

# **DB Audit Expert™ 3.1**

## **Application Programming Interface Reference**

**Version 1.0**

# Contents

<b>About This Guide</b> .....	<b>7</b>
Intended Audience.....	7
Conventions used in this document.....	7
Abbreviations and Product Reference Terms.....	8
Trademarks.....	8
<b>CHAPTER 1, DB Audit API Architecture, Classes and Methods</b> .....	<b>9</b>
Architecture Overview.....	9
Database Profiles and Connections Setup.....	10
Manually Configuring Database Connections .....	10
Programmatically Configuring Database Connections.....	11
Testing Database Connections .....	11
Audit Setup API.....	11
Overview of System Audit Classes .....	11
Overview of Data-Change Audit Classes .....	12
Audit Reporting API .....	12
RMI Interface API .....	12
<b>CHAPTER 2, Installation</b> .....	<b>14</b>
Client-Side API Installation .....	14
Database Server-Side Installation .....	14
DB2 for Linux, Unix and Windows.....	14
DB2 for zSeries and iSeries Platforms .....	15
Oracle for Windows .....	15
Oracle for non-Windows Platforms.....	15
Microsoft SQL Server .....	15
Sybase ASE and ASA .....	16
<b>CHAPTER 3, Database Connectivity</b> .....	<b>17</b>
Profile Manager Class and Related Classes .....	17
Overview.....	17
Instantiating Profile Manager Class.....	18
getInstance .....	18
initInstance.....	18
Database Driver Class.....	19
Constructor .....	19
getName .....	19
setName.....	20
getJdbcDriverPath.....	20
setJdbcDriverPath.....	21
getJdbcDriverClass .....	21
setJdbcDriverClass .....	22
getJdbcDriverType .....	22
setJdbcDriverType .....	23
Managing Database Drivers .....	23
getDbDriverList .....	24

getDbDriverNames.....	24
getDbDriver.....	25
addDbDriver.....	25
deleteDbDriver.....	26
Database Connection Profile Class.....	26
Constructor.....	26
getName.....	27
setName.....	27
getDriverName.....	28
setDriverName.....	28
getJdbcURL.....	29
setJdbcURL.....	29
getUsername.....	30
setUsername.....	31
getPassword.....	31
setPassword.....	32
getRepositoryDatabase.....	33
setRepositoryDatabase.....	33
getConnectionParameter.....	34
setConnectionParameter.....	35
isLoggedOnAsSYSDBA.....	35
getDbPath.....	36
setDbPath.....	37
getMailSender.....	37
setMailSender.....	38
getMailServer.....	38
setMailServer.....	39
getMailPassword.....	39
setMailPassword.....	40
Managing and Using Database Profiles.....	41
getDbProfileList.....	41
getDbProfileNames.....	41
getDbProfile.....	42
addDbProfile.....	42
deleteDbProfile.....	43
createDbConnect.....	43
Default Configuration Generator Class.....	44
Overview.....	44
Database Connectivity Class.....	44
Overview.....	44
Connecting/Disconnecting to/from Database.....	44
connect.....	44
disconnect.....	45
Testing Connection.....	45
Using Connection.....	46
Getting Connection Information.....	47
isConnected.....	47
checkConnected.....	47
getDbVersion.....	48
getDbProfile.....	48
getDbDriver.....	49

getConnection .....	49
repositoryUsed .....	49
<b>CHAPTER 4, System Audit Management .....</b>	<b>51</b>
Class Hierarchy .....	51
Parameter Names and Values .....	52
Oracle System Audit Management Tasks .....	53
Set System Audit State .....	53
Install System Audit .....	53
Uninstall System Audit .....	53
Enable System Audit .....	54
Disable System Audit .....	54
Set Audit Operations and Filters .....	55
Add SQL Statement Audit .....	55
Remove SQL Statement Audit .....	59
Add Object Access Audit .....	60
Remove Object Access Audit .....	62
Add Session Audit .....	64
Remove Session Audit .....	65
Configure Alternative Audit Trail .....	67
Install/Uninstall Oracle Alternative Audit Trail .....	67
Schedule/Remove Oracle Alternative Data Transfer Job .....	69
Set Advanced Audit Options .....	71
Install/Uninstall Oracle SYS Operations Audit .....	71
Install/Uninstall Server Errors Auditing and Alerting .....	73
Move SYS.AUD\$ Table to Non-SYSTEM tablespace .....	75
Informational Methods .....	77
Get System Audit Status .....	77
Install DB Audit Mail Sending SQL Procedure .....	78
Microsoft SQL Server System Audit Management Tasks .....	79
Set System Audit State .....	79
Install System Audit .....	79
Uninstall System Audit .....	84
Enable System Audit .....	84
Disable System Audit .....	84
Set Audit Operations and Filters .....	85
Update Audit Operations .....	85
Update Audit Filters .....	87
Set Advanced Audit Options .....	88
Update Audit Queue Size .....	88
Informational Methods .....	89
Get Current Audit Settings .....	89
Get System Audit Status .....	90
Install DB Audit Mail Sending SQL Procedure .....	91
Sybase SQL Server and ASE System Audit Management Tasks .....	92
Set System Audit State .....	92
Install System Audit .....	92
Uninstall System Audit .....	92
Enable System Audit .....	93
Disable System Audit .....	94
Set Audit Operations and Filters .....	95

Add Server-level Operations Audit .....	95
Remove Server-level Operations Audit .....	99
Add Database-level Operations .....	100
Remove Database-level Operations Audit .....	103
Add Schema Object-level Audit.....	104
Remove Schema Object-level Audit.....	107
Add Login-level Audit .....	109
Remove Login-level Audit .....	111
Set Advanced Audit Options.....	112
Update Audit Queue Size.....	112
Set 'Suspend Audit When Device Full' Status .....	113
Add New System Audit Trail Table.....	115
Attach Threshold Procedures.....	116
Uninstall Threshold Procedures .....	117
Informational Methods .....	118
Get System Audit Status .....	118
Get System Audit Trail Tables Count.....	119
Get Audit Queue Size .....	120
Get 'Suspend Audit When Device Full' Status .....	121
Install DB Audit Mail Sending SQL Procedure .....	122
DB2 System Audit Management Tasks.....	123
Set System Audit State.....	124
Install System Audit.....	124
Uninstall System Audit .....	126
Enable System Audit.....	127
Disable System Audit.....	127
Set Audit Operations .....	128
Flush Audit Data .....	129
Informational Methods .....	130
Get System Audit Status .....	130
Install DB Audit Mail Sending SQL Procedure .....	131
Common System Audit Management Tasks (All Database Systems) .....	132
Truncate System Audit Trail Table .....	132
Archive System Audit Trail Data to a Table.....	133
Archive System Audit Trail Data to a File.....	135
Uninstall Audit Repository Objects .....	136
<b>CHAPTER 5, Data-change Audit Management .....</b>	<b>138</b>
Class Hierarchy.....	138
Parameter Names and Values.....	139
Data-change Audit Management Tasks .....	139
Install Data-change Audit for a Table .....	139
Uninstall Data-change Audit for a Table.....	144
Enable Data-change Audit Trigger .....	146
Disable Data-change Audit Trigger .....	147
Truncate Data-change Audit Trail Table .....	148
Archive Data-change Audit Trail to a Table.....	150
Archive Data-change Audit Trail to a File.....	152
Configure Settings for Data-change Audit Reports .....	153
Get Table Aliases.....	153

Set Table Alias .....	154
Get Column Aliases .....	156
Set Column Alias.....	157
<b>CHAPTER 6, Generating Audit Reports .....</b>	<b>160</b>
Class Hierarchy.....	160
Parameter Names and Values.....	161
Generating Reports .....	161
Common Report Filters and Parameters.....	164
System Audit Reports .....	165
Data-change Audit Reports .....	166
<b>CHAPTER 7, API Invocation Methods .....</b>	<b>168</b>
Direct Invocation from Java Programs .....	168
Direct Invocation from Non-Java Programs.....	168
Indirect Invocation Using Batch Files .....	169
Remote Invocation Using RMI functionality.....	169
<b>APPENDIX A, Hardware and Software Requirements .....</b>	<b>171</b>
<b>APPENDIX B, Licensing.....</b>	<b>173</b>

# About This Guide

This manual describes the features of the DB Audit Expert's API product, including how to invoke the API functions, which functions are available, their parameters and available options. The described features and how-to instructions apply to all supported database management systems running on any platform, unless otherwise noted.

## Intended Audience

This document is for application and database developers who need to build back-end auditing functions into their database applications.

## Conventions used in this document

This section describes the style conventions used in this document.

### *Italic*

An *italic* font is used for filenames, URLs, emphasized text, and the first usage of technical terms.

### Monospace

A monospaced font is used for code fragments and data elements.

### **Bold**

A **bold** font is used for important messages, names of options, names of controls and menu items, and keys.

### User Input

Keys are rendered in **bold** to stand out from other text. Key combinations that are meant to be typed simultaneously are rendered with "+" sign between the keys, such as:

#### **Ctrl+F**

Keys that are meant to be typed in sequence will be separated with commas, for example:

#### **Alt+S, H**

This would mean that the user is expected to type the Alt and S keys simultaneously and then to type the H key.

### Graphical symbols



– This symbol is used to indicate DBMS specific options and issues and to mark useful auditing tips.



– This symbol is used to indicate important notes.

## Abbreviations and Product Reference Terms

**DBMS** – Database Management System

**Oracle** – This refers to all supported Oracle® database servers

**SQL Server** – This refers to all versions of Microsoft® SQL Server™ database servers.

**ASE** – This refers to all versions of the Sybase® SQL Server™ and Sybase® Adaptive Server® Enterprise database servers.

**ASA** – This refers to all versions of the Sybase® Adaptive Server® Anywhere database servers.

**DB2** – This refers to all versions of the IBM® DB2® database servers.

The terms 'DB Audit Expert' and 'DB Audit' are used interchangeably in this document – they both refer to the same product]

## Trademarks

DB Audit, DB Audit Expert, DB Mail for Oracle, 24x7 Automation Suite, 24x7 Scheduler, DB Tools for Oracle are trademarks of SoftTree Technologies, Inc.

Windows NT, Windows 2000, Windows XP are registered trademarks of Microsoft Corporation. UNIX is the registered trademark of the X/Open Consortium. Sun, SunOS, Solaris, SPARC are trademarks or registered trademarks of Sun Microsystems, Inc. Ultrix, Digital UNIX and DEC are trademarks of Digital Equipment Corporation. HP-UX is a trademark of Hewlett-Packard Co. IRIX is a trademark of Silicon Graphics, Inc. AIX is a trademark of International Business Machines, Inc. AT&T is a trademark of American Telephone and Telegraph, Inc.

Microsoft SQL Server is a registered trademark of Microsoft Corporation.

Oracle is a registered trademark of Oracle Corporation.

Sybase, Adaptive Server, Adaptive Server Anywhere, Adaptive Server Enterprise, Anywhere Studio are registered trademarks of Sybase, Inc. or its subsidiaries.

IBM, DB2, UDB are registered trademarks of International Business Machines Corporation

All other trademarks appearing in this document are trademarks of their respective owners. All rights reserved.

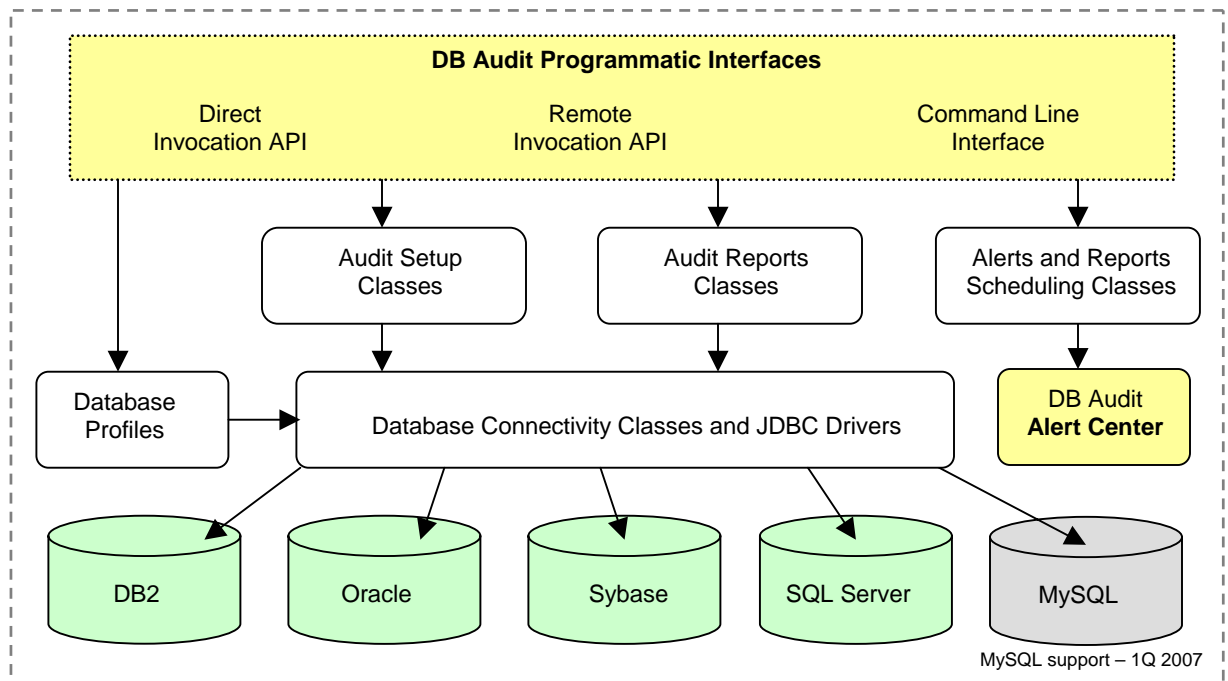


# CHAPTER 1, DB Audit API Architecture, Classes and Methods

## Architecture Overview

DB Audit Java API implementation consists of 6 functional class groups:

1. Database Profile Management
2. Database Connectivity
3. Audit Setup
4. Audit Reporting
5. Command Line Interface
6. RMI-based Client/Server Interface



These class groups allow:

- Integration and packaging of DB Audit as a JAR file for direct loading and method invocation from Java and non-Java programs.
- Invoking DB Audit methods from any application that can execute shell commands, in other words, allow use of console-mode interface
- Running DB Audit as a server application and using Java RMI to invoke its methods remotely from Java client programs. This interface can be used by remote thin clients to remotely install audit functions, run audit reports and manage audit settings.

## Database Profiles and Connections Setup

### Manually Configuring Database Connections

1. Locate **profiles.xml** file in the installation directory. Open this file in any text editor, for example, in UNIX vi or in Windows Notepad.
2. In the opened file locate pre-configured profile matching the type of the database connection you want to setup, for example, for Oracle connection, find "Oracle Profile." Rename this profile using a more descriptive name, for example, "Oracle Financials Server."
3. Enter the database connection URL (**jdbcUrl** field), **user** and **password** fields of the chosen profile in the **profileList** section. Note that the formats of URL strings are DBMS specific and differ for different drivers, database systems and communication protocols. Check your JDBC driver documentation for what to put in the JDBC URL.

Use the following formats for TCP/IP based connections using pre-configured JDBC drivers packaged with DB Audit API:

 **DB2:** jdbc:db2://[DB2\_server\_name]:[port(default – 50000)]/[database\_name]

Example: jdbc:db2://MY\_BIG\_SERVER:50000/DATAWARE

 **SQL Server:** jdbc:sqlserver://[MSSQL\_server\_name]:[port(default – 1433)]

Example: jdbc:sqlserver://MY\_BIG\_SERVER:1433


**Note:** To connect to a named instance through, you must specify the port number that is associated with the named instance, instead of the name of the named instance. For more information see Microsoft Knowledge Base article #313225 <http://support.microsoft.com/kb/313225>.

**Note about using Trusted/Windows Authentication connections:**


**SQL Server 2000:** At present, due to limitations in the Microsoft JDBC driver, Windows Authentication is not supported. The database server must be configured to use Mixed Authentication mode and the connection must be made using SQL Server user account..

**SQL Server 2005:** To use Windows Authentication, add *integratedSecurity=true* option to the end of the database URL value. In addition, you must copy **sqljdbc\_auth.dll** file to your application directory so that the JDBC driver can find it in the current directory. On a 32-bit processor system, copy sqljdbc\_auth.dll file from the x86 folder; on a 64-bit processor system, copy sqljdbc\_auth.dll file from the x64 folder. For more information see "Building the Connection URL" document posted on Microsoft web site <http://msdn2.microsoft.com/en-us/ms378428.aspx>


Example: jdbc:sqlserver://MY\_BIG\_SERVER:1433;integratedSecurity=true;

 **Oracle:** jdbc:oracle:thin:@[Oracle\_server\_name]:[port(default – 1521)]:[sid]

Example: jdbc:oracle:thin:@MY\_BIG\_SERVER:1521:ORCL


 **ASE, ASA:** jdbc:sybase:Tds:[Sybase\_server\_name]:[port(default – 2048)]/[database\_name]


Example: jdbc:sybase:Tds: MY\_BIG\_SERVER:2048/MASTER


 **Important Note:** Saving user and password values in the profile properties is

optional. If specified you can use the profile later for subsequent connections without providing user and password values each time. **This option is not recommended for security reasons because the saved user and password values are stored as open text. Anyone who has access to the profiles.xml file can read them.** If you do not specify user and password values in the profile, you will need to provide them as parameters in the database connection method or on the command line if you use the command line interface.

4. Enter additional parameters if required

 **Oracle:** If a SYS connection to Oracle is needed, you must use SYSDBA option. Specify "sysdba" (without quotes) in the **connectionParameter** field.

 **SQL Server and ASE:** The **repositoryDatabase** field has to be specified in the profile. This value controls in which database DB Audit will search/create its audit repository tables.

 **DB2:** The **dbPath** field has to be specified in the profile. This value instructs DB Audit where to find DB2/SQLLIB/FUNCTION directory on the server.

5. If necessary, configure connections to additional database servers adding additional profiles and entering their properties as instructed above in steps 3 and 4.

## Programmatically Configuring Database Connections

Use the Database Profiles Manager class to manage profiles programmatically. For more information read [Database Connection Profile Manager Class](#) topic in CHAPTER 3.

## Testing Database Connections

The simplest way to verify your **profiles.xml** file contains the correct settings is to run the test console using the following command line:

```
java -jar dbaudit.jar /D profileName [/U user] [/P password] /T
```

The profileName here is the name of the profile as it is entered in **profiles.xml** file. User and password parameters are required if they are not saved with the profile.

# Audit Setup API

## Overview of System Audit Classes

A set of classes is provided for installing and configured audit settings for system audit. The base interface class is AuditSetup whose full name is **com.softtreetech.com.dbaudit.auditsetup.system.AuditSetup**. In the same package are also available several descendant interface classes for specific database systems and their implementations.

- AuditSetupDB2Impl
- AuditSetupMssqlImpl
- AuditSetupOracleImpl
- AuditSetupSybaseImpl

All required classes are stored within **dbaudit.jar** library. These classes and their methods are described in [CHAPTER 4, System Audit Management](#).

## Overview of Data-Change Audit Classes

A set of classes is provided for installing and configured audit settings for data-change audit. The base interface class is AuditSetupData whose full name is **com.softreotech.com.dbaudit.auditsetup.data.AuditSetupData**. In the same package are also available several descendants interface classes for specific database systems and their implementations.

- AuditSetupDataDB2Impl
- AuditSetupDataMssqlImpl
- AuditSetupDataOracleImpl
- AuditSetupDataSybaseImpl

All required classes are stored within **dbaudit.jar** library. These classes and their methods are described in [CHAPTER 5, Data-change Audit Management](#).

## Audit Reporting API

The Reports class is provided for running audit reports. This class internally interfaces with other database type specific classes, which normally not need to be called directly.

All required classes are stored within **dbaudit.jar** library. The Reports class and its methods is described in [CHAPTER 6, Generating Audit Reports](#).

## RMI Interface API

The RMI interface allows running DB Audit as a server on one computer or appliance and invoking its functions remotely from other computers and appliances. You can run DB Audit API in a server mode using the following command (the entire command must be entered on a single line):

```
java -Djava.security.policy=security.policy -cp .;dbaudit.jar  
com.softreotech.dbaudit.rmi.DBAuditRMIServer
```

All the classes required for the server are located within **dbaudit.jar** file. The main class for running the server is **DBAuditRMIServer**.

The **DBAuditRMIClient.jar** file contains the classes required for running the client part.

The detailed description of RMI interface and its classes and methods is described in [Remote Invocation Using RMI functionality](#) topic in CHAPTER7.

# CHAPTER 2, Installation

## Client-Side API Installation

The installation of DB Audit's API is straightforward and consists of 1 or 2 steps:

1. Unzip **dbaudit\_NNN.zip** file to the main directory of your application or any other directory from where you can load the unzipped files. Note that that NNN in the ZIP file name is substituted with the API build number.
2. If you are going to use DB Audit Alert Center, unzip **alertcenter\_NNN.zip** file to the same directory where you unzipped files from dbaudit\_NNN.zip. Note that that NNN in the ZIP file name is substituted with the API build number. Numbers in dbaudit and alertcenter zip files can differ but that doesn't mean they are incompatible.

## Database Server-Side Installation



**Important:** Database server side installation is DBMS and feature specific.

### DB2 for Linux, Unix and Windows

Read the following if you are going to use system auditing with DB2. You can skip this topic if you are not going to run system auditing.

**You must perform the following steps:**

1. From the DB Audit API installation directory, from the **server\_side\_files/db2** subdirectory, copy **dbauditRunner.jar** file to your DB2 server to **[db2 home]/sqllib/function** subdirectory. If DB2 is running on a Windows system, also copy **dbauditRunnerSrv.exe** file to your DB2 server to **[db2 home]/sqllib/function** subdirectory.
2. Add **dbauditRunner.jar** file to the Java CLASSPATH environment variable for the DB2 instance owner and if required bounce your DB2 instance in order for the new CLASSPATH to take effect.
3. Start DB Audit interface service on your DB2 server running the following command in **[db2 home]/sqllib/function** directory:

```
java -jar dbauditRunner.jar
```

4. **On Linux and Unix systems:** Add this command to the profile of the DB2 instance owner so that it can start automatically when your DB2 computer starts.

**On Windows systems:** Run dbauditRunnerSrv.exe program from the command line with **-install** parameter, which will install DB Audit interface service as Windows native services.

```
dbauditRunnerSrv.exe -install
```

By default the service will be installed under LocalSystem account. If DB2 instance is run under a different account, it is recommended that you change the DB Audit's service to run under the same account your DB2 instance is run. This can be done in Windows

Control Panel's Administrative Tools. Use the Services applet to modify service security settings and properties.

5. Schedule periodic run of Audit Record Flushing and Loading Procedures. See "Scheduling System Audit Record Flushing and Loading Procedures" topic in CHAPTER 3 of the DB Audit User's Guide for detailed instructions on how to do that.

It is important that you have the system audit installation for DB2 completed before you attempt to run Audit Record Flushing and Loading Procedures. These procedures can be installed using either **installSysAudit** API method or using DB Audit's Management Console. See "DB2 System Audit Management" topic in CHAPTER 5 for more information.

## DB2 for zSeries and iSeries Platforms

DB Audit doesn't require additional files to be installed on the server.

## Oracle for Windows

Read the following if you are going to audit activities of privileged users connected as SYSDBA or SYSOPER. You can skip this topic if you are not going to run this type of auditing.

**You must perform the following installation steps:**

1. From the DB Audit API installation directory, from the **server\_side\_files/oracle** subdirectory, copy **OraLogMonitor.exe** file to your Oracle server to **C:\Windows** subdirectory or any other subdirectory in the system search path.
2. Create new entry in the system registry to store path to the Oracle directory where you want audit files to be generated  
Registry path:  
*HKEY\_LOCAL\_MACHINE\SOFTWARE\SoftTree Technologies, Inc.\DB Audit\EventLog*  
Registry value name:  
*AuditPath*  
Registry value type:  
*String*
3. From that directory run **OraLogMonitor.exe** program from the command line with **-install** parameter, which will install DB Audit Event Log Monitoring Service. For example,

```
C:\Windows> OraLogMonitor.exe - install
```

## Oracle for non-Windows Platforms

DB Audit doesn't require additional files to be installed on the server.


## Microsoft SQL Server

Read the following if you are going to use system auditing with SQL Server. You can skip this

topic if you are not going to run system auditing.

**You must perform the following installation step:**

1. From the DB Audit API installation directory, from the **server\_side\_files/mssql** subdirectory, copy **xp\_dbaudit.dll** file to your SQL Server computer to **[SQL Server home]\Binn** subdirectory.

 **Tip:** If you are not sure from which directory your SQL Server instance is started you can find out this in the properties of the SQL Server service. Open Services applet from Administrative Programs group in the Windows Control Panel. Locate the required SQL Server instance in the services list. The service name for the default instance is usually "MSSQLSERVER." Double-click the service name to open service properties dialog. The **Path to Executable** is displayed below the service name and description. This is the directory where you need to copy **xp\_dbaudit.dll** file.

## Sybase ASE and ASA

DB Audit doesn't require additional files to be installed on the server.



# CHAPTER 3, Database Connectivity

## Profile Manager Class and Related Classes

### Overview

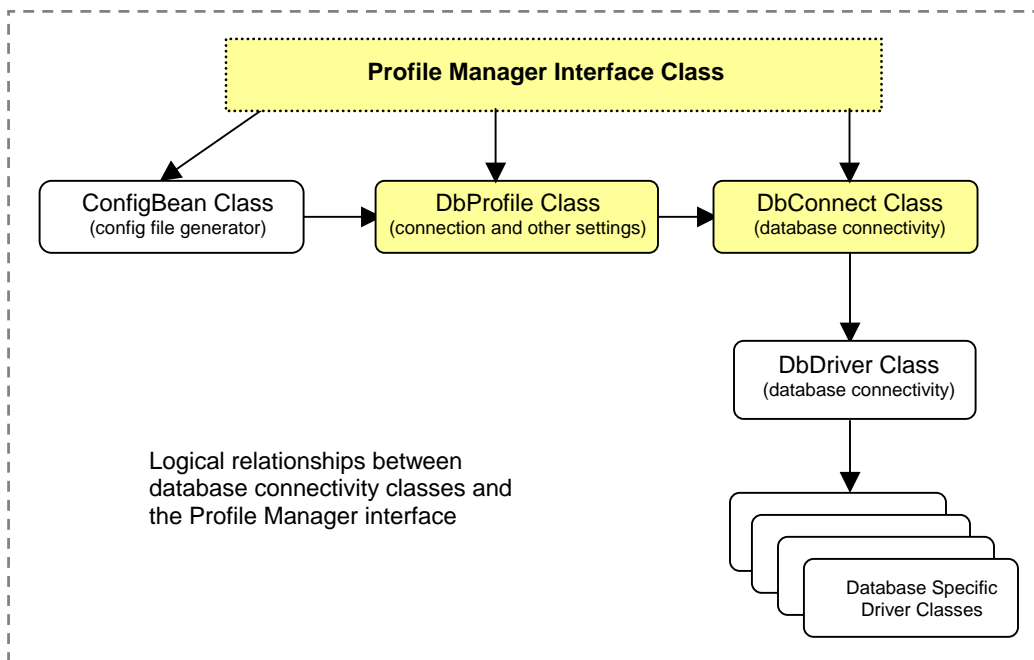
The **ProfileManager** class provides methods for configuring database driver settings and managing connection profiles. The full class name is **com.softtreetech.com.dbaudit.ProfileManager** for the class declaration and **com.softtreetech.com.dbaudit.ProfileManagerImpl** for the class implementation.

The class must be instantiated and initialized before it can be used to manage connection profiles. Use **getInstance** method to get the default instance with default configuration parameters or use **initInstance** method to initialize the profile manager interface using custom profile path. Once you have obtained an instance of the **ProfileManager**, you can add/delete/update drivers and profiles in the profiles configuration file. You can also use it to create an instance of the **DbConnectImpl** class and then use it to connect to a database.

Examples:

```
ProfileManager profileManager = ProfileManagerImpl.getInstance();
```

```
ProfileManager profileManager = ProfileManagerImpl.initInstance("/home/myapp/profiles.xml");
```



The profile manager uses several auxiliary classes for storing profile settings and managing connections. The highlighted classes provide database connectivity methods and are used with direct Java interface or RMI-based Java interface. Note that the DB Audit command line interface hides all internal class complexity and automatically establishes database connections as required.

**ConfigBean** – this class implements configuration bean and is used internally to read, write and parse the profiles configuration file. The **DefaultConfigGenerator** class can be used to generate

the default configuration file containing 4 pre-configured database drivers and 1 sample connection profile for each supported DBMS type. For detailed description of this class methods and properties read [Default Configuration Generator Class](#) topic.

**DbDriver** – this class implements JDBC driver interface including driver type, driver search path, and the name of the driver class that is invoked when the driver is loaded. For detailed description of this class methods and properties read [Database Driver Class](#) topic.

**DbProfile** – this class is used internally to hold database connection profile properties and some additional parameters associated with the database connection. Profile properties include but not limited to the following: profile name, database URL, driver used, user name, password, repository database name, connection type, DB2 system path, email server name for sending alerts, email account and password for sending alerts. For detailed description of this class methods and properties read [Database Connection Profile Class](#) topic.

## Instantiating Profile Manager Class

### *getInstance*

```
public ProfileManager getInstance()  
    throws java.io.IOException
```

Description:

Instantiates and initializes the profile manager using default configuration.

Parameters:

None

Return:

ProfileManager object

Throws:

java.lang.IOException – if the default configuration file cannot be found or accessed.

Example:

```
ProfileManager profileManager = ProfileManagerImpl.getInstance();
```

### *initInstance*

```
public ProfileManager initInstance(String pathToProfilesXML)  
    throws java.io.IOException
```

Description:

Instantiates and initializes the profile manager using custom path to the configuration file.

Parameters:

pathToProfilesXML – full path to the profiles configuration file.

Return:

ProfileManager object

Throws:

java.lang.IOException – if the specified configuration file cannot be found or accessed.

Example:

```
ProfileManager profileManager = ProfileManagerImpl.getInstance("/home/myapp/profiles.xml");
```

## Database Driver Class

Database driver class **DbDriver** encapsulates properties and methods describing type and location of database client libraries used by DB Audit for communicating with the database. The following useful properties and methods are available in this class:

### Constructor

```
public DbDriver()
```

Description:

Creates new instance of DbDriver class.

Parameters:


None

### getName

```
public java.lang.String getName()
```

Description:

Returns descriptive driver name, for example, "Oracle".

 **Tip:** do not confuse driver name with driver type. Drivers are available from many vendors and their names could be really anything, not necessarily matching name of the type of the database system they are used for. Driver names often contain driver type and version numbers.

Parameters:

None

Return:

String description of driver name, for example, "Oracle".

Example:

1. Create ProfileManager instance using default configuration file and obtain list of registered drivers:

```
ProfileManager profileManager = ProfileManagerImpl.getInstance();  
java.util.List driverList = profileManager.getDbDriverList();
```

2. Print name of each registered driver to the standard output:


```
for (int i = 0; i < driverList.size(); i++)  
    System.out.println(driverList.get(i).getName());
```

**setName**

```
public void setName(java.lang.String name)
```

Description:

Sets descriptive driver name, for example, "Oracle".

 **Tip:** do not confuse driver name with driver type. Drivers are available from many vendors and their names could be really anything, not necessarily matching name of the type of the database system they are used for. Driver names often contain driver type and version numbers.

Parameters:

Name – driver name.

Return:

None

Example:

Create a new instance of DbDriver class, set driver properties and register it with the ProfileManager:

```
DbDriver driver = newDbDriver();
driver.setName("My DB2 driver version 1.4");
driver.setJdbcDriverType(DbDriver.DB2_TYPE);
driver.setJdbcDriverPath("db2/db2jcc.jar;db2/db2jcc_license_cu.jar");
driver.setJdbcDriverClass("com.ibm.db2.jcc.DB2Driver");
```


```
ProfileManager profileManager = ProfileManagerImpl.getInstance();
profileManager.addDbDriver(driver);
```

**getJdbcDriverPath**

```
public java.lang.String getJdbcDriverPath()
```

Description:

Returns path to driver libraries.

 **Tip:** A driver path can consists of one or more libraries – JAR files. Path can be specified as an absolute path or as a path relative to the location of **dbaudit.jar** file. In case of multiple files, the path values must be separated by a semicolon, for example:

```
db2/db2jcc.jar;db2/db2jcc_license_cu.jar
```

Parameters:

None

Return:

Semicolon-separated names of driver libraries including their paths

Example:

1. Create ProfileManager instance using default configuration file and obtain list of registered drivers:

```
ProfileManager profileManager = ProfileManagerImpl.getInstance();
java.util.List driverList = profileManager.getDbDriverList();
```

2. Print path of each registered driver to the standard output:


```
for (int i = 0; i < driverList.size(); i++)
    System.out.println(driverList.get(i).getJdbcDriverPath());
```

### **setJdbcDriverPath**

public void **setJdbcDriverPath**(java.lang.String jdbcDriverPath)

Description:

Sets path to driver libraries.

 **Tip:** A driver path can consists of one or more libraries – JAR files. Path can be specified as an absolute path or as a path relative to the location of dbaudit.jar file. In case of multiple files, the path values must be separated by a semicolon, for example:

```
db2/db2jcc.jar;db2/db2jcc_license_cu.jar
```

Parameters:

jdbcDriverPath – Semicolon-separated names of driver libraries including their path..

