Seam International Module

Reference Guide

Ken Finnigan

Lincoln Baxter III

Introduction v
1. Installation 1
2. Locales 3
2.1. Default Locale 3
2.2. User Locale 4
2.3. Available Locales 4
3. Timezones 7
3.1. Default TimeZone 7
3.2. User TimeZone 7
3.3. Available TimeZones 8
4. Messages 9

Introduction

The goal of Seam International is to provide a unified approach to configuring locale, timezone and language. With features such as Status messages propogation to UI, multiple property storage implementations and more.

Installation

Most features of Seam International are installed automatically by including `seam-international.jar` in the web application library folder. If you are using *Maven* [http://maven.apache.org/] as your build tool, you can add the following dependency to your `pom.xml` file:

```
<dependency>
   <groupId>org.jboss.seam</groupId>
   <artifactId>seam-international</artifactId>
   <version>${seam-international-version}</version>
</dependency>
```

Tip

Replace ${seam-international-version} with the most recent or appropriate version of Seam International.

Locales

2.1. Default Locale

In a similar fashion to TimeZones we have an application `Locale` retrieved by

```
@Inject
java.util.Locale lc;
```

accessible via EL with "defaultLocale".

By default the `Locale` will be set to the JVM default, unless you override the `DefaultLocaleProducer` Bean via the Seam Config module. Here are a few examples of XML that can be used to define the various types of `Locales` that are available.

This will set the default language to be French.

```
<beans xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:s="urn:java:seam:core"
  xmlns:lc="urn:java:org.jboss.seam.international.locale"
  xsi:schemaLocation="
    http://java.sun.com/xml/ns/javaee
    http://docs.jboss.org/cdi/beans_1_0.xsd">

   <lc:DefaultLocaleProducer>
      <s:replaces/>
      <lc:defaultLocaleKey>fr</lc:defaultLocaleKey>
   </lc:DefaultLocaleProducer>
</beans>
```

This will set the default language to be English with the country of US.

```
<beans xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:s="urn:java:seam:core"
  xmlns:lc="urn:java:org.jboss.seam.international.locale"
  xsi:schemaLocation="
    http://java.sun.com/xml/ns/javaee
    http://docs.jboss.org/cdi/beans_1_0.xsd">
    <lc:DefaultLocaleProducer>
        <s:replaces/>
        <lc:defaultLocaleKey>en_US</lc:defaultLocaleKey>
    </lc:DefaultLocaleProducer>
</beans>
```

As you can see from the previous examples, you can define the `Locale` with `lang_country_variant`. It's important to note that the first two parts of the locale definition are not expected to be greater than 2 characters otherwise an error will be produced and it will default to the JVM `Locale`.

2.2. User Locale

The Locale associated with the User Session can be retrieved by

```
@Inject
@UserLocale
java.util.Locale locale;
```

which is EL accessible via `userLocale`.

By default the `Locale` will be the same as that of the application when the User Session is initially created. However, changing the User's `Locale` is a simple matter of firing an event to update it. An example would be

```
@Inject
@Changed
Event<java.util.Locale> localeEvent;

public void setUserLocale()
{
    Locale canada = Locale.CANADA;
    localeEvent.fire(canada);
}
```

2.3. Available Locales

We've also provided a list of available Locales that can be accessed via

```
@Inject
```

```
List<java.util.Locale> locales;
```

The locales that will be returned with this can be defined with XML configuration of the `AvailableLocales` Bean such as

```
<beans xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:s="urn:java:ee"
  xmlns:lc="urn:java:org.jboss.seam.international.locale"
  xsi:schemaLocation="
    http://java.sun.com/xml/ns/javaee
    http://docs.jboss.org/cdi/beans_1_0.xsd">

   <lc:LocaleConfiguration>
      <lc:supportedLocaleKeys>
         <s:value>en</s:value>
         <s:value>fr</s:value>
      </lc:supportedLocaleKeys>
   </lc:LocaleConfiguration>
</beans>
```

Timezones

To support a more developer friendly way of handling TimeZones we have incorporated the use of Joda-Time through their `DateTimeZone` class. Don't worry, it provides convenience methods to convert to JDK `TimeZone` if required.

3.1. Default TimeZone

Starting at the application level the module provides a `DateTimeZone` that can be retrieved with

```
@Inject
DateTimeZone applicationTimeZone;
```

It can also be accessed through EL by the name "defaultTimeZone"!

By default the `TimeZone` will be set to the JVM default, unless you override the `DefaultTimeZoneProducer` Bean using the Seam Config module. For instance, adding this XML into `seam-beans.xml` or `beans.xml` will change the default `TimeZone` of the application to be Tijuana!

```
<beans xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:s="urn:java:seam:core"
  xmlns:tz="urn:java:org.jboss.seam.international.timezone"
  xsi:schemaLocation="
    http://java.sun.com/xml/ns/javaee
    http://docs.jboss.org/cdi/beans_1_0.xsd">

   <tz:DefaultTimeZoneProducer>
      <s:specializes/>
      <tz:defaultTimeZoneId>America/Tijuana</tz:defaultTimeZoneId>
   </tz:DefaultTimeZoneProducer>
</beans>
```

3.2. User TimeZone

We also have a `DateTimeZone` that is scoped to the User Session which can be retrieved with

```
@Inject
@UserTimeZone
```

```
DateTimeZone userTimeZone;
```

It can also be accessed through EL using "userTimeZone".

By default the `TimeZone` will be the same as the application when the User Session is initialised. However, changing the User's `TimeZone` is a simple matter of firing an event to update it. An example would be

```
@Inject
@Changed
Event<DateTimeZone> tzEvent;

public void setUserTimeZone()
{
    DateTimeZone tijuana = DateTimeZone.forID("America/Tijuana");
    tzEvent.fire(tijuana);
}
```

3.3. Available TimeZones

We've also provided a list of available TimeZones that can be accessed via

```
@Inject
List<DateTimeZone> timeZones;
```

Messages

There are currently two ways to create a message within the module.

The first would mostly be used when you don't want to add the generated message directly to the UI, but want to log it out, or store it somewhere else

```
@Inject
MessageFactory factory;

public String getMessage()
{
    MessageBuilder builder = factory.info("There are {0} cars, and they are all {1}; {1} is the best
 color.", 5, "green");#
    return builder.build().getText();
}
```

The second is to add the message to a list that will be returned to the UI for display.

```
@Inject
Messages messages;

public void setMessage()
{
    messages.info("There are {0} cars, and they are all {1}; {1} is the best color.", 5, "green");
}
```

Either of these methods supports the four message levels which are info, warning, error and fatal.

Both the MessageFactory and Messages classes support four ways in which to create a Message:

- Directly adding the message
- Directly adding the message and replacing parameters
- Retrieving the message from a bundle
- Retrieving the message from a bundle and replacing parameters

Examples for each of these are:

```
messages.info("Simple Text");
messages.info("Simple Text with {0} parameter", 1);
```

```
messages.info(new BundleKey("org.jboss.international.seam.test.TestBundle", "key1"));
```

```
messages.info(new BundleKey("org.jboss.international.seam.test.TestBundle", "key2"), 1);
```

The above examples assume that there is a properties file existing at `org.jboss.international.seam.test.TestBundle.properties` with `key1` being a simple text string and `key2` including a single parameter.