Oracle® Audit Vault and Database Firewall

Developer's Guide

Release 12.1.2

E27779-06

August 2016

ORACLE®

Oracle Audit Vault and Database Firewall Developer's Guide, Release 12.1.2

E27779-06

Primary Author: Karthik Shetty

Contributing Authors: Tulika Das, Gigi Hanna, Janis Greenberg, Sheila Moore

Contributors: Shyam Bellam, Raghav Hanumantharau, Srivatsan Kannan, Sayali Nafde, Sreekumar Seshadri, Vipul Shah, Madhusudhan Reddy Yellu

Contents

Preface vii

Audience vii
Documentation Accessibility vii
Related Documents vii
Conventions viii

1 Introduction

1.1 What Is Oracle Audit Vault and Database Firewall? 1-1
1.1.1 How Oracle Audit Vault Server and Agent Work 1-2
1.2 What Are Audit Collection Plug-ins? 1-2
1.2.1 Types of Audit Collection Plug-ins 1-3
1.2.1.1 Determining Which Audit Collection Plug-in Type to Create 1-3
1.3 Audit Vault Events and Fields 1-3
1.3.1 Core Fields 1-3
1.3.1.1 CommandClass and Target Types 1-4
1.3.2 Other Audit Vault Fields 1-4
1.3.2.1 Large Fields 1-4
1.3.2.2 Extension Field 1-4
1.3.2.3 Marker Field 1-4
1.3.3 Storing Audit Records in Audit Vault 1-4
1.4 The Collection Process 1-5
1.4.1 Flow of Collection: User 1-5
1.4.2 Flow of Control Inside the Audit Collection Plug-in 1-6
1.4.3 Collection Concepts 1-7
1.4.3.1 Collection Thread 1-7
1.4.3.2 Collection Phase 1-7
1.4.3.3 Mapping 1-7
1.4.3.4 Checkpoint of a Trail 1-7
1.4.3.5 Recovery Phase Of Data Collection 1-7
1.4.3.6 Audit Trail Clean-Up 1-8
1.5 General Procedure for Writing Audit Collection Plug-ins 1-8

2 Setting Up Your Development Environment

2.1 Before You Set Up the Development Environment 2-1
2.2 Setting Up the Development Environment 2-1

2.3 Audit Collection Plug-in Directory Structure ... 2-2
2.3.1 General Directory Structure ... 2-2
2.3.2 Audit Collection Plug-in Directory Structure ... 2-3
2.3.3 Staging a plugin-manifest.xml File ... 2-3

3 Audit Collection Plug-ins

3.1 About Audit Collection Plug-ins ... 3-1
3.2 About Mapper Files ... 3-2
3.3 Database Table Collection Plug-ins ... 3-2
3.3.1 Requirements for Database Table Collection Plug-ins ... 3-3
3.3.2 Example Audit Trail for a Database Table Collection Plug-in ... 3-3
3.3.3 Creating a Database Table Mapper File ... 3-4
3.4 XML File Collection Plug-ins ... 3-8
3.4.1 Requirements for XML File Collection Plug-ins ... 3-8
3.4.2 Example Audit Trail for an XML File Collection Plug-in ... 3-9
3.4.3 Creating the XML File Audit Collection Mapper File ... 3-10
3.4.4 XML Transformation for Non-Standard Audit Records ... 3-14
3.4.4.1 Additional Requirement for XML Transformation Using XSL ... 3-14
3.4.4.2 Changes Required to Transform Non-Standard Audit Records ... 3-14
3.4.4.3 Sample Non-Standard XML Audit Data Record ... 3-14
3.4.4.4 Creating an XSL File for Transformation ... 3-15
3.5 Secured Target Collection Attributes ... 3-17
3.6 Pre-Processing Audit Data ... 3-17

4 Packaging Audit Collection Plug-ins

4.1 Flow of Packaging ... 4-1
4.2 External Dependencies ... 4-2
4.3 Creating New Versions of Your Audit Collection Plug-ins ... 4-2
4.4 Description of Plug-in Manifest File ... 4-2
4.5 avpack Tool ... 4-4

5 Testing Audit Collection Plug-ins

5.1 Requirements for Testing Audit Collection Plug-ins ... 5-1
5.2 Typical Audit Collection Plug-in Testing Processes ... 5-1
5.3 Deploying an Oracle Audit Vault Agent ... 5-2
5.4 Redeploying the Oracle Audit Vault Agent ... 5-3

A Audit Vault Server Fields

A.1 AVDF Fields ... A-1
A.1.1 Core Fields ... A-1
A.1.2 Large Fields ... A-2
A.1.3 Marker Field ... A-2
A.1.4 Extension Field ... A-2
A.2 Actions and Target Types ... A-2
A.2.1 Actions ... A-3
A.2.2 Target Types ... A-5

B Schemas

B.1 Sample Schema for a plugin-manifest.xml file B-1
B.2 Database Table Collection Plug-in Mapper File B-4
B.3 Schema for XML File Collection Plug-in Mapper File B-6

C Example Code

C.1 Database Table Collection Plug-in Example C-1
C.1.1 Database Table Collection Plug-in Mapper File C-1
C.1.2 Database Table Collection Plug-in Manifest File C-4
C.2 XML File Collection Plug-in Example C-5
C.2.1 XML File Collection Plug-in Mapper File C-5
C.2.2 XML File Collection Plug-in Manifest file C-8

D Bundled JDBC Drivers

D.1 About Bundled JDBC drivers D-1
D.1.1 Connecting URLs D-2
D.1.2 Driver class D-2

Glossary

Index

List of Tables

3–1 AUD Audit Table Data Fields and Mappings 3-3
3–2 Audit Data Fields in XML Audit Records and Mappings 3-10
D–1 JDBC Drivers and Connecting URLs D-1

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.

This preface contains:

- Audience
- Documentation Accessibility
- Related Documents
- Conventions

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.

Related Documents

For more information about Oracle Audit Vault and Database Firewall, see these documents:

- *Oracle Audit Vault and Database Firewall Administrator's Guide*
- *Oracle Audit Vault and Database Firewall Auditor's Guide*
- *Oracle Audit Vault and Database Firewall Installation Guide*

Oracle Documentation Search Engine

To access the database documentation search engine directly, visit:

http://tahiti.oracle.com/

Oracle Technology Network (OTN)

You can download free release notes, installation documentation, updated versions of this guide, white papers, 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/overview/index.html

For the Oracle Audit Vault and Database Firewal Discussion Forums, visit

http://forums.oracle.com/forums/forum.jspa?forumID=391

Oracle Store

Printed documentation is available for sale in the Oracle Store at:

https://shop.oracle.com

My Oracle Support (formerly Oracle*MetaLink*)

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

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.