Return:

None

Example:

Create a new instance of DbDriver class, set driver properties and register it with the ProfileManager:

```
DbDriver driver = newDbDriver();
driver.setName("My DB2 driver version 1.4");
driver.setJdbcDriverType(DbDriver.DB2_TYPE);
driver.setJdbcDriverPath("db2/db2jcc.jar;db2/db2jcc_license_cu.jar");
driver.setJdbcDriverClass("com.ibm.db2.jcc.DB2Driver");
```

```
ProfileManager profileManager = ProfileManagerImpl.getInstance();
profileManager.addDbDriver(driver);
```

### **getJdbcDriverClass**

public java.lang.String **getJdbcDriverClass**()

Description:

Returns name of the main driver class used as an interface class for the JDBC compliant database driver.

Parameters:

None

Return:

Name of the main driver class.

Example:

1. Create ProfileManager instance using default configuration file and obtain list of registered drivers:
 

```
ProfileManager profileManager = ProfileManagerImpl.getInstance();
java.util.List driverList = profileManager.getDbDriverList();
```
2. Print name of main class of each registered driver to the standard output:
 

```
for (int i = 0; i < driverList.size(); i++)
    System.out.println(driverList.get(i).getJdbcDriverClass());
```

### **setJdbcDriverClass**

public void **setJdbcDriverClass**(java.lang.String jdbcDriverClass)

Description:

Sets name of the main driver class which can be used as an interface class for the JDBC compliant database driver.

Parameters:

Name of the main driver class.

Return:

None

Example:

Create a new instance of DbDriver class, set driver properties and register it with the ProfileManager:

```
DbDriver driver = newDbDriver();
driver.setName("My DB2 driver version 1.4");
driver.setJdbcDriverType(DbDriver.DB2_TYPE);
driver.setJdbcDriverPath("db2/db2jcc.jar;db2/db2jcc_license_cu.jar");
driver.setJdbcDriverClass("com.ibm.db2.jcc.DB2Driver");

ProfileManager profileManager = ProfileManagerImpl.getInstance();
profileManager.addDbDriver(driver);
```

### **getJdbcDriverType**

public int getJdbcDriverType()

Description:

Returns type of the database driver.

Parameters:

None

Return:

Integer value whose value matches one of the following constants pre-defined in the **DbDriver** class:

- **DbDriver.DB2\_TYPE**
- **DbDriver.MSSQL\_TYPE**
- **DbDriver.ORACLE\_TYPE**

- **DbDriver.SYBASE\_TYPE**

Example:

1. Create ProfileManager instance using default configuration file and obtain list of registered drivers:
 

```
ProfileManager profileManager = ProfileManagerImpl.getInstance();
java.util.List driverList = profileManager.getDbDriverList();
```
2. Print type of each registered driver to the standard output:
 

```
for (int i = 0; i < driverList.size(); i++)
    System.out.println(Integer.toString(driverList.get(i).getJdbcDriverType()));
```

### **setJdbcDriverType**

public void **setJdbcDriverType**(int jdbcDriverType)

Description:

Sets type of the database driver.

Parameters:

Type value that must be one of the following constants pre-defined in the **DbDriver** class:

- **DbDriver.DB2\_TYPE**
- **DbDriver.MSSQL\_TYPE**
- **DbDriver.ORACLE\_TYPE**
- **DbDriver.SYBASE\_TYPE**

Return:

None

Example:

Create a new instance of DbDriver class, set driver properties and register it with the ProfileManager:

```
DbDriver driver = new DbDriver();
driver.setName("My DB2 driver version 1.4");
driver.setJdbcDriverType(DbDriver.DB2_TYPE);
driver.setJdbcDriverPath("db2/db2jcc.jar;db2/db2jcc_license_cu.jar");
driver.setJdbcDriverClass("com.ibm.db2.jcc.DB2Driver");

ProfileManager profileManager = ProfileManagerImpl.getInstance();
profileManager.addDbDriver(driver);
```

## **Managing Database Drivers**

DB Audit API comes with 4 drivers pre-configured – 1 for each supported database type. In case if you want to use a non-default driver or have a more recent version of a database driver available you can use the following methods to manage existing and register new drivers.

### ***getDbDriverList***

```
public java.util.List getDbDriverList()
    throws java.io.IOException
```

Description:

Returns alphabetically sorted list of drivers from profiles.xml file.

Parameters:

None.

Return:

List of `DbDriver` objects sorted by driver name

Throws:

`java.lang.IOException` – if the configuration file cannot be accessed.

Example:

1. Create `ProfileManager` instance using default configuration file and obtain list of registered drivers:  

```
ProfileManager profileManager = ProfileManagerImpl.getInstance();
java.util.List driverList = profileManager.getDbDriverList();
```
2. Print name of each registered driver to the standard output:  

```
for (int i = 0; i < driverList.size(); i++)
    System.out.println(driverList.get(i).getName());
```

### ***getDbDriverNames***

```
public String[] getDbDriverNames()
    throws java.io.IOException
```

Description:

Returns string array of database driver names from profiles.xml file.

Parameters:

None

Return:

Array of driver names

Throws:

`java.lang.IOException` – if the configuration file cannot be accessed.

Example:

1. Create `ProfileManager` instance using default configuration file and obtain list of registered driver names:  

```
ProfileManager profileManager = ProfileManagerImpl.getInstance();
String name[] = profileManager.getDbDriverNames();
```
2. Print name of each registered driver to the standard output:  

```
for (int i = 0; i < name.length; i++)
    System.out.println(name(i));
```



**getDbDriver**

```
public DbDriver getDbDriver(java.lang.String driverName)
    throws java.io.IOException
```

Description:

Finds driver information by driver name

Parameters:

driverName – driver name as it is specified in profiles.xml file, for example, "Oracle"

Return:

DbDriver object or null if not found

Throws:

java.lang.IOException – if the configuration file cannot be accessed.

Example:

Create ProfileManager instance using default configuration file and obtain instance of Oracle driver class:

```
ProfileManager profileManager = ProfileManagerImpl.getInstance();
DbDriver driver = profileManager.getDbDriver ("Oracle");
```

**addDbDriver**

```
public void addDbDriver(DbDriver driver)
    throws java.io.IOException
```

Description:

Registers new JDBC driver or updates registration of an existing one whose name matches.

Parameters:

driver – object of DbDriver class whose properties describe JDBC driver interface.

Return:

None

Throws:

java.lang.IOException – if the configuration file cannot be accessed.

Usage:

To add a new driver to the configuration file, first create a new DbDriver object, for example, *DbDriver driver = new DbDriver()* then call setters of the created DbDriver object to populate its properties. After that you can call addDbDriver method to add the prepared driver to the configuration.

Example:

Create a new instance of DbDriver class, set driver properties and register it with the ProfileManager:

```
DbDriver driver = newDbDriver();
driver.setName("My DB2 driver version 1.4");
driver.setJdbcDriverType(DbDriver.DB2_TYPE);
driver.setJdbcDriverPath("db2/db2jcc.jar;db2/db2jcc_license_cu.jar");
driver.setJdbcDriverClass("com.ibm.db2.jcc.DB2Driver");

ProfileManager profileManager = ProfileManagerImpl.getInstance();
profileManager.addDbDriver(driver);
```

### **deleteDbDriver**

```
public void deleteDbDriver(java.lang.String driverName)
    throws java.io.IOException
```

Description:

Deletes driver registration from the profiles configuration file.

Parameters:

driverName – driver name, for example, "Oracle"

Return:

None

Throws:

java.lang.IOException – if the configuration file cannot be accessed or written.

Example:

Create ProfileManager instance using default configuration file and unregister database driver who name is My Oracle Driver:

```
ProfileManager profileManager = ProfileManagerImpl.getInstance();
profileManager.deleteDbDriver("My Oracle Driver");
```

## **Database Connection Profile Class**

### **Constructor**

```
public DbProfile()
```

Description:

Creates new instance of DbProfile class.

Parameters:

None

**getName**

```
public java.lang.String getName()
```

## Description:

Returns profile name, for example, "DataWare Server Connection"

## Parameters:

None

## Return:

Profile name

## Example:

1. Create ProfileManager instance using default configuration file and obtain list of configured database connection profiles:
 

```
ProfileManager profileManager = ProfileManagerImpl.getInstance();
java.util.List profileList = profileManager.getDbProfileList();
```
2. Print name of each configured profile to the standard output:
 


```
for (int i = 0; i < profileList.size(); i++)
    System.out.println(profileList.get(i).getName());
```

**setName**

```
public void setName(java.lang.String name)
```

## Description:

Sets profile name, for example, " DataWare Server Connection".

 **Tip:** do not confuse profile name with driver name or driver type. Profile name is simply a descriptive name attached to a set of database connection properties, which you can use later to refer to a particular database connection.

## Parameters:

name – profile name.

## Return:

None

## Example:

Create a new instance of DbProfile class, set profile properties and register it with the ProfileManager:

```
DbProfile profile = newDbProfile();
profile.setName("DataWare Server Connection");
profile.setDriverName("OracleDriver");
profile.setJdbcUrl("jdbc:oracle:thin:@MY_BIG_SERVER:1521:ORCL");
profile.setMailServer("smtp.my-email-server.com");
profile.setMailSender("db_audit@my-company.com");

ProfileManager profileManager = ProfileManagerImpl.getInstance();
profileManager.addDbProfile(profile);
```

**getDriverName**

```
public java.lang.String getDriverName()
```

## Description:

Returns name of registered database driver associated with the specified connection profile. See [Managing Database Drivers](#) topic for more details.

## Parameters:

None

## Return:

Driver name

## Example:

1. Create ProfileManager instance using default configuration file and obtain list of configured database connection profiles:  

```
ProfileManager profileManager = ProfileManagerImpl.getInstance();  
java.util.List profileList = profileManager.getDbProfileList();
```
2. Print driver name of each configured profile to the standard output:  

```
for (int i = 0; i < profileList.size(); i++)  
    System.out.println(profileList.get(i).getDriverName());
```

**setDriverName**

```
public void setDriverName(java.lang.String driverName)
```

## Description:

Associates previously registered database driver with this profile.

 **Tip:** do not confuse profile name with driver name or driver type.

## Parameters:

driverName – JDBC driver name. This is the name used with the driver registration. If you use default configuration file, driver name can be either of the following: OracleDriver, Db2Driver, MssqlDriver, SybaseDriver.

## Return:

None

Example:

Create a new instance of DbProfile class, set profile properties and register it with the ProfileManager:

```
DbProfile profile = newDbProfile();
profile.setName("DataWare Server Connection");
profile.setDriverName("OracleDriver");
profile.setJdbcUrl("jdbc:oracle:thin:@MY_BIG_SERVER:1521:ORCL");
profile.setMailServer("smtp.my-email-server.com");
profile.setMailSender("db_audit@my-company.com");

ProfileManager profileManager = ProfileManagerImpl.getInstance();
profileManager.addDbProfile(profile);
```

### **getJdbcURL**

```
public java.lang.String getJdbcURL()
```

Description:

Returns database server connection URL database driver with this profile.



**Tip:** do not confuse profile name with driver name or driver type.

Parameters:

None

Return:

Database connection string in URL format

Example:

1. Create ProfileManager instance using default configuration file and obtain list of configured database connection profiles:

```
ProfileManager profileManager = ProfileManagerImpl.getInstance();
java.util.List profileList = profileManager.getDbProfileList();
```

2. Print database connection URL of each configured profile to the standard output:

```
for (int i = 0; i < profileList.size(); i++)
    System.out.println(profileList.get(i).getJdbcURL());
```

### **setJdbcURL**

```
public void setJdbcURL(java.lang.String jdbcURL)
```

Description:

Sets connection string in JDBC URL format.

Parameters:

jdbcURL – Database connection URL string



**Tip:** URL strings are DBMS specific and differ for different drivers, database systems and communication protocols. Check your JDBC driver documentation for what to put in the JDBC URL. Use the following formats for TCP/IP based connections using pre-configured JDBC drivers

packaged with DB Audit API:

```
jdbc:db2://[DB2_server_name]:[port(default – 50000)]/[database_name]
jdbc:sqlserver://[MSSQL_server_name]:[port(default – 1433)]
jdbc:oracle:thin:@[Oracle_server_name]:[port(default – 1521)]:[sid]
jdbc:sybase:Tds:[Sybase_server_name]:[port(default – 2048)]/[database_name]
```

See [Database Profiles and Connections Setup](#) topic in CHAPTER 1 for more information and for examples for each supported DBMS type.

Return:

None

Example:

Create a new instance of DbProfile class, set profile properties and register it with the ProfileManager:

```
DbProfile profile = newDbProfile();
profile.setName("DataWare Server Connection");
profile.setDriverName("OracleDriver");
profile.setJdbcUrl("jdbc:oracle:thin:@MY_BIG_SERVER:1521:ORCL");
profile.setMailServer("smtp.my-email-server.com");
profile.setMailSender("db_audit@my-company.com");

ProfileManager profileManager = ProfileManagerImpl.getInstance();
profileManager.addDbProfile(profile);
```

## **getUsername**

```
public java.lang.String getUsername()
```

Description:

Returns optional user name saved with the profile.

Parameters:

None

Return:

Connection user name if one is saved with the profile or null otherwise.

Example:

1. Create ProfileManager instance using default configuration file and obtain list of configured database connection profiles:
 

```
ProfileManager profileManager = ProfileManagerImpl.getInstance();
java.util.List profileList = profileManager.getDbProfileList();
```
2. Print user name saved with each configured profile:
 

```
for (int i = 0; i < profileList.size(); i++)
    if (profileList.get(i).getName() == null)
        System.out.println("<User name not saved>");
    else
        System.out.println("User: " + profileList.get(i).getName());
```

**setUsername**

```
public void setUsername(java.lang.String username)
```

## Description:

Sets connection user name. Saving connection names and passwords with a profile is optional. If saved, connections can be later established using the simple "connect" method without additional parameters and users don't need to know them. If not saved, they can be provided by users later at the time of connection, perhaps using some interactive prompt for the user and password values.

## Parameters:

username – User name to be used for the database connection.

## Return:

None

## Example:

Create a new instance of DbProfile class, set profile properties and register it with the ProfileManager:

```
DbProfile profile = newDbProfile();
profile.setName("DataWare Server Connection");
profile.setDriverName("OracleDriver");
profile.setJdbcUrl("jdbc:oracle:thin:@MY_BIG_SERVER:1521:ORCL");
profile.setUsername("auditer");
profile.setPassword("secret");
profile.setMailServer("smtp.my-email-server.com");
profile.setMailSender("db_audit@my-company.com");

ProfileManager profileManager = ProfileManagerImpl.getInstance();
profileManager.addDbProfile(profile);
```

## Example 2:

Create a new instance of DbProfile class, obtain profile for "DATAWARE" database and update user and password values for that profile:

```
ProfileManager profileManager = ProfileManagerImpl.getInstance();
DbProfile profile = profileManager.getDbProfile("Dataaware");

profile.setUsername("auditer");
profile.setPassword("secret");

profileManager.deleteDbProfile("Dataaware");
profileManager.deleteDbProfile(profile);
```

**getPassword**

```
public java.lang.String getPassword()
```

## Description:

Returns optional password saved with the profile.

**Parameters:**

None

**Return:**

Connection password if one is saved with the profile or null otherwise.

**Example:**

1. Create ProfileManager instance using default configuration file and obtain list of configured database connection profiles:

```
ProfileManager profileManager = ProfileManagerImpl.getInstance();
java.util.List profileList = profileManager.getDbProfileList();
```

2. Print password saved with each configured profile:

```
for (int i = 0; i < profileList.size(); i++)
    if (profileList.get(i).getPassword() == null)
        System.out.println("<Password not saved>");
    else
        System.out.println("Password: " + profileList.get(i).getPassword());
```

***setPassword***

```
public void setPassword(java.lang.String password)
```

**Description:**

Sets connection password. Saving connection names and passwords with a profile is optional. If saved, connections can be later established using the simple "connect" method without additional parameters and users don't need to know them when running your application. If not saved, they can be provided by users later at the time of connection, perhaps using some interactive prompt for the user and password values.

**Parameters:**

password – Password to be used for the database connection.

**Return:**

None

**Example:**

Create a new instance of DbProfile class, set profile properties and register it with the ProfileManager:

```
DbProfile profile = newDbProfile();
profile.setName("DataWare Server Connection");
profile.setDriverName("OracleDriver");
profile.setJdbcUrl("jdbc:oracle:thin:@MY_BIG_SERVER:1521:ORCL");
profile.setUsername("auditer");
profile.setPassword("secret");
profile.setMailServer("smtp.my-email-server.com");
profile.setMailSender("db_audit@my-company.com");

ProfileManager profileManager = ProfileManagerImpl.getInstance();
profileManager.addDbProfile(profile);
```



## Example 2:

Create a new instance of DbProfile class, obtain profile for "Dataware" database and update user and password values for that profile:

```
ProfileManager profileManager = ProfileManagerImpl.getInstance();
DbProfile profile = profileManager.getDbProfile("Dataware");

profile.setUsername("auditer");
profile.setPassword("secret");

profileManager.deleteDbProfile("Dataware");
profileManager.deleteDbProfile(profile);
```

**getRepositoryDatabase**

```
public java.lang.String getRepositoryDatabase()
```

## Description:

Returns name of the repository database used by auditing processes running in the database server for which the profile is used.



**SQL Server, ASE:** This value is only applicable and required for SQL Server and Sybase ASE systems. In ASE it is only required for data-change auditing. In ASE system auditing processes always use **sybsecurity** database.



**Important Note:** This returns the name saved using setRepositoryDatabase method. The current profile settings can be different from effective settings used in previously installed audit procedures.

## Parameters:

None

## Return:

Name of the repository database as it is saved with the profile.

## Example:

1. Create ProfileManager instance using default configuration file and obtain list of configured database connection profiles:

```
ProfileManager profileManager = ProfileManagerImpl.getInstance();
java.util.List profileList = profileManager.getDbProfileList();
```

2. Print password saved with each configured profile:

```
for (int i = 0; i < profileList.size(); i++)
    if (profileList.get(i).getName() == null)
        System.out.println("<Password not saved>");
    else
        System.out.println("Password: " + profileList.get(i).getName());
```

**setRepositoryDatabase**


```
public void setRepositoryDatabase(java.lang.String repositoryDatabase)
```

**Description:**

Saves in the profile settings name of the repository database used by auditing processes.

**Parameters:**

repositoryDatabase – Name of repository database used to store local audit trail data for the database server pointed by this profile.

 **SQL Server, ASE:** This value is only applicable and required for SQL Server and Sybase ASE systems. In ASE it is only required for data-change auditing. In ASE system auditing processes always use **sybsecurity** database.

**Return:**

None

**Example:**

Create a new instance of DbProfile class, set profile properties and register it with the ProfileManager:

```
DbProfile profile = newDbProfile();
profile.setName("DataWare Server Connection");
profile.setDriverName("MssqlDriver");
profile.setJdbcUrl("jdbc:sqlserver://MY_BIG_SERVER:1433");
profile.setRepositoryDatabase("auditdb");
profile.setMailServer("smtp.my-email-server.com");
profile.setMailSender("db_audit@my-company.com");


ProfileManager profileManager = ProfileManagerImpl.getInstance();
profileManager.addDbProfile(profile);
```

**getConnectionParameter**

```
public java.lang.String getConnectionParameter()
```

**Description:**

Returns additional connection parameters associated with the profile.

 **Oracle:** The only connection parameter supported at this time is SYSDBA connection type that can be optionally used with Oracle connections.

**Parameters:**

None

**Return:**

Additional connection parameters if available or null otherwise

**Example:**

1. Create ProfileManager instance using default configuration file and obtain list of configured database connection profiles:
 

```
ProfileManager profileManager = ProfileManagerImpl.getInstance();
java.util.List profileList = profileManager.getDbProfileList();
```

2. Print connection parameters saved with each configured profile:


```
for (int i = 0; i < profileList.size(); i++)
    if (profileList.get(i).getConnectionParameter() == null)
        System.out.println("<Not available>");
    else
        System.out.println(profileList.get(i).getConnectionParameter());
```

### **setConnectionParameter**

```
public void setConnectionParameter(java.lang.String connectionParameter)
```

#### Description:

Sets additional connection parameters associated with the profile.

 **Oracle:** The only connection parameter supported at this time is SYSDBA connection type that can be optionally used with Oracle connections.

#### Parameters:

connectionParameter – connection parameter controlling connection type, one of the following:

- null – this is the same as Normal
- Normal
- SYSDBA
- SYSOPER

#### Return:

None

#### Example:

Create a new instance of DbProfile class, set profile properties and register it with the ProfileManager:

```
DbProfile profile = new DbProfile();
profile.setName("Oracle Server Connection");
profile.setDriverName("OracleDriver");
profile.setJdbcUrl("jdbc:oracle:thin:@MY_BIG_SERVER:1521:ORCL");
profile.setUsername("sys");
profile.setPassword("secret");
profile.setConnectionParameter("SYSDBA");
profile.setMailServer("smtp.my-email-server.com");
profile.setMailSender("db_audit@my-company.com");

ProfileManager profileManager = ProfileManagerImpl.getInstance();
profileManager.addDbProfile(profile);
```

### **isLoggedOnAsSYSDBA**

```
public boolean isLoggedOnAsSYSDBA()
```

**Description:**

Returns effective settings for Oracle connection established using an instance of the DBProfile class.

**Parameters:**

None

**Return:**

Returns true if the profile currently connected to an Oracle database using SYSDBA connection type or false otherwise.

**Example:**

```
DbProfile profile = ... obtain profile referenbce ...;
if (profile.isLoggedOnAsSYSDBA())
    ... perform some action here ...;
else
    System.out.println("Must be connection as SYSDBA to perform this action");
```

**getDbPath**

```
public java.lang.String getDbPath()
```

**Description:**

Returns path to the database server SQLLIB directory associated with the profile. The value returned is the one previously set using [setDbPath](#) method.



**DB2:** This method is only applicable to DB2 connections.

**Parameters:**

None

**Return:**

DB2's path to SQLLIB directory if previously set using setDBPath method or null otherwise

**Example:**

1. Create ProfileManager instance using default configuration file and obtain list of configured database connection profiles:

```
ProfileManager profileManager = ProfileManagerImpl.getInstance();
java.util.List profileList = profileManager.getDbProfileList();
```

2. Print DB2 path saved with each configured DB2 profile:

```
for (int i = 0; i < profileList.size(); i++)
{
    DbDriver driver = profileManeger.getDbDriver(profileList.get(i).getDriverName());
    if (driver.getJdbcDriverType != DbDriver.DB2_TYPE)
        System.out.println("<Not applicable to this database type>");
    else
        if (profileList.get(i).getDbPath() == null)
            System.out.println("<Path not specified>");
        else
            System.out.println(profileList.get(i).getDbPath());
}
```

**setDbPath**

```
public void setDbPath(java.lang.String dbPath)
```

## Description:

Saves in the profile settings path to the DB2 SQLLIB directory on the DB2 server.

## Parameters:

dbPath – path to the DB2 SQLLIB directory.



**DB2:** This method is only applicable to DB2 connections.

## Return:

None

## Example:

Create a new instance of DbProfile class, set profile properties and register it with the ProfileManager:

```
DbProfile profile = newDbProfile();
profile.setName("DataWare Server Connection");
profile.setDriverName("Db2Driver");
profile.setJdbcUrl("jdbc:db2://MY_BIG_SERVER:5000:TOOLS");
profile.setDbPath("/home/db2/v8/sqllib");
profile.setMailServer("smtp.my-email-server.com");
profile.setMailSender("db_audit@my-company.com");

ProfileManager profileManager = ProfileManagerImpl.getInstance();
profileManager.addDbProfile(profile);
```

**getMailSender**

```
public java.lang.String getMailSender()
```

## Description:

Retrieves email address of the account used in DB Audit email sending procedures and alerts. This is the account from which alert emails will be originated. Returned value is the one saved in the profile settings using [setMailSender](#) method.



**Important Note:** This method retrieves current profile settings. These settings can be different from effective email settings used in previously installed email procedures.

## Parameters:

None

## Return:

Email address or account name if previously set using setMailSender method or null otherwise.

## Example:

1. Create ProfileManager instance using default configuration file and obtain list of configured database connection profiles:
 

```
ProfileManager profileManager = ProfileManagerImpl.getInstance();
java.util.List profileList = profileManager.getDbProfileList();
```

2. Print mail sender address saved with each configured profile:
 

```
for (int i = 0; i < profileList.size(); i++)
    if (profileList.get(i).getMailSender() == null)
        System.out.println("<Email sender not saved>");
    else
        System.out.println("Email sender: " + profileList.get(i).getMailSender());
```

### **setMailSender**

```
public void setMailSender(java.lang.String mailSender)
```

#### Description:

Saves in the profile settings email address of the account for use in DB Audit email sending procedures and alerts. This is the account from which alert emails will be originated.



**Important Note:** This method simply updates the profile settings. In order to update effective email settings in already installed email procedures you must execute the **installDBAuditMailSendingSQLProcedure** method which is described in [CHAPTER 4, System Audit Management](#)

#### Parameters:

mailSender – email address or name recognized by your email server.

#### Return:

None

#### Example:

Create a new instance of DbProfile class, set profile properties and register it with the ProfileManager:

```
DbProfile profile = newDbProfile();
profile.setName("DataWare Server Connection");
profile.setDriverName("OracleDriver");
profile.setJdbcUrl("jdbc:oracle:thin:@MY_BIG_SERVER:1521:ORCL");
profile.setMailServer("smtp.my-email-server.com");
profile.setMailSender("db_audit@my-company.com");

ProfileManager profileManager = ProfileManagerImpl.getInstance();
profileManager.addDbProfile(profile);
```

### **getMailServer**

```
public java.lang.String getMailServer()
```

#### Description:

Retrieves name or IP address of the email server as it was saved in the profile settings using [setMailServer](#) method.



**Important Note:** This method retrieves current profile settings. These settings can be different from effective email settings used in previously installed email procedures.

#### Parameters:

mailServer – network computer name or IP address of your email server.

Return:

None

Example:

Create a new instance of DbProfile class, set profile properties and register it with the ProfileManager:

```
DbProfile profile = newDbProfile();
profile.setName("DataWare Server Connection");
profile.setDriverName("OracleDriver");
profile.setJdbcUrl("jdbc:oracle:thin:@MY_BIG_SERVER:1521:ORCL");
profile.setMailServer("smtp.my-email-server.com");
profile.setMailSender("db_audit@my-company.com");
```

```
ProfileManager profileManager = ProfileManagerImpl.getInstance();
profileManager.addDbProfile(profile);
```

### ***setMailServer***

```
public void setMailServer(java.lang.String mailServer)
```

Description:

Saves in the profile settings name or IP address of the email server for use in DB Audit email sending procedures and alerts.



**Important Note:** This method simply updates the profile settings. In order to update effective email settings in already installed email procedures you must execute the **installDBAuditMailSendingSQLProcedure** method which is described in [CHAPTER 4, System Audit Management](#)

Parameters:

mailServer – network computer name or IP address of your email server.

Return:

None

Example:

Create a new instance of DbProfile class, set profile properties and register it with the ProfileManager:

```
DbProfile profile = newDbProfile();
profile.setName("DataWare Server Connection");
profile.setDriverName("OracleDriver");
profile.setJdbcUrl("jdbc:oracle:thin:@MY_BIG_SERVER:1521:ORCL");
profile.setMailServer("smtp.my-email-server.com");
profile.setMailSender("db_audit@my-company.com");
```

```
ProfileManager profileManager = ProfileManagerImpl.getInstance();
profileManager.addDbProfile(profile);
```

### ***getMailPassword***

```
public java.lang.String getMailPassword()
```

## Description:

Retrieves email password saved in the profile settings using [setMailPassword](#) method.



**Important Note:** This method retrieves current profile settings. These settings can be different from effective email settings used in previously installed email procedures.



**Tip:** Email password is typically not required when using email server in the same network domain.

## Parameters:

None

## Return:

Email password if previously set using `setMailPassword` method or null otherwise.

## Example:

1. Create ProfileManager instance using default configuration file and obtain list of configured database connection profiles:

```
ProfileManager profileManager = ProfileManagerImpl.getInstance();
java.util.List profileList = profileManager.getDbProfileList();
```

2. Print mail password saved with each configured profile:

```
for (int i = 0; i < profileList.size(); i++)
    if (profileList.get(i).getMailPassword() == null)
        System.out.println("<Email password not saved>");
    else
        System.out.println("Email password: " + profileList.get(i).getMailPassword());
```

**setMailPassword**

```
public void setMailPassword(java.lang.String mailPassword)
```

## Description:

Saves in the profile settings email password of the account for use in DB Audit email sending procedures and alerts. This is the account from which alert emails will be originated.



**Tip:** Email password is typically not required when using email server in the same network domain.



**Important Note:** This method simply updates the profile settings. In order to update effective email settings in already installed email procedures you must execute the **installDBAuditMailSendingSQLProcedure** method which is described in [CHAPTER 4, System Audit Management](#)

## Parameters:

mailPassword – email password required by your email server for email sender authentication.

## Return:

None



Example:

Create a new instance of DbProfile class, set profile properties and register it with the ProfileManager:

```
DbProfile profile = new DbProfile();
profile.setName("DataWare Server Connection");
profile.setDriverName("OracleDriver");
profile.setJdbcUrl("jdbc:oracle:thin:@MY_BIG_SERVER:1521:ORCL");
profile.setMailServer("smtp.my-email-server.com");
profile.setMailSender("db_audit@my-company.com");
profile.setMailPassword("secret")

ProfileManager profileManager = ProfileManagerImpl.getInstance();
profileManager.addDbProfile(profile);
```

## Managing and Using Database Profiles

### ***getDbProfileList***

```
public java.util.List getDbProfileList()
    throws java.io.IOException
```

Description:

Returns alphabetically sorted list of database profiles from profiles.xml file.

Parameters:

None.

Return:

List of DbProfile objects sorted by profile name

Throws:

java.lang.IOException – if the configuration file cannot be accessed.

Example:

1. Create ProfileManager instance using default configuration file and obtain list of configured database connection profiles:
 

```
ProfileManager profileManager = ProfileManagerImpl.getInstance();
java.util.List profileList = profileManager.getDbProfileList();
```
2. Print name of each configured profile to the standard output:
 

```
for (int i = 0; i < profileList.size(); i++)
    System.out.println(profileList.get(i).getName());
```

### ***getDbProfileNames***

```
public String[] getDbProfileNames()
    throws java.io.IOException
```

Description:

Returns string array of database profile names from profiles.xml file.

Parameters:

None.

Return:

Array of profile names

Throws:

java.lang.IOException – if the configuration file cannot be accessed.

Example:

1. Create ProfileManager instance using default configuration file and obtain names of configured database connection profiles:  

```
ProfileManager profileManager = ProfileManagerImpl.getInstance();  
String name[] = profileManager.getDbProfileNames();
```
2. Print name of each configured profile to the standard output:  

```
for (int i = 0; i < name.length; i++) System.out.println(name[i]);
```

### **getDbProfile**

```
public DbProfile getDbProfile(java.lang.String profileName)  
    throws java.io.IOException
```

Description:

Finds and returns database profile information by its name

Parameters:

profileName – profile name

Return:

DbProfile object or null if not found

Throws:

java.lang.IOException – if the configuration file cannot be accessed.

### **addDbProfile**

```
public void addDbProfile(DbProfile profile)  
    throws java.io.IOException
```

Description:

Creates a new database profile or updates an existing one whose name matches.

Parameters:

profile – database profile object with completed properties. Properties password, connection type and repositoryDatabase are optional.

Return:

None

Throws:

java.lang.IOException – if the configuration file cannot be accessed or written.

Usage:

To add a new profile to the configuration file, first create a new DbProfile object, for example, `DbProfile driver = new DbProfile()` then call setters of the created DbProfile object to populate its properties. After that you can call addDbProfile method to add the prepared profile to the

configuration.

Example:

Create a new instance of DbProfile class, set profile properties and register it with the ProfileManager:

```
DbProfile profile = new DbProfile();
profile.setName("My DB2 Profile");
profile.setDriverName("Db2Driver");
profile.setJdbcUrl("jdbc:db2://MY_BIG_SERVER:50000/DATAWARE");
profile.setMailServer("smtp.my-email-server.com");
profile.setMailSender("db_audit@my-company.com");
```

```
ProfileManager profileManager = ProfileManagerImpl.getInstance();
profileManager.addDbProfile(profile);
```

### **deleteDbProfile**

```
public void deleteDbProfile(java.lang.String profileName)
    throws java.io.IOException
```

Description:

Deletes database profile if it can be found by its name

Parameters:

profileName – profile name

Return:

None

Throws:

java.lang.IOException – if the configuration file cannot be accessed or written.

Example:

Delete profile with the name "Dataware":

```
ProfileManager profileManager = ProfileManagerImpl.getInstance();
profileManager.deleteDbProfile("Dataware");
```

### **createDbConnect**

```
public DbConnect createDbConnect(java.lang.String profileName)
    throws java.io.IOException
```

Description:

Creates new DbConnectImpl object and initializes it with the data from the given profile. The returned object is effectively prepared for establishing a new database connection.

Parameters:

profileName – profile name

Return:

DbConnectImpl object

Throws:

java.lang.IOException – if the configuration file cannot be accessed or written.

## Default Configuration Generator Class

### Overview

The **DefaultConfigGenerator** class can be used to generate default configuration file profiles.xml using preconfigured driver settings for Oracle, DB2, Microsoft SQL Server and Sybase database drivers and also create 1 profile for each named database system. The full class name is **com.softtreetech.dbaudit.DefaultConfigGenerator**.

To create default configuration file simply create an instance of **DefaultConfigGenerator** class, for example:

```
DefaultConfigGenerator dcg = new DefaultConfigGenerator();
```

## Database Connectivity Class

### Overview

Database connectivity class encapsulates all database connectivity functions for all supported database systems and provides single **DbConnect** object for interfacing with the database.

