Joda-Time Plugin - Reference Documentation

Authors: Rob Fletcher

Version: 1.5

Table of Contents

1 Introduction

The Joda-Time Plugin integrates the Joda-Time date/time library into Grails. The plugin…

- Provides the ability to bind from form inputs to Joda Time fields on domain or command objects.
- Adds unit testing support for domain classes that use Joda Time properties.
- Supports JSON and XML rendering of Joda Time types.
- Provides tag-libs for input and output of Joda Time data.
- Enhances Grails' scaffolding to support domain classes with Joda Time fields.
- Adds compatibility and consistency methods to Joda Time types so that they integrate better with Groovy.
- Provides templates for rendering inputs using the Fields plugin.

Known Issues

- It is currently not possible to do certain types of criteria query with *DateTime* properties mapped using *PersistentLocalDateTimeWithZone* (or any other multi-column Hibernate UserType). Aggregate functions (max, min, avg, count, etc.) in projections will not work on such properties and neither will the *'between'* criterion.
- Data binding to properties of embedded types does not work correctly (see GPJODATIME-21).

Release Notes

1.4

- GPJODATIME-24 Adds templates for the Fields plugin.
- GPJODATIME-15 Fixes compatibility with Grails < 2.
- GPJODATIME-24 Adds templates for Fields plugin compatibility.
- GPJODATIME-23 Fixes rendering of date/time pickers with null values.

1.3.1

- GPJODATIME-17 Adds binding and scaffolding support for `Instant`.

1.3

- Upgrades joda-time library to 2.0.
- Removes `joda-time-hibernate` as a default dependency so that plugin can be used with alternate GORM implementations.
- Adds Grails 2 unit testing support.
- Adds joda:time tag.
- Adds `next()` and `previous()` methods to `ReadableInstant` and `ReadablePartial`.
- Adds overridden `step` method on `Range`.

1.2

- Upgrades joda-time library to 1.6.2
- Upgrades joda-time-hibernate to 1.2

1.1

- Adds HTML5 binding and tag libs

1.0

- Adds joda:formatPeriod tag
- Allows joda:inputPattern to accept String attributes for convenience
- Fixes bug with joda:format that meant selected format was never displayed

0.5.1

- Upgrades joda-time-hibernate to version 1.2
- Allows no-selection option to work with StructuredDateTimeEditor

0.5

- Adds JSON and XML rendering support.
- Adds binding and scaffolding support for Duration and Period.
- Fixes compatibility problem with Grails 1.1 and 1.1.1.

0.4.3

- No longer automatically tries to install scaffolding templates (use grails install-joda-time-templates instead).
- list.gsp and show.gsp templates are not used when application's Grails version is 1.2+ as they are not needed.

0.4.2

- No longer tries to reinstall templates that are already installed.
- Adds DateTimeUtils.withCurrentTimestampFixed and withCurrentTImestampOffset
- Supports all Groovy mathematical operators on Joda Seconds, Minutes, Hours, Days, Months, Years, etc. classes.

0.4.1

- Fixes template installation on Windows machines where Ant's patch task does not work.

0.4

- Fixes corrupted template files from previous version.
- Fixes template compatibility with Grails 1.0.4.

0.3

- Adds the dynamic format(String) method on ReadableInstant and ReadablePartial.
- Fixes installation script for Grails 1.0.4.
- Plugin requires Grails 1.0.4 + as earlier versions don't support registering custom editors.

0.2

- Fixes bug where registration of structured date/time editor overrides registration of text -> date/time editor.

0.1

- Initial release

Contact

If you have questions, comments or suggestions please contact me via the Grails user mailing list. Please raise bugs against the JodaTime Plugin project on the Grails JIRA. Plugin source code is hosted on GitHub

2 Persistence

Persistence with Hibernate

In order to persist Joda Time properties with Hibernate you first need to add one of the following dependencies to your application's `BuildConfig.groovy` file:

To use the newer Usertype Library

```
compile "org.jadira.usertype:usertype.jodatime:1.9"
```

> ⚠ There is a version 2.0 of the Usertype library but it requires Hibernate 4 so is not currently compatible with Grails.

To use the older Joda-Time Hibernate Library

```
compile("joda-time:joda-time-hibernate:1.3") {
    excludes "joda-time", "hibernate"
}
```

Adding persistent types to your domain classes

To persist Joda Time properties you can use the Hibernate UserType implementations in the mapping block of your class. For example:

```
import org.joda.time.*
import org.jadira.usertype.dateandtime.joda.*

class Person {
    String name
    LocalDate birthdate
    static mapping = {
        birthdate type: PersistentLocalDate
    }
}
```

This even works with some of the special functionality in Grails. For example fields `dateCreated` and `lastUpdated` in a Grails domain object will be updated automatically by the framework. Such properties do not have to be `java.util.Date` instances the functionality works fine if they are `org.joda.time.DateTime` or other types instead.

