Grails Audit Logging Plugin - Reference Documentation

Robert Oschwald

Version 2.0.1, 2016-02-04

Table of Contents

Chapter 1. Description

The Audit Logging plugin adds Grails GORM Events based Audit Logging capabilities to a Grails project and it also adds support to domain models for hooking into the Grails GORM events system.

Support for the following closures are added: onSave, onDelete, and onChange. The onChange closure can be used in two ways. First, it can be used with no parameters or it can be used with two parameters, oldMap and newMap. The first parameter is a LinkedHashMap representing the old state of the object before the change was applied, the second parameter is a LinkedHashMap representing the state of the object after changes are applied.

GORM Events hook into the "beforeInsert" and the "beforeUpdate" events which work great for preventing updates but do not work well for "Audit Logging" where we would need critical information about the entity that is only available after these actions complete. We've chosen to prefix the handler names with "on" so that they do not conflict with other handler names in other existing plugins.

1.1. Compatibility

- For Grails 3.x use version 2.0.0 or above.
- For Grails 2.x use version 1.0.0 or above.
- For Grails 1.3.x use version 0.5.5.3.
- For Grails ⇐ 1.2.0 use version 0.5.3.

1.2. ORM implementation dependency

Starting with version 1.0.0, this plugin is ORM mapper agnostic, so you can use it with the ORM mapper of your choice (Hibernate3, Hibernate4, MongoDB, etc.).

Please note, that only Hibernate3 and Hibernate4 are tested during development. If an issue occurs with your used ORM mapper, please file a GitHub issue.

Chapter 2. Change Log

2.0.1

- Fix #117 Clean build. Version 2.0.0 had issues with Spring Security due to unclean build.
- Fix #116 (paritially). Replacement Patterns do work, but trailing dots are ignored for now due to Grails 3.0.x limitations.
    - 2.0.0
- Grails 3 support.
- audit-quickstart command to create the AuditLog domain artifact
- 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
- GRAILSPLUGINS-1515
- GRAILSPLUGINS-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 GRAILSPLUGINS-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.

Chapter 3. Installation

Add to your build.gradle project dependencies block:

```
/** Project dependencies **/
dependencies {
   ...
   compile 'org.grails.plugins:audit-logging:{version}'
}
```

Run

```
    grails compile
```

or another script that triggers dependency resolution. Afterwards, perform audit-quickstart to generate the domain classes and add the initial configuration settings.

3.1. Create your Domain Artifact using audit-quickstart

After installing the plugin, you must perform the following command to let the plugin create the audit-logging domain class within your project.

```
    grails audit-quickstart <your.package.name> <YourAuditLogEventClassName>
```

example:

```
    grails audit-quickstart org.myaudit.example AuditTrail
```

Afterwards, set your needed mapping and constraint settings accordinghly in the created Domain Artifact.

Chapter 4. Plugin Usage

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

4.1. 1. Auditing of domain class

```
static auditable = true
```

Enables audit logging using the introduced domain class AuditLogEvent which will record insert, update, and delete events. Update events will be logged in detail with the property name and the old and new values.

4.2. 2. Auditing plus domain event handlers

You may use the optional event handlers in your Domain classes. Example:

```
class Person {
 static auditable = true
 Long id
 Long version
 String firstName
 String middleName
 String lastName

 String email

 static constraints = {
     firstName(nullable:true,size:0..60)
     middleName(nullable:true,size:0..60)
     lastName(nullable:false,size:1..60)
     email(email:true)
 }

 def onSave = {
     println "new person inserted"
     // may optionally refer to newState map
 }
 def onDelete = {
     println "person was deleted"
     // may optionally refer to oldState map
 }
 def onChange = { oldMap,newMap ->
    println "Person was changed ..."
    oldMap.each({ key, oldVal ->
        if(oldVal != newMap[key]) {
         println " * $key changed from $oldVal to " + newMap[key]
        }
    })
 }//*/
}
```

4.3. 3. Use event handlers only (no auditing)

You may choose to disable the audit logging and only use the event handlers. You would do this by specifying in your domain class:

```
static auditable = [handlersOnly:true]
```

With "handlersOnly:true" specified, no AuditLogEvents will be persisted to the database and only the event handlers will be called.