The full class name is **com.softtreetech.com.dbaudit.DbConnect** for the class declaration and **com.softtreetech.com.dbaudit.DbConnectImpl** for the class implementation.

### Connecting/Disconnecting to/from Database

#### ***connect***

```
public void connect( java.lang.String username, java.lang.String password)  
    throws java.lang.ClassNotFoundException, java.sql.SQLException
```

Description:

Establishes new connection, if it does not already exist

Parameters:

username – username to be used. If null, then the saved user name from profile settings is used for the new connection.

password – password to be used. If null, then the saved password from profile settings is used for the new connection.

Return:

None

Throws:

java.lang.ClassNotFoundException – if the JDBC driver could not be loaded

java.sql.SQLException – if a connection error occurs

Example:

```
// connect to database  
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Oracle Profile");  
con.connect("user", "password");  
// .... Do something here ....  
// disconnect when done  
con.disconnect();
```

### ***disconnect***

```
public void disconnect()  
    throws java.sql.SQLException
```

Description:

Disconnects from the database.

Parameters:

None

Return:

None

Throws:

java.sql.SQLException – if an error occurs.

Example:

```
// connect to database  
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Oracle Profile");  
con.connect("user", "password");  
// .... Do something here ....  
// disconnect when done  
con.disconnect();
```

### ***Testing Connection***

Using the console interface, run the following:

```
java -jar dbaudit.jar /D profileName [/U user] [/P password] /T
```

User and password are optional. If not specified on the command line the console will use values saved with the profile.

Using the API functions methods, code the following:

```
try {
    DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("profileName");
    // also can use DbConnect con = new DbConnectImpl("profileName", db_type) method
    con.connect("user", "password");
    con.disconnect();
}
catch (Exception e) {
    System.out.println("Cannot connect. " + e.getMessage());
}
```

User and password values are optional. If null value is specified for the user the DbConnect class will use values saved with the profile.

### Using Connection

1. Create an instance of type DbConnect class:

```
DbConnect connection = new DbConnectImpl("My Profile", "DB2");
```

or using driver type independent method

```
DbConnect connection = ProfileManagerImpl.getInstance().createDbConnect("My Profile");
```

2. Connect to database:

```
connection.connect("username", "password");
```

or for a sysdba connection in Oracle

```
connection.connect("username", "password", "sysdba");
```

3. Run required SQL commands using available JDBC methods, for example, any of the following standard JDBC methods can be used:


ResultSet connection.executeQuery(String sql) – to execute a query that returns a ResultSet. For more information see Java documentation for executeQuery method.

int connection.executeUpdate(String sql) – to execute an update-like query that doesn't return a result set. For more information see Java documentation for executeUpdate method.

int connection.execute(String sql) – to execute an arbitrary query having or not having a ResultSet. For more information see Java documentation for executeUpdate method.

For example,

```
connection.execute("ALTER SESSION SET TIME_ZONE='05:00'");
```

 **Tip:** The DbConnectImpl class exposes many additional "execute" like methods which are extended the 3 base methods listed above and provide convenient way to execute query with parameters, queries of specific SQL types and more. Their names and their parameter names are self-explanatory. These methods are documented in the JavaDocs files provided with the DB Audit API library.

## Getting Connection Information

### ***isConnected***

```
public boolean isConnected()
```

Description:

Reports current connection state.

Parameters:

None

Return:

true if connected and false otherwise

Example:

```
void MyFunction(DbConnect con)
{
    if (con.isConnected())
        // ... execute some commands here ...
    else
        System.out.println("Not connected to the database");
}
```

### ***checkConnected***

```
public void checkConnected()
    throws IllegalStateException
```

Description:

Checks current connection state and throws exception if not connected to the database

Parameters:

None

Return:

None

Throws:

java.lang.IllegalStateException – if not connected to the database

Example:

```
void MyFunction(DbConnect con)
{
    try {
        con.checkConnected();
        // ... execute some commands here ...
    }
    catch (Exception e) {
        System.out.println("Not connected to the database");
    }
}
```

**getDbVersion**

```
public DbVersion getDbVersion()  
    throws java.sql.SQLException
```

## Description:

Reports version of the database the object is connected to.

## Parameters:

None

## Return:

DbVersion object containing database version info.

## Throws:

java.lang.SQLException – if an error occurs.

## Example:

```
// connect to database  
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Oracle Profile");  
con.connect("user", "password");  
// check Oracle version. If version 10g or later run Text of SQL Queries report  
DbVersion ver = con.getDbVersion();  
if (ver.isSameOrLater("10.0"))  
{  
    Reports reports = ReportsFactory.getReportsInstance(con, null, null);  
    reports.doTextOfSQLQueriesReport("SCOTT", null, null, null);  
}  
else  
{  
    System.out.println("SQL Queries report is not supported in this version");  
}  
// disconnect from database  
con.disconnect();
```

**getDbProfile**

```
public DbProfile getDbProfile()
```

## Description:

Returns database profile info associated with the current connection.

## Parameters:

None

## Return:

DbProfile object

## Example:

```
void MyFunction(DbConnect con)  
{  
    // print name of the profile associated with the connection  
    System.out.println("Connected to " + con.getDbProfile().getName());  
}
```



***getDbDriver***

```
public DbDriver getDbDriver()
```

Description:

Returns database driver info associated with the current connection.

Parameters:

None

Return:

DbDriver object

Example:

```
void MyFunction(DbConnect con)  
{  
    // print name of the database driver associated with the connection  
    System.out.println("Using database driver " + con.getDbDriver().getName());  
}
```

***getConnection***

```
public java.sql.Connection getConnection()
```

Description:

Returns JDBC native database connection object associated with the current connection. If the database is connected, the returned object can be used to execute various operations in that database.

Parameters:

None

Return:

java.sql.Connection object

Example:

```
ResultSet MyFunction(DbConnect con)  
{  
    // obtain reference to internal JDBC connection class  
    Connection jdbcCon = con.getConnection();  
    // ... do something with this connection here ...  
    // ... for example, execute a SELECT query and return the created ResultSet object  
    Statement stmt = jdbcCon.createStatement();  
    return stmt.executeQuery("SELECT * FROM v$parameters");  
}
```

***repositoryUsed***

```
public boolean repositoryUsed()
```

Description:

Reports whether the connected database server type requires a repository database to store audit data.

Parameters:

None

Return:

Return true if repository is supported and false otherwise

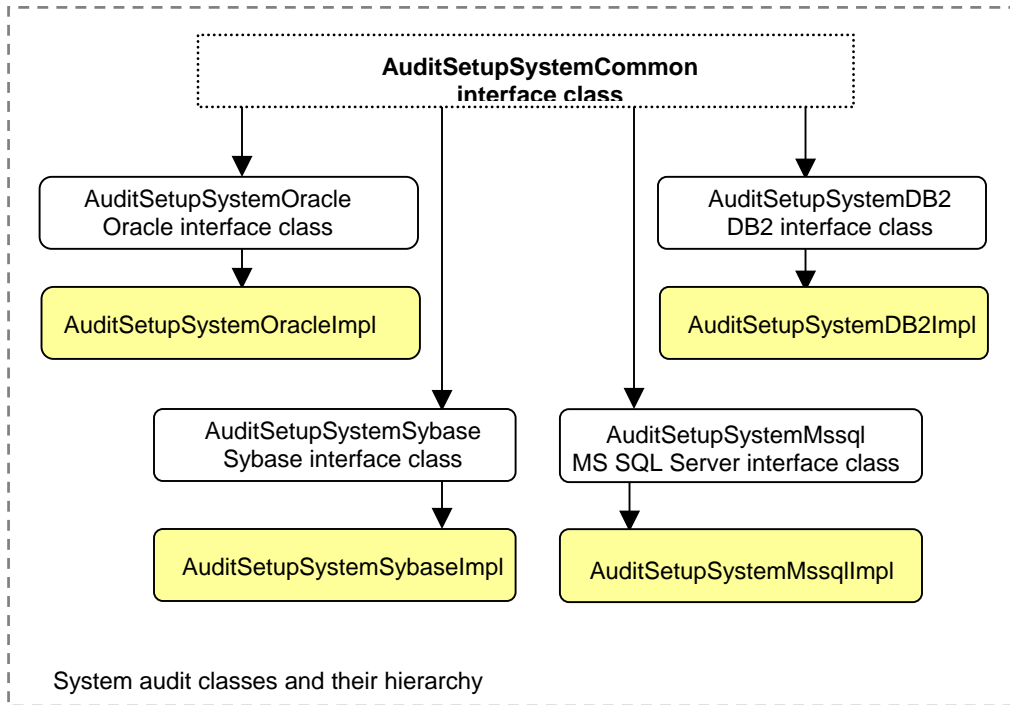
Example:

```
void MyFunction(DbConnect con)
{
    if ( con.repositoryUsed() )
        System.out.println("We need a repository for this database type");
    else
        System.out.println("We don't need a repository for this database type");
}
```

# CHAPTER 4, System Audit Management

## Class Hierarchy

The following diagram demonstrates internal hierarchy of DB Audit classes used for system audit management tasks. All described classes are part of **com.softtreetech.dbaudit.auditsetup.system** package. As you can see on the diagram, the system audit setup is based on single common interface while specific implementations are provided to perform DBMS specific audit setups.



The highlighted classes provide DBMS specific implementations. Use them with direct Java interface or RMI-based Java interface. Note that the DB Audit command line interface hides all internal class complexity and provides flat keyword based interface. In case if you use the direct Java interface or RMI-based Java interface be sure to instantiate the correct classes when attempting to execute database-system specific audit setups or make changes in the existing configuration.

The following topics describe methods for performing DBMS specific system audit management tasks. For reader's convenience, all available methods are organized in task-oriented groups. Each topic describes method's specification, parameters and return values, and also includes code samples demonstrating how to call these methods.

## Parameter Names and Values



### Important notes for using DB Audit command line interface :

- Be sure to enclose any parameter names and values containing spaces in double quotes. If you do not do that, the Operation System will pass each word as a separate parameter or parameter value. This in turn will lead to incorrect results.

For example, if you want to install system auditing in DB2 to audit all DDL schema changes, you need to specify Object Drop and Create as the name of the operation to audit. The simplified syntax for this command is

```
java -jar dbaudit.jar /D db2ProfileName [/U user] [/P password] /A /F
mode:installSysAudit;operations:<OPERATIONS>;optionWhen:<OPTION_WHEN>
```

Note that everything after the /F command line switch (the text in bold) is a single command line parameter which contains certain keywords and their values. Therefore if you enter it without quotes as in the below example

```
java -jar dbaudit.jar /D db2ProfileName /A /F mode:installSysAudit;operations:Object
Drop and Create;optionWhen:ALWAYS
```

you are going to have problems because DB Audit will receive 8 command line parameter values:

```
/D
db2ProfileName
/A
/F
mode:installSysAudit;operations:Object
Drop
and
Create;optionWhen:ALWAYS
```

instead of the required 5 parameter values:

```
/D
db2ProfileName
/A
/F
mode:installSysAudit;operations:Object Drop and Create;optionWhen:ALWAYS
```

The correct command in this case is (the following must be entered on a single line)

```
java -jar dbaudit.jar /D db2ProfileName /A /F "mode:installSysAudit;operations:Object Drop
and Create;optionWhen:ALWAYS"
```

- All commands and parameter names are **cAsE SeNsItIvE**
- Semicolon symbol is used as keyword and parameter separator. Use of semicolon in parameter values is not supported!

## Oracle System Audit Management Tasks

### Set System Audit State

#### *Install System Audit*

Oracle system audit is a native feature of Oracle databases that is installed with the Oracle server. There is no need to install or uninstall any additional programs or procedures and that is why DB Audit

See "System audit trail management" topic in chapter 3 of DB Audit User's Guide for information on how to activate and deactivate Oracle system audit processes.

DB Audit provides several methods that can be used to enhance the default Oracle auditing and ensure the default system auditing is manageable. For more information, see the following topics:

[Configure and Manage Alternative Audit Trail](#)

[Install/Uninstall Oracle SYS Operations Audit](#)

[Move SYS.AUD\\$ Table to Non-SYSTEM tablespace](#)

[Install/Uninstall Server Errors Auditing and Alerting](#)

#### *Uninstall System Audit*

Oracle system audit is a native feature of Oracle databases that is installed with the Oracle server and cannot be uninstalled.

In case if you want to disable the system auditing see "System audit trail management" topic in chapter 3 of DB Audit User's Guide for information on how to activate and deactivate Oracle system audit processes.

Use the methods described in [Uninstall Audit Repository Objects](#) topic to uninstall both the system audit and data-change audit objects and other options and enhancements that you have previously installed using DB Audit API or DB Audit GUI.



**Important Notes:** The "uninstall" method completely removes DB Audit from the database, it removes all previously installed audit settings, stored procedures and tables **including data-change audit trail tables** and triggers and also removes DB\_AUDIT schema and user. The "uninstall" method **does not remove Oracle native system audit trail tables**.

#### Using the console interface

```
java -jar dbaudit.jar /D oracleProfileName [/U user] [/P password] /A /F mode:uninstallAudit
```

Parameters:

oracleProfileName – required. Name of an existing database connection profile configured for an Oracle database.

user – optional. Name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

Example uninstallation (the following must be entered on a single line):

```
java -jar dbaudit.jar /D oracleProfileName /A /F "mode:uninstallAudit "
```

### Using the API functions

```
void auditSetup.uninstallAudit();
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:


None

Example:

1. Connect to the database, for example:  
*DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Oracle Profile");  
con.connect("user", "password");*
2. Create an instance of the AuditSetupSystemOracle class  
*AuditSetupSystemOracle auditSetup =  
(AuditSetupSystemOracle)AuditSetupSystemCommon.getInstance(con);*
3. Uninstall DB Audit tables, procedures, schema and user  
*auditSetup.uninstallAudit();*


## Enable System Audit

To enable DB Audit compatible auditing you will need to set the AUDIT\_TRAIL=DB parameter in the Oracle instance parameters INIT.ORA file and then restart the instance because this parameter is not settable dynamically. On UNIX systems this file is normally located in the 'dbs' subdirectory under the ORACLE\_HOME directory. On Windows systems this file is named INITORCL.ORA and it is normally located in the 'Database' subdirectory under the ORACLE\_HOME directory. In Oracle versions 9i and later the instance parameters file is located in [ORACLE\_HOME]\admin\ORCL\pfile directory.

 **Important Notes:** The Error Audit and SYSDBA/SYSOPER Operations Audit are independent processes that are not affected by this method. See [Install/Uninstall Server Errors Auditing and Alerting](#) and [Install/Uninstall Oracle SYS Operations Audit](#) topics for information on how to manage these system audit methods options.

## Disable System Audit

To disable the system auditing you will need to set the AUDIT\_TRAIL=FALSE parameter in the Oracle instance parameters file and then restart the instance because this parameter is not settable dynamically.

 **Important Notes:** The Error Audit and SYSDBA/SYSOPER Operations Audit are independent processes that are not affected by this method. See [Install/Uninstall Server Errors Auditing and Alerting](#) and [Install/Uninstall Oracle SYS Operations Audit](#) topics for information on how to manage these system audit methods options.

## Set Audit Operations and Filters

### Add SQL Statement Audit

This method turns on auditing of particular SQL operation types such as ALTER TABLE, DELETE, SELECT and so on. For easy of use Oracle groups different types of SQL operations. For example, to audit security related operations such as GRANT and REVOKE for a table, view or snapshot, use GRANT TABLE group. The following table can be used as reference for available SQL Statement Type names. See "Tables of Auditing Options" topic in the Oracle SQL Reference for a complete list of SQL Statement Type names supported by your Oracle server version.

You can call this method multiple types adding different operation types with different options and filters as required.

SQL Statement Type	Comments
ALL	Audit all statements. This option is exclusive. If you specify ALL, do not specify any other group.
ALTER SEQUENCE	ALTER SEQUENCE
ALTER TABLE	ALTER TABLE
CLUSTER	CREATE CLUSTER AUDIT CLUSTER DROP CLUSTER TRUNCATE CLUSTER
COMMENT TABLE	COMMENT ON TABLE table, view, snapshot, procedure, function, package COMMENT ON COLUMN table.column, view.column, snapshot.column
DATABASE LINK	CREATE DATABASE LINK DROP DATABASE LINK
DELETE TABLE	DELETE FROM table, view
DIRECTORY	CREATE DIRECTORY DROP DIRECTORY
EXECUTE PROCEDURE	Execution of any procedure or function or access to any variable, library, or cursor inside a package.
GRANT DIRECTORY	GRANT privilege ON directory REVOKE privilege ON directory
GRANT PROCEDURE	GRANT privilege ON procedure, function, package REVOKE privilege ON procedure, function, package
GRANT SEQUENCE	GRANT privilege ON sequence REVOKE privilege ON sequence
GRANT TABLE	GRANT privilege ON table, view, snapshot. REVOKE privilege ON table, view, snapshot
GRANT TYPE	GRANT privilege ON TYPE REVOKE privilege ON TYPE
INDEX	CREATE INDEX ALTER INDEX

	DROP INDEX
INSERT TABLE	INSERT INTO table, view
LOCK TABLE	LOCK TABLE table, view
NOT EXISTS	All SQL statements that fail because a specified object does not exist.
PROCEDURE	CREATE FUNCTION CREATE LIBRARY CREATE PACKAGE CREATE PACKAGE BODY CREATE PROCEDURE DROP FUNCTION DROP LIBRARY DROP PACKAGE DROP PROCEDURE
PROFILE	CREATE PROFILE ALTER PROFILE DROP PROFILE
PUBLIC DATABASE LINK	CREATE PUBLIC DATABASE LINK DROP PUBLIC DATABASE LINK
PUBLIC SYNONYM	CREATE PUBLIC SYNONYM DROP PUBLIC SYNONYM
ROLE	CREATE ROLE ALTER ROLE DROP ROLE SET ROLE
ROLLBACK SEGMENT	CREATE ROLLBACK SEGMENT ALTER ROLLBACK SEGMENT DROP ROLLBACK SEGMENT
SELECT SEQUENCE	Any statement containing sequence.CURRVAL or sequence.NEXTVAL
SELECT TABLE	SELECT FROM table, view, snapshot
SEQUENCE	CREATE SEQUENCE DROP SEQUENCE
SESSION	Logons and logouts
SYNONYM	CREATE SYNONYM DROP SYNONYM
SYSTEM AUDIT	AUDIT (SQL Statements) NOAUDIT (SQL Statements)
SYSTEM GRANT	GRANT (System Privileges and Roles) REVOKE (System Privileges and Roles)
TABLE	CREATE TABLE DROP TABLE TRUNCATE TABLE
TABLESPACE	CREATE TABLESPACE ALTER TABLESPACE DROP TABLESPACE



TRIGGER	CREATE TRIGGER ALTER TRIGGER with ENABLE and DISABLE options DROP TRIGGER ALTER TABLE with ENABLE ALL TRIGGERS or DISABLE ALL TRIGGERS clauses
TYPE	CREATE TYPE CREATE TYPE BODY ALTER TYPE DROP TYPE DROP TYPE BODY
UPDATE TABLE	UPDATE table, view
USER	CREATE USER ALTER USER DROP USER
VIEW	CREATE VIEW DROP VIEW

### Using the console interface

```
java -jar dbaudit.jar /D oracleProfileName [/U user] [/P password] /A /F
"mode:addOracleStatements;statements:<STATEMENTS>;users:<USERS>;optionBy:<BY
OPTION>;optionWhen:<WHEN OPTION>"
```

#### Parameters:

oracleProfileName – required. Name of an existing database connection profile configured for an Oracle database.

user – optional. Name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<STATEMENTS> – required. Comma-separated list of SQL Statement Types supported by your version of the Oracle database. For the reference .

<BY OPTION> – One of the following values: BY ACCESS, BY SESSION

BY ACCESS causes the audit system to insert one record into the audit trail for each execution of an auditable operation – this is the default and most commonly used option. BY SESSION causes the audit system to insert only one record in the audit trail, for each user and schema object, during the session that includes an audited action, even though the user executes the same operation for the same object multiple times.

<WHEN OPTION> – optional. One of the following values: WHENEVER SUCCESSFUL, WHENEVER NOT SUCCESSFUL or ALWAYS

WHENEVER SUCCESSFUL can be used to audit only successful executions of the audited statement.

WHENEVER NOT SUCCESSFUL can be used to audit only unsuccessful executions of the audited statement.

ALWAYS can be used to audit both successful and unsuccessful executions of the audited statement. This is the default option.

<USERS> – optional. Comma-separated list of database user names. If no specified then all users are audited.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D oracleProfileName /A /F "mode:addOracleStatements;statements:SYSTEM
AUDIT,SYSTEM GRANT,TABLE;users:TESTUSER,TESTUSER1;
optionBy:BY ACCESS;optionWhen:ALWAYS"
```

### Using the API functions

```
void auditSetup.addAuditStatements(String[] sqlStatements, String[] users, String optionBy, String
optionWhen);
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

sqlStatements – String array of SQL Statement Types supported by your version of the Oracle database. Consult your Oracle documentation for details. See "Tables of Auditing Options" topic in the Oracle SQL Reference.

optionBy – One of the following values: BY ACCESS, BY SESSION

BY ACCESS causes the audit system to insert one record into the audit trail for each execution of an auditable operation – this is the most commonly used option.

BY SESSION causes the audit system to insert only one record in the audit trail, for each user and schema object, during the session that includes an audited action, even though the user executes the same operation for the same object multiple times.

optionWhen – One of the following values: WHENEVER SUCCESSFUL, WHENEVER NOT SUCCESSFUL or ALWAYS

WHENEVER SUCCESSFUL can be used to audit only successful executions of the audited statement.

WHENEVER NOT SUCCESSFUL can be used to audit only unsuccessful executions of the audited statement.

ALWAYS can be used to audit both successful and unsuccessful executions of the audited statement.

users – String array of database user names.

Example:

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Oracle Profile");
con.connect("user", "password");
```
2. Create an instance of the AuditSetupSystemOracle class  

```
AuditSetupSystemOracle auditSetup =
(AuditSetupSystemOracle)AuditSetupSystemCommon.getInstance(con);
```
3. Add Oracle statement audits for 3 types of statements and 2 users  

```
auditSetup.addAuditStatements({"SYSTEM AUDIT", "SYSTEM GRANT", "TABLE"},
{"TESTUSER", "TESTUSER1"},
"BY ACCESS", "ALWAYS");
```

## Remove SQL Statement Audit

This method turns off auditing of particular SQL operation types. See [Add SQL Statement Audit](#) topic for more information.

You can call this method multiple times removing auditing of different operation types with different options and filters as required.

### Using the console interface

```
java -jar dbaudit.jar /D oracleProfileName [/U user] [/P password] /A /F
"mode:removeOracleStatements;statements:<STATEMENTS>;users:<USERS>;optionBy:<BY
OPTION>;optionWhen:<WHEN OPTION>"
```

#### Parameters:

oracleProfileName – required. Name of an existing database connection profile configured for an Oracle database.

user – optional. Name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<STATEMENTS> – required. Comma-separated list of SQL Statement Types supported by your version of the Oracle database. Consult your Oracle documentation for details. See "Tables of Auditing Options" topic in the Oracle SQL Reference.

<BY OPTION> – required. One of the following values: BY ACCESS, BY SESSION

BY ACCESS causes the audit system to insert one record into the audit trail for each execution of an auditable operation – this is the default and most commonly used option.

BY SESSION causes the audit system to insert only one record in the audit trail, for each user and schema object, during the session that includes an audited action, even though the user executes the same operation for the same object multiple times.

<WHEN OPTION> – optional. One of the following values: WHENEVER SUCCESSFUL, WHENEVER NOT SUCCESSFUL or ALWAYS

WHENEVER SUCCESSFUL can be used to audit only successful executions of the audited statement.

WHENEVER NOT SUCCESSFUL can be used to audit only unsuccessful executions of the audited statement.

ALWAYS can be used to audit both successful and unsuccessful executions of the audited statement. This is the default option.

<USERS> – optional. Comma-separated list of database user names. If not specified then ALL\_USERS option is assumed.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D oracleProfileName /A /F
"mode:removeOracleStatements;statements:SYSTEM AUDIT, SYSTEM
GRANT, TABLE;users:TESTUSER,TESTUSER1;optionBy:
BY ACCESS;optionWhen:ALWAYS"
```

### Using the API functions

```
void auditSetup.removeAuditStatements(String[] sqlStatements, String[] users, String optionBy, String
optionWhen);
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

sqlStatements – String array of SQL Statement Types supported by your version of the Oracle database. Consult your Oracle documentation for details. See "Tables of Auditing Options" topic in the Oracle SQL Reference.

optionBy – One of the following values: BY ACCESS, BY SESSION

BY ACCESS causes the audit system to insert one record into the audit trail for each execution of an auditable operation – this is the most commonly used option.

BY SESSION causes the audit system to insert only one record in the audit trail, for each user and schema object, during the session that includes an audited action, even though the user executes the same operation for the same object multiple times.

optionWhen – One of the following values: WHENEVER SUCCESSFUL, WHENEVER NOT SUCCESSFUL or ALWAYS

WHENEVER SUCCESSFUL can be used to audit only successful executions of the audited statement.

WHENEVER NOT SUCCESSFUL can be used to audit only unsuccessful executions of the audited statement.

ALWAYS can be used to audit both successful and unsuccessful executions of the audited statement.

users – String array of database user names.

Example:

1. Connect to the database, for example:  
*DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Oracle Profile");*  
*con.connect("user", "password");*
2. Create an instance of the AuditSetupSystemOracle class  
*AuditSetupSystemOracle auditSetup =*  
*(AuditSetupSystemOracle)AuditSetupSystemCommon.getInstance(con);*
3. Remove Oracle statement audits for 3 types of statements and 2 users  
*auditSetup.addAuditStatements({"SYSTEM AUDIT", " SYSTEM GRANT", "TABLE"},*  
 *{" TESTUSER", "TESTUSER1"},*  
 *"BY ACCESS", "ALWAYS");*

### Add Object Access Audit

This method turns on access auditing for a specific database object or group of objects. This is similar to using SQL Statement Audit with an audit scope limited to specific objects. See [Add SQL Statement Audit](#) topic for more information on supported operation types.

You can call this method multiple times adding different operation types with different options and filters as required.

### Using the console interface

```
java -jar dbaudit.jar /D oracleProfileName [/U user] [/P password] /A /F
"mode:addOracleObjects;objects:<OBJECTS>;operations:<OPERATIONS>;optionBy:<BY
OPTION>;optionWhen:<WHEN OPTION>"
```

#### Parameters:

`oracleProfileName` – required. Name of an existing database connection profile configured for an Oracle database.

`user` – optional. Name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used.

`password` – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

`< OBJECTS >` – required. Comma-separated list of fully qualified object names such as table names, view, etc., including schema names.

`< OPERATIONS >` – required. Comma-separated list of SQL Statement Types supported by your version of the Oracle database. Consult your Oracle documentation for details.

`<BY OPTION>` – optional. One of the following values: BY ACCESS, BY SESSION

BY ACCESS causes the audit system to insert one record into the audit trail for each execution of an auditable operation – this is the default and most commonly used option.

BY SESSION causes the audit system to insert only one record in the audit trail, for each user and schema object, during the session that includes an audited action, even though the user executes the same operation for the same object multiple times.

`<WHEN OPTION>` – optional. One of the following values: WHENEVER SUCCESSFUL, WHENEVER NOT SUCCESSFUL or ALWAYS

WHENEVER SUCCESSFUL can be used to audit only successful executions of the audited statement.

WHENEVER NOT SUCCESSFUL can be used to audit only unsuccessful executions of the audited statement.

ALWAYS can be used to audit both successful and unsuccessful executions of the audited statement. This is the default option.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D oracleProfileName /A /F
"mode:addOracleObjects;objects:DEMO.EMPLOYEE,DEMO.DEPT;operations:DELETE,
UPDATE; optionBy:BY ACCESS;optionWhen:ALWAYS"
```

### Using the API functions

```
void auditSetup.addAuditObjects(String[] objects, String[] operations, String optionBy, String
```

optionWhen);

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

objects – String array of fully qualified object names such as table names, view names, etc. Specified names should be fully qualified and include schema name.

operations – String array of SQL Statement Types supported by your version of the Oracle database. Consult your Oracle documentation for details.

optionBy – One of the following values: BY ACCESS, BY SESSION

BY ACCESS causes the audit system to insert one record into the audit trail for each execution of an auditable operation – this is the most commonly used option.

BY SESSION causes the audit system to insert only one record in the audit trail, for each user and schema object, during the session that includes an audited action, even though the user executes the same operation for the same object multiple times.

optionWhen – One of the following values: WHENEVER SUCCESSFUL, WHENEVER NOT SUCCESSFUL or ALWAYS

WHENEVER SUCCESSFUL can be used to audit only successful executions of the audited statement.

WHENEVER NOT SUCCESSFUL can be used to audit only unsuccessful executions of the audited statement.

ALWAYS can be used to audit both successful and unsuccessful executions of the audited statement.

Example:

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Oracle Profile");
con.connect("user", "password");
```
2. Create an instance of the AuditSetupSystemOracle class  

```
AuditSetupSystemOracle auditSetup =
(AuditSetupSystemOracle)AuditSetupSystemCommon.getInstance(con);
```
3. Add Object Access Audits for 2 objects and 2 operation types  

```
auditSetup.addAuditObjects({"DEMO.EMPLOYEE", "DEMO.DEPT"},
{"DELETE", "UPDATE"},
"BY ACCESS", "ALWAYS");
```

### **Remove Object Access Audit**

This method turns off access auditing for specific database objects. See [Add Object Access Audit](#) topic for more information.

You can call this method multiple times removing auditing for different objects with different options and filters as required.

#### **Using the console interface**

```
java -jar dbaudit.jar /D oracleProfileName [/U user] [/P password] /A /F "mode:
removeOracleObjects;objects:<OBJECTS>;operations:<OPERATIONS>;optionWhen:<WHEN
OPTION>"
```

**Parameters:**

oracleProfileName – required. Name of an existing database connection profile configured for an Oracle database.

user – optional. Name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

< OBJECTS > – required. Comma-separated list of fully qualified object names such as table names, view names, etc. Specified names should be fully qualified and include schema name.

< OPERATIONS > – required. Comma-separated list of SQL Statement Types supported by your version of the Oracle database. Consult your Oracle documentation for details.

<BY OPTION> – optional. One of the following values: BY ACCESS, BY SESSION

BY ACCESS causes the audit system to insert one record into the audit trail for each execution of an auditable operation – this is the default and most commonly used option.

BY SESSION causes the audit system to insert only one record in the audit trail, for each user and schema object, during the session that includes an audited action, even though the user executes the same operation for the same object multiple times.

<WHEN OPTION> – optional. One of the following values: WHENEVER SUCCESSFUL, WHENEVER NOT SUCCESSFUL or ALWAYS

WHENEVER SUCCESSFUL can be used to audit only successful executions of the audited statement.

WHENEVER NOT SUCCESSFUL can be used to audit only unsuccessful executions of the audited statement.

ALWAYS can be used to audit both successful and unsuccessful executions of the audited statement. This is the default option.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D oracleProfileName /A /F  
"mode:removeOracleObjects;objects:DEMO.EMPLOYEE,DEMO.DEPT;operations:DELETE,  
UPDATE; optionWhen:ALWAYS"
```

**Using the API functions**

```
void auditSetup.removeAuditObjects(String[] objects, String[] operations, String optionWhen);
```

**Throws:**

java.sql.SQLException – if an error occurs.

**Parameters:**

objects – String array of fully qualified object names such as table names, view, etc., including schema names.

operations – String array of SQL Statement Types supported by your version of the Oracle

database. Consult your Oracle documentation for details.

optionBy – One of the following values: BY ACCESS, BY SESSION

BY ACCESS causes the audit system to insert one record into the audit trail for each execution of an auditable operation – this is the most commonly used option.

BY SESSION causes the audit system to insert only one record in the audit trail, for each user and schema object, during the session that includes an audited action, even though the user executes the same operation for the same object multiple times.

optionWhen – One of the following values: WHENEVER SUCCESSFUL, WHENEVER NOT SUCCESSFUL or ALWAYS

WHENEVER SUCCESSFUL can be used to audit only successful executions of the audited statement.

WHENEVER NOT SUCCESSFUL can be used to audit only unsuccessful executions of the audited statement.

ALWAYS can be used to audit both successful and unsuccessful executions of the audited statement.

Example:

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Oracle Profile");
con.connect("user", "password");
```
2. Create an instance of the AuditSetupSystemOracle class  

```
AuditSetupSystemOracle auditSetup =
(AuditSetupSystemOracle)AuditSetupSystemCommon.getInstance(con);
```
3. Remove Object Access Audits for 2 objects and 2 operation types  

```
auditSetup.removeAuditStatements({"DEMO.EMPLOYEE", "DEMO.DEPT"},
{"DELETE", "UPDATE"}, ALWAYS);
```

## Add Session Audit

This method turns on auditing of database logons and logoffs. The result is identical to result of adding SQL Statement Audit using SESSION as the operation type for all database users.

### Using the console interface

```
java -jar dbaudit.jar /D oracleProfileName [/U user] [/P password] /A /F
"mode:addOracleSession;optionWhen:<WHEN OPTION>"
```

Parameters:

oracleProfileName – required. Name of an existing database connection profile configured for an Oracle database.

user – optional. Name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<WHEN OPTION> – optional. One of the following values: WHENEVER SUCCESSFUL,



**WHENEVER NOT SUCCESSFUL or ALWAYS**

WHENEVER SUCCESSFUL can be used to audit only successful executions of the audited statement.

WHENEVER NOT SUCCESSFUL can be used to audit only unsuccessful executions of the audited statement.

