandatory.

`Service Details` provides information about REST service corresponding to the audit trail.

`Query Format` describes the format of the REST query for providing the start time and end time query parameters.

`Time Format` describes timestamp format for start time and end time.

`Next Link Start Tag` provides the next link start tag of the REST URL.

`Next Link Pattern` provides JSON path expression of the next link of the REST URL.

`REST Authentication` describes the authentication mechanism used to connect to the target.

`BasicAuth` indicates the authentication mechanism is Basic Authentication.

- Field Mapping Information

```
<FieldMappingInfo>
```

`FieldMappingInfo` provides mapping information from target fields to various Audit Vault fields, contained in these child elements, `CoreFields`, `LargeFields`, `ExtensionField`, and `MarkerField`.

Field mappings include `<Map>` elements, which contain `<Name>` elements that hold target field names, and `<MapTo>` elements that hold Audit Value field names that targets are mapped to.

There should be no many-to-one mappings from target fields to Audit Vault Server fields. For example, the following is invalid:

```
<!-- Invalid code
<Map>
    <Name>$.USER_ID</Name>
    <MapTo>UserName</MapTo>
</Map>
<Map>
    <Name>$.OS_USER_ID</Name>
    <MapTo>UserName</MapTo>
</Map>  -->
```

– Core Fields

  `<CoreFields>`

  `CoreFields` provides mapping from target fields to core fields of Audit Vault Server. Target fields specified in core field mappings must be of SQL data type, either a string or a data type that can convert to string.

  The following elements contain core fields.

```
<Map>
 <Name>$.EVENT_TIME</Name>
  <MapTo>EventTimeUTC</MapTo>
  <TimestampPattern>yyyy-MM-dd HH:mm:ss.SSS</TimestampPattern>
</Map>
```

  `EventTimeUTC` provides event time mapping information. The value in `TimestampPattern` specifies the timestamp format for event time. `EventTimeUTC` and `TimestampPattern` are mandatory.

  When specifying the `TimestampPattern`, use the supported patterns and characters of the Java `SimpleDateFormat` class, *NOT* Oracle Database specific patterns.

  For multibyte characters such as Chinese, specific words such as Month should be added into the pattern as characters in `SimpleDateFormat`. The AM and PM indicators are obtained based on locale, but should be explicitly mentioned in the TimestampPattern that you provide in the mapper file.

```
<Map>
  <Name>$.USER_ID</Name>
  <MapTo>UserName</MapTo>
```

```
</Map>
```

`UserName` represents the user who performed the action. If the mapping is not provided, Audit Data Collection still starts successfully, but every audit record is treated as invalid.

```
<Map>
  <Name>$.OS_USER_ID</Name>
  <MapTo>OSUserName</MapTo>
</Map>
 <Map>
  <Name>$.ACTION</Name>
  <MapTo>CommandClass</MapTo>
</Map>
```

`CommandClass` represents the action of the event. If the mapping is not provided, Audit Data Collection still starts successfully, but all audit records are treated as invalid.

```
<Transformation>
  <ValueTransformation from="1" to="CREATE"/>
  <ValueTransformation from="2" to="INSERT"/>
  <ValueTransformation from="3" to="SELECT"/>
  <ValueTransformation from="4" to="CREATE"/>
  <ValueTransformation from="15" to="READ"/>
  <ValueTransformation from="30" to="LOGON"/>
  <ValueTransformation from="34" to="LOGOFF"/>
  <ValueTransformation from="35" to="ACQUIRE"/>
</Transformation>
```

`CommandClass` contains a `Transformation` field with `ValueTransformation` values, from targets to the Audit Vault Server `CommandClass` field. These transformations are mandatory.

The *to* attributes are values for the `CommandClass` field. If you can meaningfully map an event to one of these values, Oracle recommends that you do so. If this is not possible, use a value that appropriately reflects the action that generated the audit event.

```
<Map>
  <Name>$.OBJ_NAME</Name>
  <MapTo>TargetObject</MapTo>
</Map>
<Map>
  <Name>$.USER_HOST</Name>
  <MapTo>ClientHostName</MapTo>
</Map>

<Map>
   <Name>$.TERMINAL</Name>
   <MapTo>TerminalName</MapTo>
```

```
</Map>
<Map>
  <Name>$.OBJ_CREATOR</Name>
  <MapTo>TargetOwner</MapTo>
</Map>
<Map>
  <Name>$.STATUS</Name>
  <MapTo>EventStatus</MapTo>
        <Transformation>
          <ValueTransformation from="0" to="FAILURE"/>
          <ValueTransformation from="1" to="SUCCESS"/>
          <ValueTransformation from="2" to="UNKNOWN"/>
        </Transformation>
</Map>
```

`EventStatus` contains a `Transformation` field with `ValueTransformation` values, from targets to Audit Vault `EventStatus` fields. These transformations are mandatory.

```
</CoreFields>
```

- Large Fields Information

```
<LargeFields>
  <Map>
    <Name>$.SQL_TEXT</Name>
    <MapTo>CommandText</MapTo>
  </Map>
<Map>
   <Name>$.COMMAND_PARAMETER</Name>
   <MapTo>CommandParam</MapTo>
  </Map>
</LargeFields>
```

`LargeFields` are target fields mapped to large fields in the Audit Vault Server. The specified target fields must be of SQL data type `CLOB` or `String`, or be convertible to `String`.

- Extension Fields

```
<ExtensionField>
   <Name>$.DB_ID</Name>
   <Name>$.INSTANCE</Name>
   <Name>$.PROCESS</Name>
</ExtensionField>
```

`ExtensionField`s are target field names that must be stored as a name-value pair in the `Extension` field in Audit Vault Server. Target fields specified must be of SQL data type `CLOB` or `String`, or be convertible to `String`.

– Marker Fields

```
<MarkerField>
  <Name>$.SESSION_ID</Name>
  <Name>$.ENTRY_ID</Name>
</MarkerField>
```

`MarkerField` contains a list of target fields that uniquely identify each audit record. The target fields specified must be of SQL data type `CLOB` or `String`, or be convertible to `String`. `MarkerField` is mandatory.

See Also:

- JSON REST Collection Plug-in Example
- Core Fields
- Actions and Target Types
- Large Fields
- Extension Field
- Marker Field

3.7 Target Collection Attributes

You must define collection attributes before you can use Oracle AVDF plug-ins to collect data from the audit trail.

For Database Table plug-in, JSON, CSV, XML file collection plug-ins, and REST plug-in, you need to set the audit collection attributes during target registration. This has to be completed after deploying the collection plug-in in the Audit Vault Server and before starting the audit trail which uses the plug-in.

You define collection attributes using the `AVCLI` command `ALTER SECURED TARGET`.

Required target attributes are:

- `av.collector.securedtargetversion` **(Mandatory)**: Current version of the target. This version information helps in choosing the correct mapper file for the audit trail if there are multiple mapper files in the `templates` directory of the collection plug-in.
- `av.collector.atcintervaltime`: The collection plug-in writes the time, up to which audit data has been collected from the trail, to a file. This file will be present in the `av/atc` directory in the agent home. Also, this file contains the time in UTC time zone. This information can help some third party utilities to clean up audit data from a trail. Note that collection plug-in does not perform the audit data clean-up, it just writes this information to a file. `atcintervaltime`: specifies how frequently the collection plug-in should update the time information in the file. The value of the attribute is in minutes.
- `av.collector.timezoneoffset` **(Mandatory)**: Offset of the target event time from UTC time zone. This helps the collector to report event time correctly to the Audit Vault Server by adjusting the time zones. This attribute is not needed for an XML file collection plug-in

if the event time itself contains the time zone information. An example of this setting is as follows:

```
av.collector.TimeZoneOffset = +5:30
```

- `av.collector.enableArchivedTime` **(Optional)**: This attribute is applicable only for Oracle table trail. It is set to `true` by default. When `enableArchivedTime` is set to `true` and if `INIT_CLEANUP` procedure is called for the trail, then the last archived timestamp in `dbms_audit_mgmt` package is updated to the current checkpoint time based on the ATC interval time. When set to `false`, the last archived timestamp for the trail is not updated in the target database. In that case, the user must ensure that the audit records with timestamps less than the checkpoint time should not be purged from the audit table. The user can view the Audit Vault Server database checkpoint table in `AVSYS` schema to verify the checkpoint time of the trail until the records have been collected. If you want to change the attribute value, then the trail must be restarted after the attribute has been updated.

3.8 Preprocessing Audit Data

Learn about the requirements to use Oracle AVDF collection plug-ins.

In general, collection plug-ins can only be used to collect audit trails that conform to the requirements presented in this chapter.

For other audit trails, you can use the Audit Vault Java API.

However, there can be other reasons why you cannot collect audit records directly with a collection plug-in, but you can collection them indirectly.

It can be possible to preprocess these audit trails to generate entries in database tables or XML files in a format that allows collection plug-ins to collect them. For example, IBM DB2 on Linux, Unix, and Microsoft Windows all require you to execute the `db2audit` program to extract audit records from a proprietary binary format into a text file. To extract new records, you must run this program periodically as the user who owns the DB2 software.

While you cannot define a collection plug-in to read the file directly, It is possible that you can write a program that reads the file periodically, extracts new audit records, and writes them to a new XML file in a directory. Each run of this program can create a new XML file that contains only the new records. You can then define a collection plug-in to read these XML files, and collect the audit records into Oracle Audit Vault Server.

Related Topics

- Java-Based Audit Trail Collection Plug-ins
  Oracle Audit Vault and Database Firewall provides a set of Java-based audit trial collection plug-in, which enable you to create custom plug-ins.

4
Java-Based Audit Trail Collection Plug-ins

Oracle Audit Vault and Database Firewall provides a set of Java-based audit trial collection plug-in, which enable you to create custom plug-ins.

4.1 About Java-Based Collection Plug-ins

For situations where a template-based collection plug-in cannot easily handle audit data, you can use Java-based collection plug-ins

Creating a custom collection plug-in using the Java APIs gives you much flexibility in how you design your collection plug-in.

In general, use the Java type of collection plug-in if you need it to:

- Read trails not written in database tables or XML files.
- Read complex trails written to tables or XML files.

4.2 JDK Requirement for Using the Java-Based Collection Plug-in

To use a Java-based collection plug-in with Oracle Audit Vault and Database Firewall, you must have the JDK to compile and test your code.

Because the collection plug-in runs under the same JVM after it is shipped, Oracle recommends that you use the same JDK as the JDK that you use to start the agent. Compile your classes with the JDK by setting the `-target` option of the `javac` compiler to the same JDK version. Refer to the JDK documentation for details.

4.3 About the Flow of Control Inside the Java-Based Collection Plug-in

Learn how Oracle Audit Vault accesses an audit trail, maps the trail to Oracle Audit Vault events, starts the correct Java-based collection plug-in, and creates audit records.

When a collection plug-in accesses an audit trail, it extracts an audit record and its related fields from the audit trail. Next, it maps the audit record to an Oracle Audit Vault event, and all the fields to Oracle Audit Vault fields. The collection plug-in then passes the Oracle Audit Vault event and fields to the Collection Framework, which sends the information to the Oracle Audit Vault Server.

The sequence of control processes for the audit trail collection is as follows.

Control Process Sequence for Audit Trail Collection

1. The Oracle Audit Vault Server commands the Agent Framework to create a thread to collect from a specific audit trail.

2. The new thread, just created by the agent, collects a specific audit trail.

   At this point, control is handed to the Collection Framework.

3. Within the thread, the Collection Framework connects to the Oracle Audit Vault Server, and queries for configuration information for the audit trail being collected.

   In addition, it requests information for the last checkpoint set for that trail.

4. With the information it now has, the Collection Framework uses the plug-in manifest file to determine the correct Java class to start within the correct collection plug-in. It passes the configuration information to this class, and asks it to initialize itself.

5. After the collection plug-in has initialized itself, the Collection Framework loops repeatedly. Within each loop, the Collection Framework does the following:

   - Asks the collection plug-in for any additional audit records in the audit trail.

     The collection plug-in transforms (by mapping) any further audit records into the form of audit records that Audit Vault expects, and hands them to the Collection Framework through the Collection API.

6. The collection plug-in can periodically send Checkpoint and metric information to the Collection Framework. The collection plug-in can do so in the same flow when it has the control, for example when the Collection Framework calls `hasNext()`.

7. If the Oracle Audit Vault Server sends commands to the Collection Framework, the Collection Framework passes them to the collection plug-in to act on.

   If the Collection Framework receives a `STOP` command from the Audit Vault Server, it notifies the collection plug-in to stop sending record. Then it exits the collection thread and shuts itself down.

   If the Collection Framework receives a `RECONFIGURE` command from the Audit Vault Server, it notifies the collection plug-in to set an attribute using `setAttribute()`.

Related Topics

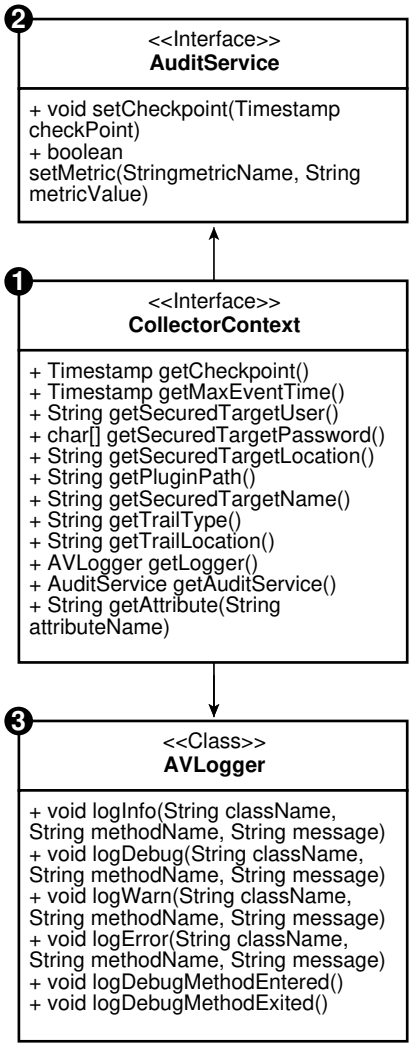
- Collection Thread
  Learn about how Oracle Audit Vault collection threads are run.
- Collection Phase
  The **collection phase** is the phase in which Audit Vault collects audit trail records.
- Checkpoint of a Trail
  A **Checkpoint**, or a checkpoint of a trail, is the point up to which audit records were committed to the Oracle Audit Vault Server.
- Mapping
  The mappings required from targets to Audit Vault Server depends on the fields in the target records.

4.4 Useful Classes and Interfaces in the Collection Framework

Learn about Oracle Audit Vault Java-based collection plug-in classes and interfaces that can be particularly useful for your own collections

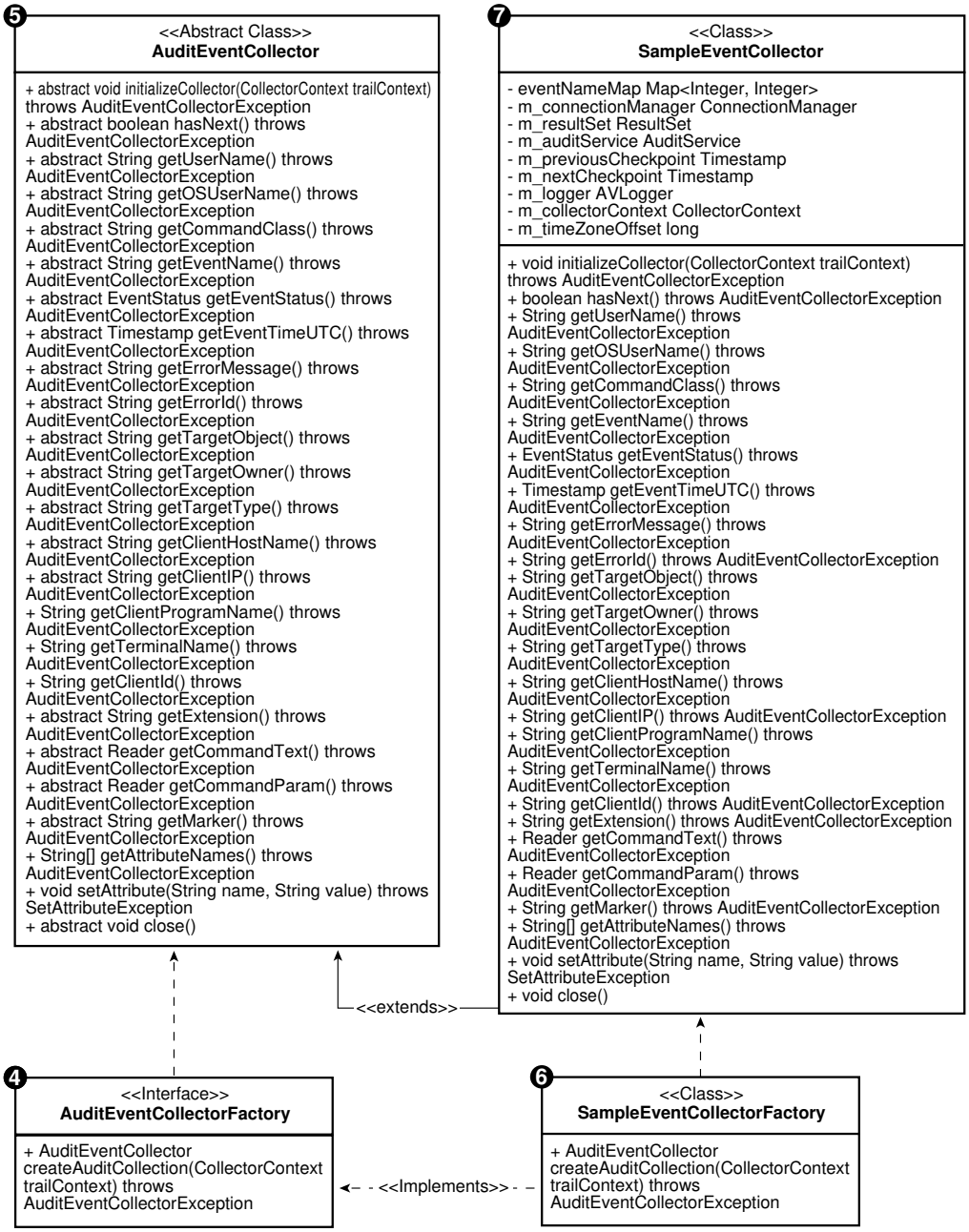
The image below shows the relationships between the classes and interfaces from the AuditService, CollectorContext, and Class AVLogger.

Figure 4-1 Classes and Interfaces from AuditService, CollectorContext, and Class AVLogger



The following diagram shows the various classes and interfaces in the Collection Framework that you need to know about to write a Java-based collection plug-in.

Figure 4-2 Classes and Interfaces from Collection Framework Used in Collection Plug-in



4.5 How to Create a Java-Based Collection Plug-in

Review the tasks required to create and use Java-based collection plug-ins for Oracle AVDF.

4.5.1 About Creating a Java-Based Collection Plug-in

The Oracle Audit Vault documentation provides examples of a Java-based collection plug-in implementation, including a hypothetical source that writes events to a table named `AUD`.

Implementing Oracle Audit Vault and Database Firewall involves writing Java classes that implement the `AuditEventCollectorFactory` interface, and that extend the `AuditEventCollector` class, which are part of the Audit Vault Collection Framework. The same Java class can both extend the `AuditEventCollector` class. and implement the `AuditEventCollectorFactory` interface. Alternately, you can choose to write two separate classes.The sample consists of two classes, `SampleEventCollectorFactory` which implements the `AuditEventCollectorFactory` interface and `SampleEventCollector` which extends from the `AuditEventCollector` class.

Related Topics

- Example Audit Trail for a Database Table Collection Plug-in
  This example audit trail shows the details of audit trail. This example file is used in other locations to demonstrate the creation and structure of a sample mapper file for Oracle Audit Vault and Database Firewall.
- Java Collection Plug-in Code
  This examples shows a complete Java-based collection plug-in.

4.5.2 Using the AuditEventCollectorFactory to Get the AuditEventCollector Object

During runtime the collection plug-in can require multiple implementations of AuditEventCollector. The AuditEventCollectorFactory object enables this capability for Oracle Audit Vault and Database Firewall.

The Collection Framework does not create an instance of the `AuditEventCollector` object directly. Instead, it creates an instance of the `AuditEventCollectorFactory` class and using the factory object, gets the `AuditEventCollector` object. This is because the collection plug-in may require multiple implementations of `AuditEventCollector`. The collection plug-in decides at run time which implementation to use. Therefore, every collection plug-in should have an implementation of `AuditEventCollectorFactory`.

In the following example, the `createAuditEventCollector()` always creates and returns an instance of the `SampleAuditEventCollector` class.

Example 4-1 Creating a SampleAuditEventCollector Class

```
public class SampleEventCollectorFactory implements
AuditEventCollectorFactory {

   public AuditEventCollector createAuditCollection(
         CollectorContext collectorContext) throws
AuditEventCollectorException {
      return new SampleEventCollector();
   }
}
```

4.5.3 Using the CollectorContext Class When Creating a Java-Based Collection Plug-in

Learn how to use source attributes, which provide the Oracle AVDF collection plug-in with information about the source that is needed to collect the audit trail effectively.

The Collection Framework passes an instance of the `CollectorContext` class to the collection plug-in through the `initializeCollector` method. This instance can be queried by the collection plug-in to obtain information needed to collect the audit trails generated by the source.

4.5.3.1 Basic Source Attributes

To obtain audit trail collection successfully, there are basic source attributes that provide the collection plug-in with information about the source for Oracle Audit Vault and Database Firewall.

Basic source attributes for Oracle Audit Vault and Database Firewall include the user name, password, and connection string. These attributes are returned by these methods respectively: `getSecuredTargetUser`, `getSecuredTargetPassword`, and `getSecuredTargetLocation`. You can retrieve pther source attributes by using `getAttributes`.

When the Oracle Audit Vault administrator registers the source, you can require the Oracle Audit Vault administrator to provide the required information, in the form of source attributes. Oracle Audit Vault stores these attributes in the Oracle Audit Vault Server repository, and provides them to the collector code on startup.

Some collection plug-ins do not need to connect to the source and in these cases, the Collection Framework may return null for these methods.

Related Topics

- Additional Source or Trail Attributes
  You can retreive other attributes that you find the Oracle Audit Vault and Database Firewall collector needs by passing the attribute name to the `getAttribute` method.

4.5.3.2 Basic Trail Attributes

The checkpoint and trail name attributes are the basic attributes that Oracle Audit Vault and Database Firewall obtains.

Checkpoint attributes are returned by the `getCheckpoint` method, and trail name attributes are returned by the `getTrailLocation` method. The collection plug-in should use these attributes as follows:

- The checkpoint returned is the last checkpoint the collection plug-in has set for this trail when it ran the last time. The collection plug-in should start sending only those records which have an event time greater than or equal to the checkpoint. If the collection plug-in is starting for the first time, the collection plug-in receives this value as null. In this case, the collection plug-in must send the records from the beginning.

- Trail name indicates the target containing the audit events, typically, a table or directory name.

You can retrieve other trail attributes by using `getAttributes`.

Related Topics

- Additional Source or Trail Attributes
  You can retreive other attributes that you find the Oracle Audit Vault and Database Firewall collector needs by passing the attribute name to the `getAttribute` method.

4.5.3.3 Utility Instances

Learn how to use the AVLogger and AuditService attributes with the Oracle Audit Vault collector.

`AVLogger` and `AuditService` are returned by the `getLogger` and `getAuditService` methods, respectively. The collector should use these attributes as follows:

- The `AVLogger` instance logs various messages.
- The `AuditService` instance sends checkpoints and metrics to the Audit Vault Server.

These methods never return null.

Related Topics

- Java-Based Collection Plug-in Utility APIs
  In addition to the Collection Framework, the Oracle Audit Vault API includes Java utility APIs that make the task of writing a collector easier.

4.5.3.4 Additional Source or Trail Attributes

You can retreive other attributes that you find the Oracle Audit Vault and Database Firewall collector needs by passing the attribute name to the `getAttribute` method.

For example, suppose that a source required `SourceVersion` to collect audit data. In that scenario, to obtain the value of `SourceVersion`. the collector for the source calls `collectorContext.getAttribute('SourceVersion')`.