Chapter 5. 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.plugins.auditLog". If you use the old configuration key, the plugin will log a notice.

5.1. Auditing Current User Information (actorClosure)

With version >= 0.5, additional configuration can be specified in the application.groovy file of the project to help log the authenticated user for various security systems. For many security systems the defaults will work fine. To specify a property of the userPrincipal to be logged as the actor name (the person performing the action which triggered the event, e.g. the username) add these lines to application.groovy:

```
grails {
  plugin {
   auditLog {
     actorClosure = { request, session ->
      session.user?.username
     }
   }
  }
}
```

It is currently not possible to define the actorClosure as a property like

```
grails.plugin.auditLog.actorClosure = {...}
```

5.1.1. Spring Security Core Plugin 1.1.2 or higher

If you save data from an unprotected URL (configAttribute:IS_AUTHENTICATED_ANONYMOUSLY), the principal is a String-object not a Principal-object. To cope with this behaviour you should use the following Closure definition in application.groovy:

```
import org.codehaus.groovy.grails.plugin.springsecurity.SpringSecurityUtils
```

```
grails {
  plugin {
     auditLog {
        actorClosure = { request, session ->
          if (request.applicationContext.springSecurityService.principal instanceof
groovy.lang.String){
           return request.applicationContext.springSecurityService.principal
          }
          def username = request.applicationContext.springSecurityService.principal?
.username
          if (SpringSecurityUtils.isSwitched()){
           username = SpringSecurityUtils.switchedUserOriginalUsername+" AS "+username
          }
          return username
        }
     }
  }
}
```

5.1.2. Spring Security Core 1.0.1 Plugin

Based off of Jorge Aguilera's example, the Spring Security Plugin uses the SpringSecurityService.

```
grails {
  plugin {
    auditLog {
      actorClosure = { request, session ->
       request.applicationContext.springSecurityService.principal?.username
      }
    }
  }
}
```

5.1.3. Acegi Plugin

Thanks to Jorge Aguilera for his example on how to integrate with the Acegi plugin:

```
grails {
  plugin {
    auditLog {
      actorClosure = { request, session ->
       return request.applicationContext.authenticateService.principal()?.username
      }
    }
  }
}
```

5.1.4. CAS Authentication

For example if you are using a system such as CAS you can specify the CAS user attribute using a special configuration property to get the CAS user name. In application.groovy add the following lines:

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

```
grails {
  plugin {
    auditLog {
     actorClosure = { request, session ->
       session?.getAttribute(CASFilter.CAS_FILTER_USER)
     }
    }
  }
}
```

... and the audit_log table will have a record of which user and what controller triggered the event.

5.1.5. Shiro Plugin

With Shiro, add the following lines to use the currently logged in user's username:

```
grails {
  plugin {
    auditLog {
     actorClosure = { request, session ->
       org.apache.shiro.SecurityUtils.getSubject()?.getPrincipal()
     }
    }
  }
}
```

5.1.6. Other security systems

If you are using a custom authentication system in your controller that puts the user data into the session you can set up the actorClosure to work with your security system instead.

5.2. Property Ignore List

It's possible to configure which properties get ignored by auditing. The default ignore field list is:

```
['version','lastUpdated'].
```

If you want to provide your own ignore list specify the ignore list like this in domain classes:

```
static auditable = [ignore:['version','lastUpdated','myField']]
```

If instead you want to trigger on version and lastUpdated changes you may specify an empty ignore list:

```
static auditable = [ignore:[]]
```

5.3. Verbose mode

You can enable verbose mode. If enabled, column by column change logging in insert and delete events is enabled. Old- and new values are stored in detailed to the audit logging table. Enable verbose logging with:

```
verbose = true
```

This setting is disabled by default.

When enabling verbose audit logging, you may get errors if you explicitly flush the session. In this case, do not enable verbose logging. Starting with version 1.0.1 of the plugin, additional closures are available to disable logging or verbose mode in a code block.

5.4. Logging of associated objectIds (since 0.5.5)

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

```
logIds = true
```

This setting is disabled by default.