ALWAYS can be used to audit both successful and unsuccessful executions of the audited statement. This is the default option.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D oracleProfileName /A /F
"mode:addOracleSession;optionWhen:WHENEVER NOT SUCCESSFUL"
```

**Using the API functions**

```
void auditSetup.addAuditSession(String optionWhen);
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

optionWhen – One of the following values: WHENEVER SUCCESSFUL, WHENEVER NOT SUCCESSFUL or ALWAYS

WHENEVER SUCCESSFUL can be used to audit only successful executions of the audited statement.

WHENEVER NOT SUCCESSFUL can be used to audit only unsuccessful executions of the audited statement.

ALWAYS can be used to audit both successful and unsuccessful executions of the audited statement.

Example:

1. Connect to the database, for example:
 

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Oracle Profile");
con.connect("user", "password");
```
2. Create an instance of the AuditSetupSystemOracle class
 

```
AuditSetupSystemOracle auditSetup =
(AuditSetupSystemOracle)AuditSetupSystemCommon.getInstance(con);
```
3. Add Oracle sessions audit for failed connection attempts
 

```
auditSetup.addAuditSession ("WHENEVER NOT SUCCESSFUL");
```

**Remove Session Audit**

This method turns off auditing of database logons and logoffs. See [Add Session Audit](#) topic for more information.

**Using the console interface**

```
java -jar dbaudit.jar /D oracleProfileName [/U user] [/P password] /A /F
"mode:removeOracleSession;optionWhen:<WHEN OPTION>"
```

**Parameters:**

oracleProfileName – required. Name of an existing database connection profile configured for an Oracle database.

user – optional. Name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<WHEN OPTION> – optional. One of the following values: WHENEVER SUCCESSFUL, WHENEVER NOT SUCCESSFUL or ALWAYS

WHENEVER SUCCESSFUL can be used to audit only successful executions of the audited statement.

WHENEVER NOT SUCCESSFUL can be used to audit only unsuccessful executions of the audited statement.

ALWAYS can be used to audit both successful and unsuccessful executions of the audited statement. This is the default option.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D oracleProfileName /A /F
"mode:removeOracleSession;optionWhen:WHENEVER NOT SUCCESSFUL"
```

**Using the API functions**

```
void auditSetup.removeAuditSession(String optionWhen);
```

**Throws:**

java.sql.SQLException – if an error occurs.

**Parameters:**

optionWhen – One of the following values: WHENEVER SUCCESSFUL, WHENEVER NOT SUCCESSFUL or ALWAYS

WHENEVER SUCCESSFUL can be used to audit only successful executions of the audited statement.

WHENEVER NOT SUCCESSFUL can be used to audit only unsuccessful executions of the audited statement.

ALWAYS can be used to audit both successful and unsuccessful executions of the audited statement.

**Example:**

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Oracle Profile");
con.connect("user", "password");
```
2. Create an instance of the AuditSetupSystemOracle class  

```
AuditSetupSystemOracle auditSetup =
(AuditSetupSystemOracle)AuditSetupSystemCommon.getInstance(con);
```
3. Remove Oracle sessions audit for failed connection attempts  

```
auditSetup.removeAuditSession ("WHENEVER NOT SUCCESSFUL");
```

## Configure Alternative Audit Trail

For information about available alternative audit trail tables and when and how to use them, read CHAPTER 3 in DB Audit User's Guide. The process description and manual installation steps are described in "Configuring Advanced Options for Oracle" topic, in section "Audit Method and Storage."

### Tips:

- Installation of the alternative audit trail tables does not automatically create audit trail data transfer job. You should schedule this job after successfully installing audit trail tables. See next topic for more information on how to schedule such job.
- Uninstallation of the alternative audit trail tables does not automatically remove the scheduled audit trail data transfer job. You should remove that job before uninstalling the alternative audit tables. See next topic for more information on how to remove that job.

## ***Install/Uninstall Oracle Alternative Audit Trail***

### **Using the console interface**

To install the alternative audit trail:

```
java -jar dbaudit.jar /D oracleProfileName [/U user] [/P password] /A /F  
mode:installAlternativeAudit;tablespaces:<TABLE TABLESPACE>,<INDEX TABLESPACE>
```

To uninstall the alternative audit trail and revert back to system default tables:

```
java -jar dbaudit.jar /D oracleProfileName [/U user] [/P password] /A /F mode:uninstallAlternativeAudit
```

Parameters:

oracleProfileName – required. Name of an existing database connection profile configured for an Oracle database.

user – optional. Name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<TABLE TABLESPACE> – required. Name of the tablespace to use for audit trail tables.

<INDEX TABLESPACE> – required. Name of the tablespace to use for audit trail indexes.

Example installation:

1. Install alternative audit trail tables, views and procedures (the following must be entered on a single line):  

```
java -jar dbaudit.jar /D oracleProfileName /A /F
mode:installAlternativeAudit;tablespaces:AUDIT_TABLES,AUDIT_INDEXES
```
2. Now schedule the audit trail data transfer job (the following must be entered on a single line):  

```
java -jar dbaudit.jar /D oracleProfileName /A /F
mode:scheduleAlternativeDataTransferJob;interval:5;timeUnit:minutes
```

Example uninstallation:

1. Remove the data transfer job (the following must be entered on a single line):  

```
java -jar dbaudit.jar /D oracleProfileName /A /F
mode:removeScheduledAlternativeDataTransferJob
```
2. Drop alternative audit trail tables, views and procedures (the following must be entered on a single line):  

```
java -jar dbaudit.jar /D oracleProfileName /A /F mode:uninstallAlternativeAudit
```

### Using the API functions

To install the alternative audit trail use:

```
void auditSetup.installAlternativeAudit(String tableTableSpace, String indexTableSpace);
```

To uninstall the alternative audit trail and revert back to system default tables use:

```
void auditSetup.uninstallAlternativeAudit();
```

Throws:

`java.sql.SQLException` – if an error occurs.

Parameters:

`tableTableSpace` – name of the tablespace to use for audit trail tables.

`indexTableSpace` – name of the tablespace to use for audit trail indexes.

Example installation:

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Oracle Profile");
con.connect("user", "password");
```
2. Create an instance of the `AuditSetupSystemOracle` class  

```
AuditSetupSystemOracle auditSetup =
(AuditSetupSystemOracle)AuditSetupSystemCommon.getInstance(con);
```
3. Install alternative audit trail tables, views and procedures  

```
auditSetup.installAlternativeAudit("AUDIT_TABLES", "AUDIT_INDEXES");
```
4. Now schedule the audit trail data transfer job  

```
auditSetup.scheduleAlternativeDataTransferJob(new Schedule(5, Schedule.MINUTE));
```

Example uninstallation:

1. Connect to the database, for example:  
*DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Oracle Profile");  
con.connect("user", "password");*
2. Create an instance of the AuditSetupSystemOracle class  
*AuditSetupSystemOracle auditSetup =  
(AuditSetupSystemOracle)AuditSetupSystemCommon.getInstance(con);*
3. Remove the data transfer job  
*auditSetup.removeAlternativeDataTransferJob();*
4. Drop alternative audit trail tables, views and procedures  
*auditSetup.uninstallAlternativeAudit();*

## **Schedule/Remove Oracle Alternative Data Transfer Job**

### **Using the console interface**

To schedule new the alternative audit trail data transfer job or update frequency of an already scheduled job:

```
java -jar dbaudit.jar /D oracleProfileName [/U user] [/P password] /A /F
mode:scheduleAlternativeDataTransferJob;interval:<SCHEDULE
INTERVAL>;timeUnit:<SCHEDULE TIME UNIT>
```

Parameters:

oracleProfileName – required. Name of an existing database connection profile configured for an Oracle database.

user – optional. Name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<SCHEDULE INTERVAL> – required. Schedule interval (job frequency expressed as an integer number)

<SCHEDULE TIME UNIT> – required. Schedule frequency unit of measure. Must one of the following values: minutes, hours, days.

To remove the alternative audit trail data transfer job:

```
java -jar dbaudit.jar /D oracleProfileName [/U user] [/P password] /A /F
mode:removeScheduledAlternativeDataTransferJob
```

Parameters:

oracleProfileName – required. Name of an existing database connection profile configured for an Oracle database.

user – optional. Name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

Example job scheduling (the following must be entered on a single line):

```
java -jar dbaudit.jar /D oracleProfileName /A /F  
mode:scheduleAlternativeDataTransferJob;interval:5;timeUnit:minutes
```

Example job removal (the following must be entered on a single line):

```
java -jar dbaudit.jar /D oracleProfileName /A /F  
mode:removeScheduledAlternativeDataTransferJob
```

### Using the API functions

To schedule new the alternative audit trail data transfer job or update frequency of an already scheduled job:

```
Schedule schedule = new Schedule(long interval, int timeUnit);  
void auditSetup.scheduleAlternativeDataTransferJob(Schedule schedule);
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

interval – schedule job frequency expressed either in hours, minutes or in seconds.

timeUnit – schedule frequency unit of measure, use constants available in Schedule class: Schedule.MINUTE, Schedule.HOUR, Schedule.DAY.

schedule – an instance of Schedule class.

To remove the alternative audit trail data transfer job:

```
void auditSetup.removeAlternativeDataTransferJob();
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

None

Example installation:

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Oracle Profile");  
con.connect("user", "password");
```
2. Create an instance of the AuditSetupSystemOracle class  

```
AuditSetupSystemOracle auditSetup =  
(AuditSetupSystemOracle)AuditSetupSystemCommon.getInstance(con);
```

3. Install alternative audit trail tables, views and procedures  
*auditSetup.installAlternativeAudit("AUDIT\_TABLES", "AUDIT\_INDEXES");*
4. Now schedule the audit trail data transfer job  
*auditSetup.scheduleAlternativeDataTransferJob(new Schedule(5, Schedule.MINUTE));*

Example uninstallation:

1. Connect to the database, for example:  
*DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Oracle Profile");*  
*con.connect("user", "password");*
2. Create an instance of the AuditSetupSystemOracle class  
*AuditSetupSystemOracle auditSetup =*  
*(AuditSetupSystemOracle)AuditSetupSystemCommon.getInstance(con);*
3. Remove the data transfer job  
*auditSetup.removeAlternativeDataTransferJob();*
4. Drop alternative audit trail tables, views and procedures  
*auditSetup.uninstallAlternativeAudit();*

## Set Advanced Audit Options

### *Install/Uninstall Oracle SYS Operations Audit*

Methods described in this topic can be used to install tables and procedures for auditing privileged users connected as SYSDBA or SYSOPER. For more information about auditing of privileged users in Oracle read topic "Auditing Privileged Users connected as SYSDBA or SYSOPER" in CHAPTER 3 of DB Audit User's Guide.



**Important Note:** The described below methods only install/uninstall DB Audit's infrastructure for auditing privileged users, they do not set or unset Oracle's parameter **AUDIT\_SYS\_OPERATIONS = TRUE** in the initialization parameters file [INIT.ORA]. This is a static parameter. It cannot be set dynamically using the ALTER SYSTEM command. The database must be bounced for the parameter change to take effect.

#### Using the console interface

To install the SYS operations audit trail tables and procedures:

```
java -jar dbaudit.jar /D oracleProfileName [/U user] [/P password] /A /F
mode:installSYSOperationsAudit;dirName:<DIRECTORY NAME>
```

Parameters:

oracleProfileName – required. Name of an existing database connection profile configured for an Oracle database.

user – optional. Name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<DIRECTORY NAME> – required. Full name of the server directory into which the audit trail files are written

To uninstall the SYS operations audit trail tables and procedures:

```
java -jar dbaudit.jar /D oracleProfileName [/U user] [/P password] /A /F  
mode:uninstallSYSOperationsAudit
```

Parameters:

oracleProfileName – required. Name of an existing database connection profile configured for an Oracle database.

user – optional. Name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

Example installation (the following must be entered on a single line):

```
java -jar dbaudit.jar /D oracleProfileName /A /F  
"mode:installSYSOperationsAudit;dirName:/oracle/rdbms/audit"
```

Example uninstallation:

```
java -jar dbaudit.jar /D oracleProfileName /A /F "mode:uninstallSYSOperationsAudit"
```

### Using the API functions

To install the SYS operations audit trail tables and procedures:

```
void auditSetup.installSYSOperationsAudit(String dirName);
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

dirName – full name of the server directory into which the audit trail files are written

To uninstall the SYS operations audit trail tables and procedures:

```
void auditSetup.uninstallSYSOperationsAudit();
```

Throws:

java.sql.SQLException – if an error occurs.



Parameters:

None

Example installation:

1. Connect to the database, for example:  
*DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Oracle Profile");  
con.connect("user", "password");*
2. Create an instance of the AuditSetupSystemOracle class  
*AuditSetupSystemOracle auditSetup =  
(AuditSetupSystemOracle)AuditSetupSystemCommon.getInstance(con);*
3. Install SYS audit trail tables, views and procedures  
*auditSetup.installSYSOperationsAudit("/oracle/rdbms/audit");*

Example uninstallation:

1. Connect to the database, for example:  
*DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Oracle Profile");  
con.connect("user", "password");*
2. Create an instance of the AuditSetupSystemOracle class  
*AuditSetupSystemOracle auditSetup =  
(AuditSetupSystemOracle)AuditSetupSystemCommon.getInstance(con);*
3. Drop SYS audit trail tables, views and procedures  
*auditSetup.unInstallSYSOperationsAudit();*

## ***Install/Uninstall Server Errors Auditing and Alerting***

Methods described in this topic can be used to install and uninstall auditing of database errors with optional real-time email alerts.



**Important Note:** In order to use the real-time email alerting option with the Oracle server errors audit you must have DB Audit mail sending SQL procedure installed as described in the previous topic.

### **Using the console interface**

To install the server errors audit:

```
java -jar dbaudit.jar /D oracleProfileName [/U user] [/P password] /A /F
mode:installServerErrorsAudit;recipient:<RECIPIENT_EMAIL>;cc:<CC>;errors:<ERRORS>;
```

Parameters:

*oracleProfileName* – required. Name of an existing database connection profile configured for an Oracle database.

*user* – optional. Name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used.

*password* – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<RECIPIENT\_EMAIL> – optional. Email address of the recipient or email group where to email real-time error alerts. If not specified, errors are only logged to the audit trail table, but the email alert is not sent.

<CC> – optional. Email CC address of the recipient or email group where to email real-time error alerts. Do not specify this parameter if CC is not required.

<ERRORS> – optional. Comma-separated list of error numbers used as a filter for the error auditing. Error numbers here are the same that Oracle uses for system errors recorded in the alert log and other places, typically in ORA-nnnnn format. If specified, only numeric values must be used. If not specified, then all errors are captured.

To uninstall the server errors audit:

```
java -jar dbaudit.jar /D oracleProfileName [/U user] [/P password] /A /F  
mode:uninstallServerErrorsAudit
```

Parameters:

oracleProfileName – required. Name of an existing database connection profile configured for an Oracle database.

user – optional. Name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

Example installation (the following must be entered on a single line):

```
java -jar dbaudit.jar /D oracleProfileName /A /F  
"mode:installServerErrorsAudit;recipient:dba_group@mycompany.org;  
errors:1303,54,60,1000,1113"
```

Example uninstallation:

```
java -jar dbaudit.jar /D oracleProfileName /A /F "mode:uninstallServerErrorsAudit"
```

### Using the API functions

To install the server errors audit objects:

```
void auditSetup.installTriggerForMonitoringErrors(String recipient, String cc, String errors);
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

recipient – Email address of the recipient or email group where to email real-time error alerts.

If null is specified, errors are only logged to the audit trail table, but the email alert is not sent.

cc – Email CC address of the recipient or email group where to email real-time error alerts. Use null value if CC is not required.

errors. Comma-separated list of error numbers used as a filter for the error auditing. Error numbers here are the same that Oracle uses for system errors recorded in the alert log and other places, typically in ORA-nnnnn format. To capture all errors specify null value for this parameter.

To uninstall the server errors audit objects:

```
void auditSetup.uninstallTriggerForMonitoringErrors();
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

None

Example installation:

1. Connect to the database, for example:  
*DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Oracle Profile");  
con.connect("user", "password");*
2. Create an instance of the AuditSetupSystemOracle class  
*AuditSetupSystemOracle auditSetup =  
(AuditSetupSystemOracle)AuditSetupSystemCommon.getInstance(con);*
3. Install mail sending procedure  
*auditSetup.installDBAuditMailSendingSQLProcedure(null);*
4. Install errors audit log table and trigger  
*auditSetup.installTriggerForMonitoringErrors("dba\_group@mycompany.org", null,  
"1303,54,60,1000,1113");*

Example uninstallation:

1. Connect to the database, for example:  
*DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Oracle Profile");  
con.connect("user", "password");*
2. Create an instance of the AuditSetupSystemOracle class  
*AuditSetupSystemOracle auditSetup =  
(AuditSetupSystemOracle)AuditSetupSystemCommon.getInstance(con);*
3. Uninstall errors audit log table and trigger  
*auditSetup.uninstallTriggerForMonitoringErrors();*

### **Move SYS.AUD\$ Table to Non-SYSTEM tablespace**

This method can be used to move system audit trail table SYS.AUD\$ from the SYSTEM tablespace to a user-defined tablespace. This is needed to prevent situation with the SYSTEM tablespace being filled up with potentially devastating effects on the database access and availability. For more details read "Moving Oracle native system audit trail table out of the SYSTEM tablespace" topic in CHAPTER

## 3, System Auditing in the DB Audit User's Guide.

**Using the console interface**

```
java -jar dbaudit.jar /D oracleProfileName [/U user] [/P password] /A
mode:moveSysAudit;tablespace:<TABLESPACE>
```

**Parameters:**

oracleProfileName – required. Name of an existing database connection profile configured for an Oracle database.

user – optional. Name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<TABLESPACE> – required. Name of the tablespace to which you want to move SYS.AUD\$ table. Note that the same tablespace is used for both table data and table indexes.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D oracleProfileName /A /F
"mode:moveSysAudit;tablespace:AUDIT_TABLES"
```

**Using the API functions**

```
void auditSetup.moveSysAudOracle(String tableTableSpace, String indexTableSpace);
```

**Throws:**

java.sql.SQLException – if an error occurs.

**Parameters:**

tableTableSpace – name of the tablespace to which you want to move SYS.AUD\$ table.

indexTableSpace – name of the tablespace to which you want to move SYS.AUD\$ table indexes.

**Example:**

1. Connect to the database, for example:
 

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Oracle Profile");
con.connect("user", "password");
```
2. Create an instance of the AuditSetupSystemOracle class
 

```
AuditSetupSystemOracle auditSetup =
(AuditSetupSystemOracle)AuditSetupSystemCommon.getInstance(con);
```
3. Move AUD\$ table to AUDIT\_DATA and AUDIT\_INDEXES tablespaces
 

```
auditSetup.moveSysAudOracle("AUDIT_DATA", "AUDIT_INDEXES");
```

## Informational Methods

### *Get System Audit Status*

#### Using the console interface

```
java -jar dbaudit.jar /D oracleProfileName [/U user] [/P password] /A /F mode:isSysAuditRunning
```

This will print current audit enabled/disabled status to the console

#### Parameters:

`oracleProfileName` – required. Name of an existing database connection profile configured for an Oracle database.

`user` – optional. Name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used.

`password` – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

#### Example:

```
java -jar dbaudit.jar /D oracleProfileName /A /F "mode:isSysAuditRunning"
```

#### Using the API functions

```
boolean auditSetup.isSysAuditRunning();
```

This returns 'true' if the audit is currently enabled and 'false' otherwise.

#### Throws:

`java.sql.SQLException` – if an error occurs.

#### Parameters:

None

#### Example:

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Oracle Profile");  
con.connect("user", "password");
```
2. Create an instance of the `AuditSetupSystemOracle` class  

```
AuditSetupSystemOracle auditSetup =  
(AuditSetupSystemOracle)AuditSetupSystemCommon.getInstance(con);
```
3. Get system audit status  

```
boolean auditRunning = auditSetup.isSysAuditRunning();  
if (!auditRunning)  
    // run report ...  
else  
    // print error message ...
```

## Install DB Audit Mail Sending SQL Procedure

DB Audit mail sending procedure is used in real-time email alerts that can be sent by both system audit trail and data-change audit trail processes.

### Using the console interface

```
java -jar dbaudit.jar /D oracleProfileName [/U user] [/P password] /A /F  
mode:installDBAuditMailSendingSQLProcedure
```

Parameters:

oracleProfileName – required. Name of an existing database connection profile configured for an Oracle database.

user – optional. Name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

Example:

```
java -jar dbaudit.jar /D oracleProfileName /A /F mode:installDBAuditMailSendingSQLProcedure
```

### Using the API functions

```
void auditSetup.installDBAuditMailSendingSQLProcedure(String tablespaceDB2);
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

tablespaceDB2 – this parameter is not used with Oracle and must be passed as a null value.

Example:

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Oracle Profile");  
con.connect("user", "password");
```
2. Create an instance of the AuditSetupSystemOracle class  

```
AuditSetupSystemOracle auditSetup =  
(AuditSetupSystemOracle)AuditSetupSystemCommon.getInstance(con);
```
3. Install mail sending procedure  

```
auditSetup.installDBAuditMailSendingSQLProcedure(null);
```

## Microsoft SQL Server System Audit Management Tasks

### Set System Audit State

#### *Install System Audit*



**Important Notes:** Before you run the DB Audit installation method for Microsoft SQL Server, make sure you have **xp\_dbaudit.dll** file copied to your **[SQL Server home]\Binn** directory manually or programmed this step as part of your audit installation procedures. To copy this file you can use available operation system commands or other appropriate methods. The direct location of the BINN directory depends on your SQL Server version and installation path. The standard directory path for default SQL Server 2000 instance is *C:\Program Files\Microsoft SQL Server\MSSQL\Binn*. The standard directory path for default SQL Server 2005 instance is *C:\Program Files\Microsoft SQL Server 2005\MSSQL.1\Binn*.

Three versions of the xp\_dbaudit.dll are provided with the API:

- A version for SQL Server 32-bit running on 32-bit Windows system. This file can be found in the DB Audit API **[db\_audit\_api\_home]\sqlserver** directory.
- A version for SQL Server 64-bit running on 64-bit Windows system based on AMD64 or Intel EM64 and compatible processors with x86 instruction set. This file can be found in the DB Audit API **[db\_audit\_api\_home]\sqlserver\64\x86** directory.
- A version for SQL Server 64-bit running on 64-bit Windows system based on Intel Itanium processors and IA64 instruction set. This file can be found in the DB Audit API **[db\_audit\_api\_home]\sqlserver\64\IA64** directory.

Make sure to install the correct version of the DLL. Failure to do so will prevent the startup of the auditing process.

The following table describes operation types, which can be audited in SQL Server.

Operation Type	Comments
All	This is a catch-all name that can be used to enable or disable auditing of all operation types listed in this table.
Login (failed)	Indicates that a login attempt to SQL Server from a client failed.
Login (successful)	Occurs when a user successfully logs in to SQL Server.
Logout	Occurs when a user logs out of SQL Server.
Error	Indicates that error events have been logged in the SQL Server error log.
Exception	Indicates that an exception has occurred in SQL Server.
Privileged Operation (CREATE/DROP/etc...)	Occurs when a statement permission (such as CREATE TABLE) is used.
Schema Object Access	Occurs when an object permission (such as

	SELECT) is used, both successfully or unsuccessfully.
Schema Object Derived Permission	Occurs when a CREATE, ALTER, and DROP object commands are issued.
Backup/Restore	Occurs when a BACKUP or RESTORE command is issued.
DBCC	Occurs when DBCC commands are issued.
Grant/Deny/Revoke Privilege	Occurs every time a GRANT, DENY, REVOKE for a statement permission is issued by any user in SQL Server.
Grant/Deny/Revoke Object Access	Occurs every time a GRANT, DENY, REVOKE for an object permission is issued by any user in SQL Server.
Grant/Deny/Revoke Login	Occurs when a Microsoft Windows login right is added or removed; for sp_grantlogin, sp_revokelogin, and sp_denylogin.
Login Properties Change	Occurs when a property of a login, except passwords, is modified; for sp_defaultdb and sp_defaultlanguage.
Login Password Change	Occurs when a SQL Server login password is changed. Passwords are not recorded.
Role Password Change	Occurs when a password of an application role is changed.
Add/Drop Login	Occurs when a SQL Server login is added or removed; for sp_addlogin and sp_droplogin.
Add/Remove Login to Server Role	Occurs when a login is added or removed from a fixed server role; for sp_addsrvrolemember, and sp_dropsrvrolemember.
Add Database User	Occurs when a login is added or removed as a database user (Windows or SQL Server) to a database; for sp_grantdbaccess, sp_revokedbaccess, sp_adduser, and sp_dropuser.
Add/Drop Role Member	Occurs when a login is added or removed as a database user (fixed or user-defined) to a database; for sp_addrolemember, sp_droprolemember, and sp_changegroup.
Add/Drop Role	Occurs when a login is added or removed as a database user to a database; for sp_addrole and sp_droprole.
Tracing Settings Change	Occurs when trace modifications are made using SQL Profiler or stored procedures.

### Using the console interface



```
java -jar dbaudit.jar /D mssqlProfileName [/U user] [/P password] /A /F
mode:installSysAudit;operations:<OPERATIONS>;optionWhen:<OPTION_WHEN>;databases:
<DATABASES>;logins:<LOGINS>;osUsers:<OS_USERS>;dbusers:<DB_USERS>;osDomains:
<OS_DOMAINS>;hosts:<HOSTS>;applications:<APPLICATIONS>;objects:<OBJECTS>;query:
<QUERY>;auditSysObjects:<AUDIT_SYS_OBJECTS>;topLevel:
<AUDIT_TOP_LEVEL_QUERIES_ONLY>;ignoreSQLAgent:<IGNORE_SQL_AGENT>;
queueSize:<QUEUE_SIZE>
```

#### Parameters:

**mssqlProfileName** – required. Name of an existing database connection profile configured for a SQL Server database.

**user** – optional. Logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used.

**password** – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

**<OPERATIONS>** – required. Comma-separated list of types of operations to audit. The names of the supported operation types are listed in the Operations table in the beginning of this topic.



#### Important Notes:

- "All" is an exclusive type that cannot be used along with other. This is a special type that means auditing of all types of operations occurring in the database server.
- Operation type names must be entered exactly as they appear in the table.

**<WHEN OPTION>** – required. One of the following values: WHENEVER SUCCESSFUL, WHENEVER NOT SUCCESSFUL or ALWAYS

WHENEVER SUCCESSFUL can be used to audit only successful executions of the audited statement.

WHENEVER NOT SUCCESSFUL can be used to audit only unsuccessful executions of the audited statement.

ALWAYS can be used to audit both successful and unsuccessful executions of the audited statement.

**<DATABASES>** – optional. Comma-separated list of databases to audit. Used as an audit time filter. If not specified, all databases are audited.



**Important Note:** Certain operation types affecting the entire server are always audited regardless of the database filter.

**<LOGINS>** – optional. Comma-separated list of login names to audit. Used as an audit time filter. If not specified, all logins are audited.

**<OS\_USERS>** – optional. Comma-separated list of network user names to audit. This parameter is used as an audit time filter. If not specified, all users are audited.

**<DB\_USERS>** – optional. Comma-separated list of database user names to audit. This parameter is used as an audit time filter. If not specified, all users are audited.

**<OS\_DOMAINS>** – optional. Comma-separated list of network domain names whose users to audit. This parameter is used as an audit time filter. If not specified, all domain users are audited.

<HOSTS> – optional. Comma-separated list of connecting computer names to audit (as they appear in the HostName column returned by sp\_who2 system stored procedure). This parameter is used as an audit time filter. If not specified, all connections from all workstations are audited.

<APPLICATIONS> – optional. Comma-separated list of application names to audit (as they appear in the ProgramName column returned by sp\_who2 system stored procedure). This parameter is used as an audit time filter. If not specified, all applications are audited.

<OBJECTS> – optional. Comma-separated list of object names to audit. Specified names should be fully qualified and include schema name. This parameter is used as an audit time filter. If not specified, all objects are audited with an additional filtering controlled by <AUDIT\_SYS\_OBJECTS> and <AUDIT\_TOP\_LEVEL\_QUERIES\_ONLY> parameters.

<QUERY> – optional. A substring that must be present in queries to audit. This parameter is used as an audit time filter.

<AUDIT\_SYS\_OBJECTS> – optional. One of the following values: yes, no. The default value is "yes" meaning that access to system catalog objects is also audited.

<AUDIT\_TOP\_LEVEL\_QUERIES\_ONLY> – optional. One of the following values: yes, no. The default value is "no" meaning to all queries are subject to auditing including queries executed from within stored procedures, triggers and other code objects.

<IGNORE\_SQL\_AGENT> – optional. One of the following values: yes, no. The default value is "yes" meaning that special "keep alive" queries sent to the database every few seconds by the SQL Agent service need not be audited.

<QUEUE\_SIZE> – optional. Audit queue size. Default value is 2

Examples (the following must be entered on a single line):

```
java -jar dbaudit.jar /D mssqlProfileName /A /F "mode:installSysAudit;operations:
Grant/Deny/Revoke Privilege,Grant/Deny/Revoke Object Access,Grant/Deny/Revoke Login,Login
Properties Change,Login Password Change,Role Password Change,Add/Drop
Login,Add/Remove Login to Server Role,Add Database User,Add/Drop Role Member,Add/Drop
Role;optionWhen:ALWAYS;"
```

### Using the API functions

API: void auditSetup.installSysAudit(String[] operations, String globalOptions, String[] databases, String logins[], String[] osUsers, String[] users, String[] osDomains, String[] hosts, String[] applications, String[] objects, String query, boolean auditSysObjects, boolean topLevel, boolean ignoreSQLAgent, int queueSize);

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

operations – String array of types of operations to audit. The names of the supported operation types are listed in the Operations table in the beginning of this topic.



#### Important Notes:

- "All" is an exclusive type that cannot be used with other. In this context. This is a special item that means auditing of all types of operations occurring in the database server.
- Operation type names must be entered exactly as they appear in the list above.

globalOptions – One of the following values: WHENEVER SUCCESSFUL, WHENEVER NOT SUCCESSFUL or ALWAYS

WHENEVER SUCCESSFUL can be used to audit only successful executions of the audited statement.

WHENEVER NOT SUCCESSFUL can be used to audit only unsuccessful executions of the audited statement.

ALWAYS can be used to audit both successful and unsuccessful executions of the audited statement.

databases – String array of databases to audit. This parameter is used as an audit time filter. If null used for the parameter value, all databases are audited.



**Important Note:** Certain operation types affecting the entire server are always audited regardless of the database filter.

logins – String array of login names to audit. This parameter is used as an audit time filter. If null used for the parameter value, all logins are audited.

osUsers – String array of network user names to audit. This parameter is used as an audit time filter. If null used for the parameter value, all users are audited.

users – String array of database user names to audit. This parameter is used as an audit time filter. If null used for the parameter value, all users are audited.

osDomains – String array of network domain names whose users to audit. This parameter is used as an audit time filter. If null used for the parameter value, all domain users are audited.

hosts – String array of connecting computer names to audit (as they appear in the HostName column returned by sp\_who2 system stored procedure). This parameter is used as an audit time filter. If null used for the parameter value, all connections from all workstations are audited.

applications – String array of application names to audit (as they appear in the ProgramName column returned by sp\_who2 system stored procedure). This parameter is used as an audit time filter. If null used for the parameter value, all applications are audited.

objects – String array of object names to audit for access. This parameter is used as an audit time filter. If null used for the parameter value, all objects are audited with an additional filtering controlled by <AUDIT\_SYS\_OBJECTS> and <AUDIT\_TOP\_LEVEL\_QUERIES\_ONLY> parameters.

query – A substring that must be present in queries to audit. This parameter is used as an audit time filter.

auditSysObjects – true or false. True value means that access to system catalog objects is also audited.

topLevel – true or false. False value means that all queries are subject to auditing including queries executed from within stored procedures, triggers and other code objects.

ignoreSQLAgent – true or false. True value means that special "keep alive" queries sent to the database every few seconds by the SQL Agent service need not be audited.

queueSize – Audit queue size. Should be always set to 2.

Example:

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My MSSQL Profile");
con.connect("user", "password");
```


2. Create an instance of the AuditSetupSystemMssql class
 

```
AuditSetupSystemMssql auditSetup =
(AuditSetupSystemMssql)AuditSetupSystemCommon.getInstance(con);
```
3. Install mail sending procedure
 

```
auditSetup.installSysAudit( { "Grant/Deny/Revoke Privilege",
                              "Grant/Deny/Revoke Object Access",
                              "Grant/Deny/Revoke Login",
                              "Login Properties Change",
                              "Login Password Change",
                              "Role Password Change",
                              "Add/Drop Login,Add/Remove Login to Server Role",
                              "Add Database User,Add/Drop Role Member",
                              "Add/Drop Role"}, "ALWAYS", null, null, null, null,
                              null, null, null, null, null, false, false, true, 2 );
```

## Uninstall System Audit

Use the methods described in [Uninstall Audit Repository Objects](#) topic to uninstall both the system audit and data-change audit objects and other options and enhancements that you have previously installed using DB Audit API or DB Audit GUI.


 **Important Notes:** The "uninstall" method completely removes DB Audit from the database, it removes all previously installed audit settings, stored procedures and tables in the repository database **including data-change audit trail tables** and triggers and also removes db\_audit schema, user and login. The "uninstall" method does not remove xp\_dbaudit.dll file from the operation system. You should delete this file using available operation system commands or advise users how to delete that file manually.

## Enable System Audit

The auditing is enabled immediately after the installation. No special actions are required to enable it.

In case if the audit has been disabled using methods described in the [Disable System Audit](#) topic you can enable it again by re-executing system audit installation method but without installation of any additional files. See the [Install System Audit](#) topic for more details.

