

Grails Audit Logging Plugin

Robert Oswald

Version 7.0.0-SNAPSHOT, 2026-08-08

Table of Contents

1. Description	2
1.1. Grails 8 Runtime and CLI Architecture	2
1.2. Compatibility	2
1.3. GORM Compatibility	2
2. Installation	3
2.1. Prerequisites	3
2.2. Repository and Runtime Dependency	3
2.3. Automatic CLI Companion	3
2.4. Create the Audit Domain Artifact	3
2.5. Next Steps	4
3. Upgrading to 7.0.x	5
3.1. 6.x to 7.x Checklist	5
3.2. Breaking Changes	5
3.3. Unchanged Behavior	5
4. Upgrading to 5.0.x	6
5. Upgrading to 4.0.x	7
6. Upgrading to 3.0.x	8
6.1. Domain Classes	8
6.2. Configuration Changes	8
6.3. Transactional Behavior	9
6.4. Handlers Removed	9
6.5. Actor Closure	10
6.6. Stampable Trait	10
7. Configuration	11
7.1. Verbose mode	13
7.2. Logging of Associated Ids	13
7.3. Property Value Masking	13
7.4. Verbose Log Truncation Length	14
7.5. Disable All Auditing	14
7.6. Log Full Class Name	14
7.7. Ignoring Specific Events	15
7.8. Runtime Overrides	15
7.9. Example Configuration	15
8. Usage	16
8.1. Basic	16
8.2. Context Overrides	16
8.3. Domain Overrides	17
8.4. Customized Auditable Trait	18

8.5. Request Resolvers	18
9. Stamping	21
9.1. Stamping Configuration	21
10. Implementation	23
11. Auditing and Transactions	24
11.1. AuditLogEventListener	24
11.2. StampEventListener	24
11.3. Plugin Descriptor	24
11.4. Auditable trait	24
11.5. Stampable trait	24
12. Change Log	25

Other formats: [PDF](#), [EPUB](#)

Chapter 1. Description

The Audit Logging plugin adds customizable audit tracking when domain classes implement the `Auditable` trait. It also supports create and update user stamping with the `Stampable` trait.

1.1. Grails 8 Runtime and CLI Architecture

Plugin 7.x supports Grails 8 and Java 21 only. Applications declare the runtime `audit-logging` artifact. Its `Grails-Cli-Artifact` manifest entry identifies `org.grails.plugins:audit-logging-cli`, which Grails 8 provisions only for CLI execution. This keeps CLI dependencies out of application startup and packaged artifacts while making `audit-quickstart` available from the direct Grails CLI and interactive shell.

Plugin 7.x does not support Grails 7. Use plugin 6.x for Grails 7 applications. Existing historical plugin lines remain available for earlier Grails releases.

1.2. Compatibility

Grails Version	Audit Logging Plugin Version
8.0.x	7.0.x, Java 21 required
7.0.x	6.0.x
4.0.x	5.0.x recommended; 4.0.x maintenance also remains available through Grails 4.0.9
3.3.x	3.0.x
3.0.0 to 3.2.0	2.0.x
2.0.0 to 2.5	1.0.x
1.3.x	0.5.5.3
< 1.2.0	0.5.3

1.3. GORM Compatibility

Since plugin 5.x, auditing participates in the current Hibernate transaction and requires a Hibernate datastore. Persistence events from non-Hibernate datastores are not audited. The audit domain may use the same Hibernate datasource as the audited entity or another configured Hibernate datasource.

Chapter 2. Installation

2.1. Prerequisites

The 7.x plugin line supports Grails 8 and Java 21 only. Use the 6.x plugin line for Grails 7 applications.

2.2. Repository and Runtime Dependency

Applications use only the runtime plugin dependency. During 7.0.x snapshot development, configure the scoped Maven Central Snapshots repository and add the snapshot version in `build.gradle`:

```
repositories {
    maven {
        url = uri('https://central.sonatype.com/repository/maven-snapshots/')
        mavenContent { snapshotsOnly() }
        content {
            includeModule('org.grails.plugins', 'audit-logging')
            includeModule('org.grails.plugins', 'audit-logging-cli')
        }
    }
}

dependencies {
    implementation 'org.grails.plugins:audit-logging:7.0.0-SNAPSHOT'
}
```

When released 7.x artifacts are available, use the released version from Maven Central. The snapshot is not a final release, and availability depends on publication.

2.3. Automatic CLI Companion

Do not add `org.grails.plugins:audit-logging-cli` to the application. The runtime JAR advertises that companion with the `Grails-Cli-Artifact` manifest entry. Grails 8 provisions the companion only for CLI execution, so its dependencies stay out of `bootRun`, `bootJar`, and `bootWar`.