5.5. Property value masking (since 0.5.5)

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:

```
static auditable = [mask:'password','otherField']
```

5.6. 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 TRUNCATE_LENGTH in application.groovy:

```
 TRUNCATE_LENGTH = 400 // don't forget to ensure "oldMap" and "newMap" fields are
large enough!
```

When you set TRUNCATE_LENGTH to a value > 255 you must ensure that oldMap and newMap 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 forgot to set the constraints in your AuditLog class while setting TRUNCATE_LENGTH > 255, a truncation warning may occur and only partial information is logged.

5.7. Transactional AuditLog events

In application.groovy, you may specify whether the Audit Log uses transactions or not. If set to true then the logger will begin and commit transactions around audit log save events. If set to false (the default), the AuditLog may be persisted without a transaction wrapping its call to save. This setting should not be changed from defaults lightly as it can cause problems in integration testing.

```
transactional = true
```

You are only likely to want to change the defaults if you are working with a transactional database in test and production.

5.8. Disable auditing by config (since 0.5.5.3)

You can disable auditing by config. If you disable auditing, event handlers are still triggered but no changes are comitted 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 Enviroment. With version >= 1.0.0 of the plugin, you can disable auditing on a per-datasource base as well Currently, disabling the plugin on a per-datasource base does not work. See GPAUDITLOGGING-68

```
disabled = true
```

Disabling in DataSource.groovy is currently not possible.

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

5.9. nonVerboseDelete logging (since 1.0.1)

If verbose logging is enabled (see above), you can log deletes in a non-verbose manner. This means, only the delete event is logged, but not the properties the deleted object hold prior the deletion.

```
nonVerboseDelete = true
```

This setting is "false" by default (verbosity of deletes depend on the verbose setting).

5.10. log full domain class name (since 1.0.3)

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.

```
logFullClassName = true
```

This setting is "false" by default (entity name is logged).

5.11. getAuditLogUri closure (since 1.0.4)

By default, the "uri" field is filled with the request uri which caused the action. You can define a closure "getAuditLogUri" on a per-domain object base to define what should be written to the AuditLog "uri" field.

```
class User {
  static auditable = true
  static belongsTo = [Client]

  def getAuditLogUri = {
    clientId as String
  }
}
```

You need to take special care how you obtain the "uri" data in the getAuditLogUri closure. It is recommended to not perform costly calls.

5.12. Domain class stamping support (since 1.0.4)

Since version 1.0.4, it is possible to enable domain class stamping support. With this feature enabled, all domain classes annotated with @Stamp or with field "static stampable = true" will get the fields dateCreated, lastUpdated, createdBy, lastUpdatedBy using an AST transformation. You can create your own StampASTTransformation implementation for your specific needs. The createdBy and lastUpdatedBy fieldnames can be declared in application.groovy. These fields will be filled with the result of the actor closure on the event PreInsert, PreUpdate and PreDelete. Thanks to tkvw for this feature.

```
stampEnabled = true // enable stamping support
stampAlways = false // always stamp domain classes, regardless of @Stamp or static
stampable = true existence
stampCreatedBy = 'createdBy' // fieldname
stampLastUpdatedBy = 'lastUpdatedBy' // fieldname
```

5.13. Ignoring certain events (since 1.0.5 / 2.0.0)

Since version 1.0.5, it is possible to ignore certain events on a per-domain base.

```
static auditable = [ignoreEvents:["onChange","onSave"]]
```

5.14. Example configuration

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

```
// AuditLog Plugin config
grails {
  plugin {
    auditLog {
     auditDomainClassName = 'my.example.project.MyAuditTrail'
     verbose = true // verbosely log all changed values to db
     logIds = true  // log db-ids of associated objects.
     TRUNCATE_LENGTH = 1000
     cacheDisabled = true
     logFullClassName = true
     replacementPatterns = ["local.example.xyz.":""] // replace with empty string.
     actorClosure = { request, session ->
        // SpringSecurity Core 1.1.2
        if (request.applicationContext.springSecurityService.principal instanceof
groovy.lang.String){
           return request.applicationContext.springSecurityService.principal
        }
        def username = request.applicationContext.springSecurityService.principal?
.username
        if (SpringSecurityUtils.isSwitched()){
           username = SpringSecurityUtils.switchedUserOriginalUsername+" AS "+username
        }
        return username
     }
     stampEnabled = true
     stampAlways = true
    }
  }
}
```

Chapter 6. Implementation

6.1. AuditLogEventListener

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

6.2. 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