## Disable System Audit

 **Important Note:** This method removes previously installed audit stored procedures but does not remove audit settings and audit trail tables. Therefore, it effectively stops the auditing but keeps all the data and settings in place. In case if you want to enable the auditing later you can rerun the installation method and it will pickup all retained settings.

### Using the console interface

```
java -jar dbaudit.jar /D mssqlProfileName [/U user] [/P password] /A /F mode:disableSysAudit
```

**Parameters:**

`mssqlProfileName` – required. Name of an existing database connection profile configured for a SQL Server database.

`user` – optional. Logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used.

`password` – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D mssqlProfileName /A /F "mode:disableSysAudit"
```

**Using the API functions**

```
void auditSetup.disableSysAudit();
```

**Throws:**

`java.sql.SQLException` – if an error occurs.

**Parameters:**

None

**Example:**

1. Connect to the database, for example:  
*DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My MSSQL Profile");  
con.connect("user", "password");*
2. Create an instance of the `AuditSetupSystemMssql` class  
*AuditSetupSystemMssql auditSetup =  
(AuditSetupSystemMssql)AuditSetupSystemCommon.getInstance(con);*
3. Disable system audit  
*auditSetup.disableSysAudit();*

**Set Audit Operations and Filters*****Update Audit Operations***

This method updates current audit configuration, setting which database operations to audit. This method is more efficient than reinstalling the entire audit configuration and related procedures using Uninstall then Install methods. For more information on supported audited operation types, see [Install System Audit](#) topic.

**Using the console interface**

```
java -jar dbaudit.jar /D mssqlProfileName [/U user] [/P password] /A /F  
mode:updateAuditOperations;operations:<OPERATIONS>
```

**Parameters:**

`mssqlProfileName` – required. Name of an existing database connection profile configured for a SQL Server database.

`user` – optional. Logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used.

`password` – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

`<OPERATIONS>` – required. List of database operation types to audit. See [Install System Audit](#) topic for details.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D mssqlProfileName /A /F "mode:updateAuditOperations;operations:
Grant/Deny/Revoke Privilege,Grant/Deny/Revoke Object Access,Grant/Deny/Revoke Login,Login
Properties Change,Login Password Change,Role Password Change,Add/Drop
Login,Add/Remove Login to Server Role,Add Database User,Add/Drop Role Member,Add/Drop
Role,"
```

**Using the API functions**

```
void auditSetup.updateAuditOperations(String[] operations);
```

**Throws:**

`java.sql.SQLException` – if an error occurs.

**Parameters:**

`operations` – String array of database operation types to audit. See [Install System Audit](#) topic for details.

**Example:**

1. Connect to the database, for example:
 

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My MSSQL Profile");
con.connect("user", "password");
```
2. Create an instance of the `AuditSetupSystemMssql` class
 

```
AuditSetupSystemMssql auditSetup =
(AuditSetupSystemMssql)AuditSetupSystemCommon.getInstance(con);
```
3. Update audit settings
 

```
auditSetup.updateAuditOperations( { "Grant/Deny/Revoke Privilege",
                                   "Grant/Deny/Revoke Object Access",
                                   "Grant/Deny/Revoke Login",
                                   "Login Properties Change",
                                   "Login Password Change",
                                   "Role Password Change",
                                   "Add/Drop Login,Add/Remove Login to Server Role",
                                   "Add Database User,Add/Drop Role Member",
                                   "Add/Drop Role"} );
```

## Update Audit Filters

This method updates current audit configuration, setting audit filters. This method is more efficient than reinstalling the entire audit configuration and related procedures using Uninstall/Install methods. For more information on supported audit filters see [Install System Audit](#) topic.

### Using the console interface

```
java -jar dbaudit.jar /D mssqlProfileName [/U user] [/P password] /A /F mode:updateAuditFilters;
optionWhen:<OPTION_WHEN>;databases:<DATABASES>;logins:<LOGINS>;osUsers:
<OS_USERS>;dbusers:<DB_USERS>;osDomains:<OS_DOMAINS>;hosts:<HOSTS>;
applications:<APPLICATIONS>;objects:<OBJECTS>;query:<QUERY>;auditSysObjects:
<AUDIT_SYS_OBJECTS>;topLevel:<AUDIT_TOP_LEVEL_QUERIES_ONLY>;
ignoreSQLAgent:<IGNORE_SQL_AGENT>;
```

#### Parameters:

All parameters are the same as for Install System Audit. See [Install System Audit](#) topic for details.

The following parameters are optional. Not specified parameters are assumed as not used for filtering: <OPTION\_WHEN>, <DATABASES>, <LOGINS>, <OS\_USERS>, <DB\_USERS>, <OS\_DOMAINS>, <HOSTS>, <APPLICATIONS>, <OBJECTS>, <QUERY> .

The following parameters are required and must be provided on the command line: <AUDIT\_SYS\_OBJECTS>, <AUDIT\_TOP\_LEVEL\_QUERIES\_ONLY>, <IGNORE\_SQL\_AGENT>

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D mssqlProfileName /A /F "mode:
updateAuditFilters;logins:sa,john,peter;applications:SQL Query
Analyzer;auditSysObjects:yes;topLevel:no;ignoreSQLAgent:yes"
```

### Using the API functions

```
void auditSetup.updateAuditFilters(String globalOptions, String[] databases, String logins[],
String[] osUsers, String[] users, String[] domains, String[] hosts,
String[] applications, String[] objects, String query, boolean auditSysObjects,
boolean topLevel, boolean ignoreSQLAgent);
```

#### Throws:

java.sql.SQLException – if an error occurs.

#### Parameters:

All parameters are the same as for Install System Audit. See [Install System Audit](#) topic for details.

#### Example:

1. Connect to the database, for example:

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My MSSQL Profile");
con.connect("user", "password");
```
2. Create an instance of the AuditSetupSystemMssql class

```
AuditSetupSystemMssql auditSetup =
(AuditSetupSystemMssql)AuditSetupSystemCommon.getInstance(con);
```

- Update audit settings

```
auditSetup.updateAuditFilters("ALWAYS", null, {"sa", "john", "peter"}, null, null,
null, null, {"SQL Query Analyzer"}, null, null, false, false, true);
```

## Set Advanced Audit Options

### Update Audit Queue Size

This method updates current audit configuration, setting audit queue size. This method is more efficient than reinstalling the entire audit configuration and related procedures using Uninstall/Install methods.



**Important Note:** The default queue size is 2. This value should not be changed unless you are running audit in a very busy database and the auditing processes heavily affect the database performance.

#### Using the console interface

```
java -jar dbaudit.jar /D mssqlProfileName [/U user] [/P password] /A /F
mode:updateAuditQueueSize;queueSize:<QUEUE_SIZE>
```

Parameters:

mssqlProfileName – required. Name of an existing database connection profile configured for a SQL Server database.

user – optional. Logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<QUEUE\_SIZE> – required. New audit event queue size. Normally this parameter should be always set to 2.

Example:

```
java -jar dbaudit.jar /D mssqlProfileName /A /F "mode:updateAuditQueueSize;queueSize:2"
```

#### Using the API functions

```
void auditSetup.updateQueueSize(int queue_size);
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

queue\_size – New audit event queue size. Normally this parameter should be always set to 2.



Example:

1. Connect to the database, for example:  
*DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My MSSQL Profile");*  
*con.connect("user", "password");*
2. Create an instance of the AuditSetupSystemMssql class  
*AuditSetupSystemMssql auditSetup =*  
*(AuditSetupSystemMssql)AuditSetupSystemCommon.getInstance(con);*
3. Update audit settings  
*auditSetup.updateQueueSize(2);*

## Informational Methods

### ***Get Current Audit Settings***

Methods described in this topic can be used to obtain audit settings stored in the repository database. These methods provide an alternative to the use of "Get Global Audit Settings" report. They are specific to the Microsoft SQL Server environment while the mentioned report is generic and can be used with multiple database systems

#### **Using the console interface**

```
java -jar dbaudit.jar /D mssqlProfileName [/U user] [/P password] /A /F
mode:getAuditSettings;showFilters:<SHOW_FILTERS>
```

Parameters:

mssqlProfileName – required. Name of an existing database connection profile configured for a SQL Server database.

user – optional. Logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<SHOW\_FILTER> – required. Supported values: yes, no. If 'yes', both audited operation types and audit-time filters are printed to the console; if 'no' only audited operation types are printed to the screen. . For a list of operation type names see description of <OPERATIONS> parameter in [Install System Audit](#) topic in [Microsoft SQL Server System Audit Management](#) section of this chapter. In the same topic you can also find descriptions of supported audit filters.

Return:

XML formatted string containing separate elements for each audited operation type, audit queue size, and if filters are requested a separate element for each filter.

Example:

```
java -jar dbaudit.jar /D mssqlProfileName /A /F mode:getAuditSettings;showFilters:yes
```

**Using the API functions**

```
String auditSetup.getAuditSettings(boolean showFilters);
```

Returns:

Depending on the value of the **showFilters** parameter this method returns either both audited operation types and audit-time filters or only audited operation types. The result is a XML formatted string containing separate elements for each audited operation type, audit queue size, and if filters are requested a separate element for each filter.

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

All parameters are the same as for Install System Audit. See [Install System Audit](#) topic for details.

Example:

1. Connect to the database, for example:  
*DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My MSSQL Profile");  
con.connect("user", "password");*
2. Create an instance of the AuditSetupSystemMssql class  
*AuditSetupSystemMssql auditSetup =  
(AuditSetupSystemMssql)AuditSetupSystemCommon.getInstance(con);*
3. Get audit settings and filters  
*String settings = auditSetup.getAuditSettings (true);*

**Get System Audit Status****Using the console interface**

```
java -jar dbaudit.jar /D mssqlProfileName [/U user] [/P password] /A /F mode:isSysAuditRunning
```

This will print current audit enabled/disabled status to the console

Parameters:

mssqlProfileName – required. Name of an existing database connection profile configured for a SQL Server database.

user – optional. Logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

Example:

```
java -jar dbaudit.jar /D mssqlProfileName /A /F "mode:isSysAuditRunning"
```

**Using the API functions**

```
boolean auditSetup.isSysAuditRunning();
```

This returns 'true' if the audit is currently enabled and 'false' otherwise.

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

None

Example:

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My MSSQL Profile");
con.connect("user", "password");
```
2. Create an instance of the AuditSetupSystemMssql class  

```
AuditSetupSystemMssql auditSetup =
(AuditSetupSystemMssql)AuditSetupSystemCommon.getInstance(con);
```
3. Get system audit status  

```
boolean auditRunning = auditSetup.isSysAuditRunning();
if (!auditRunning)
    // run report ...
else
    // print error message ...
```

**Install DB Audit Mail Sending SQL Procedure**

DB Audit mail sending procedure is used in real-time email alerts that can be sent by both system audit trail and data-change audit trail processes.

**Using the console interface**

```
java -jar dbaudit.jar /D mssqlProfileName [/U user] [/P password] /A /F
mode:installDBAuditMailSendingSQLProcedure
```

Parameters:

mssqlProfileName – required. Name of an existing database connection profile configured for a SQL Server database.

user – optional. Logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

Example:

```
java -jar dbaudit.jar /D mssqlProfileName /A /F mode:installDBAuditMailSendingSQLProcedure
```

**Using the API functions**

```
void auditSetup.installDBAuditMailSendingSQLProcedure(String tablespaceDB2);
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

tablespaceDB2 – this parameter is not used with SQL Server and must be passed as a null value.

Example:

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My MSSQL Profile");  
con.connect("user", "password");
```
2. Create an instance of the AuditSetupSystemMssql class  

```
AuditSetupSystemMssql auditSetup =  
(AuditSetupSystemMssql)AuditSetupSystemCommon.getInstance(con);
```
3. Install mail sending procedure  

```
auditSetup.installDBAuditMailSendingSQLProcedure(null);
```

## Sybase SQL Server and ASE System Audit Management Tasks

### Set System Audit State

#### *Install System Audit*

Sybase SQL Server and ASE system audit is a native feature of Sybase databases that is installed with the sybsecurity system database using SQL scripts provided with your version of the Sybase database server in documented in the Database Administration manuals.

DB Audit provides several methods that can be used to enhance the default Sybase auditing and ensure the default system auditing is manageable. For more information, see the following topics:

[Add New System Audit Trail Table](#)

[Update Audit Queue Size](#)

[Attach Threshold Procedures](#)

[Set 'Suspend Audit When Device Full' Status](#)

#### *Uninstall System Audit*

Sybase system audit repository and procedures are native features of Sybase databases that are installed with the sybsecurity database.

Use the methods described in [Uninstall Audit Repository Objects](#) topic to uninstall the additional options and enhancements that you have previously installed using DB Audit API or DB Audit GUI.



**Important Notes:** The "uninstall" method completely removes DB Audit from the database, it removes all previously installed audit settings, DB Audit stored procedures and tables **including data-change audit trail tables** and triggers and also removes db\_audit schema and user. The "uninstall" method **does not remove Sybase native system audit trail tables**.

## Enable System Audit

This method turns on system auditing using previously configured audit settings and options. Note that auditing settings and scope can be changed at any time, regardless of whether the auditing is enabled or disabled.

### Using the console interface

```
java -jar dbaudit.jar /D sybaseProfileName [/U user] [/P password] /A /F mode:enableSysAudit
```

Parameters:

sybaseProfileName – required. Name of an existing database connection profile configured for a Sybase database.

user – optional. Logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

Example:

```
java -jar dbaudit.jar /D sybaseProfileName /A /F "mode:enableAudit "
```

### Using the API functions

```
void systemAudit.enableSystemAudit();
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

None

Example:

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Sybase Profile");  
con.connect("user", "password");
```

2. Create an instance of the AuditSetupSystemSybase class  

```
AuditSetupSystemSybase auditSetup =
(AuditSetupSystemSybase)AuditSetupSystemCommon.getInstance(con);
```
3. Enable system audit  

```
auditSetup.enableAudit();
```

### **Disable System Audit**

This method turns off system auditing but does not remove audit settings and audit trail tables. Therefore, it effectively stops the auditing but keeps all the data and settings in place. In case if you want to enable the auditing later you can use the method described in [Enable System Audit](#) topic and that method will pickup all retained settings.

#### **Using the console interface**

```
java -jar dbaudit.jar /D sybaseProfileName [/U user] [/P password] /A /F mode:disableSysAudit
```

Parameters:

sybaseProfileName – required. Name of an existing database connection profile configured for a Sybase database.

user – optional. Logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D sybaseProfileName /A /F "mode:disableSysAudit"
```

#### **Using the API functions**

```
void auditSetup.disableSysAudit();
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

None

Example:

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Sybase Profile");
con.connect("user", "password");
```

2. Create an instance of the AuditSetupSystemSybase class  

```
AuditSetupSystemSybase auditSetup =
(AuditSetupSystemSybase)AuditSetupSystemCommon.getInstance(con);
```
3. Disable system audit  

```
auditSetup.disableSysAudit();
```

## Set Audit Operations and Filters

### Add Server-level Operations Audit

This method turns on auditing of particular SQL operation types affecting the entire server, such as LOGINS, DISK operations (DISK INIT etc...), SERVER BOOTS, SECURITY and so on.

You can call this method multiple times adding different operation types with different options as required.

The following tables describe server-wide operation types, which can be audited in different Sybase versions.

#### Versions prior to 11.5

Operation Type	Comments
ALL	This is a catch-all name that can be used to enable or disable auditing of all operation types listed in this table.
LOGINS (failed)	Enables or disables auditing of failed login attempts by all users.
LOGINS (successful)	Enables or disables auditing of successful login attempts by all users.
LOGOUTS	Enables or disables auditing of all logouts from the server, including unintentional logouts such as dropped connections.
SERVER BOOTS	Enables or disables generation of an audit record when the server is rebooted.
RPC CONNECTIONS (failed)	Enables or disables auditing of failed RPC connection attempts (whenever a user from another host connects to the local server to run a procedure via a remote procedure call)
RPC CONNECTIONS (successful)	Enables or disables auditing of successful RPC connection attempts (whenever a user from another host connects to the local server to run a procedure via a remote procedure call)
ROLES (failed)	Enables or disables auditing of failed attempts to use of the set role command to turn roles on and off.
ROLES (successful)	Enables or disables auditing of successful attempts to use of the set role command to turn roles on and off.
PRIVILEGED ROLES (failed)	Enables or disables auditing of failed attempts to use of the set role command to turn sa/sso/oper role on and off.
PRIVILEGED ROLES (successful)	Enables or disables auditing of successful attempts to use of the set role command to turn sa/sso/oper role on and off.
ERRORS	Enables or disables auditing of user session errors.

FATAL ERRORS	Enables or disables auditing of user session fatal errors (errors that break the user's connection to the server and require the client program to be restarted)
ADHOC RECORDS	Enables or disables usage of sp_addauditrecord command (sp_addauditrecord allows users to send text to the audit trail)

**Version 11.5 and 12.x**

Operation Type	Comments
ALL	This is a catch-all name that can be used to enable or disable auditing of all operation types listed in this table.
LOGINS	Enables or disables auditing of login attempts by all users.
LOGOUTS	Enables or disables auditing of all logouts from the server, including unintentional logouts such as dropped connections.
RPC CONNECTIONS	Enables or disables auditing of RPC connection attempts (whenever a user from another host connects to the local server to run a procedure via a remote procedure call)
ERRORS	Enables or disables auditing of user session errors.
ADHOC	Enables or disables usage of sp_addauditrecord command (sp_addauditrecord allows users to send text to the audit trail)
SECURITY	Enables or disables auditing of server-wide security-relevant events, such as using kill command, online database, role toggling, server boots and shutdowns, and so on...
DBCC	Enables or disables auditing of all executions of the dbcc command.
DISK	Enables or disables auditing of all executions of the disk-related commands (disk init, disk refit, disk reinit, disk mirror, disk unmirror, disk remirror)
SERVER BOOTS	Enables or disables auditing of all server boots and shutdowns

**Version 15.0 and up**

Same as for Sybase version 12 (see table above) plus 3 additional operation types

Operation Type	Comments
QUIESQE	Enables/disables auditing of <b>quiesce database</b> commands
MOUNT	Enables/disables auditing of <b>mount database</b> commands
UNMOUNT	Enables/disables auditing of <b>unmount database</b> commands

**Using the console interface**

```
java -jar dbaudit.jar /D sybaseProfileName [/U user] [/P password] /A /F
mode:addAuditedGlobalOperations;operations:<OPERATIONS>;optionWhen:<OPTION_WHEN>
```



**Parameters:**

`sybaseProfileName` – required. Name of an existing database connection profile configured for a Sybase database.

`user` – optional. Logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used.

`password` – optional. Password to be used with the database connection. If specified, `user` parameter must be also specified and visa versa.

`<OPERATIONS>` – required. Comma-separated list of types of server-level operations to audit.

Note: Not supported values are silently ignored. For a list of supported values, see tables provided in the beginning of this topic.

`<WHEN OPTION>` – required. One of the following values: `WHENEVER SUCCESSFUL`, `WHENEVER NOT SUCCESSFUL` or `ALWAYS`

`WHENEVER SUCCESSFUL` can be used to audit only successful executions of the audited operation type.

`WHENEVER NOT SUCCESSFUL` can be used to audit only unsuccessful executions of the audited operation type.

`ALWAYS` can be used to audit both successful and unsuccessful executions of the audited operation type. This is the default option.

**Important Notes:**

- Operation type names must be entered exactly as they appear in the tables above.
- In Sybase versions prior to 11.5, the operation completion status has a limited use in the auditing filters. In some cases, the status is predefined in the operation type and because of that the `<WHEN OPTION>` value is not used. To filter on particular statuses use matching operation types. For example, to audit only failed logins use "LOGINS (failed)" or to audit both failed and successful logins specify both "LOGINS (failed)" and "LOGINS (successful)" values. It is recommended that when working with Sybase versions prior to 11.5 you specify `ALWAYS` as the value for the `<WHEN OPTION>` parameter.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D sybaseProfileName /A /F
mode:addAuditedGlobalOperations;operations:LOGINS,PRIVILEGED ROLES,RPC
CONNECTIONS,SECURITY;optionWhen:ALWAYS
```

**Using the API functions**

```
void addAuditedGlobalOperations(String[] operations, String optionWhen);
```

**Throws:**

`java.sql.SQLException` – if an error occurs.

**Parameters:**

`sybaseProfileName` – required. Name of an existing database connection profile configured for a Sybase database.

`user` – optional. Logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used.

`password` – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

`operations` – String array of types of operations to audit. Note: Not supported values are silently ignored. For a list of supported values, see tables provided in the beginning of this topic.

`optionWhen` – One of the following values: `WHENEVER SUCCESSFUL`, `WHENEVER NOT SUCCESSFUL` or `ALWAYS`

`WHENEVER SUCCESSFUL` can be used to audit only successful executions of the audited operation type.

`WHENEVER NOT SUCCESSFUL` can be used to audit only unsuccessful executions of the audited operation type.

`ALWAYS` can be used to audit both successful and unsuccessful executions of the audited operation type.

**Important Notes:**

- Operation type names must be entered exactly as they appear in the tables above.
- In Sybase versions prior to 11.5, the operation completion status has a limited use in the auditing filters. In some cases, the status is predefined in the operation type and because of that the `optionWhen` value is not used. To filter on particular statuses use matching operation types. For example, to audit only failed logins use "LOGINS (failed)" or to audit both failed and successful logins specify both "LOGINS (failed)" and "LOGINS (successful)" values. It is recommended that when working with Sybase versions prior to 11.5 you specify `ALWAYS` as the value for the `optionWhen` parameter.

**Example:**

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Sybase Profile");
con.connect("user", "password");
```
2. Create an instance of the `AuditSetupSystemSybase` class  

```
AuditSetupSystemSybase auditSetup =
(AuditSetupSystemSybase)AuditSetupSystemCommon.getInstance(con);
```
3. Enable auditing of several security related operation types  

```
auditSetup.addAuditedGlobalOperations ( { "LOGINS",
                                     "PRIVILEGED ROLES",
                                     "RPC CONNECTIONS,
                                     "SECURITY" }, "ALWAYS");
```

## Remove Server-level Operations Audit

This method turns off auditing of particular SQL operation types affecting the entire server database objects whose auditing has been previously enabled. See [Add Server-level Operations Audit](#) topic for more information.

You can call this method multiple times adding different operation types with different options as required or call it once specifying complete list of all required operations.

### Using the console interface

```
java -jar dbaudit.jar /D sybaseProfileName [/U user] [/P password] /A /F
mode:removeAuditedGlobalOperations;operations:<OPERATIONS>
```

#### Parameters:

`sybaseProfileName` – required. Name of an existing database connection profile configured for a Sybase database.

`user` – optional. Logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used.

`password` – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

`<OPERATIONS>` – same as in [Add Server-level Operations Audit](#)

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D sybaseProfileName /A /F
mode:removeAuditedGlobalOperations;operations:LOGINS,PRIVILEGED ROLES,RPC
CONNECTIONS,SECURITY
```

### Using the API functions

```
void removeAuditedGlobalOperations(String[] operations);
```

#### Throws:

`java.sql.SQLException` – if an error occurs.

#### Parameters:

`operations` – same as in [Add Server-level Operations Audit](#)

#### Example:

1. Connect to the database, for example:  
`DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Sybase Profile");`  
`con.connect("user", "password");`
2. Create an instance of the `AuditSetupSystemSybase` class  
`AuditSetupSystemSybase auditSetup =`  
`(AuditSetupSystemSybase)AuditSetupSystemCommon.getInstance(con);`

3. Disable auditing of several security related operation types  
`auditSetup.removeAuditedGlobalOperations ( { "LOGINS",  
"PRIVILEGED ROLES",  
"RPC CONNECTIONS",  
"SECURITY" } );`

### **Add Database-level Operations**

This method turns on auditing of particular SQL operation types affecting the entire database or could be applied to any object in the database.

You can call this method multiple times adding different operation types with different options as required.

The following tables describe database-wide operation types, which can be audited in different Sybase versions.

#### **Versions prior to 11.5**

<b>Operation Type</b>	<b>Comments</b>
ALL	This is a catch-all name that can be used to enable or disable auditing of all operation types listed in this table.
DB ACCESS	Enables/disables auditing of any access to the database from another database (execution of SQL commands from within another database that refer to objects in the audited database).
DROP	Enables/disables auditing of various object destructions: DROP DATABASE, DROP TABLE, DROP PROCEDURE, DROP TRIGGER, DROP VIEW.
GRANT	Enables/disables auditing of the GRANT commands.
REVOKE	Enables/disables auditing of the REVOKE commands.
TRUNCATE TABLE	Enables/disables auditing of the TRUNCATE TABLE commands.
USE	Enables/disables auditing of the USE database commands."

#### **Version 11.5 and 12.x**

<b>Operation Type</b>	<b>Comments</b>
ALL	This is a catch-all name that can be used to enable or disable auditing of all operation types listed in this table.
CREATE	Enables/disables auditing of various object creations: CREATE DATABASE, CREATE TABLE, CREATE PROCEDURE, CREATE TRIGGER, CREATE RULE, CREATE DEFAULT, EXEC SP_ADDMESSAGE, CREATE VIEW.
DB ACCESS (direct)	Enables/disables auditing of any access to the database from another database (execution of SQL commands from within another database that refer to objects in the audited database).
DB ACCESS (indirect using system-functions)	Enables/disables auditing of any access to the database via Transact-SQL built-in functions (execution of system functions

using system-functions)	from within another database that refer to objects in the audited database).
DROP	Enables/disables auditing of various object destructions: DROP DATABASE, DROP TABLE, DROP PROCEDURE, DROP TRIGGER, DROP RULE, DROP DEFAULT, EXEC SP_DROPMESSAGE, DROP VIEW.
DUMP	Enables/disables auditing of the DUMP DATABASE and DUMP TRANSACTION commands.
GRANT	Enables/disables auditing of the GRANT commands.
REVOKE	Enables/disables auditing of the REVOKE commands.
TRUNCATE TABLE	Enables/disables auditing of the TRUNCATE TABLE commands.
UNBIND	Enables/disables auditing of the SP_UNBINDEFULT, SP_UNBINDRULE, SP_UNBINDMSG stored procedures.
BIND	Enables/disables auditing of the SP_BINDEFULT, SP_BINDRULE, SP_BINDMSG stored procedures.
BCP	Enables/disables auditing of the BCP IN processes.
LOAD	Enables/disables auditing of the LOAD DATABASE, LOAD TRANSACTION commands.
SET USER	Enables/disables auditing of executions of the SETUSER command.
ALTER	Enables/disables auditing of various object changes: ALTER DATABASE, ALTER TABLE, ALTER VIEW.

**Version 15.0 and up**

Same as for Sybase version 12 (see table above) plus 2 additional operation types

Operation Type	Comments
INSTALL	Enables/disables auditing of installation of Java classes in database.
REMOVE	Enables/disables auditing of removal of Java classes in database.

**Using the console interface**

```
java -jar dbaudit.jar /D sybaseProfileName [/U user] [/P password] /A /F
mode:addAuditedDatabaseOperations;database:<DATABASE>;operations:<OPERATIONS>;optionW
hen:<OPTION_WHEN>
```

**Parameters:**

sybaseProfileName – required. Name of an existing database connection profile configured for a Sybase database.

user – optional. Logon name to be used with the database connection. If not specified, logon

name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<DATABASE> – required. Name of the database whose activities will be audited.

<OPERATIONS> – required. Comma-separated list of types of database-level operations to audit.

Note: Not supported values are silently ignored. For a list of supported values, see tables provided in the beginning of this topic. Operation type names must be entered exactly as they are documented.

<WHEN OPTION> – required. One of the following values: WHENEVER SUCCESSFUL, WHENEVER NOT SUCCESSFUL or ALWAYS

WHENEVER SUCCESSFUL can be used to audit only successful executions of the audited operation type.

WHENEVER NOT SUCCESSFUL can be used to audit only unsuccessful executions of the audited operation type.

ALWAYS can be used to audit both successful and unsuccessful executions of the audited operation type. This is the default option.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D sybaseProfileName /A /F  
mode:addAuditedDatabaseOperations;database:pubs;operations:ALTER,DROP,CREATE;option  
When:ALWAYS
```

This command enables auditing of all DDL operation types in database PUBS.

### Using the API functions

```
void addAuditedDatabaseOperations(String database, String[] operations, String optionWhen);
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

database – Name of the database whose activities will be audited.

operations – String array of types of database-level operations to audit. Note: Not supported values are silently ignored. For a list of supported values, see tables provided in the beginning of this topic.

optionWhen – One of the following values: WHENEVER SUCCESSFUL, WHENEVER NOT SUCCESSFUL or ALWAYS

WHENEVER SUCCESSFUL can be used to audit only successful executions of the audited operation type.

WHENEVER NOT SUCCESSFUL can be used to audit only unsuccessful executions of the audited operation type.

ALWAYS can be used to audit both successful and unsuccessful executions of the audited operation type.

Example:

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Sybase Profile");
con.connect("user", "password");
```
2. Create an instance of the AuditSetupSystemSybase class  

```
AuditSetupSystemSybase auditSetup =
(AuditSetupSystemSybase)AuditSetupSystemCommon.getInstance(con);
```
3. Enable auditing of all DDL operation types in database PUBS  

```
auditSetup.addAuditedDatabaseOperations( "pubs",
{ "ALTER", "DROP", "CREATE" },
"ALWAYS");
```

### **Remove Database-level Operations Audit**

This method turns off auditing of particular SQL operation types affecting the entire database whose auditing has been previously enabled. See [Add Database-level Operations](#) topic for more information.

You can call this method multiple times removing different operation as required or call it once specifying complete list of all required operations.

#### **Using the console interface**

```
java -jar dbaudit.jar /D sybaseProfileName [/U user] [/P password] /A /F
mode:removeAuditedDatabaseOperations;database:<DATABASE>;operations:<OPERATIONS>
```

Parameters:

sybaseProfileName – required. Name of an existing database connection profile configured for a Sybase database.

user – optional. Logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<DATABASE> – required. Name of the database whose activities will not be audited.

<OPERATIONS> – required. Comma-separated list of types of database-level operations not to be audited.

Note: Not supported values are silently ignored. For a list of supported values, see tables provided in the beginning of this topic. Operation type names must be entered exactly as they are documented.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D sybaseProfileName /A /F
mode:removeAuditedDatabaseOperations;database:pubs;operations:ALTER,DROP,CREATE
```

This command disables auditing of all DDL operation types in database PUBS.

### Using the API functions

```
void removeAuditedDatabaseOperations(String database, String[] operations);
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

database – Name of the database whose activities will not be audited.

operations – String array of types of database-level operations not to audit. Note: Not supported values are silently ignored. For a list of supported values, see tables provided in the beginning of this topic.

Example:

1. Connect to the database, for example:  
*DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Sybase Profile");*  
*con.connect("user", "password");*
2. Create an instance of the AuditSetupSystemSybase class  
*AuditSetupSystemSybase auditSetup =*  
*(AuditSetupSystemSybase)AuditSetupSystemCommon.getInstance(con);*
3. Disable auditing of all DDL operation types in database PUBS  
*auditSetup.removeAuditedDatabaseOperations("pubs", { "ALTER", "DROP", "CREATE" });*

### Add Schema Object-level Audit

This method turns on auditing of particular SQL operation types affecting schema objects.

You can call this method multiple times adding different operation types with different options as required.

The following tables describe operation types, which can be audited in different Sybase versions.

#### Versions prior to 11.5

Operation Type	Comments
ALL	This is a catch-all name that can be used to enable or disable auditing of all operation types listed in this table.
SELECT	Enables/disables auditing of SELECT operations for tables and views.
INSERT	Enables/disables auditing of INSERT operations for tables and



	views.
DELETE	Enables/disables auditing of DELETE operations for tables and views.
UPDATE	Enables/disables auditing of UPDATE operations for tables and views.
EXECUTE	Enables/disables auditing of EXECUTE operations for procedures and triggers.

**Versions 11.5 and later**

Operation Type	Comments
ALL	This is a catch-all name that can be used to enable or disable auditing of all operation types listed in this table.
SELECT	Enables/disables auditing of SELECT operations for tables and views.
INSERT	Enables/disables auditing of INSERT operations for tables and views.
DELETE	Enables/disables auditing of DELETE operations for tables and views.
UPDATE	Enables/disables auditing of UPDATE operations for tables and views.
EXECUTE PROCEDURE	Enables/disables auditing of EXECUTE procedure operations.
EXECUTE TRIGGER	Enables/disables auditing of trigger operations.
ACCESS	Enables/disables auditing of any access to the object via Transact-SQL built-in functions (execution of system functions that refer to the chosen objects).
REFERENCE	Enables/disables auditing of creating of a reference between tables involving the chosen object.

**Using the console interface**

```
java -jar dbaudit.jar /D sybaseProfileName [/U user] [/P password] /A /F
mode:addAuditedSchemaObject;database:<DATABASE>;object:<OBJECT>;objectType:<OBJECT_
TYPE>;operations:<OPERATIONS>;optionWhen:<OPTION_WHEN>
```

**Parameters:**

sybaseProfileName – required. Name of an existing database connection profile configured for a Sybase database.

user – optional. Logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<DATABASE> – required. Name of the database containing the object.

<OBJECT> – required. Full object name in dot notation such as SCHEMA\_NAME.OBJECT\_NAME.

<OBJECT\_TYPE> – required. Type of the object such as TABLE, PROCEDURE, etc...

<OPERATIONS> – required. Comma-separated list of types of object-level operations to audit.

Note: Not supported values are silently ignored. For a list of supported values, see tables provided in the beginning of this topic. Operation type names must be entered exactly as they are documented.

<WHEN OPTION> – required. One of the following values: WHENEVER SUCCESSFUL, WHENEVER NOT SUCCESSFUL or ALWAYS

WHENEVER SUCCESSFUL can be used to audit only successful executions of the audited operation type.