2.4. Create the Audit Domain Artifact

For a new application, run quickstart once with the direct Grails CLI:

```
grails audit-quickstart org.example.audit AuditLogEvent
```

Or start the interactive shell and run the same command:

```
./gradlew shell
grails> audit-quickstart org.example.audit AuditLogEvent
```

The command creates the audit domain class and registers it with `grails.plugin.auditLog.auditDomainClassName`. Open the generated class to adjust mappings and constraints for the application. Preserve its required nullability constraints, then update the database schema through DDL or the database-migration plugin.

The generated `dateCreated` and `lastUpdated` properties default to `java.util.Date`. Both properties may use any type supported by Grails' `DefaultTimestampProvider`, but they must use the same type.

`audit-quickstart` is a legacy script packaged in `META-INF/commands`, not an `ApplicationCommand`. Do not use `runCommand` or expect a generated `auditQuickstart` Gradle task.

2.5. Next Steps

Add the `Auditable` trait to every domain class that should be audited:

```
import grails.plugins.orm.auditable.Auditable

class MyDomain implements Auditable {
    String value
}
```

To enable stamping, add the `Stampable` trait as well:

```
import grails.plugins.orm.auditable.Auditable
import grails.plugins.orm.auditable.Stampable

class MyDomain implements Auditable, Stampable {
    String value
}
```

See [Configuration](#) to configure audit behavior.

Chapter 3. Upgrading to 7.0.x

3.1. 6.x to 7.x Checklist

1. Upgrade the application to Grails 8 and Java 21.
2. Replace the 6.x runtime dependency with `org.grails.plugins:audit-logging:7.0.0-SNAPSHOT` while the 7.x snapshot is the available development artifact.
3. Add `https://central.sonatype.com/repository/maven-snapshots/` to the build repositories when resolving the snapshot. Use Maven Central and a released version when a released 7.x artifact is available.
4. Keep the existing `AuditLogEvent` class, its mappings and constraints, and all `grails.plugin.auditLog.*` configuration.
5. Do not rerun `audit-quickstart`. It does not overwrite the existing class and can append duplicate configuration. If you intentionally redesign the audit domain, update the class and configuration manually.
6. Run `./gradlew --refresh-dependencies shell` if the CLI companion is not discovered.

3.2. Breaking Changes

Plugin 7.x supports Grails 8 and Java 21 only. It does not support Grails 7.

The CLI companion is now a separate artifact. Consumer applications still declare only `org.grails.plugins:audit-logging`. The runtime JAR advertises `org.grails.plugins:audit-logging-cli` through `Grails-Cli-Artifact`, and Grails 8 provisions it only when executing CLI commands. Do not add the companion directly.

`audit-quickstart` remains available as `grails audit-quickstart <package> <DomainClass>`, or from `./gradlew shell` followed by `audit-quickstart <package> <DomainClass>`. It is a legacy script in `META-INF/commands`, not an `ApplicationCommand`, so do not use `runCommand` or a generated `auditQuickstart` task.

3.3. Unchanged Behavior

The audit domain class remains application-owned. Existing customizations continue to be the source of truth, and configuration keys remain under `grails.plugin.auditLog.*`.

The auditing and stamping traits, including `Auditable` and `Stampable`, keep their established usage. Follow the configuration guide for supported settings.

Chapter 4. Upgrading to 5.0.x

- The 5.0.0 version has some significant changes to the 4.0.x version.
 - The configuration keys are the same as in 4.0.x
 - The mechanism to audit log transaction data has changed. We now participate in transactions and queue up audit logs. The audit logs are flushed with the current transaction. If the current transaction fails, no audit logs of this transaction are committed. Due to this major change, the plugin is currently not ORM agnostic and for now depends on Hibernate. As the plugin itself does not support other GORM implementations like Mongo (see #179), this shouldn't affect current projects.

Chapter 5. Upgrading to 4.0.x

The 4.0.0 version of the plugin is backwards compatible with 3.0.x. Any existing configuration and customization should work out of the box.

Chapter 6. Upgrading to 3.0.x

The 3.0.x version of the plugin is a major rewrite of the codebase. Therefore you need to upgrade your code for this version.

6.1. Domain Classes

Prior versions of the plugin used a static property syntax to enable and customize logging:

```
// Legacy enable/disable
static auditable = true

// Legacy whitelist
static auditable = [auditableProperties: ['name', 'famous', 'lastUpdated']]
```

Starting with version 3.0.0, the plugin uses an `Auditable` trait and/or a `Stampable` trait as both a marker as well as a way to configure and override auditing behavior.

