Jodatime Plugin - Reference Documentation

Rob Fletcher

Version 2.1.0

Table of Contents

Chapter 1. Introduction

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

In addition to this document, you may want to read the official Joda-time documentation here.

1.1. Features

- 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.
- 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.

1.2. 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).

Old JIRA links for Grails are not working anymore, so please ignore the issues links in documentation. Also, all issues must be reported to Joda-time GitHub Issues.

1.3. Release Notes

2.1.0-SNAPSHOT

- Grails 3.3.x support.
- All tests converted to new Trait based tests
- Wrappers for fields-plugin removed. They cause too much trouble
- SimpleMapJodaTimeMarshaller currently moved out of compiled scope
- Correcting the field templates

2.0.0

- Migrated the plugin to support Grails 3

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

1.4. Acknowledgments

Many thanks to all the users who reported issues and sent pull requests.

1.4.1. Authors and Contributors

- Rob Fletcher
- Burt Beckwith
- James Cook
- Michael Cameron
- Graeme Rocher
- Sergey Ponomarev
- Robert Oschwald
- Colin Harrington
- Puneet Behl

- Soeren Berg Glasius

1.5. License

This plugin is released under the Apache License, Version 2.0

Chapter 2. Persistence

2.1. 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:

2.1.1. 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.

2.1.2. To use the older Joda-Time Hibernate Library

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

2.1.3. 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.

2.1.4. 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 `application.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`

2.1.5. 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.

2.2. Persistence with other GORM implementations

Chapter 3. 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).

3.1. 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 particularly useful when using rich UI type controls that may require a fixed format.

3.2. 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.

3.2.1. 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

Input type	Format
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*

Chapter 4. Tags

4.1. joda:dateField

4.1.1. Purpose

Renders an HTML5 *date* input for *Joda-Time* properties

4.1.2. Examples

```
<joda:dateField name="myProperty" value="${new LocalDate()}" />
<joda:dateField name="myProperty" value="${myBean.myProperty}" />
```

4.1.3. Description

The value should be a *ReadableInstant* or *ReadablePartial* that supports at least the *year*, *monthOfYear* and *dayOfMonth* fields and will be formatted correctly for the input type.

4.2. joda:datePicker

4.2.1. Purpose

Renders a date picker input for *Joda-Time* properties in a similar way to the standard `g:datePicker` tag

4.2.2. Examples

```
<joda:datePicker name="myDate" value="${new LocalDate()}" noSelection="['':'-Choose-
']"/>
<joda:datePicker name="myDate" value="${new LocalDate()}" years="${1930..1970}"/>
<joda:datePicker name="myDate" value="${new LocalDate()}" years="[1930, 1940, 1950,
1960, 1970]"/>
```

4.2.3. Description

This tag is based on the default *g:datePicker* tag and exhibits very similar functionality. However, it is designed to be used with *Joda-Time* properties. All the attributes are as-per *g:datePicker* except that *'value'* and *'default'* will expect either a ReadablePartial or ReadableInstant instance or a String in *ISO8601* date/time format (such a String can be a partial representation depending on the precision attribute).

4.3. joda:dateTimePicker

4.3.1. Purpose

Renders a date/time picker input for *Joda-Time* properties in a similar way to the standard `g:datePicker` tag

4.3.2. Examples

```
<joda:dateTimePicker name="myDate" value="${new DateTime()}" noSelection="['':'-
Choose-']"/>
<joda:dateTimePicker name="myDate" value="${new DateTime()}" precision="second"
years="${1930..1970}"/>
<joda:dateTimePicker name="myDate" value="${new DateTime()}" years="[1930, 1940, 1950,
1960, 1970]"/>
```

4.3.3. Description

This tag is based on the default *g:datePicker* tag and exhibits very similar functionality. However, it is designed to be used with *Joda-Time* properties. All the attributes are as-per *g:datePicker* except that *'value'* and *'default'* will expect either a DateTime or LocalDateTime instance or a String in *ISO8601* date/time format (such a String can be a partial representation depending on the precision attribute). The other difference from *g:datePicker* is that the *'second'* precision is supported (although like *g:datePicker* the tag uses *'minute'* as the default precision).

4.4. joda:dateTimeZoneSelect

4.4.1. Purpose

This tag renders a select for DateTimeZone values. It is very similar to the standard *g:timeZoneSelect* tag.

4.4.2. Examples

```
<joda:dateTimeZoneSelect name="myField" value="${myValue}" />
```

4.4.3. Description

Attributes

- `name` - The name for the backing form field (as per `g:textField` and other standard tags)
- `id` (optional) - The *id* for the backing form field. Defaults to the same as *name*
- `value` (optional) - The currently selected `DateTimeZone` value. Defaults to `DateTimeZone.getDefault()`

Unresolved directive in tags.adoc - include::ref/tags/dateTimeField.adoc[]

Unresolved directive in tags.adoc - include::ref/tags/dateTimeLocalField.adoc[]

inlcude::ref/tags/format.adoc[]

4.5. joda:formatPeriod

4.5.1. Purpose

This tag renders a Duration or Period value.

4.5.2. Examples

```
<joda:formatPeriod value="${myValue}" />
<joda:formatPeriod value="${myValue}" fields="days,hours,minutes" />
```

4.5.3. Description

Attributes

- `value` (required) - The value to format which can be a `Period` or `Duration` instance.
- `fields` (optional) - A comma separated list of the fields to provide input elements for. Valid values are *"years"*, *"months"*, *"weeks"*, *"days"*, *"hours"*, *"minutes"*, *"seconds"* and *"millis"* with the default being *"hours,minutes,seconds"*

Values are normalized using the *fields* specified according to the rules in Period.normalizedStandard although the tag will also silently drop *years* and *months* from the *value* if they are not contained in the specified *fields* as otherwise an `UnsupportedOperationException` would be thrown by Joda-Time.