Adding default type mappings

To avoid having to add type mappings to every single domain class in your application you can create a default mappings block in `Config.groovy` that maps Joda Time types to their Hibernate user type implementation. For example:

```
grails.gorm.default.mapping = {
    "user-type" type:
org.jadira.usertype.dateandtime.joda.PersistentDateTime, class:
org.joda.time.DateTime
    "user-type" type:
org.jadira.usertype.dateandtime.joda.PersistentLocalDate, class:
org.joda.time.LocalDate
    // … define as many other user type mappings as you need
}
```

The Joda Time plugin can even do this for you so long as you do not already have a `grails.gorm.default.mapping` section in your `Config.groovy` file. Just run `grails install-joda-time-gorm-mappings` and the plugin will add the standard mappings for all supported user types to your `Config.groovy` file.

If you are using the older Joda Time Hibernate library then the package names for the user type classes is `org.joda.time.contrib.hibernate`

Multi-column UserTypes

To use multi-column types such as PersistentDateTimeWithZone you need to include explicit mapping of both column names. For example:

```
import org.joda.time.*
import org.jadira.usertype.dateandtime.joda.*

class User {
    DateTime registered
    mapping {
        registered type: PersistentDateTimeWithZone, {
            column name: "registered_timestamp"
            column name: "registered_zone"
        }
    }
}
```

You can use any name you like for the columns.

It is currently not possible to do certain types of criteria query with *DateTime* properties mapped using *PersistentDateTimeWithZone* (or any other multi-column Hibernate UserType). Aggregate functions (max, min, avg, count, etc.) in projections will not work on such properties and neither will the *'between'* criterion.

Persistence with other GORM implementations

3 Data binding

Data Binding

The Joda-Time plugin adds automatic binding support for the following types…

- LocalTime
- LocalDate
- LocalDateTime
- DateTime
- Duration
- Period
- DateTimeZone
- Instant

Properties can be bound to simple text fields or to picker controls (see below).

Binding text inputs

By default text fields are bound in a locale-sensitive manner (e.g. to enter a LocalDate in the *en_GB* locale the format is *dd/MM/yy* , in the *en_US* locale the format is *MM/dd/yy* and so on). Alternatively formats can be defined per type in config using keys such as *'jodatime.format.org.joda.time.DateTime'* and *'jodatime.format.org.joda.time.LocalDate'* which is partiularly useful when using rich UI type controls that may require a fixed format.

HTML5 input types

The HTML5 standard supports several new input types that potentially useful for binding to Joda-Time properties. The input formats for those types are fixed rather than being locale-sensitive as they are designed for picker controls rendered by the browser or JavaScript. If you wish to use HTML5 inputs in your project simply set the config value *'jodatime.format.html5 = true'* in *Config.groovy* and the correct binding formats will be used.

> ⚠ Any formats configured using *'jodatime.format.<classname>'* will override the HTML5 format for that type.

HTML5 input formats

For reference, the formats for the various input types are as follows:

Input type	Format
month	yyyy-MM
week	xxxx-'W'ww
date	yyyy-MM-dd
time	HH:mm:ss.SSS
datetime-local	yyyy-MM-dd'T'HH:mm:ss.SSS
datetime	yyyy-MM-dd'T'HH:mm:ss.SSSZZ

For all types Seconds and milliseconds can be omitted on input but are always rendered on output. The time zone for *datetime* inputs can be either the literal *'Z'* representing the *UTC* time zone or a value such as *+05:30* or *-08:00*

4 Tags

Tag Libs

The plugin makes several new tags available. See the reference section for details.

5 Scaffolding

Scaffolding

The Joda-Time plugin enhances Grails' dynamic scaffolding so it is compatible with domain classes with the following types…

- LocalTime
- LocalDate
- LocalDateTime
- DateTime
- Duration
- Period
- DateTimeZone
- Instant

To install scaffolding templates use the command:

```
grails install-joda-time-templates
```

Create and edit views use the *joda:dateTimePicker, joda:datePicker, joda:timePicker* or *joda:periodPicker* (see above) as appropriate. Columns on list views are sortable.

6 JSON and XML

JSON and XML Rendering

The plugin registers JSON and XML converters for…

- LocalTime
- LocalDate
- LocalDateTime
- DateTime
- DateTimeZone

7 Integrating with other plugins

Integrating With Other Plugins

Fields

The Joda-Time plugin supplies templates so that inputs for Joda-Time properties will render correctly. You can override these templates in your own application if you want to render the inputs differently. See the Fields plugin documentation for details on how to do so.

Grails UI

The Grails-UI Plugin's datePicker tag works quite well with *Joda-Time* types such as *DateTime, LocalDate* or *LocalDateTime* instances. The tag's *formatString* argument can easily be set with *joda:inputPattern* (see above). The value attribute needs to be converted to a *java.util.Date* instance. Examples of use might be:

A *DateTime* property with time input:

```
<gui:datePicker id="myDateTimeProperty" value=
"${myInstance?.myDateTimeProperty?.toDate()}" includeTime="true"
formatString="${joda.inputPattern()}"/>
```

A *LocalDate* property:

```
<gui:datePicker id="myLocalDateProperty" value=
"${myInstance?.myLocalDateProperty?.toDateTime()?.toDate()}" formatString=
"${joda.inputPattern(type: org.joda.time.LocalDate)}"/>
```