1 Introduction

This chapter introduces the Audit Collection Plug-in for Audit Vault and Database Firewall. The chapter begins by briefly describing Oracle Audit Vault and Database Firewall software (AVDF) and then explains the collection plug-in. Later chapters describe how you can create custom collection plug-ins to collect audit information written to database tables, XML files, or other locations.

This chapter contains:

- What Is Oracle Audit Vault and Database Firewall?
- What Are Audit Collection Plug-ins?
- Audit Vault Events and Fields
- The Collection Process
- General Procedure for Writing Audit Collection Plug-ins

1.1 What Is Oracle Audit Vault and Database Firewall?

Oracle Audit Vault and Database Firewall is a software system that collects, consolidates, protects, and analyzes audit data from multiple, distributed, heterogeneous systems. It is comprised of these components:

- Oracle 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.
- Oracle 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.
- Oracle 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 secured 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 secured target systems of various types. Collection plug-ins collect audit data from an audit trail generated by a secured 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 secured target. These collection plug-ins collect data from databases such as Oracle, SQL Server, Sybase ASE, and DB2 as described in *Oracle Audit Vault and Database Firewall Administrator's Guide*.

1.1.1 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 trails are generated by different secured target types such as database tables or XML audit records.

A secured 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:

- Secured Target

  A secured target is a software or hardware system performing a specific function. As part of performing that function, the secured target system generates an audit trail. A secured target is an instance of a secured target type and has specific properties such as connection credentials and trail types.

- Secured Target Type

  A secured target type represents a collection of a particular type of secured target that generates the same type of audit data. Oracle Database, for example, is a secured 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 secured target. Examples of audit trails are:

  - For secured targets that write data into files, the trail is the directory path plus the file mask.
  - For secured targets that write audit data into a database table, the name of the table is the trail for that secured target. `SYS.AUD$` is an example of a database table audit trail in an Oracle database.

1.2 What Are Audit Collection Plug-ins?

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. See "What Is Oracle Audit Vault and Database Firewall?" on page 1-1.

Starting with Oracle Audit Vault and Database Firewall release 12.1.1, developers, as well as third-party vendors, can build custom collection plug-ins. These collection plug-ins can collect audit data from a new **secured target type** or support new audit trails written by a secured target type that AVDF already knows about, and make them available to users.

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

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

This guide describes how you can create these custom collection plug-ins and deploy them into existing Oracle Audit Vault and Database Firewall installations.

1.2.1 Types of Audit Collection Plug-ins

You can create two types of collection plug-ins. The actual type you need to create depends on the properties of the audit trail being collected.

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

1.2.1.1 Determining Which Audit Collection Plug-in Type to Create

This section describes the two types of collection plug-ins and explains the properties that are required to use each type.

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 Files: Stored in XML files based on the Audit Vault XML Audit File format

See Also:

- "Database Table Collection Plug-in Example" on page C-1
- "XML File Collection Plug-in Example" on page C-5

1.3 Audit Vault Events and Fields

Monitoring the activity, the *stream of events*, that occur in a secured 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 secured target system is an **audit record**.

Oracle Audit Vault and Database Firewall release 12.1.1 greatly simplifies the representation of events and event structure as compared to Audit Vault 10.3.

In Oracle Audit Vault and Database Firewall release 12.1.1 the following applies:

- Each secured target logs events as audit events that occur on that secured target. Audit records capture information about audit events.
- Audit records typically have a secured 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.

1.3.1 Core Fields

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.

In Oracle Audit Vault and Database Firewall release 12.1.1, the core fields, which describe the actions that 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, the location, of the action.
- `TargetType`, `TargetOwner`, and `TargetObject` fields: The object type, object owner, or target of the action.

For a complete list of core fields, see core fields listed in Appendix A, Section A.1.1, "Core Fields".

1.3.1.1 CommandClass and Target Types

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

Examples of `CommandClasses` are `Logon`, `Select`, `Update`, and `Shutdown`.

These are listed in Appendix A, Section A.2, "Actions and Target Types".

1.3.2 Other Audit Vault Fields

In addition to core fields, Audit Vault Server understands these categories:

- Large Fields
- Marker Field
- Extension Field

1.3.2.1 Large Fields

Large fields are fields that can contain arbitrarily large amounts of data. See Section A.1.2, "Large Fields" for further information.

1.3.2.2 Extension Field

The Extension field provides a way to make available secured target fields that do not have a semantically equivalent Audit Vault field and do not map to Core or Large fields.

The developer determines the format used to store the fields.

See Section A.1.4, "Extension Field" for further information.

1.3.2.3 Marker Field

The Marker field is a unique identifier of a record in a trail. It is constructed out of one or more fields in an audit record.

See Section A.1.3, "Marker Field" for further information.

1.3.3 Storing Audit Records in Audit Vault

As a plug-in developer, you must map the various events that occur within secured targets, and their fields, to the various fields allowed by Audit Vault.

If a field in the audit record maps to one of the named fields (core, large, or marker fields) in Audit Vault, you should map it as such.

If a field in the audit record does not map to one of the named fields, you can map it to an extension field of your choosing.

For the `Action` and `TargetType` Audit Vault Server fields, see the list of field values in the "Actions and Target Types" on page A-2. If your audit record maps to one of these values semantically, we strongly encourage you to use that value. You are, however, free to use values not listed in the appendix.

You are strongly encouraged to follow these basic guidelines when you store values in Audit Vault.

- Do not store IDs that reference objects in the secured target database. Audit Vault does not have access to these objects, and the stored values will be meaningless. Store literal names of objects instead, so they can be understood by the auditors,
- Follow defined Audit Vault conventions. For example, all the `ACTION` fields and `TARGET TYPE` fields in Audit Vault have uppercase values. Please follow this convention, unless this is not applicable to your secured target type, and would cause the data stored in Audit Vault to be interpreted incorrectly,
- Map to the values documented in "Actions and Target Types" on page A-2 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 a field in the audit record of a secured target merits becoming a core field, please contact Oracle so that it can be added to the model appropriately.

1.4 The Collection Process

This section covers the following topics:

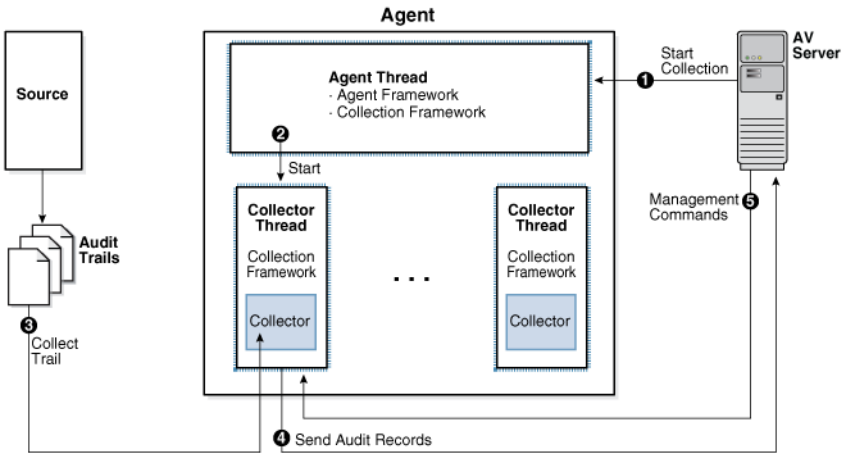
- Flow of Collection: User
- Flow of Control Inside the Audit Collection Plug-in
- Collection Concepts

1.4.1 Flow of Collection: User

1. You, the developer, create a collection plug-in and provide it to the user.
2. The user deploys the plug-in into the Audit Vault Server. The act of deploying a plug-in into the server creates a new version of the Audit Vault Agent. This new agent contains collector code from the collection plug-in.
3. The user then deploys the new Agent onto the host where it needs to run.
4. From then on, the user can start collecting audit trails supported by the collector code.
5. User starts collecting audit trails supported by the collector code.