If the attribute is present, then the `getAttribute` method returns the attribute value as a String. Otherwise, the method returns null.

If the collector receives a null or an invalid value for any mandatory attribute, then it must throw an `AuditEventCollectorException` exception from the `initializeCollector` method. After throwing an exception, the Collection Framework shuts down.

Related Topics

- Changing Oracle AVDF Attributes at Run Time
  If you are an administrator, then you can change the attributes that a Java-based collector plug-in uses during Oracle AVDF audit trail collection.

4.5.4 Initializing the Java-Based Collection Plug-in

Use these examples to understand how to initialize a Java-based collection plug-in, and how to start audit events collection with Oracle Audit Vault and Database Firewall.

The first thing the Collection Framework does after the collection thread starts is to initialize the collector.

The Collection Framework calls the `initializeCollector()` method of the `AuditEventCollector` class. The collector sets up the environment appropriately to enable it to start collecting audit events. For example, for a database table collection plug-in, this method connects to the database. For an XML file collection plug-in, this method parses the file mask and may open a particular file to start with. The collection plug-in may also want to retrieve various attributes from the collector context at this point. If there is an error in setting up the environment, this method throws `AuditEventCollectorException` with an appropriate error message.

Example 4-2 Initializing a Java-Based Collection Plug-in

The following is a example of how to initialize a Java-based collection plug-in:

```
private AVLogger m_logger;
   private CollectorContext m_collectorContext;
   private long m_timeZoneOffset;
   private AuditService m_auditService;
   private Timestamp m_previousCheckpoint;

   public void initializeCollector(CollectorContext collectorContext)
         throws AuditEventCollectorException {
      m_collectorContext = collectorContext;
      m_auditService = m_collectorContext.getAuditService();
      m_previousCheckpoint = m_collectorContext.getCheckpoint();
      m_logger = m_collectorContext.getLogger();
      // Get other attributes of the Source.
      String offset =
m_collectorContext.getAttribute("TimeZoneOffset");
      if (offset != null) {          m_timeZoneOffset =
getTimeZoneOffsetInMs(offset);
      }
      connectToSource();
 }
```

Example 4-3 Using the ConnectionManager Utility to Connect and Retrieve Audit Records From a Database

If a collector must connect to a database to retrieve audit records, then it must use the `ConnectionManager` utility API provided with Oracle Audit Vault. The following example shows how to use the ConnectionManager utilityL

```
private ConnectionManager m_connectionManager;

   private void connectToSource() throws AuditEventCollectorException {
      m_logger.logDebugMethodEntered();
      // Get connection information from collector context.
```

```
      String user = m_collectorContext.getSecuredTargetUser();
      String password = new
String(m_collectorContext.getSecuredTargetPassword());
      String connectionString =
m_collectorContext.getSecuredTargetLocation();
      // Create a ConnectionManager object.
      try {
         m_connectionManager = new ConnectionManagerImpl(connectionString,
               user, password.toCharArray());
         m_connection = m_connectionManager.getConnection();
      } catch (AuditException ex) {
         throw new AuditEventCollectorException(
               ErrorCodes.FAILED_CONNECT_TO_SOURCE,
               new Object[] { connectionString }, ex);
      }
      m_logger.logDebugMethodExited();
   }
```

Related Topics

- Using the AVLogger API to Log Messages
  To log errors, warnings, informational, and debug messages into the Oracle Audit Vault and Database Firewall logs, you can use the `AVLogger` API.
- Example of Using the ConnectionManager API to Connect to Database Sources
  See how to use the ConnectionManager API to manage Oracle Audit Vault and Database Firewall Java compnent connections to databases.

4.5.5 Connecting, Fetching Events, and Setting Checkpoints

Use these examples to understand how to connect a Java-based collection plug-in, how to fetch events, and how to set checkpoints with Oracle Audit Vault and Database Firewall.

After initialization, the Collection Framework repeatedly calls the `hasNext()` method of the collector, which internally calls the `fetchEvents()` method. In this method, the collector fetches audit records from the audit trail in the form of a `ResultSet`.

The range that is fetched starts from the point that was just finished to nearly the current time. The next fetch is performed when the current `ResultSet` is exhausted

The collection plug-in sets the checkpoint whenever one `ResultSet` finishes processing, but before the next one starts. The timing of this is important. When a collection plug-in sets the checkpoint with `Timestamp` *`t`*, the collection plug-in must ensure that all records with an event time less than *`t`* are already sent to Collection Framework. However, because `ResultSet` does not give records in any particular order, setting the checkpoint before the end of a `ResultSet` can be incorrect. Additionally, the collection plug-in does not use the current time as the upper bound of the range, but rather uses the current time minus the delta time. Using this time method allows for possible delays between generating the event, and inserting the event into the table. When the collection plug-in runs the query, all events up to the upper bound are fetched in the `ResultSet` to honor the checkpoint contract. The 5-second delay time (the delta) ensures that all of the records, up to the upper bound, are already in the table.

Example 4-4 Fetching ResultSets and Setting Checkpoints

This example shows how collectors should obtain the `Connection` from `ConnectionManager` whenever they need one.

```
   private ResultSet m_resultSet;
   private Timestamp m_nextCheckpoint;

   private void fetchEvents() throws AuditEventCollectorException {
      m_logger.logDebugMethodEntered();
      if (m_nextCheckpoint != null) {
         m_auditService.setCheckpoint(m_nextCheckpoint);
         m_previousCheckpoint = m_nextCheckpoint;
      }
     // It is not good to hold on to the Connection for long. As this
is the
      // only place we can release the connection, we release and
reacquire the
      // connection.
      try {
         if (m_connection != null) {
            m_connectionManager.releaseConnection(m_connection);
         }
      } catch (AuditException ex) {
         throw new AuditEventCollectorException(
               ErrorCodes.FAILED_TO_RELEASE_CONNECTION_TO_DB, null,
ex);
      }

      try {
         m_connection = m_connectionManager.getConnection();
      } catch (AuditException ex) {
         throw new AuditEventCollectorException(
               ErrorCodes.FAILED_TO_GET_CONNECTION_TO_DB, null, ex);
      }

     // This is the upper bound which is current time minus 5 seconds.
      m_nextCheckpoint = new Timestamp(System.currentTimeMillis() -
5000);
     String query = null;
      try {
         if (m_previousCheckpoint == null) {
            query = "select * from AUD where EVENT_TIME <= ?";
            m_preparedStatement = m_connection.prepareStatement(query);
            m_preparedStatement.setTimestamp(1, m_nextCheckpoint);
         } else {
            query = "select * from AUD where EVENT_TIME > ? and
EVENT_TIME <= ?";
            m_preparedStatement = m_connection.prepareStatement(query);
            m_preparedStatement.setTimestamp(1, m_previousCheckpoint);
            m_preparedStatement.setTimestamp(2, m_nextCheckpoint);
         }
         m_resultSet = m_preparedStatement.executeQuery();
      } catch (SQLException ex) {
         throw new AuditEventCollectorException(
```

```
                ErrorCodes.ERROR_GETTING_DATA_FROM_SOURCE,
                new Object[] { query }, ex);
        }
        m_logger.logDebugMethodExited();
    }
```

Example 4-5 Using hasNext to Fetch Records

```
hasNext()fetchEvents()

     public boolean hasNext() throws AuditEventCollectorException {
      boolean hasMore;
      try {
         if(m_resultSet == null) {
            fetchEvents();
            return m_resultSet.next();
         }
         hasMore = m_resultSet.next();
         if (!hasMore) {
            fetchEvents();
            hasMore = m_resultSet.next();
         }
      } catch (SQLException ex) {
         throw new AuditEventCollectorException(
               ErrorCodes.ERROR_GETTING_DATA_FROM_SOURCE, null, ex);
      }
      return hasMore;
    }
```

4.5.6 Transforming Source Event Values to Audit Vault Event Values

Learn about when and how the Oracle AVDF collection plug-in transforms source event values to Oracle AVDF values.

The collector retrieves values of specific fields from source. For some fields, in addition to retrieving the values, the collection plug-in must transform the values in certain ways. This section discusses transformations that are required for all source types.

4.5.6.1 Event Time to UTC

See how Oracle Audit Vault and Database Firewall transforms an event time from a source time zone to Coordinated Universal Time (UTC).

Event Time should be sent only in UTC time zone. Therefore, it must be transformed from the source time zone to the UTC time zone before returning a value. If the column from the source database is timezone aware, then this transformation is not necessary.

Example 4-6 Transforming EventTime from Source Time Zone to UTC

```
   public Timestamp getEventTimeUTC() throws AuditEventCollectorException {
      try {
         Timestamp eventTime = m_resultSet.getTimestamp("EVENT_TIME");
         // As the method name suggests, the timestamp must be returned only
in
```

```
            // UTC timone.
            return new Timestamp(eventTime.getTime() - m_timeZoneOffset);
         } catch (SQLException ex) {
            throw new AuditEventCollectorException(
                  ErrorCodes.ERROR_GETTING_DATA_FROM_SOURCE, null, ex);
         }
      }
```

4.5.6.2 Source Event Name to Audit Vault Event Name

Use this example to create your own source event names to Oracle Audit Vault event names mapping.

Each source event name maps to one Oracle Audit Vault event name. The `getCommandClass()` method should transform the source event name into a value that Oracle Audit Vault can accept.

Example 4-7 Mapping Source Event Names to Audit Vault Event Names

```
private static final Map<Integer, String> eventNameMap = new
HashMap<Integer,
         String>();
static {
      eventNameMap.put(1, "CREATE");
      eventNameMap.put(2, "INSERT");
      eventNameMap.put(3, "SELECT");
      eventNameMap.put(4, "CREATE");
      eventNameMap.put(15, "ALTER");
      eventNameMap.put(30, "AUDIT");
      eventNameMap.put(34, "CREATE");
      eventNameMap.put(35, "ALTER");
      eventNameMap.put(51, "CREATE");
      eventNameMap.put(52, "CREATE");
   }

   public String getCommandClass() throws AuditEventCollectorException
{
      try {
         int eventId = m_resultSet.getInt("ACTION");
         return eventNameMap.get(eventId);
      } catch (SQLException ex) {
         throw new AuditEventCollectorException(
               ErrorCodes.ERROR_GETTING_DATA_FROM_SOURCE, null, ex);
      }
   }
```

Related Topics

- Oracle Audit Vault and Database Firewall Fields
  Oracle Audit Vault and Database Firewall values consist of core fields, large fields, marker fields, and extension fields.

4.5.6.3 Source Event ID to Source Event Name

Learn how to map the source event identifiers (IDs) to descriptive source event names in Oracle Audit Vault and Database Firewall.

For some sources, events reported as IDs may not mean anything if users are unfamiliar with the IDs. Therefore, it may be best to map the source event IDs to descriptive source event names. If the audit record itself contains descriptive event names, then they can directly be returned without any mapping. The source event name is optional, so the collection plug-in can return null if it does not have the information.

In the following example, you can see how to map the source event IDs to descriptive source event names.

Example 4-8 Mapping Source Event Ids to Source Event Names

```
private static final Map<Integer, String> sourceEventMap =
        new HashMap<Integer, String>();

   static {
      targetTypeMap.put(1, "OBJECT:CREATED:TABLE");
      targetTypeMap.put(2, "INSERT INTO TABLE");
      targetTypeMap.put(3, "SELECT FROM TABLE");
      targetTypeMap.put(4, "OBJECT:CREATED:TABLE");
      targetTypeMap.put(15, "OBJECT:ALTERED:TABLE");
      targetTypeMap.put(30, "AUDIT OBJECT");
      targetTypeMap.put(34, "OBJECT:CREATED:DATABASE");
      targetTypeMap.put(35, "OBJECT:ALTERED:DATABASE");
      targetTypeMap.put(51, "OBJECT:CREATED:USER");
      targetTypeMap.put(52, "OBJECT:CREATED:ROLE");
   }

   public String getEventName() throws AuditEventCollectorException {
      try {
         int eventId = m_resultSet.getInt("ACTION");
         return sourceEventMap.get(eventId);
      } catch (SQLException ex) {
         throw new AuditEventCollectorException(
               ErrorCodes.ERROR_GETTING_DATA_FROM_SOURCE, null, ex);
      }
   }
```

4.5.6.4 Mapping Source Event Name or ID to Target Type

Learn how to map source event identifiers (IDs) to Oracle Audit Vault target types.

A **target type** is the type of the object on which an event has taken place. For example, if the event is a `SELECT` operation on a table, then the target type is table. In some sources, the target type can be present within the source event name and ID. For example, an event name can be `select table`, which implies that the target type is a table. In this case, you must map the source event name or ID to a target type. Target type is an optional field, so the collection plug-in can return null if there is no such information.

In the following example, source event IDs are mapped to Oracle Audit Vault target types.

Example 4-9 Mapping Source ID to Target Type

```
private static final Map<Integer, String> targetTypeMap =
      new HashMap<Integer, String>();

   static {
      targetTypeMap.put(1, "TABLE");
      targetTypeMap.put(2, "TABLE");
      targetTypeMap.put(3, "TABLE");
      targetTypeMap.put(4, "CLUSTER");
      targetTypeMap.put(15, "TABLE");
      targetTypeMap.put(30, "OBJECT");
      targetTypeMap.put(34, "DATABASE");
      targetTypeMap.put(35, "DATABASE");
      targetTypeMap.put(51, "USER");
      targetTypeMap.put(52, "ROLE");
   }

   public String getTargetType() throws AuditEventCollectorException {
      try {
         int eventId = m_resultSet.getInt("ACTION");
         return targetTypeMap.get(eventId);
      } catch (SQLException ex) {
         throw new AuditEventCollectorException(
               ErrorCodes.ERROR_GETTING_DATA_FROM_SOURCE, null, ex);
      }
   }
```

4.5.6.5 Source Event Status to Oracle Audit Vault Event Status

Oracle Audit Vault has three `EventStatus` values. See how you can transform source event status values to the Oracle Audit Vault values.

There are only three allowed values for `EventStatus`. They are `SUCCESS`, `FAILURE`, and `UNKNOWN`. You must configure transformations of any source event values to one of the three supported Oracle Audit Vault values. In the following example, you can see how source values are transformed to Oracle Audit Vault values.

Example 4-10 Transforming Source Values to Oracle Audit Vault EventStatus Values

```
   public EventStatus getEventStatus() throws
AuditEventCollectorException {
      try {
         int status = m_resultSet.getInt("STATUS");
         if (status == 1) {
            return EventStatus.SUCCESS;
         } else if (status == 0) {
            return EventStatus.FAILURE;
         } else {
            return EventStatus.UNKNOWN;
         }
      } catch (SQLException ex) {
         throw new AuditEventCollectorException(
```

```
            ErrorCodes.ERROR_GETTING_DATA_FROM_SOURCE, null, ex);
      }
   }
```

4.5.7 Retrieving Other Audit Field Values

When field values do not require transformations, the Oracle Audit Vault Java-based collection plug-in returns the value it obtains from the source.

In the following example, the Oracle Audit Vault collection plug-in obtains a user name, and returns it.

Example 4-11 Returning Values that Do Not Need Transformation

```
public String getUserName() throws AuditEventCollectorException {
   try {
      return m_resultSet.getString("USER_ID");
   } catch (SQLException ex) {
      throw new AuditEventCollectorException(
            ErrorCodes.ERROR_GETTING_DATA_FROM_SOURCE, null, ex);
   }
}
```

4.5.8 Changing Oracle AVDF Attributes at Run Time

If you are an administrator, then you can change the attributes that a Java-based collector plug-in uses during Oracle AVDF audit trail collection.

If you are an Oracle AVDF administrator, then you can update attributes, including source attributes, at any time. To update attributes, you can use either the Audit Vault Server console, or you can use the `AVCLI` command-line tool.

How Audit Vault Server Manages Attribute Updates

If the update occurs while collectors are collecting audit trails, then first Audit Vault Server notifies all running collectors dynamically, by calling the `setAttribute` method of the collector. Next, the collection plug-in must start using the new value immediately. If the collection plug-in is not configured for the attribute value, or if the value cannot be used, then the collector responds with the message `SetAttributeException`.

In the following example, the collection plug-in receives and handles a new time zone offset to use in converting the `EventTime` to Coordinated Universal Time (UTC) time zone for all subsequent events.

Example 4-12 Changing an Oracle Audit Vault and Database Firewall Attribute

```
public void setAttribute(String name, String value)
         throws SetAttributeException {
           if (name.equalsIgnoreCase("TimeZoneOffset")) {
              m_timeZoneOffset = getTimeZoneOffsetInMs(value);
      } else {
         throw new SetAttributeException(ErrorCodes.INVALID_ATTRIBUTE_NAME,
               new Object[] { name, value }, null);
```

```
      }
   }
```

Effects After Modifying Attributes

Use the preceding example to understand how the time zone offset transformation is affected in the example you can find in the following topic:

Event Time to UTC

Related Topics

- Using the CollectorContext Class When Creating a Java-Based Collection Plug-in
  Learn how to use source attributes, which provide the Oracle AVDF collection plug-in with information about the source that is needed to collect the audit trail effectively.
- Using Exceptions in Collection Plug-ins
  An Oracle Audit Vault and Database Firewall collector can generate several different types of exceptions.

4.5.9 Changing Custom Attributes at Run Time

See how administrators can change custom collector plug-in attributes at runtime while Oracle Audit Vault and Database Firewall collects the audit trail.

As with Oracle Audit Vault and Database Firewall (Oracle AVDF) attributes, you can also change custom attributes at runtime.

If you want to change custom attributes, then you must implement the following methods to validate the custom attributes before you can use custom attributes in the `setAttribute` method.

Example 4-13 Changing a Custom Attribute

```
private static final String[] s_attributes = new String[]
{ "av.collector.configureParameter1",
"av.collector.configureParameter2" };
public String[] getAttributeNames() throws
AuditEventCollectorException {
         return s_attributes.clone();
 }

public void setAttribute(String name, String value)
         throws SetAttributeException {
           if (name.equalsIgnoreCase("configureParameter1")) {
           // use value
           }else if (name.equalsIgnoreCase("configureParameter2"))
           {
           // use value
           }else {
         throw new
SetAttributeException(ErrorCodes.INVALID_ATTRIBUTE_NAME,
               new Object[] { name, value }, null);
      }
   }
```

4.5.10 Creating Extension Fields

See an example of how to create an extension field for Java-based collection plug-ins in Oracle Audit Vault and Database Firewall.

The **extension field** contains all the fields of the source event which are of interest to the user, but do not correspond to any of the core or large fields. The collector must form one string which contains the names and values of these extra fields. The format of this string is up to the collection plug-in. The Collection Framework never tries to parse this string. In this example of creating an extension field, it uses the following format, repeating as needed:

```
<field_name>=<field_value>;
```

Note that this example extension file sends three fields:

Example 4-14 Creating an Extension Field

```
public String getExtension() throws AuditEventCollectorException {
    try {
       StringBuilder sb = new StringBuilder();
       sb.append("DB_ID=" + m_resultSet.getString("DB_ID") + ";");
       sb.append("INSTANCE=" + m_resultSet.getString("INSTANCE") + ";");
       sb.append("PROCESS=" + m_resultSet.getString("PROCESS"));
       return sb.toString();
    } catch (SQLException ex) {
       throw new AuditEventCollectorException(
             ErrorCodes.ERROR_GETTING_DATA_FROM_SOURCE, null, ex);
    }
 }
```

4.5.11 Handling Large Audit Fields

See an example of how Oracle Audit Vault and Database Firewall handles very large audit record fields.

Some audit record fields can be very large, so that returning them as a string is not feasible. Therefore, methods corresponding to those fields return an object of type `Reader`. If the source field is a character large object (a`CLOB`), then the `Reader` can be obtained by using `clob.getCharacterStream()`.

> **Note:**
>
> The `Reader` is only valid as long as the Connection to the source is alive.

You must design the collection plug-in to keep the connection to the source alive until all events using readers have been sent to the Audit Vault Server.

If the collector wants to reset the `Connection` to the source, it must do so immediately after setting the checkpoint. This is because the Collection Framework sends batches of records, and then sets the checkpoint. The time the ckeckpoint is sent is the only time that the

collector knows that all records are flushed to Oracle Audit Vault Server. Note that there are other occasions that records are sent to the Oracle Audit Vault Server, but this is the only one with a checkpoint.

If the `Reader` instance cannot be obtained directly, then the collector must create and return a `Reader`.

Example 4-15 Creating Large Fields

```
public Reader getCommandText() throws AuditEventCollectorException {
   try {
      Clob clob = m_resultSet.getClob("SQL_TEXT");
      return clob.getCharacterStream();
   } catch (SQLException ex) {
      throw new AuditEventCollectorException(
            ErrorCodes.ERROR_GETTING_DATA_FROM_SOURCE, null, ex);
   }
}
```

4.5.12 Creating Markers to Uniquely Identify Records

See an example of how to create a marker, and review guidelines for how to create markers for Oracle Audit Vault and Database Firewall.

The collector must generate markers which the collection framework uses to uniquely identify a record.

The collector must generate a unique marker for each record in a particular trail. It can use more than one event field to create a marker. It can even use information not present in the audit event, such as, table name, file name, file creation time, and so on if it is not a template-based collection plug-in. Smaller sized markers are preferred because they take less time to create, less space in recovery phase, and less time to match. Markers are useful in certain scenarios, particularly in the *recovery phase*, where any records which were sent after the last checkpoint must be filtered. When the collection plug-in starts collecting events, it starts from the last checkpoint of the last run. However, some records might have been collected after that checkpoint and sent to the Audit Vault Server. To prevent duplication, the Collection Framework compares incoming records against existing records in the Audit Vault Server using the record markers.

In the following example, The strings `Session_ID` and `Entry_ID` are used to form a marker.

Example 4-16 Creating Markers

```
public String getMarker() throws AuditEventCollectorException {
      // ENTRY_ID will identify an audit event uniquely with in a
session. Hence
      // ENTRY_ID along with SESSION_ID will uniquely identify an
audit event
      // across sessions.
      try {
         return m_resultSet.getString("SESSION_ID") + ":"
               + m_resultSet.getString("ENTRY_ID");
      } catch (SQLException ex) {
          throw new AuditEventCollectorException(
```

```
                ErrorCodes.ERROR_GETTING_DATA_FROM_SOURCE, null, ex);
        }
    }
```

4.5.13 Closing the Java-Based Collection Plug-in

See how you can close a Java-based collection plug-in for Oracle Audit Vault and Database Firewall.

The process of closing a Java-based collection plug-in begins when the Collection Framework receives a command to stop collecting a particular audit trail, causing the Collection Framework to notify the collector using the `close()` method. In this method, the collector performs clean-up tasks, such as closing database connections or file handles. Once the `close()` method returns control to the Collection Framework, the collection thread ends.

This method should not result in any exceptions errors. In case of an exception, it should log an error message, as shown in the following example.

Example 4-17 Calling Close and Releasing Resources

```
public void close() {
      try {
         if (m_resultSet != null) {
            m_resultSet.close();
            m_resultSet = null;
         }
         if (m_connectionManager != null) {
            m_connectionManager.destroy();
            m_connectionManager = null;
         }
         m_previousCheckpoint = null;
         m_nextCheckpoint = null;
         m_logger = null;
      } catch (SQLException ex) {
         m_logger.logError("SampleEventCollector", "close",
               "SQLException occurred. ", ex);
      } catch (AuditException ex) {
         m_logger.logError("SampleEventCollector", "close",
               "AuditException occurred. ", ex);
      }
   }
```

4.5.14 Using Exceptions in Collection Plug-ins

An Oracle Audit Vault and Database Firewall collector can generate several different types of exceptions.

The collector can throw the following two checked exceptions: `AuditEventCollectorException` and `SetAttributeException`.

The `setAttribute` method can throw the `SetAttributeException` exception when the method cannot set a new attribute. Upon receiving this exception, it is possible that the

Collection Framework does not stop the collector (see an example in "Changing Audit Vault and Database Firewall Attributes at Run Time").

The rest of the methods except the `close` method throw `AuditEventCollectorException`. This exception must be thrown only if an unrecoverable condition has occurred. Upon receiving this exception, the Collection Framework stops the collector. Before stopping the collector, the Collection Framework calls `close` method. See the previous sections for sample code to create and throw these exceptions.