For the first example, just adding `implements Auditable` will enable the default behavior.

```
import grails.plugins.orm.auditable.Auditable
class MyDomain implements Auditable {
    ..
}
```

For the second example above, you simply override the appropriate method on the `Auditable` trait, in this case the method is `getLogIncluded()`:

```
@Override
Collection<String> getLogIncluded() {
    ['name', 'famous', 'lastUpdated']
}
```



The getter methods on `Auditable` all follow the `getLog*()` format to minimize collisions with user methods. Typically they are `getLog<Configuration Name>()` followed by whatever the configurable value (included, excluded, mask) with a few exceptions.

6.2. Configuration Changes

To support unifying the configuration, trait, and context usage, some of the configuration properties have been renamed. The following table should help to map old values and types to the new ones:

Prior Name	New Name	New Type
auditableProperties	included	Collection<String>
defaultIgnore	excluded	Collection<String>
ignore	excluded	Collection<String>
defaultMask	mask	Collection<String>
nonVerboseDelete	verboseEvents	Collection<AuditEventType>
transactional	Removed	Inherits existing transactionality
actorClosure	Removed	Use AuditRequestResolver
actorKey	Removed	Use AuditRequestResolver
sessionAttribute	Removed	Use AuditRequestResolver
handlersOnly	Removed	Use Grails callback methods
stampAlways	Removed	Use a TraitInjector with Stampable

6.3. Transactional Behavior

Previously, the audit logging plugin had a **transactional** flag that indicated whether to include the audit log saves in a transaction. However, the audit logging plugin should really just participate (or not) in existing transactions and not make any attempts to control transactionality at that grain. For most usage, you want your Audit domain instances to be atomically created in the same transaction as the changes that are triggering them. If there's no existing transaction for the changes, it's not clear why there *should* be a transaction just for the audit events or vice versa.

The Audit instances are still saved in a new session within any existing transaction.

6.4. Handlers Removed

The handler callbacks such as **onChange**, **onSave**, etc. have been removed starting with version 3.0.0.

This behavior is already provided by Grails now using the **before*** and **after*** callback methods.

For example, you could do something like:

```
def beforeInsert() {
    def dirtyAudit = getAuditableDirtyPropertyNames()

    // Do something special if certain properties are dirty, etc.
}
```

6.5. Actor Closure

The actor closure has been replaced with a more formalized `AuditRequestResolver` strategy for resolving actor and URI information.

By default, the plugin uses the `DefaultAuditRequestResolver`, which gets Principal information from the current Grails web request context. This is essentially the same as the prior default actor closure.

If your application uses Spring Security, the plugin will register an instance of the `SpringSecurityRequestResolver` which will use the `springSecurityService` to resolve the current principal.

For other security frameworks, you can implement the `AuditRequestResolver` interface and register a bean named `auditRequestResolver` to override the resolver provided by the plugin.

6.6. Stampable Trait

The `@Stamp` annotation was removed in favor of a `Stampable` trait. This keeps the usage more in line with the direction of Grails in general as well as simplifying the implementation and usage.

However, there are some limitations to implementing this as a trait:

- Property names are `dateCreated`, `lastUpdated`, `createdBy`, `lastUpdatedBy` and are not configurable
- If you want to customized the constraints, you must do so in your domain class
- The values are technically populated on the `ValidationEvent` since they are non-nullable and must be populated prior to the default property validation.

The `stampAlways` has been removed. If you want to mark **all** of your domain objects as stampable, you could define the following `TraitInjector`:

```
@CompileStatic
class StampableTraitInjector implements TraitInjector {
    @Override
    Class getTrait() {
        Stampable
    }

    @Override
    String[] getArtefactTypes() {
        ['Domain'] as String[]
    }
}
```

Chapter 7. Configuration

The plugin configuration can be specified in the application.groovy file. It supports Environments blocks for environment-specific configuration.



Since version 2.0.0, the configuration key has changed from `auditLog` to `grails.plugin.auditLog`. If you use the old configuration key, the plugin will log a notice.