4.5.4. Configuration

Default fields can be set in `Config.groovy` using the key *jodatime.periodpicker.default.fields*

4.6. joda:inputPattern

4.6.1. Purpose

This tag outputs the expected input pattern for a given type. It can be used for example to output a label to go alongside a text field or to configure a rich input control such as the http://grails.org/plugin/grails-ui *gui:datePicker* tag[Grails UI].

4.6.2. Examples

```
<joda:inputPattern/>
<joda:inputPattern type="org.joda.time.LocalDate"/>
<joda:inputPattern locale="fr"/>
```

4.6.3. Description

Attributes

- `type` (optional) - The type to output the pattern for. Can be a `Class` or the class name. Defaults to *DateTime*
- `locale` (optional) - The locale for the pattern. Can be a `Locale` object or an ISO locale string such as *"en_GB"*. Defaults to current request locale

4.7. joda:monthField

4.7.1. Purpose

Renders an HTML5 *month* input for *Joda-Time* properties

4.7.2. Examples

```
<joda:monthField name="myProperty" value="${new LocalDate()}" />
<joda:monthField name="myProperty" value="${myBean.myProperty}" />
```

4.7.3. Description

The value should be a *ReadableInstant* or *ReadablePartial* that supports the *year* and *monthOfYear* fields and will be formatted correctly for the input type.

4.8. joda:periodPicker

4.8.1. Purpose

This tag renders an input control for a Duration or Period value.

4.8.2. Examples

```
<joda:periodPicker name="myField" value="${myValue}" />
<joda:periodPicker name="myField" value="${myValue}" fields="days,hours,minutes" />
```

4.8.3. Description

Attributes

- `name` - The name for the backing form field (as per `g:textField` and other standard tags)
- `id` (optional) - The *id* for the backing form field. Defaults to the same as *name*
- `value` (optional) - The currently selected value which can be a `Period` or `Duration` instance. Defaults to `new Period()`

- `fields` (optional) - A comma separated list of the fields to provide input elements for. Valid values are *"years", "months", "weeks", "days", "hours", "minutes", "seconds"* and *"millis"* with the default being *"hours,minutes,seconds"*

Configuration

Default fields can be set in `Config.groovy` using the key *jodatime.periodpicker.default.fields*

Internationalization

Labels for each field can be overridden in `messages.properties` using the keys *org.joda.time.DurationFieldType.hours, org.joda.time.DurationFieldType.minutes* and so on.

4.9. joda:time

4.9.1. Purpose

This tag outputs an HTML5 `<time>` tag with a correctly formatted `datetime` attribute.

4.9.2. Examples

```
<joda:time value="${new LocalDate()}"/>
<!-- output: <time datetime="2011-11-03">03-Nov-2011</time> -->

<joda:time value="${new DateTime()}"/>
<!-- output: <time datetime="2011-11-03T17:25+00:00">03-Nov-2011 17:25:00</time> -->

<joda:time value="${new LocalDate()}" pubdate=""/>
<!-- output: <time datetime="2011-11-03" pubdate="">03-Nov-2011</time> -->

<joda:time value="${new LocalDate()}"><joda:format value="${it}" pattern="d
MMMM"/></joda:time>
<!-- output: <time datetime="2011-11-03">3 November</time> -->

<joda:time value="${new LocalDate()}" var="theDate"><joda:format value="${theDate}"
pattern="d MMMM"/></joda:time>
<!-- output: <time datetime="2011-11-03">3 November</time> -->
```

4.9.3. Description

If the tag has a body then the `value` attribute is passed to it (see example above). If the body is omitted then the value is formatted as per the `joda:format` tag. If the `value` attribute is omitted then the current `DateTime` is used. If the `value` attribute is `null` then the tag outputs nothing.

Attributes

- `value` (optional) - An instance of `ReadablePartial` or `ReadableInstant` or defaults to `new DateTime()`
- `var` (optional) - A name for the variable that is passed to the tag body. Defaults to `it`

4.10. joda:timeField

4.10.1. Purpose

Renders an HTML5 *time* input for *Joda-Time* properties

4.10.2. Examples

```
<joda:timeField name="myProperty" value="${new LocalTime()}" />
<joda:timeField name="myProperty" value="${myBean.myProperty}" />
```

4.10.3. Description

The value should be a *ReadableInstant* or *ReadablePartial* that supports at least the *hourOfDay* and *minuteOfHour* fields and will be formatted correctly for the input type.

4.11. joda:weekField

4.11.1. Purpose

Renders an HTML5 *week* input for *Joda-Time* properties

4.11.2. Examples

```
<joda:weekField name="myProperty" value="${new LocalDate()}" />
<joda:weekField name="myProperty" value="${myBean.myProperty}" />
```

4.11.3. Description

The value should be a *ReadableInstant* or *ReadablePartial* that supports the *year* and *weekyear* fields and will be formatted correctly for the input type.

Chapter 5. JSON and XML Rendering

The plugin registers JSON and XML converters for:

- LocalTime
- LocalDate
- LocalDateTime
- DateTime
- DateTimeZone

Chapter 6. Integrating with Other Plugins

6.1. 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.