Related Topics

- Changing Oracle AVDF Attributes at Run Time
  If you are an administrator, then you can change the attributes that a Java-based collector plug-in uses during Oracle AVDF audit trail collection.

4.6 Java-Based Collection Plug-in Utility APIs

In addition to the Collection Framework, the Oracle Audit Vault API includes Java utility APIs that make the task of writing a collector easier.

4.6.1 About Connection to Database Sources Using ConnectionManager API

All of the Oracle Audit Vault and Database Firewall components (collectors, agents, and server) that are written in Java must use the `ConnectionManager` API to manage their connections to databases.

You use the `ConnectionManager` API to manage connections to source databases, such as Oracle Database, Microsoft SQL Server, Sybase Adaptive Server, and IBM DB2.

Benefits of Using the ConnectionManager API

- It reduces the resource usage on the database server.
- It makes client-side operations more graceful, so clients do not hang or die abruptly.
- It provides better performance.

You must instantiate a concrete implementation of the connection manager with the appropriate parameters required for setting up a connection pool. Several constructors are available for use. All optional parameters that are not supplied by the caller take default Oracle Audit Vault-specific values, as follows:

- `CONNECTION_FACTORY_CLASSNAME=oracle.jdbc.pool.OracleDataSource`
- `MIN_POOL_SIZE=0`
- `INACTIVE_CONNECTION_TIMEOUT=1800`
- `INITIAL_POOL_SIZE=0`
- `VALIDATE_CONNECTION_ON_BORROW=true`

4.6.2 Example of Using the ConnectionManager API to Connect to Database Sources

See how to use the ConnectionManager API to manage Oracle Audit Vault and Database Firewall Java compnent connections to databases.

The `ConnectionManager` API is based on the acquire, use, and release model for managing the database connections. All of the Oracle Audit Vault and Database Firewall components (collectors, agents, and server) that are written in Java must use the ConnectionManager API to manage their connections to databases.

The caller is expected to complete these steps in the order shown:

1. Create an instance of ConnectionManager API.
2. Get a connection to a database.
3. Use the connection
4. Release the connection back to the pool.
5. Repeat steps 2 through 4 as many times as needed.
6. Destroy the Connection Manager instance.

Example 4-18 Using the Connection Manager to Handle Connection Pooling

```
//Connection Manager
ConnectionManager cManager = null;

try {
  /*
   * Connection Pool Properties.
   * Set the pool properties such as URL
   * Initial pool size, Min pool size, etc.
   * The set of supported connection pool properties are
   * documented in the Oracle UCP documentation
   */
  Properties pProps = new Properties();
  pProps.put(URL, "jdbc:oracle:thin:@hostname:port:sid");

  /*
   *  Connection Properties
   *
   *  Set the connection properties here.
   *  The set of connection properties that can be set
   *  depends on the driver.  To enable SSL using the
   *  the oracle jdbc driver, you need to set the following
   *     Properties cProps = new Properties();
   *     String walletLoc =    "/path/to/walletdirectory/cwallet.sso";
   *     cProps.setProperty("oracle.net.authentication_services","(TCPS)");
   *     cProps.setProperty("javax.net.ssl.trustStore", walletLoc);
   *     cProps.setProperty("oracle.net.ssl_server_dn_match", "true")  ;
   *     cProps.setProperty("javax.net.ssl.trustStoreType","SSO");
   *     cProps.setProperty("javax.net.ssl.keyStore", walletLoc);
   *     cProps.setProperty("javax.net.ssl.keyStoreType","SSO");
   */
```

```
  Properties cProps = new Properties();

  cManager =  new ConnectionManagerImpl(pProps, cProps);

  String username;
  char[] passwd;
  Connection conn = null;

  /* Do something */
  ...

  /* Retrieve and set the username and password for user1 */
  username = "user1";
  passwd   = "user1passwd".toCharArray();

  /* Get a connection as "user1"*/
  conn = cManager.getConnection(username, passwd);


  /* Use the "user1" connection and do something useful */
  ...

  /* Release the connection */
  cManager.releaseConnection(conn);

  /* Retrieve and set the username and password for user2 */
  username = "user2";
  passwd   = "user2passwd".toCharArray();

  /* Get a connection as "user2" */
  conn = cManager.getConnection(username, passwd);


  /* Use the "user2" connection and do something useful */
  ...

  /* Release the connection */
  cManager.releaseConnection(conn);

} catch (Exception e) {
  /* Take appropriate action here */

} finally {
  if (cManager != null) {
    try {
      cManager.destroy();
    } catch (AuditException ae) {
      /* Take appropriate action here */
    }
}
```

The `ConnectionManager` API is designed so that a caller can acquire and release database connections using different user credentials at any point in time. For example, a caller can acquire a connection using `alice`'s database credentials, and

then later on acquire a connection with `robert`'s database credentials using the same connection manager.

> **Note:**
>
> Ensure that the caller does not do the following:
>
> - Keep a reference to the connection locally (through an instance or class variable).
> - Hold on to the connection for a long time.
>
> These requirements enable the connection pool to automatically recover connections that have the following behaviors:
>
> - They have exceeded the `TIME_TO_LIVE` time limit.
> - They have abandoned connections, that is, connections that have not been in use for a while.
> - There are connections that have been borrowed too many times. This requirement ensures that they to avoid resource leaks.

Related Topics

- Initializing the Java-Based Collection Plug-in
  Use these examples to understand how to initialize a Java-based collection plug-in, and how to start audit events collection with Oracle Audit Vault and Database Firewall.

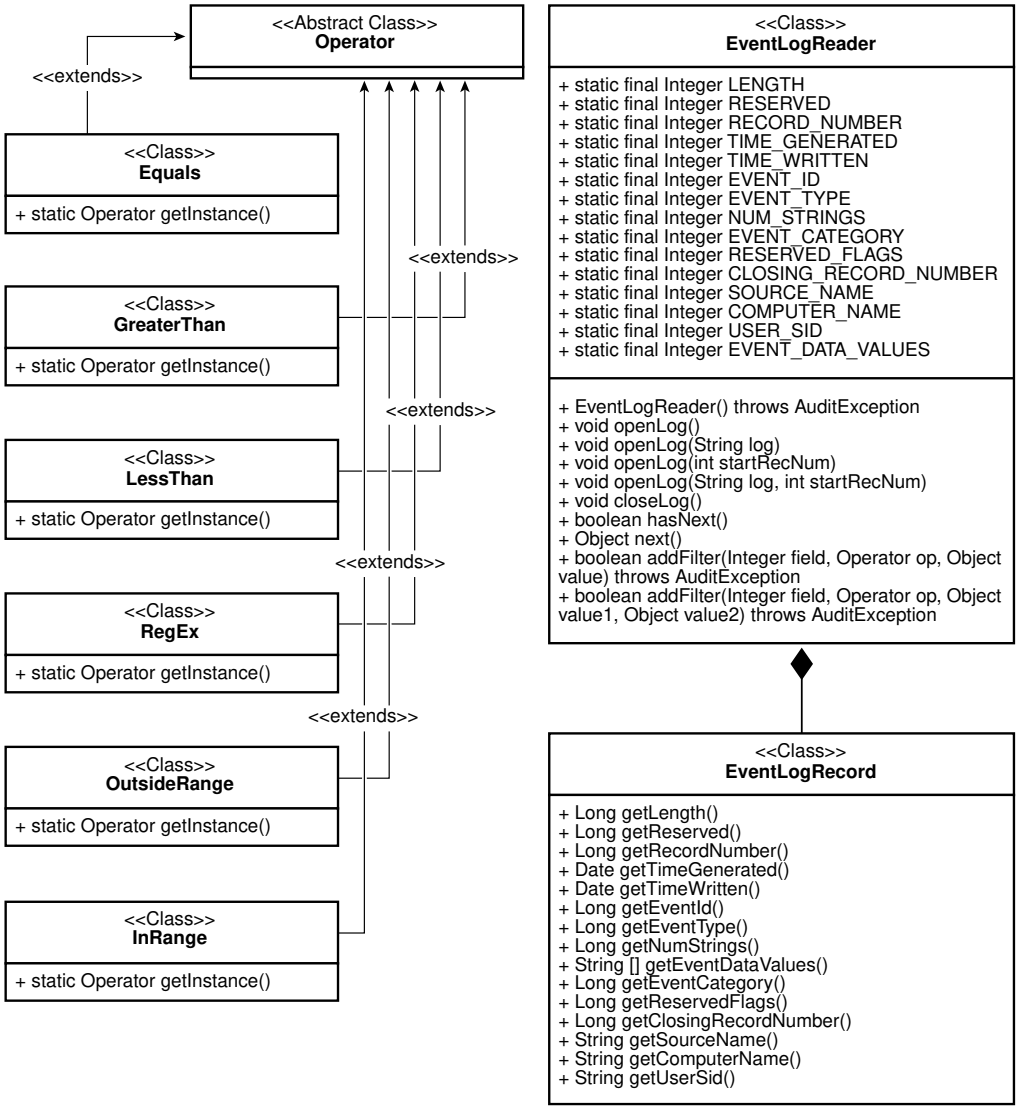
4.6.3 Using the Windows Event Log Access API

To parse Microsoft Windows event logs, you can use the Microsoft Windows `EventLog` API.

The Windows `EventLog` API is a wrapper on Windows APIs that access the Windows Event Log. This API is available only on the Windows platform, for collectors that need to extract audit records.

The following diagram shows the classes that you can use to parse the Windows event logs.

Figure 4-3 Structure of Windows Event Logs



The `EventLogRecord` class contains one record in the event log. The `EventLogReader` class helps to fetch event log records one by one. Operator classes help filter the event log records. An operator works on a particular field of event log record and determines whether the record is to be filtered based on the value of the field. For example, you can use the `Equals` operator to filter all event log records where the value of the field does not equal the value specified. The `InRange` and `OutsideRange` operators are ternary operators. The rest are binary operators.

To collect event log records, follow these steps:

1. Create the `EventLogReader` instance.

   For example, to open the application event log:

```
EventLogReader eventLogReader = new EventLogReader();
eventLogReader.openLog()
```

To open other event logs such security or system event logs, use the overloaded method `openLog(String log)`. For example, to open a security event log:

```
EventLogReader eventLogReader = new EventLogReader();
eventLogReader.openLog("Security");
```

To open an application event log from a specific record number, use the `openLog(int startRecNum)` overloaded method. For example, to open an application event log from audit record number `1234`:

```
EventLogReader eventLogReader = new EventLogReader();
eventLogReader.openLog(1234);
```

To open security or system event logs from a specific record number, use the overloaded method, `openLog(String log, int startRecNum)`. For example, to open a security event log from record number `4321`:

```
EventLogReader eventLogReader = new EventLogReader();
eventLogReader.openLog("Security",4321);
```

2. Add the appropriate filters.

   For example, to bind an equals filter to the `SourceName` field, so that the `EventLogReader` only receives records that have the source name `MSSQL$SQLEXPRESS`:

```
eventLogReader.addFilter(EventLogReader.SOURCE_NAME,
Equals.getInstance(), "MSSQL$SQLEXPRESS");
```

   To get event records between `Timestamp`, `m_lowerBoundTime`, and `m_upperBoundTime`, use following filters:

```
m_eventLogReader.addFilter(EventLogReader.TIME_GENERATED,
GreaterThan.getInstance(), m_lowerBoundTime);

m_eventLogReader.addFilter(EventLogReader.TIME_GENERATED,
LessThan.getInstance(), m_upperBoundTime);
```

3. Fetch and process the `EventLogRecord`.

   The following example code obtains the next `EventLogRecord`, and extracts various fields from it.

```
if(eventLogReader.hasNext()) {
              EventLogRecord record =
(EventLogRecord)eventLogReader.next();
              Long eventID = record.getEventId();
              String userID = record.getUserSid();
              String hostName = record.getComputerName();
  ...
   }
```

4. Close the `EventLogReader` instance.

When the collector is stopped, use the following code to close down the `EventLogReader` instance:

```
eventLogReader.closeLog();
```

Related Topics

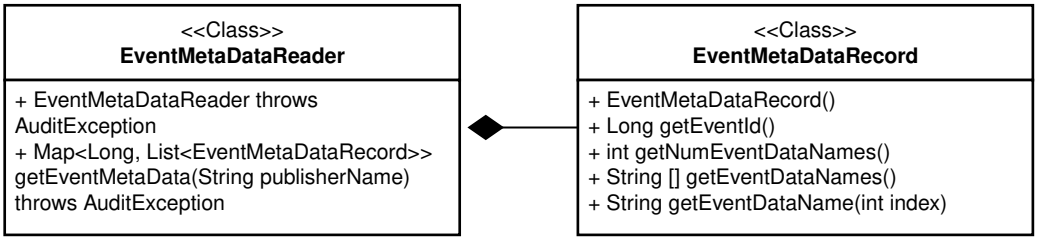
- Closing the Java-Based Collection Plug-in
  See how you can close a Java-based collection plug-in for Oracle Audit Vault and Database Firewall.

4.6.4 Using Windows EventMetaData API

To obtain metadata of events, you can use this Microsoft Windows Metadata Java API procedure.

Microsoft Windows provides a new API that can obtain metadata of events from version 2008 and on. Given a publisher name, this API obtains metadata for each event. The following figure illustrates how the Windows Metadata Java is a wrapper over the Windows API.

Figure 4-4 EventMetaData_Classes



The `EventMetaDataRecord` contains the metadata of one event. The `EventMetaDataReader` helps to fetch event metadata records one by one. Use this API as follows:

1. Create an instance of `EventMetaDataReader`.

   ```
   EventMetaDataReader eventMetaDataReader = new EventMetaDataReader();
   ```

2. Obtain metadata of all events for a publisher.

   ```
   Map<Long, List<EventMetaDataRecord>> eventLogRecordMap =
   eventMetaDataReader
      getEventMetaData("Microsoft-Windows-Security-Auditing");
   ```

3. Obtain the metadata list of a particular event from the map.

   ```
   List<EventMetaDataRecord> eventRecordList = m_eventRecordMap
      get(m_eventLogRecord.getEventId());
   ```

4. Obtain metadata from the list, and obtain event data names from it.

```
EventMetaDataRecord eventMetaDataRecord = eventRecordList.get(i);
for(int i=0; i<eventMetaDataRecord.getNumEventDataNames(); i++) {
   String eventDataName = eventMetaDataRecord.getEventDataName(i);
   ....
}
```

4.6.5 Using the AVLogger API to Log Messages

To log errors, warnings, informational, and debug messages into the Oracle Audit Vault and Database Firewall logs, you can use the `AVLogger` API.

Example 4-19 Using the AVLogger API

```
import oracle.av.platform.common.util.AVLogger;
import oracle.av.platform.common.exception.AuditException;
import oracle.av.platform.common.AuditErrorCodes;

public class Test
{
  public static void main(String[] args)  {
     /* Logger objects */
     AVLogger myModule = null;
     try {
       /* get Logger instances; this will auto-create the logger instance */
       /* if one does not exist */
       myModule = AVLogger.getLogger("someModule");
       /* print INFO level message */
       /* check log level if you are concatenating strings to avoid
expensive */
       /* string operations */
       if(myModule.isInfoEnabled()) {
           avServer.logInfo("Testing INFO level message...." + "another
String" +
             "one more string");
       }
       /* No need to check the log level if there is no string concatenation
*/
       myModule.logInfo("Testing INFO level message for another
component....");
       /* changing the log level dynamically */
       myModule.setLogLevel(AVLogger.AV_LOG_LEVEL_DEBUG);
       myModule.logWarn("Testing WARN level message ....");
       myModule.logDebug("Testing DEBUG level message ....");
       /* Reset the log level back to INFO */
       myModule.setLogLevel(AVLogger.AV_LOG_LEVEL_INFO);
       /* Testing Exceptions: For now on, all exceptions will have */
       /* an OAV-XXXX error code printed out automatically as long as */
       /* they derive from AuditException object */
       throw new AuditException (ErrorCodes.INTERNAL_ERROR, null, null);
     } catch (Exception e) {
       myModule.logError(e);
     }
```

```
    }
}
```

Related Topics

- Initializing the Java-Based Collection Plug-in
  Use these examples to understand how to initialize a Java-based collection plug-in, and how to start audit events collection with Oracle Audit Vault and Database Firewall.

4.6.6 Using the Oracle XML Developer's Kit to Parse XML Files

If you are developing collections, then you can use the Oracle XML Developer's Kit to parse XML files and extract audit records from them.

The Oracle XML Developer's Kit is included. and available to use to develop collections.

> **See Also:**
>
> *Oracle XML Developer's Kit Programmer's Guide* for detailed information.

4.7 Using an Audit Trail Cleanup with Java-Based Collection Plug-ins

Learn how you can enable audit trail clean-up on the source after Oracle Audit Vault and Database Firewall has archived an audit trail.

Audit trail clean-up is a feature that some sources provide to remove audit records after they have been archived. If this type of feature exists in the source, then an Oracle Audit Vault collection plug-in can integrate with the feature, to tell the source to what extent the audit trail has been archived. When Oracle Audit Vault provides archive status information, a source is enabled to clean up the audit trail (remove the original audit data) to that point, because the Oracle Audit Vault status indicates that the audit trail is archived, and deleting the audit trail to the point of the archive record results in no loss of data. The Oracle Audit Vault collection plug-in gives the clean-up utility information about the checkpoint, which is he point up to which data has been collected.

The collection plug-in can write archive status information into the directory *agent_home*`\av\atc`, to a file with a trail-specific name, using the following syntax *SecuredTargetName_TrailID*`.atc` (for example, `oracl_1.atc`).

The content of the `atc` file should consist of the following:

- `securedTargetType=Oracle`
- `SecuredTargetName=orcl`
- `TrailType=TABLE`
- `TrailName=sys.aud$`

- `2016-04-15 10:26:53.7` (This time stamp represents the last checkpoint for these settings.)

The target clean-up utility can parse the checkpoint from the `atc` file and purge audit records till this timestamp from audit trail.

For example, Oracle Database sources provide a target cleanup utility, in the `DMBS_AUDIT_MGMT` PL/SQL package. The Oracle Database prepackaged collection plug-ins integrate with the `DMBS_AUDIT_MGMT` package, which enables audit trail cleanup operations on the source.

4.8 Java-Based Collection Plug-in Security Considerations

Oracle strongly recommends that you review security guidelines before developing Java-based collection plug-ins.

For sources, such as databases, that require a connection in order to extract audit records, it is your responsibility, as the developer, to properly document the privileges needed to perform this task. Oracle recommends that the account used for connection have only the minimal privileges needed for the job. Any extra privileges can create a security issue.

You must also parse the input audit records properly, and protect Oracle Audit Vault and Database Firewall (Oracle AVDV) from malicious data. For instance, audit records can be crafted to inject SQL or HTML into the audit trail, which could expose data stored in Oracle AVDF to attacks. You must ensure that all incoming audit data is sanitized properly before it is given to the Collection Framework.

5
Packaging Audit Collection Plug-ins

Learn about the steps you need to perform to package collection plug-ins.

5.1 Flow of Packaging

Review the flow of packaging audit trails with Oracle Audit Vault and Database Firewall. The tools required for packaging the plug-in are available in the SDK.

Setting Up Your Development Environment described the directory structure of the staging area, all the shipping objects such as the JDBC driver (if needed), the mapper file, any executables, and any Oracle-supplied patches.

For Java collectors, it also includes appropriate locations for the compiled code and Java JAR files.

Audit Collection Plug-ins described the mapper file.

Java-Based Audit Trail Collection Plug-ins describes how to create Java-based collection plug-ins.

For Java collectors, it also includes Java code

Now you are in a position to create a `plugin-manifest.xml` file that describes where everything resides, what Audit Vault and Database Firewall should do with it, and then package everything into a `.zip` file to ship to the Audit Vault and Database Firewall Administrator.

5.2 Creating a plugin_manifest.xml File for Shipping

After you have created packaging for your audit trails, you are in a position to create a `plugin-manifest.xml` file for Oracle Audit Vault and Database Firewall.

The `plugin-manifest.xml` file describes where everything resides, indicates what Oracle Audit Vault and Database Firewall should do with it, and then how to package everything into a `.zip` file that can be shipped to the Oracle Audit Vault and Database Firewall Administrator.

1. When the collection plug-in program is ready to be packaged, create a directory structure.

   Note that the directory structure is slightly different for Java-based plug-ins than for collection plug-ins.

2. Create a `plugin-manifest.xml` file. This file describes the collection plug-in and the relevant parameters that provide the Audit Vault Collection Framework necessary information to instantiate and run the collection plug-in.

3. Package the collection plug-in files, the `plugin-manifest.xml` file, and any additional jars that collection plug-in depends on at run-time.

4. Run the `avpack` tool. The `avpack` tool validates and generates a `.zip` package that represents an collection plug-in package.

The `avpack` tool runs a number of validity checks (such as whether the directory structure is correctly populated, the manifest file is well-formed, and is without errors, and so on), then generates the collection plug-in package, in the form of a zip file, for deployment.

See Also:

- Audit Collection Plug-in Directory Structure
- Description of Plug-in Manifest File
- Example Code for example `plugin-manifest.xml` files specific to your type of collection plug-in.
- avpack Tool

5.3 External Dependencies

In the packaging process, **external dependencies** are files that are needed during runtime, but that can be unavailable when you package Oracle Audit Vault collection plug-ins.

An example of an external dependey is if your collection plug-in depends on a third-party component that the end-user licenses, or a component that has an issue related to licensing or copyright. In that scenario, it is possible that you are unable to package this component. If you cannot package the required component, then this is an external dependency. To resolve this dependency, you expect that the end-user provides the required compnent during collection plug-in deployment.

For these scenarios, the `plugin-manifest.xml` exposes the `unresolved-external` element. `avpack` does not file-check files under this element, but during deployment time, `avpack` will fail to deploy the collection plug-in if the `$OH/av/dropins` folder does not contain these files.

In the following example, `foo.jar` is an external dependency:

```
<unresolved-external>
   <file>foo.jar</file>
</unresolved-external>
```

During deployment, `avpack` checks to see if the file `foo.jar` is present in the `$OH/av/dropins` folder on the Oracle Audit Vault Server. If the file is missing, then `avpack` fails to deploy the collection plug-in. Instead, it issues a message stating that external dependencies are not being met.

To resolve the issue, the user must acquire the file, and make it available in the `$OH/av/dropins` folder. After the external dependency is provided, `avpack` can deploy the collection plug-in successfully.

5.4 Creating New Versions of Your Audit Collection Plug-ins

If you create new versions of the collection plug-ins, then you can easily plug them in to replace existing versions without difficulty in Oracle Audit Vault and Database Firewall.

To update an existing collection plug-in to a newer version, use the `avcli` command-line tool with the `DEPLOY PLUGIN` command,

Collection plug-ins are cumulative in nature. All necessary files are created and updated.

To remove or undeploy collection plug-ins, use the `avcli` tool and the `UNDEPLOY PLUGIN` commands. These commands are atomic; that is, they are all or nothing transaction, which helps maintain a high degree of system stability.

Related Topics

- *Oracle Audit Vault and Database Firewall Administrator's Guide*
- *Oracle Audit Vault and Database Firewall Administrator's Guide*

5.5 avpack Tool

The `avpack` tool is a command-line based tool written in Java that packages the various collection plug-in objects such as code files, configuration files, and so on.

Prerequisites

You must complete the following prerequsites for using the avtool:

- Install and package (run) the `avpack` plug-in tool on the same platform on which the agent will run.

  The packaged `avpack` plug-in for Linux can be used for all platforms, but the packaged `avpack` plug-in for Windows can only be used for Windows platforms.
- Place collection plug-in artifacts following the recommended directory structure. Then, you can use `avpack` to generate a collection plug-in package.

File Path

You can stage the collection plug-in files in any directory that is accessible by the `avpack` tool. The `avpack` tool validates the directory structure, and then parses and verifies the `plugin-manifest.xml` file. The tool also performs some basic verifications, such as verifying that all the files specified in the `plugin-manifest.xml` are staged in their corresponding directories, and so on.

Syntax

```
avpack -stagedir directory name -o archive filename [-l loglevel ]
[-h]
```

Options

Each option must be prefixed with a minus sign (-).

Option	Description
`-stagedir`	The directory under which the collection plug-in artifacts are staged. Contents of this directory will be archived in the generated plug-in archive.

