Quick Reference

Version 2.1.0

Quick Reference

Chapter 1. Dynamic Methods

1.1. div

1.1.1. Purpose

Divides an instance of `Days`, `Hours`, `Minutes`, `Months`, `Seconds`, `Weeks` or `Years` by an integer.

1.1.2. Examples

```
assert Years.THREE / 3 == Years.ONE
```

1.1.3. Description

Provides compatibility with Groovy's mathematical operators but is otherwise identical to the `dividedBy()` method in the Joda-Time API.

1.2. format

1.2.1. Purpose

Converts a `ReadableInstant` or `ReadablePartial` to a `String`

1.2.2. Examples

```
new DateTime().format("yyyy-MM-dd")
new LocalTime().format("HH:mm")
```

1.2.3. Description

Returns a String representation of the object according to the specified pattern. See "DateTimeFormat":http://joda-time.sourceforge.net/api-release/org/joda/time/format/DateTimeFormat.html for description of patterns. This is provided for compatibility with Groovy's enhancment to the `Date` class.

1.3. multiply

1.3.1. Purpose

Multiplies an instance of `Days`, `Hours`, `Minutes`, `Months`, `Seconds`, `Weeks` or `Years` by an integer.

1.3.2. Examples

```
assert Years.ONE * 3 == Years.THREE
```

1.3.3. Description

Provides compatibility with Groovy's mathematical operators but is otherwise identical to the `multipliedBy()` method in the Joda-Time API.

1.4. negative

1.4.1. Purpose

Negates an instance of `Days`, `Hours`, `Minutes`, `Months`, `Seconds`, `Weeks` or `Years`

1.4.2. Examples

```
assert -Years.THREE == Years.years(-3)
```

1.4.3. Description

Provides compatibility with Groovy's mathematical operators but is otherwise identical to the `negated()` method in the Joda-Time API.

1.5. next

1.5.1. Purpose

Increments a `ReadableInstant` or `ReadablePartial` instance allowing use in Groovy ranges.

1.5.2. Examples

```
def today = new LocalDate()
assert today.next() == today.plusDays(1)
```

1.5.3. Description

The increment is generally 1 day for compatibility with the `next` method Groovy adds to `java.util.Date` however since not all `ReadablePartial` implementations support days the following exceptions exist:

- `org.joda.time.LocalTime` and `org.joda.time.TimeOfDay` increment by hours.
- `org.joda.time.YearMonth` increments by months.

1.6. previous

1.6.1. Purpose

Decrements a `ReadableInstant` or `ReadablePartial` instance allowing use in Groovy ranges.

1.6.2. Examples

```
def today = new LocalDate()
assert today.previous() == today.minusDays(1)
```

1.6.3. Description

The increment is generally 1 day for compatibility with the `previous` method Groovy adds to `java.util.Date` however since not all `ReadablePartial` implementations support days the following exceptions exist:

- `org.joda.time.LocalTime` and `org.joda.time.TimeOfDay` decrement by hours.
- `org.joda.time.YearMonth` decrements by months.

1.7. step

1.7.1. Purpose

Overrides the standard `step` method on a `Range` to produce a date/time range using a different field for the increment.

1.7.2. Examples

```
def start = new LocalDate()
def end = new LocalDate().plusYears(1)
def range = start..end
assert range.step(DurationFieldType.months()).size() == 13
assert range.step(2, DurationFieldType.months()).size() == 7
```

1.7.3. Description

Four variations exist:

- `List step(DurationFieldType)`
- `List step(int, DurationFieldType)`
- `void step(DurationFieldType, Closure)`
- `void step(int, DurationFieldType, Closure)`

1.8. withCurrentMillisFixed

1.8.1. Purpose

Fixes the current time in the scope of a Closure.

1.8.2. Examples

```
DateTimeUtils.withCurrentMillisFixed(0L) {
    assert DateTime.currentTimeMillis() == 0
}
assert DateTime.currentTimeMillis() != 0
```

1.8.3. Description

Allows tests to simulate the current time during the scope of a Closure. This uses the Joda-Time methods setCurrentTimeMillis and setCurrentMillisSystem before and after invoking the Closure.

1.9. withCurrentMillisOffset

1.9.1. Purpose

Offsets the current time in the scope of a Closure.

1.9.2. Examples

```
DateTimeUtils.withCurrentMillisOffset(1000) {
    assert DateTime.currentTimeMillis() > System.currentTimeMillis()
}
```

1.9.3. Description

Allows tests to simulate the current time during the scope of a Closure. This uses the Joda-Time methods setCurrentTimeMillis and setCurrentMillisSystem before and after invoking the Closure.

Chapter 2. Tags

2.1. joda:dateField

2.1.1. Purpose

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

2.1.2. Examples

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

2.1.3. 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.

2.2. joda:datePicker

2.2.1. Purpose

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

2.2.2. 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]"/>
```

2.2.3. 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).

2.3. joda:dateTimePicker

2.3.1. Purpose

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

2.3.2. 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]"/>
```

2.3.3. 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).

2.4. joda:dateTimeZoneSelect

2.4.1. Purpose

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

2.4.2. Examples

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

2.4.3. 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::tags/dateTimeField.adoc[]

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

inlcude::tags/format.adoc[]

2.5. joda:formatPeriod

2.5.1. Purpose

This tag renders a Duration or Period value.

2.5.2. Examples

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

2.5.3. 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.

2.5.4. Configuration

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

2.6. joda:inputPattern

2.6.1. 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].

2.6.2. Examples

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

2.6.3. 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

2.7. joda:monthField

2.7.1. Purpose

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

2.7.2. Examples

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

2.7.3. 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.

2.8. joda:periodPicker

2.8.1. Purpose

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

2.8.2. Examples

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

2.8.3. 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.

2.9. joda:time

2.9.1. Purpose

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

2.9.2. 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> -->
```

2.9.3. 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`

2.10. joda:timeField

2.10.1. Purpose

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

2.10.2. Examples

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

2.10.3. 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.

2.11. joda:weekField

2.11.1. Purpose

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

2.11.2. Examples

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

2.11.3. 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.