Property	Default Value	Meaning
grails.plugin.auditLog.auditDomainClassName	(Required) Use <code>grails audit-quickstart <package> <DomainClass></code> to create the artifact in a new application. Existing applications should retain their configured audit domain class.	Domain Class Name for the Audit Domain class.
grails.plugin.auditLog.disabled	<code>false</code>	If set to true, audit logging is completely disabled (no listeners are registered).
grails.plugin.auditLog.verbose	<code>true</code>	Column by column change logging in insert and delete events (enabled by default). You can disable it if you are not interested in value changes, or if you want to set verbosity based on event type (see verboseEvents setting below)
grails.plugin.auditLog.verboseEvents	<code>[INSERT, UPDATE, DELETE]</code>	Set of <code>AuditEventType</code> enums. Enables verbose logging on a per-event type basis. If <code>verbose = true</code> (default), all event types are logged verbosely, regardless of this setting.
grails.plugin.auditLog.ignoreEvents	<code>[]</code>	Set of <code>AuditEventType</code> enums that can be used to disable logging on a per-event type basis. For example, setting <code>[AuditEventType.DELETE]</code> would disable audit logging for all DELETE events only.
grails.plugin.auditLog.failOnError	<code>false</code>	If the save of the Audit domain fails, enabling this will cause the failure to be thrown out of the listener potentially failing the entire transaction.

Property	Default Value	Meaning
grails.plugin.auditLog.logIds	true	If set to true, object-ids of associated objects are logged.
grails.plugin.auditLog.logFullClassName	true	Log the full entity name including the package name
grails.plugin.auditLog.defaultActor	'SYS'	The default actor (user) to log if no user context can be found using the configured <code>AuditRequestResolver</code> implementation (see below).
grails.plugin.auditLog.excluded	['version', 'lastUpdated', 'lastUpdatedBy']	These properties are not logged and will not trigger verbose logging when changed.
grails.plugin.auditLog.included	null (all persistent properties)	This is a global whitelist of properties to log and will override anything in the excluded list. A null value indicates that all persistent properties not in the <code>excluded</code> list specifically should be logged.
grails.plugin.auditLog.mask	['password']	Any property with this name will be masked using the <code>propertyMask</code> attribute (below).
grails.plugin.auditLog.propertyMask	*****	String to use when masking properties.
grails.plugin.auditLog.truncateLength	maxSize of newValue/oldValue	Allow overriding the maximum length allowed in the <code>oldValue</code> and <code>newValue</code> fields before truncating. This property defaults to the max size allowed by the <code>constraint</code> , so it's best to just control it that way.
grails.plugin.auditLog.stampEnabled	true	Enable stamping support for entities with the <code>Stampable</code> trait (dateCreated, lastUpdated, createdBy, lastUpdatedBy).
grails.plugin.auditLog.usePersistentDirtyPropertyValues	true	By default, the old persistent value is obtained using <code>getPersistentValue()</code> . On newer Grails versions, this could fail. If the original value is always NULL in your case, you can set this property to "false", so <code>getOriginalValue()</code> is used.



You can view the above defaults in the `DefaultAuditLogConfig.groovy`

7.1. Verbose mode

When enabled (default), per-property logging is enabled for all operations.

```
grails.plugin.auditLog.verbose = true
```

or you can enable verbose logging for specific event types using:

```
grails.plugin.auditLog.verboseEvents = [AuditEventType.UPDATE]
```

If both `verbose=true` and `verboseEvents` are set, `verbose` has precedence.

If `verbose=true`, or `verboseEvents` contains `AuditEventType.DELETE`, deletion of an entity will cause all properties of the deleted entity to be logged, except associations (Collections), as this is currently not possible. See issue #153



Verbose logging will insert 1 row of data per property of your domain object for all or `verboseEvents` type operations. This can result in a large amount of additional data, sometimes far more than the actual operation itself. Consider disabling verbose mode for bulk operations using the `AuditLogContext.withoutVerboseAuditLog { }` closure.

7.2. Logging of Associated Ids

You can log the object-ids of associated objects. Logging will be performed in the format: "[id:<objId>]objDetails".

```
grails.plugin.auditLog.logIds = true
```

This setting is enabled by default.

7.3. Property Value Masking

You can configure properties to mask on a per-Domain-Class base. If properties are defined as masked, their values are not stored into the audit log table if verbose mode is enabled. Instead, a mask of "" will be logged.

By default, "password" properties are masked. You can mask property fields in domain classes like this:

```
@Override
Collection<String> getLogMask() {
    ['password', 'otherField']
}
```

7.4. Verbose Log Truncation Length

If you enabled verbose mode, you can configure the truncation length of detail information in the `oldValue` and `newValue` columns (Default is 255). Configure the `truncateLength` in `application.groovy`:

```
truncateLength = 400
```



When you set `truncateLength` to a value > 255 you must ensure that `oldValue` and `newValue` fields in your audit-log domain class are large enough. Example setting with the same `maxSize` constraints as the former "largeValueColumnTypes" setting:

```
static constraints = {  
    // For large column support (as in < 1.0.6 plugin versions)  
    oldValue(nullable: true, maxSize: 65534)  
    newValue(nullable: true, maxSize: 65534)  
}
```

When you forget to set the constraints in your `AuditLog` class while setting `truncateLength > 255`, a truncation warning may occur and only partial information is logged.