Option	Description
`-o`	The name for the generated plug-in archive file. It should end with a `.zip` extension. (for example, `myplugin.zip`).
`-l`	(Optional) Sets the log level to the level specified. Supported log levels: `INFO`, `WARNING`, `ERROR`, and `DEBUG`. Default log level is `INFO`.
`-h`	(Optional) Display help for the `avpack` tool.

Usage Notes

You use the `plugin-manifest.xml` file to specify the key files that the collection plug-in must have to run. The `avpack` utility checks for the existence of these files, but zips everything contained in `stagedir`, so you do not need to list every file unless you want it to be verified by `avpack`.

After validation is complete, the tool packages the files into a `.zip` plug-in package suitable for deployment with Oracle Audit Vault and Database Firewall.

Related Topics

- Audit Collection Plug-in Directory Structure
  Learn about the Oracle Audit Vault collection plug-in directory structure, the development environment, and how to stage plug-in `manifest.xml` files.

6
Testing Audit Collection Plug-ins

Find out about the testing you can do for your collection plug-ins.

Be sure to analyze your database and audit trails for other issues that require testing.

6.1 Requirements for Testing Audit Collection Plug-ins

To prepare for testing, deploy the Audit Vault Server and an Audit Vault Agent, and check your systems and audit trails.

You should prepare for testing by performing the following:

- Deploy the Audit Vault Server and an Audit Vault Agent.
- Have an available source system, a system that generates the audit events.
- Ensure that the agent is deployed on the same computer where the audit trail resides if the audit trail must be collected locally (for example, if it is written to operating system files).

Related Topics

- Before Setting Up the Development Environment
  To develop audit collection plug-ins, you must first set up the development environment. This set up provides a consistent environment for developing and testing the collection plug-ins.

6.2 Typical Audit Collection Plug-in Testing Processes

A typical audit collection testing process for collections plug-ins should look like this.

When you are testing procedures, your sequence of tasks should be similar to the following:

1. Perform functional testing:
   a. Deploy the collection plug-in in the generated `.zip` archive that you created earlier in your test Oracle Audit Vault Server environment.
   b. Redeploy the agent (containing the updated plug-in artifacts) into your test Oracle Audit Vault agent environment.
   c. Register the source using the `AVCLI` utility.
   d. Issue an `AVCLI START COLLECTION` command to start gathering records from the audit trail supported by this collection plug-in.
   e. Validate the process, by looking at the data reports through the AVDF Console, to ensure that:

      - Records in the source are now in the Oracle Audit Vault Server.

      - The data makes sense.

      - Fields are correctly mapped.

- Values are valid.

f. Issue an `AVCLI STOP COLLECTION` command.

g. Undeploy the collection plug-in..

h. Redeploy the agent as described in Step 1b.

2. Perform failure testing to see what happens when various things go wrong.

   Some examples of failure are network failure, a source shutting down in the middle of collection, a power outage, and malformed input data. In all cases, the collection plug-in should not crash, and should be able to recover gracefully, continuing collection from where it left off. The guarantee you need to provide is that each audit record is sent to the Audit Vault Server once, and exactly once, regardless of failure.

3. Analyze performance by checking how many of these components the collection plug-in uses:

   - The CPU
   - The memory
   - The disk I/O
   - The network I/O

4. Check the performance under stress.

   Some examples of stress are thirty days of continuous use, heavy event volume, or collection of trails for multiple sources at the same time, both on the same host, and on multiple hosts.

5. Perform security testing (for example, see if you can inject HTML or SQL).

6. Perform internationalization testing. Test the ability to handle data in multiple input languages.

7. If bugs are found, fix them and then repeat these steps.

> **See Also:**
>
> - *Oracle Audit Vault and Database Firewall Administrator's Guide* to register the source
> - *Oracle Audit Vault and Database Firewall Administrator's Guide* to start gathering records from the audit trail supported by this collection plug-in
> - *Oracle Audit Vault and Database Firewall Administrator's Guide* to undeploy the collection plug-in

Related Topics

- STOP COLLECTION FOR TARGET
- Packaging Audit Collection Plug-ins
  Learn about the steps you need to perform to package collection plug-ins.

- Redeploying the Oracle Audit Vault Agent
  While testing the collection plug-in, it can be necessary to redeploy the agent for various reasons.

6.3 Deploying an Audit Vault Agent

See how you register an Agent, create an Agent home directory, install the Agent, and start the Agent.

This Agent can be on the same computer as the Audit Vault Server, or on a different one.

1. Register the Agent host using the `AVCLI` command `REGISTER HOST`.
2. Create a directory (`$AGENT_HOME`) on the Agent host.
3. Copy the `agent.jar` from the Audit Vault Server `$ORACLE_HOME/av/jlib/agent.jar` to the `$AGENT_HOME`.
4. Install the Agent using following command:

   ```
   $ java -jar agent.jar -d $AGENT_HOME
   ```

5. Start the Agent using `-key` option. When prompted, enter the activation key that was provided by the Oracle AVDF administrator. As you type it, this key is not displayed.

   ```
   $ $AGENT_HOME/bin/agentctl start -key
   Enter activation key:
   ```

   Subsequently, starting the Agent does not require the user to provide the activation key. The Agent can be started using the following command:

   ```
   $ $AGENT_HOME/bin/agentctl start
   ```

It can take several seconds before the Agent comes to a complete stop, and the Agent process is shut down.

Activation is a one time activity. You do not have to do it again.

You can stop the Agent at any time by using the following command:

```
$ $AGENT_HOME/bin/agentctl stop
```

Related Topics

- REGISTER HOST
- About Registering Hosts

6.4 Redeploying the Oracle Audit Vault Agent

While testing the collection plug-in, it can be necessary to redeploy the agent for various reasons.

Before you redeploy an agent, .the agent must be already set up, and a directory created.

1. Copy the `agent.jar` from the Audit Vault Server to a local directory.

2. Update the agent by using the following command:

```
$ java -jar agent.jar -d $AGENT_HOME
```

3. Start the agent with the `$AGENT_HOME/bin/agentctl` start command.

> **Note:**
>
> The agent automatically determines if it is an upgrade or a new install depending on the destination directory provided to the `java -jar agent.jar` command.

A

Audit Vault Server Fields

You can map Oracle Audit Vault and Database Firewall events and fields in your collection plug-ins.

A.1 Oracle Audit Vault and Database Firewall Fields

Oracle Audit Vault and Database Firewall values consist of core fields, large fields, marker fields, and extension fields.

A.1.1 Core Fields

To monitor and filter audit records for all source types in Oracle Audit Vault, you use **core fields**.

Core fields are fundamental to all source types. They are central to the description of an event. These fields are present in most audit records, for reporting, filtering, and so on.

Core Field Definitions

EventTimeUTC: Required: The time stamp that indicates when the event occurred. If the event has more than one time stamp (for example, an event start time stamp and an event end time stamp), then the **collection plug-in** must assign a time stamp to this field. If this field contains `NULL`, then Oracle Audit Vault shuts down the collection plug-in.

UserName: Required: The user who performed the action in the application or system that generated the audit record. If this field contains `NULL`, then the audit record is invalid.

CommandClass: Required: The action performed in the event (for example, `SELECT` or `DELETE`). If this field contains `NULL`, then the audit record is invalid.

OSUserName: The user who logged into the operating system that generated the audit record. If the user logged into the operating system as `JOHN` but performed the action as `SCOTT`, then this field contains `JOHN` and the User Name field contains `SCOTT`.

TargetType: The type of the target object on which the action was performed. For example, if the user selected from a table, then the target type is `TABLE`.

TargetObject: The name of the object on which the action was performed. For example, if the user selected from a table, then the Target Object field contains the name of the table.

TargetOwner: The name of the owner of the target on which the action was performed. For example, if the user had selected from a table owned by user `JOHN`, then the Target Owner field contains the user name `JOHN`.

ClientIP: The IP address of the host (Host Name) from where the user initiated the action.

ClientId: Client identifier of the user whose actions were audited.

ClientHostName: The host computer from where the user initiated the action. For example, if the user performed the action from an application on a server, then this field contains the name of the server.

TerminalName: Name of the UNIX terminal that was the source of the event.

EventName: The name of the event as is from the audit trail.

EventStatus: The status of the event. There are three possible values for `EventStatus: SUCCESS`, `FAILURE`, and `UNKNOWN`.

ErrorId: The error code of an action.

ErrorMessage: The error message of an action.

Related Topics

- Actions
  The **Action** field describes the nature of user activity that triggers generation of an audit record. It is similar to the verb part of a sentence, it describes the activity.
- Target Types
  The `TargetType` field describes the type of object on which a user action operates. It is similar to a noun that describes the object of a user action.

A.1.2 Large Fields

In Oracle Audit Vault, **Large fields** are fields that can contain arbitrarily large amounts of data.

Large Field Definitions

For large fields, use the following:

- **CommandText:** Contains the text of the command that caused the event, which can be a SQL statement, a PL/SQL statement, and so on. This is also a core field.
- **CommandParam:** Contains the parameters of the command that caused the event. This is also a core field.

A.1.3 Marker Field

In Oracle Audit Vault, **marker fields** are fields that uniquely identify a record in a trail.

Marker Field Definitions

Marker Field of a Record: The marker is a string that uniquely identifies a record in a trail. During the recovery process, Oracle Audit Vault uses this field to filter the duplicate records. The collection plug-in provides the marker field, which is typically a concatenated subset of the fields of an audit record. For example, with Oracle Database, the session ID and Entry ID (a unique identifier within a session) define a marker.

A.1.4 Extension Field

Extension fields store fields that cannot be accommodated in core or large fields, as name-value pairs, separated by delimiter, inside a single Audit Vault field.

Extension Field Definition

The extension field contains character large object (CLOB) columns. The `RLS$INFO` column describes the configured row level security policies. The RLS$INFO information is mapped to the extension field in Oracle Audit Vault and Database Firewall.

Extension Field Values

To populate the extension field column, you must set the *AUDIT_TRAIL* parameter of the target to `DB EXTENDED`.

A.2 Actions and Target Types

When you build collection plug-ins, you can use the target types and actions that Oracle Audit Vault can detect.

If you are building a collection plug-in, then you should use these fields in your mapper file, if the fields mapped semantically. Otherwise, you can use your own values.

A.2.1 Actions

The **Action** field describes the nature of user activity that triggers generation of an audit record. It is similar to the verb part of a sentence, it describes the activity.

Purpose

Describes the nature of user activity that triggers generation of an audit record.

Oracle Audit Vault and Database Firewall strongly recommends mapping audit events to an appropriate value for the **Action** field, if the user activity semantically maps to it.

Permitted Actions

Audit Vault Server is currently aware of the following actions:

```
END
ACCESS
ACQUIRE
ALTER
ANALYZE
APPLY
ARCHIVE
ASSIGN
ASSOCIATE
AUDIT
AUTHENTICATE
AUTHORIZE
BACKUP
```

```
BIND
BLOCK
CACHE
CALCULATE
CALL
CANCEL
CLOSE
COMMIT
COMMUNICATE
COMPARE
CONFIGURE
CONNECT
CONTROL
CONVERT
COPY
CREATE
DDL
DEADLOCK
DELETE
DEMOTE
DENY
DENY
DISABLE
DISASSOCIATE
DISCONNECT
DML
DROP
ENABLE
EXCEED
EXECUTE
EXPIRE
EXPORT
FAIL
FILTER
FINISH
GET
GRANT
IMPORT
INHERIT
INITIALIZE
INSERT
INSTALL
INVALID
INVALIDATE
LOAD
LOCK
LOGIN
LOGOUT
MIGRATE
MOUNT
MOVE
NOAUDIT
NOTIFY
NOTIFY
OPEN
```

```
PAUSE
PROMOTE
PROXY
PUBLISH
QUARANTINE
RAISE
READ
RECEIVE
RECOVER
REDO
REFRESH
REGISTER
RELEASE
REMOTE CALL
RENAME
RENEW
REQUEST
RESET
RESTORE
RESUME
RETRIEVE
REVOKE
ROLLBACK
ROLLFORWARD
SAVEPOINT
SEARCH
SELECT
SEND
SET
START
STOP
SUBMIT
SUBSCRIBE
SUSPEND
SYNCHRONIZE
TRANSACTION MANAGEMENT
TRUNCATE
UNDO
UNINSTALL
UNKNOWN
UNLOCK
UNMOUNT
UNREGISTER
UNSUBSCRIBE
UPDATE
VALIDATE
VIOLATE
WAIT
WRITE
```

A.2.2 Target Types

The `TargetType` field describes the type of object on which a user action operates. It is similar to a noun that describes the object of a user action.

Purpose

Describes the type of object on which a user action operates.

Oracle Audit Vault and Database Firewall strongly recommends mapping audit events to an appropriate value for the `TargetType` field, if the user activity semantically maps to it.

Permitted Objects

Oracle Audit Vault Server is currently aware of the following target types:

```
ALL TRIGGERS
APP ROLE
APPLICATION
ASSEMBLY
AUTHORIZATION
BROKER QUEING
BUFFERPOOL
CHECKPOINT
CLUSTER
CONNECTION
CONTEXT
CONTROL FILE
DATABASE
DATABASE LINK
DBA_RECYCLEBIN
DEFAULT
DIMENSION
DIRECTORY
EDITION
EVALUATION
EVENT MONITOR
EXPRESSION
FLASHBACK
FLASHBACK ARCHIVE
FUNCTION
INDEX
INDEXES
INDEXTYPE
INSTANCE
JAVA
LIBRARY
MATERIALIZED VIEW
MATERIALIZED VIEW LOG
MESSAGE
METHOD
MINING MODEL
NODE
```

```
NODEGROUP
OBJECT
OPERATOR
OUTLINE
PACKAGE
PACKAGE BODY
PRIVILEGE
PROCEDURE
PROFILE
PUBLIC DATABASE LINK
PUBLIC SYNONYM
RESOURCE COST
RESTORE POINT
REVOKE
REWRITE EQUIVALENCE
ROLE
ROLLBACK SEG
RULE
SAVEPOINT
SAVEPOINT
SCHEMA
SEQUENCE
SESSION
STATISTICS
SUBSCRIPTION
SUMMARY
SYNONYM
SYSTEM
TABLE
TABLE OR SCHEMA POLICY
TABLESPACE
TAPE
TRACE
TRANSACTION
TRIGGER
TYPE
TYPE BODY
UNKNOWN
USER
USER LOGON
USER OR PROGRAM UNIT LABEL
USER_RECYCLEBIN
VIEW
```

B
Schemas

Oracle AVDF uses these schemas for plug-in manifest files and collection plug-ins.

B.1 Sample Schema for a plugin-manifest.xml file

To validate any `plugin-manifest.xml` file that you author, Oracle recommends that you use the sample schema for a `plugin-manifest.xml file`.

Example B-1 Sample plugin-manifest.xsd file

```
<?xml version="1.0" encoding="utf-8"?>
<!-- This schema defines the structure of the Oracle Audit Vault Plugin -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://foobar.example.com/av/plugin"
           targetNamespace="http://foobar.example.com/av/plugin"
           elementFormDefault="qualified">

  <xs:element name="plugin">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="targetVersion">
          <xs:complexType>
            <xs:attribute name="min" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
        <xs:element name="extensionSet">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="extensionPoint">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="fileList">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="jars" minOccurs="0" maxOccurs="1">
                            <xs:complexType>
                              <xs:sequence>
                                <xs:element minOccurs="0" maxOccurs="unbounded" name="include">
                                  <xs:complexType>
                                    <xs:attribute name="file" type="xs:string" use="required" />
                                    <xs:attribute name="permission" type="xs:string" use="optional" />
                                  </xs:complexType>
                                </xs:element>
                              </xs:sequence>
                            </xs:complexType>
                          </xs:element>
                          <xs:element name="templates" minOccurs="0" maxOccurs="1">
                            <xs:complexType>
                              <xs:sequence>
                                <xs:element minOccurs="0" maxOccurs="unbounded" name="include">
                                  <xs:complexType>
                                    <xs:attribute name="file" type="xs:string" use="required" />
                                    <xs:attribute name="permission" type="xs:string" use="optional" />
```

```
                        </xs:complexType>
                      </xs:element>
                    </xs:sequence>
                  </xs:complexType>
                </xs:element>
                <xs:element name="bin" minOccurs="0" maxOccurs="1">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element minOccurs="0" maxOccurs="unbounded" name="include">
                        <xs:complexType>
                          <xs:attribute name="file" type="xs:string" use="required" />
                          <xs:attribute name="permission" type="xs:string"
use="optional" />
                        </xs:complexType>
                      </xs:element>
                    </xs:sequence>
                  </xs:complexType>
                </xs:element>
                <xs:element name="config" minOccurs="0" maxOccurs="1">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element minOccurs="0" maxOccurs="unbounded" name="include">
                        <xs:complexType>
                          <xs:attribute name="file" type="xs:string" use="required" />
                          <xs:attribute name="permission" type="xs:string"
use="optional" />
                        </xs:complexType>
                      </xs:element>
                    </xs:sequence>
                  </xs:complexType>
                </xs:element>
                <xs:element name="shell" minOccurs="0" maxOccurs="1">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element minOccurs="0" maxOccurs="unbounded" name="include">
                        <xs:complexType>
                          <xs:attribute name="file" type="xs:string" use="required" />
                          <xs:attribute name="permission" type="xs:string"
use="optional" />
                        </xs:complexType>
                      </xs:element>
                    </xs:sequence>
                  </xs:complexType>
                </xs:element>
                <xs:element name="patch" minOccurs="0" maxOccurs="1">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element minOccurs="0" maxOccurs="unbounded" name="include">
                        <xs:complexType>
                          <xs:attribute name="file" type="xs:string" use="required" />
                          <xs:attribute name="permission" type="xs:string"
use="optional" />
                        </xs:complexType>
                      </xs:element>
                    </xs:sequence>
                  </xs:complexType>
                </xs:element>
                <xs:element name="unresolved-external" minOccurs="0" maxOccurs="1">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element minOccurs="0" maxOccurs="unbounded" name="include">
```

```
                              <xs:complexType>
                                <xs:attribute name="file" type="xs:string" use="required" />
                                <xs:attribute name="permission" type="xs:string" use="optional" />
                              </xs:complexType>
                            </xs:element>
                          </xs:sequence>
                        </xs:complexType>
                      </xs:element>
                    </xs:sequence>
                  </xs:complexType>
                </xs:element>
                <xs:element name="securedTargetTypeInfo">
                  <xs:complexType>
                    <xs:attribute name="name" type="xs:string" use="required" />
                  </xs:complexType>
                </xs:element>
                <xs:element minOccurs="0" maxOccurs="unbounded" name="trailInfo">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="trailType" type="xs:string" />
                      <xs:element minOccurs="0" name="trailLocation" type="xs:string" />
                      <xs:element maxOccurs="unbounded" name="className">
                        <xs:complexType>
                          <xs:attribute name="name" type="xs:string" use="required" />
                          <xs:attribute name="securedTargetVersion" type="xs:decimal"
use="optional" />
                        </xs:complexType>
                      </xs:element>
                    </xs:sequence>
                  </xs:complexType>
                </xs:element>
                <xs:element minOccurs="0" maxOccurs="unbounded" name="eventPatch">
                  <xs:complexType>
                    <xs:attribute name="name" type="xs:string" use="required" />
                    <xs:attribute name="order" type="xs:unsignedByte" use="required" />
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
              <xs:attribute name="type" type="xs:string" use="required" />
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" use="required" />
  <xs:attribute name="id" type="xs:string" use="required" />
  <xs:attribute name="version" type="xs:string" use="required" />
  <xs:attribute name="provider-name" type="xs:string" use="required" />
  <xs:attribute name="copyright" type="xs:string" use="optional" />
 </xs:complexType>
</xs:element>
</xs:schema>
```

Related Topics

- Example Code
  Learn from examples the different types of collection plug-ins, including database tables, XML files, and Java-based file collection plug-ins.

- Staging a plugin-manifest.xml File
  You must stage the plugin-manifest.xml file directly under the `STAGE_DIR_ROOT` directory.

B.2 Database Table Collection Plug-in Mapper File

See an example of an Oracle Audit Vault and Database Firewall database table collection plug-in mapper file.

To see how to create your own plug-in mapper file, review the structure of this example.

Example B-2 Database Table Collection Plug-in Mapper Schema

```
<?xml version="1.0"?>

<!--
 Copyright (c) 2013, 2015, Oracle and/or its affiliates. All rights reserved.
-->

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:tnp="http://
foobar.example.com/avdf/ezcollector/namepattern" elementFormDefault="qualified">
<xsd:include schemaLocation="ezCollectorTemplate_schema.xsd"/>
<xsd:import schemaLocation="NamePattern_schema.xsd" namespace="http://
foobar.example.com/avdf/ezcollector/namepattern" />

<!-- XML Document Structure-->
<xsd:element name="AVTableCollectorTemplate" >
    <xsd:complexType>
        <xsd:all>
            <!-- Audit table name -->
            <xsd:element name="TableName" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
            <xsd:element name="NamePattern" type="tnp:NamePatternType" minOccurs="0"
maxOccurs="1"/>
            <!-- Database connection information -->
            <xsd:element name="ConnectionInfo" minOccurs="1" maxOccurs="1">
                <xsd:complexType>
                    <xsd:sequence>
                        <!-- JDBC datasource class -->
                        <xsd:element name="DataSource" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
                     </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <!-- Secured Target to AV server fields Mapping for Core, Large,
Extension fields and Marker-->
            <xsd:element name="FieldMappingInfo" type="FieldMappingInfoType"
minOccurs="1" maxOccurs="1"/>
            <!-- Event Filter. This is optional. If it is not used, all the audit
events will be collected-->
            <xsd:element name="EventFilter" type="EventFilterType" minOccurs="0"
maxOccurs="1"/>
        </xsd:all>
        <!-- Secured Target Type-->
```

```
        <xsd:attribute name="securedTargetType" type="xsd:string" use="required"/>
        <!-- Max Secured Target version supported by the template-->
        <xsd:attribute name="maxSecuredTargetVersion" type="xsd:string" use="required"/>
        <!-- Min Secured Target version supported by the template-->
        <xsd:attribute name="minSecuredTargetVersion" type="xsd:string"/>
        <!-- Template file version-->
        <xsd:attribute name="version" type="xsd:string" use="required"/>
    </xsd:complexType>
</xsd:element>
</xsd:schema>
```

B.3 Schema For XML File Collection Plug-in Mapper File

See how to set up a schema for an XML file collection plug-in mapper file for Oracle Audit Vault and Database Firewall.

In the following example, you can see how to set up a mapper schema:

Example B-3 XML file collection plug-in Mapper Schema

```
<?xml version="1.0"?>

<!--
 Copyright (c) 2013, 2019, Oracle and/or its affiliates. All rights reserved.
-->

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:include schemaLocation="ezCollectorTemplate_schema.xsd"/>

<!-- XML Document Structure-->
<xsd:element name="AVXMLCollectorTemplate" >
    <xsd:complexType>
        <xsd:all>
            <xsd:element name="HeaderInfo"  minOccurs="1" maxOccurs="1">
                <xsd:complexType>
                    <xsd:all>
                        <!-- StartTag tag contains Root element of XML Audit data file-->
                        <xsd:element name="StartTag" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                    </xsd:all>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="RecordInfo"  minOccurs="1" maxOccurs="1">
                <xsd:complexType>
                    <xsd:all>
                        <!-- start tag of xml audit record in XML audit file-->
                        <xsd:element name="StartTag" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                    </xsd:all>
                </xsd:complexType>
            </xsd:element>
            <!-- tag for specifying xsl transformation related information -->
            <xsd:element name="XslTransformation" minOccurs="0" maxOccurs="1">
```

```
                <xsd:complexType>
                                        <xsd:all>
                                                <!-- tag to specify name of XSL file--
>
                                                <xsd:element name="XslFile"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
                                                <!-- tag for specifying Root element
of source XML Audit data file to be transformed-->
                                                <xsd:element
name="SourceFileStartTag" type="xsd:string" minOccurs="1" maxOccurs="1"/>
                                        </xsd:all>
                                </xsd:complexType>
            </xsd:element>
            <!-- Secured Target to AV server fields Mapping for Core, Large,
Extension fields and Marker-->
            <xsd:element name="FieldMappingInfo" type="FieldMappingInfoType"
minOccurs="1" maxOccurs="1"/>
            <!-- Event Filter. This is optional. If it is not used, all the audit
events will be collected-->
            <xsd:element name="EventFilter" type="EventFilterType" minOccurs="0"
maxOccurs="1"/>
        </xsd:all>
        <!-- Secured Target Type-->
        <xsd:attribute name="securedTargetType" type="xsd:string" use="required"/>
        <!-- Max Secured Target version supported by the template-->
        <xsd:attribute name="maxSecuredTargetVersion" type="xsd:string"
use="required"/>
        <!-- Min Secured Target version supported by the template-->
        <xsd:attribute name="minSecuredTargetVersion" type="xsd:string"/>
        <!-- Template file version-->
        <xsd:attribute name="version" type="xsd:string" use="required"/>
    </xsd:complexType>
</xsd:element>
</xsd:schema>
```

B.4 Schema For JSON File Collection Plug-in Mapper File

See how to set up a schema for a JSON file collection plug-in mapper file for Oracle Audit Vault and Database Firewall.