The next section describes what happens inside the collection plug-in after the user starts collection.

Figure 1–1 Flow of Collection



1.4.2 Flow of Control Inside the Audit Collection Plug-in

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

This section describes this process in detail.

1. The Audit Vault Server commands the Agent Framework to create a thread to collect a specific audit trail. See "Collection Thread" on page 1-7l
2. The new thread, just created by the agent, collects a specific audit trail.

   At this point control is handed to the Collection Framework. See "Collection Phase" on page 1-7.
3. Within the thread, the Collection Framework connects to the 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. See "Checkpoint of a Trail" on page 1-7.
4. With the information it now has, the Collection Framework determines, through the plug-in manifest file, the correct Java class to invoke within the correct collection plug-in. It passes the configuration information to this class, and asks it to initialize itself.
5. Once 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 that Audit Vault Server expects, and hands them to the Collection Framework through the Collection API. See "Mapping" on page 1-7
   - Sends the Audit Vault Server any checkpoint information and other metric data received from the collection plug-in.

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 Audit Vault Server, it notifies the collector to stop sending records, then exits the collection thread, shuts itself down.

1.4.3 Collection Concepts

This section explains some of the concepts related to the collection process.

1.4.3.1 Collection Thread

The agent starts threads of collection. 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 which the collection plug-in interfaces with. The collection plug-in also uses utility APIS (not shown) if it needs to.

1.4.3.2 Collection Phase

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 depend on the audit trail. Once a secured target audit record is retrieved from the trail, the collection plug-in must be able to transform (map) it into an audit record hat can be sent to the Audit Vault Server. See "Mapping" on page 1-7.

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

1.4.3.3 Mapping

The mappings required depend on the secured target records.

These types of mapping are required:

- Event Mapping: Maps a secured target-specific event to an Audit Vault Event.
- Field Mapping: Maps the various field of the secured target records to Audit Vault 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 secured target field to an Audit Vault Field, or one secured target audit event to one Audit Vault Event.

1.4.3.4 Checkpoint of a Trail

The **checkpoint** or the checkpoint of a trail is the point (as a timestamp) up to which the audit records were committed to the Audit Vault Server. The collection plug-in sets a checkpoint periodically so that it can resume from the last checkpoint when restarted.

1.4.3.5 Recovery Phase Of Data Collection

Audit Vault provides a *once and only once* guarantee to its users, stating that each audit record is archived in the Audit Vault Server once and only once. It implements a checkpoint and recovery mechanism for this purpose.

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 had 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. The agent, however, automatically filters out records that were committed *after* the last checkpoint and re-collected when the collection plug-in restarts. In fact, collection plug-ins built using this SDK (see Chapter 3) write the `EventTimeUTC` field into a file with the extension `.atc`. A script can subsequently read this file and delete audit records as appropriate.

1.4.3.6 Audit Trail Clean-Up

Audit Trail clean-up is a feature that some secured targets provide to clean up audit records once they've been *archived*. If this type of feature exists in the secured target, an Audit Vault collection plug-in can integrate with it, to tell the secured target to what extent the audit trail has been archived. This enables the secured target to clean up the audit trail (remove the original audit data) to that point, knowing this will not result in 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 secured target system by invoking the appropriate interface of the feature. For instance, the system may read a timestamp from a file in the filesystem 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 secured targets provides this type of utility in the `DMBS_AUDIT_MGMT` package, and the Oracle Database prepackaged collection plug-ins integrate with it.

1.5 General Procedure for Writing Audit Collection Plug-ins

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 secured target type or a new audit trail for an existing secured target type.
2. Check *Oracle Audit Vault and Database Firewall Administrator's Guide* to see if Oracle provides a plug-in that does what you want. If so, use it; do not *reinvent the wheel*.

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

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

3. Understand the events that your secured 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 (see Appendix A, "Audit Vault Server Fields"). If the events or fields you need are not available, you can use extension fields (see "Extension Field" on page 1-4). 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 secured target types. If you believe your fields satisfy this criterion, please contact Oracle Support.

4. Decide which type of collection plug-in to write, as discussed in "Types of Audit Collection Plug-ins" on page 1-3.

5. Set up the development environment. See Chapter 2, "Setting Up Your Development Environment".

6. Learn more about the type of plug-in you are creating in Chapter 3, "Audit Collection Plug-ins".

7. Determine the following for your collection plug-in:

   - How to connect to the secured target.
   - How to interrogate the secured 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 as described in "Audit Trail Clean-Up" on page 1-8.

9. Set up the collection plug-in parameters.

10. Create a plug-in manifest file to describe the collection plug-in. See Chapter 4, "Packaging Audit Collection Plug-ins".

11. Run the `avpack` utility to package the plug-in (see Chapter 4, "Packaging Audit Collection Plug-ins").

12. Test the plug-in the staging environment (see Chapter 5, "Testing Audit Collection Plug-ins").

13. If the plug-in works, make it available to the Audit Vault administrator to deploy in the development environment, using the command-line commands described in *Oracle Audit Vault and Database Firewall Administrator's Guide*.

2

Setting Up Your Development Environment

This chapter describes the process of setting up the Oracle Audit Vault Server development environment.

This chapter contains:

- Before You Set Up the Development Environment
- Setting Up the Development Environment
- Audit Collection Plug-in Directory Structure

2.1 Before You Set Up the Development Environment

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

Before you set up a developer environment, do the following:

- **Obtain and install Oracle Audit Vault and Database Firewall 12.1.2.** 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 secured 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. See *Oracle Audit Vault and Database Firewall Installation Guide* for more information.
- To help you decide which type of collection plug-in to use, see "Determining Which Audit Collection Plug-in Type to Create" on page 1-3.

2.2 Setting Up the Development Environment

Before setting up the development environment, download the Oracle AVDF SDK from the Audit Vault Server console in Oracle Audit Vault and Database Firewall:

To download the SDK, do the following:

1. Log in to the Audit Vault Server console as an administrator.
2. Click the **Settings** tab, and then click **Plug-ins** (under the System subsection).
3. Click **Download SDK**.
4. Unzip the SDK into an empty directory.