7.5. Disable All Auditing

You can disable auditing by config. If you disable auditing, event handlers are still triggered but no changes are committed to the audit log table. This can be used e.g. if you need to bootstrap many objects and want to programmatically disable auditing to not slow down the bootstrap process or if you want to audit log by Environment.

```
grails.plugin.auditLog.disabled = true
```

This setting is "false" by default (auditing is enabled).

7.6. Log Full Class Name

By default, only the entity class name is logged. If you want to log the entity full name (including the package name), you can enable full logging. Thanks to `tcrossland` for this feature.

```
grails.plugin.auditLog.logFullClassName = true
```

This setting is "true" by default (full name is logged).

7.7. Ignoring Specific Events

To ignore certain events on a per-domain base, override the `getLogIgnoreEvents()` method:

```
@Override
Collection<AuditEventType> getLogIgnoreEvents() {
    [AuditEventType.INSERT]
}
```

You can also ignore them either globally with:

```
grails.plugin.auditLog.ignoreEvents = [AuditEventType.INSERT]
```

Or for a specific logging context by using:

```
AuditLogContext.withConfig(ignoreEvents = [AuditEventType.INSERT]) {
    //
    // Anything here will only log UPDATE and DELETE events
    //
}
```

7.8. Runtime Overrides

See the [Context Overrides](#) for help using `AuditLogContext` to override configuration for a block.

7.9. Example Configuration

Example `application.groovy` configuration with various settings as described above:

```
// AuditLog Plugin config
grails {
    plugin {
        auditLog {
            auditDomainClassName = 'my.example.project.MyAuditTrail'
            verbose = false
            verboseEvents = [AuditEventType.UPDATE, AuditEventType.INSERT]
            failOnError = true
            excluded = ['version', 'lastUpdated', 'lastUpdatedBy']
            mask = ['password']
            logIds = true
            stampEnabled = true
        }
    }
}
```

Chapter 8. Usage

You can use the `grails-audit-logging` plugin in several ways.

8.1. Basic

To enable auditing using the default configuration, simply implement the `Auditable` trait.

For example:

```
class Author implements Auditable {
    String name
    Integer age

    static hasMany = [books: Book]
}
```

This will enable auditing for the domain using your configuration in `grails.plugin.auditLog` merged with the defaults from `DefaultAuditLogConfig.groovy` (in the plugin jar).

8.2. Context Overrides

There are many scenarios where you want to use a default configuration for most operations, but need to explicitly override for certain scenarios.

One fairly common example is the need to disable audit logging completely for a particular set of changes. This is often the case for bulk operations or operations that might be performed "by the system" such as cleanup, purge, expiration jobs, etc.

8.2.1. Audit Log Context

You can use the `AuditLogContext` object to override almost any configuration parameter for a given block

For example:

```
// Enable verbose logging with full class name excluding only properties named 'ssn'
AuditLogContext.withConfig(verbose: true, logFullClassName: true, excluded: ['ssn']) {
    ... }

// Disable verbose logging
AuditLogContext.withConfig(verbose: false) { ... }

// Disable all logging
AuditLogContext.withConfig(disabled: true) { ... }
```

The last two are so common, there's a method that does them by name:

```
// Disable verbose logging
AuditLogContext.withoutVerboseAuditLog { ... }

// Disable all logging
AuditLogContext.withoutAuditLog { ... }
```



The AuditLogContext stores the context using ThreadLocal, so configuration is only present for the current thread. If you start something asynchronously, be sure you setup the context within the new execution thread.

8.3. Domain Overrides

The context gives you programmatic control over configuration of auditing but you can go even finer grained, if needed, to control auditing on a per-entity basis.

To override auditing for a specific entity or based on runtime values, just override the trait method:

```
// Assuming this domain has a status attribute, disable auditing for anything not
active, for example
@Override
boolean isAuditable() {
    this.status == ACTIVE
}
```

Another example is to override which properties get ignored for a specific entity:

```
@Override
Collection<String> getLogExcluded() {
    ['myField', 'version']
}
```

Control the specific whitelist of attributes logged:

```
@Override
Collection<String> getLogIncluded() {
    ['whiteListField1', 'whiteListField2']
}
```

Customize the id that is logged for the entity, for example instead of using Author's `id` property, log the Author's name and age as `Tolkien:85`:

```
@Override
```

```
String getLogEntityId() {  
    "${name}:${age}"  
}
```

The ability to override and augment the default behavior provided by the trait is what makes the trait implementation so flexible.