In the following example, you can see how to set up a mapper schema:

Example B-4 JSON file collection plug-in Mapper Schema

```
<?xml version="1.0"?>

<!--
 Copyright (c) 2013, 2017, Oracle and/or its affiliates. All rights reserved.

-->

<xsd:schema xmlns:xsd="http://foobar.example.org/2001/XMLSchema">
<xsd:include schemaLocation="ezCollectorTemplate_schema.xsd"/>
```

```
<!-- XML Document Structure-->
<xsd:element name="AVJSONCollectorTemplate" >
    <xsd:complexType>
        <xsd:all>
            <xsd:element name="HeaderInfo"  minOccurs="0" maxOccurs="1">
                <xsd:complexType>
                    <xsd:all>
                        <!-- StartTag tag contains Root element of XML Audit data file-->
                        <xsd:element name="StartTag" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
                    </xsd:all>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="RecordInfo"  minOccurs="1" maxOccurs="1">
                <xsd:complexType>
                    <xsd:all>
                        <!-- start tag of xml audit record in XML audit file-->
                        <xsd:element name="StartTag" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                    </xsd:all>
                </xsd:complexType>
            </xsd:element>
            <!-- Secured Target to AV server fields Mapping for Core, Large, Extension
fields and Marker-->
            <xsd:element name="FieldMappingInfo" type="FieldMappingInfoType" minOccurs="1"
maxOccurs="1"/>
            <!-- Event Filter. This is optional. If it is not used, all the audit events
will be collected-->
            <xsd:element name="EventFilter" type="EventFilterType" minOccurs="0"
maxOccurs="1"/>
        </xsd:all>
        <!-- Secured Target Type-->
        <xsd:attribute name="securedTargetType" type="xsd:string" use="required"/>
        <!-- Max Secured Target version supported by the template-->
        <xsd:attribute name="maxSecuredTargetVersion" type="xsd:string" use="required"/>
        <!-- Min Secured Target version supported by the template-->
        <xsd:attribute name="minSecuredTargetVersion" type="xsd:string"/>
        <!-- Template file version-->
        <xsd:attribute name="version" type="xsd:string" use="required"/>
    </xsd:complexType>
</xsd:element>
</xsd:schema>
```

B.5 Schema For CSV File Collection Plug-in Mapper File

See how to set up a schema for a CSV file collection plug-in mapper file for Oracle Audit Vault and Database Firewall.

In the following example, you can see how to set up a mapper schema:

Example B-5 CSV file collection plug-in Mapper Schema

```
<?xml version="1.0"?>

<!--
 Copyright (c) 2013, 2017, Oracle and/or its affiliates. All rights reserved.

-->

<xsd:schema xmlns:xsd="http://foobar.example.org/2001/XMLSchema">
<xsd:include schemaLocation="ezCollectorTemplate_schema.xsd"/>

<!-- XML Document Structure-->
<xsd:element name="AVCSVCollectorTemplate" >
    <xsd:complexType>
        <xsd:all>
            <xsd:element name="HeaderInfo"  minOccurs="0" maxOccurs="1">
                <xsd:complexType>
                    <xsd:all>
                        <!-- StartTag tag contains Root element of XML Audit data
file-->
                        <xsd:element name="StartTag" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
                    </xsd:all>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="RecordInfo"  minOccurs="1" maxOccurs="1">
                <xsd:complexType>
                    <xsd:all>
                        <!-- start tag of xml audit record in XML audit file-->
                        <xsd:element name="StartTag" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                    </xsd:all>
                </xsd:complexType>
            </xsd:element>
            <!-- Secured Target to AV server fields Mapping for Core, Large,
Extension fields and Marker-->
            <xsd:element name="FieldMappingInfo" type="FieldMappingInfoType"
minOccurs="1" maxOccurs="1"/>
            <!-- Event Filter. This is optional. If it is not used, all the audit
events will be collected-->
            <xsd:element name="EventFilter" type="EventFilterType" minOccurs="0"
maxOccurs="1"/>
        </xsd:all>
        <!-- Secured Target Type-->
        <xsd:attribute name="securedTargetType" type="xsd:string" use="required"/>
        <!-- Max Secured Target version supported by the template-->
        <xsd:attribute name="maxSecuredTargetVersion" type="xsd:string"
use="required"/>
        <!-- Min Secured Target version supported by the template-->
        <xsd:attribute name="minSecuredTargetVersion" type="xsd:string"/>
        <!-- Template file version-->
        <xsd:attribute name="version" type="xsd:string" use="required"/>
    </xsd:complexType>
```

```
</xsd:element>
</xsd:schema>
```

B.6 Schema For JSON REST Collection Plug-in Mapper File

See how to set up a schema for a JSON REST collection plug-in mapper file for Oracle Audit Vault and Database Firewall.

In the following example, you can see how to set up a mapper schema:

Example B-6 JSON REST collection plug-in Mapper Schema

```
<?xml version="1.0"?>

<!--
 Copyright (c) 2013, 2017, Oracle and/or its affiliates. All rights reserved.
-->

<xsd:schema xmlns:xsd="http://foobar.example.org/2001/XMLSchema">
    <xsd:include schemaLocation="ezCollectorTemplate_schema.xsd" />

    <!-- Field ValueTransformation Type-->
    <xsd:complexType name="ParamType">
        <xsd:attribute name="Name" type="xsd:string" use="required"/>
        <xsd:attribute name="Value" type="xsd:string" use="required"/>
    </xsd:complexType>
    <!-- Field Transformation Type-->
    <xsd:complexType name="AuthenticationParamType">
        <xsd:sequence>
            <xsd:element name="Param" type="ParamType" minOccurs="0" maxOccurs="20" />
        </xsd:sequence>
    </xsd:complexType>
    <!-- XML Document Structure -->
    <xsd:element name="AVRESTCollectorTemplate">
        <xsd:complexType>
            <xsd:all>
                <xsd:element name="ResourceName" type="xsd:string"
                    minOccurs="1" maxOccurs="1" />
                <xsd:element name="HeaderInfo" minOccurs="0" maxOccurs="1">
                    <xsd:complexType>
                        <xsd:all>
                            <!-- StartTag tag contains Root element of XML Audit data file
-->
                            <xsd:element name="StartTag" type="xsd:string"
                                minOccurs="0" maxOccurs="1" />
                        </xsd:all>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="RecordInfo" minOccurs="1" maxOccurs="1">
                    <xsd:complexType>
                        <xsd:all>
                            <!-- start tag of xml audit record in XML audit file -->
                            <xsd:element name="StartTag" type="xsd:string"
```

```
                                minOccurs="1" maxOccurs="1" />
                        </xsd:all>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="ServiceDetails" minOccurs="1"
                    maxOccurs="1">
                    <xsd:complexType>
                        <xsd:all>
                            <xsd:element name="QueryFormat" type="xsd:string"
                                minOccurs="1" maxOccurs="1" />
                            <xsd:element name="TimeFormat" type="xsd:string"
                                minOccurs="1" maxOccurs="1" />
                            <xsd:element name="NextLink" minOccurs="0" maxOccurs="1">
                                <xsd:complexType>
                                    <xsd:all>
                                        <xsd:element name="NextLinkStartTag"
type="xsd:string"
                                            minOccurs="0" maxOccurs="1" />
                                        <xsd:element name="NextLinkPattern"
type="xsd:string"
                                            minOccurs="1" maxOccurs="1" />
                                    </xsd:all>
                                </xsd:complexType>
                            </xsd:element>
                            <xsd:element name="RESTAuthentication" minOccurs="1"
                                maxOccurs="1">
                                <xsd:complexType>
                                    <xsd:all>
                                        <xsd:element name="BasicAuth" minOccurs="0"
maxOccurs="1"> </xsd:element>
                                        <xsd:element name="OAuth2.0" minOccurs="0"
                                            maxOccurs="1">
                                            <xsd:complexType>
                                                <xsd:all>
                                                   <xsd:element name="grant_type"
type="xsd:string"
                                                        minOccurs="0" maxOccurs="1" />
                                                   <xsd:element name="scope"
type="xsd:string"
                                                        minOccurs="0" maxOccurs="1" />
                                                </xsd:all>
                                            </xsd:complexType>
                                        </xsd:element>
                                        <xsd:element name="Custom" minOccurs="0"
maxOccurs="1">
                                            <xsd:complexType>
                                                <xsd:all>
                                                    <xsd:element
name="authentication_class" type="xsd:string"
                                                        minOccurs="1" maxOccurs="1" />
                                                    <xsd:element
name="AuthenticationParam" type="AuthenticationParamType"
                                                        minOccurs="0" maxOccurs="1" />
                                                </xsd:all>
                                            </xsd:complexType>
```

```
                                    </xsd:element>
                                </xsd:all>
                            </xsd:complexType>
                        </xsd:element>
                    </xsd:all>
                </xsd:complexType>
            </xsd:element>
            <!-- Secured Target to AV server fields Mapping for Core, Large, Extension
                fields and Marker -->
            <xsd:element name="FieldMappingInfo" type="FieldMappingInfoType"
                minOccurs="1" maxOccurs="1" />
            <!-- Event Filter. This is optional. If it is not used, all the audit
                events will be collected -->
            <xsd:element name="EventFilter" type="EventFilterType"
                minOccurs="0" maxOccurs="1" />
        </xsd:all>
        <!-- Secured Target Type -->
        <xsd:attribute name="securedTargetType" type="xsd:string"
            use="required" />
        <!-- Max Secured Target version supported by the template -->
        <xsd:attribute name="maxSecuredTargetVersion" type="xsd:string"
            use="required" />
        <!-- Min Secured Target version supported by the template -->
        <xsd:attribute name="minSecuredTargetVersion" type="xsd:string" />
        <!-- Template file version -->
        <xsd:attribute name="version" type="xsd:string" use="required" />
    </xsd:complexType>
</xsd:element>
</xsd:schema>
```

B.7 Schema For REST Collector Plug-in Mapper File

See how to set up a schema for a REST collector plug-in mapper file for Oracle Audit Vault and Database Firewall.

In the following example, you can see how to set up a schema:

Example B-7 REST Collector Plug-in Mapper File

```
<?xml version="1.0"?>

<!--
 Copyright (c) 2013, 2019, Oracle and/or its affiliates. All rights reserved.
-->

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:include schemaLocation="ezCollectorTemplate_schema.xsd" />

    <!-- Field ValueTransformation Type-->
    <xsd:complexType name="ParamType">
        <xsd:attribute name="Name" type="xsd:string" use="required"/>
        <xsd:attribute name="Value" type="xsd:string" use="required"/>
    </xsd:complexType>
    <!-- Field Transformation Type-->
```

```
    <xsd:complexType name="AuthenticationParamType">
        <xsd:sequence>
            <xsd:element name="Param" type="ParamType" minOccurs="0" maxOccurs="20" />
        </xsd:sequence>
    </xsd:complexType>
    <!-- XML Document Structure -->
    <xsd:element name="AVRESTCollectorTemplate">
        <xsd:complexType>
            <xsd:all>
                <xsd:element name="ResourceName" type="xsd:string"
                    minOccurs="1" maxOccurs="1" />
                <xsd:element name="HeaderInfo" minOccurs="0" maxOccurs="1">
                    <xsd:complexType>
                        <xsd:all>
                            <!-- StartTag tag contains Root element of XML Audit data
file -->
                            <xsd:element name="StartTag" type="xsd:string"
                                minOccurs="0" maxOccurs="1" />
                        </xsd:all>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="RecordInfo" minOccurs="1" maxOccurs="1">
                    <xsd:complexType>
                        <xsd:all>
                            <!-- start tag of xml audit record in XML audit file -->
                            <xsd:element name="StartTag" type="xsd:string"
                                minOccurs="1" maxOccurs="1" />
                        </xsd:all>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="ServiceDetails" minOccurs="1"
                    maxOccurs="1">
                    <xsd:complexType>
                        <xsd:all>
                            <xsd:element name="QueryFormat" type="xsd:string"
                                minOccurs="1" maxOccurs="1" />
                            <xsd:element name="TimeFormat" type="xsd:string"
                                minOccurs="1" maxOccurs="1" />
                            <xsd:element name="NextLink" minOccurs="0" maxOccurs="1">
                                <xsd:complexType>
                                    <xsd:all>
                                        <xsd:element name="NextLinkStartTag"
type="xsd:string"
                                            minOccurs="0" maxOccurs="1" />
                                        <xsd:element name="NextLinkPattern"
type="xsd:string"
                                            minOccurs="1" maxOccurs="1" />
                                    </xsd:all>
                                </xsd:complexType>
                            </xsd:element>
                            <xsd:element name="RESTAuthentication" minOccurs="1"
                                maxOccurs="1">
                                <xsd:complexType>
                                    <xsd:all>
                                        <xsd:element name="BasicAuth" minOccurs="0"
```

```
maxOccurs="1"> </xsd:element>
                                        <xsd:element name="OAuth2.0" minOccurs="0"
                                            maxOccurs="1">
                                            <xsd:complexType>
                                                <xsd:all>
                                                   <xsd:element name="grant_type"
type="xsd:string"
                                                        minOccurs="0" maxOccurs="1" />
                                                   <xsd:element name="scope"
type="xsd:string"
                                                        minOccurs="0" maxOccurs="1" />
                                                </xsd:all>
                                            </xsd:complexType>
                                        </xsd:element>
                                        <xsd:element name="Custom" minOccurs="0"
maxOccurs="1">
                                            <xsd:complexType>
                                                <xsd:all>
                                                    <xsd:element
name="authentication_class" type="xsd:string"
                                                        minOccurs="1" maxOccurs="1" />
                                                    <xsd:element name="AuthenticationParam"
type="AuthenticationParamType"
                                                        minOccurs="0" maxOccurs="1" />
                                                </xsd:all>
                                            </xsd:complexType>
                                        </xsd:element>
                                    </xsd:all>
                                </xsd:complexType>
                            </xsd:element>
                        </xsd:all>
                    </xsd:complexType>
                </xsd:element>
                <!-- Secured Target to AV server fields Mapping for Core, Large, Extension
                    fields and Marker -->
                <xsd:element name="FieldMappingInfo" type="FieldMappingInfoType"
                    minOccurs="1" maxOccurs="1" />
                <!-- Event Filter. This is optional. If it is not used, all the audit
                    events will be collected -->
                <xsd:element name="EventFilter" type="EventFilterType"
                    minOccurs="0" maxOccurs="1" />
            </xsd:all>
            <!-- Secured Target Type -->
            <xsd:attribute name="securedTargetType" type="xsd:string"
                use="required" />
            <!-- Max Secured Target version supported by the template -->
            <xsd:attribute name="maxSecuredTargetVersion" type="xsd:string"
                use="required" />
            <!-- Min Secured Target version supported by the template -->
            <xsd:attribute name="minSecuredTargetVersion" type="xsd:string" />
            <!-- Template file version -->
            <xsd:attribute name="version" type="xsd:string" use="required" />
        </xsd:complexType>
    </xsd:element>
```

```
</xsd:schema>
```

B.8 Schema For Name Pattern Collection Plug-in Mapper File

See how to set up a schema for a name pattern collection plug-in mapper file for Oracle Audit Vault and Database Firewall.

In the following example, you can see how to set up a schema:

Example B-8 Name Pattern Collection Plug-in Mapper File

```
<?xml version="1.0"?>

<!--
 Copyright (c) 2013, 2019, Oracle and/or its affiliates. All rights reserved.
-->

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://
foobar.example.com/avdf/ezcollector/namepattern" targetNamespace="http://
foobar.example.com/avdf/ezcollector/namepattern" elementFormDefault="qualified" >
<!--Name Pattern-->
<xsd:simpleType name="DateFormatValues">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="yyyy_MM_dd"/>
        <xsd:enumeration value="dd_MM_yyyy"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="NameFormatDateType">
    <xsd:all>
         <xsd:element name="Format" type="DateFormatValues" minOccurs="1"
maxOccurs="1"/>
     <xsd:element name="Inc" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    </xsd:all>
</xsd:complexType>

<xsd:complexType name="NameFormatNumberType">
    <xsd:all>
         <xsd:element name="Format" type="xsd:string" minOccurs="1" maxOccurs="1"/>
     <xsd:element name="Inc" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    </xsd:all>
</xsd:complexType>

<xsd:complexType name="NameFormat">
    <xsd:choice>
         <xsd:element name="DateFormat" type="NameFormatDateType"/>
         <xsd:element name="NumberFormat" type="NameFormatNumberType"/>
         <xsd:element name="StringFormat" type="xsd:string"/>
    </xsd:choice>
</xsd:complexType>
```

```
<xsd:complexType name="NamePatternType">
    <xsd:choice>
        <xsd:element name="RollNamePattern">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="Name" type="xsd:string" minOccurs="1"
maxOccurs="1000"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="IncrementNamePattern">
            <xsd:complexType>
                <xsd:all>
                    <!-- Type and Format of the pattern Type Date, Format yyyy_MM_dd -->
                    <xsd:element name="NamePrefix" type="NameFormat" minOccurs="1"
maxOccurs="1"/>
                    <xsd:element name="Name" type="NameFormat" minOccurs="1" maxOccurs="1"/>
                    <xsd:element name="NameSuffix" type="NameFormat" minOccurs="1"
maxOccurs="1"/>
                </xsd:all>
            </xsd:complexType>
        </xsd:element>
    </xsd:choice>
</xsd:complexType>
</xsd:schema>
```

B.9 Schema For JSON Collector Plug-in Mapper File

See how to set up a schema for JSON collector plug-in mapper file for Oracle Audit Vault and Database Firewall.

In the following example, you can see how to set up a schema:

Example B-9 JSON Collector Plug-in Mapper File

```
<?xml version="1.0"?>

<!--
 Copyright (c) 2013, 2019, Oracle and/or its affiliates. All rights reserved.
-->

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:include schemaLocation="ezCollectorTemplate_schema.xsd"/>

<!-- XML Document Structure-->
<xsd:element name="AVJSONCollectorTemplate" >
    <xsd:complexType>
        <xsd:all>
            <xsd:element name="HeaderInfo"  minOccurs="0" maxOccurs="1">
                <xsd:complexType>
                    <xsd:all>
                        <!-- StartTag tag contains Root element of XML Audit data file-->
                        <xsd:element name="StartTag" type="xsd:string" minOccurs="0"
```

```
maxOccurs="1"/>
                    </xsd:all>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="RecordInfo"  minOccurs="1" maxOccurs="1">
                <xsd:complexType>
                    <xsd:all>
                        <!-- start tag of xml audit record in XML audit file-->
                        <xsd:element name="StartTag" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                    </xsd:all>
                </xsd:complexType>
            </xsd:element>
            <!-- Secured Target to AV server fields Mapping for Core, Large,
Extension fields and Marker-->
            <xsd:element name="FieldMappingInfo" type="FieldMappingInfoType"
minOccurs="1" maxOccurs="1"/>
            <!-- Event Filter. This is optional. If it is not used, all the audit
events will be collected-->
            <xsd:element name="EventFilter" type="EventFilterType" minOccurs="0"
maxOccurs="1"/>
        </xsd:all>
        <!-- Secured Target Type-->
        <xsd:attribute name="securedTargetType" type="xsd:string" use="required"/>
        <!-- Max Secured Target version supported by the template-->
        <xsd:attribute name="maxSecuredTargetVersion" type="xsd:string"
use="required"/>
        <!-- Min Secured Target version supported by the template-->
        <xsd:attribute name="minSecuredTargetVersion" type="xsd:string"/>
        <!-- Template file version-->
        <xsd:attribute name="version" type="xsd:string" use="required"/>
    </xsd:complexType>
</xsd:element>
</xsd:schema>
```

B.10 Schema For EZCollector Plug-in Mapper File

See how to set up a schema for an EZCollector plug-in mapper file for Oracle Audit Vault and Database Firewall.

In the following example, you can see how to set up a schema:

Example B-10 EZCollector Plug-in Mapper File

```
<?xml version="1.0"?>

<!--
 Copyright (c) 2013, 2019, Oracle and/or its affiliates. All rights reserved.
-->

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--Existing Set of Core Fields-->
<xsd:simpleType name="CoreFieldValues">
```

```
    <xsd:restriction base="xsd:string">
    <xsd:enumeration value="EventTimeUTC"/>
    <xsd:enumeration value="UserName"/>
    <xsd:enumeration value="OSUserName"/>
    <xsd:enumeration value="CommandClass"/>
    <xsd:enumeration value="TargetObject"/>
    <xsd:enumeration value="ClientHostName"/>
    <xsd:enumeration value="ClientIP"/>
    <xsd:enumeration value="ClientProgramName"/>
    <xsd:enumeration value="TargetOwner"/>
    <xsd:enumeration value="ErrorId"/>
    <xsd:enumeration value="ErrorMessage"/>
    <xsd:enumeration value="EventStatus"/>
    <xsd:enumeration value="EventName"/>
    <xsd:enumeration value="TargetType"/>
    <xsd:enumeration value="TerminalName"/>
    <xsd:enumeration value="ClientId"/>
    </xsd:restriction>
</xsd:simpleType>

<!--Existing Set of Large Fields-->
<xsd:simpleType name="LargeFieldValues">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="CommandText"/>
        <xsd:enumeration value="CommandParam"/>
    </xsd:restriction>
</xsd:simpleType>

<!-- Field ValueTransformation Type-->
<xsd:complexType name="ValueTransformationType">
    <xsd:attribute name="from" type="xsd:string" use="required"/>
    <xsd:attribute name="to" type="xsd:string" use="required"/>
</xsd:complexType>

<!-- Field FieldTransformation Type-->
<xsd:complexType name="FieldTransformationType">
    <xsd:attribute name="from" type="xsd:string" use="required"/>
    <xsd:attribute name="to" type="xsd:string" use="required"/>
</xsd:complexType>

<!-- Field Transformation Type-->
<xsd:complexType name="TransformationType">
    <xsd:sequence>
        <xsd:element name="ValueTransformation" type="ValueTransformationType"
minOccurs="0" maxOccurs="2000" />
        <xsd:element name="FieldTransformation" type="FieldTransformationType"
minOccurs="0" maxOccurs ="2000" />
    </xsd:sequence>
</xsd:complexType>

<!--FieldMappingInfo-->
<xsd:complexType name="FieldMappingInfoType">
    <xsd:all>
        <!-- Core Field Mapping-->
        <xsd:element name="CoreFields" minOccurs="1" maxOccurs="1">
```

```
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="Map" minOccurs="1" maxOccurs ="14">
                        <xsd:complexType>
                            <xsd:all>
                                <xsd:element name="Name" type="xsd:string" />
                                <xsd:element name="MapTo"
type="CoreFieldValues" />
                                <xsd:element name="TimestampPattern"
type="xsd:string" minOccurs="0" maxOccurs="1" />
                                <xsd:element name="Transformation"
type="TransformationType" minOccurs="0" maxOccurs="1" />
                            </xsd:all>
                        </xsd:complexType>
                    </xsd:element>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <!-- Large Field Mapping -->
        <xsd:element name="LargeFields" minOccurs="0" maxOccurs="1">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="Map" minOccurs="0" maxOccurs="2">
                        <xsd:complexType>
                            <xsd:all>
                                <xsd:element name="Name" type="xsd:string" />
                                <xsd:element name="MapTo" type="LargeFieldValues" />
                                <xsd:element name="Transformation"
type="TransformationType" minOccurs="0" maxOccurs="1" />
                            </xsd:all>
                        </xsd:complexType>
                    </xsd:element>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <!-- List of fields to be mapped to extensible fields-->
        <xsd:element name="ExtensionField" minOccurs="0" maxOccurs="1">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="Name" type="xsd:string" minOccurs="0"
maxOccurs ="500" />
                    <xsd:element name="ComplexName" minOccurs="0" maxOccurs ="500" >
                        <xsd:complexType>
                            <xsd:all>
                                <xsd:element name="Name" type="xsd:string" />
                                <xsd:element name="RegExp" type="xsd:string" />
                            </xsd:all>
                        </xsd:complexType>
                    </xsd:element>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <!-- List of fields which uniquely identify each audit record-->
        <xsd:element name="MarkerField" minOccurs="1" maxOccurs="1">
            <xsd:complexType>
```

```
                <xsd:sequence>
                    <xsd:element name="Name" type="xsd:string"  minOccurs="1"
maxOccurs="20"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
    </xsd:all>
</xsd:complexType>

<!-- Filter Type-->
<xsd:complexType name="FilterType">
    <xsd:sequence>
        <!-- Provide all Included or Excluded values for given source field name-->
        <xsd:element name="Value" minOccurs="1" maxOccurs="1000" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>

<!-- Event Filter Type-->
<xsd:complexType name="EventFilterType">
    <xsd:sequence>
    <!-- Source Field Name through which audit events will be filtered-->
    <xsd:element name="FieldName" type="xsd:string"/>
    <xsd:choice>
        <!-- Use either Include or Exclude to filter audit events-->
        <xsd:element name="Include" type="FilterType" minOccurs="1" maxOccurs="1"/>
        <xsd:element name="Exclude" type="FilterType" minOccurs="1" maxOccurs="1"/>
    </xsd:choice>
    </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

C

Example Code

Learn from examples the different types of collection plug-ins, including database tables, XML files, and Java-based file collection plug-ins.

C.1 Database Table Collection Plug-in Example

See examples of Oracle Audit Vault database table collection plug-in mapper files and database table plug-in manifest files.

C.1.1 Database Table Collection Plug-in Mapper File

Learn which Oracle Audit Vault attributes and fields are mandatory and which are optional for database table collection plug-in mapper files.

Oracle Audit Vault database table collection plug-in mapper files have certain mandatory fields. S

Mandatory Fields

These attributes and fields are mandatory:

- `securedTargetType`
- `maxSecuredTargetVersion`
- `version`
- `TableName`
- `Driver`
- `EventTimeUTC`
- `CommandClass` transformations
- `EventStatus` transformations
- `MarkerField`

Optional Fields

Source names that map to Oracle Audit Vault Server fields are not mandatory. However, if the information is not provided when data collection starts, then all audit records are treated as invalid:

- `UserName`
- `CommandClass`

Example C-1 Sample XML Mapper File for a Database Table Collection Plug-in