To set up an environment for developing collection plug-ins, follow these steps:

1. Set the `AV_SDK_HOME` variable to the directory you extracted the SDK to.

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 relevant jars for your collection plug-in project.

   For example:

   ```
   $ export CLASSPATH=$AV_SDK_HOME/av/jlib
   ```

4. Create directories as necessary.

   See "Audit Collection Plug-in Directory Structure" on page 2-2 for the directory structure that you should use.

2.3 Audit Collection Plug-in Directory Structure

This section contains these topics:

- General Directory Structure
- Audit Collection Plug-in Directory Structure
- Staging a plugin-manifest.xml File

2.3.1 General Directory Structure

Example 2–1 shows a general directory structure.

Example 2–1 General Directory Structure

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

In Example 2–1, 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 naming convention similar to that described in the "Description of Plug-in Manifest File" on page 4-2.
- `patches`: Holds any OPatch patches for secured 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.

2.3.2 Audit Collection Plug-in Directory Structure

Example 2–2 shows the structure of a stage directory for a collection plug-in.

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
```

See "Description of Plug-in Manifest File" on page 4-2.

For a collection plug-in, place all mapper files in the `templates` directory, as shown in Example 2–2. This placement directs the collection plug-in to load the relevant template file based on the information that the file contains.

2.3.3 Staging a plugin-manifest.xml File

The `plugin-manifest.xml` file is a core XML file that describes the collection plug-in and defines its attributes. You must stage the `plugin-manifest.xml` file directly under the *`STAGE_DIR_ROOT`* directory, as follows:

- **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" on page 4-2 for description and lists of attributes
- Appendix C, "Example Code" for a complete sample file
- "Sample Schema for a plugin-manifest.xml file" on page B-1

3

Audit Collection Plug-ins

This chapter describes Audit Collection Plug-ins and how to create them.

This chapter covers these topics:

- About Audit Collection Plug-ins
- About Mapper Files
- Database Table Collection Plug-ins
- XML File Collection Plug-ins
- Secured Target Collection Attributes
- Pre-Processing Audit Data

3.1 About Audit Collection Plug-ins

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 secured target fields to Audit Vault Server fields and other details for secured target types and audit trails. Model XML Mapper files are provided in "Database Table Collection Plug-in Mapper File" on page C-1 and "XML File Collection Plug-in Mapper File" on page C-5.

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 secured target audit trail. This chapter discusses Mapper files in general and focuses on their creation in "Creating a Database Table Mapper File" on page 3-4 and "Creating the XML File Audit Collection Mapper File" on page 3-10.

2. Create a plugin-manifest file for this secured target type. See "Description of Plug-in Manifest File" on page 4-2.
3. Create the collection plug-in by packaging the mapper file and plugin-manifest file. See Chapter 4, "Packaging Audit Collection Plug-ins".

You can now deploy this collection plug-in at the Audit Vault Server and use it to collect audit data after adding the secured target and any necessary collection attributes for this secured target. See "Secured Target Collection Attributes" on page 3-17.

3.2 About Mapper Files

Mapper files are XML files which mainly contain information about which secured target fields you must collect from the audit trail and how these secured target fields map to Audit Vault Server fields. Mapper files are specific to a secured target type, and contains secured target information such as `securedTargetType` and `securedTargetVersion`, and so on.

Mapper files cover these details:

- The supported secure target name and secured target version.
- Mapping information from secured target fields to Audit Vault Server fields.
- Secured 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. See "Audit Collection Plug-in Directory Structure" on page 2-3.

See Also:

- "Database Table Collection Plug-ins" on page 3-2 for mapper files that retrieve audit records from database audit tables
- "XML File Collection Plug-ins" on page 3-8 for mapper files that retrieve audit records from XML files

3.3 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 present in "Database Table Collection Plug-in Mapper File" on page B-4.

This section covers these topics:

- Requirements for Database Table Collection Plug-ins
- Example Audit Trail for a Database Table Collection Plug-in
- Creating a Database Table Mapper File

3.3.1 Requirements for Database Table Collection Plug-ins

You can use database table collection plug-ins for reading audit trails from secured target database tables if the following criteria are met:

- Audit data must be stored in a single database table.
- The secured 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 AVDF.
- 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.
- The audit trail must contain a single column or group of columns that uniquely identify each audit record.

See schemas in Appendix B, "Schemas".

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

This section contains details of an example audit trail, which is used for demonstrating the creation and structure of a sample mapper file throughout the chapter.

Table 3–1 lists the structure for the hypothetical secured target type, `DBSOURCE`, that generates and stores audit data in a table `AUD`.

Table 3–1 AUD Audit Table Data Fields and Mappings

Secured 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

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

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

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

3.3.3 Creating a Database Table Mapper File

This section explains how to create an XML mapper file for a database table collection plug-in. It briefly describes each XML element and attribute used in this type of mapper file. You can read descriptions of all fields in Section A.1, "AVDF Fields."

See Also:

- The complete example, "Database Table Collection Plug-in Mapper File" on page C-1
- "Example Audit Trail for a Database Table Collection Plug-in" on page 3-3

The following is a step-by-step example that shows how to create a mapper file for database table collection plug-in:

- 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.

- Secured Target Connection Information

```
<ConnectionInfo>
   <Driver>platform.jdbc.dbsource.DBSourceDataSource</Driver>
</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 secured 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 secured target field names and <MapTo> elements that hold Audit Value field names that secured targets are mapped to.

There should be no many-to-one mappings from secured 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>  -->
```

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

- Core Fields

  ```
  <CoreFields>
  ```

  CoreFields provides mapping from secured target fields to core fields of Audit Vault Server. The data type of secured 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, which are described in "Core Fields" on page A-1.

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

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

  EVENT_TIME secured 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 secured targets to the Audit Vault `CommandClass` field. These transformations are mandatory.

The *to* attributes are values for the `CommandClass` field, as described in the "Actions and Target Types" on page A-2. 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>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 secured 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 secured target fields mapped to large fields in the Audit Vault Server, such as `CommandText` or `CommandParam`. The specified secured target fields must be of SQL data type `CLOB` or `String`, or be convertible to `String`.

Large fields are described in "Large Fields" on page A-2.

- Extension Field

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

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

See "Extension Field" on page A-2.

- Marker Field

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

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

The marker field is described in "Marker Field" on page A-2.

- 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>
```

3.4 XML File Collection Plug-ins

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 this schema, "Schema for XML File Collection Plug-in Mapper File" on page B-6.

This section covers these topics:

- Requirements for XML File Collection Plug-ins
- Example Audit Trail for an XML File Collection Plug-in
- Creating the XML File Audit Collection Mapper File
- XML Transformation for Non-Standard Audit Records

3.4.1 Requirements for XML File Collection Plug-ins

You can use collection plug-ins for reading audit trails from XML audit record files if the XML files meet the following criteria:

- 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 secured 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.