Most of the **Auditable** methods rely on the `AuditLogContext` for defaults. If you override the trait method, ensure you consider the keeping the default behavior if you are supplementing.

8.4. Customized Auditable Trait

You could also extend the trait entirely to override the default auditing behavior:

```
trait MyAuditable<D> extends Auditable<D> {  
    // Customize formatting behavior for new and old values  
    @Override  
    String convertLoggedPropertyToString(String propertyName, Object value) {  
        if (value instanceof MySpecialThing) {  
            return ((MySpecialThing)value).formatAsString()  
        }  
        super.convertLoggedPropertyToString(propertyName, value)  
    }  
  
    // Customize to populate custom attributes on the audit log entity  
    @Override  
    boolean beforeSaveLog(Object auditEntity) {  
        auditEntity.naturalKey = getNaturalKey()  
    }  
}
```

8.5. Request Resolvers

Audit logging usually requires auditing the user and/or request context for a specific action. The plugin supports a pluggable method of resolving request context by registering an **auditRequestResolver** bean that applications can override if needed.



Spring-Security-Core plugin between version 3.2.1 and 4.0.1 has been set to `MODE_INHERITABLETHREADLOCAL`. This can lead to current actor problems, as it might keep the former security context. This setting was changed back to the secure `MODE_THREADLOCAL` in Spring-Security-Core 4.0.1. If you use a Spring-Security-Core plugin and see weird actor behavior, either upgrade the Spring-Security-Core plugin, or explicitly set the `SecurityContextHolder` strategy to `MODE_THREADLOCAL`. <https://github.com/grails/grails-spring-security-core/issues/587>

8.5.1. Audit Request Resolvers

The plugin ships with two resolvers:

```
class DefaultAuditRequestResolver implements AuditRequestResolver { ... }
```

and

```
class SpringSecurityRequestResolver extends DefaultAuditRequestResolver { ... }
```

The default resolver uses the `principal` in the active `GrailsWebRequest` to resolve the Actor name and Request URI for logging purposes.

If a bean named `springSecurityService` is available, the second resolver is registered which uses the `currentUser()` method to resolve the user context.

For other authentication strategies, you can implement and override the `auditRequestResolver` bean with your own implementation of:

```
interface AuditRequestResolver {  
    /**  
     * @return the current actor  
     */  
    String getCurrentActor()  
  
    /**  
     * @return the current request URI or null if no active request  
     */  
    String getCurrentURI()  
}
```

Just register your resolver in `resources.groovy`:

```
beans = {  
    auditRequestResolver(CustomAuditRequestResolver) {  
        customService = ref('customService')  
    }  
}
```

Below are a few examples for other common security frameworks.

Acegi Plugin

```
/**  
 * @author Jorge Aguilera  
 */
```

```

class AcegiAuditResolver extends DefaultAuditRequestResolver {
    def authenticateService

    @Override
    String getCurrentActor() {
        authenticateService?.principal()?.username ?: super.getCurrentActor()
    }
}

```

CAS Authentication

```

import edu.yale.its.tp.cas.client.filter.CASFilter

class CASAuditResolver extends DefaultAuditRequestResolver {
    def authenticateService

    @Override
    String getCurrentActor() {
        GrailsWebRequest request = GrailsWebRequest.lookup()
        request?.session?.getAttribute(CASFilter.CAS_FILTER_USER)
    }
}

```

Shiro Plugin

```

@Component('auditRequestResolver')
class ShiroAuditResolver extends DefaultAuditRequestResolver {
    @Override
    String getCurrentActor() {
        org.apache.shiro.SecurityUtils.getSubject()?.getPrincipal()
    }
}

```

Chapter 9. Stamping

Stamping adds the following attributes to a domain object via the `Stampable` trait:

```
trait Stampable<D> extends GormEntity<D> {  
    Date dateCreated  
    Date lastUpdated  
  
    String createdBy  
    String lastUpdatedBy  
}
```

The attributes will assume the default constraints, which should be fine for most cases.

You can configure the `constraints` and `mappings` block as usual in your domain class to customize the stampable properties.

9.1. Stamping Configuration

Previous versions used an AST transformation to apply the stamping and could allow more configuration. For now, the stamping support has been boiled down to just the basic trait attributes and the ability to disable the plugin globally.

```
// Enable or disable stamping  
grails.plugin.auditLog.stampEnabled = true
```



If you disable stamping but have `Stampable` entities, they will likely fail validation since the plugin will not be populating the fields which are still added to the domain objects. The main reason to disable stamping is to prevent the listener registration in the case that you just aren't using the `Stampable` support.