```
<AVTableCollectorTemplate securedTargetType="DBSOURCE"
minSecuredTargetVersion="10.2.0"
```

```
          maxSecuredTargetVersion="11.0" version="1.0" >
              <!--Example Template for a database Collector-->
              <!-- Attributes: securedTargetType,
maxSecuredTargetVersion,
                          and version are mandatory;
                          minSecuredTargetVersion attribute is
optional -->
              <!-- Accepted Format for min/maxSecuredTargetVersion and
                        version attribute value is numbers separated
by
                          dots (For example: 12.2,10.3.2, 11.2.3.0
etc..)-->
             <!-- Audit Table Information  -->
             <!-- Name of Audit Table: Mandatory information -->
  <TableName>dummy_auditTable</TableName>
             <!-- Source Connection Information -->
  <ConnectionInfo>
             <DataSource>oracle.jdbc.pool.OracleDataSource</
DataSource>
    <!--Datasource class name for current secured target type:
                  Mandatory information -->
    </ConnectionInfo>
             <!-- This Gives Mapping Information of Source Fields to
various AV
                   Fields(core and large fields)  -->
             <!-- There should be no many-to-one mappings from source
fields to
                   AV Server fields -->
  <FieldMappingInfo>
             <!-- Mapping of Source Fields to Core Fields of AV
server  -->
             <!-- Source fields specified in core field mappings must
be of SQL
                  Datatype: String OR convertible to String-->
      <CoreFields>
           <Map>
             <!-- Mandatory: EventTime mapping information -->
        <Name>EVENT_TIME</Name>
        <MapTo>EventTimeUTC</MapTo>
      </Map>
         <Map>
             <!-- If UserName core field mapping is not provided,
Audit Data
               Collection still starts successfully, but every audit
record
               will be treated as invalid -->
        <Name>USER_ID</Name>
        <MapTo>UserName</MapTo>
      </Map>
      <Map>
        <Name>OS_USER_ID</Name>
        <MapTo>OSUserName</MapTo>
      </Map>
      <Map>
```

```
              <!-- If source name, the ACTION field, for CommandClass core
field
                mapping is not  provided, Audit Data Collection still starts
                successfully, but all audit records are treated as invalid --
>

        <Name>ACTION</Name>
        <MapTo>CommandClass</MapTo>

              <!-- Mandatory: value transformation from secured target field
value
                  to command class field value. Value of "to" Attribute is
from AV
                  Event set  -->

        <Transformation>
          <ValueTransformation from="1" to="CREATE"/>
          <ValueTransformation from="2" to="INSERT"/>
          <ValueTransformation from="3" to="SELECT"/>
          <ValueTransformation from="4" to="CREATE"/>
          <ValueTransformation from="15" to="READ"/>
          <ValueTransformation from="30" to="LOGON"/>
          <ValueTransformation from="34" to="LOGOFF"/>
          <ValueTransformation from="35" to="ACQUIRE"/>
        </Transformation>
      </Map>
      <Map>
        <Name>OBJ_NAME</Name>
        <MapTo>TargetObject</MapTo>
      </Map>
      <Map>
        <Name>USER_HOST</Name>
        <MapTo>ClientHostName</MapTo>
      </Map>
      <Map>
        <Name>OBJ_CREATOR</Name>
        <MapTo>TargetOwner</MapTo>
      </Map>
      <Map>
        <Name>STATUS</Name>
        <MapTo>EventStatus</MapTo>

              <!-- Value transformation for "STATUS" source field value.
                Mandatory: EventStatus value transformation.
                There are three possible values for EventStatus:
                SUCCESS, FAILURE, UNKNOWN -->
        <Transformation>
          <ValueTransformation from="0" to="FAILURE"/>
          <ValueTransformation from="1" to="SUCCESS"/>
          <ValueTransformation from="2" to="UNKNOWN"/>
        </Transformation>
      </Map>
    </CoreFields>

            <!-- Mapping of Source Fields to Large Fields of AV server i.e
```

```
fields
                 with huge content  -->
            <!-- Secured target fields specified in large field
mappings must be
                of SQL Datatype:CLOB OR SQL Datatype:String OR
convertible to
                String -->
    <LargeFields>
      <Map>
        <Name>SQL_TEXT</Name>
        <MapTo>CommandText</MapTo>
      </Map>
      <Map>
        <Name>COMMAND_PARAMETER</Name>
        <MapTo>CommandParam</MapTo>
      </Map>
    </LargeFields>

            <!-- These secured target fields are collected in a single
extension
               field, all name-value pairs separated by standard
delimiter -->
            <!-- Secured target fields specified in extension field
mapping must
                  be of SQL Datatype:String OR convertible to String --
>
    <ExtensionField>
      <Name>DB_ID</Name>
      <Name>INSTANCE</Name>
      <Name>PROCESS</Name>
      <Name>TERMINAL</Name>
    </ExtensionField>

            <!-- Mandatory: Secured target fields for MarkerField
               A group of secured target fields to uniquely identify
each Audit
               Record -->
            <!-- Secured target fields specified to be used as
MarkerField mapping
                  must be of SQL Datatype:String OR convertible to
String -->
    <MarkerField>
      <Name>SESSION_ID</Name>
      <Name>ENTRY_ID</Name>
    </MarkerField>
  </FieldMappingInfo>
</AVTableCollectorTemplate>
```

Related Topics

- Audit Vault Server Fields
  You can map Oracle Audit Vault and Database Firewall events and fields in your collection plug-ins.

C.1.2 Database Table Collection Plug-in Manifest File

See an example of a database table collection plug-in manifest file.

This is a sample manifest file for a database table collection plug-in.

Example C-2 Sample Manifest File for a Database Table Collection Plug-in

```
<?xml version="1.0"?>

<plugin xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://xmlns.oracle.com/av/plugin plugin-
manifest.xsd"
        xmlns="http://xmlns.oracle.com/av/plugin"
        name="HRMS-Template"
        id="com.oracle.av.plugin"
        version="1.0"
        provider-name="Oracle Corp."
        copyright="Copyright Oracle Corp. 2011">

               <!-- targetVersion: Version of Oracle Audit Vault supported
by this
                     plugin. This is represented by the "min"  attribute of
                     <targetVersion> tag      -->
   <targetVersion min="11.1.0.0.0"/>

   <extensionSet>
      <extensionPoint type= "securedTargetType">
               <!-- Tag: fileList: Lists all files that ship with the
plugin   -->
         <fileList>
            <jars></jars>
            <templates>
               <include file="DBSource-Mapper.xml"/>
            </templates>
            <bin></bin>
            <config></config>
            <shell></shell>
            <patch></patch>
            <unresolved-external>
            </unresolved-external>
         </fileList>
               <!-- Tag:  securedTargetTypeInfo: Contains secured target
type and
                     trail information  -->
         <securedTargetTypeInfo name="DBSOURCE"/>

               <!-- Tag:  trailType: contains trail type, location ,
classname for
                    source type testSource -->
         <trailInfo>
            <trailType>TABLE</trailType>
            <className
name="oracle.av.platform.agent.collfwk.Collector.table.DatabaseTableCollector
```

```
"/>
         </trailInfo>

               <!-- eventPatch: OPTIONAL field that indicates any
event patches
                  that need to be applied as part of plugin deployment
                  The files listed here must be present in the <patch>
                  tag entries. The order in which the patches need to
                  applied can be controlled via the "order" attribute
                  Patches with lower "order" value will be applied
                  first            --
>
         <eventPatch name="p6753288_11.1.2.0.0_GENERIC.zip" order="2"/>
      </extensionPoint>
   </extensionSet>
</plugin>
```

C.2 XML File Collection Plug-in Examples

Learn about the plug-in mapper file and plug-in manifest file attributes and fields for Oracle Audit Vault and Database Firewall.

C.2.1 XML File Collection Plug-In Mapper File

See an XML template collector file example, and find out about the attributes and fields used with XML file collection in Oracle Audit Vault and Database Firewall.

There are both mandatory attributes and fields, and fields that are not mandatory, but that can cause your audit records to be treated as invalid.

Mandatory Attributes and Fields for XML File Collection Plug-In Mapper Files

- `securedTargetType`
- `maxSecuredTargetVersion`
- `version`
- `HeaderInfo`
- `RecordInfo`
- `EventTimeUTC`
- `CommandClass` transformations
- `EventStatus` transformations
- `MarkerField`

Standard Fields for XML File Collection Plug-In Mapper Files

Source names that map to these Oracle Audit Vault Server fields are not mandatory. However, if the information specified by these fields is not provided, then when data collection starts, all audit records are treated as invalid:

- `UserName`

- CommandClass

Example C-3 Sample XML File Collection Plug-in Mapper File

```
<AVXMLCollectorTemplate securedTargetType="XMLSOURCE"
  maxSecuredTargetVersion="11.0"
         version="1.0">
            <!--Example Template for XML template collector-->
            <!-- Attributes: "securedTargetType", "maxSecuredTargetVersion"
and
                 "version" are mandatory attributes,
"minSecuredTargetVersion"
                  attribute is optional -->
            <!-- Accepted Format for min/maxSecuredTargetVersion and version
                  attribute value is numbers separated by dots (For example:
                  12.2,10.3.2, 11.2.3.0 etc..)-->
            <!-- Header Information like XML Header start tag  -->
  <HeaderInfo>
            <!-- Mandatory: HeaderInfo-->
            <!-- Value in this tag gives Root tag of the XML audit file-->
    <StartTag>Audit</StartTag>
  </HeaderInfo>

            <!-- Record Information like Record Start tag and conformation
to hold
                 original record  -->
  <RecordInfo>
            <!-- Mandatory: RecordInfo -->
            <!-- Provides starting tag of audit record in XML audit file -->
    <StartTag>AuditRecord</StartTag>
  </RecordInfo>

            <!-- Gives Mapping Information of Source Fields to various  AV
Fields
                   (core and large fields)  -->
            <!-- Not Allowed: many-to-one mapping from source field to
              AV Server fields -->
  <FieldMappingInfo>
            <!-- Mapping of Source Fields to Core Fields of AV server
                Source fields specified in core field mappings must be of
SQL
                Datatype: String OR convertible to String -->
      <CoreFields>
         <Map>
        <Name>EVENT_TIME</Name>
        <MapTo>EventTimeUTC</MapTo>
        <TimestampPattern>yyyy-MM-dd HH:mm:ss.SSS</TimestampPattern>
         </Map>
         <Map>
            <!-- If UserName core field mapping is not provided, Audit Data
               Collection still starts successfully, but every audit record
               will be treated as invalid -->
        <Name>USER_ID</Name>
        <MapTo>UserName</MapTo>
      </Map>
```

```
      <Map>
        <Name>OS_USER_ID</Name>
        <MapTo>OSUserName</MapTo>
      </Map>
      <Map>
                  <!-- If source name, the ACTION field, for
CommandClass
                     core field mapping is not provided, Audit Data
Collection
                     still starts successfully, but all audit records
are treated
                     as invalid -->
        <Name>ACTION</Name>
        <MapTo>CommandClass</MapTo>
              <!-- Mandatory: value transformations from source to
Action
                   field value. Value of "to" Attribute is from AV
Event set  -->
        <Transformation>
          <ValueTransformation from="1" to="CREATE"/>
          <ValueTransformation from="2" to="INSERT"/>
          <ValueTransformation from="3" to="SELECT"/>
          <ValueTransformation from="4" to="CREATE"/>
          <ValueTransformation from="15" to="READ"/>
          <ValueTransformation from="30" to="LOGON"/>
          <ValueTransformation from="34" to="LOGOFF"/>
          <ValueTransformation from="35" to="ACQUIRE"/>
        </Transformation>
      </Map>
      <Map>
        <Name> OBJ_NAME</Name>
        <MapTo>TargetObject</MapTo>
      </Map>
      <Map>
        <Name>USER_HOST</Name>
        <MapTo>ClientHostName</MapTo>
      </Map>
      <Map>
        <Name>OBJ_CREATOR</Name>
        <MapTo>TargetOwner</MapTo>
      </Map>
      <Map>
        <Name>STATUS</Name>
        <MapTo>EventStatus</MapTo>
            <!-- Specifying value transformation for Status source
field value.
                 Mandatory: EventStatus value transformation.
                 There are three possible values for EventStatus:
                 SUCCESS, FAILURE, UNKNOWN -->
        <Transformation>
          <ValueTransformation from="0" to="FAILURE"/>
          <ValueTransformation from="1" to="SUCCESS"/>
          <ValueTransformation from="2" to="UNKNOWN"/>
        </Transformation>
      </Map>
```

```
    </CoreFields>

            <!-- Mapping of Source Fields to Large Fields of AV server i.e
fields
                with huge content  -->
            <!-- Source fields specified in large field mappings must be of
SQL
                Datatype:CLOB OR SQL Datatype:String OR convertible to
String -->
    <LargeFields>
      <Map>
        <Name>SQL_TEXT</Name>
        <MapTo>CommandText</MapTo>
      </Map>
      <Map>
        <Name>COMMAND_PARAMETER</Name>
        <MapTo>CommandParam</MapTo>
      </Map>
    </LargeFields>

            <!-- These Source fields will be collected in a single extension
               field, all name-value pairs are separated by standard
delimiter -->
            <!-- Source fields specified in extension field mapping must be
of
               SQL Datatype:String OR convertible to String -->
    <ExtensionField>
      <Name>DB_ID</Name>
      <Name>INSTANCE</Name>
      <Name>PROCESS</Name>
      <Name>TERMINAL</Name>
    </ExtensionField>

            <!-- This is group of source fields for uniquely identifying
each
                 Audit Record Marker -->
            <!-- Source fields specified to be used as Marker field mapping
must
                 be of SQL Datatype:String OR convertible to String -->
            <!-- Mandatory: Source fields for MarkerField -->
    <MarkerField>

      <Name>SESSION_ID</Name>
      <Name>ENTRY_ID</Name>
    </MarkerField>
  </FieldMappingInfo>
</AVXMLCollectorTemplate>
```

Related Topics

- Audit Vault Server Fields
  You can map Oracle Audit Vault and Database Firewall events and fields in your collection plug-ins.

C.2.2 XML File Collection Plug-In Manifest File

See an XML file collection plug-in manifest file example used with XML file collection in Oracle Audit Vault and Database Firewall.

This is a sample manifest file for an XML file collection plug-in.

Example C-4 Sample Manifest File for an XML File Collection Plug-in

```
<?xml version="1.0"?>

<plugin xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://xmlns.oracle.com/av/plugin plugin-
manifest.xsd"
        xmlns="http://xmlns.oracle.com/av/plugin"
        name="Oracle-XML-Template"
        id="com.oracle.av.plugin"
        version="1.0"
        provider-name="Oracle Corp."
        copyright="Copyright Oracle Corp. 2011">

            <!-- targetVersion: Version of Oracle Audit Vault
supported by
              this plugin. This is represented by the "min" attribute
of
              targetVersion> tag                -->
   <targetVersion min="11.1.0.0.0"/>

   <extensionSet>
      <extensionPoint type= "securedTargetType">
            <!-- fileList: Lists *all* the files that ship with the
plugin  -->
         <fileList>
            <jars></jars>
            <templates>
                 <include file="XMLSource-Mapper.xml"/>
            </templates>
            <bin></bin>
            <config></config>
            <shell></shell>
            <patch></patch>
            <unresolved-external></unresolved-external>


         </fileList>

            <!-- securedTargetTypeInfo: Contains source type and trail
information
                    -->
         <securedTargetTypeInfo name="oracle"/>

            <!-- trailType: contains trail type, location , classname
for
               source type testSource -->
```

```
         <trailInfo>
            <trailType>DIRECTORY</trailType>
            <className
name="oracle.av.platform.agent.collfwk.ezcollector.xml.XMLFileCollector"/>

         </trailInfo>

            <!-- eventPatch: OPTIONAL field that indicates any event
patches
                that need to be applied as part of plugin deployment-->
                The files listed here must be present in the <patch>-->
                tag entries. The order in which the patches need to -->
                applied can be controlled via the "order" attribute -->
                Patches with lower "order" value will be applied    -->
                first                                               -->
         <eventPatch name="p6753288_11.1.2.0.0_GENERIC.zip" order="2"/>
      </extensionPoint>
   </extensionSet>
</plugin>
```

C.3 JSON File Collection Plug-in Example

Learn about the JSON plug-in mapper file and plug-in manifest file attributes and fields for Oracle Audit Vault and Database Firewall.

C.3.1 JSON File Collection Plug-In Mapper File

See a JSON template collector file example, and find out about the attributes and fields used with JSON file collection in Oracle Audit Vault and Database Firewall.

There are both mandatory attributes and fields, and fields that are not mandatory, but that can cause your audit records to be treated as invalid.

Mandatory Attributes and Fields for JSON File Collection Plug-In Mapper Files

- `securedTargetType`
- `maxSecuredTargetVersion`
- `version`
- `HeaderInfo`
- `RecordInfo`
- `EventTimeUTC`
- `CommandClass` transformations
- `EventStatus` transformations
- `MarkerField`

Standard Fields for JSON File Collection Plug-In Mapper Files

Source names that map to these Oracle Audit Vault Server fields are not mandatory. However, if the information specified by these fields is not provided, then when data collection starts, all audit records are treated as invalid:

- UserName
- CommandClass

Example C-5 Sample JSON File Collection Plug-in Mapper File

```
<?xml version="1.0" encoding="UTF-8"?>
<AVJSONCollectorTemplate securedTargetType="JSONSOURCE"
maxSecuredTargetVersion="11.0" version="1.0">
   <!--Example Template for JSON template collector-->
   <!-- Attributes: "securedTargetType", "maxSecuredTargetVersion" and
   "version" are mandatory attributes, "minSecuredTargetVersion"
   attribute is optional -->
   <!-- Accepted Format for min/maxSecuredTargetVersion and version
   attribute value is numbers separated by dots (For example:
   12.2,10.3.2, 11.2.3.0 etc..)-->
   <!-- Header Information like JSON Header start tag -->
   <HeaderInfo>
      <!-- Mandatory: HeaderInfo-->
      <!-- Value in this tag gives Root tag of the JSON audit file-->
      <StartTag>ITEMS</StartTag>
   </HeaderInfo>
   <!-- Record Information like Record Start tag and conformation to
hold
   original record -->
   <RecordInfo>
      <!-- Mandatory: RecordInfo -->
      <!-- Provides starting tag of audit record in JSON audit file -->
      <StartTag>SESSION_ID</StartTag>
   </RecordInfo>
   <!-- Gives Mapping Information of Source Fields to various AV Fields
   (core and large fields) -->
   <!-- Not Allowed: many-to-one mapping from source field to
   AV Server fields -->
   <FieldMappingInfo>
      <!-- Mapping of Source Fields to Core Fields of AV server
      Source fields specified in core field mappings must be of SQL
      Datatype: String OR convertible to String -->
      <CoreFields>
         <Map>
            <Name>$.EVENT_TIME</Name>
            <MapTo>EventTimeUTC</MapTo>
            <TimestampPattern>yyyy-MM-dd HH:mm:ss.SSS</
TimestampPattern>
         </Map>
         <Map>
            <!-- If UserName core field mapping is not provided, Audit
Data
            Collection still starts successfully, but every audit
record
            will be treated as invalid -->
            <Name>$.USER_ID</Name>
            <MapTo>UserName</MapTo>
         </Map>
         <Map>
```

```
            <Name>$.OS_USER_ID</Name>
            <MapTo>OSUserName</MapTo>
         </Map>
         <Map>
            <!-- If source name, the ACTION field, for CommandClass
            core field mapping is not provided, Audit Data Collection
            still starts successfully, but all audit records are treated
            as invalid -->
            <Name>$.ACTION</Name>
            <MapTo>CommandClass</MapTo>
            <!-- Mandatory: value transformations from source to Action
            field value. Value of "to" Attribute is from AV Event set -->
            <Transformation>
               <ValueTransformation from="1" to="CREATE" />
               <ValueTransformation from="2" to="INSERT" />
               <ValueTransformation from="3" to="SELECT" />
               <ValueTransformation from="4" to="CREATE" />
               <ValueTransformation from="15" to="READ" />
               <ValueTransformation from="30" to="LOGON" />
               <ValueTransformation from="34" to="LOGOFF" />
               <ValueTransformation from="35" to="ACQUIRE" />
            </Transformation>
         </Map>
         <Map>
            <Name>$.OBJ_NAME</Name>
            <MapTo>TargetObject</MapTo>
         </Map>
         <Map>
            <Name>$.USER_HOST</Name>
            <MapTo>ClientHostName</MapTo>
         </Map>
         <Map>
            <Name>$.OBJ_CREATOR</Name>
            <MapTo>TargetOwner</MapTo>
         </Map>
         <Map>
            <Name>$.STATUS</Name>
            <MapTo>EventStatus</MapTo>
            <!-- Specifying value transformation for Status source field
value.
            Mandatory: EventStatus value transformation.
            There are three possible values for EventStatus:
            SUCCESS, FAILURE, UNKNOWN -->
            <Transformation>
               <ValueTransformation from="0" to="FAILURE" />
               <ValueTransformation from="1" to="SUCCESS" />
               <ValueTransformation from="2" to="UNKNOWN" />
            </Transformation>
         </Map>
      </CoreFields>
      <!-- Mapping of Source Fields to Large Fields of AV server i.e fields
      with huge content -->
      <!-- Source fields specified in large field mappings must be of SQL
      Datatype:CLOB OR SQL Datatype:String OR convertible to String -->
      <LargeFields>
```

```
            <Map>
               <Name>$.SQL_TEXT</Name>
               <MapTo>CommandText</MapTo>
            </Map>
            <Map>
               <Name>$.COMMAND_PARAMETER</Name>
               <MapTo>CommandParam</MapTo>
            </Map>
         </LargeFields>
         <!-- These Source fields will be collected in a single extension
         field, all name-value pairs are separated by standard delimiter
-->
         <!-- Source fields specified in extension field mapping must be
of
         SQL Datatype:String OR convertible to String -->
         <ExtensionField>
            <Name>$.DB_ID</Name>
            <Name>$.INSTANCE</Name>
            <Name>$.PROCESS</Name>
            <Name>$.TERMINAL</Name>
         </ExtensionField>
         <!-- This is group of source fields for uniquely identifying each
         Audit Record Marker -->
         <!-- Source fields specified to be used as Marker field mapping
must
         be of SQL Datatype:String OR convertible to String -->
         <!-- Mandatory: Source fields for MarkerField -->
         <MarkerField>
            <Name>$.SESSION_ID</Name>
            <Name>$.ENTRY_ID</Name>
         </MarkerField>
      </FieldMappingInfo>
</AVJSONCollectorTemplate>
```

Related Topics

- Audit Vault Server Fields
  You can map Oracle Audit Vault and Database Firewall events and fields in your collection plug-ins.

C.3.2 JSON File Collection Plug-In Manifest File

See a JSON file collection plug-in manifest file example used with JSON file collection in Oracle Audit Vault and Database Firewall.

This is a sample manifest file for an JSON file collection plug-in.

Example C-6 Sample Manifest File for a JSON File Collection Plug-in

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin xmlns="http://foobar.example.com/av/plugin" xmlns:xsi="http://
foobar.example.org/2001/XMLSchema-instance" xsi:schemaLocation="http://
xmlns.oracle.com/av/plugin plugin-manifest.xsd" name="Oracle-XML-
Template" id="com.oracle.av.plugin" version="1.0" provider-
```

```
name="Oracle Corp." copyright="Copyright Oracle Corp. 2011">
   <!-- targetVersion: Version of Oracle Audit Vault supported by
   this plugin. This is represented by the "min" attribute of
   targetVersion> tag -->
   <targetVersion min="11.1.0.0.0" />
   <extensionSet>
      <extensionPoint type="securedTargetType">
         <!-- fileList: Lists *all* the files that ship with the plugin -->
         <fileList>
            <jars />
            <templates>
               <include file="JSONSource-Mapper.xml" />
            </templates>
            <bin />
            <config />
            <shell />
            <patch />
            <unresolved-external />
         </fileList>
         <!-- securedTargetTypeInfo: Contains source type and trail
information-->
         <securedTargetTypeInfo name="json_file_secured_target" />
         <!-- trailType: contains trail type, location , classname for
         source type testSource -->
         <trailInfo>
            <trailType>DIRECTORY</trailType>
            <className
name="oracle.av.platform.agent.collfwk.ezcollector.json.MultiJSONFileCollecto
rFactory" />
         </trailInfo>
         <!-- eventPatch: OPTIONAL field that indicates any event patches
         that need to be applied as part of plugin deployment
         The files listed here must be present in the patch
         tag entries. The order in which the patches need to
         applied can be controlled via the "order" attribute
         Patches with lower "order" value will be applied first -->
         <eventPatch name="p6753288_11.1.2.0.0_GENERIC.zip" order="2" />
      </extensionPoint>
   </extensionSet>
</plugin>
```

C.4 CSV File Collection Plug-in Example

Learn about the CSV plug-in mapper file and plug-in manifest file attributes and fields for Oracle Audit Vault and Database Firewall.