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

This section contains an example audit trail which is used for the creation and structure of a sample mapper file for an XML file collection plug-in throughout the chapter.

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>
```

Table 3–2 lists the audit record structure and mappings to Audit Vault Server fields for the hypothetical secured 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

Secured 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	extension field	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

3.4.3 Creating the XML File Audit Collection Mapper File

This section describes the process that you must follow to create an XML file collection plug-in mapper file. See "XML File Collection Plug-in Example" on page C-5 for the complete example.

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

- 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 secured 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 secured target field names, and <MapTo> elements that hold Audit Value field names that secured targets are mapped to.

There should be no many-to-one mappings from secured 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 secured target fields to core fields of Audit Vault Server. Secured 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, which are described in "Core Fields" on page A-1.

```
<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 secured targets to the Audit Vault Server CommandClass field. These transformations are mandatory.

The *to* attributes are values for the CommandClass field, as described in the "Actions and Target Types" on page A-2. 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>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 secured 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 secured target fields mapped to large fields in the Audit Vault Server. The specified secured target fields must be of SQL data type `CLOB` or `String`, or be convertible to `String`.

Large fields are described in "Large Fields" on page A-2.

- Extension Fields

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

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

See "Extension Field" on page A-2.

- Marker Fields

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

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

Marker fields are described in "Marker Field" on page A-2.

3.4.4 XML Transformation for Non-Standard Audit Records

If you have audit records in a non-standard audit data format, that is, unlike the audit record example shown in "Example Audit Trail for an XML File Collection Plug-in" on page 3-9, you can apply XML transformation using XSL on the XML audit records. To do this, 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 you can enhance file collection plug-ins to support a variety of XML audit data formats.

3.4.4.1 Additional Requirement for XML Transformation Using XSL

To transform non-standard audit records into the current format, the following is required:

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

3.4.4.2 Changes Required to Transform Non-Standard Audit Records

In order to transform non-standard audit records, you must perform these steps:

1. Add a section such as this, to the mapper file after `<RecordInfo>`, as shown in Section 3.4.3, "Creating the XML File Audit Collection Mapper File", specifying the name of XSL file to be used for transformation and the `SourceFileStartTag` for the file to be transformed, see "Sample Non-Standard XML Audit Data Record" on page 3-14.

   ```
   <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. For an example, see "Creating an XSL File for Transformation" on page 3-15
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.

3.4.4.3 Sample Non-Standard XML Audit Data Record

This section contains an sample of an XML data record that needs to be transformed in the proper XML format required for an XML file collection plug-in. Your source system may 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>
```

3.4.4.4 Creating an XSL File for Transformation

This an example of an XSL transformation file that defines transformation rules. You need to create a version that can transform the source audit records that your system creates and place it in the `templates` folder of the plugin.

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>
```

The following 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 Example 3–3.

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>
```

3.5 Secured Target Collection Attributes

For database table and XML file collection plug-ins, you need to set secured target collection attributes after you have deployed the collection plug-in in the Audit Vault Server and registered the secured target but before the plug-in starts collection from the audit trail.

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

Required secured target attributes are:

- `av.collector.securedtargetversion` **(Mandatory)**: Current version of the secured 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 timezone. 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 secured target event time from UTC timezone. This helps the collector to report event time correctly to the Audit Vault Server by adjusting the timezones. This attribute is not needed for an XML file collection plug-in if the event time itself contains the timezone information.

3.6 Pre-Processing Audit Data

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

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

You may be able to pre-process 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 Windows all require you to execute the `db2audit` program to extract audit records from a proprietary binary format into a text file. This program must be run periodically, as the user who owns the DB2 software, to extract new records.

While you cannot define a collection plug-in to read the file directly, you may be able to 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 may 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 Audit Vault Server.

4

Packaging Audit Collection Plug-ins

This chapter describes steps you need to perform to package collection plug-ins.

This chapter contains these topics:

- Flow of Packaging
- External Dependencies
- Creating New Versions of Your Audit Collection Plug-ins
- Description of Plug-in Manifest File
- avpack Tool

4.1 Flow of Packaging

The previous chapters have taken you to the point where you can package the collection plug-in.

Chapter 2 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.

Chapter 3 described the mapper file.

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

1. When the collection plug-in program is ready to be packaged, create a directory structure as illustrated in the "Audit Collection Plug-in Directory Structure" on page 2-2.
2. Create a `plugin-manifest.xml` file. Refer to the "Description of Plug-in Manifest File" on page 4-2, if needed. 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. See "Example Code" on page C-1 for example `plugin-manifest.xml` files specific to your type of 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. See "avpack Tool" on page 4-4.

   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.

4.2 External Dependencies

External dependencies, in this packaging process, are files that will be needed during runtime, but may not be available when you package the collection plug-in. For example, if your collection plug-in depends on a third-party component that the end-user licenses, or a component with an issue related to licensing or copyright, you may not able to package this component, and will expect the end-user to provide it 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 Audit Vault Server. If the file is missing, `avpack` will fail to deploy the collection plug-in 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. Then, `avpack` can deploy the collection plug-in successfully.

4.3 Creating New Versions of Your Audit Collection Plug-ins

If you create new versions of the collection plug-ins, you can easily plug them in to replace existing versions without difficulty.

Using the `avcli` command-line tool, you can use the `DEPLOY PLUGIN` command, described in *Oracle Audit Vault and Database Firewall Administrator's Guide*, to update an existing collection plug-in to a newer version. collection plug-ins are cumulative in nature. All necessary files are created and updated.

Collection plug-ins can be removed or undeployed, using the `avcli` tool and the `UNDEPLOY PLUGIN` command, described in *Oracle Audit Vault and Database Firewall Administrator's Guide*. This command is atomic, that is, it is an all or nothing transaction, which helps maintain a high degree of system stability.

4.4 Description of Plug-in Manifest File

The `plugin-manifest.xml` file is a core XML file that describes the collection plug-in and 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, separated by periods: `majr.minr.minr.patch.hotfix`.
  - **provider:** The name of the provider, typically, the company or organization.

  - **copyright**: Any copyright notices for the collection plug-in.

- **TargetVersion**: Oracle Audit Vault and Database Firewall Version that the collection plug-in is compatible with. The **min** attribute represents the minimum version of the target.
- **extensionSet**: A set of `extensionPoints`.
- **ExtensionPoint:** Each `extensionPoint` uniquely identifies the area of AVDF that is being extended by the collection plug-in. Currently, 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`. See "External Dependencies" on page 4-2 for further information.
  - **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 which 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 and will be ignored if set.
    - **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.

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:

  For sample manifest plug-in files for specific collection plug-ins:

  - "Database Table Collection Plug-in Manifest File" on page C-4
  - "XML File Collection Plug-in Manifest file" on page C-8

4.5 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.