If you want to mark **all** of your domain objects as stampable, you could define the following `TraitInjector`:

```
@CompileStatic  
class StampableTraitInjector implements TraitInjector {  
    @Override  
    Class getTrait() {  
        Stampable  
    }  
  
    @Override  
    String[] getArtefactTypes() {  
        ['Domain'] as String[]  
    }  
}
```

}

Chapter 10. Implementation

Most of the plugin code is marked as `@CompileStatic`.

Chapter 11. Auditing and Transactions

All audit events for the current transaction are queued up. They are flushed after the transaction has successfully been committed. This allows the `AuditLogListener` to use its own session and transaction.

11.1. AuditLogEventListener

The Audit Logging plugin registers a `PersistenceEventListener` (`AuditLogListener`) bean per datasource, which listens to GORM events.

11.2. StampEventListener

The plugin registers a separate `StampEventListener` that responds to validate events. Ideally, we would stamp when an insert or update event occurs, but since the attributes are non-null by default, we need to populate them before the default validation is triggered.

11.3. Plugin Descriptor

The Plugin Descriptor (`AuditLogListenerGrailsPlugin`) configures the plugin during startup.

- Configures the plugin either by default values - see `DefaultAuditLogConfig.groovy` - or by user configured settings.
- Registers a `PersistenceEventListener` bean per datasource

11.4. Auditable trait

Enabling auditing on a Domain class is done by implementing the `Auditable` trait.

11.5. Stampable trait

Enabling stamping on a Domain class is done by implementing the `Stampable` trait. This trait adds the properties `dateCreated`, `lastUpdated`, `createdBy` and `lastUpdatedBy` to the domain class.

Chapter 12. Change Log

- 7.0.0-SNAPSHOT
 - Raised the supported baseline to Grails 8 and Java 21. Grails 7 applications should remain on plugin 6.x.
 - Split CLI support into the `audit-logging-cli` companion. The runtime plugin advertises the companion so Grails provisions it for CLI commands without adding CLI dependencies to application runtime or packaged artifacts.
 - Updated plugin registration for the Grails 8 BeanRegistrar lifecycle.
 - Updated the build toolchain, example application, and CI to the Grails 8 and Java 21 baseline.
 - Improved configuration reload behavior so runtime configuration changes are applied consistently.
 - Fixed Windows wrapper termination and diagnostics when Java cannot be discovered.
 - Existing audit domain customizations and `grails.plugin.auditLog.*` configuration remain supported. Do not rerun quickstart during an upgrade; manage any intentional audit-domain redesign manually.
- 6.0.0
 - Updated for Grails 7
- 5.0.0
 - New: Participate in current Transaction by queuing audit logs. Supports nested transactions. Plugin is currently NOT ORM agnostic and needs Hibernate due to this change (PR #212). See documentation.
 - Fix #213 nested withoutAuditLog calls
 - Fix #192 Column `date_created` cannot be null (PR #218)
 - Fix #175 `javax.persistence.TransactionRequiredException: no transaction is in progress`
 - Fix #173 Changes to `withNewSession` semantics make it useless for audit log listener
- 4.0.2
 - Fix #210 Add support for non-Date `dateCreated/lastUpdated` types
 - Fix #203 Prevent exception if `AuditRequestResolverBean` is not initialized (e.g. in tests)
- 4.0.1
 - Fix #189 `IllegalArgumentException` in validator
- 4.0.0
 - Initial support for Grails 4
- 3.0.5
 - Add support for `getOriginalValue()` using `config` parameter.
 - Fix #193 `StackOverflow` with composite ID and `Auditable`

- 3.0.4
 - Fix #153 NPE on delete for Hibernate PersistentCollection associations
 - Fix documentation for verboseEvents
- 3.0.3
 - Fix #180, Fixed checking of the pull request in the travis-build.sh
 - Fix #176, #178, #179 Mongo datastores don't have a dataSourceName property
- 3.0.2
 - Fix #174 Merge default config before application ctx is refreshed
- 3.0.1
 - Fix #169 Crash if project is not using Spring-Security-Core plugin
 - Minor changes to EventLogController
- 3.0.0
 - Major rewrite of plugin to be trait based
 - Removed support for handler callbacks
 - Consolidated and cleaned up configuration
 - Added AuditLogContext to allow configuration overrides at the block level
- 2.0.6
 - Fix #142 Re-introduced truncateLength support (and changed config parameter from *TRUNCATE_LENGTH* to *truncateLength*)
 - Fixed verbose param default description in documentation (Thanks to Matthew Moss)
 - Fix #139 Allow whitelist of properties to be used instead of a ignore list.
- 2.0.5
 - Only pass session to actorClosure if a session actually exists. (Thanks to Dennie de Lange)
 - Updated syncHibernateState to use correct name array (Thanks to Matthias Born)
 - Fix ignore list not used for insert and delete (Backport from 1.x)
 - Fix #147 Document per-datasource auditLog.disabled config key
- 2.0.4
 - Added option to specify createdBy,lastUpdatedBy, dateCreated,lastUpdated fieldnames per domainclass
 - and removed blank constraint for nullable stampable properties.
 - Remove preDelete as stampable event, does not make sense to stamp a delete event. (Thanks to Dennie de Lange)
 - Constraint fixes
- 2.0.3
 - Fix #129 Issue with Hibernate stamping. Stamping was ignored with dynamicUpdate = true and stamping was ignored on cascading saves. (thanks to Dennie de Lange)