C.4.1 CSV File Collection Plug-In Mapper File

See a CSV template collector file example, and find out about the attributes and fields used with CSV file collection in Oracle Audit Vault and Database Firewall.

There are both mandatory attributes and fields, and fields that are not mandatory, but that can cause your audit records to be treated as invalid.

Mandatory Attributes and Fields for CSV File Collection Plug-In Mapper Files

- `securedTargetType`
- `maxSecuredTargetVersion`
- `version`
- `HeaderInfo`
- `RecordInfo`
- `EventTimeUTC`
- `CommandClass` transformations
- `EventStatus` transformations
- `MarkerField`

Standard Fields for CSV File Collection Plug-In Mapper Files

Source names that map to these Oracle Audit Vault Server fields are not mandatory. However, if the information specified by these fields is not provided, then when data collection starts, all audit records are treated as invalid:

- `UserName`
- `CommandClass`

Example C-7 Sample CSV File Collection Plug-in Mapper File

```
<?xml version="1.0" encoding="UTF-8"?>
<AVCSVCollectorTemplate securedTargetType="csv_file_secured_target"
maxSecuredTargetVersion="11.0" version="1.0">
   <!--Example Template for CSV template collector-->
   <!-- Attributes: "securedTargetType", "maxSecuredTargetVersion" and
   "version" are mandatory attributes, "minSecuredTargetVersion"
   attribute is optional -->
   <!-- Accepted Format for min/maxSecuredTargetVersion and version
   attribute value is numbers separated by dots (For example:
   12.2,10.3.2, 11.2.3.0 etc..)-->
   <!-- Header Information like CSV Header start tag -->
   <HeaderInfo>
      <!-- Mandatory: HeaderInfo-->
      <!-- Hardcoded to CSV -->
      <StartTag>CSV</StartTag>
   </HeaderInfo>
   <!-- Record Information like Record Start tag and conformation to
hold
   original record -->
   <RecordInfo>
      <!-- Mandatory: RecordInfo -->
      <!-- Hardcoded to CSV -->
      <StartTag>CSV</StartTag>
   </RecordInfo>
   <!-- Gives Mapping Information of Source Fields to various AV Fields
   (core and large fields) -->
```

```
   <!-- Not Allowed: many-to-one mapping from source field to
   AV Server fields -->
   <FieldMappingInfo>
      <!-- Mapping of Source Fields to Core Fields of AV server
      Source fields specified in core field mappings must be
      either string OR convertible to string -->

      <!-- CSV files have "COMMA" as field delimiter -->
      <!-- The first field has index 0, second field has index 1 and so on --
>

      <CoreFields>
         <!-- In our CSV sample data, EVENT_TIME field has index 2 -->
         <!-- Hence 2 is used in below EventTimeUTC mapping -->
         <Map>
            <Name>2</Name>
            <MapTo>EventTimeUTC</MapTo>
            <TimestampPattern>yyyy-MM-dd'T'HH:mm:ss.SSSZ</TimestampPattern>
         </Map>
         <Map>
            <!-- If UserName core field mapping is not provided, Audit Data
            Collection still starts successfully, but every audit record
            will be treated as invalid -->
            <Name>5</Name>
            <MapTo>UserName</MapTo>
         </Map>

         <Map>
            <!-- If source name, the ACTION field, for CommandClass
            core field mapping is not provided, Audit Data Collection
            still starts successfully, but all audit records are treated
            as invalid -->
            <Name>1</Name>
            <MapTo>CommandClass</MapTo>
            <!-- Mandatory: value transformations from source to Action
            field value. Value of "to" Attribute is from AV Event set -->
            <Transformation>
               <ValueTransformation from="createUser" to="CREATE" />
               <ValueTransformation from="createCollection" to="CREATE" />
               <ValueTransformation from="authenticate" to="AUTHENTICATE" />
               <ValueTransformation from="dropCollection" to="DROP" />
               <ValueTransformation from="dropUser" to="DROP" />
            </Transformation>
         </Map>
         <Map>
            <Name>1</Name>
            <MapTo>TargetObject</MapTo>
            <Transformation>
               <FieldTransformation from="createUser" to="6" />
               <FieldTransformation from="createCollection" to="6" />
               <FieldTransformation from="authenticate" to="6" />
               <FieldTransformation from="dropCollection" to="6" />
               <FieldTransformation from="dropUser" to="6" />
            </Transformation>
         </Map>
```

```
         <Map>
            <Name>1</Name>
            <MapTo>TargetType</MapTo>
            <Transformation>
               <ValueTransformation from="createUser" to="USER" />
               <ValueTransformation from="createCollection"
to="COLLECTION" />
               <ValueTransformation from="authenticate" to="USER" />
               <ValueTransformation from="dropCollection"
to="COLLECTION" />
               <ValueTransformation from="dropUser" to="USER" />
            </Transformation>
         </Map>
         <Map>
            <Name>3</Name>
            <MapTo>ClientIP</MapTo>
         </Map>

         <Map>
            <Name>7</Name>
            <MapTo>EventStatus</MapTo>
            <!-- Specifying value transformation for Status source
field value.
            Mandatory: EventStatus value transformation.
            There are three possible values for EventStatus:
            SUCCESS, FAILURE, UNKNOWN -->
            <Transformation>
               <ValueTransformation from="0" to="FAILURE" />
               <ValueTransformation from="100" to="SUCCESS" />
               <ValueTransformation from="200" to="UNKNOWN" />
            </Transformation>
         </Map>
      </CoreFields>
      <!-- Mapping of Source Fields to Large Fields of AV server i.e
fields
      with huge content -->
      <!-- Source fields specified in large field mappings must be of
SQL
      Datatype:CLOB OR SQL Datatype:String OR convertible to String -->
      <LargeFields>
         <Map>
            <Name>11</Name>
            <MapTo>CommandText</MapTo>
         </Map>
         <Map>
            <Name>12</Name>
            <MapTo>CommandParam</MapTo>
         </Map>
      </LargeFields>
      <!-- These Source fields will be collected in a single extension
      field, all name-value pairs are separated by standard delimiter
-->
      <!-- Source fields specified in extension field mapping must be
of
      SQL Datatype:String OR convertible to String -->
```

```
      <ExtensionField>
         <ComplexName>
            <Name>10</Name>
            <DisplayName>sessionid</DisplayName>
         </ComplexName>
         <ComplexName>
            <Name>13</Name>
            <DisplayName>entryid</DisplayName>
         </ComplexName>
      </ExtensionField>
      <!-- This is group of source fields for uniquely identifying each
      Audit Record Marker -->
      <!-- Source fields specified to be used as Marker field mapping must
      be of SQL Datatype:String OR convertible to String -->
      <!-- Mandatory: Source fields for MarkerField -->
      <MarkerField>
         <Name>10</Name>
         <Name>13</Name>
      </MarkerField>
   </FieldMappingInfo>
</AVCSVCollectorTemplate>
```

Related Topics

- Audit Vault Server Fields
  You can map Oracle Audit Vault and Database Firewall events and fields in your collection plug-ins.

C.4.2 CSV File Collection Plug-In Manifest File

See a CSV file collection plug-in manifest file example used with CSV file collection in Oracle Audit Vault and Database Firewall.

This is a sample manifest file for an CSV file collection plug-in.

Example C-8 Sample Manifest File for a CSV File Collection Plug-in

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin xmlns="http://xmlns.oracle.com/av/plugin" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://
xmlns.oracle.com/av/plugin plugin-manifest.xsd" name="Oracle-XML-Template"
id="com.oracle.av.plugin" version="1.0" provider-name="Oracle Corp."
copyright="Copyright Oracle Corp. 2011">
   <!-- targetVersion: Version of Oracle Audit Vault supported by
   this plugin. This is represented by the "min" attribute of
   targetVersion> tag -->
   <targetVersion min="20.4.0.0.0" />
   <extensionSet>
      <extensionPoint type="securedTargetType">
         <!-- fileList: Lists *all* the files that ship with the plugin -->
         <fileList>
            <jars />
            <templates>
               <include file="CSVSource-Mapper.xml" />
```

```
            </templates>
            <bin />
            <config />
            <shell />
            <patch />
            <unresolved-external />
         </fileList>
         <!-- securedTargetTypeInfo: Contains source type and trail
information-->
         <securedTargetTypeInfo name="csv_file_secured_target" />
         <!-- trailType: contains trail type, location , classname for
         source type testSource -->
         <trailInfo>
            <trailType>DIRECTORY</trailType>
            <className
name="oracle.av.platform.agent.collfwk.ezcollector.csv.GenericCSVFileCo
llectorFactory" />
         </trailInfo>
      </extensionPoint>
   </extensionSet>
</plugin>
```

C.5 JSON REST Collection Plug-in Example

Learn about the JSON REST plug-in mapper file and plug-in manifest file attributes and fields for Oracle Audit Vault and Database Firewall.

C.5.1 JSON REST Collection Plug-In Mapper File

See a JSON REST template collector file example, and find out about the attributes and fields used with JSON file collection in Oracle Audit Vault and Database Firewall.

There are both mandatory attributes and fields, and fields that are not mandatory, but that can cause your audit records to be treated as invalid.

Mandatory Attributes and Fields for JSON REST Collection Plug-In Mapper Files

- `securedTargetType`
- `maxSecuredTargetVersion`
- `version`
- `HeaderInfo`
- `RecordInfo`
- `EventTimeUTC`
- `CommandClass` transformations
- `EventStatus` transformations
- `MarkerField`
- `QueryFormat`
- `TimeFormat`

- NextLinkStartTag
- NextLinkPattern
- RESTAuthentication
- BasicAuth

Standard Fields for JSON REST Collection Plug-In Mapper Files

Source names that map to these Oracle Audit Vault Server fields are not mandatory. However, if the information specified by these fields is not provided, then when data collection starts, all audit records are treated as invalid:

- UserName
- CommandClass

Example C-9 Sample JSON REST Collection Plug-in Mapper File

```
<?xml version="1.0" encoding="UTF-8"?>
<AVJSONCollectorTemplate securedTargetType="JSONSOURCE"
maxSecuredTargetVersion="11.0" version="1.0">
   <!--Example Template for JSON template collector-->
   <!-- Attributes: "securedTargetType", "maxSecuredTargetVersion" and
   "version" are mandatory attributes, "minSecuredTargetVersion"
   attribute is optional -->
   <!-- Accepted Format for min/maxSecuredTargetVersion and version
   attribute value is numbers separated by dots (For example:
   12.2,10.3.2, 11.2.3.0 etc..)-->

   <!-- REST url corresponding to the the specific audit trail -->
   <ResourceName>/audit_events/get_events/</ResourceName>
   <!-- Header Information like JSON Header start tag -->
   <HeaderInfo>
      <!-- Mandatory: HeaderInfo-->
      <!-- Value in this tag gives Root tag of the JSON audit file-->
      <StartTag>ITEMS</StartTag>
   </HeaderInfo>
   <!-- Record Information like Record Start tag and conformation to hold
   original record -->
   <RecordInfo>
      <!-- Mandatory: RecordInfo -->
      <!-- Provides starting tag of audit record in JSON audit file -->
      <StartTag>SESSION_ID</StartTag>
   </RecordInfo>
   <!-- Details of the REST Service -->
   <ServiceDetails>
      <!-- Query format for providing the start time and end time query
parameters -->
      <QueryFormat>{startTime}/{endTime}</QueryFormat>
      <!-- Timestamp format for start time and end time -->
      <TimeFormat>yyyy-MM-dd hh:mm:ss.SSS</TimeFormat>
      <NextLink>
         <!-- Next link start tag -->
         <NextLinkStartTag>next</NextLinkStartTag>
         <!-- Next link pattern -->
```

```
            <NextLinkPattern>$.next.$ref</NextLinkPattern>
         </NextLink>
         <!-- Authentication mechanism for REST Service -->
         <RESTAuthentication>
            <!-- Username and password based Basic Authentication -->
            <BasicAuth/>
         </RESTAuthentication>
      </ServiceDetails>
      <!-- Gives Mapping Information of Source Fields to various AV Fields
      (core and large fields) -->
      <!-- Not Allowed: many-to-one mapping from source field to
      AV Server fields -->
      <FieldMappingInfo>
         <!-- Mapping of Source Fields to Core Fields of AV server
         Source fields specified in core field mappings must be of SQL
         Datatype: String OR convertible to String -->
         <CoreFields>
            <Map>
               <Name>$.EVENT_TIME</Name>
               <MapTo>EventTimeUTC</MapTo>
               <TimestampPattern>yyyy-MM-dd HH:mm:ss.SSS</
TimestampPattern>
            </Map>
            <Map>
               <!-- If UserName core field mapping is not provided, Audit
Data
               Collection still starts successfully, but every audit
record
               will be treated as invalid -->
               <Name>$.USER_ID</Name>
               <MapTo>UserName</MapTo>
            </Map>
            <Map>
               <Name>$.OS_USER_ID</Name>
               <MapTo>OSUserName</MapTo>
            </Map>
            <Map>
               <!-- If source name, the ACTION field, for CommandClass
               core field mapping is not provided, Audit Data Collection
               still starts successfully, but all audit records are
treated
               as invalid -->
               <Name>$.ACTION</Name>
               <MapTo>CommandClass</MapTo>
               <!-- Mandatory: value transformations from source to Action
               field value. Value of "to" Attribute is from AV Event set
-->
               <Transformation>
                  <ValueTransformation from="1" to="CREATE" />
                  <ValueTransformation from="2" to="INSERT" />
                  <ValueTransformation from="3" to="SELECT" />
                  <ValueTransformation from="4" to="CREATE" />
                  <ValueTransformation from="15" to="READ" />
                  <ValueTransformation from="30" to="LOGON" />
                  <ValueTransformation from="34" to="LOGOFF" />
```

```
                  <ValueTransformation from="35" to="ACQUIRE" />
               </Transformation>
            </Map>
            <Map>
               <Name>$.OBJ_NAME</Name>
               <MapTo>TargetObject</MapTo>
            </Map>
            <Map>
               <Name>$.USER_HOST</Name>
               <MapTo>ClientHostName</MapTo>
            </Map>
            <Map>
               <Name>$.OBJ_CREATOR</Name>
               <MapTo>TargetOwner</MapTo>
            </Map>
            <Map>
               <Name>$.STATUS</Name>
               <MapTo>EventStatus</MapTo>
               <!-- Specifying value transformation for Status source field
value.
               Mandatory: EventStatus value transformation.
               There are three possible values for EventStatus:
               SUCCESS, FAILURE, UNKNOWN -->
               <Transformation>
                  <ValueTransformation from="0" to="FAILURE" />
                  <ValueTransformation from="1" to="SUCCESS" />
                  <ValueTransformation from="2" to="UNKNOWN" />
               </Transformation>
            </Map>
         </CoreFields>
         <!-- Mapping of Source Fields to Large Fields of AV server i.e fields
         with huge content -->
         <!-- Source fields specified in large field mappings must be of SQL
         Datatype:CLOB OR SQL Datatype:String OR convertible to String -->
         <LargeFields>
            <Map>
               <Name>$.SQL_TEXT</Name>
               <MapTo>CommandText</MapTo>
            </Map>
            <Map>
               <Name>$.COMMAND_PARAMETER</Name>
               <MapTo>CommandParam</MapTo>
            </Map>
         </LargeFields>
         <!-- These Source fields will be collected in a single extension
         field, all name-value pairs are separated by standard delimiter -->
         <!-- Source fields specified in extension field mapping must be of
         SQL Datatype:String OR convertible to String -->
         <ExtensionField>
            <Name>$.DB_ID</Name>
            <Name>$.INSTANCE</Name>
            <Name>$.PROCESS</Name>
            <Name>$.TERMINAL</Name>
         </ExtensionField>
         <!-- This is group of source fields for uniquely identifying each
```

```
      Audit Record Marker -->
      <!-- Source fields specified to be used as Marker field mapping
must
      be of SQL Datatype:String OR convertible to String -->
      <!-- Mandatory: Source fields for MarkerField -->
      <MarkerField>
         <Name>$.SESSION_ID</Name>
         <Name>$.ENTRY_ID</Name>
      </MarkerField>
   </FieldMappingInfo>
</AVJSONCollectorTemplate>
```

Related Topics

- Audit Vault Server Fields
  You can map Oracle Audit Vault and Database Firewall events and fields in your collection plug-ins.

C.5.2 JSON REST Collection Plug-In Manifest File

See a JSON REST collection plug-in manifest file example used with JSON file collection in Oracle Audit Vault and Database Firewall.

This is a sample manifest file for an JSON REST collection plug-in.

Example C-10 Sample Manifest File for a JSON REST Collection Plug-in

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin xmlns="http://foobar.example.com/av/plugin" xmlns:xsi="http://
foobar.example.org/2001/XMLSchema-instance" xsi:schemaLocation="http://
xmlns.oracle.com/av/plugin plugin-manifest.xsd" name="Oracle-XML-
Template" id="com.oracle.av.plugin" version="1.0" provider-
name="Oracle Corp." copyright="Copyright Oracle Corp. 2011">
   <!-- targetVersion: Version of Oracle Audit Vault supported by
   this plugin. This is represented by the "min" attribute of
   targetVersion> tag -->
   <targetVersion min="11.1.0.0.0" />
   <extensionSet>
      <extensionPoint type="securedTargetType">
         <!-- fileList: Lists *all* the files that ship with the
plugin -->
         <fileList>
            <jars />
            <templates>
               <include file="RESTJSONSource-Mapper.xml" />
            </templates>
            <bin />
            <config />
            <shell />
            <patch />
            <unresolved-external />
         </fileList>
         <!-- securedTargetTypeInfo: Contains source type and trail
information-->
```

```
         <securedTargetTypeInfo name="json_rest_secured_target" />
         <!-- trailType: contains trail type, location , classname for
         source type testSource -->
         <trailInfo>
            <trailType>REST</trailType>
            <className
name="oracle.av.platform.agent.collfwk.ezcollector.json.JSONRESTCollectorFact
ory" />
         </trailInfo>
         <!-- eventPatch: OPTIONAL field that indicates any event patches
         that need to be applied as part of plugin deployment
         The files listed here must be present in the patch
         tag entries. The order in which the patches need to
         applied can be controlled via the "order" attribute
         Patches with lower "order" value will be applied first -->
         <eventPatch name="p6753288_11.1.2.0.0_GENERIC.zip" order="2" />
      </extensionPoint>
   </extensionSet>
</plugin>
```

C.6 Java-Based Collection Plug-in Example

Learn about the Java plug-in code and Java-based collection plug-in manifest file packages and structure for Oracle Audit Vault and Database Firewall.

C.6.1 Java Collection Plug-in Code

This examples shows a complete Java-based collection plug-in.

This example is the end result of the discussion, "How to Create a Java-Based Collection Plug-in".