You must lay out the collection plug-in artifacts following the directory structure recommended in "Audit Collection Plug-in Directory Structure" on page 2-2. Then, you can use `avpack` to generate a collection plug-in package.

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. It also performs some basic sanity checks such as verifying that all the files specified in the `plugin-manifest.xml` are staged in their corresponding directories, and so on.

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`.

Once validation is complete, the tool packages the files into a `.zip` plug-in package suitable for deployment with Oracle Audit Vault and Database Firewall.

Usage:

```
avpack -stagedir <directory name> -o <archive filename> [-l <loglevel> ]
```

Where:

`directory`: The directory under which the collection plug-in artifacts are staged. Contents of this directory will be archived in the generated plug-in archive.

`archive filename`: The name for the generated plug-in archive file. It should end with a `.zip` extension. (for example, `myplugin.zip`).

`log level`: Optional: Sets the log level to the level specified. Supported log levels: `INFO`, `WARNING`, `ERROR`, and `DEBUG`. Default log level is `INFO`.

For further help, see `avpack -h` output.

5

Testing Audit Collection Plug-ins

This chapter provides a general description of the kind of testing that you might want to do for your collection plug-ins. Be sure to analyze your database and audit trails for other issues that require testing.

This chapter contains:

- Requirements for Testing Audit Collection Plug-ins
- Typical Audit Collection Plug-in Testing Processes
- Deploying an Oracle Audit Vault Agent
- Redeploying the Oracle Audit Vault Agent

5.1 Requirements for Testing Audit Collection Plug-ins

You should prepare for testing by performing the following:

- Deploy the Audit Vault Server and an Audit Vault Agent, as described in "Before You Set Up the Development Environment" on page 2-1.
- 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).

5.2 Typical Audit Collection Plug-in Testing Processes

The typical testing process as follows:

1. Perform functional testing:
   a. Deploy the collection plug-in in the generated `.zip` archive that you created in Chapter 4, "Packaging Audit Collection Plug-ins," 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, as described in "Redeploying the Oracle Audit Vault Agent" on page 5-3.
   c. Register the source using the `AVCLI` utility. See *Oracle Audit Vault and Database Firewall Administrator's Guide*.
   d. Issue an `AVCLI START COLLECTION` command to start gathering records from the audit trail supported by this collection plug-in. See Oracle Audit Vault and Database Firewall Administrator's Guide.

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. See Oracle Audit Vault and Database Firewall Administrator's Guide.

g. Undeploy the collection plug-in. See Oracle Audit Vault and Database Firewall Administrator's Guide.

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.

5.3 Deploying an Oracle Audit Vault Agent

This agent can be on the same computer as the Audit Vault Server or a different one.

To deploy the agent, follow these steps:

1. Register the agent host using the `AVCLI` command `REGISTER HOST`. See *Oracle Audit Vault and Database Firewall Administrator's Guide*.

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. Send the activation request to the Audit Vault Server Administrator using the following command:

   ```
   $ $AGENT_HOME/bin/agentctl activate
   ```

   The Audit Vault Server Administrator must approve the activation request using either the following command or the Administrator Console.

   ```
   avcli> activate host '<agent host>';
   ```

   The activation key generated during the activation approval process must be sent to agent administrator.

6. Start the agent using the activation key provided by the Audit Vault Administrator:

   ```
   $ $AGENT_HOME/bin/agentctl start -key <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
   ```

7. The agent can be stopped anytime using the following command:

   ```
   $ $AGENT_HOME/bin/agentctl stop
   ```

   It may take several seconds before the agent comes to a complete stop and the agent process is shutdown.

   Activation is a one-time activity. You will not have to do it again.

5.4 Redeploying the Oracle Audit Vault Agent

You may need to redeploy the agent for various reasons while testing the collection plug-in. It is assumed that an agent is already set up and a directory created.

To redeploy the agent, follow these steps:

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 by invoking 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

This appendix contains the AVDF events and fields that you can map to in your collection plug-ins.

This appendix covers these topics:

- AVDF Fields
- Actions and Target Types

A.1 AVDF Fields

This section discusses the different types of AVDF values:

- Core Fields
- Large Fields
- Marker Field
- Extension Field

A.1.1 Core Fields

Core fields are fundamental to all source types and central to the description of an event. These fields are present in most audit records, for reporting, filtering, and so on.

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. See "Actions" on page A-3.

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`. See "Target Types" on page A-5.

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.

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.

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.

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 large field.

CommandParam: Contains the parameters of the command that caused the event. This is also a large field.

A.1.2 Large Fields

Large fields are fields that can contain arbitrarily large amounts of data.

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

Marker Field of a Record: The marker is a string that uniquely identifies a record in a a trail. During the recovery process, 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, in Oracle database, the session Id and Entry id (a unique identifier within a session) define a marker.

A.1.4 Extension Field

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

A.2 Actions and Target Types

This section contains lists of target types and actions that Audit Vault is aware of. 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.

This section covers the following:

- Actions
- Target Types

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.

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. Audit Vault Server is current aware of the following actions:

Create

Read

Select

Insert

Delete

Remove

Truncate

Update

Modify

Execute

Communicate

Set

Get

Verify

Logon

Logoff

Authorize

Violate

Acquire

Release

Enable

Disable

Backup

Restore

Open

Close

Apply

Grant

Revoke

Deny
Suspend
Resume
Commit
Savepoint
Checkpoint
Rollback
Rollforward
Copy
Move
Rename
Analyze
Audit
Noaudit
Migrate
Validate
Startup
Shutdown
Unmount
Mount
Invalid
Associate
Disassociate
Deny
Proxy
Initialize
Unknown
Subscribe
Unsubscribe
User configurable event
DDL
Control
Undo
Access
Deadlock
DML
Transaction Control

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.

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.

Audit Vault Server is current aware of the following target types:

DATABASE

OBJECT

OPERATOR

OUTLINE

PROCEDURE

PUBLIC DATABASE LINK

TYPE BODY

CONTROL FILE

FLASHBACK

BROKER QUEING

BUFFERPOOL

SCHEMA

SYSTEM

TRIGGER

PRIVILEGE

EVENT MONITOR

RULE

EVALUATION

USER

STATISTICS

METHOD

CONTEXT

MESSAGE

VIEW

CONNECTION

TAPE

SAVEPOINT

USER OR PROGRAM UNIT LABEL

APP ROLE

EDITION

FLASHBACK ARCHIVE

MATERIALIZED VIEW LOG
NODEGROUP
PACKAGE BODY
RESOURCE COST
ROLE
INDEXTYPE
USER_RECYCLEBIN
SAVEPOINT
ASSEMBLY
CLUSTER
FUNCTION
JAVA
MINING MODEL
PUBLIC SYNONYM
REWRITE EQUIVALENCE
SEQUENCE
SUMMARY
DEFAULT
AUTHORIZATION
INSTANCE
NODE
CHECKPOINT
EXPRESSION
DATABASE LINK
DIMENSION
INDEX
PACKAGE
SYNONYM
TABLE
TABLESPACE
TYPE
DIRECTORY
LIBRARY
RESTORE POINT
ALL TRIGGERS
APPLICATION
TRANSACTION

USER LOGON

REVOKE

UNKNOWN

MATERIALIZED VIEW

SESSION

TABLE OR SCHEMA POLICY

INDEXES

PROFILE

ROLLBACK SEG

TRACE

DBA_RECYCLEBIN

SUBSCRIPTION

B

Schemas

This chapter contains schemas for plug-in manifest files and collection plug-ins.