WHENEVER NOT SUCCESSFUL can be used to audit only unsuccessful executions of the audited operation type.

ALWAYS can be used to audit both successful and unsuccessful executions of the audited operation type. This is the default option.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D sybaseProfileName /A /F  
mode:addAuditedSchemaObject;database:pubs;object:dbo.MyTable;objectType:TABLE;operation  
s:SELECT,DELETE;optionWhen:WHENEVER NOT SUCCESSFUL
```

This command enables auditing of SELECT and DELETE operations on table dbo.MyTable in database PUBS whenever these operations fail.

### Using the API functions

```
void auditSetup.addAuditedSchemaObject(String database, String object, String objectType, String[]  
operations, String optionWhen);
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

database – Name of the database containing the object whose activities will be audited.

object – Full object name in dot notation such as SCHEMA\_NAME.OBJECT\_NAME

objectType – Type of the object such as TABLE, PROCEDURE, etc...

operations – String array of types of database-level operations to audit. Note: Not supported values are silently ignored. For a list of supported values, see tables provided in the beginning of

this topic.

optionWhen – One of the following values: WHENEVER SUCCESSFUL, WHENEVER NOT SUCCESSFUL or ALWAYS

WHENEVER SUCCESSFUL can be used to audit only successful executions of the audited operation type.

WHENEVER NOT SUCCESSFUL can be used to audit only unsuccessful executions of the audited operation type.

ALWAYS can be used to audit both successful and unsuccessful executions of the audited operation type.

Example:

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Sybase Profile");
con.connect("user", "password");
```
2. Create an instance of the AuditSetupSystemSybase class  

```
AuditSetupSystemSybase auditSetup =
(AuditSetupSystemSybase)AuditSetupSystemCommon.getInstance(con);
```
3. Enable auditing of all failed SELECT and DELETE operations for table dbo.MyTable in database PUBS  

```
auditSetup.addAuditedSchemaObject( "pubs", "dbo.MyTable", "TABLE",
{ "SELECT", "DELETE" },
"WHENEVER NOT SUCCESSFUL");
```

### **Remove Schema Object-level Audit**

This method turns off auditing of particular SQL operation types affecting schema objects whose auditing has been previously enabled. See [Add Schema Object-level Audit](#) topic for more information.

You can call this method multiple times removing different operation types for different objects as required.

#### **Using the console interface**

```
java -jar dbaudit.jar /D sybaseProfileName [/U user] [/P password] /A /F
mode:removeAuditedSchemaObject;database:<DATABASE>;object:<OBJECT>;operations:<OPERATIONS>
```

Parameters:

sybaseProfileName – required. Name of an existing database connection profile configured for a Sybase database.

user – optional. Logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<DATABASE> – required. Name of the database containing the object.

<OBJECT> – required. Full object name in dot notation such as SCHEMA\_NAME.OBJECT\_NAME.

<OPERATIONS> – required. Comma-separated list of types of database-level operations that should not be audited.

Note: Not supported values are silently ignored. For a list of supported values, see tables provided in the beginning of this topic. Operation type names must be entered exactly as they are documented.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D sybaseProfileName /A /F  
mode:removeAuditedSchemaObject;database:pubs;object:dbo.MyTable;objectType:TABLE;opera  
tions:SELECT,DELETE;optionWhen:ALWAYS
```

This command disables auditing of all SELECT and DELETE operations for table dbo.MyTable in database PUBS.

### Using the API functions

```
void auditSetup.removeAuditedSchemaObject(String database, String object, String[] operations,  
String optionWhen);
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

database – Name of the database containing the object whose activities will not be audited.

object – Full object name in dot notation such as SCHEMA\_NAME.OBJECT\_NAME

operations – String array of types of object-level operations to audit. Note: Not supported values are silently ignored. For a list of supported values, see tables provided in the beginning of this topic.

optionWhen – One of the following values: WHENEVER SUCCESSFUL, WHENEVER NOT SUCCESSFUL or ALWAYS

WHENEVER SUCCESSFUL can be used to audit only successful executions of the audited operation type.

WHENEVER NOT SUCCESSFUL can be used to audit only unsuccessful executions of the audited operation type.

ALWAYS can be used to audit both successful and unsuccessful executions of the audited operation type.

Example:

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Sybase Profile");
con.connect("user", "password");
```
2. Create an instance of the AuditSetupSystemSybase class  

```
AuditSetupSystemSybase auditSetup =
(AuditSetupSystemSybase)AuditSetupSystemCommon.getInstance(con);
```
3. Disable auditing of all SELECT and DELETE operations for table dbo.MyTable in database PUBS  

```
auditSetup.removeAuditedSchemaObject( "pubs", "dbo.MyTable",
{ "SELECT", "DELETE" },
"ALWAYS" );
```

### Add Login-level Audit

This method turns on auditing of particular SQL operation types for a logon.

You can call this method multiple times adding different operation for different logons and options as required.

The following tables describe types of operations that can be audited for a logon.

Operation Type	Comments
ALL	This is a catch-all name that can be used to enable or disable auditing of all operation types listed in this table.
TABLE ACCESS	Enables/disables auditing of tables accessible by the selected login (SELECT, DELETE, UPDATE, or INSERT access in a table).
VIEW ACCESS	Enables/disables auditing of view accessible by the selected login (SELECT, DELETE, UPDATE, or INSERT access in a view).
SQL TEXT	Enables/disables auditing the text of commands that a user sends to the database.

### Using the console interface

```
java -jar dbaudit.jar /D sybaseProfileName [/U user] [/P password] /A /F
mode:addAuditedLogin;login:<LOGIN>;operations:<OPERATIONS>;optionWhen:<OPTION_WHEN>
```

Parameters:

sybaseProfileName – required. Name of an existing database connection profile configured for a Sybase database.

user – optional. Logon name to be used with the database connection. If not specified, logon

name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<LOGIN> – required. Database login whose activities need to be audited.

<OPERATIONS> – required. Comma-separated list of login-level types of operations to audit.

Note: Not supported values are silently ignored. For a list of supported values, see tables provided in the beginning of this topic. Operation type names must be entered exactly as they are documented.

<WHEN OPTION> – required. One of the following values: WHENEVER SUCCESSFUL, WHENEVER NOT SUCCESSFUL or ALWAYS

WHENEVER SUCCESSFUL can be used to audit only successful executions of the audited operation type.

WHENEVER NOT SUCCESSFUL can be used to audit only unsuccessful executions of the audited operation type.

ALWAYS can be used to audit both successful and unsuccessful executions of the audited operation type. This is the default option.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D sybaseProfileName /A /F  
mode:addAuditedLogin;login:scott;operations:TABLE ACCESS,VIEW  
ACCESS;optionWhen:WHENEVER NOT SUCCESSFUL
```

This command enables auditing of table and view access for login SCOTT whenever these operations fail.

### Using the API functions

```
void auditSetup.addAuditedLogin(String login, String[] operations, String optionWhen);
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

login – Database login whose activities need to be audited

operations – String array of types of login-level operations to audit. Note: Not supported values are silently ignored. For a list of supported values, see tables provided in the beginning of this topic.

optionWhen – One of the following values: WHENEVER SUCCESSFUL, WHENEVER NOT SUCCESSFUL or ALWAYS

WHENEVER SUCCESSFUL can be used to audit only successful executions of the audited operation type.

WHENEVER NOT SUCCESSFUL can be used to audit only unsuccessful executions of the

audited operation type.

ALWAYS can be used to audit both successful and unsuccessful executions of the audited operation type.

Example:

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Sybase Profile");
con.connect("user", "password");
```
2. Create an instance of the AuditSetupSystemSybase class  

```
AuditSetupSystemSybase auditSetup =
(AuditSetupSystemSybase)AuditSetupSystemCommon.getInstance(con);
```
3. Enable auditing of all failed table and view accesses for login SCOTT  

```
auditSetup.addAuditedLogin( "scott",
{ "TABLE ACCESS", "VIEW ACCESS" },
"WHENEVER NOT SUCCESSFUL");
```

### **Remove Login-level Audit**

This method turns off auditing of particular SQL operation types executed by specific logins whose auditing has been previously enabled. See [Add Login-level Audit](#) topic for more information.

You can call this method multiple types removing different operation types for different logins as required.

#### **Using the console interface**

```
java -jar dbaudit.jar /D sybaseProfileName [/U user] [/P password] /A /F
mode:removeAuditedLogin;login:<LOGIN>;operations:<OPERATIONS>
```

Parameters:

sybaseProfileName – required. Name of an existing database connection profile configured for a Sybase database.

user – optional. Logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<LOGIN> – required. Database login whose activities do need to be audited anymore.

<OPERATIONS> – required. Comma-separated list of login-level types of operations to audit.

Note: Not supported values are silently ignored. For a list of supported values, see tables provided in the beginning of this topic. Operation type names must be entered exactly as they are documented.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D sybaseProfileName /A /F
mode:removeAuditedLogin;login:scott;operations:TABLE ACCESS,VIEW ACCESS
```

This command disables auditing of table and view access for login SCOTT.

### Using the API functions

```
void auditSetup.removeAuditedLogin(String login, String[] operations);
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

login – Database login whose activities do not need to be audited anymore

operations – String array of types of login-level operations to audit. Note: Not supported values are silently ignored. For a list of supported values, see tables provided in the beginning of this topic.

Example:

1. Connect to the database, for example:  
*DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Sybase Profile");*  
*con.connect("user", "password");*
2. Create an instance of the AuditSetupSystemSybase class  
*AuditSetupSystemSybase auditSetup =*  
*(AuditSetupSystemSybase)AuditSetupSystemCommon.getInstance(con);*
3. Disable auditing of all table and view accesses for login SCOTT  
*auditSetup.removeAuditedLogin("scott", {"TABLE ACCESS", "VIEW ACCESS"});*

## Set Advanced Audit Options

### Update Audit Queue Size

Audit queue size parameter establishes the size of the audit queue. Because this parameter affects audit event processing, changing this parameter does not take effect until ASE is restarted. The default size of the audit queue is 100 event records. The value can be in the 1 to 65,335 range. The large size you set for the audit queue, the more memory is required for the audit cache and longer it may take for the audit record to be flushed to the disk. Also note that the memory is taken from the total memory allocated to the data cache. On the other hand, larger queue size improve the overall process performance because the caching of audit records and flushing them to disk is performed by 2 independent asynchronous processes.



**Using the console interface**

```
java -jar dbaudit.jar /D sybaseProfileName [/U user] [/P password] /A /F
mode:updateQueueSize;queueSize:<QUEUE_SIZE>
```

## Parameters:

sybaseProfileName – required. Name of an existing database connection profile configured for a Sybase database.

user – optional. Logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<QUEUE\_SIZE> – required. Maximum number of queued events before they are flushed to disk.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D sybaseProfileName /A /F mode:updateQueueSize;queueSize:500
```

**Using the API functions**

```
void auditSetup.updateQueueSize(int queue_size);
```

## Throws:

java.sql.SQLException – if an error occurs.

## Parameters:

queueSize – Maximum number of queued events before they are flushed to disk.

## Example:

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Sybase Profile");
con.connect("user", "password");
```
2. Create an instance of the AuditSetupSystemSybase class  

```
AuditSetupSystemSybase auditSetup =
(AuditSetupSystemSybase)AuditSetupSystemCommon.getInstance(con);
```
3. Set new queue size  

```
auditSetup.updateQueueSize( 500 );
```

**Set 'Suspend Audit When Device Full' Status**

This method controls what ASE does when an audit device becomes completely full. If "suspend" is enabled and the device is full, all auditing processes stop until a user with SSO permissions logs on to the system and clears the audit trail tables. If disabled, ASE truncates the next audit table and starts

using it. If that table hasn't been archived, old audit trail records become lost.

### Using the console interface

```
java -jar dbaudit.jar /D sybaseProfileName [/U user] [/P password] /A /F  
mode:setSuspendAuditWhenDeviceFull;option:<OPTION>
```

Parameters:

`sybaseProfileName` – required. Name of an existing database connection profile configured for a Sybase database.

`user` – optional. Logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used.

`password` – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

`<OPTION>` – required. Controls whether to suspend or not to suspend the audit when the device storing audit tables becomes full. Supported values: yes, no.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D sybaseProfileName /A /F  
mode:setSuspendAuditWhenDeviceFull;option:no
```

### Using the API functions

```
void auditSetup.setSuspendAuditWhenDeviceFull(boolean value);
```

Throws:

`java.sql.SQLException` – if an error occurs.

Parameters:

`value` – Controls whether to suspend or not to suspend the audit when the device storing audit tables becomes full.

Example:

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Sybase Profile");  
con.connect("user", "password");
```
2. Create an instance of the `AuditSetupSystemSybase` class  

```
AuditSetupSystemSybase auditSetup =  
(AuditSetupSystemSybase)AuditSetupSystemCommon.getInstance(con);
```
3. Turn off the 'suspend' option  

```
auditSetup.setSuspendAuditWhenDeviceFull( false );
```

## Add New System Audit Trail Table

This method adds new audit trail table to existing audit trail configuration, expanding the total size of the audit trail.

### Using the console interface

```
java -jar dbaudit.jar /D sybaseProfileName [/U user] [/P password] /A /F
mode:addSysAuditTrailTable;device:<DEVICE>
```

#### Parameters:

sybaseProfileName – required. Name of an existing database connection profile configured for a Sybase database.

user – optional. Logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<DEVICE> – required. Name of the device where the table will be created.

#### Example:

```
java -jar dbaudit.jar /D sybaseProfileName /A /F mode:addSysAuditTrailTable;device:aud_space
```

### Using the API functions

```
void auditSetup.addSysAuditTrailTable(String device);
```

#### Throws:

java.sql.SQLException – if an error occurs.

#### Parameters:

device – Name of the device where the table will be created.

#### Example:

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Sybase Profile");
con.connect("user", "password");
```
2. Create an instance of the AuditSetupSystemSybase class  

```
AuditSetupSystemSybase auditSetup =
(AuditSetupSystemSybase)AuditSetupSystemCommon.getInstance(con);
```
3. Add new table  

```
auditSetup.addSysAuditTrailTable( 'aud_space' );
```

## Attach Threshold Procedures

This method installs DB Audit's so called "threshold" stored procedures and associates them with audit trail tables in the sybsecurity database. This "threshold" procedures get automatically executed when the audit trail tables become nearly full. When executed the threshold procedures automatically switch current audit trail table to the next available then archive the full table and truncate it freeing the space.

### Using the console interface

```
java -jar dbaudit.jar /D sybaseProfileName [/U user] [/P password] /A /F
mode:attachThresholdProcedures
```

#### Parameters:

sybaseProfileName – required. Name of an existing database connection profile configured for a Sybase database.

user – optional. Logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

#### Example:

```
java -jar dbaudit.jar /D sybaseProfileName /A /F mode:attachThresholdProcedures
```

### Using the API functions

```
void auditSetup.attachThresholdProcedures();
```

#### Throws:

java.sql.SQLException – if an error occurs.

#### Parameters:

None.

#### Example:

1. Connect to the database, for example:
 

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Sybase Profile");
con.connect("user", "password");
```
2. Create an instance of the AuditSetupSystemSybase class
 

```
AuditSetupSystemSybase auditSetup =
(AuditSetupSystemSybase)AuditSetupSystemCommon.getInstance(con);
```
3. Install threshold procedures and attach them to the audit trail tables
 

```
auditSetup.attachThresholdProcedures();
```

## **Uninstall Threshold Procedures**

This method detaches and uninstalls previously installed DB Audit's "threshold" stored procedures

### **Using the console interface**

```
java -jar dbaudit.jar /D sybaseProfileName [/U user] [/P password] /A /F  
mode:uninstallThresholdProcedures
```

Parameters:

sybaseProfileName – required. Name of an existing database connection profile configured for a Sybase database.

user – optional. Logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

Example:

```
java -jar dbaudit.jar /D sybaseProfileName /A /F mode:uninstallThresholdProcedures
```

### **Using the API functions**

```
void auditSetup.uninstallThresholdProcedures();
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

None.

Example:

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Sybase Profile");  
con.connect("user", "password");
```
2. Create an instance of the AuditSetupSystemSybase class  

```
AuditSetupSystemSybase auditSetup =  
(AuditSetupSystemSybase)AuditSetupSystemCommon.getInstance(con);
```
3. Uninstall threshold procedures  

```
auditSetup.uninstallThresholdProcedures();
```

## Informational Methods

### ***Get System Audit Status***

This method reports current status of the system audit process. It can be used to determine whether the auditing is currently enabled or not.

#### **Using the console interface**

```
java -jar dbaudit.jar /D sybaseProfileName [/U user] [/P password] /A /F mode:isSysAuditEnabled
```

#### Parameters:

sybaseProfileName – required. Name of an existing database connection profile configured for a Sybase database.

user – optional. Logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

#### Return:

Current system audit status is printed to the screen, yes/no

#### Example:

```
java -jar dbaudit.jar /D sybaseProfileName /A /F mode:isSysAuditEnabled
```

#### **Using the API functions**

```
boolean auditSetup.isSysAuditEnabled();
```

#### Throws:

java.sql.SQLException – if an error occurs.

#### Parameters:

None.

#### Return:

Current system audit status

#### Example:

1. Connect to the database, for example:  
*DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Sybase Profile");  
con.connect("user", "password");*
2. Create an instance of the AuditSetupSystemSybase class  
*AuditSetupSystemSybase auditSetup =  
(AuditSetupSystemSybase)AuditSetupSystemCommon.getInstance(con);*

- Determine if sys audit is enabled  
`System.out.println( auditSetup.isSysAuditEnabled( ) );`

### **Get System Audit Trail Tables Count**

This method returns number of audit trail tables used by the system audit processes in sysbsecurity database.

#### **Using the console interface**

```
java -jar dbaudit.jar /D sybaseProfileName [/U user] [/P password] /A /F  
mode:getSysAuditTrailTablesCount
```

#### Parameters:

`sybaseProfileName` – required. Name of an existing database connection profile configured for a Sybase database.

`user` – optional. Logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used.

`password` – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

#### Return:

Number of audit trail tabled used by the system auditing is printed to the screen

#### Example:

```
java -jar dbaudit.jar /D sybaseProfileName /A /F mode:getSysAuditTrailTablesCount
```

#### **Using the API functions**

```
int auditSetup.getSysAuditTrailTablesCount();
```

#### Throws:

`java.sql.SQLException` – if an error occurs.

#### Parameters:

None.

#### Return:

Number of audit trail tabled used by the system auditing

#### Example:

- Connect to the database, for example:  
`DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Sybase Profile");  
con.connect("user", "password");`

2. Create an instance of the AuditSetupSystemSybase class  
*AuditSetupSystemSybase auditSetup =  
(AuditSetupSystemSybase)AuditSetupSystemCommon.getInstance(con);*
3. Get table count  
*System.out.println( auditSetup.getSysAuditTrailTablesCount( ) );*

### **Get Audit Queue Size**

This method returns current ASE audit queue size.

#### **Using the console interface**

```
java -jar dbaudit.jar /D sybaseProfileName [/U user] [/P password] /A /F mode:getQueueSize
```

Parameters:

sybaseProfileName – required. Name of an existing database connection profile configured for a Sybase database.

user – optional. Logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

Return:

Audit queue size – maximum number of audit events that can be queued before they are flushed to the disk. This number is printed to the screen.

Example:

```
java -jar dbaudit.jar /D sybaseProfileName /A /F mode:getQueueSize
```

#### **Using the API functions**

```
int auditSetup.getQueueSize ();
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

None.

Return:

Audit queue size – maximum number of audit events that can be queued before they are flushed to the disk.



Example:

1. Connect to the database, for example:  
*DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Sybase Profile");*  
*con.connect("user", "password");*
2. Create an instance of the AuditSetupSystemSybase class  
*AuditSetupSystemSybase auditSetup =*  
*(AuditSetupSystemSybase)AuditSetupSystemCommon.getInstance(con);*
3. Get queue size  
*System.out.println( auditSetup.getQueueSize( ) );*

### **Get 'Suspend Audit When Device Full' Status**

This method returns current state of "suspend audit when device full" option. The "suspend" option controls what ASE does when an audit device becomes completely full. If "suspend" is enabled and the device is full, all auditing processes stop until a user with SSO permissions logs on to the system and clears the audit trail tables. If disabled, ASE truncates the next audit table and starts using it. If that table hasn't been archived, old audit trail records become lost.

#### **Using the console interface**

```
java -jar dbaudit.jar /D sybaseProfileName [/U user] [/P password] /A /F
mode:getSuspendAuditWhenDeviceFull
```

Parameters:

sybaseProfileName – required. Name of an existing database connection profile configured for a Sybase database.

user – optional. Logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

Return:

State of "suspend audit when device full" option value is printed to the screen: yes/no

Example:

```
java -jar dbaudit.jar /D sybaseProfileName /A /F mode:getSuspendAuditWhenDeviceFull
```

#### **Using the API functions**

```
boolean auditSetup.getSuspendAuditWhenDeviceFull();
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

None.

Return:

Current value of "suspend audit when device full" option

Example:

1. Connect to the database, for example:  
*DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Sybase Profile");*  
*con.connect("user", "password");*
2. Create an instance of the AuditSetupSystemSybase class  
*AuditSetupSystemSybase auditSetup =*  
*(AuditSetupSystemSybase)AuditSetupSystemCommon.getInstance(con);*
3. Get "suspend" option value  
*System.out.println( auditSetup.getSuspendAuditWhenDeviceFull( ) );*

## Install DB Audit Mail Sending SQL Procedure

DB Audit mail sending procedure is used in real-time email alerts that can be sent by both system audit trail and data-change audit trail processes.

### Using the console interface

```
java -jar dbaudit.jar /D SybaseProfileName [/U user] [/P password] /A /F
mode:installDBAuditMailSendingSQLProcedure
```

Parameters:

sybaseProfileName – required. Name of an existing database connection profile configured for a Sybase database.

user – optional. Logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

Example:

```
java -jar dbaudit.jar /D sybaseProfileName /A /F mode:installDBAuditMailSendingSQLProcedure
```

### Using the API functions

```
void auditSetup.installDBAuditMailSendingSQLProcedure(string tablespaceDB2);
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

tablespaceDB2 – this parameter is not used with ASE and must be passed as a null value.

Example:

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My Sybase Profile");
con.connect("user", "password");
```
2. Create an instance of the AuditSetupSystemSybase class  

```
AuditSetupSystemSybase auditSetup =
(AuditSetupSystemSybase)AuditSetupSystemCommon.getInstance(con);
```
3. Install mail sending procedure  

```
auditSetup.installDBAuditMailSendingSQLProcedure(null);
```

## DB2 System Audit Management Tasks



**Important Notes:** Before you run the DB Audit installation method for DB2, make sure you have done the following steps by running appropriate operation system commands manually or programming them as part of your audit installation procedures:

1. Copy **dbauditRunner.jar** file to your **[db2 home]/sqllib/function** directory. This file can be found in the **[db\_audit\_api\_home]\DB2** directory.
2. Add **dbauditRunner.jar** file to the Java CLASSPATH environment variable for the DB2 instance owner and bounce your DB2 instance in order for the new CLASSPATH to take effect.
3. After that start DB Audit interface service on your DB2 server running the following command in **[db2 home]/sqllib/function** directory:

```
java -jar dbauditRunner.jar
```

4. **On Unix systems:** add this command to the profile of the DB2 instance owner so that it can start automatically when your DB2 computer starts.

**On Windows systems:** use **dbauditRunnerSrv.exe** program to install the audit process runner service which will start the audit processes automatically when the system starts. This file can be found in the **[db\_audit\_api\_home]\DB2** directory. To install the service copy **dbauditRunnerSrv.exe** to **[db2 home]\SQLLIB\function** directory and from there run once the following command:

```
dbauditRunnerSrv.exe /install
```

5. Schedule periodic run of the audit trail loading procedures. These procedures will periodically flush audit trail records to the disk and then load the flushed data into the system audit trail tables stored in the database. Methods described in the [Flush Audit Data](#) topic can be used to flush and load the audit trail data. They can be scheduled and run using any appropriate scheduling tool of your choice. For example, you can use DB2 Script Center for this purpose. A step-by-step example for using DB2 Script Center is provided in DB Audit User's Guide. See "Scheduling System Audit Record Flushing and Loading Procedures" topic in CHAPTER 3, System Auditing in DB Audit User's Guide.

## Set System Audit State

### *Install System Audit*

This method turns on auditing of database operations. Operations are grouped into functional groups also called operation types. If auditing has been configured already, this method overwrites previous settings.

The following table describes available operation types (groups), which can be audited in DB2.

Operation Type	Comments
ALL	This is a catch-all name that can be used to enable or disable auditing of all operation types listed in this table.
Audit Settings Changes and Access	This option generates records when audit settings are changed or when the audit log is accessed.
Authorization Checking	This option generates records during authorization checking of attempts to access or manipulate DB2 UDB objects or functions.
Object Drop and Create	This option generates records when creating or dropping data objects.
Security Changes	This option generates records when granting or revoking: object or database privileges, or DBADM authority. Records are also generated when the database manager security configuration parameters SYSADM_GROUP, SYSCTRL_GROUP, or SYSMANT_GROUP are modified.
System Administration Activities	This option generates records when operations requiring SYSADM, SYSMANT, or SYSCTRL authority are performed.
User Authentication	This option generates records when authenticating users or retrieving system security information.
Operation Context	<p>This option generates additional records showing the context data of performed database operations, for example, an SQL statement for dynamic SQL, a package identifier for static SQL, or an extended indicator of the type of operation being performed, such as CONNECT. This contextual information might very valuable analyzing audit results. Note that the context records contain the same value in the event correlator column in the audit trail as the primary event record, so that a group of records can be associated with a single database operation.</p> <p>Warning: Use of this option typically creates visible impact on the DB2 performance. It should not be used in heave loaded production systems.</p>

**Using the console interface**

```
java -jar dbaudit.jar /D db2ProfileName [/U user] [/P password] /A /F
mode:installSysAudit;operations:<OPERATIONS>;optionWhen:<OPTION_WHEN>;tablespace:<TABLESPACE>
```

**Parameters:**

db2ProfileName – required. Name of an existing database connection profile configured for a DB2 database.

user – optional. Name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<OPERATIONS> – required. Comma-separated list of types of operations to audit. The names of the supported operation types are listed in the Operations table in the beginning of this topic.

**Important Notes:**

- "All" is an exclusive type that cannot be used along with other. This is a special type that means auditing of all types of operations occurring in the database server.
- Operation type names must be entered exactly as they appear in the table.

<WHEN OPTION> – required. One of the following values: WHENEVER SUCCESSFUL, WHENEVER NOT SUCCESSFUL or ALWAYS

WHENEVER SUCCESSFUL can be used to audit only successful executions of the audited statement.

WHENEVER NOT SUCCESSFUL can be used to audit only unsuccessful executions of the audited statement.

ALWAYS can be used to audit both successful and unsuccessful executions of the audited statement.

<TABLESPACE> – optional. Name of the tablespace where to install system audit trail tables. This is only used if the audit trail tables do not exist yet.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D db2ProfileName /A /F "mode:installSysAudit;operations:Object Drop and Create,Security Changes;optionWhen:ALWAYS;tablespace:AUDIT_TSPACE"
```

This installs and enables auditing for DDL and security changes in the database

**Using the API functions**

```
void auditSetup.installSysAudit(String globalOption, String[] operations, String tableSpace);
```

**Throws:**

java.sql.SQLException – if an error occurs.

Parameters:

globalOption – One of the following values: WHENEVER SUCCESSFUL, WHENEVER NOT SUCCESSFUL or ALWAYS

WHENEVER SUCCESSFUL can be used to audit only successful executions of the audited statement.

WHENEVER NOT SUCCESSFUL can be used to audit only unsuccessful executions of the audited statement.

ALWAYS can be used to audit both successful and unsuccessful executions of the audited statement.

operations – String array of types of operations to audit. The names of the supported operation types are listed in the Operations table in the beginning of this topic.



#### Important Notes:

- "All" is an exclusive type that cannot be used with other. In this context, this is a special item that means auditing of all types of operations occurring in the database server.
- Operation type names must be entered exactly as they appear in the list above.

tableSpace – Name of the tablespace where to install system audit trail tables. This is only used if the audit trail tables do not exist yet. Pass null value to use the default tablespace associated with DB\_AUDIT user.

Example:

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My DB2 Profile");
con.connect("user", "password");
```
2. Create an instance of the AuditSetupSystemDB2 class  

```
AuditSetupSystemDB2 auditSetup =
(AuditSetupSystemDB2)AuditSetupSystemCommon.getInstance(con);
```
3. Install and enable auditing for DDL and security changes in the database  

```
auditSetup.installSysAudit( "ALWAYS",
{ "Object Drop and Create", "Security Changes" },
"AUDIT_TSPACE" );
```

## Uninstall System Audit

Use the methods described in [Uninstall Audit Repository Objects](#) topic to uninstall both the system audit and data-change audit objects and other options and enhancements that you have previously installed using DB Audit API or DB Audit GUI.



**Important Notes:** The "uninstall" method completely removes DB Audit from the database, it removes all previously installed audit settings, stored procedures and tables in the repository database **including data-change audit trail tables** and triggers and also removes db\_audit schema and user. The "uninstall" method does not remove dbAuditRunner.jar file from the operation system. You should delete this file using available operation system commands or advise users how to delete that file manually.

### **Enable System Audit**

The auditing is enabled immediately after the installation. No special actions are required to enable it.

In case if the audit has been disabled using methods described in the [Disable System Audit](#) topic you can enable it again by re-executing system audit installation method but without installation of any additional files. See the [Install System Audit](#) topic for more details.

### **Disable System Audit**

This method turns off system auditing but does not remove audit settings and audit trail tables. Therefore, it effectively stops the auditing but keeps all the data and settings in place. In case if you want to enable the auditing later you can use the "install" method described in [Install System Audit](#) topic and that method will pickup all retained settings.

#### **Using the console interface**

```
java -jar dbaudit.jar /D db2ProfileName [/U user] [/P password] /A /F mode:disableSysAudit
```

Parameters:

db2ProfileName – required. Name of an existing database connection profile configured for a DB2 database.

user – optional. Name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

Example:

```
java -jar dbaudit.jar /D db2ProfileName /A /F "mode:disableSysAudit"
```

#### **Using the API functions**

```
void auditSetup.disableSysAudit();
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

None

Example:

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My DB2 Profile");  
con.connect("user", "password");
```

2. Create an instance of the AuditSetupSystemDB2 class  

```
AuditSetupSystemDB2 auditSetup =
(AuditSetupSystemDB2)AuditSetupSystemCommon.getInstance(con);
```
3. Disable auditing  

```
auditSetup.disableSysAudit( );
```

## Set Audit Operations

This method can be used to update audit settings without reinstalling the audit system.

### Using the console interface

```
java -jar dbaudit.jar /D db2ProfileName [/U user] [/P password] /A /F
mode:updateAuditOperations;operations:<OPERATIONS>;optionWhen:<OPTION_WHEN>
```

Parameters:

db2ProfileName – required. Name of an existing database connection profile configured for a DB2 database.

user – optional. Name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<OPERATIONS> – required. List of database operation types to audit. See [Install System Audit](#) topic for details.

<WHEN OPTION> – required. One of the following values: WHENEVER SUCCESSFUL, WHENEVER NOT SUCCESSFUL or ALWAYS

WHENEVER SUCCESSFUL can be used to audit only successful executions of the audited statement.

WHENEVER NOT SUCCESSFUL can be used to audit only unsuccessful executions of the audited statement.

ALWAYS can be used to audit both successful and unsuccessful executions of the audited statement.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D db2ProfileName /A /F "mode:updateAuditOperations;operations:
Object Drop and Create,Security Changes;optionWen:ALWAYS"
```

### Using the API functions

```
void auditSetup.updateAuditOperations(String globalOption, String[] operations);
```



Throws:

`java.sql.SQLException` – if an error occurs.

Parameters:

`operations` – String array of database operation types to audit. See [Install System Audit](#) topic for details.

Example:

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My DB2 Profile");
con.connect("user", "password");
```
2. Create an instance of the `AuditSetupSystemDB2` class  

```
AuditSetupSystemDB2 auditSetup =
(AuditSetupSystemDB2)AuditSetupSystemCommon.getInstance(con);
```
3. Update audit settings  

```
auditSetup.updateAuditOperations( "ALWAYS",
{ "Object Drop and Create", "Security Changes" } );
```

## Flush Audit Data

This method can be used to flush audit events to the audit trail tables. The flushed data becomes immediately available for reporting and for alert monitoring. This method should be called periodically, preferably every few minutes.



**Important Notes:** The "flush" method parses audit data accumulated in binary audit log files stored in `DB2/security` directory and loads this data in a structured format into audit trail tables. If the method is not called for a long time, the processing of large audit log files may require significant time. The frequency of flushing depends on the audit requirements and database usage. The more events are audited, the more often this method should be executed.

### Using the console interface

```
java -jar dbaudit.jar /D db2ProfileName [/U user] [/P password] /A /F mode:flushAuditData
```

Parameters:

`db2ProfileName` – required. Name of an existing database connection profile configured for a DB2 database.

`user` – optional. Name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used.

`password` – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

Example:

```
java -jar dbaudit.jar /D db2ProfileName /A /F mode:flushAuditData
```

**Using the API functions**

```
void auditSetup.flushAuditData();
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

None

Example:

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My DB2 Profile");  
con.connect("user", "password");
```
2. Create an instance of the AuditSetupSystemDB2 class  

```
AuditSetupSystemDB2 auditSetup =  
(AuditSetupSystemDB2)AuditSetupSystemCommon.getInstance(con);
```
3. Update audit settings  

```
auditSetup.updateAuditOperations( "ALWAYS",  
                                  { "Object Drop and Create", "Security Changes" } );
```

**Informational Methods*****Get System Audit Status***

This method reports current status of the system audit process. It can be used to determine whether the auditing is currently enabled or not.

**Using the console interface**

```
java -jar dbaudit.jar /D db2ProfileName [/U user] [/P password] /A /F mode:isSysAuditRunning
```

Parameters:

db2ProfileName – required. Name of an existing database connection profile configured for a DB2 database.

user – optional. Name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

Return:

Current system audit status is printed to the screen, yes/no

Example:

```
java -jar dbaudit.jar /D db2ProfileName /A /F mode:isSysAuditEnabled
```

### Using the API functions

```
boolean auditSetup.isSysAuditEnabled();
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

None.