Example C-11 SampleEventCollectorFactory.java

```
 package oracle.av.plugin.sample.collector;

import oracle.av.platform.agent.collfwk.AuditEventCollector;
import oracle.av.platform.agent.collfwk.AuditEventCollectorException;
import oracle.av.platform.agent.collfwk.AuditEventCollectorFactory;
import oracle.av.platform.agent.collfwk.CollectorContext;

public class SampleEventCollectorFactory implements
AuditEventCollectorFactory {

   public AuditEventCollector createAuditCollection(
         CollectorContext collectorContext) throws
AuditEventCollectorException {
      // It simply creates and returns an instance of SampleEventCollector
      return new SampleEventCollector();
   }

}
```

Example C-12 SampleEventCollector.java

```
package oracle.av.plugin.sample.collector;

import java.io.Reader;
import java.sql.Clob;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Timestamp;
import java.util.HashMap;
import java.util.Map;


import oracle.av.platform.agent.collfwk.AuditEventCollector;
import oracle.av.platform.agent.collfwk.AuditEventCollectorException;
import oracle.av.platform.agent.collfwk.AuditService;
import oracle.av.platform.agent.collfwk.CollectorContext;
import oracle.av.platform.agent.collfwk.SetAttributeException;
import oracle.av.platform.common.ErrorCodes;
import oracle.av.platform.common.dao.ConnectionManager;
import oracle.av.platform.common.dao.ConnectionManagerImpl;
import oracle.av.platform.common.exception.AuditException;
import oracle.av.platform.common.util.AVLogger;

/**
 * This collector collects events from AUD table and sends them to
Collection
 * Framework. It connects to the Source database during initialization
and uses
 * the same connection till close() is called. It maintains one
ResultSet
 * containing events. Once the ResultSet gets exhausted, the Collector
sets a
 * checkpoint and creates another ResultSet.
 *
 * @author myellu
 *
 */
public class SampleEventCollector extends AuditEventCollector {

   // The delay used when querying events.
   private static final long DELAY = 5 * 1000;

   private static final Map<Integer, String> eventNameMap =
               new HashMap<Integer, String>();
   static {
      eventNameMap.put(1, "CREATE");
      eventNameMap.put(2, "INSERT");
      eventNameMap.put(3, "SELECT");
      eventNameMap.put(4, "CREATE");
      eventNameMap.put(15, "ALTER");
      eventNameMap.put(30, "AUDIT");
      eventNameMap.put(34, "CREATE");
```

```
      eventNameMap.put(35, "ALTER");
      eventNameMap.put(51, "CREATE");
      eventNameMap.put(52, "CREATE");
   }

   // This map contains mapping from the source event ids to Audit Vault
target
   // types.


private static final Map<Integer, String> targetTypeMap = new
HashMap<Integer,
    String>();

   static {
      targetTypeMap.put(1, "TABLE");
      targetTypeMap.put(2, "TABLE");
      targetTypeMap.put(3, "TABLE");
      targetTypeMap.put(4, "CLUSTER");
      targetTypeMap.put(15, "TABLE");
      targetTypeMap.put(30, "OBJECT");
      targetTypeMap.put(34, "DATABASE");
      targetTypeMap.put(35, "DATABASE");
      targetTypeMap.put(51, "USER");
      targetTypeMap.put(52, "ROLE");
   }

   // This map contains mapping from the source event ids to Source Event
Names.
   // This is necessary since source event ids do not describe the Source
Event.
   private static final Map<Integer, String> sourceEventMap = new
HashMap<Integer,
      String>();

   static {
      targetTypeMap.put(1, "OBJECT:CREATED:TABLE");
      targetTypeMap.put(2, "INSERT INTO TABLE");
      targetTypeMap.put(3, "SELECT FROM TABLE");
      targetTypeMap.put(4, "OBJECT:CREATED:TABLE");
      targetTypeMap.put(15, "OBJECT:ALTERED:TABLE");
      targetTypeMap.put(30, "AUDIT OBJECT");
      targetTypeMap.put(34, "OBJECT:CREATED:DATABASE");
      targetTypeMap.put(35, "OBJECT:ALTERED:DATABASE");
      targetTypeMap.put(51, "OBJECT:CREATED:USER");
      targetTypeMap.put(52, "OBJECT:CREATED:ROLE");
   }


   // holds a connection to the Source database.
   private ConnectionManager m_connectionManager;

   // Connection to the Source.
   private Connection m_connection;
```

```
   // PreparedStatement used to get ResultSet.
   private PreparedStatement m_preparedStatement;


   // holds the ResultSet containing records.
   private ResultSet m_resultSet;

   // AuditService will be used to set checkpoint.
   private AuditService m_auditService;

   // previous checkpoint set.
   private Timestamp m_previousCheckpoint;

   // next checkpoint to be set.
   private Timestamp m_nextCheckpoint;

   private AVLogger m_logger;

   // The CollectorContext received from the Collection Framework.
   private CollectorContext m_collectorContext;

   private long m_timeZoneOffset;

   /**
    * It connects to the database using the credentials and Connection
String
    * from the CollectorContext.
    *
    * @throws AuditEventCollectorException
    */
   private void connectToSource() throws AuditEventCollectorException {
      m_logger.logDebugMethodEntered();
      // Get connection information from collector context.
      String user = m_collectorContext.getSecuredTargetUser();
      String password = new
String(m_collectorContext.getSecuredTargetPassword());
      String connectionString =
m_collectorContext.getSecuredTargetLocation();
      // Create a ConnectionManager object.
      try {
         m_connectionManager = new
ConnectionManagerImpl(connectionString,
               user, password.toCharArray());
         m_connection = m_connectionManager.getConnection();
      } catch (AuditException ex) {
         throw new AuditEventCollectorException(
               ErrorCodes.FAILED_CONNECT_TO_SOURCE,
               new Object[] { connectionString }, ex);
      }
      m_logger.logDebugMethodExited();
   }

/**
    * converts the timone offset specified in String to a number of
    * milliseconds.
```

```
     *
     */
    private long getTimeZoneOffsetInMs(String offset) {
       if (offset == null)
          return 0;
       long timeZoneOffset;
       /** process offset to get value in milliseconds */
       int hour = Integer.parseInt(offset.substring(1, 3));
       int min = Integer.parseInt(offset.substring(4, 6));
       timeZoneOffset = (hour * 60 * 60 + min * 60) * 1000;
       if (offset.charAt(0) == '-')
          timeZoneOffset *= -1;
       return timeZoneOffset;
    }

    /**
     * Initializes the Collector with the values from CollectorContext. It
also
     * connects to the database.
     */
    public void initializeCollector(CollectorContext collectorContext)
          throws AuditEventCollectorException {
       m_collectorContext = collectorContext;
       m_auditService = m_collectorContext.getAuditService();
       m_previousCheckpoint = m_collectorContext.getCheckpoint();
       m_logger = m_collectorContext.getLogger();
       // Get the timone offset for the Source.
       String offset = m_collectorContext.getAttribute("TimeZoneOffset");
       if (offset != null) {
          m_timeZoneOffset = getTimeZoneOffsetInMs(offset);
      }
       connectToSource();
       fetchEvents();
    }

    /**
     * Queries the Source to get audit events that occurred from previous
     * checkpoint to the current time. Apart from during the initialization,
this
     * method should be called only when ResultSet is exhausted. There are two
     * reasons for this. <br>
     * 1. This method will set the checkpoint. Checkpoint should only be set
when
     * the ResultSet is exhausted as the results with in the ResultSet can be
in
     * random order.<br>
     * 2. This method will create a new ResultSet. Hence the contents of the
old
     * ResultSet will be inaccessible after this function is called.
     *
     * @throws AuditEventCollectorException
     */
    private void fetchEvents() throws AuditEventCollectorException {
       m_logger.logDebugMethodEntered();
       if (m_nextCheckpoint != null) {
```

```
            m_auditService.setCheckpoint(m_nextCheckpoint);
            m_previousCheckpoint = m_nextCheckpoint;
         }

        // It is not good to hold on to the Connection for long. As this
is the
         // only place we can release the connection, we release and
reacquire the
         // connection.
         try {
            if (m_connection != null) {
               m_connectionManager.releaseConnection(m_connection);
            }
         } catch (AuditException ex) {
            throw new AuditEventCollectorException(
                  ErrorCodes.FAILED_TO_RELEASE_CONNECTION_TO_DB, null,
ex);
         }

         try {
            m_connection = m_connectionManager.getConnection();
         } catch (AuditException ex) {
            throw new AuditEventCollectorException(
                  ErrorCodes.FAILED_TO_GET_CONNECTION_TO_DB, null, ex);
         }

         // Now we need to aim for the next checkpoint. We will query for
all
         // events from previous checkpoint to the next checkpoint. So we
want to
         // make sure that all the events with event time lesser than the
next
         // checkpoint are already available in the table. However, the
events
         // might take a small amount of time before they are present in
the table.
         // Hence the next checkpoint we aim will be current time minus
delta time.
         m_nextCheckpoint = new Timestamp(System.currentTimeMillis() -
DELAY);
         String query = null;
         try {
            if (m_previousCheckpoint == null) {
               query = "select * from AUD where EVENT_TIME <= ?";
               m_preparedStatement = m_connection.prepareStatement(query);
               m_preparedStatement.setTimestamp(1, m_nextCheckpoint);
            } else {
               query = "select * from AUD where EVENT_TIME > ? and
EVENT_TIME <= ?";
               m_preparedStatement = m_connection.prepareStatement(query);
               m_preparedStatement.setTimestamp(1, m_previousCheckpoint);
               m_preparedStatement.setTimestamp(2, m_nextCheckpoint);
            }
            m_resultSet = m_preparedStatement.executeQuery();
         } catch (SQLException ex) {
```

```
            throw new AuditEventCollectorException(
                  ErrorCodes.ERROR_GETTING_DATA_FROM_SOURCE,
                  new Object[] { query }, ex);
      }
      m_logger.logDebugMethodExited();
   }

   /**
    * If the result set is not exhausted this will return true. If it has
    * exhausted, it will query to get the events till the current time. If it
    * could get any events, it will return true, false otherwise.
    */
   public boolean hasNext() throws AuditEventCollectorException {
      boolean hasMore;
      try {
         if(m_resultSet == null) {
            fetchEvents();
            return m_resultSet.next();
         }
         hasMore = m_resultSet.next();
         if (!hasMore) {
            fetchEvents();
            hasMore = m_resultSet.next();
         }
      } catch (SQLException ex) {
         throw new AuditEventCollectorException(
               ErrorCodes.ERROR_GETTING_DATA_FROM_SOURCE, null, ex);
      }
      return hasMore;
   }

   // All the getter methods make use of the ResultSet get methods and return
   // the value appropriately.

   public String getUserName() throws AuditEventCollectorException {
      try {
         return m_resultSet.getString("USER_ID");
      } catch (SQLException ex) {
         throw new AuditEventCollectorException(
               ErrorCodes.ERROR_GETTING_DATA_FROM_SOURCE, null, ex);
      }
   }

   public String getOSUserName() throws AuditEventCollectorException {
      try {
         return m_resultSet.getString("OS_USER_ID");
      } catch (SQLException ex) {
         throw new AuditEventCollectorException(
               ErrorCodes.ERROR_GETTING_DATA_FROM_SOURCE, null, ex);
      }
   }

   public String getCommandClass() throws AuditEventCollectorException {
      try {
         int eventId = m_resultSet.getInt("ACTION");
```

```
            return eventNameMap.get(eventId);
         } catch (SQLException ex) {
           throw new AuditEventCollectorException(
                  ErrorCodes.ERROR_GETTING_DATA_FROM_SOURCE, null, ex);
         }
      }

      public String getEventName() throws AuditEventCollectorException {
         try {
            int eventId = m_resultSet.getInt("ACTION");
            return sourceEventMap.get(eventId);
         } catch (SQLException ex) {
            throw new AuditEventCollectorException(
                  ErrorCodes.ERROR_GETTING_DATA_FROM_SOURCE, null, ex);
         }
      }

      public EventStatus getEventStatus() throws
   AuditEventCollectorException {
         try {
            int status = m_resultSet.getInt("STATUS");
            if (status == 1) {
               return EventStatus.SUCCESS;
            } else if (status == 0) {
               return EventStatus.FAILURE;
            } else {
               return EventStatus.UNKNOWN;
            }
         } catch (SQLException ex) {
            throw new AuditEventCollectorException(
                  ErrorCodes.ERROR_GETTING_DATA_FROM_SOURCE, null, ex);
         }
      }

      public Timestamp getEventTimeUTC() throws
   AuditEventCollectorException {
         try {
            Timestamp eventTime = m_resultSet.getTimestamp("EVENT_TIME");
            // As the method name suggests, the timestamp must be
   returned only in
            // UTC timone.
            return new Timestamp(eventTime.getTime() - m_timeZoneOffset);
         } catch (SQLException ex) {
            throw new
   AuditEventCollectorException(               ErrorCodes.ERROR_GETTING_DA
   TA_FROM_SOURCE, null, ex);
         }
      }

      public String getErrorMessage() throws AuditEventCollectorException
   {
         // There is no corresponding field for ErrorMessage. Hence we
         // return NULL always.
         return null;
      }
```

```
   public String getErrorId() throws AuditEventCollectorException {
      // There is no corresponding field for ErrorId. Hence we
      // return NULL always.
      return null;
   }

   public String getTargetObject() throws AuditEventCollectorException {
      try {
         return m_resultSet.getString("OBJ_NAME");
      } catch (SQLException ex) {
         throw new
AuditEventCollectorException(              ErrorCodes.ERROR_GETTING_DATA_FRO
M_SOURCE, null, ex);
      }
   }

   public String getTargetType() throws AuditEventCollectorException {
       try {
         int eventId = m_resultSet.getInt("ACTION");
         return targetTypeMap.get(eventId);
      } catch (SQLException ex) {
         throw new AuditEventCollectorException(
               ErrorCodes.ERROR_GETTING_DATA_FROM_SOURCE, null, ex);
      }
}

   public String getTargetOwner() throws AuditEventCollectorException {
      try {
         return m_resultSet.getString("OBJ_CREATOR");
      } catch (SQLException ex) {
         throw new AuditEventCollectorException(
               ErrorCodes.ERROR_GETTING_DATA_FROM_SOURCE, null, ex);
      }
   }

   public String getClientHostName() throws AuditEventCollectorException {
      try {
         return m_resultSet.getString("USER_HOST");
      } catch (SQLException ex) {
         throw new AuditEventCollectorException(
               ErrorCodes.ERROR_GETTING_DATA_FROM_SOURCE, null, ex);
      }
   }

   public String getClientIP() throws AuditEventCollectorException {
      // There is no corresponding field for IP address. Hence we
      // return NULL always.
      return null;
   }

   public String getExtension() throws AuditEventCollectorException {
      try {
         StringBuilder sb = new StringBuilder();
         // Here we will put those fields which are not sent in other getter
```

```
            // methods.
            sb.append("DB_ID=" + m_resultSet.getString("DB_ID") + ";");
            sb.append("INSTANCE=" + m_resultSet.getString("INSTANCE") +
";");
            sb.append("PROCESS=" + m_resultSet.getString("PROCESS"));
            return sb.toString();
         } catch (SQLException ex) {
            throw new AuditEventCollectorException(
                  ErrorCodes.ERROR_GETTING_DATA_FROM_SOURCE, null, ex);
         }
      }

      public Reader getCommandText() throws AuditEventCollectorException {
         try {
            // Clobs and the Readers contained in the Clobs are alive only
            // as long as the Connection to the Source is alive. So if
the Source
            // Connection is closed, Collection Framework will fail when
it tries
            // to send the events to AV Server. If there is any need to
close and
            // recreate a connection that should be done immediately
after setting
            // the checkpoint. Setting the checkpoint causes the
Collection
            // Framework to flush all the events it is holding. So
immediately
            // after setting the checkpoint, we are sure that the
Framework is not
            // holding any events.
            Clob clob = m_resultSet.getClob("SQL_TEXT");
            return clob.getCharacterStream();
         } catch (SQLException ex) {
            throw new AuditEventCollectorException(
                  ErrorCodes.ERROR_GETTING_DATA_FROM_SOURCE, null, ex);
         }
      }

      public Reader getCommandParam() throws AuditEventCollectorException
{
         try {
            Clob clob = m_resultSet.getClob("SQL_BIND");
            return clob.getCharacterStream();
         } catch (SQLException ex) {
            throw new AuditEventCollectorException(
                  ErrorCodes.ERROR_GETTING_DATA_FROM_SOURCE, null, ex);
         }
      }

      public String getMarker() throws AuditEventCollectorException {
         // ENTRY_ID will identify an audit event uniquely with in a
session. Hence
         // ENTRY_ID along with SESSION_ID will uniquely identify an
audit event
         // across sessions.
```

```
      try {
         return m_resultSet.getString("SESSION_ID") + ":"
               + m_resultSet.getString("ENTRY_ID");
      } catch (SQLException ex) {
         throw new AuditEventCollectorException(
               ErrorCodes.ERROR_GETTING_DATA_FROM_SOURCE, null, ex);
      }
   }

   public void setAttribute(String name, String value)
         throws SetAttributeException {
      if (name.equalsIgnoreCase("TimeZoneOffset")) {
               m_timeZoneOffset = getTimeZoneOffsetInMs(value);
      } else {
         throw new SetAttributeException(ErrorCodes.INVALID_ATTRIBUTE_NAME,
               new Object[] { name, value }, null);
      }
   }

   public void close() {
      try {
         if (m_resultSet != null) {
            m_resultSet.close();
            m_resultSet = null;
         }
         if (m_connectionManager != null) {
            m_connectionManager.destroy();
            m_connectionManager = null;
         }
         m_previousCheckpoint = null;
         m_nextCheckpoint = null;
         m_logger = null;
      } catch (SQLException ex) {
         m_logger.logError("SampleEventCollector", "close",
               "SQLException occurred. ", ex);
      } catch (AuditException ex) {
         m_logger.logError("SampleEventCollector", "close",
               "AuditException occurred. ", ex);
      }
   }
}
```

C.6.2 Java Based Collection Plug-in Manifest File

See how to set up a Java-based collection plug-in for Oracle Audit Vault and Database Firewall.

This is a sample manifest file for a Java-based collection.

Example C-13 Java-Based Manifest File

```
<?xml version="1.0"?>

<plugin xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://xmlns.oracle.com/av/plugin plugin-manifest.xsd"
        xmlns="http://xmlns.oracle.com/av/plugin"
```

```
        name="Sample Plugin"
        id="com.oracle.av.plugin.sample"
        version="12.1.0.0.0"
        provider-name="Oracle Corp."
        copyright="Copyright Oracle Corp. 2011">

            <!-- targetVersion: Version of Oracle Audit Vault supported by
              this plugin. This is represented by the "min" attribute of
              targetVersion> tag                -->

   <targetVersion min="12.1.0.0.0"/>

   <extensionSet>
      <extensionPoint type= "securedTargetType">
            <!-- fileList: Lists *all* the files that ship with the plugin -->
         <fileList>
            <jars>
               <include file="samplecollector.jar"/>
               <!-- All your collector Java jar binaries go here -->
            </jars>
            <templates>
            <bin>
               <!-- All your collector native binaries go here -->
            </bin>
            <config>
               <!-- Any configuration information (such as .properites files)
                    go here -->
            </config>
            <shell>
               <!-- Any shell scripts that your collector relies on go here -->
            </shell>
            <patch>
               <!-- Oracle provided patches go here -->
            </patch>
            <unresolved-external>
               <!-- Any files belonging to the unresolved-external category here
-->
            </unresolved-external>

         </fileList>
            <!-- securedTargetTypeInfo: Contains source type and trail
information
                 -->
         <securedTargetTypeInfo name="Sample"/>
            <!-- trailType: contains trail type, location , classname for
               source type testSource -->
         <trailInfo>
           <trailType>TABLE</trailType>
            <className
name="oracle.av.plugin.sample.collector.SampleEventCollectorFactory" />
         </trailInfo >
      </extensionPoint>
   </extensionSet>
</plugin>
```

D
Bundled JDBC Drivers

Learn about the JDBC drivers that are bundled with the Oracle Audit Vault and Database Firewall (Oracle AVDF) SDK.

D.1 About Bundled JDBC Drivers

Learn about the five JDBC drivers that are bundled with Oracle Audit Vault and Database Firewall.

When you create a collection plug-in, you can use it to extract audit records from a database table. To do this, you must have a JDBC driver to connect to the database. Drivers for most common databases are bundled with the SDK.

The Oracle AVDF SDK ships with 5 different JDBC drivers, some that are standard to the product and some that are proprietary drivers provided by Oracle for specific third-party databases.

- Standard
  - Oracle
  - MySQL
- Proprietary
  - Sybase
  - Microsoft SQL Server
  - DB2

You are not required to use any of these JDBC drivers. You can use drivers that you have acquired elsewhere. However, if you plan to use any of the listed drivers, then in your mapper file, and when registering a target, you must provide the information in the following table:

Table D-1 JDBC Drivers and Connecting URLs

Database	Driver Class	Connecting URL
Oracle	`oracle.jdbc.pool.OracleDataSource`	`jdbc:oracle:thin:@host:port:sid`
MySQL	N/A	`jdbc:av:mysql://host:port`
SQLServer	`oracle.av.platform.jdbcx.sqlserver.SQLServerDataSource`	`jdbc:av:sqlserver://host:port`
DB2	`oracle.av.platform.jdbcx.db2.DB2DataSource`	`jdbc:av:db2://host:port/dbname`
Sybase	`oracle.av.platform.jdbcx.sybase.SybaseDataSource`	`jdbc:av:sybase://host:port`

D.2 Connecting URLs

Use the correct Connection URL for the type of JDBC driver that you use with Oracle Audit Vault and Database Firewall.

You use Connection URLs to specify the location of a database target when you register the target on the GUI or through AVCLI. The format of the Connection URL depends on the JDBC driver that you use. Each of the JDBC drivers shipped with Oracle Audit Vault and Database Firewall specifies the format required by the JDBC driver in question in the table in "About JDBC Drivers and Connecting URLs." .

Additionally, to use specific encryption methods in the connecting URL, you must set the `EncryptionMethod` property. In the following syntax example, note that the variable *`encryptionmethod`* can be `SSL`, `requestSSL`, or `loginSSL`:

```
jdbc:av:[sqlserver]://hostname: port;
[EncryptionMethod=encryptionmethod].
```

Use this url to register a Target, by entering it into the Target Location field, with the Advanced mode selected.

Related Topics

- About Bundled JDBC Drivers
  Learn about the five JDBC drivers that are bundled with Oracle Audit Vault and Database Firewall.
- Creating a Database Table Mapper File
  Learn how to create an Oracle Audit Vault XML mapper file for a database table collection plug-in, and learn about each XML element and attribute used in this type of mapper file.

> **See Also:**
>
> *Target Connection Information* block in "Creating a Database Table Mapping File. Creating a Database Table Mapper File.

D.3 DataSource Class

To enable a Java application to interact with the Oracle Audit Vault platform, use this JDBC DataSource class.

```
oracle.av.platform.jdbcx.dbsource.DBSourceDataSource
```

Glossary

audit record

A record that represents a database event.

audit record field

A component of an audit record. Each audit record field represents an attribute of the event that the record represents. If the record is in a table, then its fields are columns.

audit trail

A location of audit records on the secured target. For example:

- If the secured target writes audit records into files (called **audit files**), then the directory path plus the file mask is an audit trail.
- If the source writes audit records into a database table (called an **audit table**), then the name of the table is an audit trail.
- If the source writes some audit records into files of directory x, some into database table y, and some into files of directory z, then the source has three different audit trails: directory x plus the file mask, table y, and directory z plus the file mask.

audit trail cleanup

The process that purges audit records from the secured target after they are stored in Audit Vault Server repository. The collection plug-in provides the checkpoint to either the source or a utility that has permission to delete records from the source, and the source or utility purges the original records.

Audit Vault Server field

An audit record field in Oracle Audit Vault and Database Firewall, as opposed to an audit record field on a secured target (see collection plug-in). An Audit Vault Server field is either a core field, an extension field, or a large field.

checkpoint

The point in an audit trail after which a collection plug-in will start collecting audit records. If the collection plug-in has collected no records from the audit trail, then the checkpoint is

immediately before the first record. If the collection plug-in started collecting records and then stopped, then the checkpoint is immediately after the last record that it collected.

collection plug-in

A plug-in that adds an audit trail collection capability to Oracle Audit Vault and Database Firewall. It gets audit record semantics from a mapper file and reads audit records from either an audit table or XML audit files.

Command Text field

A large field that contains the text of the command that caused the event.

Command Parameter field

A large field that contains the parameters of the command that caused the event.

core field

An Audit Vault Server field that has a corresponding field in audit records generated by almost every source. That is, almost every collection plug-in maps a source audit record field to each core field. Oracle Audit Vault and Database Firewall uses core fields for filtering and reporting. The core fields are described and listed in "Core Fields".

extension field

An Audit Vault Server field that is not a core field but must be stored in Oracle Audit Vault Server.

large field

An Audit Vault Server field of the data type `CLOB` (described in *Oracle Database SQL Language Reference*). A large field is either a Command Text field or a Command Parameter field.

mapper file

An XML file that describes the audit records that a specific secured target writes into either an audit table or XML audit files. The mapper file specifies the audit record fields to collect from the source, how to map them to Audit Vault Server fields, and which fields to use for recovery. A mapper file always specifies the secured target, the maximum version of the source type that the mapper file supports, and the mapper file version. A mapper file can also specify the minimum version of the source type that it

supports and an incremental field for calculating the checkpoint. The default for the incremental field is the event time field.

Marker field

An audit record field that uniquely identifies the record within an audit trail. An collection plug-in uses marker fields to avoid collecting duplicate records during recovery.

plug-in

An application that adds a capability to another application (and usually cannot run independently).

recovery

The phase of data collection where an collection plug-in that stopped and restarted tries to reach its checkpoint. Resuming collection immediately after the checkpoint ensures that the collector does not miss any records. To avoid collecting duplicate records during recovery, the collector checks the Marker field of each record.

secured target

A secured target is a supported database or non-database product that you secure using an Audit Vault Agent, a Database Firewall, or both.

secured target type

A category of auditing source. For example, Oracle Database is a secured target type, a collection of Oracle Database instances that generate audit records with the same fields. Secured target types generate semantically identical audit records (that is, audit records that have the same fields).

Index

A

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

W

X