- Sample Schema for a plugin-manifest.xml file
- Database Table Collection Plug-in Mapper File
- Schema for XML File Collection Plug-in Mapper File

B.1 Sample Schema for a plugin-manifest.xml file

This is the schema for a `plugin-manifest.xml` file. Please use this Schema to validate any `plugin-manifest.xml` that you author. Typically, the type of collection plug-in requires you to modify the `trailInfo` section, and possibly other sections. See Appendix C, "Example Code" for information specific to the collection plug-in type.

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://xmlns.oracle.com/av/plugin"
           targetNamespace="http://xmlns.oracle.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="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 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="srcVersion" 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>
```

See Also: "Staging a plugin-manifest.xml File" on page 2-3

B.2 Database Table Collection Plug-in Mapper File

Example B–2 Database Table Collection Plug-in Mapper Schema

```
<?xml version="1.0"?>
<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="TargetOwner"/>
  <xsd:enumeration value="ErrorId"/>
  <xsd:enumeration value="ErrorMessage"/>
  <xsd:enumeration value="EventStatus"/>
  <xsd:enumeration value="EventName"/>
  <xsd:enumeration value="TargetType"/>
  </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>


<!-- 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"/>
      <!-- Database connection information -->
      <xsd:element name="ConnectionInfo" minOccurs="1" maxOccurs="1">
        <xsd:complexType>
          <xsd:sequence>
            <!-- Datasource class-->
            <xsd:element name="Driver" type="xsd:string"  minOccurs="1"
                   maxOccurs="1"/>
           </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <!-- Source to AV server fields Mapping for Core, Large, Extension fields
                    and Marker-->
      <xsd:element name="FieldMappingInfo" minOccurs="1" maxOccurs="1">
          <xsd:complexType>
          <xsd:all>
            <!-- Core Field Mapping -->
            <xsd:element name="CoreFields" minOccurs="1" maxOccurs="1">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name="Map" minOccurs="1" maxOccurs ="13">
                    <xsd:complexType>
                      <xsd:all>
                        <xsd:element name="Name" type="xsd:string" />
                        <xsd:element name="MapTo" type="CoreFieldValues" />
                        <xsd:element name="Transformation" minOccurs="0"
                                     maxOccurs="1">
                          <xsd:complexType>
                            <xsd:sequence>
                              <xsd:element name="ValueTransformation"
                                           minOccurs="1" maxOccurs="unbounded">
                                <xsd:complexType>
                                  <xsd:attribute name="from" type="xsd:string"
                                              use="required"/>
                                  <xsd:attribute name="to" type="xsd:string"
                                              use="required"/>
                                </xsd:complexType>
                              </xsd:element>
                            </xsd:sequence>
                          </xsd:complexType>
                        </xsd:element>
                      </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:all>
                      </xsd:complexType>
                    </xsd:element>
                  </xsd:sequence>
                </xsd:complexType>
              </xsd:element>
              <!-- List of fields to be mapped to extension fields-->
              <xsd:element name="ExtensionField" minOccurs="0" maxOccurs="1">
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name="Name" type="xsd:string" minOccurs="0"
                                  maxOccurs ="unbounded" />
                  </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="unbounded"/>
                  </xsd:sequence>
                </xsd:complexType>
              </xsd:element>
            </xsd:all>
          </xsd:complexType>
        </xsd:element>
      </xsd:all>
      <!-- Source Type-->
      <xsd:attribute name="securedTargetType" type="xsd:string" use="required"/>
      <!-- Max source version supported by the template-->
      <xsd:attribute name="maxSecuredTargetVersion" type="xsd:string"
use="required"/>
      <!-- Min source 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

Example B–3 XML file collection plug-in Mapper Schema

```
<?xml version="1.0"?>
```

```
<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="TargetOwner"/>
  <xsd:enumeration value="ErrorId"/>
  <xsd:enumeration value="ErrorMessage"/>
  <xsd:enumeration value="EventStatus"/>
  <xsd:enumeration value="EventName"/>
  <xsd:enumeration value="TargetType"/>
  </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>
<!-- 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>
      <!-- Source to AV server fields Mapping for Core, Large, Extension fields
                           and Marker-->
      <xsd:element name="FieldMappingInfo" minOccurs="1" maxOccurs="1">
          <xsd:complexType>
          <xsd:all>
            <!-- Core Field Mapping-->
            <xsd:element name="CoreFields" minOccurs="1" maxOccurs="1">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name="Map" minOccurs="1" maxOccurs ="13">
                    <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" minOccurs="0"
                            maxOccurs="1">
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name="ValueTransformation"
                                  minOccurs="1" maxOccurs="unbounded">
                      <xsd:complexType>
                        <xsd:attribute name="from" type="xsd:string"
                                  use="required"/>
                        <xsd:attribute name="to" type="xsd:string"
                                  use="required"/>
                      </xsd:complexType>
                    </xsd:element>
                  </xsd:sequence>
                </xsd:complexType>
              </xsd:element>
            </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:all>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <!-- List of fields to be mapped to extension fields-->
  <xsd:element name="ExtensionField" minOccurs="0" maxOccurs="1">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Name" type="xsd:string" minOccurs="0"
                            maxOccurs ="unbounded" />
      </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="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

```
          </xsd:all>
        </xsd:complexType>
      </xsd:element>
    </xsd:all>
    <!-- Source Type-->
    <xsd:attribute name="securedTargetType" type="xsd:string" use="required"/>
    <!-- Max source version supported by the template-->
    <xsd:attribute name="maxSecuredTargetVersion" type="xsd:string"
use="required"/>
    <!-- Min source 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>
```

C

Example Code

This appendix contains examples for the different types of collection plug-ins:

- Database Table Collection Plug-in Example
- XML File Collection Plug-in Example

C.1 Database Table Collection Plug-in Example

This section covers these topics:

- Database Table Collection Plug-in Mapper File
- Database Table Collection Plug-in Manifest File

C.1.1 Database Table Collection Plug-in Mapper File

These attributes and fields are mandatory:

- `securedTargetType`
- `maxSecuredTargetVersion`
- `version`
- `TableName`
- `Driver`
- `EventTimeUTC`
- `CommandClass` transformations
- `EventStatus` transformations
- `MarkerField`

Source names that map to these Audit Vault Server fields are not mandatory, but if the information is not provided, when data collection starts, all audit records are treated as invalid:

- `UserName`
- `CommandClass`

> **See Also:** Appendix A, "Audit Vault Server Fields" for lists of fields and events

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 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>
```

