The `address-book` Application

The `address-book` example application is a simple web application that stores contact data. It uses a single entity class, `Contact`, that uses the Java API for JavaBeans Validation (Bean Validation) to validate the data stored in the persistent attributes of the entity, as described in Validating Persistent Fields and Properties.

Bean Validation Constraints in `address-book`

The `Contact` entity uses the `@NotNull`, `@Pattern`, and `@Past` constraints on the persistent attributes.

The `@NotNull` constraint marks the attribute as a required field. The attribute must be set to a non-null value before the entity can be persisted or modified. Bean Validation will throw a validation error if the attribute is null when the entity is persisted or modified.

The `@Pattern` constraint defines a regular expression that the value of the attribute must match before the entity can be persisted or modified. This constraint has two different uses in `address-book`.

- The regular expression declared in the `@Pattern` annotation on the `email` field matches email addresses of the form *`name@domain name.top level domain`*, allowing only valid characters for email addresses. For example, `username@example.com` will pass validation, as will`firstname.lastname@mail.example.com`. However, `firstname,lastname@example.com`, which contains an illegal comma character in the local name, will fail validation.

- The `mobilePhone` and `homePhone` fields are annotated with a `@Pattern` constraint that defines a regular expression to match phone numbers of the form `(`*`xxx`*`)` *`xxx-xxxx`*.

The `@Past` constraint is applied to the birthday field, which must be a `java.util.Date` in the past.

Here are the relevant parts of the `Contact` entity class:

```
@Entity
public class Contact implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    @NotNull
    protected String firstName;
    @NotNull
    protected String lastName;
    @Pattern(regexp="[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\\."
        +"[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*"
        +"@(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?",
             message="{invalid.email}")
    protected String email;
    @Pattern(regexp="^\\(?(\\d{3})\\)?[- ]?(\\d{3})[- ]?(\\d{4})$",
             message="{invalid.phonenumber}")
    protected String mobilePhone;
    @Pattern(regexp="^\\(?(\\d{3})\\)?[- ]?(\\d{3})[- ]?(\\d{4})$",
             message="{invalid.phonenumber}")
    protected String homePhone;
    @Temporal(javax.persistence.TemporalType.DATE)
    @Past
    protected Date birthday;
    ...
}
```

Specifying Error Messages for Constraints in `address-book`

Some of the constraints in the `Contact` entity specify an optional message:

```
@Pattern(regexp="^\\(?(\\d{3})\\)?[- ]?(\\d{3})[- ]?(\\d{4})$",
             message="{invalid.phonenumber}")
    protected String homePhone;
```

The optional message element in the `@Pattern` constraint overrides the default validation message. The message can be specified directly:

```
@Pattern(regexp="^\\(?(\\d{3})\\)?[- ]?(\\d{3})[- ]?(\\d{4})$",
             message="Invalid phone number!")
    protected String homePhone;
```

The constraints in `Contact`, however, are strings in the resource bundle *tut-install*`/examples/persistence/address-book/src/java/ValidationMessages.properties`. This allows the validation messages to be located in one single properties file and the messages to be easily localized. Overridden Bean Validation messages must be placed in a resource bundle properties file named `ValidationMessages.properties` in the default package, with localized resource bundles taking the form `ValidationMessages_`*locale-prefix*`.properties`. For example,`ValidationMessages_es.properties` is the resource bundle used in Spanish speaking locales.

Validating `Contact` Input from a JavaServer Faces Application

The `address-book` application uses a JavaServer Faces web front end to allow users to enter contacts. While JavaServer Faces has a form input validation mechanism using tags in Facelets XHTML files,`address-book` doesn't use these validation tags. Bean Validation constraints in JavaServer Faces managed beans, in this case in the `Contact` entity, automatically trigger validation when the forms are submitted.

The following code snippet from the `Create.xhtml` Facelets file shows some of the input form for creating new `Contact` instances:

```
<h:form>
    <table columns="3" role="presentation">
        <tr>
            <td><h:outputLabel value="#{bundle.CreateContactLabel_firstName}"
                                 for="firstName" /></td>
            <td><h:inputText id="firstName"
                               value="#{contactController.selected.firstName}"
                               title="#{bundle.CreateContactTitle_firstName}" /></td>
            <td><h:message for="firstName" /></td>
        </tr>
        <tr>
            <td><h:outputLabel value="#{bundle.CreateContactLabel_lastName}"
                                 for="lastName" /></td>
            <td><h:inputText id="lastName"
                               value="#{contactController.selected.lastName}"
                               title="#{bundle.CreateContactTitle_lastName}" /></td>
            <td><h:message for="lastName" /></td>
        </tr>
        ...
    </table>
</h:form>
```

The `<h:inputText>` tags `firstName` and `lastName` are bound to the attributes in the `Contact` entity instance `selected` in the `ContactController` stateless session bean. Each `<h:inputText>` tag has an associated `<h:message>` tag that will display validation error messages. The form doesn't require any JavaServer Faces validation tags, however.

Running the `address-book` Example

You can use either NetBeans IDE or Ant to build, package, deploy, and run the `address-book` application.

To Run the `address-book` Example Using NetBeans IDE

1. **From the File menu, choose Open Project.**
2. **In the Open Project dialog, navigate to:**

   ```
   tut-install/examples/persistence/
   ```

3. **Select the `address-book` folder.**
4. **Select the Open as Main Project and Open Required Projects check boxes.**
5. **Click Open Project.**
6. **In the Projects tab, right-click the `address-book` project and select Run.**

After the application has been deployed, a web browser window appears at the following URL:

```
http://localhost:8080/address-book/
```

7. **Click Show All Contact Items, then Create New Contact. Type values in the form fields; then click Save.**

   If any of the values entered violate the constraints in `Contact`, an error message will appear in red beside the form field with the incorrect values.

To Run the `address-book` Example Using Ant

1. **In a terminal window, go to:**

   ```
   tut-install/examples/persistence/address-book/
   ```

2. **Type the following command:**

   ```
   ant
   ```

   This will compile and assemble the `address-book` application.

3. **Type the following command:**

   ```
   ant deploy
   ```

   This will deploy the application to GlassFish Server.

4. **Open a web browser window and type the following URL:**

   ```
   http://localhost:8080/address-book/
   ```

   Tip - As a convenience, the `all` task will build, package, deploy, and run the application. To do this, type the following command:

   ```
   ant all
   ```

5. **Click Show All Contact Items, then Create New Contact. Type values in the form fields; then click Save.**

   If any of the values entered violate the constraints in `Contact`, an error message will appear in red beside the form field with the incorrect values.