- Fix #130 Docs for verbose mode
- 2.0.2
 - Fix #118, use Grails 3.0.10 internally.
 - Fix #126 Support Many-To-Many logging (thanks to Andrey Zhuchkov)
- 2.0.1
 - Fix #117 Clean build. Version 2.0.0 had issues with Spring Security due to unclean build.
 - Fix #116 (partially). Replacement Patterns do work, but trailing dots are ignored for now due to Grails 3.0.x limitations.
- 2.0.0
 - First Grails 3 version. Thanks to Graeme Rocher.
 - Added audit-quickstart command to create the AuditLog domain artifact
 - #96 Make identifiers available in the maps during onChange event. Thanks to dmahapatro.
 - branch: master.
 - For 1.0.x plugin version (Grails2), see 1.x_maintenance branch
- 1.0.5
 - Migration of JIRA to GitHub Issues
 - Fix #92 (Support for ignoring certain Events)
 - Starting with this release, the main branch for the 1.0.x series is 1.x_maintenance. Master branch is for Grails 3.0 support, now. Both branches will be tested by Travis-CI.
- 1.0.4
 - GPAUDITLOGGING-69 allow to set uri per domain object
 - GPAUDITLOGGING-62 Add identifier in handler map
 - GPAUDITLOGGING-29 support configurable id mapping for AuditLogEvent
 - GPAUDITLOGGING-70 support configurable datasource name for AuditLogEvent
 - GPAUDITLOGGING-74 Impossible to log values of zero or false
 - GPAUDITLOGGING-75 Support automatic (audit) stamping support on entities
- 1.0.3
 - GPAUDITLOGGING-64 workaround for duplicate log entries written per configured dataSource
 - GPAUDITLOGGING-63 logFullClassName property
- 1.0.2
 - GPAUDITLOGGING-66
- 1.0.1
 - closures
 - nonVerboseDelete property

- provide domain identifier to onSave() handler
- 1.0.0
 - Grails >= 2.0
 - ORM agnostic implementation
 - major cleanup and new features
 - fix #99 Plugin not working with MongoDB as Only Database
 - Changed issue management url to GH.
 - #13 Externalize AuditTrailEvent domain to user
- 0.5.5.3
 - Added ability to disable audit logging by config.
- 0.5.5.2
 - Added issueManagement to plugin descriptor for the portal. No changes in the plugin code.
- 0.5.5.1
 - Fixed the title. No changes in the plugin code.
- 0.5.5
 - collections logging
 - log ids
 - replacement patterns
 - property value masking
 - large fields support
 - fixes and enhancements
- 0.5.4
 - compatibility issues with Grails 1.3.x
- 0.5.3
 - GRAILSPLUGINS-2135
 - GRAILSPLUGINS-2060
 - an issue with extra JAR files that are somehow getting released as part of the plugin
- 0.5.2
 - GRAILSPLUGINS-1887 and GRAILSPLUGINS-1354
- 0.5.1
 - fixes regression in field logging
- 0.5
 - GRAILSPLUGINS-391
 - GRAILSPLUGINS-1496
 - GRAILSPLUGINS-1181

- GRAILSPUGINS-1515
- GRAILSPUGINS-1811
- changes to AuditLogEvent domain object uses composite id to simplify logging
- changes to AuditLogListener uses new domain model with separate transaction
- for logging action to avoid invalidating the main hibernate session.
- 0.4.1
 - repackaged for Grails 1.1.1 see GRAILSPUGINS-1181
- 0.4
 - custom serializable implementation for AuditLogEvent so events can happen inside a webflow context.
 - tweak application.properties for loading in other grails versions
 - update to views to show URI in an event
 - fix missing oldState bug in change event
- 0.3
 - actorKey and username features allow for the logging of user or userPrincipal for most security systems.
 - Fix #31 disable hotkeys for layout.