Return:

Current system audit status

Example:

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My DB2 Profile");
con.connect("user", "password");
```
2. Create an instance of the AuditSetupSystemDB2 class  

```
AuditSetupSystemDB2 auditSetup =
(AuditSetupSystemDB2)AuditSetupSystemCommon.getInstance(con);
```
3. Determine if sys audit is enabled  

```
System.out.println( auditSetup.isSysAuditEnabled( ) );
```

## Install DB Audit Mail Sending SQL Procedure

### Using the console interface

DB Audit mail sending procedure is used in real-time email alerts that can be sent by both system audit trail and data-change audit trail processes.

### Using the console interface

```
java -jar dbaudit.jar /D db2ProfileName [/U user] [/P password] /A /F
mode:installDBAuditMailSendingSQLProcedure;tablespace:<TABLESPACE>
```

Parameters:

db2ProfileName – required. Name of an existing database connection profile configured for a DB2 database.

user – optional. Name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used.

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<TABLESPACE> – optional. Name of DB2 tablespace where to install email sending procedure and auxiliary objects.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D db2ProfileName /A /F  
mode:installDBAuditMailSendingSQLProcedure;tablespace:TOOLS
```

### Using the API functions

```
void auditSetup.installDBAuditMailSendingSQLProcedure(String tablespaceDB2);
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

tablespaceDB2 – Name of DB2 tablespace where to install email sending procedure and auxiliary objects. Pass null value to use the default tablespace associated with DB\_AUDIT user.

Example:

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("My DB2 Profile");  
con.connect("user", "password");
```
2. Create an instance of the AuditSetupSystemDB2 class  

```
AuditSetupSystemDB2 auditSetup =  
(AuditSetupSystemDB2)AuditSetupSystemCommon.getInstance(con);
```
3. Install mail sending procedure  

```
auditSetup.installDBAuditMailSendingSQLProcedure("TOOLS");
```

## Common System Audit Management Tasks (All Database Systems)

The methods described in the following topics work identically for all supported database systems and feature same parameters and return values. They can be accessed via the common AuditSetupSystemCommon interface class as well as DBMS specific interfaces.

For simplicity, in the following topics we will refer to the common **AuditSetupSystemCommon** interface class only, which is sufficient for all described tasks.

### Truncate System Audit Trail Table

This method erases the audit data stored in system audit trail tables. Some database systems also automatically shrink the truncated tables and their indexed and deallocate the freed disk space, making it available for other objects. The exact behavior is DBMS specific.

**Using the console interface**

```
java -jar dbaudit.jar /D ProfileName [/U user] [/P password] /A /F mode:truncateSystemAuditTrailTable
```

## Parameters:

ProfileName – required. Name of an existing database connection profile.

user – optional. For Oracle and DB2 connections, name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used. For SQL Server and ASE connections, logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

## Example:

```
java -jar dbaudit.jar /D someProfileName /A /F mode:truncateSystemAuditTrailTable
```

**Using the API functions**

```
void auditSetup.truncateSystemAuditTrailTable();
```

## Throws:

java.sql.SQLException – if an error occurs.

## Parameters:

None

## Example:

4. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("Some Profile");  
con.connect("user", "password");
```
5. Create an instance of the AuditSetupSystemCommon class  

```
AuditSetupSystemCommon auditSetup = AuditSetupSystemCommon.getInstance(con);
```
6. Update audit settings  

```
auditSetup.truncateSystemAuditTrailTable( );
```

**Archive System Audit Trail Data to a Table**

This method can be used to copy audit trail data from the system audit trail table to another table located on the same server. In SQL Server and Sybase the target table can be located in the same or different database on the same server.

**Using the console interface**

```
java -jar dbaudit.jar /D ProfileName [/U user] [/P password] /A /F
mode:archiveSystemAuditTrailTable;destDatabase:<DESTINATION_DATABASE>;destSchema:<DES
TINATION_SCHEMA>;destTable:<DESTINATION_TABLE>
```

**Parameters:**

ProfileName – required. Name of an existing database connection profile.

user – optional. For Oracle and DB2 connections, name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used. For SQL Server and ASE connections, logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<DESTINATION\_DATABASE> – required in SQL Server and Sybase, not used in Oracle and DB2. Name of the database in which you want to store the archived data. The database must exist at the time of the call.

<DESTINATION\_SCHEMA> – required. Name of the schema containing the table in which to store the archived data. See description of <DESTINATION\_TABLE> parameter for more details.

<DESTINATION\_TABLE> – required. Name of the table in which to store the archived data.

**Important Notes:**

- If the specified table already exists, the audit trail data is appended to that table. The structure of the destination table must much exactly the structure of the system audit trail table.
- If the specified table does not exist, it is automatically created. In this case the destination schema owner must have permissions to create tables. In Oracle and in DB2 the schema owner must also have permissions to allocate space in the destination tablespace.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D someProfileName /A /F
mode:archiveSystemAuditTrailTable;destSchema:HIST_SCHEMA;destTable:SYS_AUDIT_TRAIL
```

**Using the API functions**

```
void auditSetup.archiveSystemAuditTrailTable(String destDatabase, String destTable, String
destSchema);
```

**Throws:**

java.sql.SQLException – if an error occurs.

**Parameters:**

`destDatabase` – Name of the database in which you want to store the archived data. The database must exist at the time of the call. This parameter is not applicable for Oracle and DB2 systems and must be passed as null value.

`destSchema` – required. Name of the schema containing the table in which to store the archived data. See description of `destTable` parameter for more details.

`destTable` – required. Name of the table in which to store the archived data.

**Important Notes:**

- If the specified table already exists, the audit trail data is appended to that table. The structure of the destination table must match exactly the structure of the system audit trail table.
- If the specified table does not exist, it is automatically created. In this case the destination schema owner must have permissions to create tables. In Oracle and in DB2 the schema owner must also have permissions to allocate space in the destination tablespace.

**Example:**

1. Connect to the database, for example:  
`DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("Some Profile");`  
`con.connect("user", "password");`
2. Create an instance of the `AuditSetupSystemCommon` class  
`AuditSetupSystemCommon auditSetup = AuditSetupSystemCommon.getInstance(con);`
3. Archive audit trail to `hist_schema.sys_audit_trail` table  
`auditSetup.archiveSystemAuditTrailTable( null, "sys_audit_trail", "hist_schema" );`

## Archive System Audit Trail Data to a File

This method can be used to copy audit trail data from the system audit trail table to an operation system file in a tab-separated format. The target file must be writeable and accessible from the system where you are running DB Audit API.

Tip: In case if you need to get archived data in some other format, for example as XML, use the available DB Audit reports. See CHAPTER 6 for more information.

**Using the console interface**

```
java -jar dbaudit.jar /D ProfileName [/U user] [/P password] /A /F
mode:exportSystemAuditTrail;filePath:<FILE_PATH>
```

**Parameters:**

`ProfileName` – required. Name of an existing database connection profile.

`user` – optional. For Oracle and DB2 connections, name of the user to be used with the database

connection. If not specified, user name saved in the profile settings is used. For SQL Server and ASE connections, logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<FILE\_PATH> – required, Name of the target file.



**Important Notes:** If the specified file already exists, the system will attempt to overwrite it.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D someProfileName /A /F
"mode:exportSystemAuditTrail;filePath:C:\audit\archive_as_of_10_10_2006.txt"
```

### Using the API functions

```
void exportSystemAuditTrail(String filePath);
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

filePath – Name of the target file.



**Important Notes:** If the specified file already exists, the system will attempt to overwrite it.

Example:

1. Connect to the database, for example:  
*DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("Some Profile");*  
*con.connect("user", "password");*
2. Create an instance of the AuditSetupSystemCommon class  
*AuditSetupSystemCommon auditSetup = AuditSetupSystemCommon.getInstance(con);*
3. Archive audit trail to hist\_schema.sys\_audit\_trail table  
*auditSetup.exportSystemAuditTrailTable("C:\audit\archive\_as\_of\_10\_10\_2006.txt");*

## Uninstall Audit Repository Objects



**Important Notes:** This method completely removes DB Audit from the database, it removes all previously installed audit settings, stored procedures and tables **including data-change audit trail tables** and triggers and also removes db\_audit schema and user. The "uninstall" method **does not remove native system audit trail tables such as these provided in Oracle and Sybase database systems.**

### Using the console interface

```
java -jar dbaudit.jar /D profileName [/U user] [/P password] /A /F mode:uninstallAudit
```



Parameters:

None

Example uninstallation:

```
java -jar dbaudit.jar /D myProfileName /A /F "mode:uninstallAudit "
```

### Using the API functions

`void auditSetup.uninstallAudit();`

Throws:

`java.sql.SQLException` – if an error occurs.

Parameters:

None

Example:

1. Connect to the database:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("Some Profile");  
con.connect("user", "password");
```
2. Create an instance of the `AuditSetupSystemCommon` class  

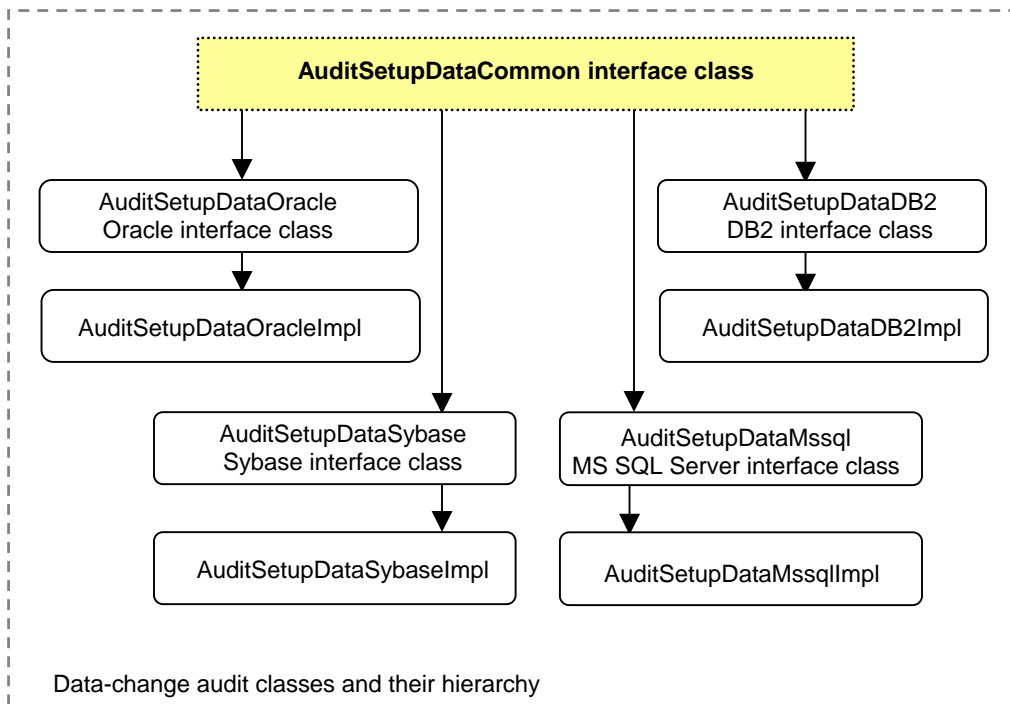
```
AuditSetupSystemCommon auditSetup = AuditSetupSystemCommon.getInstance(con);
```
3. Uninstall DB Audit tables, procedures, schema and user  

```
auditSetup.uninstallAudit();
```

# CHAPTER 5, Data-change Audit Management

## Class Hierarchy

The following diagram demonstrates internal hierarchy of DB Audit classes used for data-change audit management tasks. All described classes are part of **com.softreotech.dbaudit.auditsetup.data** package. As you can see on the diagram, the data-change audit setup is based on a single common interface while specific implementations are provided to perform DBMS specific data-change audit management tasks. Note that the DB Audit command line interface hides all internal class complexity and provides flat keyword based interface.



The following topics describe methods for performing DBMS specific data-change audit management tasks. For reader's convenience, methods are organized in task-oriented groups. Each topic covers method specification, parameters and return values, and also includes code samples demonstrating how to call these methods.

For simplicity, in the following topics we will refer to the common **AuditSetupDataCommon** interface class only, which is sufficient for all described tasks.

## Parameter Names and Values



### Important notes for using DB Audit command line interface :

- The same rules that we have described in CHAPTER 4 also apply to command line parameter names and values used in the data-change auditing. Please see [Parameter Names and Values](#) topic in CHAPTER 4 for more information.

## Data-change Audit Management Tasks

### Install Data-change Audit for a Table

Use this method to install data-change auditing trigger and audit trail table for a business table whose data changes you need to audit and record in the audit trail

#### Using the console interface

```
java -jar dbaudit.jar /D ProfileName [/U user] [/P password] /A /F
mode:setDataAudit;auditDb:<AUDIT_DB>;tableOwner:<TABLE_OWNER>;tableName:<TABLE_NAME>;userMapProc:<USER_MAP_PROCEDURE>;tablespaceDB2:<TABLESPACE_DB2>;usersExcluded:<USERS_EXCLUDED>;appsExcluded:<APPS_EXCLUDED>;auditInserts:<AUDIT_INSERTS>;auditDeletes:<AUDIT_DELETES>;auditUpdates:<AUDIT_UPDATES>;auditNotify:<AUDIT_NOTIFY>;mailKeyColumns:<MAIL_KEY_COLUMNS>;mailRecipient:<MAIL_RECIPIENT>;mailCC:<MAIL_CC>;filterColumns:<FILTER_COLUMNS>;columns:<COLUMNS>;users:<USERS>;apps:<APPLICATIONS>
```

#### Parameters:

ProfileName – required. Name of an existing database connection profile.

user – optional. For Oracle and DB2 connections, name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used. For SQL Server and ASE connections, logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<AUDIT\_DB> – required for SQL Server and ASE, not required for Oracle and DB2. This is the name of the database containing the table whose changes will be audited.

<TABLE\_OWNER> – required. Schema name of the table whose changes will be audited.

<TABLE\_NAME> – required. Name of the table whose changes will be audited.

<USER\_MAP\_PROCEDURE> – optional. Name of a user-defined stored procedure that can be

used to map database user name to an application user name. The procedure is invoked from the auditing trigger when changes are recorded to the audit trail. For more information see "Setting User Name Mapping" topic in CHAPTER 4, Data Change Auditing of DB Audit User's Guide.

<TABLESPACE\_DB> – required for DB2. Name of the tablespace in which to create the audit trail table. Please note that in Oracle this parameter is ignored and the table is always created in the default tablespace for DB\_AUDIT user; in SQL Server and ASE the table is always created in the audit repository database.

<USERS\_EXCLUDED> – required only if <USERS> parameter is specified. This parameter can have the following values: yes, no. Yes value makes the auditing trigger ignore changes made by users whose names are specified in the <USERS> parameter. No value makes the auditing trigger only record changes made by users whose names are specified in the <USERS> parameter. If <USERS> parameter is not specified then <USERS\_EXCLUDED> parameter is ignored.

<APPS\_EXCLUDED> – required only if <APPLICATIONS> parameter is specified. This parameter can have the following values: yes, no. Yes value makes the auditing trigger ignore changes made by applications whose names are specified in the <APPLICATIONS> parameter. No value makes the auditing trigger only record changes made by applications whose names are specified in the <APPLICATIONS> parameter. If <APPLICATIONS> parameter is not specified then <APPLICATIONS\_EXCLUDED> parameter is ignored.

<AUDIT\_INSERTS> – required. This parameter can have the following values: yes, no. Yes value makes the auditing trigger capture records inserted into the audited table, no – ignore inserts.

<AUDIT\_DELETES> – required. This parameter can have the following values: yes, no. Yes value makes the auditing trigger capture records deleted from the audited table, no – ignore deletes.

<AUDIT\_UPDATES> – required. This parameter can have the following values: yes, no. Yes value makes the auditing trigger capture records updated in the audited table, no – ignore updates.

<AUDIT\_NOTIFY> – required. This parameter can have the following values: yes, no. Yes value makes the auditing trigger generate real-time email alerts when changes are made in the audited table. What is considered as a change is driven by values of <AUDIT\_INSERTS>, <AUDIT\_DELETES>, <AUDIT\_UPDATES> parameters. On top of that, additional filters might be created by <USERS>, <APPLICATIONS>, and <FILTER\_COLUMNS> parameters. If this parameter is set to yes, you must also provide a value for <MAIL\_RECIPIENT> parameter.

<MAIL\_KEY\_COLUMNS> – optional. Comma-delimited list of column names whose values you want to include into the email notification message. If not specified, only primary key columns are included.

<MAIL\_RECIPIENT> – required if <AUDIT\_NOTIFY> value is yes. This is the email address of the person or email group who will receive email notifications when changes occur in the audited table.

<MAIL\_CC> – optional. This is the email address of the person or email group who will receive carbon copies of email notifications when changes occur in the audited table.

<FILTER\_COLUMNS> – optional. Comma-delimited list of column names that define column-level filter for auditing of UPDATE operations. The auditing trigger only processes a change when a value of at least of the listed columns gets changed. If this parameter is not specified or empty, column-level filter is not used and every UPDATE is considered as a change.

<COLUMNS> – optional. Comma-delimited list of column names that you want to have in the audit trail table. If not specified all columns from the audited table are recorded in the audit trail table. Some exceptions apply for large binary columns. See DB Audit User's Guide for more information.

<USERS> – optional. In Oracle and in DB2 this is a comma-delimited list of user names for use in user-level audit filters. In SQL Server and ASE this is a comma-delimited list of login names for use in audit filters. Note that <USERS\_EXCLUDED> parameter controls type of user-level audit filters.

<APPLICATIONS> – optional. Comma-delimited list of application names for use in application-level audit filters. Note that <APPLICATIONS\_EXCLUDED> parameter controls type of application-level audit filters.



**Tip:** Use your database native tools to lookup correct application names and they are seen from the database side. For example, in SQL Server you can use the `sp_who2` procedure while the application is running, to find out its database connection and the name. See the name appearing in the ProgramName column. Similarly in Oracle, you can run a SELECT from `v$session` system view and check the value of the Program column. The name entered into application filters must be entered exactly as it appears in your database.

Example 1 (the following must be entered on a single line):

This will install a simple auditing trigger capturing all changes in the audited table

```
java -jar dbaudit.jar /D someProfileName /A /F
"mode:setDataAudit;auditDb:NORTHWIND;tableOwner:DBO;tableName:TEST_TABLE;usersExcluded:no;appsExcluded:no;auditInserts:yes;auditDeletes:yes;auditUpdates:yes;auditNotify:no"
```

Example 2 (the following must be entered on a single line):

This will install a sophisticated auditing trigger capturing all changes in the audited table when value of the PRICE column is affected, except changes made by application "Batch Data Loader." It will also generate email alerts when changes occur and send them to [me@mycompany.com](mailto:me@mycompany.com) recipient.

```
java -jar dbaudit.jar /D someProfileName /A /F
"mode:setDataAudit;auditDb:NORTHWIND;tableOwner:DBO;tableName:TEST_TABLE;usersExcluded:no;appsExcluded:no;auditInserts:yes;auditDeletes:yes;auditUpdates:yes;auditNotify:no;auditNotify:yes;mailKeyColumns:PRODUCT_ID,PRODUCT_NAME,COLOR,PRICE;mailRecipient:ME@MYCOMPANY.COM;filterColumns:PRICE;apps:Batch Data Loader;appsExcluded:yes"
```

### Using the API functions

```
int auditSetup.setDataAudit(DataAuditOption auditOptions, String tablespaceDB2, String auditDb, String userMapProc);
```

**Throws:**

java.sql.SQLException – if an error occurs.

**Return:**

Return 1 if the auditing has been installed successfully and the new trigger and audit trail table have been built; Return 0 if it has found an existing trigger and audit trail table and successfully updated. Raises an exception in all other cases.

**Parameters:**

For easy of coding all data-change audit options are wrapped into a separate class **DataAuditOption**. An instance of this class is used for **auditOptions** parameter in the installation method.

```
DataAuditOption dataAuditOption = new DataAuditOption(String owner, String table,  
boolean usersExcluded, boolean applicationsExcluded,  
boolean auditInserts, boolean auditDeletes, boolean auditUpdates,  
boolean auditNotify, String[] mailKeyColumns, String mailRecipient, String mailCC,  
String[] filterColumns, String[] columns, String[] users, String[] applications);
```

**auditDb** – required for SQL Server and ASE, not required for Oracle and DB2. This is the name of the database containing the table whose changes you want to audit. For Oracle and DB2 specify null value.

**userMapProc** – Name of a user-defined stored procedure that can be used to map database user name to an application user name. The procedure is invoked from the auditing trigger when changes are recorded to the audit trail. For more information see "Setting User Name Mapping" topic in CHAPTER 4, Data Change Auditing of DB Audit User's Guide. If user name mapping is not required specify null value for this parameter

**owner** – Schema name of the table whose changes will be audited.

**table** – Name of the table whose changes will be audited.

**tablespaceDB2** – required for DB2 only, in all other DBMS the value of this parameter is ignored. Name of the tablespace in which to create the audit trail table. Please note that in Oracle this parameter is ignored and the table is always created in the default tablespace for DB\_AUDIT user; in SQL Server and ASE the table is always created in the audit repository database.

**usersExcluded** – TRUE value makes the auditing trigger ignore changes made by users whose names are specified in the "users" parameter. FALSE value makes the auditing trigger only record changes made by users whose names are specified in the "users" parameter. If the "users" parameter is null or empty then usersExcluded parameter is ignored.

**applicationsExcluded** – TRUE value makes the auditing trigger ignore changes made by applications whose names are specified in the "applications" parameter. FALSE value makes the auditing trigger only record changes made by applications whose names are specified in the "applications" parameter. If the "applications" parameter is not specified then applicationsExcluded parameter is ignored.

**auditInserts** – TRUE value makes the auditing trigger capture records inserted into the audited table, FALSE – ignore inserts.

**auditDeletes** – TRUE value makes the auditing trigger capture records deleted from the audited table, FALSE – ignore deletes.

**auditUpdates** – TRUE value makes the auditing trigger capture records updated in the audited table, FALSE – ignore updates.

**auditNotify** – TRUE value makes the auditing trigger generate real-time email alerts when changes are made in the audited table. What is considered as a change is driven by values of **auditInserts**, **auditDeletes**, **auditUpdates** parameters. On top of that, additional filters might be created by users, applications, and **filterColumns** parameters. If this parameter is set to FALSE, you must also provide a value for **mailRecipient** parameter.

**mailKeyColumns** – String array of column names whose values you want to include into the email notification message. If this is null value or empty, then only primary key columns are included.

**mailRecipient** – Email address of the person or email group who will receive email notifications when changes occur in the audited table. If not required, specify null. If **auditNotify** is set to FALSE, the value of **mailRecipient** parameter is ignored.


**mailCC** – The email address of the person or email group who will receive carbon copies of email notifications when changes occur in the audited table. If **auditNotify** is set to FALSE, the value of **mailCC** parameter is ignored.

**filterColumns** – String array of column names that define column-level filter for change event for UPDATE operations. The auditing trigger only processes a change when a value of at least of the listed columns gets changed. If this parameter is not specified or empty, column-level filter is not used and every UPDATE is considered as a change.

**columns** – String array of column names that you want to have in the audit trail table. If not specified all columns from the audited table are recorded in the audit trail table. Some exceptions apply for large binary columns. See DB Audit User's Guide for more information.

**users** – In Oracle and in DB2 this is a string array of user names for use in user-level audit filters. In SQL Server and ASE this is a string array of login names for use in audit filters. Note that **usersExcluded** parameter controls type of user-level audit filters.

**applications** – String array of application names for use in application-level audit filters. Note that **applicationsExcluded** parameter controls type of application-level audit filters.

 **Tip:** Use your database native tools to lookup correct application names and they are seen from the database side. For example, in SQL Server you can use the **sp\_who2** procedure while the application is running, to find out its database connection and the name. See the name appearing in the **ProgramName** column. Similarly in Oracle, you can run a SELECT from **v\$session** system view and check the value of the **Program** column. The name entered into application filters must be entered exactly as it appears in your database.

Example 1 (simple auditing for all changes):

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("Some Profile");  
con.connect("user", "password");
```

2. Create an instance of the AuditSetupDataCommon class  

```
AuditSetupDataCommon auditSetup = AuditSetupDataCommon.getInstance(con);
```
3. Install data-change auditing for table TEST\_TABLE  

```
DataAuditOption options = new DataAuditOption( "DBO", "TEST_TABLE",
false, false, true, true, true, false, null, null, null, null, null );
auditSetup.setDataAudit( options, null, "NORTHWIND", null );
```

Example 2 (sophisticated auditing capturing all changes in the audited table when value of the PRICE column is affected, except changes made by application "Batch Data Loader." It will also generate email alerts when changes occur and send them to [me@mycompany.com](mailto:me@mycompany.com) recipient.):

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("Some Profile");
con.connect("user", "password");
```
2. Create an instance of the AuditSetupDataCommon class  

```
AuditSetupDataCommon auditSetup = AuditSetupDataCommon.getInstance(con);
```
3. Install data-change auditing for table TEST\_TABLE  

```
DataAuditOption options = new DataAuditOption( "DBO", "TEST_TABLE",
false, true, true, true, true, true,
{ "PRODUCT_ID", "PRODUCT_NAME", "COLOR", "PRICE" },
"ME@MYCOMPANY.COM", null, { "PRICE " }, null, null,
{ "Batch Data Loader" } );
auditSetup.setDataAudit( options, null, "REPOSITORY", null );
```

Additionally the following method is provided for greater control over existing auditing objects.

```
int auditSetup.setDataAudit(DataAuditOption auditOptions, String tableSpaceDB2, String auditDb,
String userMapProc, boolean modifyTriggerIfExists);
```

This method provides an additional parameter `modifyTriggerIfExists` which controls what to do if the specified business table already has an auditing trigger built. If TRUE, the existing trigger and the associated audit trail table will be updated, if not they will be rebuilt.



**Important Notes:** In case of existing trigger modifications, DB Audit will analyze the existing audit trail table and compare its structure against the required structure. If the structures are compatible it will reuse the existing table with the new trigger and preserve the existing audit trail data, otherwise it will drop the existing audit trail table and build a new one.

## Uninstall Data-change Audit for a Table

This method can be used to remove the data-change auditing trigger and the associated audit trail table for the specified business table. Warning all audit data for that table will be permanently destroyed. Use the "disable" method described in [Disable Data-change Audit Trigger](#) topic if you simply want to stop the auditing temporarily and retains the audit data.

### Using the console interface

```
java -jar dbaudit.jar /D ProfileName [/U user] [/P password] /A /F
mode:unsetDataAudit;auditDb:<AUDIT_DB>;tableOwner:<TABLE_OWNER>;tableName:<TABLE_NAME>
ME>
```



**Parameters:**

ProfileName – required. Name of an existing database connection profile.

user – optional. For Oracle and DB2 connections, name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used. For SQL Server and ASE connections, logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<AUDIT\_DB> – required for SQL Server and ASE, not required for Oracle and DB2. This is the name of the database containing the table whose changes are being audited.

<TABLE\_OWNER> – required. Schema name of the table whose changes are being audited.

<TABLE\_NAME> – required. Name of the table whose changes are being audited.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D someProfileName /A /F
mode:unsetDataAudit;auditDb:NORTHWIND;tableOwner:DBO;tableName:TEST_TABLE
```

**Using the API functions**

```
void auditSetup.unsetDataAudit(String auditDb, String owner, String table);
```

**Throws:**

java.sql.SQLException – if an error occurs.

**Parameters:**

auditDb – required for SQL Server and ASE, not required for Oracle and DB2. This is the name of the database containing the table whose changes are being audited. For Oracle and DB2 specify null value.

owner – Schema name of the table whose changes are being audited.

table – Name of the table whose changes are being audited.

**Example:**

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("Some Profile");
con.connect("user", "password");
```
2. Create an instance of the AuditSetupDataCommon class  

```
AuditSetupDataCommon auditSetup = AuditSetupDataCommon.getInstance(con);
```

3. Uninstall data-change auditing for table TEST\_TABLE  
`auditSetup.setDataAudit("NORTHWIND", "DBO", "TEST_TABLE");`

## Enable Data-change Audit Trigger

This method can be used to re-enable the auditing previously stopped using the "disable" method described in [Disable Data-change Audit Trigger](#) topic.

### Using the console interface

```
java -jar dbaudit.jar /D ProfileName [/U user] [/P password] /A /F  
mode:enableDataAuditTrigger;auditDb:<AUDIT_DB>;tableOwner:<TABLE_OWNER>;tableName:<TABLE_NAME>
```

#### Parameters:

ProfileName – required. Name of an existing database connection profile.

user – optional. For Oracle and DB2 connections, name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used. For SQL Server and ASE connections, logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<AUDIT\_DB> – required for SQL Server and ASE, not required for Oracle and DB2. This is the name of the database containing the table whose changes are being audited.

<TABLE\_OWNER> – required. Schema name of the table whose changes are being audited.

<TABLE\_NAME> – required. Name of the table whose changes are being audited.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D someProfileName /A /F  
mode:enableDataAuditTrigger;auditDb:NORTHWIND;tableOwner:DBO;tableName:TEST_TABLE
```

### Using the API functions

```
void auditSetup.enableDataAuditTrigger(String auditDb, String owner, String table);
```

#### Throws:

java.sql.SQLException – if an error occurs.

**Parameters:**

auditDb – required for SQL Server and ASE, not required for Oracle and DB2. This is the name of the database containing the table whose changes are being audited. For Oracle and DB2 specify null value.

owner – Schema name of the table whose changes are being audited.

table – Name of the table whose changes are being audited.

**Example:**

1. Connect to the database, for example:  
`DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("Some Profile");  
con.connect("user", "password");`
2. Create an instance of the AuditSetupDataCommon class  
`AuditSetupDataCommon auditSetup = AuditSetupDataCommon.getInstance(con);`
3. Enabled data-change auditing for table TEST\_TABLE  
`auditSetup.enableDataAuditTrigger("NORTHWIND", "DBO", "TEST_TABLE");`

## Disable Data-change Audit Trigger

This method can be used to temporarily stop auditing of the specified business table.

**Using the console interface**

```
java -jar dbaudit.jar /D ProfileName [/U user] [/P password] /A /F  
mode:disableDataAuditTrigger;auditDb:<AUDIT_DB>;tableOwner:<TABLE_OWNER>;tableName:<TABLE_NAME>
```

**Parameters:**

ProfileName – required. Name of an existing database connection profile.

user – optional. For Oracle and DB2 connections, name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used. For SQL Server and ASE connections, logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<AUDIT\_DB> – required for SQL Server and ASE, not required for Oracle and DB2. This is the name of the database containing the table whose changes are being audited.

<TABLE\_OWNER> – required. Schema name of the table whose changes are being audited.

<TABLE\_NAME> – required. Name of the table whose changes are being audited.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D someProfileName /A /F
mode:disableDataAuditTrigger;auditDb:NORTHWIND;tableOwner:DBO;tableName:TEST_TABLE
```

### Using the API functions

```
void auditSetup.disableDataAuditTrigger(String auditDb, String owner, String table);
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

auditDb – required for SQL Server and ASE, not required for Oracle and DB2. This is the name of the database containing the table whose changes are being audited. For Oracle and DB2 specify null value.

owner – Schema name of the table whose changes are being audited.

table – Name of the table whose changes are being audited.

Example:

1. Connect to the database, for example:  
*DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("Some Profile");*  
*con.connect("user", "password");*
2. Create an instance of the AuditSetupDataCommon class  
*AuditSetupDataCommon auditSetup = AuditSetupDataCommon.getInstance(con);*
3. Disable data-change auditing for table TEST\_TABLE  
*auditSetup.disableDataAuditTrigger("NORTHWIND", "DBO", "TEST\_TABLE");*

## Truncate Data-change Audit Trail Table

This method erases the audit data stored in audit trail table associated with the specified business table. Some database systems also automatically shrink the truncated tables and their indexed and deallocate the freed disk space, making it available for other objects. The exact behavior is DBMS specific.

### Using the console interface

```
java -jar dbaudit.jar /D ProfileName [/U user] [/P password] /A /F
mode:truncateDataAuditTrailTable;auditDb:<AUDIT_DB>;tableOwner:<TABLE_OWNER>;tableName:
<TABLE_NAME>
```

**Parameters:**

ProfileName – required. Name of an existing database connection profile.

user – optional. For Oracle and DB2 connections, name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used. For SQL Server and ASE connections, logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<AUDIT\_DB> – required for SQL Server and ASE, not required for Oracle and DB2. This is the name of the database containing the table whose changes are being audited.

<TABLE\_OWNER> – required. Schema name of the table whose changes are being audited.