C.1.2 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="DBSOOURCE"/>

              <!-- 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 Example

This section covers these topics:

- XML File Collection Plug-in Mapper File
- XML File Collection Plug-in Manifest file

C.2.1 XML File Collection Plug-in Mapper File

These attributes and fields are mandatory:

- `securedTargetType`
- `maxSecuredTargetVersion`
- `version`
- `HeaderInfo`
- `RecordInfo`
- `EventTimeUTC`
- `CommandClass` transformations
- `EventStatus` transformations
- `MarkerField`

Source names that map to these Audit Vault Server fields are not mandatory, but if the information is not provided, when data collection starts, all audit records are treated as invalid:

- `UserName`
- `CommandClass`

> **See Also:** Appendix A, "Audit Vault Server Fields" for lists of fields and events

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>
```

C.2.2 XML File Collection Plug-in Manifest file

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>
```

D

Bundled JDBC Drivers

This appendix describes the JDBC drivers that are bundled with the Oracle AVDF SDK.

D.1 About Bundled JDBC drivers

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 and may use drivers acquired elsewhere. However, if you plan to use any of the listed drivers, you will need to provide the information in Table D–1 in your mapper file and when registering a secured target.

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		`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.1.1 Connecting URLs

You use Connection URLs to specify the location of a database Secured Target when you register the Secured 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 AVDF specifies the format required by the JDBC driver in question in Table D–1.

Additionally, to use specific encryption methods in the connecting URL, you need to set the `EncryptionMethod` property as shown in the following example:

```
jdbc:av:[sqlserver]://hostname: port;[EncryptionMethod=encryptionMethod]
```

where the encryption methods can be SSL, requestSSL, and loginSSL.

Use this url to register a Secured Target, by entering it into the Secured Target Location field, with the Advanced mode selected.

See "Secured Target Connection Information" on page 3-5.

D.1.2 Driver class

```
<Driver>platform.jdbc.dbsource.DBSourceDataSource</Driver>
```

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" on page A-1.

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 type**, 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

about Audit Collection Plug-ins, 3-1
about XML mapper files, 3-2
Action fields, A-3
actions and target types, A-2
agents
  deploying, 5-2
  redeploying, 5-3
audit collection plug-in
  packaging, 4-1
  setting up development environment for, 2-1
Audit Collection Plug-ins, 3-1
audit collection Plug-ins
  determining which to use, 1-3
audit collection plug-ins
  types of, 1-3
audit records, 1-3
  storing, 1-4
audit trail, 1-2
  clean-up, 1-8
Audit Vault Agent
  deploying, 5-2
  how it works, 1-2
Audit Vault Server
  how it works, 1-2
Audit Vault Server events
  about, 1-3
Audit Vault Server fields
  about, 1-3
AVCLI commands, 3-17
avcli commands, 5-1
av.collector.atcintervaltime, 3-17
av.collector.securedtargetversion (Mandatory), 3-17
av.collector.timezoneoffset (Mandatory), 3-17
AVDF core fields, A-1
AVDF extension fields, A-2
AVDF fields, A-1
AVDF large fields, A-2
AVDF marker fields, A-2
AVDF SDK
  downloading, 2-1
avpack tool
  how to use, 4-4

C

checkpoint
  of a trail, 1-7
clean-up
  audit trail, 1-8
collection concepts, 1-7
collection phase, 1-7
collection plug-in
  directory structure, 2-3
  upgrading (creating new versions), 4-2
collection plug-in directory structure, 2-2
collection plug-in example
  XML file, C-5
collection plug-in manifest file
  XML file, C-8
collection plug-in mapper file
  XML file, C-5
collection plug-ins
  what are they?, 1-2
collection process
  overview of the whole process, 1-5
collection thread, 1-7
creating a database table mapper file, 3-4
creating the XML file audit collection mapper file, 3-10

D

data collection
  recovery phase, 1-7
database table collection plug-in example, C-1
database table collection plug-in manifest file, C-4
database table collection plug-in mapper file, B-4, C-1
database table collection plug-ins, 3-2
  requirements, 3-3
database table mapper file
  creating, 3-4
deploy plugin command, 4-2
deploying an Oracle Audit Vault Agent, 5-2
development environment, 2-2
  directory structure, 2-2, 2-3
  plugin-manifest.xml file staging, 2-3
  requirements, 2-1
  setting up, 2-1

directory structure
collection plug-in, 2-2
collection plug-ins, 2-2
general, 2-2

E

example
database table collection plug-in, C-1
example audit trail for a database table collection plug-in, 3-3
example audit trail for an xml file collection plug-in, 3-9
extension fields, A-2
external dependencies, 4-2

F

flow of control
inside the collection plug-in, 1-6
flow of packaging, 4-1

G

general procedure
for writing collection plug-is, 1-8

L

large fields, A-2

M

mapper file
database table collection plug-in, B-4, C-1
mapper files, 3-2
mappings
from secured target to Audit Vault Server, 1-7
marker fields, A-2

O

Oracle Audit Vault and Database Firewall
what is it?, 1-1

P

packaging, 4-1
external dependencies, 4-2
flow of, 4-1
plugin id, 4-2
plug-in manifest file
database table collection, C-4
plugin-manifest.xml file
about, 2-3
description of, 4-2
sample schema, B-1
staging, 2-3
plug-ins
redeploying agent, 5-3
requirements for testing, 5-1
testing procedure, 5-1
pre-processing audit data, 3-17

R

recovery phase
of data collection, 1-7
requirements for database table collection plug-ins, 3-3
requirements for xml file collection plug-ins, 3-8

S

sample schema for a plugin-manifest.xml file, B-1
schema for xml file collection plug-in mapper file, B-6
SDK
downloading, 2-1
secured target, 1-2
secured target collection attributes, 3-17
secured target type, 1-2
setting up development environment, 2-1
staging
plugin-manifest.xml file, 2-3
storing audit records, 1-4

T

target types, A-5
trail
checkpoint, 1-7

U

undeploy plugin command, 4-2
unresolved-external tag, 4-2
upgrading
collection plug-ins, 4-2

W

what are collection plug-Ins?, 1-2
writing collection plug-Ins
general procedures for, 1-8

X

xml file audit collection mapper file
creating, 3-10
xml file collection plug-in
example audit trail for, 3-9
XML file collection plug-in example, C-5
XML file collection plug-in manifest file, C-8
XML file collection plug-in mapper file, C-5
xml file collection plug-in mapper file
schema, B-6
XML file collection plug-ins, 3-8
xml file collection plug-ins
requirements for, 3-8
XML mapper files, 3-1
about, 3-2