<TABLE\_NAME> – required. Name of the table whose changes are being audited.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D someProfileName /A /F
mode:truncateDataAuditTrailTable;auditDb:NORTHWIND;tableOwner:DBO;tableName:TEST_TA
BLE
```

**Using the API functions**

```
void auditSetup.truncateDataAuditTrailTable(String auditDb, String owner, String table);
```

**Throws:**

java.sql.SQLException – if an error occurs.

**Parameters:**

auditDb – required for SQL Server and ASE, not required for Oracle and DB2. This is the name of the database containing the table whose changes are being audited. For Oracle and DB2 specify null value.

owner – Schema name of the table whose changes are being audited.

table – Name of the table whose changes are being audited.

**Example:**

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("Some Profile");
con.connect("user", "password");
```
2. Create an instance of the AuditSetupDataCommon class  

```
AuditSetupDataCommon auditSetup = AuditSetupDataCommon.getInstance(con);
```

3. Truncate data-change audit trail for table TEST\_TABLE  
*auditSetup.truncateDataAuditTrailTable( "NORTHWIND", "DBO", "TEST\_TABLE" );*

## Archive Data-change Audit Trail to a Table

This method can be used to copy audit trail data from the audit trail table associated with the specified business table to another table located on the same server.

### Using the console interface

```
java -jar dbaudit.jar /D ProfileName [/U user] [/P password] /A /F
mode:archiveDataAuditTrailTable;auditDb:<AUDIT_DB>;tableOwner:<TABLE_OWNER>;tableName:<
TABLE_NAME>;destSchema:<DESTINATION_SCHEMA>;destTable:<DESTINATION_TABLE>
```

#### Parameters:

ProfileName – required. Name of an existing database connection profile.

user – optional. For Oracle and DB2 connections, name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used. For SQL Server and ASE connections, logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<AUDIT\_DB> – required for SQL Server and ASE, not required for Oracle and DB2. This is the name of the database containing the table whose changes are being audited.

<TABLE\_OWNER> – required. Schema name of the table whose changes are being audited.

<TABLE\_NAME> – required. Name of the table whose changes are being audited.

<DESTINATION\_SCHEMA> – required. Name of the schema containing the table in which to store the archived data. See description of <DESTINATION\_TABLE> parameter for more details.

<DESTINATION\_TABLE> – required. Name of the table in which to store the archived data.



#### Important Notes:

- If the specified table already exists, the audit trail data is appended to that table. The structure of the destination table must much exactly the structure of the data audit trail table.
- If the specified table does not exist, it is automatically created. In this case the destination schema owner must have permissions to create tables. In Oracle and in DB2 the schema owner must also have permissions to allocate space in the destination tablespace.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D someProfileName /A /F
mode:archiveDataAuditTrailTable;auditDb:NORTHWIND;tableOwner:DBO;tableName:TEST_TAB
LE;destSchema:HISTORY;destTable:TEST_TABLE_CHANGES
```

### Using the API functions

```
void auditSetup.archiveDataAuditTrailTable(String auditDb, String owner, String sourceTable, String
destTable, String destSchema);
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

auditDb – required for SQL Server and ASE, not required for Oracle and DB2. This is the name of the database containing the table whose changes are being audited. For Oracle and DB2 specify null value.

owner – Schema name of the table whose changes are being audited.

table – Name of the table whose changes are being audited.

destSchema – required. Name of the schema containing the table in which to store the archived data. See description of destTable parameter for more details.

destTable – required. Name of the table in which to store the archived data.



#### Important Notes:

- If the specified table already exists, the audit trail data is appended to that table. The structure of the destination table must match exactly the structure of the data audit trail table.
- If the specified table does not exist, it is automatically created. In this case the destination schema owner must have permissions to create tables. In Oracle and in DB2 the schema owner must also have permissions to allocate space in the destination tablespace.

Example:

1. Connect to the database, for example:

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("Some Profile");
con.connect("user", "password");
```
2. Create an instance of the AuditSetupDataCommon class

```
AuditSetupDataCommon auditSetup = AuditSetupDataCommon.getInstance(con);
```
3. Archive data audit trail for table TEST\_TABLE into table TEST\_TABLE\_CHANGES in schema HISTORY.

```
auditSetup.archiveDataAuditTrailTable("NORTHWIND", "DBO", "TEST_TABLE",
"TEST_TABLE_CHANGES", "HISTORY");
```

## Archive Data-change Audit Trail to a File

### Using the console interface

```
java -jar dbaudit.jar /D ProfileName [/U user] [/P password] /A /F
mode:exportDataAuditTrail;auditDb:<AUDIT_DB>;tableOwner:<TABLE_OWNER>;tableName:<TABLE_NAME>;filePath:<FILE_PATH>
```

#### Parameters:

ProfileName – required. Name of an existing database connection profile.

user – optional. For Oracle and DB2 connections, name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used. For SQL Server and ASE connections, logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<AUDIT\_DB> – required for SQL Server and ASE, not required for Oracle and DB2. This is the name of the database containing the table whose changes are being audited.

<TABLE\_OWNER> – required. Schema name of the table whose changes are being audited.

<TABLE\_NAME> – required. Name of the table whose changes are being audited.

<FILE\_PATH> – required, Name of the target file.



**Important Notes:** If the specified file already exists, the system will attempt to overwrite it.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D someProfileName /A /F
"mode:exportDataAuditTrail;auditDb:NORTHWIND;tableOwner:DBO;tableName:TEST_TABLE;filePath:C:\audit\archive_as_of_10_10_2006.txt"
```

### Using the API functions

```
void auditSetup.exportDataAuditTrail(String auditDb,String sourceOwner,String sourceTable,String filePath);
```

#### Throws:

java.sql.SQLException – if an error occurs.

#### Parameters:


auditDb – required for SQL Server and ASE, not required for Oracle and DB2. This is the name of the database containing the table whose changes are being audited. For Oracle and DB2 specify null value.



owner – Schema name of the table whose changes are being audited.

table – Name of the table whose changes are being audited.

filePath – Name of the target file.

-  **Important Notes:** If the specified file already exists, the system will attempt to overwrite it.

Example:

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("Some Profile");
con.connect("user", "password");
```
2. Create an instance of the AuditSetupDataCommon class  

```
AuditSetupDataCommon auditSetup = AuditSetupDataCommon.getInstance(con);
```
3. Archive data-change audit trail for table TEST\_TABLE  

```
auditSetup.archiveDataAuditTrailTable("NORTHWIND", "DBO", "TEST_TABLE",
"C:\audit\larchive_as_of_10_10_2006.txt");
```

## Configure Settings for Data-change Audit Reports

DB Audit allows setting table and column aliases that you can use in data-change audit reports instead of the real non-descriptive table and column names commonly used in business tables. Using this feature, you can provide your users with flexible user-friendly reports that do not require them to know the physical database design.

The following methods can be used to set table and column aliases for use in data-change audit reports.

### Get Table Aliases

This method returns all table aliases previously set for all business tables in the database.

#### Using the console interface

```
java -jar dbaudit.jar /D ProfileName [/U user] [/P password] /A /F mode:getTableAliases
```

Return:

Prints to the screen all available table aliases.

Parameters:

ProfileName – required. Name of an existing database connection profile.

user – optional. For Oracle and DB2 connections, name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used. For SQL Server and ASE connections, logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

Example:

```
java -jar dbaudit.jar /D someProfileName /A /F mode:getTableAliases
```

### Using the API functions

```
String auditSetup.getTableAliases();
```

Throws:

java.sql.SQLException – if an error occurs.

Return:

XML table containing records with the following elements:

```
Database
Owner
Table_Name
Table_Alias
```

Parameters:

None

Example:

1. Connect to the database, for example:  
*DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("Some Profile");  
con.connect("user", "password");*
2. Create an instance of the AuditSetupDataCommon class  
*AuditSetupDataCommon auditSetup = AuditSetupDataCommon.getInstance(con);*
3. Fetch table aliases and print result to the standard output  
*System.out.println( auditSetup.getTableAliases( ) );*

### Set Table Alias

This method sets new alias for the specified business table.

#### Using the console interface

```
java -jar dbaudit.jar /D ProfileName [/U user] [/P password] /A /F
mode:setTableAlias;auditDb:<AUDIT_DB>;tableOwner:<TABLE_OWNER>;tableName:<TABLE_NAME>;tableAlias:<TABLE_ALIAS>
```

Parameters:

ProfileName – required. Name of an existing database connection profile.

user – optional. For Oracle and DB2 connections, name of the user to be used with the database

connection. If not specified, user name saved in the profile settings is used. For SQL Server and ASE connections, logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<AUDIT\_DB> – required for SQL Server and ASE, not required for Oracle and DB2. This is the name of the database containing the table whose changes are being audited.

<TABLE\_OWNER> – required. Schema name of the table whose changes are being audited.

<TABLE\_NAME> – required. Name of the table whose changes are being audited.

<TABLE\_ALIAS> – required. New alias value.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D someProfileName /A /F
"mode:setTableAlias;auditDb:NORTHWIND;tableOwner:DBO;tableName:TABLE1538;tableAlias:
Production Line Items"
```

### Using the API functions

```
void auditSetup.setTableAlias(String auditDb, String owner, String tableName, String tableAlias);
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

auditDb – required for SQL Server and ASE, not required for Oracle and DB2. This is the name of the database containing the table whose changes are being audited. For Oracle and DB2 specify null value.

owner – Schema name of the table whose changes are being audited.

tableName – Name of the table whose changes are being audited.

tableAlias – New alias value.

Example:

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("Some Profile");
con.connect("user", "password");
```
2. Create an instance of the AuditSetupDataCommon class  

```
AuditSetupDataCommon auditSetup = AuditSetupDataCommon.getInstance(con);
```

- Set table alias for table TABLE1538 as "Production Line Items"  
*auditSetup.setTableAlias("NORTHWIND", "DBO", "TABLE1538", "Production Line Items" );*

## Get Column Aliases

This method returns all column aliases previously set for the specified business table.

### Using the console interface

```
java -jar dbaudit.jar /D ProfileName [/U user] [/P password] /A /F
mode:getColumnAliases;auditDb:<AUDIT_DB>;tableOwner:<TABLE_OWNER>;tableName:<TABLE_NAME>
```

Return:

Prints to the screen all available table aliases.

Parameters:

ProfileName – required. Name of an existing database connection profile.

user – optional. For Oracle and DB2 connections, name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used. For SQL Server and ASE connections, logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<AUDIT\_DB> – required for SQL Server and ASE, not required for Oracle and DB2. This is the name of the database containing the table whose changes are being audited.

<TABLE\_OWNER> – required. Schema name of the table whose changes are being audited.

<TABLE\_NAME> – required. Name of the table whose changes are being audited.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D someProfileName /A /F
mode:getColumnAliases;auditDb:NORTHWIND;tableOwner:DBO;tableName:TEST_TABLE
```

### Using the API functions

String auditSetup.getColumnAliases(String auditDb, String owner, String table);

Throws:

java.sql.SQLException – if an error occurs.

Return:

XML table containing records with the following elements:

Column\_Name  
Table\_Alias

Parameters:

auditDb – required for SQL Server and ASE, not required for Oracle and DB2. This is the name of the database containing the table whose changes are being audited. For Oracle and DB2 specify null value.

owner – Schema name of the table whose changes are being audited.

table – Name of the table whose changes are being audited.

Example:

1. Connect to the database, for example:  
`DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("Some Profile");  
con.connect("user", "password");`
2. Create an instance of the AuditSetupDataCommon class  
`AuditSetupDataCommon auditSetup = AuditSetupDataCommon.getInstance(con);`
3. Fetch column aliases for table TEST\_TABLE and print result to the standard output  
`System.out.println( auditSetup.getTableAliases( "NORTHWIND", "DBO", "TEST_TABLE" ) );`

## Set Column Alias

This method sets new alias for the specified column in the specified business table.

### Using the console interface

```
java -jar dbaudit.jar /D ProfileName [/U user] [/P password] /A /F
mode:setColumnAlias;auditDb:<AUDIT_DB>;tableOwner:<TABLE_OWNER>;tableName:<TABLE_NAME>;columnName:<COLUMN_NAME>;columnAlias:<COLUMN_ALIAS>
```

Parameters:

ProfileName – required. Name of an existing database connection profile.

user – optional. For Oracle and DB2 connections, name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used. For SQL Server and ASE connections, logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

<AUDIT\_DB> – required for SQL Server and ASE, not required for Oracle and DB2. This is the name of the database containing the table whose changes are being audited.

<TABLE\_OWNER> – required. Schema name of the table whose changes are being audited.

<TABLE\_NAME> – required. Name of the table whose changes are being audited.

<COLUMN\_NAME> – required. Name of the column whose alias to set or update.

<COLUMN\_ALIAS> – required. New alias value.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D someProfileName /A /F  
"mode:setColumnAlias;auditDb:NORTHWIND;tableOwner:DBO;tableName:TABLE1538;columnN  
ame:COLLINENO;columnAlias:Line Item No"
```

### Using the API functions

```
void auditSetup.setColumnAlias(String dbName, String owner, String tableName, String columnName,  
String columnAlias);
```

Throws:

java.sql.SQLException – if an error occurs.

Parameters:

auditDb – required for SQL Server and ASE, not required for Oracle and DB2. This is the name of the database containing the table whose changes are being audited. For Oracle and DB2 specify null value.

owner – Schema name of the table whose changes are being audited.

tableName – Name of the table whose changes are being audited.

tableAlias – New alias value.

columnName – Name of the column whose alias to set or update.

columnAlias – New alias value.

Example:

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("Some Profile");  
con.connect("user", "password");
```
2. Create an instance of the AuditSetupDataCommon class  

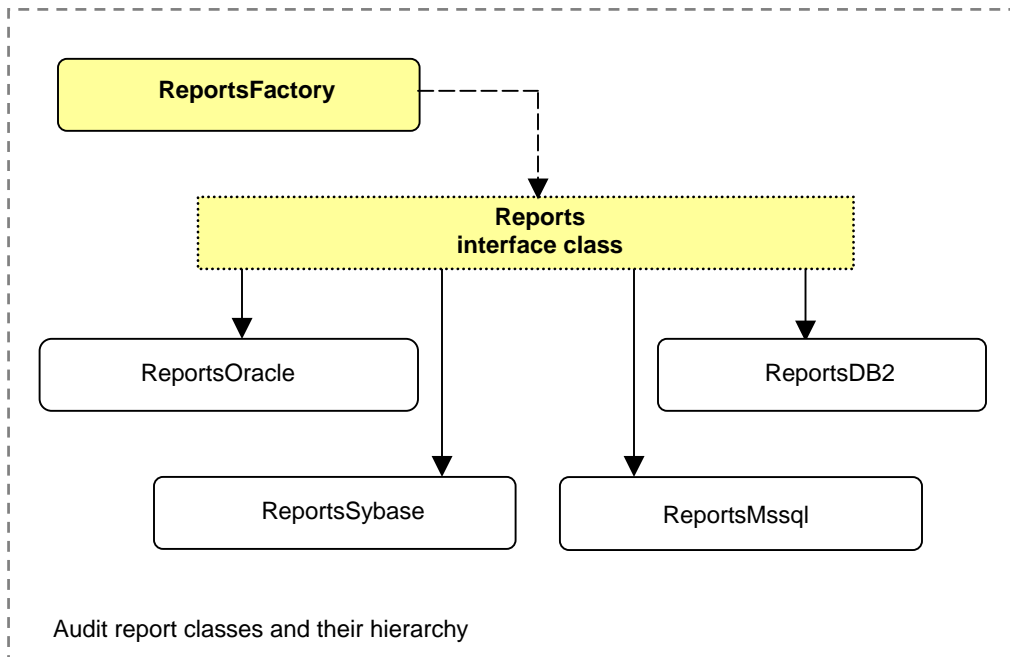
```
AuditSetupDataCommon auditSetup = AuditSetupDataCommon.getInstance(con);
```

3. Set column alias for column COLLINENO in table TABLE1538 as "Line Items No"  
*auditSetup.setColumnAlias("NORTHWIND", "DBO", "TABLE1538",  
"COLLINENO", "Line Items No");*

# CHAPTER 6, Generating Audit Reports

## Class Hierarchy

The following diagram demonstrates internal hierarchy of DB Audit classes used for audit reporting. All described classes are part of **com.softretech.dbaudit.reports** package. As you can see on the diagram, all reports are based on a single common interface while specific implementations are provided for each DBMS. Note that the DB Audit command line interface hides all internal class complexity and provides flat keyword based interface.



The following topics describe methods for generating audit reports. For simplicity, in the following topics we will refer to the common **Reports** interface class only, which is sufficient for all described tasks. The **ReportsFactory** class can be used to instantiate the correct implementation of the Reports class compatible with the type of the connected DBMS.

```
Reports reports = ReportsFactory.getReportsInstance(DbConnect connection, PrintWriter out, String reportXSL);
```

Here, the 'connection' is an instance of DbConnect, class providing database connectivity, 'out' - is an instance of the PrintWriter class where the report is to be written, and 'reportXSL' - a report XSL file path. Note that all reports are generated in XML format. If 'reportXSL' parameter is null, the default XLS formatting is applied to the XML output file. If the 'out' class is null, the output is written to reports.xml file generated in the current directory.

Below is a practical example demonstrating how to connect to the database and then run a report.

1. Connect to the database, for example:  

```
DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("Some Profile");
con.connect("user", "password");
```
2. Create an instance of the Reports class compatible with the type of DBMS we are connected to. Use default reports.xml file for the output with default XLS formatting:  

```
Reports reports = ReportsFactory.getReportsInstance(con, (PrintWriter)null, (String)null);
```



3. Run report and write results to the file:  
*reports.doUsersWithExpiredPasswordsReport( );*

## Parameter Names and Values



### Important notes for using DB Audit command line interface :

- The same rules that we have described in CHAPTER 4 also apply to command line parameter names and values used in the data-change auditing. Please see [Parameter Names and Values](#) topic in CHAPTER 4 for more information.

## Generating Reports

### Using the console interface

```
java -jar dbaudit.jar /D profileName [/U user] [/P password] /R <reportName> [/S start_date] [/E end_date] [/N user_name] [/M terminal_name] [/I session_id] [/Y event_type] [/W owner_name] [/B object_name] [/C source_db] [/G application_name] [/L xsl_file] [/O output_file]
```

### Parameters:

Except profileName and reportName, all other parameters are optional. The parameters can be used for report filtering.

profileName – required. Name of an existing database connection profile.

user – optional. For Oracle and DB2 connections, name of the user to be used with the database connection. If not specified, user name saved in the profile settings is used. For SQL Server and ASE connections, logon name to be used with the database connection. If not specified, logon name saved in the profile settings is used

password – optional. Password to be used with the database connection. If specified, user parameter must be also specified and visa versa.

All other parameters are described in [Common Report Filters and Parameters](#) topic.

Example (the following must be entered on a single line):

```
java -jar dbaudit.jar /D someProfile /R InactiveUsersWithActiveAccounts /O inactive_users.xml
```

This will execute 'Inactive Users with Active Accounts' report and write the output to inactive\_users.xml file.

**Using the API functions**

```
void doApplicationLevelAuditFiltersReport();
```

```
void doAuditTrailByApplicationReport(java.lang.String application, java.lang.String auditUser,  
java.util.Date startDate, java.util.Date endDate);
```

```
void doAuditTrailBySchemaReport(java.lang.String schema, java.lang.String auditUser, java.util.Date  
startDate, java.util.Date endDate);
```

```
void doAuditTrailByTableReport(java.lang.String sourceDb, java.lang.String owner, java.lang.String  
table, java.lang.String auditUser, java.util.Date startDate, java.util.Date endDate);
```

```
void doAuditTrailSummaryAllReport(java.lang.String auditUser, java.util.Date startDate, java.util.Date  
endDate);
```

```
void doAuditTrailSummaryByTableReport(java.lang.String sourceDb, java.lang.String owner,  
java.lang.String table, java.lang.String auditUser, java.util.Date startDate, java.util.Date endDate);
```

```
void doDatabaseErrorsReport(java.lang.String userName, java.lang.String terminal, java.lang.String  
dbName, java.lang.String ownerName, java.lang.String objectName, java.util.Date startDate,  
java.util.Date endDate);
```

```
void doDatabaseOperationsReport(java.lang.String userName, java.lang.String sessionId,  
java.lang.String terminalList, java.util.Date startDate, java.util.Date endDate);
```

```
void doDefaultSystemAuditOptionsReport();
```

```
void doEnabledDataChangeAuditsReport();
```

```
void doEnabledGlobalAuditOptionsReport();
```

```
void doEnabledLogonAuditOptionsReport();
```

```
void doEnabledSchemaObjectAuditOptionsReport();
```

```
void doEnabledSQLStatementAndOperationsAuditOptionsReport();
```

```
void doEnabledSystemPrivilegeAuditOptionsReport();
```

```
void doInactiveUsersWithActiveAccountsReport(java.util.Date startDate, java.util.Date endDate);
```

```
void doLogonLogoffAndResourceUsageAuditReport(java.lang.String userName, java.lang.String terminal, java.util.Date startDate, java.util.Date endDate);
```

```
void doObjectAccessAndStatementAuditReport(java.lang.String userName, java.lang.String terminal, java.lang.String dbName, java.lang.String ownerName, java.lang.String objectName, java.util.Date startDate, java.util.Date endDate);
```

```
void doObjectAccessAuditSummaryReport(java.lang.String userName, java.lang.String dbName, java.lang.String ownerName, java.lang.String objectName, java.util.Date startDate, java.util.Date endDate);
```

```
void doRecentAdministratorLoginsReport(java.lang.String terminal, java.util.Date startDate, java.util.Date endDate);
```

```
void doRecentlyCreatedDeletedAndModifiedUsersAndLoginsReport(java.lang.String terminal, java.util.Date startDate, java.util.Date endDate);
```

```
void doRecentlyGrantedAndRevokedPrivilegesReport(java.lang.String terminal, java.util.Date startDate, java.util.Date endDate);
```

```
void doRecentPrivilegedOperationsCreateDropAlterReport(java.lang.String terminal, java.util.Date startDate, java.util.Date endDate);
```

```
void doSessionsReport(java.lang.String userName, java.lang.String eventType, java.lang.String terminalList, java.util.Date startDate, java.util.Date endDate);
```

```
void doStatementAuditDetailReport(java.lang.String userName, java.lang.String terminal, java.lang.String dbName, java.lang.String ownerName, java.lang.String objectName, java.util.Date startDate, java.util.Date endDate);
```

```
void doStatementAuditSummaryReport(java.lang.String userName, java.lang.String dbName, java.lang.String ownerName, java.lang.String objectName, java.util.Date startDate, java.util.Date endDate);
```

```
void doTextOfSQLQueriesReport(java.lang.String userName, java.lang.String terminal, java.util.Date startDate, java.util.Date endDate);
```

```
void doUserActivityDeniedAccessToObjectsReport(java.lang.String terminal);
```

```
void doUserActivityFailedLogonsReport(java.lang.String terminal);
```

```
void doUserActivityLastLogonTimeReport();
```

```
void doUserActivitySysAdminsReport(java.lang.String param, java.util.Date startDate, java.util.Date endDate);
```

```
void doUserLevelAuditFiltersReport();
```

```
void doUsersHavingAdministrativePrivilegesReport();
```

```
void doUsersWithExpiredPasswordsReport();
```

```
void doUsersWithNonExpiringPasswordsReport();
```

```
void getRawAuditTrailData(java.lang.String terminalList, java.util.Date startDate, java.util.Date endDate);
```

All report methods throw:

`java.sql.SQLException` – if an error occurs.

Parameters:

See the following topic for description of reports parameters

## Common Report Filters and Parameters

1. `start_date`, `end_date`, `user_name`, `owner_name`, `object_name`, `terminal_name` – report filtering parameters, all optional.

Dates must be entered in the following format: `mm/dd/yyyy`

Note that `terminal_name` parameter can be either a single value or can be a comma-delimited list containing multiple values. For terminal name value equal SQL NULL value use `'?'` symbol.

Examples:

Let's say we only want reports for these terminals that appear in the audit trail table as : `"MY_BIG_SERVER", "192.168.12.34", "", "[LOCAL]", NULL`. For the `terminal_mname` parameter value we should then specify the following

```
/M MY_BIG_SERVER,192.168.12.34,,[LOCAL],?
```

or

```
/M MY_BIG_SERVER,192.168.12.34,[LOCAL],?
```

or

```
/M ,MY_BIG_SERVER,192.168.12.34,[LOCAL],?
```

In case if we only need audit records with terminal names whose value are empty string returned:

```
/M ""
```

2. `xsl_file` – xsl stylesheet file name. If not specified, the default transformation will be applied

3. `output_file` – report output file name. If not specified – report will be saved to the default "report.xml" file. Specifying keyword "console" will result in printing the report to the console.

The following 2 topics describe report aliases used with command line options names and how they are mapped to actual reports:

## System Audit Reports

Report Alias Used in Command Line Parameters and API Methods Names	DB Audit Report Name
DefaultSystemAuditOptions	Default System Audit Options Report
EnabledGlobalAuditOptions	Enabled Global Audit Options Report
EnabledSQLstatementAndOperationsAuditOptions	Enabled SQL statement and Operations Audit Options Report
EnabledSchemaObjectAuditOptions	Enabled Schema Object Audit Options Report
EnabledSystemPrivilegeAuditOptions	Enabled System Privilege Audit Options Report
EnabledLogonAuditOptions	Enabled Logon Audit Options Report
LogonLogoffAndResourceUsageAudit	Logon/Logoff and Resource Usage Audit Report
ObjectAccessAndStatementAudit	Object Access and Statement Audit Report
ObjectAccessAuditSummary	Object Access Audit Summary Report
StatementAuditDetail	Statement Audit Detail Report
StatementAuditSummary	Statement Audit Summary Report
UserActivityFailedLogons	User Activity (Failed Logons) Report
UserActivityLastLogonTime	User Activity (Last Logon Time) Report
UserActivityDeniedAccessToObjects	User Activity (Denied Access to Objects) Report
UserActivitySysAdmins	User Activity (Sys Admins) Report
DatabaseErrors	Database Errors Report
TextOfSQLQueries	Text of SQL Queries Report
RecentlyCreatedDeletedAndModifiedUsersAndLogins	Recently Created, Deleted and Modified Users and Logins Report

RecentlyGrantedAndRevokedPrivileges	Recently Granted and Revoked Privileges Report
InactiveUsersWithActiveAccounts	Inactive Users with Active Accounts Report
UsersWithExpiredPasswords	Users with Expired Passwords Report
UsersWithNonExpiringPasswordsReport	Users with Non-Expired Passwords Report
UsersHavingAdministrativePrivileges	Users Having Administrative Privileges Report
RecentAdministratorLogins	Recent Administrator Logins Report
RecentPrivilegedOperationsCreateDropAlter	Recent Privileged Operations (Create, Drop, Alter) Report
getRawAuditTrailData	Get Raw Audit Trail Data
DatabaseOperations	Database Operations Report
Sessions	Sessions Report

See DB Audit User's Guide for detailed description of each of these reports including supported filters and output columns.

## Data-change Audit Reports

Report Alias Used in Command Line Parameters and API Methods Names	DB Audit Report Name
EnabledDataChangeAudits	Enabled Data Change Audits Report
UserLevelAuditFilters	User Level Audit Filters Report
ApplicationLevelauditFilters	Application Level Audit Filters Report
AuditTrailByTable	Audit Trail By Table Report
AuditTrailBySchema	Audit Trail By Schema Report
AuditTrailByApplication	Audit Trail By Application Report
AuditTrailSummaryByTable	Audit Trail Summary By Table
AuditTrailSummaryAll	Audit Trail Summary For All Tables

See DB Audit User's Guide for detailed description of each of these reports including supported filters and output columns.



# CHAPTER 7, API Invocation Methods

## Direct Invocation from Java Programs

DB Audit API can be directly invoked from Java programs. Java programs simply need to add **dbaudit.jar** file and optionally **alercenter.jar** file to their CLASSPATH. The invocation methods for DB Audit classes are no different then invocation methods for standard Java classes.

You can find lots of examples available throughout this guide that demonstrate how to load different classes and invoke their methods. in addition, below is a complete example of small Java program that makes a connection to the database and runs UsersWithExpiredPasswordsReport reports.

```
import com.softtreetech.dbaudit.*;
import com.softtreetech.dbaudit.reports.*;
import java.io.PrintWriter;

public class docs {

    public void main() {
        try {
            // Connect to the database:
            DbConnect con = ProfileManagerImpl.getInstance().createDbConnect("Some Profile");
            con.connect("user", "password");

            // Create an instance of the Reports class compatible with the type of DBMS we are
            // connected to. Use default reports.xml file for the output with default XLS formatting:
            Reports reports = ReportsFactory.getReportsInstance(con, (PrintWriter)null, (String)null);

            // Run report and write results to the file:
            reports.doUsersWithExpiredPasswordsReport( );
        }
        catch (Exception ex) {
            // Print error message
            System.out.println(ex.getMessage());
        }
    }
}
```

## Direct Invocation from Non-Java Programs

We are working on a COM wrapper that will allow .NET and other programs invoke DB Audit API methods directly. Estimated release time for the COM wrapper is 2Q 2007.

For now please use the available command line interface. Using this method you can run the available API commands just like you run any other program.



## Indirect Invocation Using Batch Files

To invoke DB Audit API methods from command line and from batch and shell files, use methods described in CHAPTER 4, 5 and 6. Below you will find several ready to use examples:

Example 1 (the following must be entered on a single line):

This will execute 'Inactive Users with Active Accounts' report and write the output to inactive\_users.xml file

```
java -jar dbaudit.jar /D someProfile /R InactiveUsersWithActiveAccounts /O
inactive_users.xml
```

Example 2 (the following must be entered on a single line):

This will install and enable system auditing for schema DDL and security changes in a DB2 database

```
java -jar dbaudit.jar /D db2ProfileName /A /F "mode:installSysAudit;operations:Object
Drop and Create,Security Changes;optionWhen:ALWAYS;tablespace:AUDIT_TSPACE"
```

Example 3 (the following must be entered on a single line):

This will install a simple auditing trigger capturing all changes in the audited table

```
java -jar dbaudit.jar /D someProfileName /A /F
"mode:setDataAudit;auditDb:NORTHWIND;tableOwner:DBO;tableName:TEST_TABLE;users
Excluded:no;appsExcluded:no;auditInserts:yes;auditDeletes:yes;auditUpdates:yes;auditNotif
y:no"
```

Example 4 (the following must be entered on a single line):

This will install a sophisticated auditing trigger capturing all changes in the audited table when value of the PRICE column is affected, except changes made by application "Batch Data Loader." It will also generate email alerts when changes occur and send them to [me@mycompany.com](mailto:me@mycompany.com) recipient.

```
java -jar dbaudit.jar /D someProfileName /A /F
"mode:setDataAudit;auditDb:NORTHWIND;tableOwner:DBO;tableName:TEST_TABLE;users
Excluded:no;appsExcluded:no;auditInserts:yes;auditDeletes:yes;auditUpdates:yes;auditNotif
y:no;auditNotify:yes;mailKeyColumns:PRODUCT_ID,PRODUCT_NAME,COLOR,PRICE;mail
Recipient:ME@MYCOMPANY.COM;filterColumns:PRICE;apps:Batch Data
Loader;appsExcluded:yes"
```

## Remote Invocation Using RMI functionality

The RMI server part of the application is included in the same dbaudit.jar file. To start the server, simply run the start\_RMIServer.bat file provided.

RMI client application comes in the form of the DBAuditRMIClient.jar file. To start the RMI client application run the following command:

```
java -jar DBAuditRMIClient.jar <server_address><console_commands>
```

Parameters:

<server\_address> – server name or IP address.

<console\_commands> – set of switches, options, commands and parameters as they are used with the command console interface.

Example:

```
java -jar DBAuditRMIClient.jar 207.46.130.108 /D OracleProfile /U dbadmin /P dbadminpass /T
```

or

```
java -jar DBAuditRMIClient.jar MY_BIG_SERVER /D OracleProfile /U dbadmin /P dbadminpass /T
```

# APPENDIX A, Hardware and Software Requirements

## Minimum Requirements

### Client (DB Audit Management Console)

- 1 Intel or AMD-based workstation or server running one of the following operating system:
  - Windows Vista
  - Windows 2003
  - Windows XP
  - Windows 2000
  - Windows NT 4.0
  - Windows 98
  - Windows Me
- 2 At least 128 MB RAM
- 3 17 MB disk space for full installation
- 4 VGA monitor
- 5 Required database client software (consult your database system documentation for details)
- 6 If ODBC database interface is used, ODBC and ODBC database connectivity driver

### Alert Center (server)

- 1 Workstation or server running one of the following operating system:
  - Windows 2003/Windows XP/Windows 2000/Windows NT 4.0
  - Linux – Debian and compatible distributions, such as RedHat Linux, SuSe and other
  - Sun Solaris
  - HP –UX
  - Digital Unix
  - IBM AIX
  - Free BSD
  - Mac OS X
  - z/OS
  - OS/390
- 2 At least 256 MB RAM
- 3 14 MB disk space for full installation
- 4 VGA or other monitor
- 5 JRE or JDK 1.4 or better
- 6 Required database client software (consult your database system documentation for details)
- 7 If ODBC database interface is used, ODBC and ODBC database connectivity driver

### Database Server

Any of the supported database servers:

- Oracle 7.3, 8.0, 8i, 9i, 10g

- Microsoft SQL Server 6.5, 7, 2000, 2005
- Sybase SQL Server and Sybase Adaptive Server Enterprise 10.x, 11.x, 12.x
- Sybase Adaptive Server Anywhere 6, 7, 8, 9
- IBM DB2 5.x, 6.x, 7.x, 8.x for Linux, Unix, and Windows
- IBM DB2 6.x, 7.x, 8.x for z/OS and OS/390
- IBM DB2 5.x for OS/400, MVS

## APPENDIX B, Licensing

The terms and conditions of licensing depend on the distribution method of your application, extent of the usage and the type of your application.

Contact SoftTree Technologies [sales@softtreetech.com](mailto:sales@softtreetech.com) to negotiate license terms and obtain a copy of the license agreement.

SoftTree Technologies, Inc.  
Staten Island NY, 10306  
USA

Copyright 2006-2007 (C) SoftTree Technologies, Inc. All Rights Reserved